



**IMPROVEMENT OF DEEP NEURAL NETWORKS  
BY INTELLIGENT MODEL INITIALIZATION**

**Doktora Tezi**

**Burak BENLİGİRAY**

**Eskişehir, 2019**

**IMPROVEMENT OF DEEP NEURAL NETWORKS  
BY INTELLIGENT MODEL INITIALIZATION**

**Burak BENLİGİRAY**

**DOKTORA TEZİ**



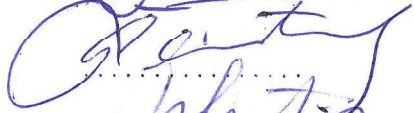


**Elektrik-Elektronik Mühendisliği Anabilim Dalı  
Danışman: Prof. Dr. Ömer Nezih GEREK**

**Eskişehir  
Eskişehir Teknik Üniversitesi  
Lisansüstü Eğitim Enstitüsü  
Ağustos 2019**

*Bu tez çalışması, BAP Komisyonu tarafından kabul edilen 1608F606 numaralı proje kapsamında desteklenmiştir.*

## FINAL APPROVAL FOR THESIS

This thesis titled “**Improvement of Deep Neural Networks by Intelligent Model Initialization**” has been prepared and submitted by **Burak BENLİGİRAY** in partial fulfillment of the requirements in “Eskişehir Technical University Directive on Graduate Education and Examination” for the Degree of Doctor of Philosophy (PhD) in Department of Electrical and Electronics Engineering has been examined and approved on 23/08/2019.

<u>Committee Members</u>	<u>Title, Name and Surname</u>	<u>Signature</u>
Member (Supervisor)	Prof. Dr. Ömer Nezih GEREK	
Member	Prof. Dr. Hakan ÇEVİKALP	
Member	Doç. Dr. Gürkan ÖZTÜRK	
Member	Doç. Dr. Tansu FİLİK	
Member	Dr. Öğr. Üyesi Hasan Serhan YAVUZ	

.....  
Prof. Dr Murat TANIŞLI

Director of Institute of Graduate Programs

## ÖZET

### AKILLI MODEL İLKLENDİRİLMESİ İLE DERİN SİNİR AĞLARININ İYİLEŞTİRİLMESİ

Burak BENLİGİRAY

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Elektronik Bilim Dalı

Eskişehir Teknik Üniversitesi, Lisansüstü Eğitim Enstitüsü, Ağustos 2019

Danışman: Prof. Dr. Ömer Nezh GEREK

Yakın zamanda ulaşılması kolaylaşan yüksek paralel işlem gücü ve etiketli görsel veriden yararlanılarak derin öğrenmede kaydedilen gelişmeler görsel tanıma başarımında kayda değer ilerlemelere neden olmuştur. Bu çalışmalardan ortaya çıkan yöntemler arasında büyük miktarda etiketli veri ile denetimli eğitilen derin evrimsel sinir ağları çeşitli görsel görevlerde tutarlı bir biçimde en iyi sonuçları vermektedir. Buna karşın bu yöntemlerin ihtiyaç duyduğu veri miktarı gerçek dünyadaki uygulamalardaki faydalarını sınırlamaktadır. Bu çalışmada derin evrimsel sinir ağlarını bu uç seviyedeki veri gereksinimden kurtarmanın yollarını araştırdık. İlk olarak, ImageNet öneğitiminden aktarılan temsillerin hedef veri dağılımı oldukça farklı olsa bile aşırı uyumu azalttığımızı gösterdik. Buna ek olarak çözümsel olarak tasarlanmış modellerle rassal olarak eğitim verisi üretmek için iki yaklaşım önerdik. Önerilen ilk yaklaşım, tanınacak hedef örüntülerin düşük seviyeli olduğu durumlarda kullanılabilir olan Gestalt ilkelerine dayalı olarak tamamen yapay eğitim verisi üretilmesidir. Buna karşılık eğer tanınacak hedef örüntü yüksek seviyeli ise eğitim verisi ImageNet benzeri mevcut bir veri kümesinden türetilir. Çözümsel olarak tasarlanmış unsurlardan faydalanan bu iki yöntem modele bilgi yerleştirerek veri bağımlılığını azaltmaktadır.

**Anahtar Sözcükler:** Derin Öğrenme, Evrimsel Sinir Ağları, Gestaltçı Eğitim, Özdenetimli Öneğitim.

## ABSTRACT

### IMPROVEMENT OF DEEP NEURAL NETWORKS BY INTELLIGENT MODEL INITIALIZATION

Burak BENLİGİRAY

Department of Electrical and Electronics Engineering  
Programme in Electronics

Eskişehir Technical University, Institute of Graduate Programs, August 2019

Supervisor: Prof. Dr. Ömer Nezir GEREK

Developments in deep learning leveraging the recent abundance of parallel computational power and visual data have resulted in significant advances in visual recognition performance. Among the methods that have emerged from this work, deep convolutional neural networks trained with a large amount of data in a supervised manner has been able to consistently deliver state of the art performance in various visual tasks. However, the amount of data this method requires limits its usefulness in real-world applications. In this study, we investigate methods to relieve deep convolutional neural networks from extreme data dependency. First, we show that transferring representations from ImageNet pretraining reduces overfitting even if the target data distribution is significantly different. In addition, we propose two approaches to stochastically generate training data using analytically designed models. The first approach is to generate entirely synthetic training data based on Gestalt principles, which is suitable when the target pattern to be recognized is low-level. Alternatively, if the target pattern to be recognized is high-level, training data is derived from an existing dataset such as ImageNet. By utilizing analytically designed elements, these two approaches inject knowledge to the model and reduce data-dependency.

**Keywords:** Deep Learning, Convolutional Neural Networks, Gestaltist Training, Self-supervised Pretraining.

23/08/2019

## STATEMENT OF COMPLIANCE WITH ETHICAL PRINCIPLES AND RULES

I hereby truthfully declare that this thesis is an original work prepared by me; that I have behaved in accordance with the scientific ethical principles and rules throughout the stages of preparation, data collection, analysis and presentation of my work; that I have cited the sources of all the data and information that could be obtained within the scope of this study, and included these sources in the references section; and that this study has been scanned for plagiarism with “scientific plagiarism detection program” used by Eskişehir Technical University, and that “it does not have any plagiarism” whatsoever. I also declare that, if a case contrary to my declaration is detected in my work at any time, I hereby express my consent to all the ethical and legal consequences that are involved.



Burak BENLİGIRAY

## TABLE OF CONTENTS

	<u>Page</u>
TITLE PAGE . . . . .	i
FINAL APPROVAL FOR THESIS . . . . .	ii
ÖZET . . . . .	iii
ABSTRACT . . . . .	iv
STATEMENT OF COMPLIANCE WITH ETHICAL PRINCIPLES AND RULES . . . . .	v
TABLE OF CONTENTS . . . . .	vi
LIST OF TABLES . . . . .	x
LIST OF FIGURES . . . . .	xi
ABBREVIATIONS . . . . .	xvii
1. INTRODUCTION . . . . .	1
2. NEURAL NETWORKS . . . . .	5
2.1 Neural Network Structure . . . . .	5
2.2 Deep Neural Networks . . . . .	11
2.3 Training Neural Networks . . . . .	12
2.3.1 Unsupervised training . . . . .	13
2.3.2 Supervised training . . . . .	14
2.4 Convolutional Neural Networks . . . . .	14

2.5	Model Initialization . . . . .	17
2.5.1	Random initialization . . . . .	18
2.5.2	Initialization for performance . . . . .	18
2.5.3	Unsupervised pretraining . . . . .	20
2.6	Transfer Learning . . . . .	21
2.6.1	Transfer deep learning . . . . .	22
2.6.2	Curriculum learning . . . . .	23
3.	IMAGENET PRETRAINING . . . . .	26
3.1	Case Study 1–Power Line Recognition from Aerial Images . . . . .	27
3.1.1	Aerial image dataset . . . . .	27
3.1.2	Proposed method . . . . .	28
3.1.3	Experimental results . . . . .	29
3.1.3.1	<i>Implementation details</i> . . . . .	31
3.1.3.2	<i>End-to-end classification</i> . . . . .	31
3.1.3.3	<i>Classifying CNN features</i> . . . . .	34
3.1.3.4	<i>Running time</i> . . . . .	36
3.1.3.5	<i>Comparison of methods</i> . . . . .	37
3.1.4	Visualization . . . . .	37
3.1.5	Conclusion . . . . .	40
3.2	Case Study 2–HEp-2 Cell Classification . . . . .	41
3.2.1	Introduction . . . . .	42
3.2.2	Related work . . . . .	43
3.2.3	Dataset and preprocessing . . . . .	44
3.2.4	Proposed method . . . . .	45
3.2.5	Experimental results . . . . .	46
3.2.6	Conclusion . . . . .	46
4.	GESTALT TRAINING . . . . .	48
4.1	Gestalt Theory of Perception . . . . .	49
4.1.1	Helmholtz principle . . . . .	51
4.2	Recognizing Color Constancies . . . . .	53
4.2.1	Modeling color constancy . . . . .	54



4.2.1.1	<i>Initial model</i>	54
4.2.1.2	<i>Difficulties with an exact model</i>	56
4.2.1.3	<i>Designing the task</i>	58
4.2.1.4	<i>Training with the generated task</i>	59
4.2.1.5	<i>Training ResNet-50 to recognize colored constancies</i>	59
4.3	<b>Deep Gradient Operator</b>	62
4.3.1	<b>Introduction</b>	62
4.3.2	<b>Generating edge examples</b>	66
4.3.3	<b>Deep gradient operator architecture</b>	67
4.3.4	<b>Experimental results</b>	68
4.3.5	<b>Conclusion</b>	72
5.	<b>SELF-SUPERVISED PRETRAINING</b>	74
5.1	<b>Self-supervised Pretraining by Ranking Spatial Coherency</b>	75
5.1.1	<b>Introduction</b>	75
5.1.2	<b>Related work</b>	77
5.1.2.1	<i>Self-supervision by spatial context</i>	78
5.1.2.2	<i>Self-supervision by cross-channel context</i>	79
5.1.2.3	<i>Self-supervision by temporal context</i>	79
5.1.2.4	<i>Other self-supervision types</i>	79
5.1.3	<b>Coherency ranking</b>	80
5.1.3.1	<i>Permutation granularity</i>	80
5.1.3.2	<i>Quantifying the coherency of permuted images</i>	81
5.1.3.3	<i>Implementation details</i>	83
5.1.4	<b>Experimental Results</b>	85
5.1.4.1	<i>Pretraining</i>	85
5.1.4.2	<i>Fine-tuning</i>	86
5.1.5	<b>Conclusion</b>	87
6.	<b>CONCLUSION</b>	90
	<b>REFERENCES</b>	95

**CV** ..... **112**



## LIST OF TABLES

	<u>Page</u>
<b>Table 3.1</b> Classification errors in percentages for the end-to-end classification method. . . . .	32
<b>Table 3.2</b> Confusion matrices for the end-to-end classification method (trained final layer and fine-tuned Stage 5, ImageNet pre-training, mean subtraction preprocessing). Rows are ground truths, and columns are predictions. . . . .	33
<b>Table 3.3</b> Classification errors in percentages for the CNN feature classification method with mean subtraction preprocessing. . . . .	35
<b>Table 3.4</b> Confusion matrices for the best CNN feature classification method (SVM classifier, ResNet-50 architecture, Stage 4 features for infrared and Stage 3 feature for visible light images, mean subtraction preprocessing). Rows are ground truths, and columns are predictions. . . . .	35
<b>Table 3.5</b> Cumulative running times of the CNN models for a single image in milliseconds. . . . .	36
<b>Table 3.6</b> Best performing configurations with each proposed method at infrared and visible light images. . . . .	37
<b>Table 3.7</b> Confusion matrix for HEp-2 cell classifications. . . . .	46

## LIST OF FIGURES

	<u>Page</u>
<b>Figure 2.1</b> A single neuron. Inputs are multiplied with weight parameters, and the results are added together with a bias term to apply a linear transformation. A nonlinear activation function is applied to this output, making the overall operation a nonlinear transformation. . . . .	6
<b>Figure 2.2</b> A layer composed of 4 parallel neurons. Each node denoted with $N$ is a neuron as illustrated in Figure 2.1. . . . .	6
<b>Figure 2.3</b> A multilayer perceptron (MLP). This MLP has two layers, each of which is composed of 4 neurons. The number of neurons in the last layer determines the output size. . . . .	7
<b>Figure 2.4</b> A locally-connected layer. Compare with Figure 2.2 and notice the smaller number of connections, which results in fewer parameters. . .	8
<b>Figure 2.5</b> A convolutional layer. The neurons with the same index (e.g., the 4 $N_0$ s) share weight parameters. . . . .	9
<b>Figure 2.6</b> A max pooling layer, which is used to downsample feature maps spatially and across channels. . . . .	10
<b>Figure 2.7</b> The AlexNet architecture. Conv denotes convolutional layers, Pool denotes max pooling layers, Norm denotes local response normalization layers (not commonly used in other architectures) and FullConn denotes fully-connected layers. . . . .	16
<b>Figure 2.8</b> The origin is a saddle point for $f(x, y) = x^2 - y^2$ . . . . .	17
<b>Figure 2.9</b> The objective function of a single parameter model. Training will converge to the global minimum only if the parameter is initialized at the part shown as a solid curve. . . . .	19
<b>Figure 2.10</b> The trajectory of training and test losses with and without pretraining.	20

<b>Figure 2.11</b>	A curriculum is composed of smoothed versions of the target task. Training at each task starts at the empty circle and ends at the arrow head. Note that unlike Figure 2.9, all initializations will converge to the global minimum when curriculum learning is used. . . . .	24
<b>Figure 3.1</b>	Examples from the aerial image dataset. The first two columns are infrared images with and without power lines, the following two columns are visible light images with and without power lines. The first two rows are easier examples, based on the metric proposed in [89]. . . . .	27
<b>Figure 3.2</b>	Two alternative methods for using CNNs for power line recognition: end-to-end classification and CNN feature classification. In end-to-end classification, the network is trained jointly. In CNN feature classification, the feature extractor and the classifier are trained sequentially. The disconnect is represented by dashed arrows. . . . .	28
<b>Figure 3.3</b>	CNN architectures used in the study, VGG-19 [19] and ResNet-50 [20]. The stages of the architectures are illustrated in different colors. Note that the convolutional and identity blocks of ResNet-50 are composed of multiple layers. . . . .	30
<b>Figure 3.4</b>	Receiver operating characteristic curves of the end-to-end classification method. The legend is in the order of decreasing area under the curve. Note that the curves overlap and occlude each other in some figures. . . . .	34
<b>Figure 3.5</b>	Receiver operating characteristic curves of the CNN feature classification method. ImageNet pre-trained ResNet-50 model is used with mean subtraction preprocessing. Stage 4 for IR and Stage 3 for VL are shown because they delivered the best results (see Table 3.3b). . . . .	35
<b>Figure 3.6</b>	The first row is images from the dataset [90], following rows are saliency maps generated using regular gradients [100], guided back-propagation [101] and integrated gradients [102], respectively. Darker colors indicate higher saliency. . . . .	38

<b>Figure 3.7</b>	The first row shows input images, second row shows saliency maps obtained by guided backpropagation [101] (darker color indicates higher saliency), and third row shows a visualization technique that can provide visual feedback for the pilot. . . . .	39
<b>Figure 3.8</b>	Steps to present the original image and the saliency map simultaneously. The saliency map is thresholded, the thresholded image is smoothed, and a heat map is produced from the resulting image. The heat map is superimposed on the original image transparently. . . . .	39
<b>Figure 3.9</b>	Activation visualizations for the model in Table 3.1a. The images are zoomed $\times 2$ for better viewing. The model was initialized by ImageNet pre-training, then Stage 5 and Final Stage were fine-tuned for power line recognition. . . . .	40
<b>Figure 3.10</b>	Examples from different classes of the ICPR 2014 dataset. The first row are “intermediate” and the second row are “positive” examples. . . . .	44
<b>Figure 3.11</b>	An original example of an intermediate intensity Golgi image and its rescaled version for better viewing. . . . .	45
<b>Figure 4.1</b>	A herring gull chick pecking the red spot on its mother’s beak. The chicks are predisposed to this visual cue innately. . . . .	48
<b>Figure 4.2</b>	Different kind of grouping principles are perceived with different $\epsilon$ -significance, which implies that a universal threshold of 1 is not optimal. For example, the threshold for perceiving faces is very low, causing frequent false alarms. . . . .	52
<b>Figure 4.3</b>	An illustration of the effect of color constancy on perception. We perceive the group of black pixels to be the parts of a single object. This applies even though the object has an arbitrary form. . . . .	53
<b>Figure 4.4</b>	The model of a darker square over a lighter background. In the case that each pixel is either dark or light with equal probability, the individual probabilities of each pixel to conform to this model is $1/2$ . . . . .	54
<b>Figure 4.5</b>	$2 \times 2$ , $3 \times 3$ and $4 \times 4$ black squares in uniform white noise images of size $256 \times 256$ . The Gestalts with smaller $NFA$ are more perceivable as expected. . . . .	55

<b>Figure 4.6</b>	Dark squares with different sizes in $28 \times 28$ uniform white noise images. It is not possible to distinguish squares of size $2 \times 2$ . . . . .	57
<b>Figure 4.7</b>	Number of false alarms ( $NFA$ ) for the square widths in Figure 4.6. Note that $NFA$ is nearly 1 for squares with size $2 \times 2$ , which indicates that they would be borderline impossible to perceive. . . . .	57
<b>Figure 4.8</b>	Instead of using the absolute $NFA$ values as in Figure 4.7, we can use them to create a difficulty index. . . . .	58
<b>Figure 4.9</b>	$8 \times 8$ squares of different shades of gray, and a negative example. . .	58
<b>Figure 4.10</b>	The CNN architecture used in the experiments. The subscripts indicate the number of neurons at each layer, and the following numbers indicate kernel sizes. . . . .	59
<b>Figure 4.11</b>	Test errors with varying square sizes. The experiments are repeated 5 times. Means are shown with circular markers and $\pm 1$ standard deviations are shown with error bars. . . . .	60
<b>Figure 4.12</b>	$224 \times 224$ colored color constancy examples. . . . .	60
<b>Figure 4.13</b>	Color constancy recognition error rates with ResNet-50 for the examples similar to the ones in Figure 4.12. . . . .	61
<b>Figure 4.14</b>	Gradient detection with the Sobel and deep gradient operators. (a) The image is convolved with two Sobel operators, which are analytically designed to detect horizontal and vertical gradients. The Euclidian norm of these two response maps gives the gradient map. (b) The gradient map is inferred with a convolutional neural network in an end-to-end manner. Darker colors in the image indicate higher gradient responses. . . . .	64
<b>Figure 4.15</b>	The first row are stochastically generated shapes. The second row are the contour images of the shapes, which are used as pixel labels. The third row are the training data obtained by adding Gaussian noise to the stochastically generated image. . . . .	67
<b>Figure 4.16</b>	A convolutional neural network with three layers, and equal input and output sizes has been trained to recognize pixels with high gradients. Darker colors in the image indicate higher gradient responses. . . . .	68

<b>Figure 4.17</b>	The first row are the training data obtained by adding Gaussian noise to the stochastically generated image. The second row are the results of the Sobel operator. The third row are the results of the Sobel operator after Gaussian blur is applied. The fourth row are the results of the proposed method. Darker colors in the image indicate higher gradient responses. . . . .	69
<b>Figure 4.18</b>	Mean binary cross-entropy losses calculated with 1000 synthetic images. Each epoch corresponds to training with 6400 unique examples.	70
<b>Figure 4.19</b>	The first row is the original image. The second row is the output of the Sobel operator. The third row are the results of the Sobel operator after Gaussian blur is applied. The fourth row is the output of the proposed method. The fifth row is the output of the proposed method when the training task is not made more difficult through additive noise. Darker colors in the image indicate higher gradient responses. .	71
<b>Figure 4.20</b>	The 16 different $3 \times 3$ kernels learned by the first layer of the deep gradient operator. The weights are linearly scaled so that the lowest value is shown in black and highest value is shown in white. . . . .	72
<b>Figure 4.21</b>	Activation maximizations of the network. A noise image is optimized for maximum activation at the output, and the gradients for this operation are shown. The gradients are higher on the contours (shown in white). . . . .	72
<b>Figure 5.1</b>	Images from the CIFAR-10 dataset, divided into $4 \times 4$ tiles, and permuted to generate degrees of coherency. (a) is the original image, (b) is a mildly incoherent example, and (c) is an intensely incoherent example. The proposed pretraining task is to rank (b) and (c) with respect to coherency. . . . .	76
<b>Figure 5.2</b>	An image from the ImageNet dataset permuted with differing granularity. Permuting coarsely-tiled images only disturbs high level patterns, while permuting finely-tiled images disturbs all levels of patterns.	81



<b>Figure 5.3</b>	A 2D color map [169] is permuted randomly, and the respective coherency values are given below. Disturbances in color gradients indicate incoherency. . . . .	82
<b>Figure 5.4</b>	Probability density function of coherency values when all patches of a $4 \times 4$ -tiled image are permuted randomly, measured with 1,000,000 samples. . . . .	83
<b>Figure 5.5</b>	Training for ranking image pairs in a batch. Batch scores are matched pairwise and ranked. . . . .	84
<b>Figure 5.6</b>	Training and validation losses across the epochs for the proposed self-supervised pretraining method. . . . .	85
<b>Figure 5.7</b>	Maximum classification accuracies on the CIFAR-10 test set using different initialization methods with different training set sizes. . . . .	87
<b>Figure 5.8</b>	Maximum classification accuracies on the STL-10 test set using different initialization methods and training batches. The dashed lines indicate the mean classification accuracy across the batches. . . . .	88

## ABBREVIATIONS

<b>AI</b>	Artificial Intelligence
<b>CNN</b>	Convolutional Neural Network
<b>DBN</b>	Deep Belief Net
<b>DCT</b>	Discrete Cosine Transform
<b>ICIP</b>	International Conference on Image Processing
<b>ICPR</b>	International Conference on Pattern Recognition
<b>ILSVRC</b>	The ImageNet Large Scale Visual Recognition Challenge
<b>IR</b>	Infrared
<b>MLP</b>	Multilayer Perceptron
<b>NB</b>	Naive Bayes
<b>RBM</b>	Restricted Boltzmann Machines
<b>ReLU</b>	Rectified Linear Unit
<b>RF</b>	Random Forests
<b>SVM</b>	Support Vector Machine
<b>VL</b>	Visible Light

## 1. INTRODUCTION

A *convolutional neural network* (CNN) is a specific type of architecture that is predominantly composed of convolutional layers, which are different from fully-connected layers in that their outputs are computed using only the inputs that are in close proximity to each other [1]. In other words, convolutional layers omit to model the correlation between spatially distant parts of the input in order to sparsify the redundancy [2]. This architecture is inherently useful for input data captured from a sensor array, where the correlation between the values read by more distant sensors are indeed weaker due to the laws of physics. An image captured by a 2D camera sensor is a good example of this kind of data, which predicts CNNs to be especially suitable for vision applications.

The CNN architecture was widely known for a long time and work on *deep learning* (learning stacked nonlinear models) has been going on for decades with relative success [3, 4]. However, it took the deep CNN model, AlexNet [5], winning The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 [6] in a decisive manner to attract the mass attention of computer vision researchers, and they have been fascinated with CNNs ever since.

CNNs allow visual recognition tasks that once required specifically designed methods [7, 8] to be conquered with a general-purpose, end-to-end approach [5, 9, 10]. Even the ultimate benchmark of visual recognition, human performance, was surpassed by CNNs in complex vision tasks [11, 12]. However, models with a large number of parameters such as deep neural networks have a notorious weakness: They tend to overfit with limited training data, which causes them to fail at generalizing their knowledge to all possible cases [13].

The leap in performance achieved by machine learning-based methods in computer vision have resulted a heavy tendency towards more data-dependent methods. This even resulted in a new field to emerge, *AutoML*, which applies machine learning to determine the hyperparameters of machine learning methods for ultimate data dependency [14, 15]. It must be noted that even Bayesian hyperparameter optimization [16] is not standard practice in deep learning applications, and hyperparameters tend to be selected by a hybrid of grid and manual search. Similarly, there have been attempts to automate neural network

architecture design [17, 18], but in almost all applications, analytically designed general-purpose architectures [5, 19, 20] are preferred. The layers themselves are all manually designed as well. For example, convolutional layers are designed to provide translation equivariance, yet not rotation equivariance. The main reason behind this is simply that the ImageNet dataset does not include upside down images of dogs. Furthermore, data augmentation is done only with crops and horizontal mirrors of the image, because the designer knows that these are the specific invariances required for most cases. Countless additional examples can be given to show that relying on domain knowledge is the norm in designing deep learning methods.

One should wonder why data dependency is presented as a magic bullet and analytical design is ostracized, considering that all significant developments in deep learning-based computer vision have resulted from analytical design (convolutional layers [1], data augmentation [21], residual connections [22], etc.). In general, analytical approaches restrict a model’s degrees of freedom, while data-dependent approaches relax them. In other words, data dependency breeds data hunger, which causes overfitting unless it is satisfied. Nevertheless, this approach is forced by the “tech giants” using grants, donations and influential researchers in their payrolls to leverage their abundant computation power and proprietary datasets. The palpable stagnation in deep learning-based computer learning research in the last few years (already referred to as the beginning of the third *AI winter* by some) can be attributed to this situation.

In this thesis, we focus on methods that are aimed to relieve deep learning-based vision applications from data dependency. In particular, we investigate three main methods:

- **ImageNet pretraining:** Pretraining with the ILSVRC object classification task is a well-established method of transfer learning that reduces overfitting for target tasks where the data is composed of natural images similar to ImageNet [23]. We present case studies where the target task data is from a significantly different domain, yet ImageNet pretraining is still very beneficial.
- **Gestalt learning:** In this approach, a generative model is analytically designed to stochastically produce a pattern that is recognized as a whole, called a *Gestalt*. To be able to design a generative model that is adequately general, the Gestalt to be generated has to be low-level (e.g., edges and blobs, rather than cats and dogs).

The generated examples are then used to train CNNs, which is a completely data-independent process.

- **Self-supervised pretraining:** This method also involves analytically designing a model that can stochastically generate data. However, unlike Gestalt learning, the generated data in this method is derived from existing data. This method can be seen as a middle ground between ImageNet pretraining and Gestalt learning.

As a result of our research, we have come to the general conclusion that data dependency is indeed a great problem for deep neural networks, which can be alleviated by regularization. Injecting knowledge to the model by pretraining can be seen as a type of regularization, as the benefits are identical. While pretraining can be done in a heavily data-dependent way (i.e., through supervised methods), there are also manual design-heavy methods (self-supervised pretraining and Gestalt learning). The reintroduction of design to computer vision allows a synthesis of the traditional theory-based methods and the more recent data-dependent methods.

If we delve deeper into our findings, we can say that supervised pretraining using the ImageNet dataset has a much more universal benefit than it is generally assumed. Our results indicate that researchers should be more liberal in employing it in unusual domains, and even in non-visual tasks.

Possibly the most unique contribution of this thesis is the introduction of a completely data-independent method of employing CNNs, which we have named Gestalt learning. This includes the modeling of a visual skill based on the Gestalt theory of perception and designing a training task to teach this skill to the model. Then, the resulting model can be used to solve the related vision problem even without fine-tuning using real data. Although cases have been presented where CNNs are trained with generated examples to solve toy problems [24], there are not any examples in the literature where this approach is used to solve a real vision problem. We apply our proposed method to an established vision problem, specifically edge detection, and demonstrate success. From a different viewpoint, we propose a method to represent theory-based vision methods in the form of a CNN, which enables the integration of these methods with the current end-to-end deep learning paradigm.

Finally, we aimed to find a middle ground between the extremely data-dependent

supervised pretraining and the data-independent Gestalt learning in the form of self-supervised pretraining. This method both familiarizes the model with a specific data distribution and also injects visual knowledge in the form of a surrogate pretraining task. Moreover, unlike supervised training which transfers all levels of representations, the proposed surrogate task can be modified to transfer only the representations below an arbitrary level.

The structure of this dissertation is as follows: We start with an introduction to deep neural networks for visual recognition in Chapter 2. Then, we introduce ImageNet pretraining and present case studies where it is shown to be beneficial even when the target data is from a vastly different domain in Chapter 3. This is followed by Chapter 4 where Gestalt learning is introduced. In Chapter 5, a self-supervised pretraining method based on spatial context is proposed that derives training data stochastically from a source dataset. We end with concluding remarks in Chapter 6.

## 2. NEURAL NETWORKS

In this chapter, we are going to introduce neural networks (historically referred to as artificial neural networks), which are hierarchical, nonlinear function approximators inspired by organic neural systems [25]. While the biomimetic narrative incited interest, neural networks have been criticized from theoretical and practical standpoints. They are commonly referred to as black boxes, as it is not clear how the approximation they perform can be analyzed [26]. Consequently, the field experienced some of its breakthroughs based on intuition and experimentation, rather than a strong theoretical background. On the practical side, neural network performance is highly dependent on the choice of hyperparameters (number of layers, learning rate, etc.), which can only be optimized empirically [27]. On top of these issues, neural networks were regularly outperformed by support vector machines [28] in various applications [29, 30, 31], especially when well-crafted representations are used as inputs. This resulted in a relative fade into obscurity, until a recent comeback in performance with deeper architectures.

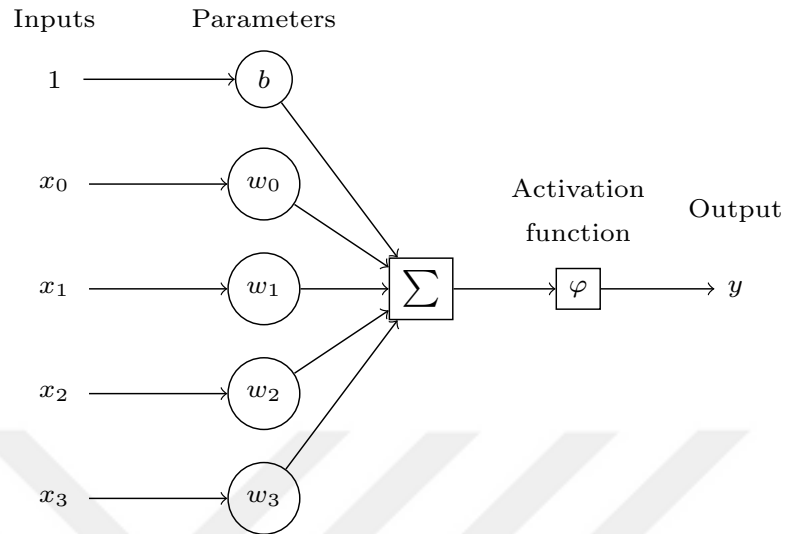
### 2.1. Neural Network Structure

The fundamental computing unit of a neural network is the neuron, which applies a non-linearity  $\varphi(\cdot)$  to the product of its inputs  $\mathbf{x}$  and weight parameters  $\mathbf{w}$  to generate an output  $y$ . An additional parameter  $b$  is commonly added to the product to induce a bias that is independent from the input [32].

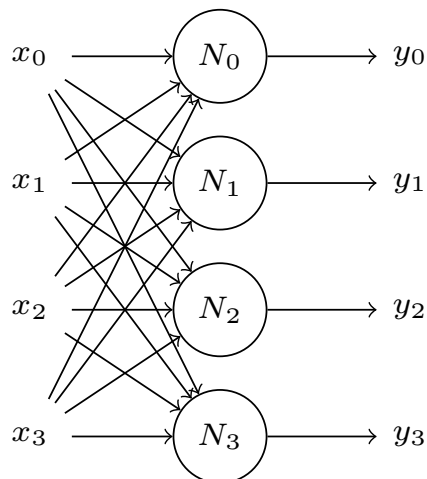
$$y = \varphi(\mathbf{w}^T \mathbf{x} + b) \tag{2.1}$$

See Figure 2.1 for an illustration of a neuron. Note that although the transformation applied by the weight and bias parameters are linear, the additional nonlinear activation function allows the neuron to approximate nonlinear functions.

Various network architectures can be obtained by interconnecting neurons in specific patterns. Most commonly, neurons are grouped in parallel as layers (see Figure 2.2 for a fully-connected layer), where each neuron in the layer receives the entire input. Then, the computation on a layer can be represented as a matrix-vector multiplication,

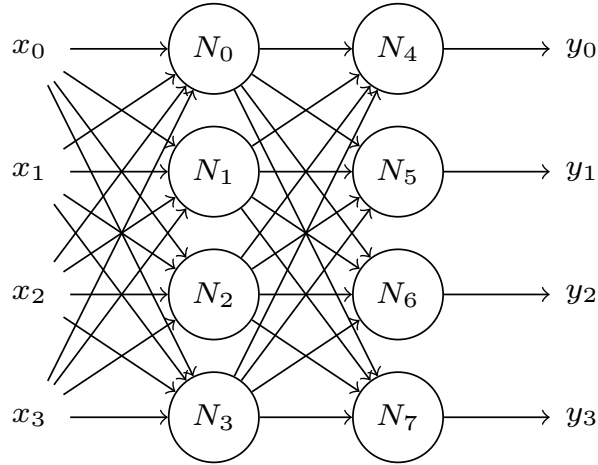


**Figure 2.1:** A single neuron. Inputs are multiplied with weight parameters, and the results are added together with a bias term to apply a linear transformation. A nonlinear activation function is applied to this output, making the overall operation a nonlinear transformation.



**Figure 2.2:** A layer composed of 4 parallel neurons. Each node denoted with  $N$  is a neuron as illustrated in Figure 2.1.





**Figure 2.3:** A multilayer perceptron (MLP). This MLP has two layers, each of which is composed of 4 neurons. The number of neurons in the last layer determines the output size.

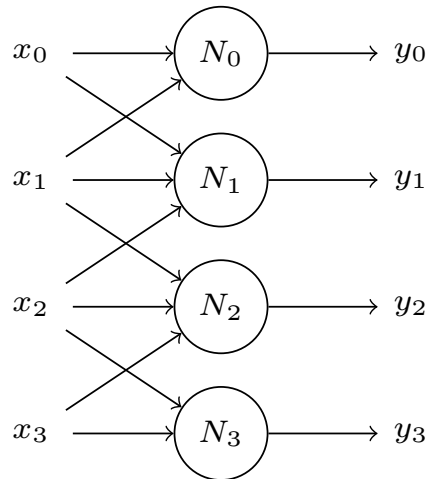
which is embarrassingly parallelizable. This can be represented as below, where  $\varphi \cdot ()$  operates element-wise on the product:

$$\mathbf{y} = \varphi \cdot (\mathbf{W}\mathbf{x} + \mathbf{b}) \quad (2.2)$$

The multilayer perceptron (MLP) is the most straightforward neural network architecture, which is essentially a stack of fully-connected layers. Since the layers are fully-connected, each neuron is connected to all neurons in the previous and following layers (see Figure 2.3). Each layer processes the data it received from the previous layer, and passes its output onto the next layer [33]. This succession generates hierarchically higher levels of data representations along the network [2].

As fully-connected layers include all possible connections, variations can be obtained by eliminating some of these. As seen in Figure 2.1, each input connection to a neuron requires an additional parameter. Removing obsolete connections reduces the number of parameters of the model, which decreases complexity without harming the model’s representation power. Therefore, it is in our greatest interest to remove as many obsolete connections as possible. Knowledge of the data distribution is very useful for this, as it allows us to analytically determine which connections to remove.

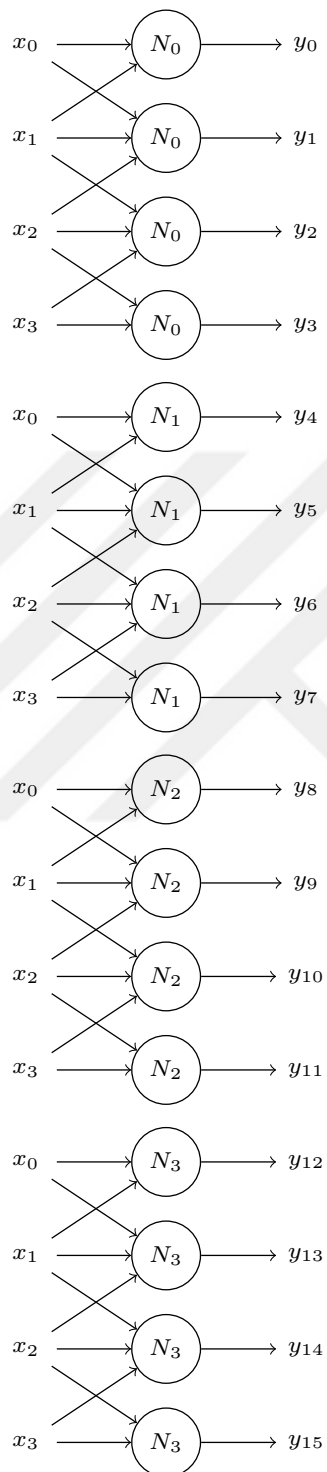
In some applications, all input examples have the same spatial alignment. For example, while doing identity verification on biometric passport photos, we can assume that all input examples are going to be aligned. Then, we can design a layer composed of neurons that are specialized on different regions. For example, a neuron only sees input data from



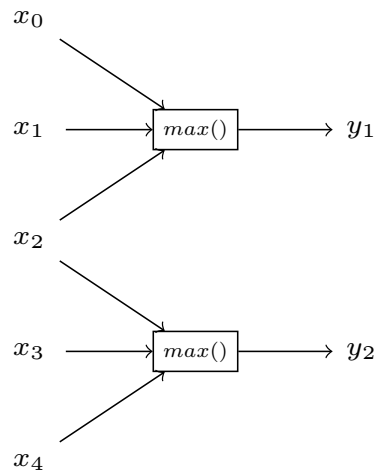
**Figure 2.4:** A locally-connected layer. Compare with Figure 2.2 and notice the smaller number of connections, which results in fewer parameters.

the eye region, while another neuron only sees input data from the mouth region. This can be achieved by a locally-connected layer (see Figure 2.4). From a machine learning-perspective, the advantage of using a locally-connected layer over a fully-connected layer is to reduce the number of model parameters, which reduces the amount of training data needed to avoid overfitting.

It is rare to encounter problems where examples are spatially aligned, so locally-connected layers are not used often. However, for spatiotemporal data, locality is always an important clue that fully-connected layers do not utilize. In these cases, convolutional layers become useful. A convolutional layer is a variation of a locally-connected layer, where each neuron is applied not to a single region, but rather to all regions in parallel. See Figure 2.5 for an illustration. Note that although there are 4 inputs and 4 neurons, since each neuron is applied at each possible position, the layer has 16 outputs. As the number of neurons did not change, this layer has the same number of parameters as the locally-connected layer in Figure 2.4. Although the number of parameters did not increase in this layer, more input connections are going to be needed in the next layer to accommodate for the larger number of outputs, which is going to increase the overall number of parameters in the model. For most applications, this is acceptable in return of achieving *translation equivariance*, which means having an equivalently translated output when the input is translated. Another advantage of convolutional layers over locally-connected layers is that since each neuron is applied to all regions, they “see” more training data. Following from the biometric photo example, a neuron may learn representations from eye regions that



**Figure 2.5:** A convolutional layer. The neurons with the same index (e.g., the 4  $N_0$ s) share weight parameters.



**Figure 2.6:** A max pooling layer, which is used to downsample feature maps spatially and across channels.

can help it recognize mouth regions better. In that regard, forcing neurons to specialize is generally not a good idea.

While convolutional layers are far superior to fully-connected layers when data locality is strong, they still cause the number of parameters to explode quickly when stacked. For this reason, they are commonly used with max pooling layers (see Figure 2.6). This layer does not have any learnable parameters, it simply passes the local maxima to the next layer, effectively downsampling the feature map spatially and across channels. In [34], max pooling layers are replaced with  $1 \times 1$  convolutional layers, which downsample the feature map only across channels. Even if max pooling is not used, stacked convolutional layers downsample the representations spatially, causing CNNs to operate similar to a bag-of-visual-words approach [7, 35]. There have been work on capsule networks as an alternative to this rather aggressive type of abstraction [36], but these architectures have yet to be proven viable for high-level tasks such as natural object recognition.

We have covered the architectural choices based on analyzing the spatial properties of the input, which are relevant to the following chapters. There are many other important elements of CNN architecture such as layer blocks [37], residual connections [22] and batch normalization layers [38]. However, we are not going to attempt to cover these subjects fully in this thesis.

## 2.2. Deep Neural Networks

Deep learning is the practice of learning stacked nonlinear models. Therefore, any network architecture that is composed of at least two layers is considered to be deep. However, an implied property of deep neural networks is that rather than relying on manually crafted descriptors, they tend to operate on the input data directly. In other words, earlier stages of a deep neural network learn representations, while later stages learn the target task, and the transition is obscure [2].

Theoretically, any function can be approximated to a given accuracy using an MLP with two layers [39]. However, this is not feasible for relatively complex problems, as the required model size (i.e., the number of neurons) will be very large. The amount of data required to train a well-generalizing neural network increases exponentially with the number of model parameters, which is a manifestation of *the curse of dimensionality* [40]. Two-layered, wide architectures are not practical for complex problems due to their computation power and training data requirements. Regarding these aspects, deeper architectures are exponentially more efficient [41, 42].

Deep learning aims to capture representations that are built on top of each other. Preferably, these representations should grow more abstract along the hierarchy to be able to disentangle the underlying causes of variation in the data [2]. A useful definition of abstraction is “being invariant against unimportant variations”, and in deep learners, this is provided by downsampling (e.g., pooling) and sparsifying (e.g., convolutional layers, regularization) elements. A good example of a deep architecture learning abstract representations without an explicit incentive is reported by Liu et al. [43]. After training a CNN to recognize identities from face images, it was observed that individual neurons from later layers selectively responded to semantic high-level attributes, such as gender, age, and race. Abstract representations are what humans depend on for high-level reasoning, and their invariance against irrelevant changes enables them to be applicable for a wide range of inputs. Since deep learning can generate abstract representations, it is a suitable tool for developing machine intelligence.

Although deep learners are efficient and have strong representation capabilities, there was a lack of success in using deep models in the earlier literature. The apparent reason is the difficulty of training [44], with two prominent milestones set along the

way. The first one was Hinton et al. successfully training a deep generative model with a greedy strategy [3]. This was followed by Krizhevsky et al. using a more traditional training strategy to outperform the state of the art methods in a complex object classification task by a wide margin [5]. Let us discuss how neural networks are trained in the next section.

### 2.3. Training Neural Networks

Neural networks approximate functions by learning suitable weight parameters. Training a network consists of defining a cost function and optimizing it for a given dataset. The most common training strategy for neural networks is iterating over the following two steps:

- Gradients of the cost function with respect to a mini-batch of examples are calculated for all layers using backpropagation [45];
- Weights are updated simultaneously by gradient descent to lower the cost function with respect to the mini-batch.

Alternatively, there have been attempts to train layers in sequence [3, 46] or adding intermediate outputs to the architecture [37] to be able to use deeper architectures, but “tricks” (various regularization methods [2], residual connections [22], batch normalization [38], etc.) were discovered later on that allowed the regular backpropagation algorithm to be used successfully with very deep architectures.

Models used for machine learning are either generative or discriminative (this is somewhat of an oversimplification, e.g., [47]). The problems we face in the real-world—such as recognition and control—can often be modeled as tasks to be solved discriminatively (e.g., “Is this product faulty?”). While training a model, if ground truth is used in the calculation of the cost function, the process is referred to as being supervised. Since discriminative models output labels or regressed values that can be compared to ground truth, they are inherently suitable to be trained in a supervised manner. Due to the fact that we want to solve problems that can be solved with discriminative models more, supervised training is used frequently both in academia and industry.

In unsupervised training, only the input examples are used to calculate the cost function. Since most of the data available on the Internet is not manually labeled, unsupervised

methods have a distinct advantage in that they can utilize more data. The problem with unsupervised training is that the related tasks tend to be generative. For example, denoising autoencoders are trained to filter out artificial noise from input examples [48]. This task essentially requires the model to generate a denoised example, which means that the model has to be generative. Although generative models can be modified to produce representations to be used in discriminative tasks, they underperform compared to authentically discriminative models [2]. Since it is difficult to model real-world problems as generative tasks, this limits the usefulness of unsupervised training.

An interesting hybrid of supervised and unsupervised training is self-supervised training [49]. Here, a task is designed analytically to generate example–label pairs without using any ground truth. Then, using the generated example–label pairs, the model is trained in a supervised manner. Nevertheless, the approach is not exactly supervised, as it does not require ground truth. The advantage of this approach is that it allows discriminative models to be trained with unlabeled data. This approach cannot outperform supervised training with a very large amount of labeled data, but it is a very good alternative if the amount of labeled data is limited.

### **2.3.1. Unsupervised training**

A generative method learns  $P(X)$  without supervision, where  $X$  is the input. In other words, it learns to generate likely inputs, depending on its hidden variables. This is not sufficient for tasks like classification or regression, which require the estimation of  $P(Y|X)$ , where  $Y$  is the desired output. Representations that are useful to estimate  $P(X)$  tend to be effective for estimating  $P(Y|X)$ ; therefore, representations learned for generation can be used as inputs of a supervised learner with strong smoothness assumptions [2]. The fact that deep generative models are not limited to generative tasks makes them widely relevant.

Hinton et al. proposed greedy, layer-wise unsupervised training of stacked restricted Boltzmann machines (RBM) [50], which is called a deep belief net (DBN) [3]. It was shown that RBMs could be replaced with alternative generative elements, such as autoencoders [51, 4]. Denoising autoencoders are a better performing variant of these networks, which are trained to remove the stochastic noise added to the input [52]. Unlike DBNs, deep Boltzmann machines consist of undirected connections of RBMs [53]. The added

top-down connections result in better performance, at the cost of estimating the intractable model expectations using Markov chain Monte Carlo [54]. There is also work on designing generative models that can be trained by backpropagation, in the hope of being able to use the backpropagation-related tricks that have helped supervised learners substantially [55].

### 2.3.2. Supervised training

Backpropagation jointly trains all layers of a neural network to approximate  $P(Y|X)$ . This appears more appealing than layer-wise training to approximate  $P(X)$ , as it is more specific to the target task. However, earlier experiments showed that training MLPs with more than 3 layers by backpropagation rarely yielded good results [33]. Greedy, layer-wise supervised training (similar to what was done with DBNs [3]) also did not produce favorable results for MLPs [4]. In fact, it was debated not long ago that it is not feasible to train a deep learner solely by supervised backpropagation [41, 56, 44].

For non-trivial tasks, the objective function of a supervised deep learner is highly non-convex [13]. Using a first-order method (e.g., gradient descent) to optimize this non-convex function of model parameters results in getting stuck at local minima. This problem was commonly solved by initializing the training at “a good basin of attraction” by unsupervised pretraining [13].

Jarrett et al. reported that with architectures optimized for the specific application, unsupervised deep learners do not have a significant advantage over supervised deep learners [35]. Supporting this claim, Krizhevsky et al. achieved a spectacular breakthrough at a complex vision task, mostly using already available methods [5]. The resulting architecture is known as the deep convolutional neural network, which is the go-to method for achieving state of the art results in vision applications today. We are going to discuss CNNs in the next section.

## 2.4. Convolutional Neural Networks

In the classic MLP, all neuron outputs of the previous layer are connected to all neuron inputs of the next layer. The resulting fully connected layers have great representation power. However, since each connection has a weight parameter, stacking many fully



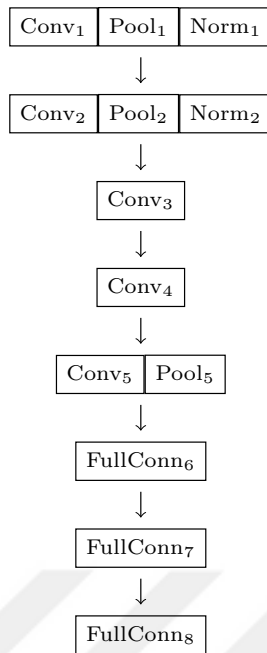
connected layers quickly increases the number of parameters in the model. Sparsifying the connections by removing unnecessary ones will result in a smaller model and reduce overfitting.

When designing a neural network architecture for a specific application, a good rule of thumb is that inputs with immediate correlations should be connected together to the neurons. In the case of simultaneous raw readings from a sensor array (e.g., an image), readings from sensors that are closer to each other will have stronger correlations. It is safe to assume that in a natural image, two neighboring pixels read similar values, while the same cannot be said for two pixels at the opposite borders of the image. Hence, if a sparser network is desired, it is best to sever connections that model the relations between distant pixel values.

Relevant features—edges, blobs and higher level features—may appear at any part of the image. The model should be able to respond to all features, regardless of their position. Then, a neuron that has learned to respond to a certain feature should be applied periodically throughout the image. The combination of only using the local pixels and repeating the product operation by translation is equivalent to convolution with a kernel. The layers where the connections are arranged to apply this operation are called convolutional layers. A neural network with convolutional layers is called a CNN, and is used as early as 1989 [1].

According to Jarrett et al., majority of object recognition systems are composed of a filter bank, a nonlinear operation and a pooling operation [35]. For example, in SIFT [57], a set of edge filters with different scales and orientations are applied—or at least, the end result is equivalent. Determining the scale and orientation is the nonlinear operation, and computing the gradient orientation histogram is the pooling operation. CNNs fit the same description, where convolutional layers are filter banks and the activation functions act as the nonlinear operations. CNNs commonly end with fully connected layers, which act as pooling operations. Additionally, some architectures use explicit pooling layers between convolutional layers [5].

CNNs are a fairly old concept, which were not considered to be an alternative to more recent object recognition methods [7, 8]. However, the success with unsupervised deep learners implied the possibility of a well performing deep CNN. Krizhevsky et al.'s implementation of an 8 layer CNN (commonly referred to as *AlexNet*, see Figure 2.7) [5]

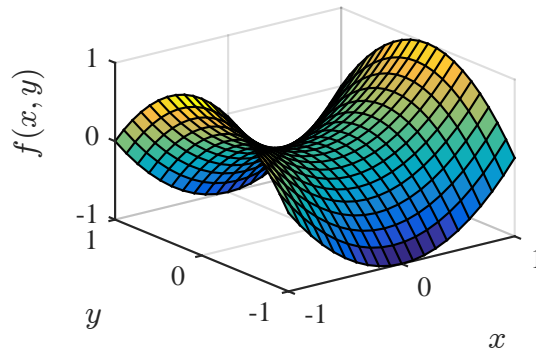


**Figure 2.7:** The AlexNet architecture. Conv denotes convolutional layers, Pool denotes max pooling layers, Norm denotes local response normalization layers (not commonly used in other architectures) and FullConn denotes fully-connected layers.

was the first one to deliver state of the art results in ILSVRC [6]. Following 2012, all tasks in ILSVRCs were won by CNNs (e.g., [58, 37, 34]).

Bengio et al. attribute the recent success of CNNs to [2]:

- GPU computation that allows for longer training [59, 10],
- Data augmentation [10],
- Large number of tasks (e.g., 1000 classes for ImageNet classification [60]),
- Convolutional architecture [1] and max-pooling layers [61],
- Rectifying nonlinearities as activation functions [62],
- Careful parameter initialization [44],
- Careful parameter update and adaptive learning rate heuristics [63],
- Layer-wise feature normalization [5],
- Dropout [64].



**Figure 2.8:** The origin is a saddle point for  $f(x, y) = x^2 - y^2$ .

Since the list above was compiled in 2013, more progress was made. Google’s team managed to secure ILSVRC 2014 with a 22-layer network [37], while Microsoft’s team won the first place in ILSVRC 2015 with a 152-layer network [34, 65]. A significant barrier to increasing depth is the accumulated shifts in the calculated gradients along the layers (the exploding/vanishing gradient problem [66]). GoogLeNet combats this by utilizing auxiliary output layers along the network during training [37]. Since these outputs are not as distant to the earlier layers as the outputs at the end of the network, the gradients they provide are less degraded. Later on, batch normalization becomes the standard solution of this problem [38], which is also used in the 152-layer ResNet [20].

Another important addition to this list are residual connections, proposed by He et al. [22]. These are identity connections that bypass groups of layers. Since the values from the earlier layer are always propagated forwards, layers only have to learn the residual differences. He et al. report that using this architecture, classification performance improves continuously with depth [20].

## 2.5. Model Initialization

Backpropagation is an iterative training algorithm; therefore, it requires the neuron weights to be initialized. Initializing all weights at zero is not preferable, because it produces a perfectly symmetrical model. Backpropagation updates weights according to their individual gradients. In the case where all weights and their gradients are the same, they will not be able to diverge and specialize [27]. Moreover, the origin of the objective function is a saddle point [33], which can trap optimization algorithms depending on first order

gradients [67]. Even though they are not local minima, gradients are zero at saddle points (see Figure 2.8).

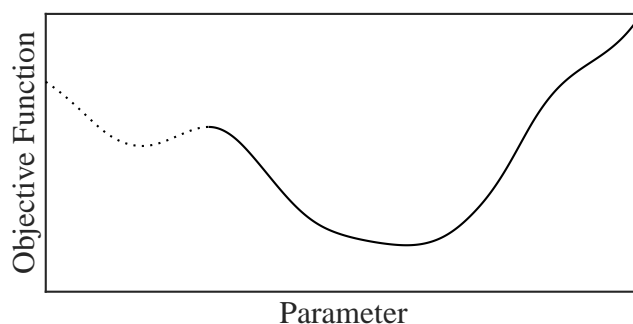
### 2.5.1. Random initialization

Weights can be sampled from a random distribution to introduce asymmetry to the initial model. However, the characteristics of the selected distribution is important. Having weights with excessively large magnitudes will cause the majority of the neurons to be saturated. The resulting initialization point will be on a flat part of the objective function, and training will be slow [33]. On the other hand, very small weights will result in small gradients, hence slow training [68]. Under these circumstances, the best solution is to use a random distribution of medium-sized, zero-mean numbers. Hinton et al. use normal distribution to initialize the first truly deep learner [3], while Saxe et al. report that all zero-mean distributions perform similarly [69]. Martens proposes sparse initialization, where only a limited number of weights for each neuron are initialized, and the rest are left at zero [70]. Glorot and Bengio normalize the initialization weights with respect to the number of fan-ins and fan-outs of each neuron [44] (commonly referred to as *Xavier initialization* by various deep learning frameworks). He et al. adapt this method to be used with rectified linear unit (ReLU) activation functions [34]. Batch normalization is a recently discovered operation that is used in-between CNN layers that reduces the dependence on the exact distribution of initialization [38].

### 2.5.2. Initialization for performance

In Section 2.5.1, we have discussed the random initialization of weights. The main objective of a good random initialization is to set up suitable conditions for training. It is noticed that failing at this step can render a network virtually untrainable. Furthermore, initialization is not only important for avoiding failure, but also for obtaining optimal training results.

Training begins at a point on the objective function decided by the initialization. Backpropagation is a greedy algorithm, which is only able to converge to the local minimum. Since backpropagation will not be able to escape the basin of attraction it was initialized at, its optimization capability is strictly dependent on the initialization [13]. As

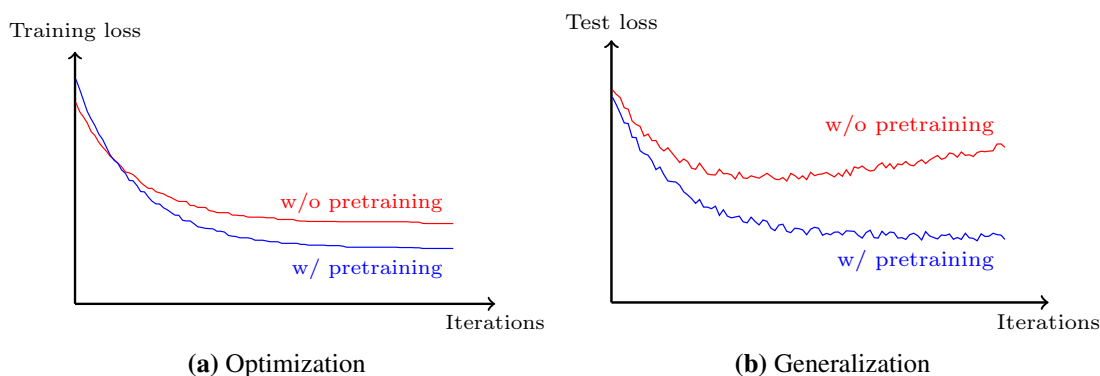


**Figure 2.9:** The objective function of a single parameter model. Training will converge to the global minimum only if the parameter is initialized at the part shown as a solid curve.

a result, a small subset of possible initializations will be able to converge to the global minimum (see Figure 2.9).

Our estimation of the target function is almost never exactly accurate, which is an issue. This may happen when the training data is not representative enough or the cost function is not suitable for the target task. The result is that even if we find the parameters that yield the minimum training loss, they may not be optimal for the target task. In this case, the solution that is more preferable to converge at is not at the global minimum, but most likely at (or near) one of the local minima. Again, a small subset of possible initializations will be able to converge to the desired minimum. Therefore, in the case where the objective function is not able to represent the target task accurately, a good initialization can provide regularization.

An interesting approach is trying to come up with a good initialization point manually. It is a frequently observed fact that the first layer of a trained CNN contains lots of Gabor-like filters [35, 71, 58]. Serre et al. set the first layer of a network to be Gabor filters based on biophysiological observations, and train only the second layer [72]. In response, Jarrett et al. report that even when compared with using random weights at the first layer, Gabor filters perform worse [35]. They also point out that even if we set the first layer to be Gabor filters, the following layers do not follow a perceptible pattern. Then, analytically manipulating weights is not a feasible initialization method.



**Figure 2.10:** The trajectory of training and test losses with and without pretraining.

### 2.5.3. Unsupervised pretraining

Before AlexNet [5], unsupervised pretraining was seen as a must step of training supervised deep learners [13]. It is not difficult to consider unsupervised pretraining as an initialization method, as its objective is to find a good starting point for training. During our discussion about random initialization, we have seen that some characteristics of the distribution that the parameters are drawn from may be critical. Erhan et al. tested if unsupervised pretraining simply provided a suitable distribution for initialization [56]. They pretrained a network, created a distribution from the resulting weights, and initialized networks with random samples from this distribution. Even though this performed better than random initialization from uniform distribution, unsupervised pretraining outperformed these two alternatives significantly. Another advantage of unsupervised pretraining reported by Erhan et al. is that it provides robustness against the differences in the random initialization seeds [56]. In other words, randomly initialized networks will have lesser performance variance if they are pretrained first. Similarly, Glorot and Bengio report that networks initialized with unsupervised training are more robust against the type of nonlinear activation function or random initialization distribution used [44].

Let us illustrate the effect of pretraining when our estimation of the target function is not accurate. See Figure 2.10. Training loss declines smoothly whether pretraining is applied or not. In the figure, pretrained training loss has saturated at a lower value, yet the opposite may also happen. However, the important metric is the test loss. Test loss is typically more volatile across iterations compared to training loss, as optimization is done with respect to the training data. Test loss without pretraining trends upward after some

point, which indicates overfitting. In contrast, the regularization provided by pretraining causes test loss to saturate at a low value.

Unsupervised pretraining seems to be the only feasible initialization method. However, AlexNet [5] and later CNNs trained with the ImageNet dataset regularly omit it. The reason for this is likely that the recent unsupervised training methods fall short when compared with the ever-improving “art” of training CNNs with backpropagation. The logical solution then is to apply supervised or self-supervised pretraining for initialization. This approach is highly analogous with transfer learning, which will be discussed in the next section.

## **2.6. Transfer Learning**

The most commonly considered case in machine learning is when the training data is sampled from the test data distribution. On the other hand, additional training data that is not directly relevant to the target task may be available. In general terms, extracting knowledge from such additional data and using it as leverage in the target task is called transfer learning [73].

More commonly, the additional data belongs to a different domain (i.e., sampled from another distribution). For example, New York Stock Exchange data can be used to train a machine learning-based trader that will operate in a smaller national stock market. Alternatively, the additional data may belong to a different task. A speech recognition dataset may be used to train a system that aims to identify music tracks.

Transfer learning requires the additional data to be of some relevance to the target task. If the knowledge that can be extracted from the additional data is not useful for the target task, the respective transfer will be fruitless. For example, it is not expected that representations learned from stock prices will be useful in speech recognition. However, this intuition may not be correct, as it can also be argued that even between these two tasks, some basic properties (e.g., frequency selectiveness) can be transferred. There is no method of estimating the degree of transferability between tasks and domains without experimentation. We are going to provide some interesting examples of unexpected transferability in Chapter 3.

### 2.6.1. Transfer deep learning

In practice, transfer learning is usually implemented as a two-step process. Firstly, a model is pretrained with the additional data. Then, the data representing the target task is used to fine-tune the model. Transfer learning is also utilized implicitly by all deep learners, even when there are no apparent pretraining and fine-tuning steps. We have mentioned that deeper architectures are more efficient, because their individual neurons have to be reused to construct a wide scale of representations. For example, a CNN may learn gradient-based high-level representations for image recognition. All of these high-level representations will be based on the same low-level oriented gradient operators. Additionally, learned representations are reused for different tasks. A CNN that has learned to detect cars and motorcycles can use the same representation for both car and motorcycle wheels.

Transfer learning is an inherent characteristic of deep learning, which may be the reason why deep learners respond well to its explicit applications. Bengio et al. investigate if shallow and deep models benefit from transfer learning differently [42]. The three tasks in the study are to classify the examples of a character set (digits, uppercase letters or lowercase letters) from the NIST dataset [74]. Out-of-domain data are produced by applying various transformations to the examples, such as blur, elastic deformation and occlusion. Transferring from different tasks and domains improved the deep model's performance in the original task greatly, beating previously published results. In contrast, the shallow model's performance is either not affected significantly, or degraded. Note the similarity of these results to the ones from the studies in Chapter 3; when only a few layers of the CNN is fine-tuned, data augmentation does not improve (or even degrade) performance.

Yosinski et al. investigate transfer learning in CNNs with a more complex task, ImageNet object classification [71]. The authors test the assumption that the initial layers of a CNN will be more general, and later layers will grow in specificity. It is reasonable to expect that transferring only the general layers will be preferable, as it is shown that some initializations (and hence transfers) may do more harm than good [13]. However, Yosinski et al.'s experiments show that even when two CNNs are trained for very different tasks (one classifies man-made objects while the other classifies natural objects), transfer-



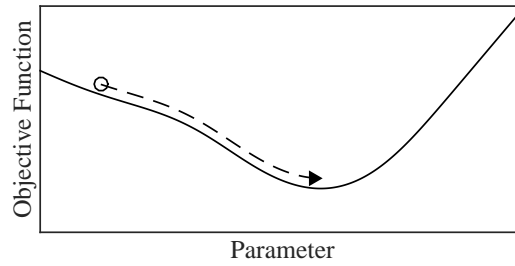
ring all layers results in the most significant increase in performance when followed by fine-tuning in the target task. This suggests that visual tasks can utilize a common set of representations, of which some are far more complex than Gabor features. A more extreme demonstration of this phenomenon is using the high level representations of a CNN trained to classify ImageNet objects to classify biomedical images without any fine-tuning with the target task (see Section 3.2). This shows that even when the tasks and domains are vastly different, transferring representations may be useful in vision applications.

### 2.6.2. Curriculum learning

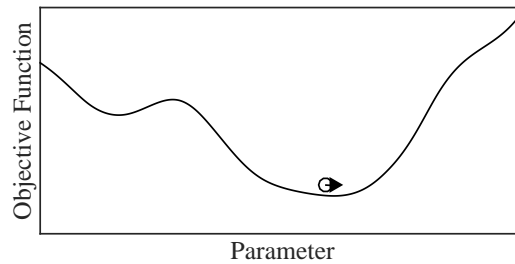
As discussed in Section 2.5.2, a single random initialization is not likely to deliver optimal results. State of the art results in the literature are produced by repeating the experiment many times with different seeds [27], and choosing a well-performing set of models to form an ensemble [37, 20]. This is akin to doing a random search in a nearly infinite parameter space, where each query takes a week of training time. There are two non-mutually exclusive improvements over this. The first one is to replace the gradient descent of the backpropagation algorithm with a less greedy alternative (e.g., a second order method [75]), so that the initial basin of attraction (w.r.t. gradient descent) can be escaped in favor of a better one. The second one is to start with an alternative objective function with a different topology than the target objective function. Even if the initialization point is at a bad basin of attraction in the target objective function, the alternative objective function is chosen such that it guides the training towards a good basin of attraction in the target objective function. Then, training is finalized in the target objective function, where the minimum of the good basin of attraction is achieved. Unsupervised pretraining and transfer learning were discussed as two methods where this approach was used.

Unsupervised pretraining and transfer learning make use of already available tasks to improve performance. There are two problems with this approach. First, given that the target task is sufficiently unique, no other task will be able to produce high level transferable representations. Second, each task switching will result in a noncontinuous change in the objective function. Instead, transferring from a series of continually evolving tasks will likely give better results with gradient descent. These two problems can be solved by specifically designing a set of training tasks for the target task.

Bengio et al. coin the term *curriculum learning* to describe pretraining with a spe-



(a) An easier version of the target task has a smoother objective function. This function is more convex, which means that it has less local minima to get trapped in.



(b) The objective function of the target task is more non-convex. Still, the optimization ends in the global minima because of good initialization.

**Figure 2.11:** A curriculum is composed of smoothed versions of the target task. Training at each task starts at the empty circle and ends at the arrow head. Note that unlike Figure 2.9, all initializations will converge to the global minimum when curriculum learning is used.

cially designed set of tasks [76]. The authors state that the benefits of curriculum learning are similar to of unsupervised pretraining. Indeed, unsupervised pretraining can be considered as a type of curriculum learning, where estimating  $P(X)$  is less complex than estimating  $P(Y|X)$ .

The main difficulty in training for a complex task is due to the non-convexity of the objective function. Instead of training directly with the highly non-convex target task, curriculum learning gradually introduces non-convexity in a set of less complex tasks. Bengio et al. propose two approaches to design these prior tasks [76]. In the first one, the target objective function is already known, and its more convex versions are produced by smoothing (see Figure 2.11). This is not applicable for most cases, as in real applications, the entire point of training is to explore the target objective function. In the second approach, the difficulty of the examples from the target dataset are rated, and used with increasing order of difficulty. Kumar et al. propose *self-paced learning* to train with a dataset using the easier examples first [77]. In later work, Gülçehre and Bengio separate

the target task into subtasks and train the network with these in sequence [24].



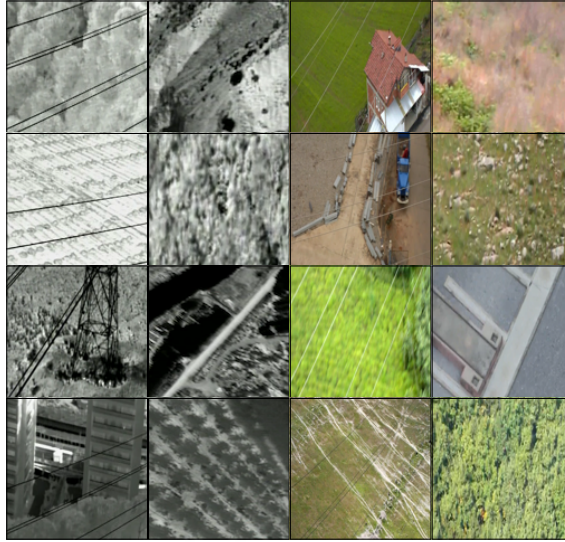
### 3. IMAGENET PRETRAINING

Regularization puts a soft limit on the flexibility of the model, which reduces overfitting. It has been shown that unsupervised pretraining after random initialization provides regularization; hence, improves generalization [41, 56, 44]. In fact, unsupervised pretraining with the target task data was seen as a necessary initial step before supervised training for some time [2]. However, the fact that AlexNet, a particularly deep architecture for its time, was trained in a purely supervised manner for ILSVRC has proven that this was not the case [5].

The understanding of the fact that deep models can be trained in a purely supervised manner with enough training data did not happen to be the end of pretraining, but only transformed it. Instead of unsupervised pretraining, the literature moved on to supervised pretraining on a large dataset for model initialization. For visual tasks, the object classification task of ILSVRC became the default pretraining task (generally referred to as *ImageNet pretraining*) [23], and this pretraining method consistently delivers state of the art results in various tasks [78, 79].

It has been proposed to use ImageNet pretrained models for general-purpose visual recognition [80, 81]. This approach yields state of the art results in tasks where natural images are used (e.g., object detection [82], image captioning [83]). However, it is easy to predict that ImageNet pretraining would be less useful if the data distribution of the target task is different, which is referred to as a *domain shift*. For example, although Liu et al. use an ImageNet pretrained model to locate the face regions, they use a separate model trained only with facial images to predict face attributes [43]. In the case where the domain shift is extreme, ImageNet pretraining is completely omitted [84, 85].

In this study, we present case studies where the domain shift between the ImageNet pretraining data and target task data is extreme. In the first one, it is shown that ImageNet pre-training can be used to reach near-perfect results for recognizing and localizing power lines from aerial images, even when the images are in infrared spectrum [86]. In further work, this method has been extended for visualization of recognized power lines for easy localization [87]. In the second case, ImageNet pre-training is used to learn representations to be used in cell classification from grayscale microscopic images, and it has been



**Figure 3.1:** Examples from the aerial image dataset. The first two columns are infrared images with and without power lines, the following two columns are visible light images with and without power lines. The first two rows are easier examples, based on the metric proposed in [89].

shown that these representations can be used to produce acceptable results even without fine-tuning [88]. These studies show that even when the target task is from a vastly different domain (aerial, microscopic, infrared and grayscale, instead of colored natural images), ImageNet pre-training is a viable model initialization method.

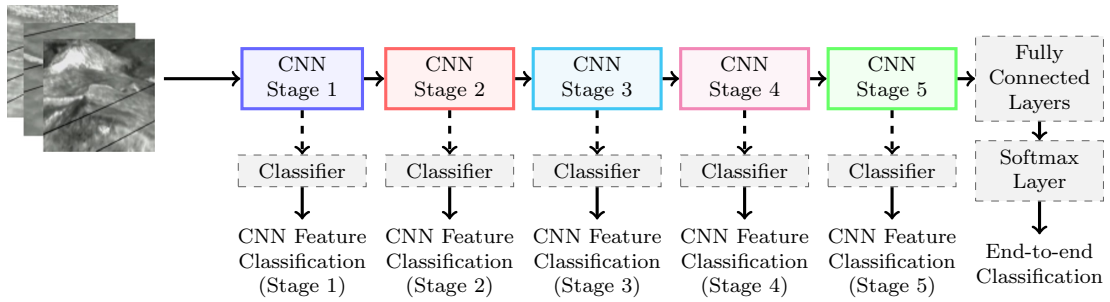
### 3.1. Case Study 1–Power Line Recognition from Aerial Images<sup>1</sup>

The subject of the first case study is to recognize power lines from aerial images. We will show that although the data distribution for this task is from a completely different domain, ImageNet pretraining followed by a brief fine-tuning results in almost perfect recognition performance. Since we are only going to be presenting our findings that relate to this thesis directly, the reader is advised to refer to the original paper [86] if they are more interested in the specific use-case.

#### 3.1.1. Aerial image dataset

For this case study, we used an aerial image dataset that we have generated in cooperation with the Turkish Electricity Transmission Corporation (TEIAS). A helicopter mounted imaging system was used to capture visible light (VL) and infrared (IR) videos from the

<sup>1</sup>This section is adapted from [86].



**Figure 3.2:** Two alternative methods for using CNNs for power line recognition: end-to-end classification and CNN feature classification. In end-to-end classification, the network is trained jointly. In CNN feature classification, the feature extractor and the classifier are trained sequentially. The disconnect is represented by dashed arrows.

air. The video resolutions were  $576 \times 325$  for IR and  $1920 \times 1080$  for VL. We inspected the videos, manually selected examples that represent the presence or absence of power lines, and resized them to  $128 \times 128$  (see Fig. 3.1). The dataset is composed of 2000 positive examples and 4000 negative examples for each domain respectively. The videos are captured from 21 different geographical locations in Turkey. The examples are chosen to provide a variety of difficulties, due to different backgrounds, lighting and weather conditions. Using the method in [89], example difficulties are graded into two tiers (see Fig. 3.1). Finally, the dataset is hosted in a public web repository [90] with the corresponding localization ground truth [91].

### 3.1.2. Proposed method

We propose two alternative methods for the usage of CNNs for power line recognition (see Fig. 3.2). The first method is end-to-end classification. In this method, we start with a CNN that was designed to be used for ILSVRC image classification. We replace the final layer of this model with a randomly initialized softmax layer with 2 outputs for binary classification. Then, we train only this final layer until convergence. Following this, we jointly fine-tune the feature extraction and classification parts.

The second method utilizes the same CNN as a feature extractor. We use only the parts up to a certain CNN stage, and remove the further layers. The output of the partial CNN is flattened, dimension-reduced with principal component analysis (PCA), and fed into a classifier. We train this classifier separately from the CNN.

We have used ResNet-50 [20] and VGG-19 [19] as the preferred architectures.

These two architectures were chosen because of their relative performances at the ILSVRC image classification task, where ResNet-50 outperforms VGG-19. We will be investigating if the performance at this general visual task is indicative of the performance at the specific task of power line recognition.

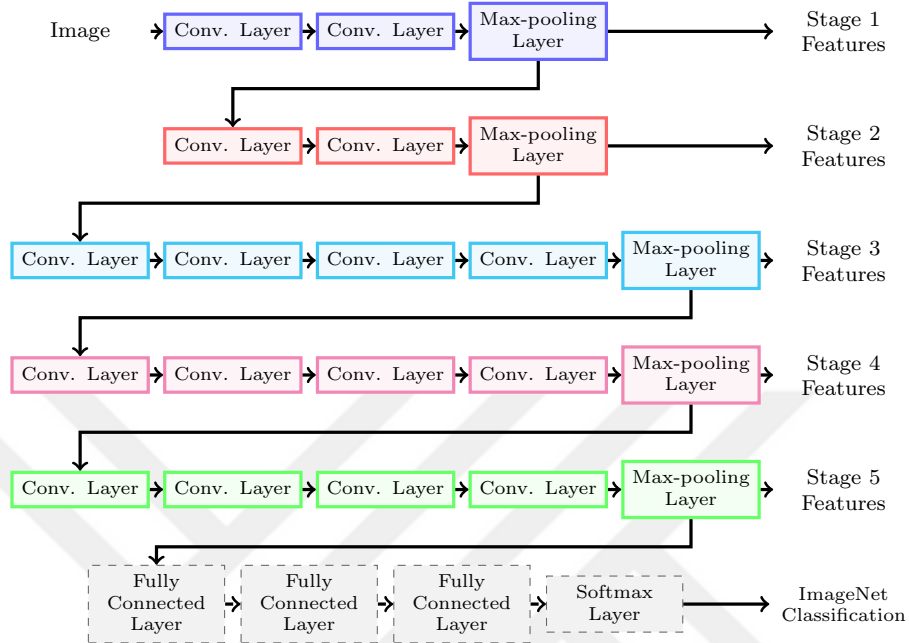
The main contribution of VGG models was showing that a fixed kernel size of  $3 \times 3$  works as good as larger kernel sizes (because earlier architectures were manually tuned for each layer [5]), and greatly simplifies the architecture design process. A single VGG-19 net is reported to achieve 7.0% top-5 test error in the ILSVRC image classification task [19]. For a simpler analysis, we considered the VGG-19 architecture as a composition of five convolutional stages, as it was implemented for Keras [92] (see Fig. 3.3a).

Szegedy et al. simplified the architecture design further, by first designing inception blocks, and using them to build GoogLeNet [37]. Similarly, ResNet is designed as a combination of blocks. The main difference of ResNet is its being much deeper than earlier architectures. This was achieved by utilizing batch normalization [38] and residual connections [22], both of which help with vanishing or exploding gradient problem seen in excessively deep models. A single ResNet-50 net is reported to achieve 5.25% top-5 validation error in the ILSVRC image classification task [20]. Similar to VGG-19, we analyzed this architecture as a composition of five convolutional stages, based on the implementation we have used [92] (see Fig. 3.3b).

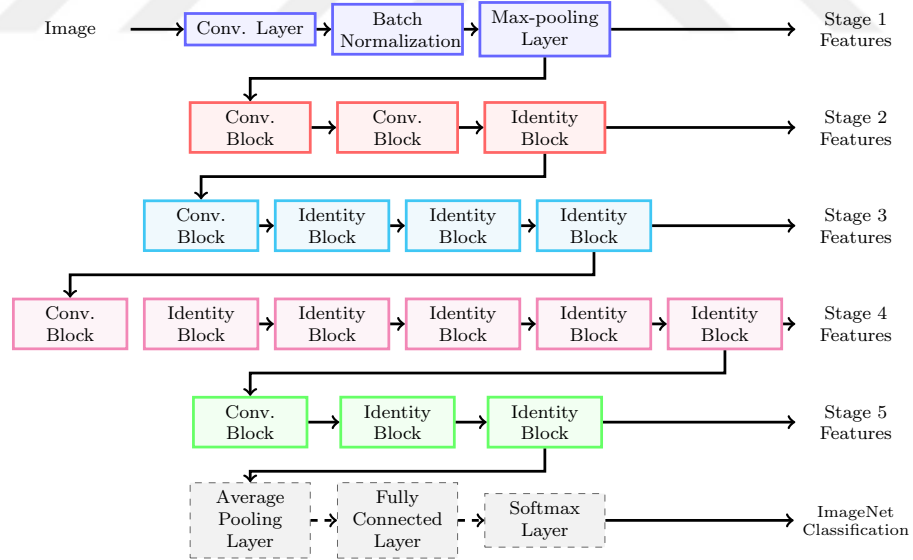
We used three different methods to classify CNN features. Support vector machines are popular classifiers that aim to obtain the hyperplane that separates the support vectors with the largest margin possible [28]. Naive Bayes classifiers assume that the classes can be represented as stationary Gaussian distributions, and assuming statistically independent samples, they maximize the a posteriori probability of an observation to reach a decision [93]. Conventional decision trees apply classification by splitting the feature space into smaller sub-categories as long as classification accuracy is not met. Random forests are ensembles of shallow decision trees for improved robustness against overfitting [94].

### **3.1.3. Experimental results**

In this section, we present experimental results for the two proposed methods: end-to-end classification and CNN feature classification. Additionally, we investigate the usage of



(a) VGG-19



(b) ResNet-50

**Figure 3.3:** CNN architectures used in the study, VGG-19 [19] and ResNet-50 [20]. The stages of the architectures are illustrated in different colors. Note that the convolutional and identity blocks of ResNet-50 are composed of multiple layers.



different net architectures and image preprocessing methods.

The abbreviations used in the tables are as follows:

IR: Infrared            SVM: Support Vector Machine

VL: Visible Light    NB: Naive Bayes

Neg: Negative        RF: Random Forests

Pos: Positive

### **3.1.3.1. *Implementation details***

For end-to-end classification, the final layer of the CNN designed for ImageNet classification is replaced with a binary softmax layer. Being the usual practice, only this layer is trained first. Then, Stage 5 and the following layers (see Fig. 3.3) are fine-tuned jointly. In CNN feature classification, flattened outputs of convolutional stages are used as CNN features. Since these features are excessively large, they are dimension-reduced to size 1,024 using PCA.

Keras with TensorFlow backend was used for the CNN [92], and Weka was used for the classifiers [95]. We ran all experiments with 10-fold cross-validation. For each fold, the dataset was segmented as 70% training data, 20% validation data and 10% test data. The learning rate for the CNN final layer was started at 0.1, and halved five times when the validation loss stopped decreasing. Fine-tuning learning rate was annealed the same way, but it was initialized at 0.01. Weight decay was set to be 0.001 for all layers. The classifier parameters were kept as the default values in Weka 3.8.

Two popular alternatives for image pre-processing were tested: (i) forcing zero-mean by subtracting the dataset average (referred to as mean subtraction), and (ii) linear normalization to 0–1 scale by dividing by 255.

### **3.1.3.2. *End-to-end classification***

We start our experiments with CNNs pre-trained for ILSVRC image classification (from now on to be referred as ImageNet pre-trained nets). These nets include filters such as edge and blob detectors in the earlier layers, which are likely to be useful in a majority of visual tasks. However, they also come with a redundant specialization in recognizing objects [58]. Therefore, it is not predictable if they will work well for the task at hand.

**Table 3.1:** Classification errors in percentages for the end-to-end classification method.

(a) ImageNet pre-training and mean subtraction preprocessing.

	<b>IR</b>	<b>VL</b>	<b>IR</b>	<b>VL</b>
<b>VGG-19</b>	1.85	8.167	4.967	8.167
<b>ResNet-50</b>	0.65	1.0	0.25	0.267
	<b>Trained last layer</b>		<b>Trained last layer &amp; Fine-tuned Stage 5</b>	

(b) ImageNet pre-training and 0–1 normalization preprocessing.

	<b>IR</b>	<b>VL</b>	<b>IR</b>	<b>VL</b>
<b>VGG-19</b>	57.183	44.433	1.467	1.967
<b>ResNet-50</b>	0.55	4.7	0.217	0.55
	<b>Trained last layer</b>		<b>Trained last layer &amp; Fine-tuned Stage 5</b>	

(c) Random weights and mean subtraction preprocessing.

	<b>IR</b>	<b>VL</b>	<b>IR</b>	<b>VL</b>
<b>VGG-19</b>	22.283	21.633	27.933	17.5
<b>ResNet-50</b>	40.517	48.983	13.333	20.5
	<b>Trained last layer</b>		<b>Trained last layer &amp; Fine-tuned Stage 5</b>	

Tables 3.1a and 3.1b show the results with ImageNet pre-trained models, and Table 3.1c show the results with a randomly initialized model.

As discussed earlier, we train the randomly initialized last layer individually, then fine-tune the net. The entire net has millions of parameters that can be fine-tuned. The number of free parameters increase the expressive power of the model. However, if there is not much training data, this also causes the model to overfit. Since our training set is relatively small, we must limit the number of free parameters in the model. For this reason, we limited the fine-tuning to only the final stage (Stage 5 in Fig. 3.3) and the following layers. In Tables 3.1a–3.1c, the results are given both for training the last layer, and fine-tuning the last stage.

See Table 3.1a for the results where the ImageNet pre-trained nets are fed mean subtracted images. We can see that just by training the last layer, we achieve considerable performance with features learned from ImageNet pre-training. ResNet-50 performs significantly better than VGG-19. Following this, we fine-tune the final stage of the nets. Here, we see that ResNet-50 performance is improved even further, resulting in the best

**Table 3.2:** Confusion matrices for the end-to-end classification method (trained final layer and fine-tuned Stage 5, ImageNet pre-training, mean subtraction preprocessing). Rows are ground truths, and columns are predictions.

<b>ResNet-50</b>					
<b>IR</b>	<b>Neg (GT)</b>	<b>Pos (GT)</b>	<b>VL</b>	<b>Neg (GT)</b>	<b>Pos (GT)</b>
<b>Neg</b>	3994	6	<b>Neg</b>	3994	6
<b>Pos</b>	9	1991	<b>Pos</b>	10	1990

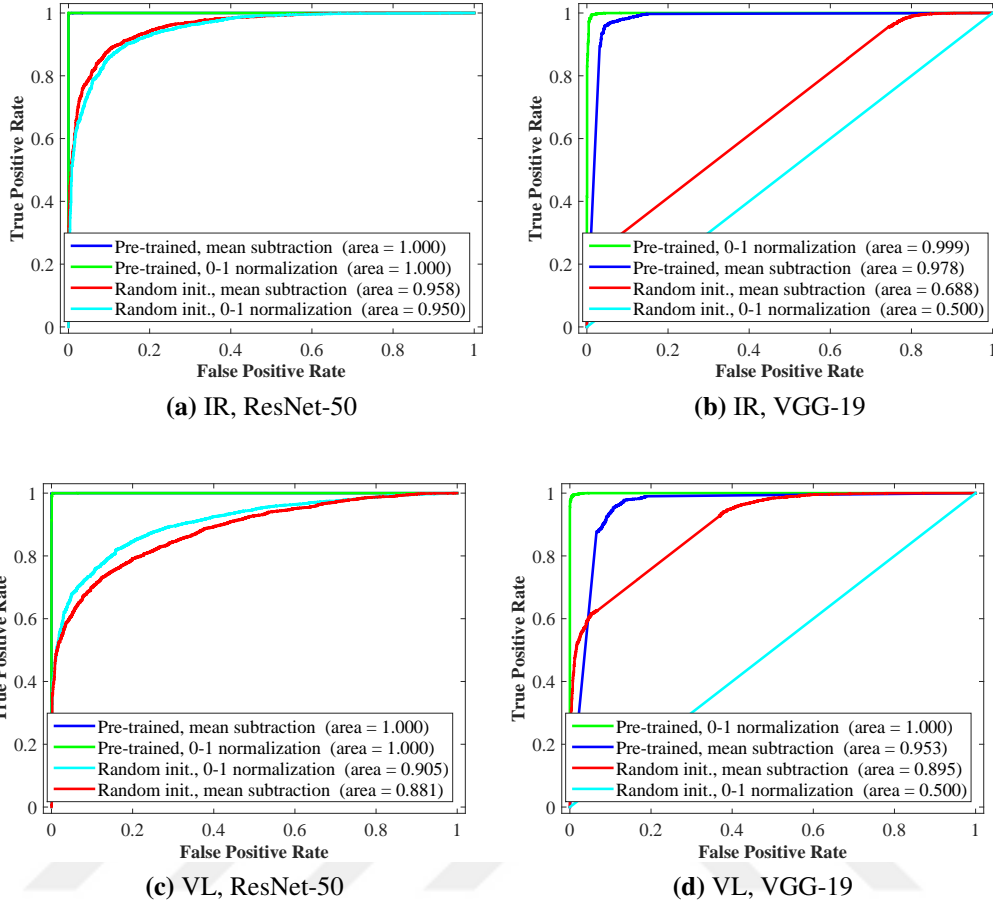
  

<b>VGG-19</b>					
<b>IR</b>	<b>Neg (GT)</b>	<b>Pos (GT)</b>	<b>VL</b>	<b>Neg (GT)</b>	<b>Pos (GT)</b>
<b>Neg</b>	3836	164	<b>Neg</b>	3691	309
<b>Pos</b>	134	1866	<b>Pos</b>	181	1819

performance that will be reported in this study. On the other hand, IR performance of VGG-19 is not improved. See Table 3.2 for the confusion matrices where Stage 5 is fine-tuned.

We repeat the previous experiment by using 0–1 normalization instead of mean subtraction as the preprocessing method. See the results in Table 3.1b and compare with Table 3.1a. ImageNet pre-training is done with mean subtraction. This dynamic range alteration results in inferior performance. What is interesting is that once we fine-tune Stage 5, the performance improves dramatically, and becomes comparable to using mean subtraction.

CNN architectures are inherently frequency selective. For this reason, they are sensitive to Gabor wavelet-like structures, even when they are composed of random weights [69]. Then, it is reasonable to expect untrained nets to perform reasonably well in power line recognition, as they do in other tasks [35]. In other words, ImageNet pre-training may not have meaningful effect in achieving the performance in Tables 3.1a and 3.1b. To test the effect of ImageNet pre-training, we generated nets with random weights using Xavier initialization [44], and repeated the experiments (see Table 3.1c for the results). It is clear that ImageNet pre-training was significantly beneficial for power line recognition, even when the images were from the IR spectrum. However, untrained nets were also able to deliver better than random predictions. See Figure 3.4 for the receiving operator characteristic curves of the experiments in this section.



**Figure 3.4:** Receiver operating characteristic curves of the end-to-end classification method. The legend is in the order of decreasing area under the curve. Note that the curves overlap and occlude each other in some figures.

### 3.1.3.3. *Classifying CNN features*

In this section, we investigate using the CNN as a feature extractor, and using a variety of classifiers. The CNNs learn fundamental representations in the earlier layers, such as directed edges, blobs and patterns. As we move on to higher layers, representations grow abstract and data-specific. Considering that the data distribution of the ImageNet dataset and our aerial images dataset is vastly different, the more abstract representations from the higher layers may not be ideal for us. To test this hypothesis, we extracted features from different stages of the CNNs, and fed them into classifiers.

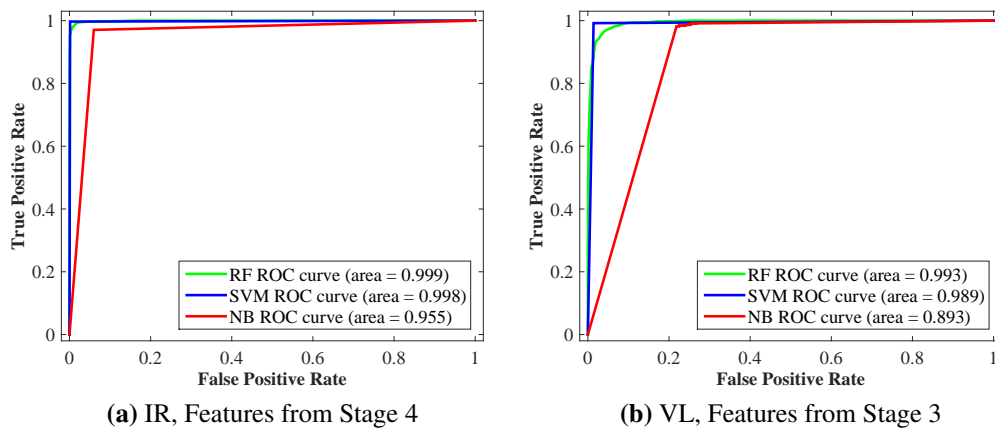
See Table 3.3a and Table 3.3b for the CNN feature classification results with mean subtraction preprocessing. It is clear that similar to the end-to-end classification results, ResNet-50 outperformed VGG-19. Again, in accordance with the earlier results, infrared

**Table 3.3:** Classification errors in percentages for the CNN feature classification method with mean subtraction preprocessing.

(a) VGG-19						
	SVM	NB	RF	SVM	NB	RF
<b>Stage 1</b>	12.383	29.917	26.2	4.933	35.233	24.95
<b>Stage 2</b>	7.4	28.633	24.433	1.9	37.883	26.8
<b>Stage 3</b>	2.267	14.167	18.667	1.15	27.617	20.6
<b>Stage 4</b>	0.917	17.4	16.467	2.0	30.55	21.233
<b>Stage 5</b>	0.85	21.183	11.75	2.85	30.15	21.9
	<b>IR</b>			<b>VL</b>		
(b) ResNet-50						
	SVM	NB	RF	SVM	NB	RF
<b>Stage 1</b>	10.07	31.183	23.4	2.717	38.233	26.733
<b>Stage 2</b>	3.95	15.65	19.267	1.7	23.483	20.317
<b>Stage 3</b>	0.833	16.733	11.85	0.883	14.733	19.167
<b>Stage 4</b>	0.417	7.767	10.417	1.67	29.917	19.3
<b>Stage 5</b>	0.417	19.75	11.1	1.083	33.483	22.417
	<b>IR</b>			<b>VL</b>		

**Table 3.4:** Confusion matrices for the best CNN feature classification method (SVM classifier, ResNet-50 architecture, Stage 4 features for infrared and Stage 3 feature for visible light images, mean subtraction preprocessing). Rows are ground truths, and columns are predictions.

IR	Neg (GT)	Pos (GT)	VL	Neg (GT)	Pos (GT)
<b>Neg</b>	3995	5	<b>Neg</b>	3978	22
<b>Pos</b>	20	1980	<b>Pos</b>	31	1969



**Figure 3.5:** Receiver operating characteristic curves of the CNN feature classification method. ImageNet pre-trained ResNet-50 model is used with mean subtraction preprocessing. Stage 4 for IR and Stage 3 for VL are shown because they delivered the best results (see Table 3.3b).

**Table 3.5:** Cumulative running times of the CNN models for a single image in milliseconds.

	<i>Stage 1</i>	<i>Stage 2</i>	<i>Stage 3</i>	<i>Stage 4</i>	<i>Stage 5</i>	<i>End-to-end</i>
<b>VGG-19</b>	3.7	4.1	5.4	8.0	10.0	10.6
<b>ResNet-50</b>	2.2	7.9	11.1	16.6	19.3	21.7

images are classified more easily. If we compare the classifiers, SVM yields the best classification performance across all conditions. See Table 3.4 for the confusion matrices, and Figure 3.5 for the receiving operator characteristic curves with the best configurations in this section.

Let us compare the performances of features extracted from different stages. Nearly in all cases where naive Bayes or random forest classifiers are used, Stage 3 features yielded the best results. In contrast, SVM performance improves with higher-level features. The best performances in Table 3.3a and 3.3b are comparable to best performances in Table 3.1a without fine-tuning, which was equivalent to extracting features from Stage 5 and classifying them with a single layer net.

#### **3.1.3.4. *Running time***

An effective power line warning system should warn pilots at a distance that would be adequate to take the necessary risk-avoidance maneuvers. In this respect, the running time of the algorithm is an important factor, as the helicopter may be approaching the hazard during the running time.

The running times given in Table 3.5 are obtained with an Nvidia GTX 1080 GPU. The best performing configuration, end-to-end classification with ResNet-50, runs in 21.7 ms, which is reasonable for a real time-application. Running times for PCA projection and classification with SVM, NB, RF are not significant ( $< 0.5$  ms) and thus are omitted. Note that it is possible to optimize the method for a lower running time by using lightweight architectures such as MobileNets [96], pruning existing architectures [97], or using specialized hardware [98].

**Table 3.6:** Best performing configurations with each proposed method at infrared and visible light images.

	<b>Proposed Methods</b>	<b>Error (%)</b>
IR	End-to-end CNN classification (ResNet-50, ImageNet pre-training, fine-tuned Stage 5, 0–1 normalization preprocessing)	0.217
	CNN features + Support Vector Machine (ResNet-50, Stage 4, ImageNet pre-training, mean subtraction preprocessing)	0.417
	DCT features + Random Forest (128×128 descriptors, classical selection) [99]	3.75
VL	End-to-end CNN classification (ResNet-50, ImageNet pre-training, fine-tuned Stage 5, mean subtraction preprocessing)	0.275
	CNN features + Support Vector Machine (ResNet-50, Stage 3, ImageNet pre-training, mean subtraction preprocessing)	0.883
	DCT features + Random Forest (128×128 descriptors, classical selection) [99]	14.6

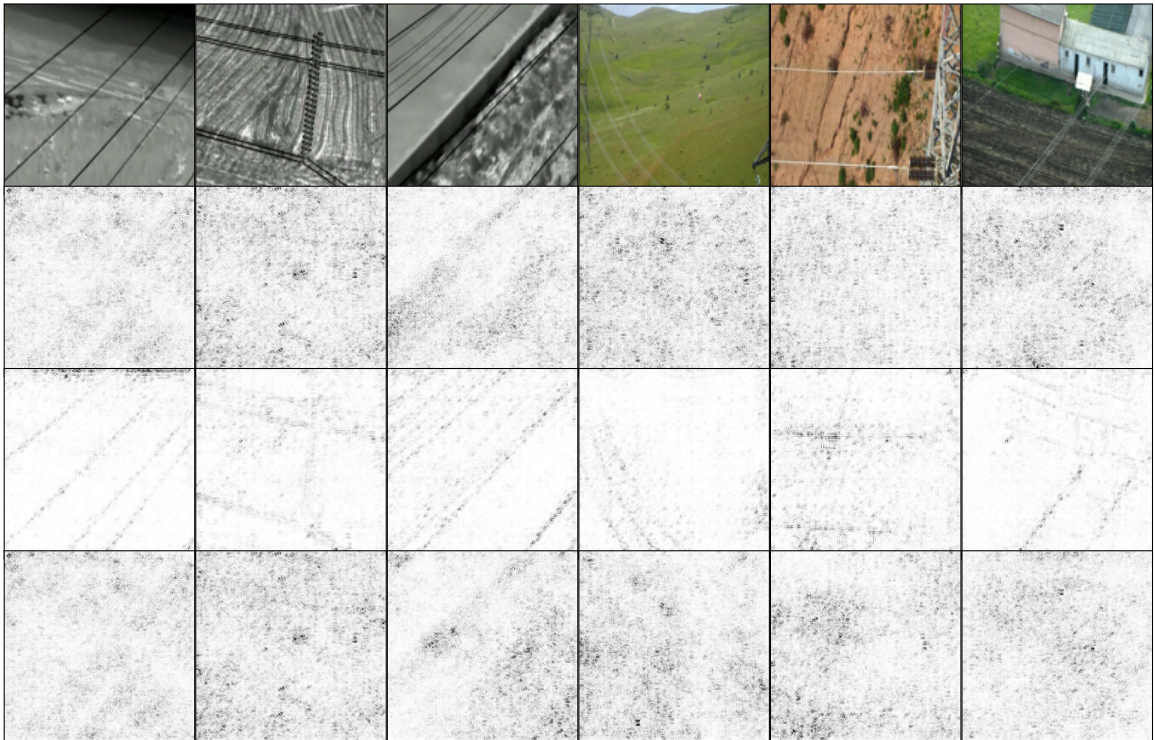
### 3.1.3.5. Comparison of methods

In this section, we present a quantitative comparison of the best performances of various methods on respective datasets (see Table 3.6). We can see that the results are consistent for both spectra. End-to-end classification with fine-tuning gives the best results, and using the CNN as a feature extractor gives comparable results. We compared these results with a recent method that uses DCT features for classification [99]. This method is applied with various parameters and classifier types (SVM, RF or NB), and the best results are given in Table 3.6. It is clear that the proposed application of deep learning on power line recognition provides a significant improvement.

### 3.1.4. Visualization<sup>2</sup>

We have presented the performance of the proposed method with quantitative experiments. In this section, we are going to analyze the proposed method (specifically, the model from Table 3.1a) further, using various visualization techniques. Let us start by investigating the saliency maps for positive examples. A saliency map for an image is obtained by backpropagating the gradient to the respective image. For this purpose, we used guided backpropagation [101], where only positive ReLU activations propagate gradients. This method is particularly suitable for the task at hand, because it emphasizes object contours, rather than its body. See Figure 3.6 for saliency maps obtained using

<sup>2</sup>This section is adapted from [87].

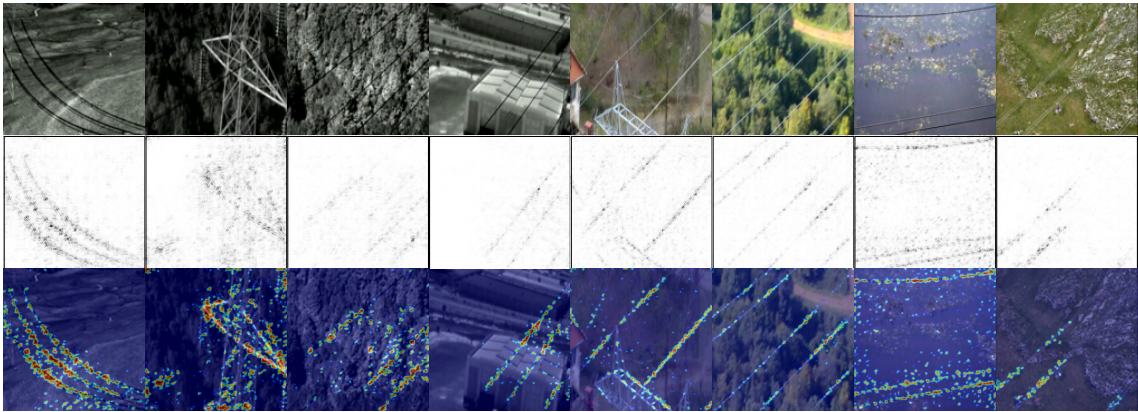


**Figure 3.6:** The first row is images from the dataset [90], following rows are saliency maps generated using regular gradients [100], guided backpropagation [101] and integrated gradients [102], respectively. Darker colors indicate higher saliency.

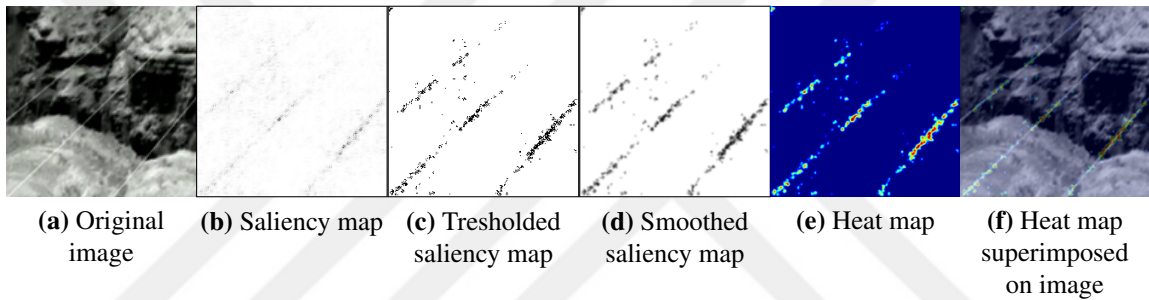
different backpropagation methods. The proposed method is sensitive to power lines that appear as curves, which implies superiority over methods that assume collinearity. Another interesting point is that the model has learned to recognize pylons, and use them as evidence of power lines. Again, a hand-crafted method based on line detection would be unable to do this. The linear structures of the buildings can be seen to be ignored, which is desirable. We can also evaluate the dataset based on these saliency maps. Specifically, the fact that the model focuses on the power lines rather than the background implies that the dataset is not biased in the way that positive examples share a similar background.

We can also use the saliency map to provide visual feedback to the pilot, as proposed in [87]. See Fig. 3.7 for examples. To achieve this, we thresholded the saliency map using Otsu’s method [103], applied Gaussian blur, and superposed it over the original image with a colormap (see Figure 3.8 for intermediate steps). The saliency map is obtained with a single backwards pass, meaning that the extra computation is comparable to what is required for the proposed method. Therefore, our proposed method can be extended to provide visual feedback and still works in real-time.



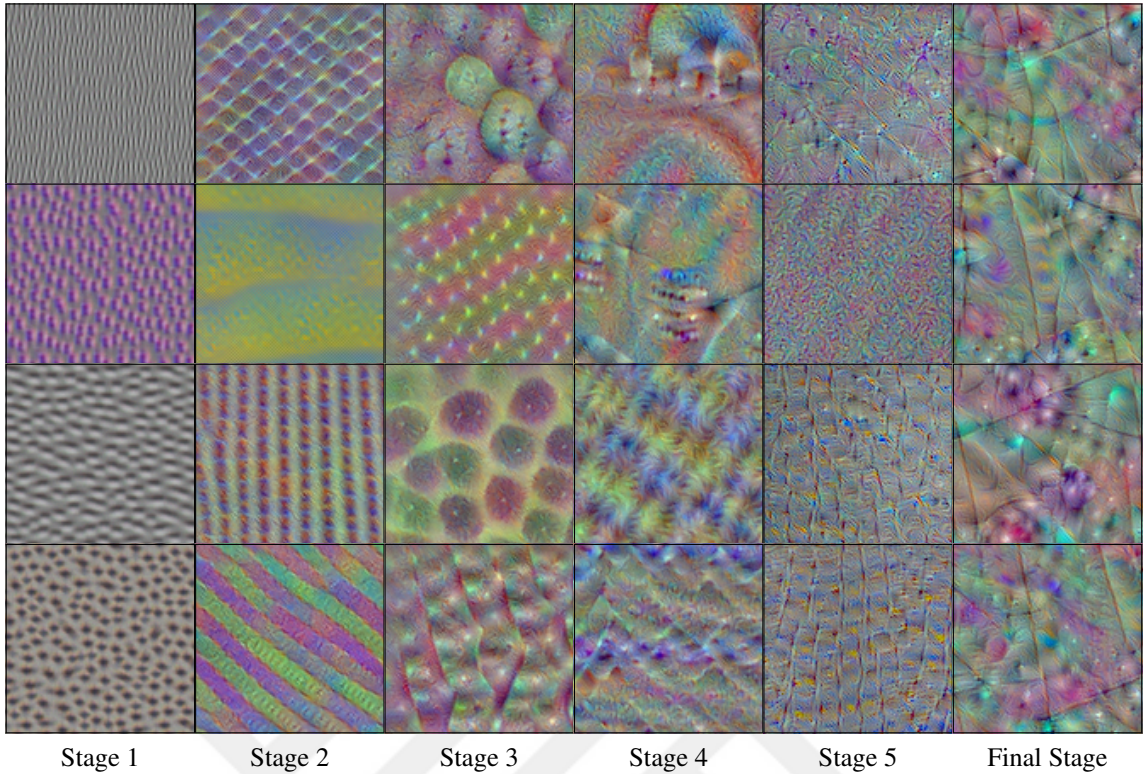


**Figure 3.7:** The first row shows input images, second row shows saliency maps obtained by guided backpropagation [101] (darker color indicates higher saliency), and third row shows a visualization technique that can provide visual feedback for the pilot.



**Figure 3.8:** Steps to present the original image and the saliency map simultaneously. The saliency map is thresholded, the thresholded image is smoothed, and a heat map is produced from the resulting image. The heat map is superimposed on the original image transparently.

To further analyze how the proposed method functions, let us visualize the activations. This is done by choosing a single neuron output from a stage, and optimizing for the highest activation. See the results in Fig. 3.9. Since this model was initialized by ImageNet pre-training we can observe that the representations grow higher in level through Stage 1–4. However, due to the fine-tuning that we apply, the representations regress to a much lower level in Stage 5. Then, a combination of these representations are used to recognize power lines in the Final Stage, as evident from the parallel linearities and pylon-like structures. We can also say that the final stage has learned to handle power line appearances of any orientation, based on the variety in the figure.



**Figure 3.9:** Activation visualizations for the model in Table 3.1a. The images are zoomed  $\times 2$  for better viewing. The model was initialized by ImageNet pre-training, then Stage 5 and Final Stage were fine-tuned for power line recognition.

### 3.1.5. Conclusion

In this study, we proposed two CNN-based power line recognition methods to be used in a real-time warning system. Unlike previous methods, where the cable lines were localized, we consider the problem as a binary classification where the scene contains a power line, or not. Both of the proposed methods use CNNs designed for ImageNet object recognition. In the first method, end-to-end classification, the CNN is modified for the target task, and trained jointly. In the second method, CNN feature classification, features are extracted from the intermediate stages of the CNN, and fed into a classifier.

The best results were obtained with end-to-end classification, where a ResNet-50 model was pre-trained with the ImageNet dataset, and its last stage was fine-tuned with power line images. In nearly all experiments, infrared images were classified more successfully. This shows that it is preferable to use infrared imaging for a power line warning system. However, the performance with visible light images was also reasonable.

Overall, ResNet-50 performed better than VGG-19, which implies that a model's

ImageNet performance tends to be indicative of its performance at other tasks. This would mean that the results we have achieved can be improved by replacing the pre-trained network with a newer one that performs better at high-level visual tasks. In addition, we show that ImageNet pre-training is significantly beneficial for power line recognition, which is consistent with recent findings in other domains [104].

Contrary to our initial expectation, higher-level features yielded better performance with the CNN feature classification method. This indicates that, while the representations in the final layers are being optimized to disentangle higher-level factors of variation (i.e., what object does an image contain), they also become better at disentangling lower-level factors of variation. Therefore, high-level features from pre-trained nets are concluded to be beneficial in a wide variety of visual tasks. End-to-end classification surpassed CNN feature classification, because it uses high-level features and allows fine-tuning.

In the experiments without fine-tuning, we observed that the preprocessing method was critical. Specifically, one should use the preprocessing method that was used in pre-training, which was mean subtraction in our case. However, fine-tuning nullifies the effect of the difference between the preprocessing methods used in pre-training and training.

Finally, we showed that even though the architectures we used were designed for ImageNet object recognition, they performed well at the target task. Aerial images of visible light and infrared spectra constitute a considerably different domain than the ImageNet dataset. Yet, ImageNet pre-training was observed to affect the experimental results positively. Moreover, the architecture that performs better at ImageNet object classification also performs better at power line recognition, regardless of if they are pre-trained or not. Considering that better performance at a general purpose visual task results in better performance at this specific visual task, the premise of a unified net for all visual tasks looks promising.

### **3.2. Case Study 2—HEp-2 Cell Classification<sup>3</sup>**

For applications for which the available data is not adequate to train a deep neural network from scratch, training for similar objectives can be used as an initialization strategy. In this study, cell images are classified using a deep neural network pretrained to classify objects in natural images. Even though classification of natural images and cell images

---

<sup>3</sup>This section is adapted from [88].

are very different objectives, this approach allows cell images to be classified with relatively high accuracy. The results show that features used for visual classification by deep convolutional neural networks may be more universal than assumed.

### 3.2.1. Introduction

Image processing and pattern recognition methods are commonly used in biomedical imaging to assist medical personnel [105, 106]. These methods can enhance biomedical images to make the relevant information more perceptible for humans [107] and automate relatively easier tasks [108]. Since qualified medical personnel is a limited and valuable kind of human resource, automating parts of their jobs creates significant value.

Indirect immunofluorescence is an important tool in diagnosing autoimmune diseases. In this method, a sample of the subject's blood serum is applied on the HEp-2 cells on a slide. The characteristics of the antibodies of the subject can be determined according to their interactions with the HEp-2 cells. Different kinds of interactions cause cells to be stained with different patterns [109]. The images of these cells are examined by expert personnel to determine the frequency of each staining type. For the results to be reliable, this may need to be repeated by multiple experts. This application can be described as a vision-based pattern recognition problem. Initially, methods developed for this problem can be used to validate human results. As performance increases, the developed methods can be used to automate the process without any human effort.

In this study, the performance of an ImageNet pretrained GoogLeNet [37] at recognizing staining patterns of HEp-2 cells is measured<sup>4</sup>. This is done by removing the final layer of GoogLeNet, which is a softmax classifier, and replacing it with a multi-class support vector machine. The remaining layers of the CNN are not fine-tuned at all using the HEp-2 cell images. Since the CNN is pretrained to recognize objects in natural images, the learned representations are not expected to be useful in this application. The experimental results indicate that the learned representations are indeed useful for cell classification, which gives important clues about the generalness of representations that CNNs utilize.

---

<sup>4</sup>Note that GoogLeNet was the state of the art architecture at the time this study was done.

### 3.2.2. Related work

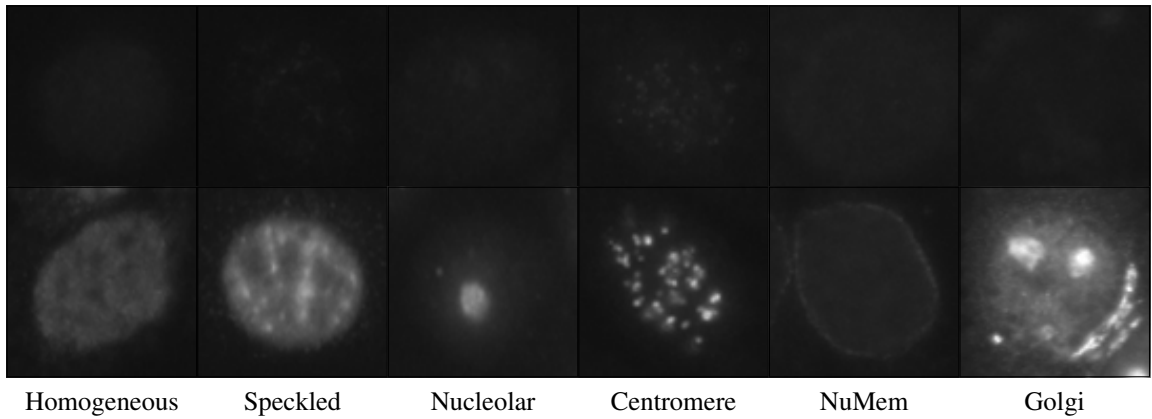
The performance metrics in this section are the mean class accuracies for the dataset compiled for the challenges held in International Conference on Image Processing (ICIP) 2013 and International Conference on Pattern Recognition (ICPR) 2014. A training set is made available to the participants, and the performances were measured with a secret test set. The methods that were not submitted to these challenges could only be tested using the training set. The results measured in this way are significantly lower than the performances measured in the challenge (e.g., 6.0% difference for Manivannan et al. [110]). While comparing performances, this factor should be kept in consideration.

The challenge held in ICIP 2013 was won by Shen et al. with 80.7% accuracy [111], and the challenge held in ICPR 2014 was won by Manivannan et al. with 87.1% accuracy [110, 112]. Shen et al. have classified SIFT features and frequency histograms of local binary descriptors using an SVM. Manivannan et al. decompose the image spatially as concentric rings and extract various local descriptors from them, which are fed to multiple SVMs. The classification results are obtained as averages of SVM outputs.

Qi et al. use a local binary descriptor variant and Fisher vectors obtained from RootSIFT descriptors as features [113]. These features are classified using an SVM and 80.0% mean class accuracy was obtained. Extracting features from different Gaussian scale spaces and using them together has been proposed as a preprocessing method [114].

Gao et al. achieve 96.8% accuracy using CNNs and augmenting the data by rotating it [115, 85]. The proposed architecture is composed of three convolutional layers, a fully-connected layer and a classification layer. Compared to other examples in the literature, this architecture is somewhat shallow [5, 37]. Shallow CNNs composed of few convolutional layers followed by fully-connected layers have already been used to classify handwritten digits [1].

While obtaining the 96.8% accuracy ratio, Gao et al. leave one example out and train with the rest of the images. With this approach, samples from the subject who the test image belongs to are used for training, which causes the performance measurement to be extremely optimistic. Leaving one subject out, rather than leaving one sample out would produce results more comparable to the literature.



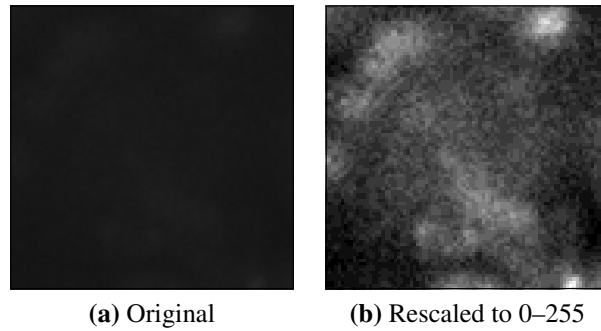
**Figure 3.10:** Examples from different classes of the ICPR 2014 dataset. The first row are “intermediate” and the second row are “positive” examples.

### 3.2.3. Dataset and preprocessing

There have been three challenges held for HEP-2 cell classification: at ICPR 2012 [116], ICIP 2013 and ICPR 2014. The dataset used in the first challenge was composed of 1,457 images (723 training–734 test). For the following challenges, the number of images were increased to 68,429 and two additional classes have been added. 13,596 of the images for the second dataset have been made available for the researchers as the training set. The remaining images were kept secret to be used for testing. For this reason, only the training set could have been used for this study. The dataset also includes a binary mask for each image, which have been obtained by manual segmentation. These masks were not used for preprocessing or classification in our study.

Examples from the six classes that are to be recognized are given in Figure 3.10. The examples on the first row are labeled as intermediate intensity and are not imaged clearly. The examples on the second row are labeled as positive intensity and display the characteristics of the related class clearly. For correct classification of intermediate examples, which constitute the 55% of the dataset, preprocessing needs to be applied.

See Figure 3.11 for an original intermediate intensity image and its rescaled version for better viewing. Although the rescaled image does not look like the positive intensity examples, its details are distinguishable. As the preprocessing method, brightness values of the images are rescaled for zero mean and 1 standard deviation.



**Figure 3.11:** An original example of an intermediate intensity Golgi image and its rescaled version for better viewing.

### 3.2.4. Proposed method

In this study, an ImageNet pretrained GoogLeNet [37] has been used. The recent studies indicate that wider and deeper CNNs achieve higher performance with adequate training data [65]. Rather than extending the traditional denser CNN architecture, GoogLeNet uses the network in network approach [117] to mimic a sparse architecture. With this approach, it has achieved a better performance than AlexNet [5] while having less parameters. Models with less parameters are less likely to overfit, require lower memory and run faster with sequential processors. For these reasons, we found this architecture to be suitable for the experiment. We should note that as we repeated the experiments in this study with other ImageNet pretrained architectures, we observed similar results.

CNNs learn representations and classifiers with the same model. The 1000-class ImageNet classifier at the end of GoogLeNet is of no use to us in this application. CNNs generally classify using a final softmax layer, which attributes a probability score for each of the classes. By removing the final layer, we can gain access the highest level features. In this study, these highest layer features are fed into a multiclass SVM [118]. The remaining layers of the CNN are not fine-tuned.

A problem about using a CNN pretrained with a different task is the difference in the input format. The ImageNet dataset is generally composed of colored images of about VGA resolution. Almost all CNNs designed for recognition scale and crop these images down to  $224 \times 224$ . Our dataset of cell images are composed of grayscale images with 60–90 width and height. For this reason, we upscaled our images to  $224 \times 224$  and repeated their channels. This probably has caused color-sensitive representations to

**Table 3.7:** Confusion matrix for HEp-2 cell classifications.

	Hom.	Speck.	Nuc.	Cent.	NuMem	Golgi
Homogeneous	<b>1873</b>	496	30	1	76	18
Speckled	537	<b>1701</b>	237	296	54	6
Nucleolar	43	148	<b>2167</b>	145	46	49
Centromere	2	328	63	<b>2343</b>	2	3
NuMem	121	99	68	1	<b>1869</b>	50
Golgi	34	12	88	4	183	<b>403</b>

become obsolete.

### 3.2.5. Experimental results

The experiments were done with the training set of the ICPR 2014 HEp-2 cell dataset. Examples were sampled from 83 subjects and the examples from the same subject appear very similar. Having examples from the same subject both in the training and test set makes the classification task unrealistically simple. To avoid this, the experiments were repeated 83 times by leaving one subject out in each iteration. The resulting mean class accuracy is 74.1%. Considering that the CNN architecture was not designed for this task, the CNN was not pretrained for a similar task and the extent of preprocessing applied to the input images, these results are genuinely surprising.

The results are given as a confusion matrix in Figure 3.7. As the Golgi class includes similar staining patterns with Nucleolar and NuMem, it was frequently confused with them, resulting in 55.7% class accuracy.

### 3.2.6. Conclusion

CNNs perform very well at image classification. For applications where there is not enough data to train a CNN, existing well-performing CNNs can be utilized. In this study, it was questioned if this approach is feasible even if the pretraining task and the target task are vastly different. To test this, the highest level features of a CNN pretrained



for ImageNet classification were used to classify HEP-2 cell images. The performance achieved in the experiment is surprising in that it comes close to the performance of methods hand-crafted for this problem.



#### 4. GESTALT TRAINING

Unlike randomly initialized artificial neural networks, visual systems of animals are not blank slates at birth. In a seminal work in perceptual psychology, Tinbergen has shown that herring gull chicks are innately predisposed to specific visual features of their parents [119]. Just moments after these chicks hatch from their eggs, they approach their mother and peck the red spot on its beak (see Figure 4.1). This triggers the mother’s regurgitating reflex, allowing it to feed its hatchling. A similar phenomenon can be observed in humans. Infants less than a day old can recognize faces and distinguish their mothers from others [120]. This ability bootstraps the mother–infant bond, increasing the probability of the mother to take care of the infant.

Having these primitive visual skills at birth increases the survival rate of the species. Therefore, there is a significant evolutionary drive for these skills to be had at birth. Although it is clear why newborns have these skills innately, the actual question is how these skills are gained. Jarrett et al. have shown that even untrained CNNs are quite successful at extracting features for high-level visual tasks (62.9% classification accuracy with the Caltech-101 object recognition dataset) [35]. However, they still had to train a classifier in a supervised manner because recognition is a discriminative task.

We have discussed in Section 2.3 that a visual system needs to be trained in a supervised manner to be able to accomplish discriminative tasks. Therefore, it can be said that animals practically undergo supervised pretraining before being born, despite not getting



**Figure 4.1:** A herring gull chick pecking the red spot on its mother’s beak. The chicks are predisposed to this visual cue innately.

any visual stimulus. Presumably, representations of useful visual primitives are encoded in their genes and utilized in initializing their visual systems. This implies the possibility of initializing a neural network to perform discriminative tasks without using any external data.

In this chapter, we introduce a new approach to deep learning-based vision doing exactly this, which we have named *Gestalt training* due to its relation to Gestalt psychology. The most novel aspect of this method is that it can be used to train deep CNNs—which are typically data-hungry—without using any external data. This is achieved by analytically designing a generative model that can stochastically produce examples of a pattern type. These patterns are special in that they transcend the meaning of their components, and are called *Gestalts*. This transcendence in meaning occurs according to the *grouping principles*, which have been studied by Gestalt psychologists [121].

We are going to start by providing a brief introduction to the Gestalt theory of perception in Section 4.1. Then, we use the proposed Gestalt training method to recognize color constancies (i.e., blobs) in Section 4.2 and edges in Section 4.3.

#### **4.1. Gestalt Theory of Perception**

Humans perceive “the whole” immediately and subconsciously [122]. For example, we understand a shape we see to be a square immediately, without the need to count its edges and check if its angles are right. In fact, it is unavoidable for neurotypical people to perceptually deconstruct a square and see it as four arbitrary line segments.

Gestaltists state that our perception of the whole is formed of a hierarchy of Gestalts. Each Gestalt is a group of lower-level Gestalts. However, Gestalts are indivisible, because on top of being a sum of parts, they have a transcendent meaning [121]. For example, a square is formed of four line segments, yet it is much more significant than an arbitrary formation of four line segments. The more transcendent meaning a Gestalt has, the more unavoidable it is to be perceived.

Although we are going to be discussing low-level perception, Gestalts apply for higher-level perception and abstract concepts as well. For example, when we see thousands of soldiers, we perceive them as an army, rather than a crowd of soldiers. In fact, this the main psychological reason for armies to enforce a very strict uniform code: Gestalts are perceived more strongly when their parts are more uniform. Therefore, the concepts

discussed in this chapter are scalable across levels of abstraction.

Laws governing the perception of lower level Gestalts are considered to be largely innate, meaning that they are not learned [121]. Perceptual grouping is one of these laws, which governs how we group separate objects. The objects that carry similar qualities are perceived to be a part of the same group. In addition, these grouping principles work hierarchically. Below are some examples of these grouping principles [123]:

- Color constancy
- Vicinity
- Similarity
- Closure
- Convexity
- Continuity of direction
- Constant width
- Past experience
- Amodal completion
- Symmetry
- Common motion

Due to the nature of the field of psychology, these definitions are generally made qualitatively. Desolneux et al. quantify these laws using a probabilistic approach, with the aim of developing parameter-free feature extraction methods [124]. The Helmholtz principle is used to do this, which states that large deviations from randomness results in the perception of a structure. Desolneux et al. use the Helmholtz principle to validate detections.

Perception of Gestalts is universal in humans, and thus it is general-purpose and required for all visual tasks. Moreover, the hierarchical structure of Gestalts strongly resembles representations that deep architectures are designed to learn. Therefore, using Gestalt principles in deep learning research looks promising.

#### 4.1.1. Helmholtz principle

Let us introduce the Helmholtz principle as described by Desolneux et al. [124]. As we have mentioned, they have developed this formulation to set validation criteria for stochastic observations. In simpler terms, one can think of this as an analytical method of setting a detection threshold.

The main argument of the Helmholtz principle is that if it is likely for an event to occur in noise spontaneously, it is not meaningful, and thus should not be detected. Here, being likely means having a probability of more than 0.5. Then, the optimal detection threshold is the one that is set exactly at the level where it would only invalidate events that are likely to occur in noise.

To calculate this, we start with the number of false alarms ( $NFA$ ):

$n$ : total number of objects

$k$ : number of objects that have the quality

$p$ : probability of an object having the quality

$N_{conf}$ : number of possible tests

$B(n, k, p)$ : the probability of at least  $k$  objects having the quality

$$NFA(n, k, p) = N_{conf} \cdot B(n, k, p) \quad (4.1)$$

$N_{conf}$  is the number of detection candidates, and  $B(n, k, p)$  is the validation threshold for a group of these candidates. If we have too many candidates, or if it is too common for at least  $k$  objects out of  $n$  to have the quality, we would have a lot of false alarms. The term “having a quality” can also be interpreted as conforming to a particular grouping principle.

Here,  $B(n, k, p)$  can be represented as a binomial sum:

$$B(n, k, p) = \sum_{i=k}^n \binom{n}{i} p^i (1-p)^{n-i} \quad (4.2)$$

Finally, the Helmholtz principle states that an event is  $\epsilon$ -significant if the following is true for random data:

$$NFA(n, k, p) \leq \epsilon \quad (4.3)$$



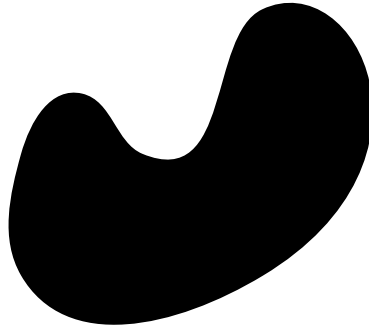
**Figure 4.2:** Different kind of grouping principles are perceived with different  $\epsilon$ -significance, which implies that a universal threshold of 1 is not optimal. For example, the threshold for perceiving faces is very low, causing frequent false alarms.

If this is true for  $\epsilon \leq 1$ , the event is defined to be significant, i.e., the probability of making this detection in noise is below 0.5.

Let us interpret the meaning of this with extreme examples. For an event to be significant, its  $NFA$  has to be less than 1 in a random example.  $NFA$  increases linearly with  $N_{conf}$ , which means that events observed under very large  $N_{conf}$  are not meaningful. For example, if we had bought more than half of the lottery tickets, one of these tickets winning the grand prize would not have surprised us. Similarly, a large  $B(n, k, p)$  causes  $NFA$  to be large, thus makes the observation meaningless. Even if we roll a die only once, the result not being one would not surprise us, as it is more likely for the result to not be one. We are going to provide a detailed use-case in Section 4.2, which is going to clarify how the Helmholtz principle is used quantitatively.

Note that deciding that an event is meaningful using the Helmholtz principle is not equal to the event being meaningful. For example, a friend can claim that he is going to throw a coin five times and always get heads, and proceed to achieve it. Here,  $N_{conf}$  is 1 (he only tried once) and  $B(n, k, p) = 0.5^5$ , so  $NFA = 1 \cdot 0.5^5 \ll 1$ . Although this indicates that we should assume that our friend has cheated, it does not prove it. The Helmholtz principle solely aims to maximize our expected accuracy in a stochastic environment, and should not be depended on to yield absolutely correct results.

Another issue with the Helmholtz principle is that in perception, the cost of a false



**Figure 4.3:** An illustration of the effect of color constancy on perception. We perceive the group of black pixels to be the parts of a single object. This applies even though the object has an arbitrary form.

positive and a false negative is almost never equal. For example, pareidolia is a phenomenon where humans see patterns in random data. These patterns are almost always faces, because evolutionarily, not recognizing a face is much more costly compared to falsely recognizing a face on a piece of wood (see Figure 4.2). Unless we can analytically determine the risks of false positives and false negatives, quantifying *NFA* analytically loses its point, as  $\epsilon$  would have to be determined empirically.

#### **4.2. Recognizing Color Constancies**

Similarity by color constancy is arguably the most basic grouping principle, which is being able to group neighboring pixels of similar colors (see Figure 4.3). Being able to recognize color constancy is closely related with two computer vision tools: blob and edge detectors. Blobs are groups of neighboring pixels with similar colors, and edges are the disruption of the similarity in color. The fact that this grouping principle is both very basic and related to these frequently used tools indicate that recognizing it is a fundamental visual skill.

To embed the maximum amount of information on an image without it being perceived, limits of human perception was studied. It was hypothesized that humans find it harder to perceive watermarks embedded in the busier regions of images [125]. Busyness is defined as being of high frequency, or in our terms, a lack of color constancy. It is proposed that more data can be embedded on image blocks with higher standard deviation. Another approach is avoiding to disturb connectivity [126], and the limits are again calculated using probability.

1/2	1/2	1/2	1/2
1/2	1/2	1/2	1/2
1/2	1/2	1/2	1/2
1/2	1/2	1/2	1/2

**Figure 4.4:** The model of a darker square over a lighter background. In the case that each pixel is either dark or light with equal probability, the individual probabilities of each pixel to conform to this model is  $1/2$ .

The Helmholtz principle is a generalization of these probabilistic approaches for all grouping principles [123]. It states that deviation from randomness results in the perception of structure. In a visual sense, randomness is defined to be white noise with uniform distribution. Gestalts are structures that are not likely to have happened in this noise by chance [127]. While the noise is predefined, we have to model the respective Gestalt.

We first calculate the expected number of spontaneous occurrences of a Gestalt in noise. The Helmholtz principle states that if this number is more than one, this Gestalt is not perceivable [123]. In contrast, a very small number of false alarms indicates that a Gestalt is easy to perceive. By using the Helmholtz principle, we can generate examples of a Gestalt that can be perceived with arbitrary difficulty. Moreover, we can threshold the difficulty level, as trying to learn an impossible task will be counterproductive.

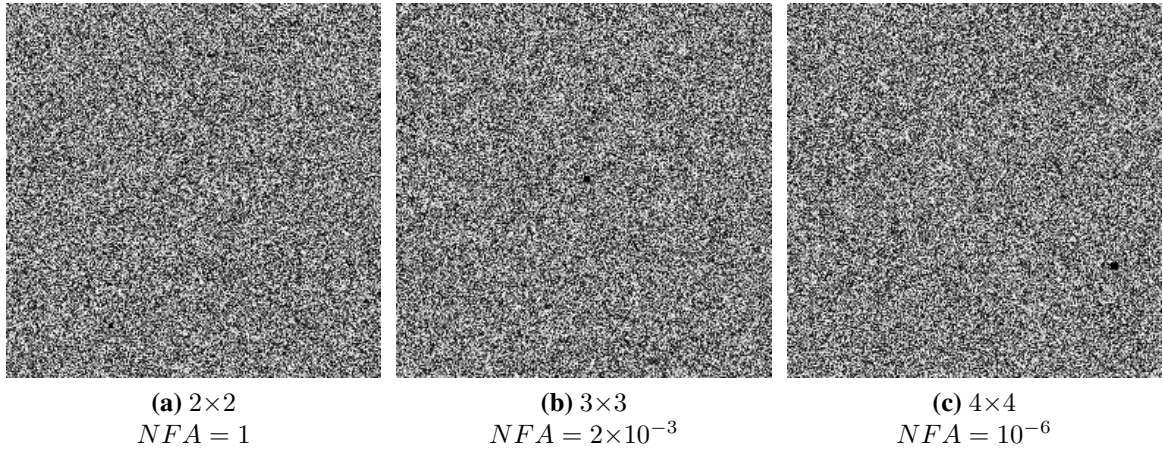
#### 4.2.1. Modeling color constancy

In this section, we are going to attempt to model color constancy, which is a difficult task even for this most basic grouping principle.

##### 4.2.1.1. *Initial model*

Let us introduce the task of detecting black  $2 \times 2$ ,  $3 \times 3$  and  $4 \times 4$  squares in a  $256 \times 256$  uniform white noise image. The pixels are assumed to be either light or dark, with equal probability (see Figure 4.4). The calculation shows that  $2 \times 2$  squares are barely perceiv-





**Figure 4.5:**  $2 \times 2$ ,  $3 \times 3$  and  $4 \times 4$  black squares in uniform white noise images of size  $256 \times 256$ . The Gestalts with smaller  $NFA$  are more perceivable as expected.

able, with an  $NFA$  very close to 1:

$$N_{conf} = 253 \times 253 = 64009 \quad (4.4)$$

$$B = 2^{-16} \quad (4.5)$$

$$NFA = N_{conf} \cdot B = 0.98 \quad (4.6)$$

See Figure 4.5 for examples of differently sized squares and the  $NFA$  values calculated according to our model. An  $NFA$  of 1 means the square should be borderline imperceivable, and be confused with other similar patterns in the noise, which is consistent with the figure.  $NFA$  decreases sharply with increasing square size, which corresponds to easily perceivable squares. While the result of the model is as expected in this particular case, the model is not very sound. First, it assumes that there are only two gray levels, which is too coarse. In addition, it is not appropriate to model larger blobs as perfect squares, and this approach underestimates  $N_{conf}$ , and thus  $NFA$ . This model results in accurate calculations only because the squares being considered are small enough.

#### 4.2.1.2. *Difficulties with an exact model*

A grayscale image with 8-bit depth has 256 levels of gray. If we update our calculations according to this:

$$B = 256^{-16} \approx 0 \quad (4.7)$$

Following from this,  $NFA$  will also be approximately zero for all square sizes, which is clearly incorrect. The error in this reasoning is the assumption that all gray levels are sufficiently different. We can perceive the difference between the colors represented by 255 and 254 when presented side by side, but we would name them both as “white”. This aspect of our perception can be approximated as being able to distinguish between a finite number of scales of gray. Say we can distinguish between 8 levels, and recalculate:

$$B = 8^{-16} \quad (4.8)$$

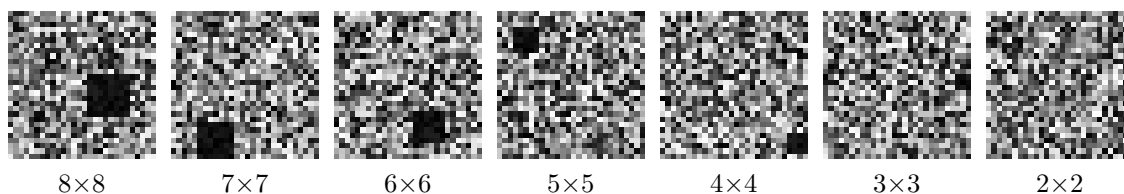
Accordingly,  $NFA$  will be far lower than what we expect. The reason for this is considering only the perfect square shapes when calculating  $N_{conf}$ . Just as colors, we perceive many similar shapes to be roughly the same. Then, we should also consider the square-like shapes when calculating  $N_{conf}$ .

Even when modeling this simple Gestalt, we encountered two very difficult questions:

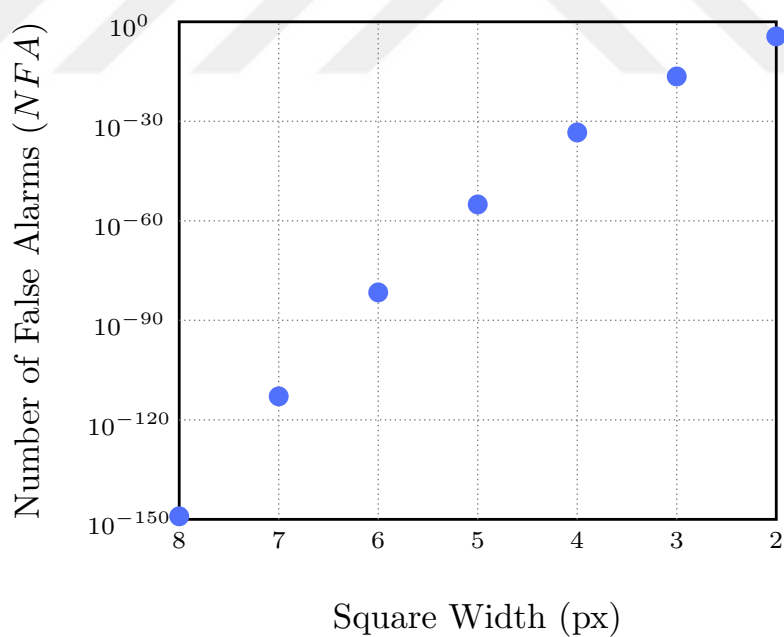
1. How many scales of gray do we see?
2. Which shapes do we perceive as a square?

Fortunately, these questions only have to be answered if we need an exact model. Instead, we can be content with the knowledge that perceiving this Gestalt will be more difficult with smaller square sizes and less constant colors. This knowledge is enough to generate tasks with different difficulties.

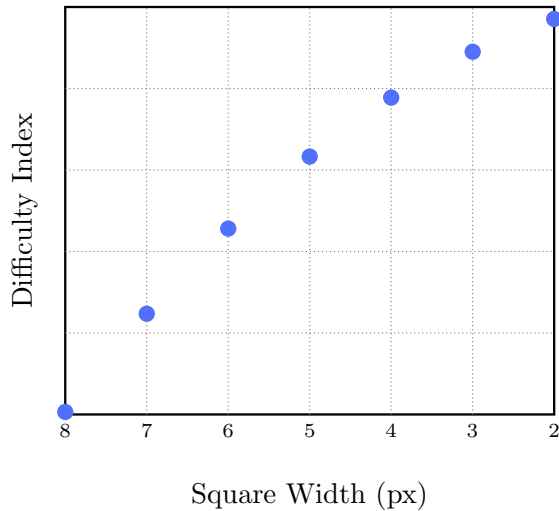
We have mentioned that we do not want to train a network for an impossible task. For this Gestalt, we can easily determine the threshold where the task becomes impossible manually, as seen in Figure 4.6. Even without knowing the exact model, we can see that for  $2 \times 2$  squares,  $NFA$  is larger than 1.



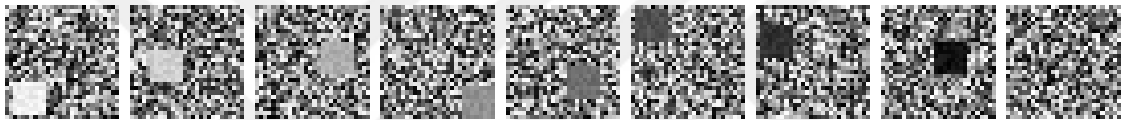
**Figure 4.6:** Dark squares with different sizes in  $28 \times 28$  uniform white noise images. It is not possible to distinguish squares of size  $2 \times 2$ .



**Figure 4.7:** Number of false alarms ( $NFA$ ) for the square widths in Figure 4.6. Note that  $NFA$  is nearly 1 for squares with size  $2 \times 2$ , which indicates that they would be borderline impossible to perceive.



**Figure 4.8:** Instead of using the absolute  $NFA$  values as in Figure 4.7, we can use them to create a difficulty index.



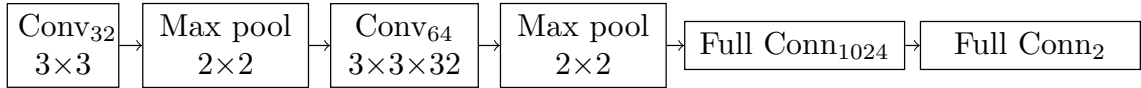
**Figure 4.9:**  $8 \times 8$  squares of different shades of gray, and a negative example.

Let us validate our observations with quantitative calculations. See Figure 4.7 for the  $NFA$  values for each square size in Figure 4.6. All square sizes larger than  $2 \times 2$  have an  $NFA$  much lesser than 1, which means that they should be easily perceivable. However,  $2 \times 2$  squares have an  $NFA$  of almost 1, which means they should be borderline impossible to perceive. Indeed, we can see that the  $2 \times 2$  square in Figure 4.6 has blended in with the noise.

We have mentioned that our model is not exactly accurate, and optimal visual systems set detection thresholds based on false positive/negative risks, which are difficult to determine analytically. Therefore, the  $NFA$  values we have calculated for Figure 4.7 are not very meaningful. For that reason, we recommend to use them as relative difficulty indices as shown in Figure 4.8.

#### 4.2.1.3. *Designing the task*

We have approached the problem as recognizing the existence of color constancy in an image. The input images may or may not contain a square with constant color (see Figure 4.9). The task is to decide if it does, which is a form of binary classification. This



**Figure 4.10:** The CNN architecture used in the experiments. The subscripts indicate the number of neurons at each layer, and the following numbers indicate kernel sizes.

may not be ideal, because it is shown that deep learners perform better when pre-trained with stronger supervision [128]. In this context, finding the location of the square would have been a stronger form of supervision than assigning a binary label. However, our experiments have shown that CNNs are able to learn to do this classification easily, thus we opted for classification for implementation convenience.

We created a dataset for each square size shown in Figure 4.6. Each individual dataset contains 10,000 positive examples of each shade of gray shown in Figure 4.9, and 80,000 negative examples (i.e., the positive–negative ratio is 1).

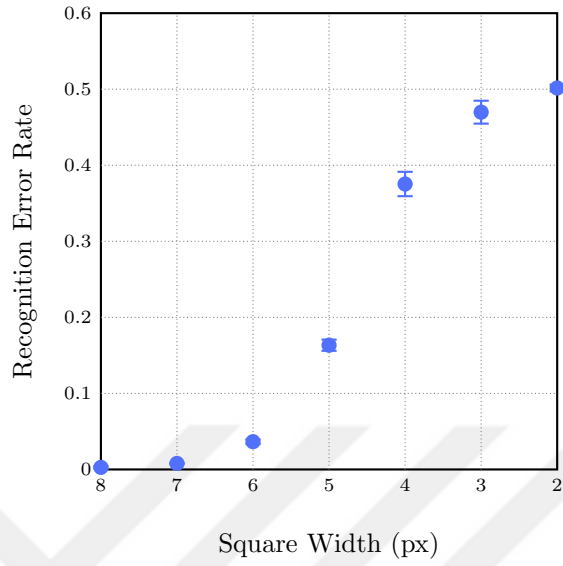
#### 4.2.1.4. *Training with the generated task*

We used a CNN architecture with two convolutional layers and two fully-connected layers (see Figure 4.10). Adam optimizer [129] is used with a training step of  $10^{-5}$  and a batch size of 128. The training is continued for up to 50 epochs. At the end of each epoch, validation error is measured, and training is stopped if 5 consecutive epochs was not able to improve upon the validation error. Then, the network state that has yielded the lowest validation error is used to measure the test error.

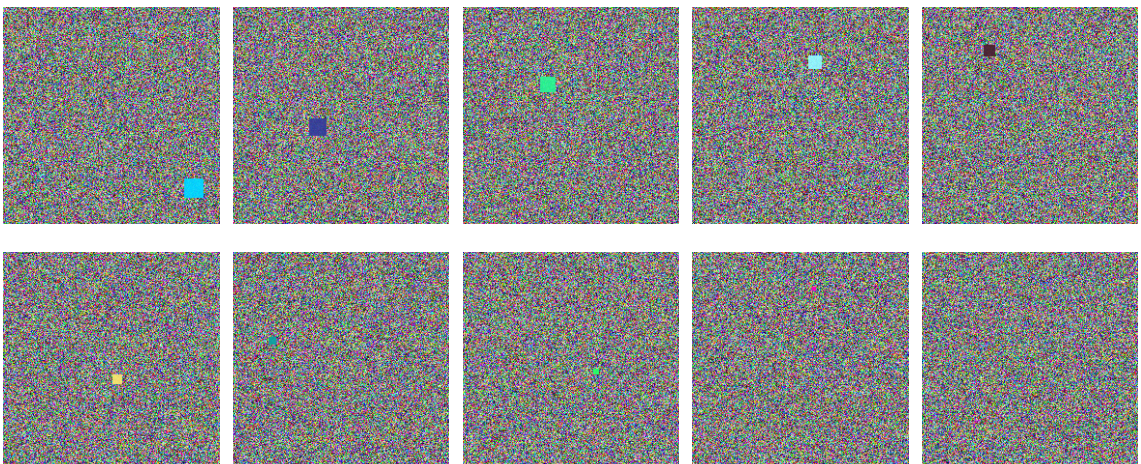
Let us discuss the results of the training, as shown in Figure 4.11. The error rate is a logistic function of the square sizes. Larger squares can be recognized easily, followed by a sharp increase in error with smaller sizes. We have calculated *NFA* for differently sized squares. *NFA*, which quantifies perception difficulty, was increasing with smaller squares. The fact that our metric of difficulty (Figure 4.8) has a direct relation with the error rates is encouraging. The error rate is 0.5 at width 2, which means that the CNN was not able to learn anything useful.

#### 4.2.1.5. *Training ResNet-50 to recognize colored constancies*

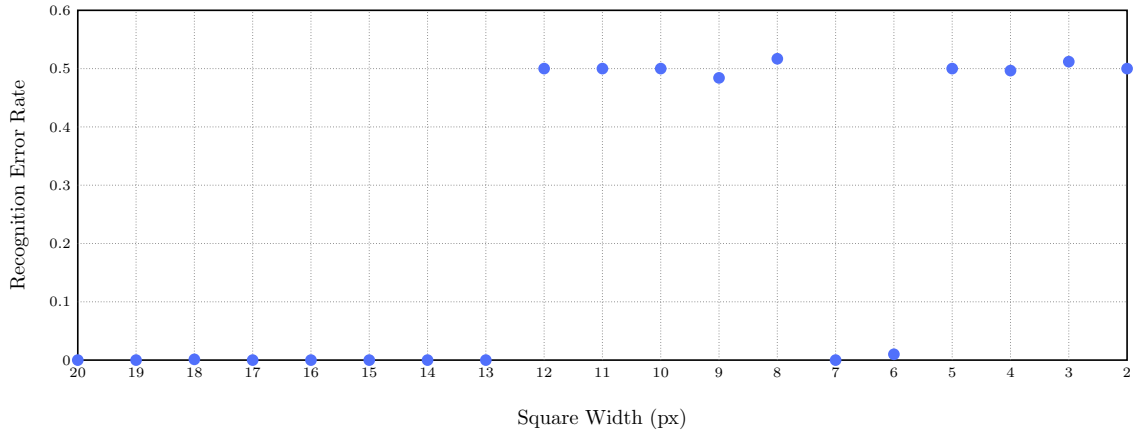
CNNs designed for ImageNet object classification take  $224 \times 224$  colored images as inputs [58, 37, 20]. For compatibility, we generated pretraining images in the same format



**Figure 4.11:** Test errors with varying square sizes. The experiments are repeated 5 times. Means are shown with circular markers and  $\pm 1$  standard deviations are shown with error bars.



**Figure 4.12:**  $224 \times 224$  colored color constancy examples.



**Figure 4.13:** Color constancy recognition error rates with ResNet-50 for the examples similar to the ones in Figure 4.12.

(see Figure 4.12). For the previous image format, we have calculated the minimum square size that can be perceived to be  $3 \times 3$ . This was validated both by observing the examples ourselves, and seeing our CNN fail for smaller squares. With the new image format, we may expect the minimum perceivable square size to increase, because the image size has increased [123]. However, since there are three channels, two neighboring pixels being of a similar color is less likely. These two factors roughly negate each other, and the minimum perceivable square size is again close to  $3 \times 3$ .

Similar to the previous pretraining experiments, we used the newly generated images to pretrain a randomly initialized network. However, we used a much more deeper network, ResNet-50 [34]. The test performance can be seen in Figure 4.13. The first thing to note is that despite the model’s size, it did not overfit. Even for squares of  $6 \times 6$  size, it can recognize color constancy almost perfectly.

In the previous experiment, the recognition error rate had a logistic function shape with a smooth transition. This time, we observe the change to be abrupt, even discontinuous. This is probably because the transition from 0 and 0.5 is squeezed between two square sizes.

These experiments are run only once to illustrate an interesting phenomenon. The network sometimes fails to converge, and this becomes more likely with smaller square sizes. We can overcome this by restarting the pretraining until convergence. We have observed a similar instability for the self-supervised training method proposed in Chapter 5. It was solved by changing the training task from binary classification to binary ranking.

We can expect this approach to improve the robustness in this condition as well.

### **4.3. Deep Gradient Operator<sup>1</sup>**

Despite the great interest on deep learning for high-level computer vision problems, low-level image processing tasks had been left to classical methods, such as directional gradient filtering, Gaussian blur, etc. The primary reason for this is that it is difficult to do manual labeling for low-level tasks (e.g., edge detection) because of the excessive workload and the ambiguity of the correct labeling. The large amount of labeled data required by supervised methods can be provided by developing a methodology that produces automatically labeled data. For this study, we have designed a probabilistic generative model that produces densely labeled synthetic images containing objects with edges at various noise levels. A convolutional neural network trained with the data produced by this model in a supervised manner learns to detect edges in real images as well, without regard to its semantics. Close inspection of the convolutional layers shows that they do not specialize on classical edge detection stages such as noise removal and directional gradient operators. Instead, the filters learned by the network are a mixture of these operations, which motivates further research about the biological mechanisms of the human visual system.

#### **4.3.1. Introduction**

The field of image processing focuses on problems such as denoising [131], edge detection [132] and image compression [133]. On the other hand, computer vision applications aim to solve higher-level problems such as recognizing the objects in an image [5] or verbally describing a scene [134]. Yet, the common objective of both fields is to recover the entangled factors of variation in the image [2].

The regions of an image that show higher variation contain more information compared to the more uniform regions. These variations are attempted to be recognized by modeling them as edges [132], corners [135], lines [136] and keypoints [57]. These types of gradient-based features can be used in various applications such as fiducial marker detection [137] and camera calibration [88].

The abrupt changes in illumination, reflectance and physical structures are projected

---

<sup>1</sup>This section is adapted from [130].

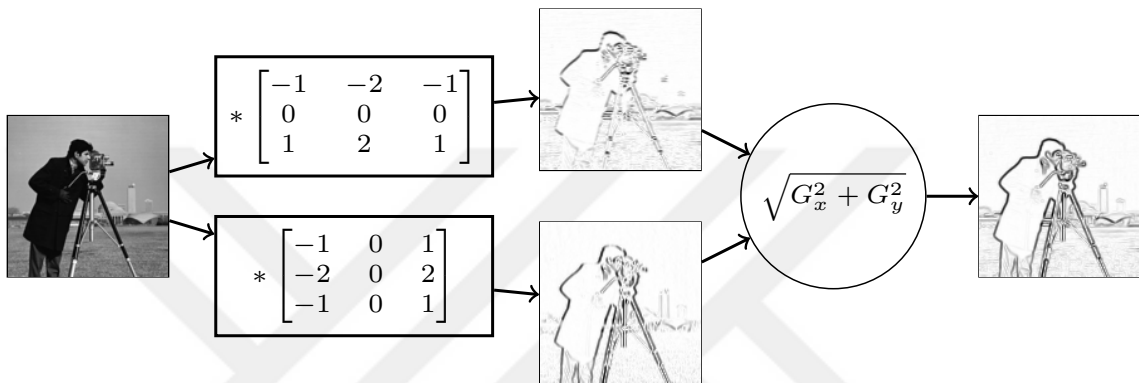


on the image as strong color gradients [138]. Since these color gradients, which are referred to as edges, contain meaningful information about the scene, there have been many studies on how to detect them [132, 139, 17]. In addition, it has been shown many times that when a deep convolutional neural network (CNN) is trained for a high-level task such as object recognition, their earlier layers learn to respond strongly to directed edges [58, 86]. This implies that edge detection is one of the building blocks of computer vision methods, including deep learning-based ones.

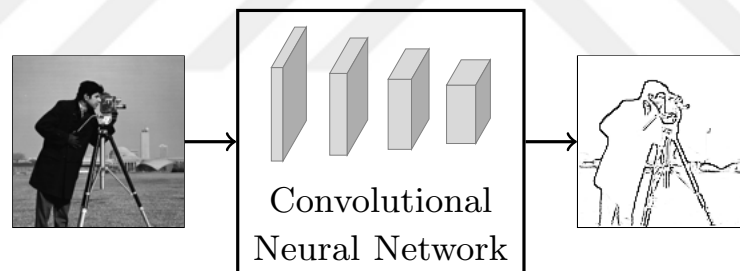
The term “edge” may be used differently in various sources in the literature. While lower-level image processing studies define all kinds of distinct color gradients as edges [132], higher-level computer vision studies do not regard color gradients caused by changes in illumination (e.g., shadows) as edges [140]. For the image segmentation problem, which is commonly studied in the context of deep learning, only the contours of entities with a semantic meaning are aimed to be detected [141, 142]. While using a deep learning-based method in this study, we focused on the low-level problem, where pixel gradients are defined as edges.

The low-level problem, where all abrupt color changes in the image are defined as edges, can be solved with similarly low-level approaches. The most common of these is convolving the image with analytically designed gradient operators (see Figure 4.14a). The image is convolved with two orthogonally-directed Sobel operators, corresponding to depiction of horizontal and vertical gradients. The resulting two response maps form size-2 (gradient) vectors, which are then fused by evaluating the Euclidian magnitude at each location. We propose to use a CNN with identical input and output sizes to infer the intensity variations around the location of each pixel (see Figure 4.14b). We call this CNN, the *deep gradient operator*, as it produces its output as a result of a series of convolution operations, as opposed to a single one. An interesting difference between the proposed and the traditional gradient methods is that in the traditional method, the gradient responses from the two analytically designed operators are fused with another analytically designed method (i.e., the Euclidian norm), while the deep gradient operator learns the fusing operation along with the gradient filters. Furthermore, the operator does not explicitly apply a Gaussian blur for noise elimination. Instead, it embeds the necessary noise elimination process into the layers by learning from labeled data.

The need for a large amount of representative and accurately labeled data is the



(a) Gradient detection with the Sobel operator.



(b) Gradient detection with the deep gradient operator.

**Figure 4.14:** Gradient detection with the Sobel and deep gradient operators. (a) The image is convolved with two Sobel operators, which are analytically designed to detect horizontal and vertical gradients. The Euclidian norm of these two response maps gives the gradient map. (b) The gradient map is inferred with a convolutional neural network in an end-to-end manner. Darker colors in the image indicate higher gradient responses.

main obstacle in the way of applying deep learning to most problems. A good work-around for this problem is to use a probabilistic generative model to produce potentially infinite number of labeled synthetic data. Using such a model is equivalent to Bayesian model-based reasoning [143], which requires extensive domain knowledge. Training the network with the data generated by such a model injects the related domain knowledge to the network.

In the literature, synthetic data generation was proposed using renders of three-dimensional models [79, 144]. Similarly, renders of text have been used to generate data for text recognition [145, 146, 143], where the fonts could be regarded as the source data for synthetic data generation. In addition, augmenting the training data with minor transformations is a common trick utilized to produce state of the art results in deep learning-based computer vision applications [5].

Although probabilistic models are frequently used to derive training data from existing data, the same is not true for using them to generate training data from scratch. To be able to design a data-independent and fully generative model, one needs to analytically determine the characteristics of the data to be generated [143]. Determining these characteristics would also allow us to develop rule-based recognition methods. For the same reason as we cannot develop a good rule-based method to recognize dogs, we cannot generate convincing dog images without depending on data. Conversely, these approaches are both feasible for lower-level problems.

In this study, we start by designing a probabilistic model to generate synthetic edge images to train the proposed deep gradient operator. This model is designed analytically by determining the characteristics of edges based on domain knowledge. Since the images are synthetically generated, their edge locations can be automatically determined. This information is used to label the images densely, so that each pixel has an individual label. The task of recognizing the edges in the labeled images is made more difficult by employing additive noise. The deep gradient operator trained with this data is observed to be capable of successfully detecting gradients in synthetic, as well as real-life images. The results are promising for injecting universal knowledge about vision to deep neural networks to reduce their data-dependence. In addition, the analysis of the trained model can provide insight on developing better rule-based methods for the vision problem being considered. Finally, close inspection of the network layers through visualization provides

insight on how biological vision systems may be seeing edges.

The rest of the paper is structured as follows: In Section 4.3.2, we describe how synthetic edge images with labels are generated. The deep gradient operator architecture used in the study is introduced in Section 4.3.3. The training methodology and the results are given in Section 4.3.4. Finally, the paper concludes with comments about the results in Section 4.3.5, along with a discussion of potential future work.

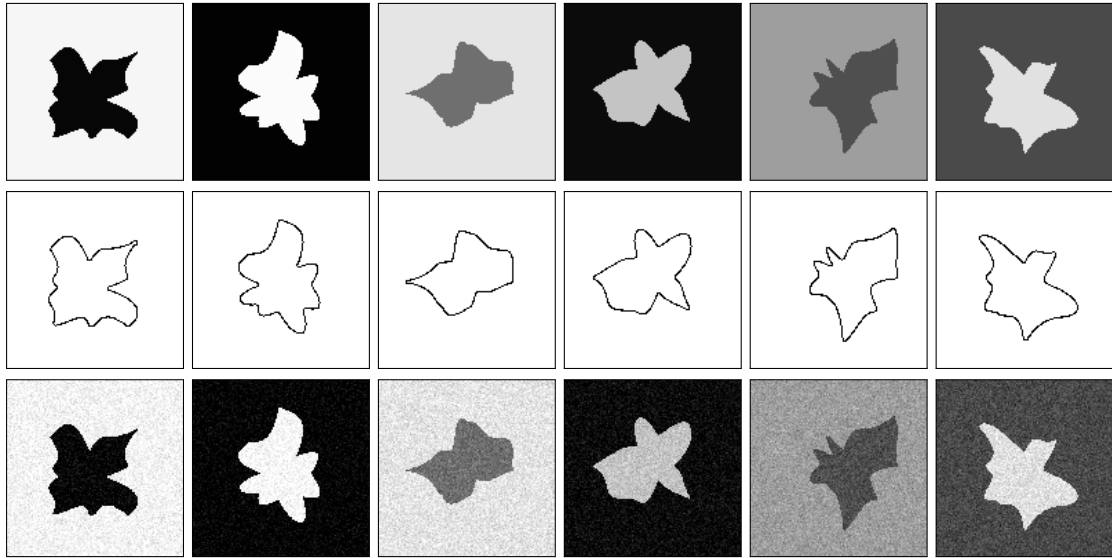
### **4.3.2. Generating edge examples**

The proposed method does not require existing training data as most machine learning-based methods do. As we have stated earlier, we are not going to use an existing dataset either directly, or to derive new samples from. Instead, the training data is going to be generated stochastically from scratch, utilizing a model designed based on analytical observations.

Considering the problem of edge detection, there is a certain duality between color constancy and edges; for an image to have one, it must also have the other. Just as image processing and computer vision applications are heavily concerned with edges, color constancy is regarded as one of the fundamental grouping principles in gestalt psychology [123]. Although rare, computer vision methods that utilize color constancy to extract features were also proposed [147].

To train the proposed method, we need to generate images with color gradients. If we approach this problem from reverse, we can see it as generating images with distinct color constancies. We can generate a closed shape with a constant color on a background with another constant color to satisfy this requirement. While generating training data stochastically, one needs to ensure that the generated outputs are adequately diverse. If the training examples are composed of similar shapes, the trained model is not going to be successful with real images.

To generate the shapes to be used in the proposed method, we uniformly sampled 25 points from a circle at the center of the image. We disturbed the locations of these samples towards random directions and fitted a Bézier curve [148] to them. We, then, filled in the curves by choosing two random colors for the foreground and background. See the first row in Figure 4.15 for examples. We also drew only the contours of these shapes, which are going to act as a dense labeling of pixels to be used for supervised training (see the



**Figure 4.15:** The first row are stochastically generated shapes. The second row are the contour images of the shapes, which are used as pixel labels. The third row are the training data obtained by adding Gaussian noise to the stochastically generated image.

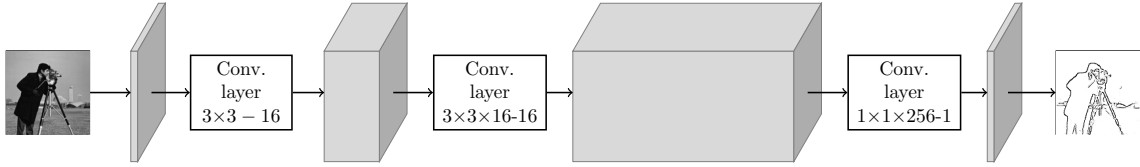
second row in Figure 4.15).

Since detecting the gradients in the original generated images is very easy, a model trained for this task would not perform robustly with real images. Therefore, we need to apply a transformation to make detecting these gradients more difficult. Although we can disturb the edges, we can also approach the problem from reverse again and disturb the color constancies. To follow the second approach, we further applied additive Gaussian noise on the images with 0 mean and 16 standard deviation. This halves the PSNR (peak signal-to-noise ratio) from 48 dB to 24 dB. See the third row in Figure 4.15 for sample outputs.

Since we have generated the training data (Figure 4.15, third row) and the labels of each pixel in the training data (Figure 4.15, second row), we could train a model in a supervised manner to detect these.

### 4.3.3. Deep gradient operator architecture

The architecture we have designed takes grayscale images with a size of  $128 \times 128$  as inputs (see Figure 4.16). In the first layer, we have defined 16 convolution kernels with  $3 \times 3$  sizes. On top of this, we added another convolutional layer with 16 convolution kernels of size  $3 \times 3 \times 16$ . Since these kernels are a lot more than the traditional two Sobel kernels, we would expect them to learn to respond to diagonal gradients, along with vertical and



**Figure 4.16:** A convolutional neural network with three layers, and equal input and output sizes has been trained to recognize pixels with high gradients. Darker colors in the image indicate higher gradient responses.

horizontal gradients. In addition, these kernels should also learn to apply some type of blurring to reduce the effect of the added noise.

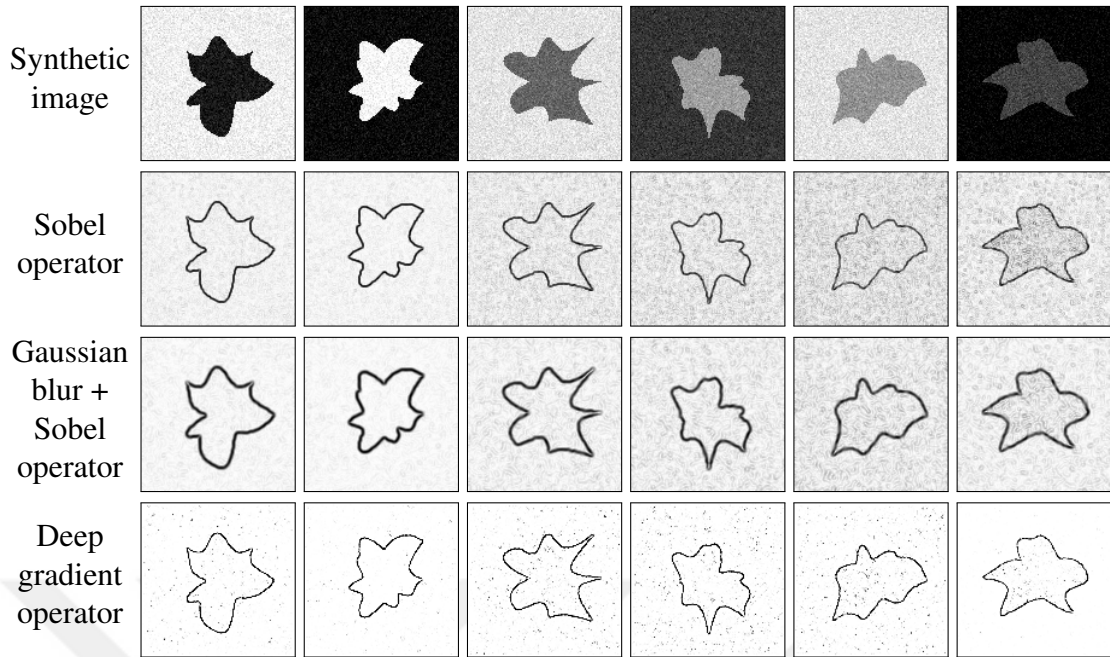
The aim of the third and final layer is to fuse the response map of size  $128 \times 128 \times 256$  propagated from the earlier layers to obtain a gradient response map. For it to return a single output per-pixel, a single convolution kernel of size  $1 \times 1 \times 256$  is defined in this layer. As a result of using two consecutive convolutional layers with a spatial span of  $3 \times 3$ , each pixel output of the proposed architecture can draw information from a  $5 \times 5$  window centered on it. This architecture is comparable with the traditional method of detecting gradients (e.g., in the Canny edge detector [132]) by applying Gaussian blur, followed by applying two  $3 \times 3$  Sobel operators and fusing their responses.

As the activation function, ReLU is used for the first two layers, and sigmoid is used for the final one. As such, a value between 0–1 is inferred for each pixel. When the model is trained for independent binary classification for each pixel, it can be said that these outputs indicate the gradient magnitude at each pixel.

#### 4.3.4. Experimental results

When the training data is generated stochastically, there is no need to reuse training data [143]. Since there would be no overfitting when the training data is not reused, the proposed method does not benefit from regularization methods such as early stopping and weight decay. In this study, we trained the model with 9,000 minibatches of size 64, which means the model has seen 576,000 unique examples. Training is stopped not because of overfitting, but rather because the loss has saturated. Adam optimizer [129] with 0.001 learning rate is used for training.

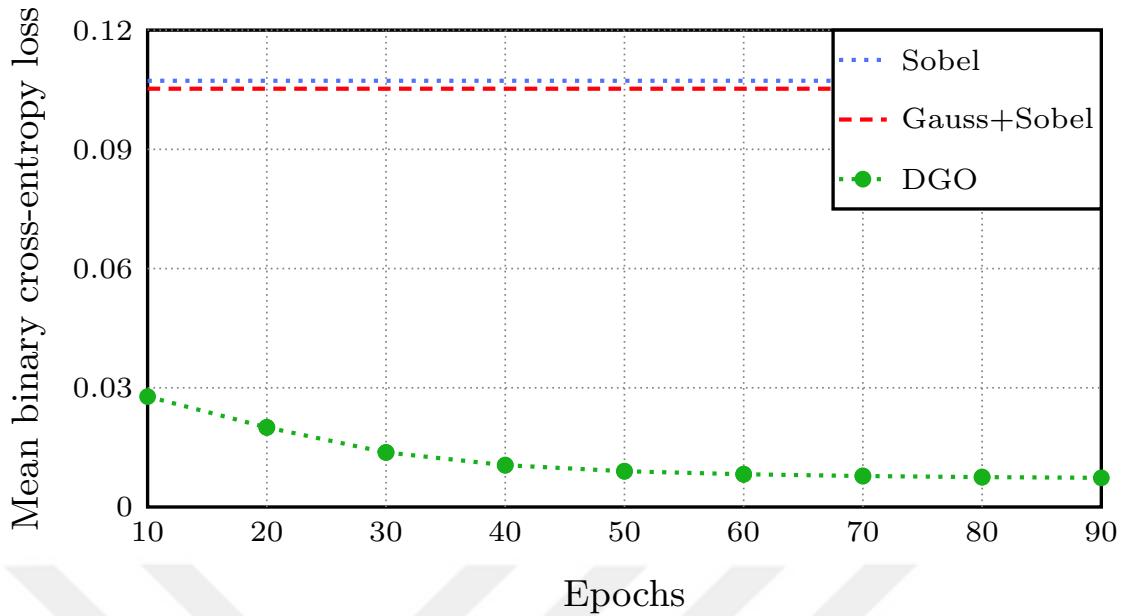
See Figure 4.17 for the detected gradients in stochastically generated images. We presented Sobel results with Gaussian smoothing as well ( $5 \times 5$  window with  $\sigma = \sqrt{2}$ ), as



**Figure 4.17:** The first row are the training data obtained by adding Gaussian noise to the stochastically generated image. The second row are the results of the Sobel operator. The third row are the results of the Sobel operator after Gaussian blur is applied. The fourth row are the results of the proposed method. Darker colors in the image indicate higher gradient responses.

that is a commonly used preprocessing step [132]. We can see that the Sobel operator by itself responds strongly to noise, and even when the image is blurred beforehand, a non-zero gradient is detected consistently on the background and foreground. On the other hand, the proposed method returns zero responses (i.e. gradients) on color constancies with very sparse speckles. In addition, it localizes the contours of the shape precisely in a thin, yet continuous manner.

We compare these methods quantitatively on 1000 synthetic images that were not used in training. The problem can be treated as a binary classification problem with binary cross-entropy loss being the evaluation metric. See Figure 4.18 for the performances of "Sobel operator by itself", "Gaussian blur followed by Sobel operator" and the proposed "deep gradient operator". The deep gradient operator outperforms the other two by a large margin even at the beginning of its training, and improves its performance further as it sees more examples. In this graph (Figure 4.18), we can clearly observe that the proposed method does not overfit, which is rather interesting, considering that no regularization was used. Yet the result can be considered as "expected", because the distribution of the training and test data are the same, and we never re-use any training data.



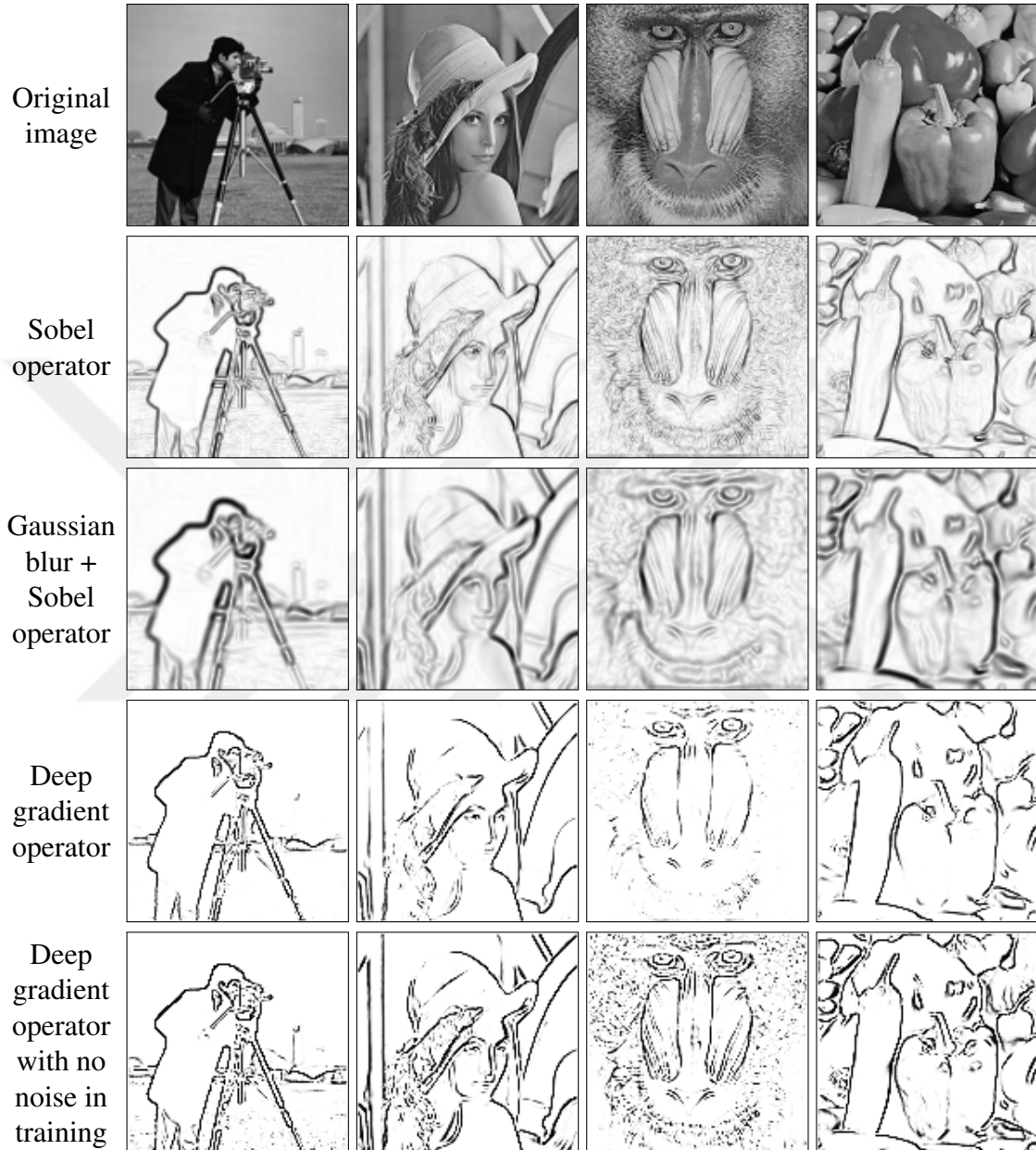
**Figure 4.18:** Mean binary cross-entropy losses calculated with 1000 synthetic images. Each epoch corresponds to training with 6400 unique examples.

See Figure 4.19 for results on real images. The Sobel operator returns a blurred output, while responding to noise-like gradients (e.g., grass behind the cameraman, feathers on Lena’s hat, baboon’s fur). Gaussian smoothing somewhat reduces its response to noise, yet it also results in a degradation in sharpness. The deep gradient operator provides a very crisp gradient detection, while responding to noise-like patterns minimally.

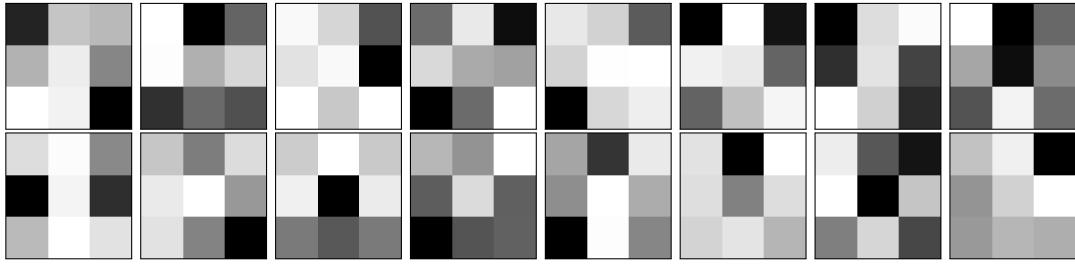
As a follow-up experiment, we omitted the additive noise while training to achieve a more responsive version of the proposed method. In the last row, we can see that without additive noise, the proposed method detects all kinds of gradients in a very sharp way. Both version can be preferred in different applications. Overall, we argue that the results provided by the proposed methods are visibly and quantitatively (cross-entropy low-wise) better; hence are expected to provide more useful features for further processing, including edge detection.

In order to present the internal characteristics of the network, we visualize what the deep gradient operator has learned. In Figure 4.20, 16 kernels from the first layer of the deep gradient operator are presented. Unlike traditional gradient approximations (such as Sobel operator), it is difficult to determine the isolated role of each these filters. However, their structures for responding to edges at different orientations are somewhat eminent, since the locations of highest and lowest weights are different for each filter. Furthermore,

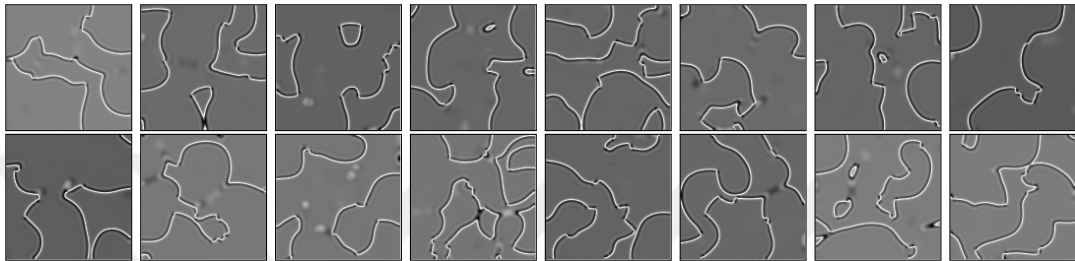




**Figure 4.19:** The first row is the original image. The second row is the output of the Sobel operator. The third row are the results of the Sobel operator after Gaussian blur is applied. The fourth row is the output of the proposed method. The fifth row is the output of the proposed method when the training task is not made more difficult through additive noise. Darker colors in the image indicate higher gradient responses.



**Figure 4.20:** The 16 different  $3 \times 3$  kernels learned by the first layer of the deep gradient operator. The weights are linearly scaled so that the lowest value is shown in black and highest value is shown in white.



**Figure 4.21:** Activation maximizations of the network. A noise image is optimized for maximum activation at the output, and the gradients for this operation are shown. The gradients are higher on the contours (shown in white).

we can also see the blending of smoothing and gradient detection operations, as the filters themselves are not very sharp.

Finally, we examine activation maximizations for the output of the network. See Figure 4.21 for gradients that maximize the response of the network to a noise image. Although the network can see around a restricted window ( $5 \times 5$ ), we notice that it greatly favors the continuity of contours. Therefore, we can deduce that the proposed method implicitly links non-connected pixels by filling the gaps. This can be confirmed from real-life images, where no gaps naturally exist between detected gradients. Furthermore, we can see that the shapes of contours we have used for training is reflected in these activation maximizations, and the model does not favor linearities at all.

#### 4.3.5. Conclusion

Training neural networks require a large amount of data. Yet, some tasks require dense supervision (e.g., pixel-level labeling), which does not lend itself well to manual labeling. In this study, we propose to use a probabilistic generative model to produce synthetic edge images with dense labels. This was possible because edges are low-level features, whose

characteristics can be determined analytically. Then, training a model with the synthetic images is equivalent to injecting the knowledge about these characteristic to the model. As a result, the necessity of feeding CNNs with large amount of labeled data could be fulfilled.

In addition to the rule-based probabilistic generative model, we propose a particular CNN architecture to be used for gradient detection, called the *deep gradient operator*. This architecture takes an image as an input, and outputs an image with the same size, with the aim of emphasizing the edge gradient magnitudes at each output pixel. This CNN is trained using the synthetic data produced by the generative model to estimate the gradientness of each pixel. The learning task is perturbed by adding noise to the synthetic images for the aim of robustness improvement in real-life images.

Although some methods such as traditional gradient operators work well on a stand-alone basis, implementing them as neural networks allows them to be integrated to neural network-based methods that can further be trained in an end-to-end manner. For example, the proposed method can be finetuned with real data for context-aware gradient detection. In that aspect, the proposed work is expected to provide a case study for pretraining. Furthermore, the deep gradient operator can be used as a building block for methods aimed to solve more complex tasks such as edge and edge segment detection. In this respect, it can be argued that the proposed method is more flexible and open to improvement compared to the traditional low-level image processing methods. Since the results in artificial and real-life images are fairly plausible, we conclude that the injection of extensive visual knowledge through synthetic data really reduces the data-dependency of deep neural networks. The fine results also inspire and motivate for further research regarding how the biological human visual system works for the task of edge detection, which is clearly not a combination of two isolated processes of “noise removal” and “gradient magnitude evaluation”.

## 5. SELF-SUPERVISED PRETRAINING

Although supervised pretraining delivers state of the art results in most cases, it has its disadvantages. The most obvious of these is that it requires the pretraining data to be labeled. Although the ImageNet dataset [60] has provided a large amount of labeled training data to propel deep learning research forward, it is an undeniable truth that the majority of data in the wild is going to be unlabeled. This dependence on labeled data becomes a bigger problem when the target task is not similar to the limited labeled dataset options. For more exotic tasks such as cell classification and power line recognition, it is obvious that labeling data is going to be much more costly than labeling natural images. Another problem of supervised pretraining is that it causes the model to gain an unnecessary specialization related to the pretraining task. For example, ImageNet pretraining goes beyond familiarizing the model with likely data distributions, and teaches the model to distinguish between breeds of dogs.

Self-supervised pretraining is an alternative to supervised pretraining. Self-supervision is generating a task with a known solution from the input data without requiring any labels, and using this solution as supervision. It is advantageous in that since the training part is supervised, models and tricks that are designed for supervised training can be used, even if with some changes. In addition, similar to unsupervised pretraining, it does not need labeled data, which makes it attractive for target tasks that cannot be associated with an appropriate labeled pretraining dataset.

In this chapter, we propose a self-supervised pretraining method based on spatial context. As we have described earlier, CNNs learn to model the correlation between nearby regions. Therefore, disturbing the input data spatially degrades the patterns within. The task we have designed for self-supervision is disturbing an image twice with different intensities, and ranking them based on coherence. This task is similar to other self-supervision tasks where a model is taught to solve a jigsaw puzzle. The advantage of the proposed method is that the pretrained model sees the entire spatially disturbed image, rather than a single of a jigsaw piece at a time. As a result, any off-the-shelf model optimized for performance can be used, and higher level representations can be taught. We showcase the benefits of the proposed method with experiments.

## 5.1. Self-supervised Pretraining by Ranking Spatial Coherency<sup>1</sup>

In self-supervised pretraining, unlabeled data is used to generate a problem with a known solution, to be used for training. Since self-supervised pretraining is essentially supervised, it is more compatible with the supervised fine-tuning to follow, compared to unsupervised methods. In this paper, we propose a self-supervised pretraining method based on the prior that the patterns are composed of local interdependencies. The proposed pretraining method consists of spatially disturbing two copies of an image, and ranking them with respect to coherency. This task cannot be fulfilled without recognizing the patterns in the image, thus training for it results in learning respective representations. Accordingly, fine-tuning after the proposed pretraining results in better performance compared to training from scratch.

### 5.1.1. Introduction

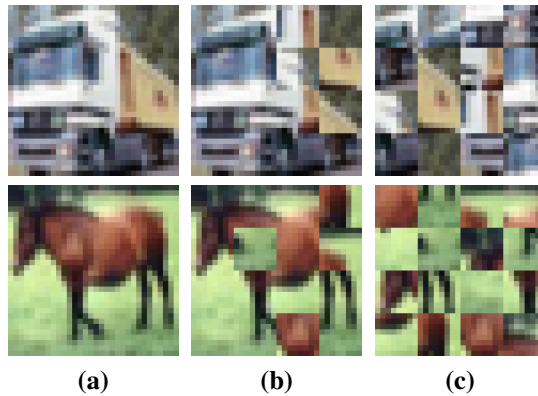
Deep learning methods that have been proposed earlier required models to be pretrained using unsupervised methods, yet this step has long been abandoned [2]. Instead, training a CNN in a supervised fashion has become the gold standard, especially for visual tasks for which a large amount of labeled training data is available [5]. Even in the cases where the labeled data for the target task is limited, supervised training on the ImageNet dataset [6] is a very popular pretraining method [23]. Surprisingly, this applies even if the target domain is vastly different [84, 85, 150].

The majority of the available data is unlabeled, which makes unsupervised approaches appealing. However, challenges such as ILSVRC [6] are dominated by fully-supervised methods [5, 151, 20]. The high performance of supervised methods is attributed to a set of tricks that are not easily generalizable to unsupervised methods [2]. Regarding this issue, self-supervised methods are a good alternative to unsupervised methods for pretraining, as they do not require labels, and are fully compatible with the supervised fine-tuning to follow.

Self-supervised methods use unlabeled real data to generate a problem with a known solution, and train the model in a supervised fashion to reach this solution. It should be noted that this is different than training with synthetic data to inject knowledge to a

---

<sup>1</sup>This section is adapted from [149].



**Figure 5.1:** Images from the CIFAR-10 dataset, divided into  $4 \times 4$  tiles, and permuted to generate degrees of coherency. (a) is the original image, (b) is a mildly incoherent example, and (c) is an intensely incoherent example. The proposed pretraining task is to rank (b) and (c) with respect to coherency.

network [152, 153]. A general self-supervised method should be heavily data-dependent, and not assume task-specific priors. In other words, the designed pretraining task should not aim to inject heuristics that may be useful in the target task, but rather familiarize the model with the data distribution.

Convolutional architectures [1] are used for vision problems based on a prior: Pixels nearby have stronger dependencies among each other, and the dependencies between faraway pixels can be omitted while modeling the data [2]. This implies that CNNs model patterns as interdependencies of local pixels, and thus these patterns can be obscured by disturbing the relative positions of the pixels. Based on this reasoning, we propose a fundamental vision problem that a CNN trained for recognition should be able to solve: Given two copies of a pattern that are spatially disturbed with different intensities, the CNN should be able to tell which one is more similar to the original (i.e., which one is more coherent). See Fig. 5.1 for an example. Even if we had not seen the original image, it would be obvious to us that the mildly incoherent image is more similar to the original than the intensely incoherent image. This is because there are more patterns in the mildly incoherent image that matches the representations that we possess. Therefore, it is reasonable to expect that training for this task is going to be beneficial in learning representations that model the training data.

To generate examples of the proposed problem, an image is spatially disturbed to varying degrees with no regard to its contents. Since this process does not require labels,

the problem is a good candidate to be used for self-supervised pretraining. Moreover, since both the proposed problem and CNNs are designed based on the same prior, we can expect the respective pretraining to be beneficial for all vision solutions where a CNN is applied.

Our main contributions can be listed as follows:

- We propose a novel self-supervised pretraining task, ranking spatially disturbed images with respect to coherency. Earlier studies aim to estimate the spatial disturbance of a single image, which is a niche problem with no obvious solution. Instead, we propose to model the problem as ranking, which is far better studied.
- A metric is proposed to quantify the coherency of an image that is disturbed spatially. This metric is used to form the ground truth regarding the ranking of spatially disturbed image pairs.
- We pose an approach regarding how the spatial disturbance granularity is to be chosen. To transfer high-level representations, the granularity should be coarse. If the target task is from a vastly different domain and only low-level representations are desired to be transferred, the granularity should be fine.

### 5.1.2. Related work

The main issue that a good initialization tries to solve is the vanishing/exploding gradients problem. Xavier initialization [44] and its variant for ReLU activations [34] are commonly used for CNN parameter initialization. These are analytical methods, which do not take the statistics of the data into account. Rather than trying to come up with an optimal parameter distribution, [154] proposes to pretrain the network for low gradient variance. They show that while this initialization method cannot outperform self-supervised pretraining, the two complement each other. This indicates that the benefits of self-supervised pretraining do not overlap with the ones of good initialization methods. Specifically, self-supervised pretraining teaches useful representations, rather than simply initializing the parameters at a favorable location on the objective function surface.

Context emerges from spatial, temporal and cross-channel correlations. A large part of these correlations are local, which allows us to use convolutional layers over fully-connected ones with success [5, 155]. Context-based self-supervised pretraining methods

disturb local correlations by permuting or removing data, and train the model to favor the original data through discriminative or generative tasks. We are going to investigate the self-supervised pretraining methods based on the dimension they operate on.

#### **5.1.2.1. *Self-supervision by spatial context***

Self-supervision by spatial context is applicable in most cases, as it only requires grayscale images, rather than multi-channel images or videos. These methods perform very well, indicating that a large part of the information in the image lies in local spatial correlations.

[156] proposes a method that extracts two random patches from a  $3 \times 3$  grid, and trains the model to classify their relative positions. Since the proposed pretraining task is classification, the loss does not distinguish between predicting south or north, while the ground truth is southwest. Another problem is that the pretraining only shows small patches to the model, which does not allow it to learn higher level representations that span a larger area. For example, the pretrained part learns to represent “head” and “arms”, and the layers at the end learn the likely relative spatial positions of these objects. However, the final layers are stripped before training for the target task, which causes this knowledge to be lost.

A good body of work utilizes permuted image patches, similar to a jigsaw puzzle. [157] samples nine image patches in a  $3 \times 3$  formation, permutes them and feeds them to a Siamese network. The pretraining task is to predict the index of the permutation. Similar to [156]’s work, the objective function does not discriminate between the amount of error. Nevertheless, the proposed pretraining outperforms most self-supervised pretraining methods. [158] develops a set of tricks that improve this method further, including chroma blurring and dependent jittering of patches.

As discussed above, predicting the correct permutation through classification is problematic. [159] proposes to solve this problem by regressing a permutation matrix, rather than hard classification of individual permutations.

An interesting self-supervision task to teach local correlations is to have them generated. [160] removes a part of an image, and has the model paint it back using L2 and adversarial loss. Since this pretraining method requires the model to be used generatively, the choice of architecture is limited.



#### **5.1.2.2. *Self-supervision by cross-channel context***

In general purpose images, cross-channel context basically means the colors in the image to be viable. We can mention [160]’s inpainting study here again, as if the original image is colored, the colors used in the inpainted area has to be inferred from surroundings. There is also a more straightforward method for using cross-channel context. [161] regresses the colors of images that are converted to grayscale using an autoencoder-like architecture. Although the main objective is to colorize, the authors demonstrate that this task also provides good self-supervision. [162] refines this method for self-supervised pretraining and improve the performance. [163] combines pretraining tasks that utilize spatial and cross-channel context. They predict the correct permutation through classification, inpaint a missing patch and colorize all patches.

#### **5.1.2.3. *Self-supervision by temporal context***

[164] mines YouTube videos for patch pairs from consecutive frames. They propose that the patch pair should be more similar, compared to a random patch, and use this as self-supervision. [165] extracts features from three consecutive frames and concatenate them. Then, they binarily classify if the concatenation order is correct. [166] provides stronger self-supervision by predicting the exact ordering of four frames through classification, similar to the studies that use spatial context simultaneously.

#### **5.1.2.4. *Other self-supervision types***

Rather than teaching which transformations damage context (e.g., permutation), [167] aims to teach which transformations do not damage context. They apply various data augmentation transformations to image patches, and train the model to classify the patches that were augmented from the same original image as the same class.

[159] mentions that similar to predicting spatial permutations, we can also permute regressions, yet this also has the problem of needing to know the ground truths. [168] pretrains by predicting egomotion. However, they use additional ground truth for egomotion. Therefore, the proposed method is more of an example of transfer learning, rather than self-supervised pretraining.

### 5.1.3. Coherency ranking

As mentioned in Section 5.1.2, there are existing methods that work on permuted image patches. The pretraining task that these methods utilize is to find the inverse permutation that is going to revert the image back to its original state. This approach is problematic in that permutation estimation with neural nets is not straightforward, as illustrated by the variety of methods proposed in the recent years [156, 157, 159].

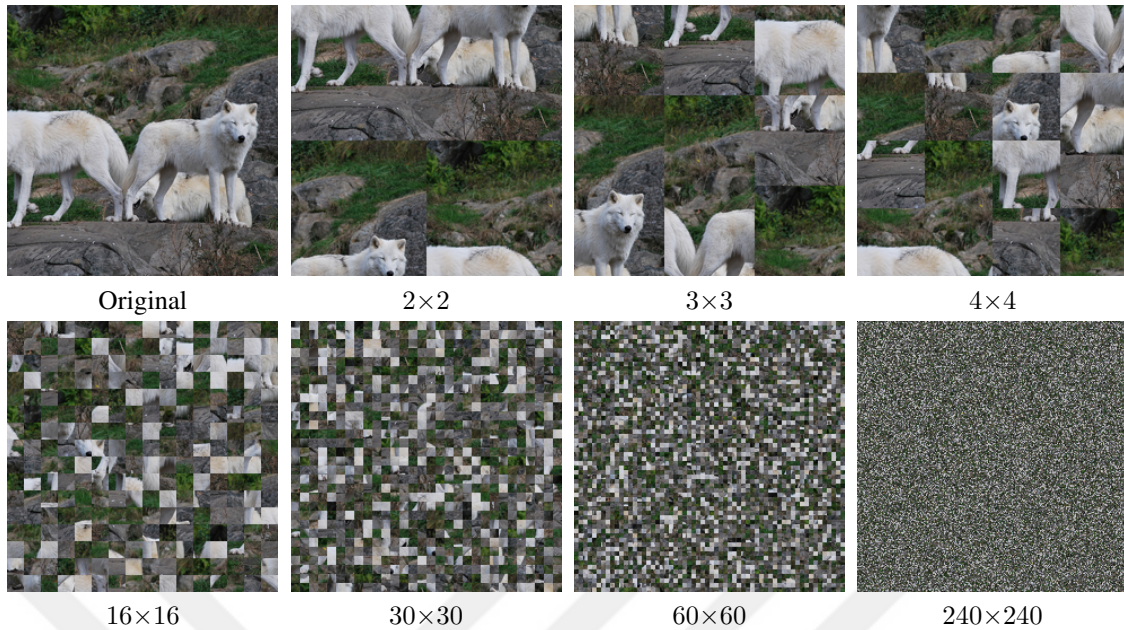
The proposed method models the pretraining task as a ranking problem. Specifically, the pretraining task is to rank two images composed of permuted image patches based on their coherency. This task is equivalent to the one proposed by previous methods, as finding the inverse permutation of a permuted image is trivial once the proposed task is learned (one can swap patches until coherency can no longer improve). The advantage of the proposed task is that binary ranking is relatively straightforward to implement with neural nets.

There are three steps to implementing the proposed method. Firstly, we need to decide on how the training images are to be permuted. Then, a method needs to be designed to quantify the coherency of the resulting images. Finally, a method to rank the image pairs with respect to their coherency values needs to be proposed. We are going to discuss these steps in the following sections.

#### 5.1.3.1. *Permutation granularity*

The first question regarding how the images are to be disturbed is granularity.  $3 \times 3$  tiling is commonly used in the literature [157, 159, 163], with little justification other than the effect on performance. Let us discuss this issue, considering the argument made for the proposed method.

We have argued that since patterns are composed of local interdependencies, they should disappear with spatial disturbance. The most straightforward method of doing this is randomly permuting the pixels of an image, as seen in Fig. 5.2,  $240 \times 240$ . In this image, both the lowest level patterns such as edges, and the highest level patterns such as wolves have disappeared completely. While ranking the original and the permuted images based on coherency, referring to both high and low level patterns would be valid. However, a model that was trained to rank the coherency of these two images would strictly depend

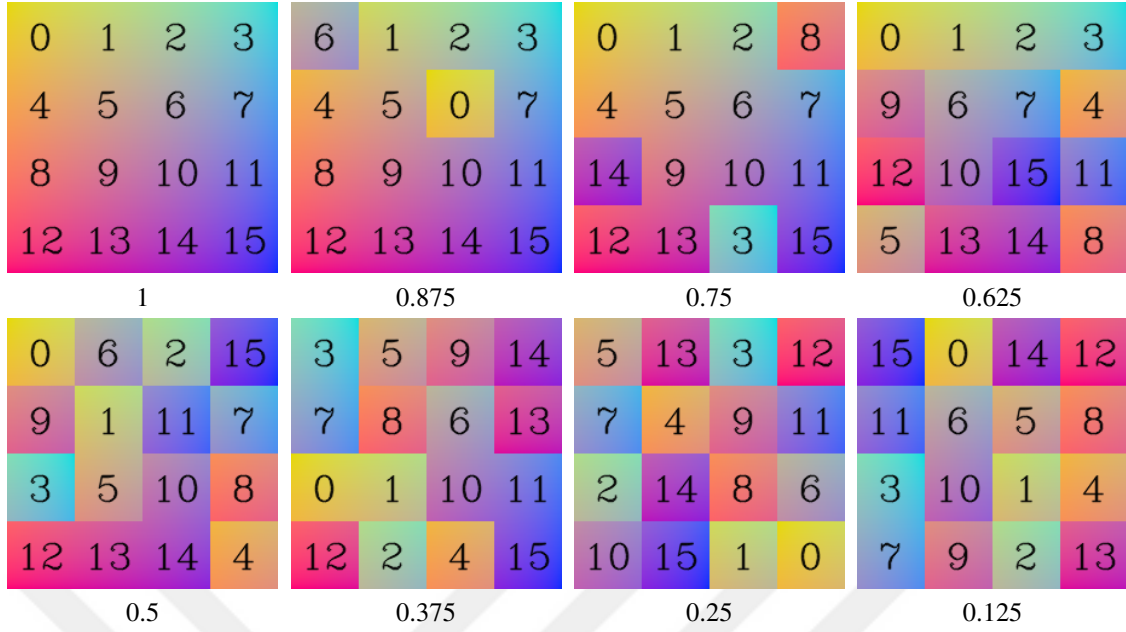


**Figure 5.2:** An image from the ImageNet dataset permuted with differing granularity. Permuting coarsely-tiled images only disturbs high level patterns, while permuting finely-tiled images disturbs all levels of patterns.

on the lower level patterns, as they are easier to learn and more generalizable. Now, let us compare this case with the opposite extremum, where the  $2 \times 2$ -tiled image and the original image from Fig. 5.2 are ranked based on coherency. Here, only the highest level patterns (the wolves) have been damaged, while lower level patterns are left almost completely intact. The model that was trained with this task has to learn to recognize the highest level patterns, which also results in lower level representations to be learned by necessity. Therefore, the degree of granularity used in this task is related to the level of representations we want to draw from the pretraining dataset. For example, if the target task does not require representations to recognize wolves, pretraining by  $2 \times 2$  tiling may be overly specific, and a finer tiling may give better results. Another factor that must be considered is the level of detail in each image. An image from the ImageNet dataset (see Fig.5.2) contains much more detail compared to an image from the CIFAR-10 dataset (see Fig.5.1), hence the tiling must be adjusted accordingly.

### 5.1.3.2. *Quantifying the coherency of permuted images*

To design a metric to estimate the coherency of permuted images, we have to make some assumptions about the data. Firstly, we assume that the original image has maximum



**Figure 5.3:** A 2D color map [169] is permuted randomly, and the respective coherency values are given below. Disturbances in color gradients indicate incoherency.

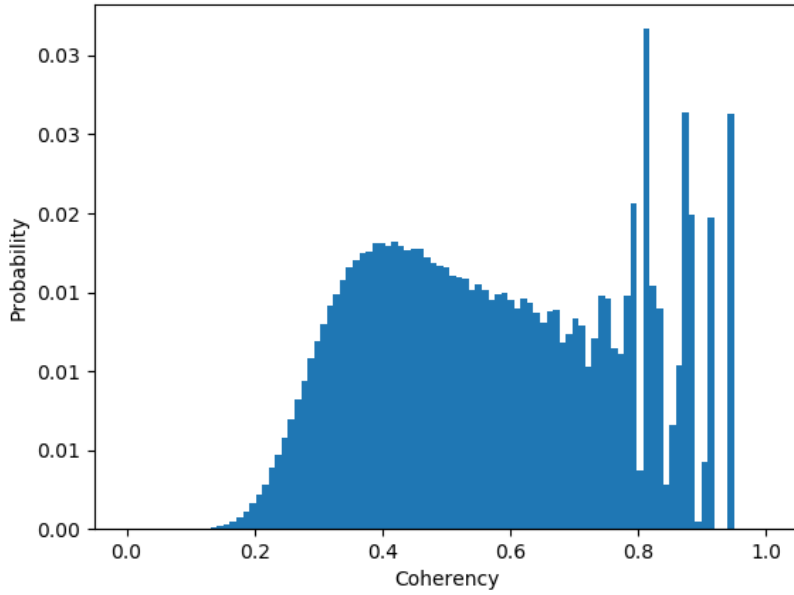
coherency. This is not always the case, as some photographs are taken specifically to produce optical illusions, but this is rare enough to be disregarded. Secondly, we assume that the information in the original image is distributed uniformly. This is obviously not correct when the scene is composed of an object and background (e.g., Fig. 5.2). Due to these assumptions, the proposed metric of coherency is merely an estimate.

We have defined a pattern as local interdependencies, and argued that spatial disturbances would damage it. Therefore, our coherency metric,  $\gamma$ , should quantify the change in relative spatial positions resulting from the permutation. Where  $n$  is the number of tiles,  $\mathbf{x}_{ij}$  is the 2D distance vector between  $i$ -th tile and  $j$ -th tile in the original image, and  $\mathbf{y}_{ij}$  is the 2D distance vector between the  $i$ -th tile and  $j$ -th tile in the permuted image,  $\gamma$  can be defined as follows:

$$\gamma = 1 - \frac{1}{n^2} \sum_i^n \sum_j^n \|\mathbf{x}_{ij} - \mathbf{y}_{ij}\| \quad (5.1)$$

See Fig. 5.3 for an illustration of permuted images and respective coherency values. The original image has the maximum coherency value of 1. As the number of tiles that are permuted increase and neighboring colors clash, the coherency value decreases.

We can observe that although some images in Fig. 5.3 are showing a positive co-

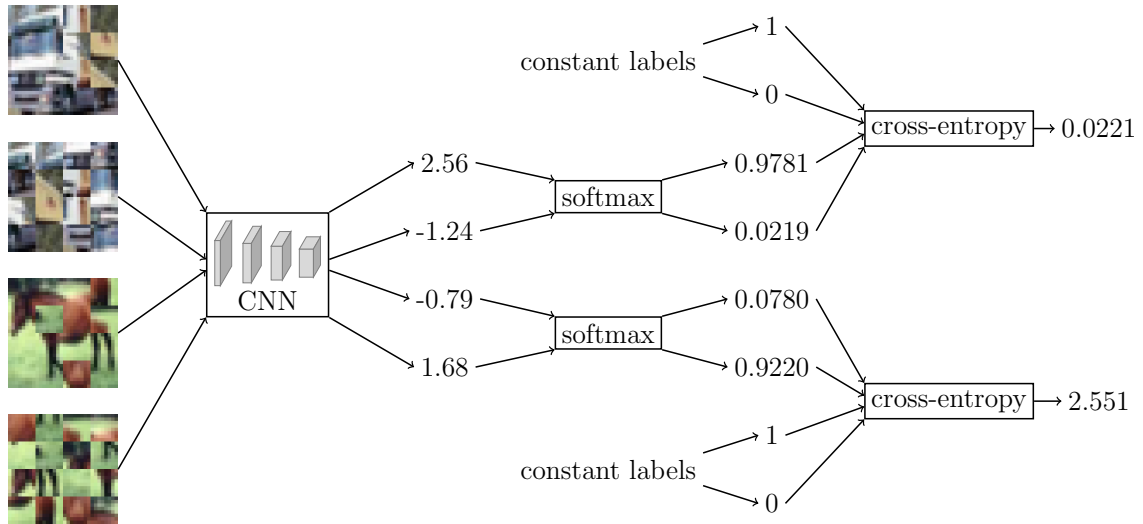


**Figure 5.4:** Probability density function of coherency values when all patches of a  $4 \times 4$ -tiled image are permuted randomly, measured with 1,000,000 samples.

herency value, they are perceived to be completely incoherent. This can be explained through the Helmholtz principle, which states that for a Gestalt to be perceived, it should be unlikely for it to occur spontaneously in noise [123]. Therefore, if the coherency value of a permutation is below the expected coherency value of a completely random permutation, that permutation would also be perceived to be incoherent. Let us calculate this threshold value using a Monte Carlo simulation approach. See Fig. 5.4 for the distribution of the resulting coherency when all tiles are randomly permuted. The cumulative distribution reaches 0.5 at the coherency value of 0.55, which is also the point where the density peaks. This indicates that if we would need to binarily classify randomly permuted images to be incoherent or not, 0.55 would be the correct threshold according to the Helmholtz principle, which can be validated visually from Fig. 5.3.

### 5.1.3.3. *Implementation details*

The final step of the proposed method is learning to rank the coherency values of randomly permuted images. To do this, we propose to feed the image pairs in the same batch, calculate a coherency score for each image using the CNN, and calculate a ranking loss using the coherency score pairs. In practice, this is identical to using a Siamese network,

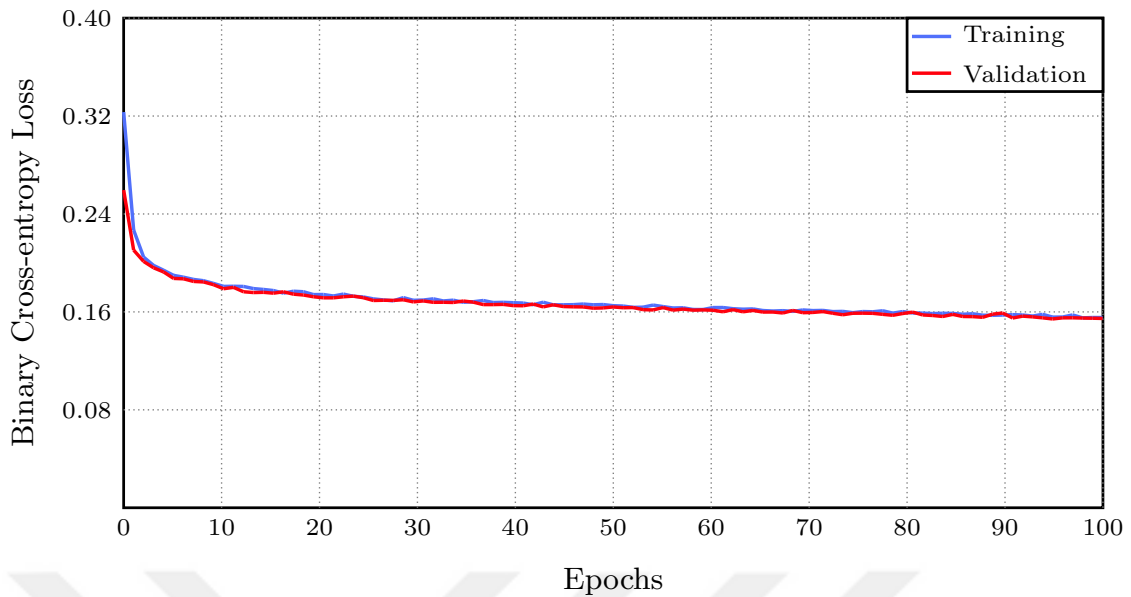


**Figure 5.5:** Training for ranking image pairs in a batch. Batch scores are matched pairwise and ranked.

where each twin takes one of the image pairs as input [170]. Note that this should not be confused with the Siamese architectures in self-supervised literature, where each twin takes only an image patch [156, 157, 159, 163].

The proposed method can use any architecture that was designed for ImageNet object classification (e.g., ResNet-101 [34]). We remove the final layer of the model, and replace it with a densely connected layer with a single output and no activation function. The resulting model produces a coherency-related score, based on a single input image. To train this model with a ranking task, we need to produce two scores from respective images, and compare them. See Fig. 5.5 for an illustration. With a batch size of 4 (chosen for illustration), we use 2 versions of each image. The CNN produces 4 scores for these images, which are then fed into the softmax functions in pairs. Following this, we calculate the cross-entropy loss similar to regular binary classification. This ranking by binary classification is similar to the work by [171], where large scale data is ranked using robust binary classification. Similarly, we are learning to rank high dimensional data by sparsely sampling pairs and binarily classifying them.

Another implementation detail about the proposed task is data generation. We generate permuted images and calculate coherency scores while the GPUs are processing the previous batch, which does not require extra processing time. The resulting training becomes truly stochastic, as each training example pair is used only once. Accordingly, the need for regularization is minimal. There is also the additional advantage of not needing



**Figure 5.6:** Training and validation losses across the epochs for the proposed self-supervised pretraining method.

excessive storage space, which would be a major problem for distribution (e.g., offline-permuted ImageNet dataset would be enormous).

#### 5.1.4. Experimental Results

We present the experimental results under two sections. In the first one, we discuss how the proposed pretraining is done and the resulting performance in the surrogate spatial coherency ranking task. Following this, we use the pretrained model as an initialization point for fine-tuning.

##### 5.1.4.1. Pretraining

The pretraining consists of training a randomly initialized ResNet-101 [20] model using the spatial coherency ranking problem proposed in Section 5.1.3. The learning rate is set to 0.0001 and the weight decay is set to 0. Recall that we have also not used any regularization for Gestaltist learning in Section 4.3.4. The proposed self-supervised pretraining and Gestaltist learning methods have the common property of never using the same training examples more than once, which makes regularization obsolete. This is why a weight decay of 0 is optimal in this case.

See Figure 5.6 for the training and validation losses of the proposed self-supervised

pretraining method. Note that even we have not used any regularization, training and validation losses are identical, and there is no trend upwards that indicates overfitting. This shows that the choice of 0 weight decay is correct. Again, we refer the reader to Figure 4.18 where the validation loss for Gestaltist learning was plotted. Similar to Figure 5.6, the validation loss has saturated and did not show signs of overfitting beyond that point.

Note that although the loss saturates at some point, it continues decreasing at a numerically insignificant rate. We have stopped training at 100 epochs as this decrease slowed down to a negligible amount. At the end of the pretraining, the model is able to rank image pairs with 91.5% accuracy.

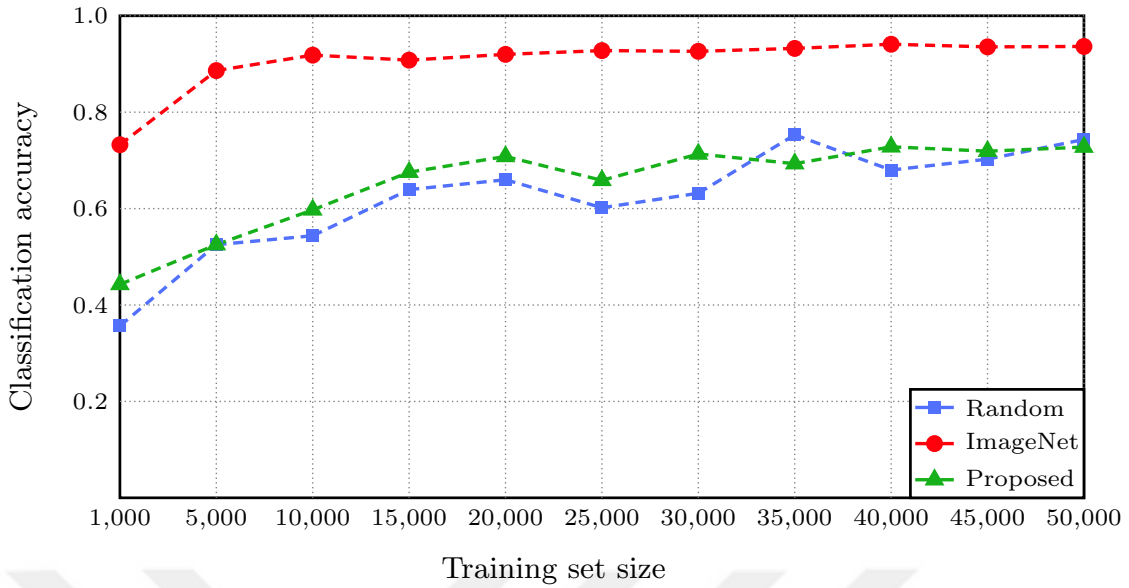
#### **5.1.4.2. *Fine-tuning***

We compared the proposed method with random initialization and supervised ImageNet pretraining in classification tasks. For this, we used the CIFAR-10 dataset and the labeled segments of the STL-10 dataset. For all initialization methods, we used ResNet-101 [20] by replacing its final layer with a dense layer and a softmax classifier with its number of outputs equal to the number of classes in the dataset. Learning rate was set to 0.0001, weight decay was set to 0.0001, and the training was stopped once validation accuracy calculated on the test set stopped increasing for 5 consecutive epochs. The classification accuracy reported is this maximum validation accuracy achieved at the test set.

The CIFAR-10 dataset is composed of 50,000 training images and 10,000 test images of  $32 \times 32$  size and 10 classes. The training images are segmented as 5 batches of 10,000 training sets. However, it is more common to use all 50,000 for training. In this study, we repeated the experiments with training set sizes of 1,000 and ranging from 5,000 to 50,000 with 5,000 increments. Furthermore, we did not apply any data augmentation. The aim of this approach is to show the effect of different pretraining methods for different amounts of fine-tuning data.

The compared initialization methods are random initialization, supervised ImageNet pretraining and the proposed self-supervised pretraining method. See Figure 5.7 for the maximum classification accuracies reached with each method. Supervised ImageNet pretraining vastly outperforms the other two initialization methods as expected. The proposed self-supervised pretraining is better when there is not enough fine-tuning





**Figure 5.7:** Maximum classification accuracies on the CIFAR-10 test set using different initialization methods with different training set sizes.

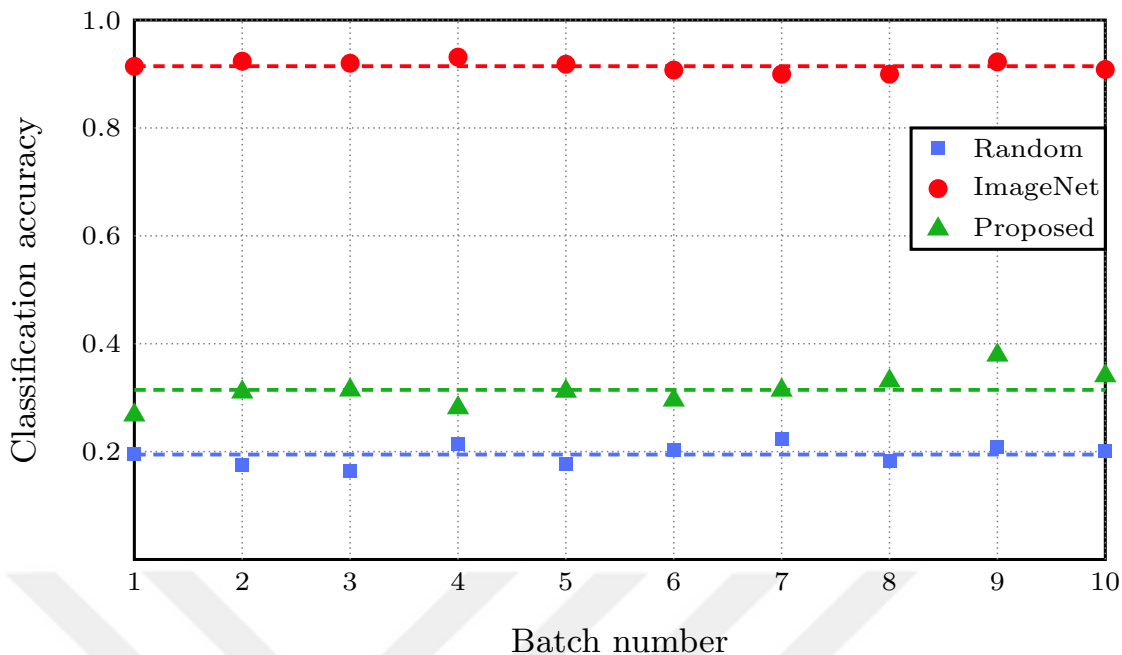
data, yet random initialization catches up once the fine-tuning set is large enough.

STL-10 also includes images from 10 classes, but this time the images are of size  $96 \times 96$ . The training images are divided into 10 folds, each fold containing 100 examples. The test set contains 8000 images. The dataset also contains 100,000 unlabeled images to be used for unsupervised training. These images could have been used for self-supervision, yet since we have already used the larger ImageNet dataset for this purpose, we do not need to do this.

See Figure 5.8 for the maximum classification accuracies reached with each method. The main difference of this experiment from the CIFAR-10 experiment is that the number of labeled images is far smaller. To close this gap, unlike the CIFAR-10 experiment, we used data augmentation methods such as translation and mirroring. We can observe once again that the proposed self-supervised pretraining method is superior to random initialization when the amount of fine-tuning data is small. Again, as expected, ImageNet pretraining outperforms both methods across all batches.

### 5.1.5. Conclusion

Self-supervised pretraining is superior to unsupervised pretraining when the pretrained model is to be fine-tuned in a supervised manner, as it is more compatible with this following step. To apply self-supervised training, one needs to design a surrogate task for



**Figure 5.8:** Maximum classification accuracies on the STL-10 test set using different initialization methods and training batches. The dashed lines indicate the mean classification accuracy across the batches.

which problems and their solutions can be generated. Furthermore, the pretrained model should only be able to solve this problem using fundamental vision skills, rather than spurious statistical relationships in the data.

CNNs are designed to recognize spatial coherencies. Therefore, a self-supervision task that depends on this skill would be suitable. Although there are self-supervision surrogate tasks that depend on recognizing spatial coherency in the literature, they are rather contrived and impose constraints on the CNN model to be pretrained. We have posed the same spatial recognition problem in a more simpler way, namely as a problem of ranking spatial coherencies of two variants of the same image. To be able to do that, we proposed methods to disturb the spatial coherencies of images and quantify these disturbances.

A deep CNN model was pretrained with the proposed self-supervised task and was able to learn to rank images based on their spatial coherencies accurately. Furthermore, the pretrained model was fine-tuned for image classification and was shown to outperform randomly initialized models when the amount of fine-tuning data was small.

The proposed pretraining method did not outperform random initialization in all cases. This is likely because the weight parameters initialized randomly is more con-

ductive to fine-tuning compared to weight parameters pretrained for 100 epochs with the ImageNet dataset. Therefore, although the pretraining task does not overfit, it may need to be stopped earlier for better performance on the fine-tuning task. We plan to discover the exact tricks to maximize the potential of the proposed self-supervised pretraining method in future work.



## 6. CONCLUSION

Deep learning is the practice of learning stacked nonlinear models. Deep hierarchical models have a large number of parameters. Although models with a large number of parameters are more representative—in a machine learning context—they also require more training data. Without a large amount of data, deep models overfit, which causes them to perform worse than a shallower model.

The obvious solution to this issue is to simply gather more training data, build larger models and use more computational resources, which has been the industry’s approach. However, this approach yields diminishing returns, and is not likely to cause new quantum leaps.

In this study, we have searched for ways to use well-performing deep models without gathering a large amount of task-specific data. In general, this is done by softly limiting the degrees of freedom a model has by constraining it. This way, although the model still has a large number of parameters, it is no longer as eager to fit. The practice of constraining the flexibility of a model to prevent it from overfitting is called regularization.

Weight decay and dropout are some of the most common regularization methods used in deep learning. In fact, in most vision applications, it is impossible to reach state of the art performance without employing some kind of regularization. In contrast, these regularization methods are detrimental while trying to learn from a generative model that can produce infinitely many examples. Therefore, if we are looking for a way to train deep models with a small amount of data, we should not look further than regularization.

Although regularization is commonly thought of as an explicit part of the objective function, it can also be applied indirectly. For example, where a model is initialized at limits the region of the parameter space it can end up at. Therefore, an intelligent initialization can be regarded as regularization, as it can have us require a smaller amount of training data to reach an equivalent performance.

For abstract tasks—such as the ones we have been dealing with in vision for the last decade—equivalent examples lie on a very wrinkly manifold in the input space. Similarly, the objective functions for these tasks are full of local minima and saddle points.

Although simple methods based on manipulating the random distributions used to initialize parameters help, they are too primitive to be relied upon by themselves for such complex problems.

Pretraining is a type of initialization, as its objective is to start the training with the target task at a favorable starting position. Moreover, it can be considered as a type of regularization, as it provides exactly the same types of benefits. Historically, unsupervised pretraining had been seen as a must step before supervised training with the target task. Later on, it was seen that with a collection of alternative regularization methods, unsupervised pretraining can be omitted, and models can be trained directly in a supervised manner. However, this does not mean that pretraining has become obsolete. On the contrary, it is general practice to initialize models by supervised pretraining with a large dataset.

The majority of the mainstream computer vision research being done today uses regular photos and videos. In these cases, ImageNet pretraining is a very good option for initialization. It can also be argued that vision research is focused on problems that respond well to ImageNet pretraining, because it allows to demonstrate a substantial improvement in performance with minimal effort.

Although ImageNet pretraining is an extremely common initialization tool, it is either that, or random initialization in today's literature. Accordingly, we asked two questions:

- 1– When does ImageNet pretraining lose its usefulness?
- 2– When that happens, what alternatives do we have other than random initialization?

In a general sense, ImageNet pretraining loses its usefulness when the target task is vastly different from ImageNet classification. There are two ways of assessing this difference. One can either look at the two tasks and intuitively decide on whether or not to use ImageNet pretraining, or they can approach the problem empirically.

There are some obstacles in the way of trying out ImageNet pretraining with a particular target task. Firstly, if a custom architecture is being used for the target task—which is the case for most out of the ordinary vision tasks—one has to do the pretraining themselves. This is very unattractive for hit-and-run researchers who mass publish unexplored use-cases of deep learning. Moreover, even if they already have a pretrained model, this

doubles the amount of experiments they need to run, as whether to pretrain or not is yet another hyperparameter. As a result, there is a tendency to omit ImageNet pretraining whenever it would not “make sense”.

In this study, we demonstrated with two use-cases from very different domains that ImageNet pretraining is likely to be helpful in more cases than commonly assumed. In the first use-case, it was shown that initialization by ImageNet pretraining is very beneficial for recognizing power lines from aerial images. Moreover, these benefits persist even when the images are in the infrared spectrum. In the second use-case, highest-level features extracted with an ImageNet pretrained model is used to classify individual cell images. Surprisingly, this method resulted comparable results to its contemporaries, even though we had not fine-tuned the model.

The results we have achieved with these two use-cases lets us answer the first question of when ImageNet pretraining loses its usefulness: Not as soon as it is commonly assumed. Therefore, we see use in further studies exploring the benefits of ImageNet pretraining in odd domains such as biomedical imaging, remote sensing, and maybe even non-vision applications.

Let us consider pretraining as a type of regularization again to understand the cases where ImageNet pretraining is not suitable. There are two reasons for a regularization method to not provide any benefits. Firstly, the regularization may be constraining the model with a false supposition. Consider weight decay, which suppresses the L2-norm of model parameters, as a high L2-norm is assumed to be indicative of overfitting. This hypothesis is often correct, which causes weight decay to improve performance. Alternatively, if we had designed a new method that maximizes the L2-norm of model parameters, it would encourage overfitting. The equivalent of this in our case is where the knowledge extracted from ImageNet pretraining not being useful in the target task, or even causing detriment. The second reason a type of regularization may not be suitable for a particular task is because its level of intensity is not optimal. If we come back to the weight decay example, the L2-norm of the model parameters are added to the loss function weighted by a  $\lambda$  parameter. If  $\lambda$  is too large, weight decay constrains the model too much, and vice versa. For both cases, the applied weight decay is not optimally beneficial. There is no  $\lambda$  parameter to tune for ImageNet pretraining, which limits its benefits proportionally to the difference between the target task and ImageNet pretraining. In the case where the differ-

ence is extreme, we observe the first case, i.e., the assumption that knowledge extracted from ImageNet pretraining is going to be beneficial in the target task proves to be false.

Then how do we use a deep model, not gather a large labeled dataset and have the model not overfit? For the first case, where the ImageNet dataset (or any alternative dataset) is not applicable to our target task, we propose to use Gestalt learning. In this approach, we analytically design a generative model that stochastically produces examples. Then, the generated examples are used to train a deep model. This method is entirely data independent, and solely relies on the domain knowledge injected to the model through the generative model design. In this sense, the proposed method is the exact opposite of the recent hyper-data dependent methods.

Designing the generative model correctly is vital for Gestalt learning to be successful. We argued that these designs should be based on the fundamentals of perceptual psychology, and demonstrated cases where doing so results in successful implementations. Moreover, we have validated that the models trained with such generative models conform to the theory of Gestalt psychology using the Helmholtz principle.

Since Gestalt learning is completely data independent, it can be used when there is absolutely no training data for a particular vision problem. However, it can also be used to initialize models where there is a very limited amount of labeled training examples available. Since it is clearly demonstrated that the knowledge gained through Gestalt learning can be beneficial at the target task, a model pretrained with Gestalt learning would be more preferable to a randomly initialized model for fine-tuning with the target task. This theory should be supported by further empirical evidence.

In the second case, we want to transfer from the ImageNet dataset (or, again, any alternative dataset), yet want to dial down  $\lambda$ , i.e., we want to decrease the specificity of the transferred knowledge. Recall that CNNs learn by modeling spatial correlations, and the locality of these correlations control the level of the representations. For example, a high level pattern, such as a person, takes up a large part of the image, while a lower level pattern, such as color constancy, can be represented by a  $3 \times 3$  blob. By allowing the CNN to only model highly local correlations, only lower level representations can be transferred.

To teach the CNN to model a specific level of spatial correlations, we proposed a novel pretraining task. An image is divided into tiles, and these tiles are shuffled with two

distinct permutations. Then, the model is asked to rank this image pair in terms of spatial coherency. The size of the tiles determine the level of representations the model can learn from this task. Again, this theorized relation between the size of the tiles and the level of transferred representations should be tested with further studies.

Note that this proposed pretraining task does not require the training examples to be labeled, yet does supervised training. This is called self-supervised training in the literature. Therefore, this method not only solves the problem of ImageNet not being completely suitable for a particular target task, it also allows any large unlabeled dataset to be used in place of ImageNet. This makes it very suitable for domains such as biomedical imaging and remote sensing where there is abundant unlabeled data.

The proposed self-supervised pretraining method can be seen as a hybrid of traditional ImageNet pretraining and Gestalt learning. It is somewhat data dependent, albeit it does not require the data to be labeled. On the other hand, the pretraining examples it uses are generated by an analytically designed generative model. This is in accordance with our problem definition, where we wanted to transfer from the ImageNet dataset, yet limit the level of representations.

In summary, we have three main findings. The first is that ImageNet pretraining is a valid initialization method even in very different domains, and should not be overlooked easily. The second is that even if there is no training data available to transfer from, deep models can be trained in a completely data independent manner. Finally, a self-supervised pretraining method based on design fundamentals of CNNs is proposed to bridge the gap between these two cases.



## REFERENCES

- [1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.
- [2] Y. Bengio, A. Courville, and P. Vincent, “Representation learning: A review and new perspectives,” *IEEE Trans. Pattern Anal. and Mach. Intell. (TPAMI)*, vol. 35, no. 8, pp. 1798–1828, 2013.
- [3] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.
- [4] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, “Greedy layer-wise training of deep networks,” in *Proc. Advances in Neural Inform. Proc. Systems (NIPS)*, pp. 153–160, 2006.
- [5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proc. Advances in Neural Inform. Proc. Systems (NIPS)*, pp. 1097–1105, 2012.
- [6] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *Int. J. Comput. Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.
- [7] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray, “Visual categorization with bags of keypoints,” in *Proc. Workshop on Statistical Learning in Comput. Vision, European Conf. Comp. Vision (ECCV)*, 2004.
- [8] F. Perronnin and C. Dance, “Fisher kernels on visual vocabularies for image categorization,” in *Proc. IEEE Conf. Comput. Vision and Pattern Recognition (CVPR)*, 2007.

- [9] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, “Learning hierarchical features for scene labeling,” *IEEE Trans. Pattern Anal. and Mach. Intell. (TPAMI)*, vol. 35, no. 8, pp. 1915–1929, 2013.
- [10] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, “Deep, big, simple neural nets for handwritten digit recognition,” *Neural Computation*, vol. 22, no. 12, pp. 3207–3220, 2010.
- [11] D. C. Cireşan, U. Meier, J. Masci, and J. Schmidhuber, “Multi-column deep neural network for traffic sign classification,” *Neural Networks*, vol. 32, pp. 333–338, 2012.
- [12] Y. Sun, X. Wang, and X. Tang, “Deeply learned face representations are sparse, selective, and robust,” in *Proc. IEEE Conf. Comput. Vision and Pattern Recognition (CVPR)*, pp. 2892–2900, 2015.
- [13] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, “Why does unsupervised pre-training help deep learning?,” *J. Mach. Learning Res. (JMLR)*, vol. 11, pp. 625–660, 2010.
- [14] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, “Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms,” in *Proc. ACM Int. Conf. Knowledge Discovery and Data Mining*, pp. 847–855, 2013.
- [15] I. Guyon, K. Bennett, G. Cawley, H. J. Escalante, S. Escalera, T. K. Ho, N. Macia, B. Ray, M. Saeed, A. Statnikov, and E. Viegas, “Design of the 2015 ChaLearn AutoML challenge,” in *Proc. Int. Joint Conf. Neural Networks*, 2015.
- [16] J. Snoek, H. Larochelle, and R. P. Adams, “Practical Bayesian optimization of machine learning algorithms,” in *Proc. Advances in Neural Inform. Proc. Systems (NIPS)*, pp. 2951–2959, 2012.
- [17] L. Xie and A. Yuille, “Genetic CNN,” in *Proc. IEEE Int. Conf. Comput. Vision (ICCV)*, pp. 1379–1388, 2017.

- [18] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” in *Proc. IEEE Conf. Comput. Vision and Pattern Recognition (CVPR)*, pp. 8697–8710, 2018.
- [19] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *CoRR*, vol. abs/1409.1556, 2014.
- [20] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vision and Pattern Recognition (CVPR)*, 2016.
- [21] D. Cireşan, U. Meier, and J. Schmidhuber, “Multi-column deep neural networks for image classification,” *arXiv preprint arXiv:1202.2745*, 2012.
- [22] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” 2016.
- [23] M. Huh, P. Agrawal, and A. A. Efros, “What makes ImageNet good for transfer learning?,” 2016.
- [24] Ç. Gülçehre and Y. Bengio, “Knowledge matters: Importance of prior information for optimization,” *J. Mach. Learning Res. (JMLR)*, vol. 17, no. 8, pp. 1–32, 2016.
- [25] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.
- [26] J. M. Benítez, J. L. Castro, and I. Requena, “Are artificial neural networks black boxes?,” *IEEE Trans. Neural Networks*, vol. 8, no. 5, pp. 1156–1164, 1997.
- [27] Y. Bengio, “Practical recommendations for gradient-based training of deep architectures,” *Neural Networks: Tricks of the Trade*, pp. 437–478, 2012.
- [28] C. Cortes and V. Vapnik, “Support-vector networks,” *Mach. Learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [29] L.-J. Cao and F. E. H. Tay, “Support vector machine with adaptive parameters in financial time series forecasting,” *IEEE Trans. Neural Networks*, vol. 14, no. 6, pp. 1506–1518, 2003.

- [30] C. H. Ding and I. Dubchak, “Multi-class protein fold recognition using support vector machines and neural networks,” *Bioinformatics*, vol. 17, no. 4, pp. 349–358, 2001.
- [31] B. Samanta, “Gear fault detection using artificial neural networks and support vector machines with genetic algorithms,” *Mech. Systems and Signal Proc.*, vol. 18, no. 3, pp. 625–644, 2004.
- [32] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition Volume 1: Foundations*. MIT Press, 1986.
- [33] M. T. Hagan, H. B. Demuth, M. H. Beale, and O. De Jesús, *Neural Network Design*. PWS Publishing Company, 1996.
- [34] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification,” in *Proc. IEEE Int. Conf. Comput. Vision (ICCV)*, pp. 1026–1034, 2015.
- [35] K. Jarrett, K. Kavukcuoglu, and Y. LeCun, “What is the best multi-stage architecture for object recognition?,” in *Proc. IEEE Conf. Comput. Vision and Pattern Recognition (CVPR)*, pp. 2146–2153, 2009.
- [36] S. Sabour, N. Frosst, and G. E. Hinton, “Dynamic routing between capsules,” in *Proc. Advances in Neural Inform. Proc. Systems (NIPS)*, pp. 3856–3866, 2017.
- [37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proc. IEEE Conf. Comput. Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.
- [38] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *Proc. Int. Conf. Mach. Learning (ICML)*, pp. 448–456, 2015.
- [39] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of Control, Signals and Syst.*, vol. 2, no. 4, pp. 303–314, 1989.

- [40] R. E. Bellman, *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.
- [41] Y. Bengio, “Learning deep architectures for AI,” *Foundations and Trends in Mach. Learning*, vol. 2, pp. 1–127, 2009.
- [42] Y. Bengio, F. Bastien, A. Bergeron, N. Boulanger-Lewandowski, T. Breuel, Y. Chherawala, M. Cisse, M. Côté, D. Erhan, J. Eustache, X. Glorot, X. Muller, S. Pannetier Lebeuf, R. Pascanu, S. Rifai, F. Savard, and G. Sicard, “Deep learners benefit more from out-of-distribution examples,” in *Proc. Int. Conf. Artificial Intell. and Statist. (AISTATS)*, pp. 164–172, 2011.
- [43] Z. Liu, P. Luo, X. Wang, and X. Tang, “Deep learning face attributes in the wild,” in *Proc. IEEE Int. Conf. Comput. Vision (ICCV)*, 2015.
- [44] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proc. Int. Conf. Artificial Intell. and Statist. (AISTATS)*, pp. 249–256, 2010.
- [45] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, pp. 533–536, 1986.
- [46] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, “Greedy layer-wise training of deep networks,” in *Proc. Advances in Neural Inform. Proc. Systems (NIPS)*, pp. 153–160, 2007.
- [47] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Proc. Advances in Neural Inform. Proc. Systems (NIPS)*, pp. 2672–2680, 2014.
- [48] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, “Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion,” *J. Mach. Learning Res. (JMLR)*, vol. 11, pp. 3371–3408, 2010.
- [49] C. Doersch and A. Zisserman, “Multi-task self-supervised visual learning,” in *Proc. IEEE Int. Conf. Comput. Vision (ICCV)*, 2017.

- [50] P. Smolensky, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, ch. Chapter 6: Information Processing in Dynamical Systems: Foundations of Harmony Theory, pp. 194—281. MIT Press, 1986.
- [51] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science*, vol. 313, no. 5786, pp. 504–507, 2006.
- [52] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” in *Proc. Int. Conf. Mach. Learning (ICML)*, pp. 1096–1103, 2008.
- [53] R. Salakhutdinov and G. E. Hinton, “Deep boltzmann machines,” in *Proc. Int. Conf. Artificial Intell. and Statist. (AISTATS)*, pp. 448–455, 2009.
- [54] Y. Bengio and A. C. Courville, “Deep learning of representations,” *Handbook on Neural Inform. Proc.*, vol. 49, pp. 1–28, 2013.
- [55] Y. Bengio, É. Thibodeau-Laufer, G. Alain, and J. Yosinski, “Deep generative stochastic networks trainable by backprop,” in *Proc. Int. Conf. Mach. Learning (ICML)*, pp. 226–234, 2014.
- [56] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent, “The difficulty of training deep architectures and the effect of unsupervised pre-training,” in *Proc. Int. Conf. Artificial Intell. and Statist. (AISTATS)*, pp. 153–160, 2009.
- [57] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *Int. J. Comput. Vision (IJCV)*, vol. 60, no. 2, pp. 91–110, 2004.
- [58] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in *Proc. European Conf. Comput. Vision (ECCV)*, pp. 818–833, 2014.
- [59] R. Raina, A. Madhavan, and A. Y. Ng, “Large-scale deep unsupervised learning using graphics processors,” in *Proc. Int. Conf. Mach. Learning (ICML)*, pp. 873–880, 2009.

- [60] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in *Proc. IEEE Conf. Comput. Vision and Pattern Recognition (CVPR)*, pp. 248–255, 2009.
- [61] Y.-L. Boureau, J. Ponce, and Y. LeCun, “A theoretical analysis of feature pooling in visual recognition,” in *Proc. Int. Conf. Mach. Learning (ICML)*, pp. 111–118, 2010.
- [62] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proc. Int. Conf. Mach. Learning (ICML)*, pp. 807–814, 2010.
- [63] I. Sutskever, J. Martens, G. Dahl, and G. E. Hinton, “On the importance of initialization and momentum in deep learning,” in *Proc. Int. Conf. Mach. Learning (ICML)*, pp. 1139–1147, 2013.
- [64] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *J. Mach. Learning Res. (JMLR)*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [65] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” in *Proc. Advances in Neural Inform. Proc. Systems (NIPS)*, pp. 91–99, 2015.
- [66] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Trans. Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.
- [67] R. Ge, F. Huang, C. Jin, and Y. Yuan, “Escaping from saddle points – online stochastic gradient for tensor decomposition,” in *Proc. Conf. Learning Theory (COLT)*, pp. 797–842, 2015.
- [68] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, “Efficient backprop,” *Neural Networks: Tricks of the Trade*, pp. 9–48, 2012.
- [69] A. Saxe, P. W. Koh, Z. Chen, M. Bhand, B. Suresh, and A. Y. Ng, “On random weights and unsupervised feature learning,” in *Proc. Int. Conf. Mach. Learning (ICML)*, pp. 1089–1096, 2011.

- [70] J. Martens, “Deep learning via Hessian-free optimization,” in *Proc. Int. Conf. Mach. Learning (ICML)*, pp. 735–742, 2010.
- [71] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features in deep neural networks?,” in *Proc. Advances in Neural Inform. Proc. Systems (NIPS)*, pp. 3320–3328, 2014.
- [72] T. Serre, L. Wolf, and T. Poggio, “Object recognition with features inspired by visual cortex,” in *Proc. IEEE Conf. Comput. Vision and Pattern Recognition (CVPR)*, pp. 994–1000, 2005.
- [73] S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Trans. Knowledge and Data Eng.*, vol. 22, no. 10, pp. 1345–1359, 2010.
- [74] P. J. Grother, “NIST special database 19 handprinted forms and characters database,” *Nat. Inst. of Standards and Technol.*, 1995.
- [75] R. Battiti, “First-and second-order methods for learning: between steepest descent and Newton’s method,” *Neural Computation*, vol. 4, no. 2, pp. 141–166, 1992.
- [76] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in *Proc. Int. Conf. Mach. Learning (ICML)*, pp. 41–48, 2009.
- [77] M. P. Kumar, B. Packer, and D. Koller, “Self-paced learning for latent variable models,” in *Proc. Advances in Neural Inform. Proc. Systems (NIPS)*, pp. 1189–1197, 2010.
- [78] J. Hosang, M. Omran, R. Benenson, and B. Schiele, “Taking a deeper look at pedestrians,” in *Proc. IEEE Conf. Comput. Vision and Pattern Recognition (CVPR)*, pp. 4073–4082, 2015.
- [79] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, “Multi-view convolutional neural networks for 3D shape recognition,” in *Proc. IEEE Int. Conf. Comput. Vision (ICCV)*, pp. 945–953, 2015.
- [80] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, “DeCAF: A deep convolutional activation feature for generic visual recognition,” in *Proc. Int. Conf. Mach. Learning (ICML)*, pp. 647–655, 2014.



- [81] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, “CNN features off-the-shelf: An astounding baseline for recognition,” in *Proc. IEEE Conf. Comput. Vision and Pattern Recognition Workshops (CVPRW)*, pp. 806–813, 2014.
- [82] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *Proc. IEEE Conf. Comput. Vision and Pattern Recognition (CVPR)*, pp. 580–587, 2014.
- [83] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, “Show and tell: A neural image caption generator,” in *Proc. IEEE Conf. Comput. Vision and Pattern Recognition (CVPR)*, pp. 3156–3164, 2015.
- [84] W. Hu, Y. Huang, L. Wei, F. Zhang, and H. Li, “Deep convolutional neural networks for hyperspectral image classification,” *J. Sensors*, 2015.
- [85] Z. Gao, L. Wang, L. Zhou, and J. Zhang, “HEp-2 cell image classification with deep convolutional neural networks,” *IEEE J. Biomedical and Health Informatics*, vol. 21, no. 2, pp. 416–428, 2017.
- [86] Ö. E. Yetgin, B. Benligiray, and Ö. N. Gerek, “Power line recognition from aerial images with deep learning,” *IEEE Trans. Aerospace and Electron. Syst.*, 2018.
- [87] B. Benligiray and Ö. N. Gerek, “Visualization of power lines recognized in aerial images using deep learning,” in *Proc. Signal Proc. and Commun. Application Conf. (SIU)*, 2018.
- [88] B. Benligiray and H. Çınar Akakin, “HEp-2 cell classification using a deep neural network trained for natural image classification,” in *Proc. Signal Proc. and Commun. Application Conf. (SIU)*, pp. 1361–1364, 2016.
- [89] J. Candamo, R. Kasturi, D. Goldgof, and S. Sarkar, “Detection of thin lines using low-quality video from low-altitude aircraft in urban settings,” *IEEE Trans. Aerospace and Electron. Syst.*, vol. 45, no. 3, pp. 937–949, 2009.
- [90] Ö. E. Yetgin and Ö. N. Gerek, “Powerline Image Dataset Extra (Infrared-IR and Visible Light-VL) - Classified (Easy and Hard),” 2017. <https://data.mendeley.com/datasets/n6wrv4ry6v/7>.

- [91] Ö. E. Yetgin and Ö. N. Gerek, “Ground Truth of Powerline Dataset (Infrared-IR and Visible Light-VL),” 2017. <https://data.mendeley.com/datasets/twxp8xccsw/8>.
- [92] F. Chollet *et al.*, “Keras,” 2015.
- [93] G. H. John and P. Langley, “Estimating continuous distributions in Bayesian classifiers,” in *Proc. Conf. Uncertainty in Artificial Intell.*, pp. 338–345, 1995.
- [94] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
- [95] E. Frank, M. A. Hall, and I. H. Witten, “The WEKA Workbench. Online Appendix for “Data Mining: Practical Machine Learning Tools and Techniques”.” Morgan Kaufmann, Fourth Edition, 2016.
- [96] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “MobileNets: Efficient convolutional neural networks for mobile vision applications,” *CoRR*, vol. abs/1704.04861, 2017.
- [97] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding,” *CoRR*, vol. abs/1510.00149, 2015.
- [98] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, “Optimizing FPGA-based accelerator design for deep convolutional neural networks,” in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, pp. 161–170, 2015.
- [99] Ö. E. Yetgin and Ö. N. Gerek, “Automatic recognition of scenes with power line wires in real life aerial images using DCT-based features,” *Digital Signal Processing*, 2017.
- [100] K. Simonyan, A. Vedaldi, and A. Zisserman, “Deep inside convolutional networks: Visualising image classification models and saliency maps,” *arXiv preprint arXiv:1312.6034*, 2013.
- [101] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, “Striving for simplicity: The all convolutional net,” in *Proc. Int. Conf. Learning Representations (ICLR) Workshop Track*, 2015.

- [102] M. Sundararajan, A. Taly, and Q. Yan, “Axiomatic attribution for deep networks,” *arXiv preprint arXiv:1703.01365*, 2017.
- [103] N. Otsu, “A threshold selection method from gray-level histograms,” *IEEE Trans. Syst., Man, and Cybern.*, vol. 9, no. 1, pp. 62–66, 1979.
- [104] N. Tajbakhsh, J. Y. Shin, S. R. Gurudu, R. T. Hurst, C. B. Kendall, M. B. Gotway, and J. Liang, “Convolutional neural networks for medical image analysis: Full training or fine tuning?,” *IEEE Trans. Medical Imaging*, vol. 35, no. 5, pp. 1299–1312, 2016.
- [105] A. Noble and D. Boukerroui, “Ultrasound image segmentation: A survey,” *IEEE Trans. Medical Imaging*, vol. 25, no. 8, pp. 987–1010, 2006.
- [106] A. Gholipour, N. Kehtarnavaz, R. Briggs, M. Devous, and K. Gopinath, “Brain functional localization: A survey of image registration techniques,” *IEEE Trans. Medical Imaging*, vol. 26, no. 4, pp. 427–451, 2007.
- [107] A. F. Laine, S. Schuler, J. Fan, and W. Huda, “Mammographic feature enhancement by multiscale analysis,” *IEEE Trans. Medical Imaging*, 1994.
- [108] B. Van Ginneken, B. T. H. Romeny, and M. A. Viergever, “Computer-aided diagnosis in chest radiography: A survey,” *IEEE Trans. Medical Imaging*, vol. 20, no. 12, pp. 1228–1241, 2001.
- [109] W. B. Storch, *Immunofluorescence in Clinical Immunology: A Primer and Atlas*. Springer, 2000.
- [110] S. Manivannan, W. Li, S. Akbar, R. Wang, J. Zhang, and S. J. McKenna, “An automated pattern recognition system for classifying indirect immunofluorescence images of HEp-2 cells and specimens,” *Pattern Recognition*, vol. 51, pp. 12–26, 2016.
- [111] L. Shen, J. Lin, S. Wu, and S. Yu, “HEp-2 image classification using intensity order pooling based features and bag of words,” *Pattern Recognition*, vol. 47, no. 7, pp. 2419–2427, 2014.

- [112] S. Manivannan, W. Li, S. Akbar, R. Wang, J. Zhang, and S. J. McKenna, “HEp-2 cell classification using multi-resolution local patterns and ensemble SVMs,” in *Proc. 1st Workshop on Pattern Recognition Techniques for Indirect Immunofluorescence Images*, pp. 37–40, 2014.
- [113] X. Qi, G. Zhao, C. Li, J. Guo, and M. Pietikäinen, “HEp-2 cell classification via fusing texture and shape information,” *CoRR*, vol. abs/1502.04658, 2015.
- [114] X. Qi, G. Zhao, J. Chen, and M. Pietikäinen, “HEp-2 cell classification: The role of Gaussian Scale Space Theory as a pre-processing approach,” *Pattern Recognition Letters*, vol. 82, pp. 36–43, 2016.
- [115] Z. Gao, J. Zhang, L. Zhou, and L. Wang, “HEp-2 cell image classification with convolutional neural networks,” in *Proc. 1st Workshop on Pattern Recognition Techniques for Indirect Immunofluorescence Images*, pp. 24–28, 2014.
- [116] P. Foggia, G. Percannella, P. Soda, and M. Vento, “Benchmarking HEp-2 cells classification methods,” *IEEE Trans. Medical Imaging*, vol. 32, no. 10, pp. 1878–1889, 2013.
- [117] M. Lin, Q. Chen, and S. Yan, “Network in network,” *CoRR*, vol. abs/1312.4400, 2013.
- [118] T. G. Dietterich and G. Bakiri, “Solving multiclass learning problems via error-correcting output codes,” *J. Artificial Intelligence Research*, pp. 263–286, 1995.
- [119] N. Tinbergen, *The Study of Instinct*. Clarendon Press/Oxford University Press, 1951.
- [120] I. Bushnell, “Mother’s face recognition in newborn infants: Learning and memory,” *Infant and Child Development: An International Journal of Research and Practice*, vol. 10, no. 1-2, pp. 67–74, 2001.
- [121] J. Wagemans, J. H. Elder, M. Kubovy, S. E. Palmer, M. A. Peterson, M. Singh, and R. von der Heydt, “A century of Gestalt psychology in visual perception: I. Perceptual grouping and figure–ground organization,” *Psychological Bulletin*, vol. 138, no. 6, pp. 1172–1217, 2012.

- [122] W. Metzger, L. T. Spillmann, S. T. Lehar, M. T. Stromeyer, and M. T. Wertheimer, *Laws of Seeing*. MIT Press, 2006.
- [123] A. Desolneux, L. Moisan, and J.-M. Morel, *From Gestalt Theory to Image Analysis: A Probabilistic Approach*. Springer-Verlag New York, 2007.
- [124] A. Desolneux, L. Moisan, and J.-M. More, “A grouping principle and four applications,” *IEEE Trans. Pattern Anal. and Mach. Intell. (TPAMI)*, vol. 25, no. 4, pp. 508–513, 2003.
- [125] D.-C. Lou, N.-I. Wu, C.-M. Wang, Z.-H. Lin, and C.-S. Tsai, “A novel adaptive steganography based on local complexity and human vision sensitivity,” *J. Syst. and Software*, vol. 83, no. 7, pp. 1236–1248, 2010.
- [126] H. Yang and A. C. Kot, “Pattern-based data hiding for binary image authentication by connectivity-preserving,” *IEEE Trans. Multimedia*, vol. 9, no. 3, pp. 475–486, 2007.
- [127] D. Lowe, *Perceptual Organization and Visual Recognition*. Springer, 1985.
- [128] Y. Sun, X. Wang, and X. Tang, “Deep learning face representation from predicting 10,000 classes,” in *Proc. IEEE Conf. Comput. Vision and Pattern Recognition (CVPR)*, pp. 1891–1898, 2014.
- [129] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014.
- [130] B. Benligiray and Ö. N. Gerek, “Deep gradient operator,” in *Proc. Int. Conf. Advanced Technologies*, 2019.
- [131] A. Buades, B. Coll, and J.-M. Morel, “A non-local algorithm for image denoising,” in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 60–65, 2005.
- [132] J. Canny, “A computational approach to edge detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 6, pp. 679–698, 1986.
- [133] D. Taubman, “High performance scalable image compression with EBCOT,” *IEEE Transactions on Image Processing*, vol. 9, no. 7, pp. 1158–1170, 2000.

- [134] Q. You, H. Jin, Z. Wang, C. Fang, and J. Luo, “Image captioning with semantic attention,” in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4651–4659, 2016.
- [135] C. Harris and M. Stephens, “A combined corner and edge detector,” in *Proc. Alvey Vision Conference*, pp. 147–151, 1988.
- [136] R. O. Duda and P. E. Hart, “Use of the Hough transformation to detect lines and curves in pictures,” *Communications of the ACM*, vol. 15, pp. 11–15, 1971.
- [137] B. Benligiray, C. Topal, and C. Akinlar, “STag: A Stable Fiducial Marker System,” *Image and Vision Computing*, 2019.
- [138] D. Marr and E. Hildreth, “Theory of edge detection,” *Proc. Royal Society B*, vol. 207, no. 1167, pp. 187–217, 1980.
- [139] C. Topal and C. Akinlar, “Edge drawing: A combined real-time edge and segment detector,” *Journal of Visual Communication and Image Representation*, vol. 23, no. 6, pp. 862–872, 2012.
- [140] F. Aràndiga, A. Cohen, R. Donat, and B. Matei, “Edge detection insensitive to changes of illumination in the image,” *Image and Vision Computing*, vol. 28, no. 4, pp. 553–562, 2010.
- [141] H. Noh, S. Hong, and B. Han, “Learning deconvolution network for semantic segmentation,” in *Proc. IEEE International Conference on Computer Vision*, pp. 1520–1528, 2015.
- [142] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask R-CNN,” in *Proc. IEEE International Conference on Computer Vision*, pp. 2980–2988, 2017.
- [143] T. A. Le, A. G. Baydin, R. Zinkov, and F. Wood, “Using synthetic data to train neural networks is model-based reasoning,” in *Proc. International Joint Conference on Neural Networks*, pp. 3514–3521, 2017.
- [144] A. Atapour-Abarghouei and T. P. Breckon, “Real-time monocular depth estimation using synthetic data with domain adaptation via image style transfer,” in *Proc.*

- IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2800–2810, 2018.
- [145] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, “Reading text in the wild with convolutional neural networks,” *International Journal of Computer Vision*, vol. 116, no. 1, pp. 1–20, 2016.
- [146] A. Gupta, A. Vedaldi, and A. Zisserman, “Synthetic data for text localisation in natural images,” in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2315–2324, 2016.
- [147] J. Matas, O. Chum, M. Urban, and T. Pajdla, “Robust wide-baseline stereo from maximally stable extremal regions,” *Image and Vision Computing*, vol. 22, no. 10, pp. 761–767, 2004.
- [148] P. Bézier, *The Mathematical Basis of the UNISURF CAD System*. Butterworth, 1986.
- [149] B. Benligiray and Ö. N. Gerek, “Self-supervised pretraining by ranking spatial coherency,” *Manuscript*, 2019.
- [150] Ö. E. Yetgin and Ö. N. Gerek, “A comparison of corner and saliency detection methods for power line detection,” in *Proc. IEEE Int. Artificial Intell. and Data Process. Symp.*, 2017.
- [151] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *Proc. Int. Conf. Learning Representations (ICLR)*, 2015.
- [152] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, “Synthetic data and artificial neural networks for natural scene text recognition,” *arXiv preprint, arXiv:1406.2227*, 2014.
- [153] X. Peng, B. Sun, K. Ali, and K. Saenko, “Learning deep object detectors from 3D models,” in *Proc. IEEE Int. Conf. Comput. Vision (ICCV)*, pp. 1278–1286, 2015.
- [154] P. Krähenbühl, C. Doersch, J. Donahue, and T. Darrell, “Data-dependent initializations of convolutional neural networks,” in *Proc. Int. Conf. Learning Representations (ICLR)*, 2016.

- [155] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, “Large-scale video classification with convolutional neural networks,” in *Proc. IEEE Conf. Comput. Vision and Pattern Recognition (CVPR)*, pp. 1725–1732, 2014.
- [156] C. Doersch, A. Gupta, and A. A. Efros, “Unsupervised visual representation learning by context prediction,” in *Proc. IEEE Int. Conf. Comput. Vision (ICCV)*, pp. 1422–1430, 2015.
- [157] M. Noroozi and P. Favaro, “Unsupervised learning of visual representations by solving jigsaw puzzles,” in *Proc. European Conf. Comput. Vision (ECCV)*, pp. 69–84, 2016.
- [158] T. N. Mundhenk, D. Ho, and B. Y. Chen, “Improvements to context based self-supervised learning,” in *Proc. IEEE Conf. Comput. Vision and Pattern Recognition (CVPR)*, 2018.
- [159] R. S. Cruz, B. Fernando, A. Cherian, and S. Gould, “DeepPermNet: Visual permutation learning,” in *Proc. IEEE Conf. Comput. Vision and Pattern Recognition (CVPR)*, 2017.
- [160] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, “Context encoders: Feature learning by inpainting,” in *Proc. IEEE Conf. Comput. Vision and Pattern Recognition (CVPR)*, pp. 2536–2544, 2016.
- [161] R. Zhang, P. Isola, and A. A. Efros, “Colorful image colorization,” in *Proc. European Conf. Comput. Vision (ECCV)*, pp. 649–666, 2016.
- [162] G. Larsson, M. Maire, and G. Shakhnarovich, “Colorization as a proxy task for visual understanding,” in *Proc. IEEE Conf. Comput. Vision and Pattern Recognition (CVPR)*, 2017.
- [163] D. Kim, D. Cho, D. Yoo, and I. S. Kweon, “Learning image representations by completing damaged jigsaw puzzles,” in *Proc. IEEE Winter Conf. Applications of Comput. Vision*, 2018.



- [164] X. Wang and A. Gupta, “Unsupervised learning of visual representations using videos,” in *Proc. IEEE Int. Conf. Comput. Vision (ICCV)*, 2015.
- [165] I. Misra, C. L. Zitnick, and M. Hebert, “Shuffle and learn: Unsupervised learning using temporal order verification,” in *Proc. European Conf. Comput. Vision (ECCV)*, pp. 527–544, 2016.
- [166] H.-Y. Lee, J.-B. Huang, M. Singh, and M.-H. Yang, “Unsupervised representation learning by sorting sequences,” in *Proc. IEEE Int. Conf. Comput. Vision (ICCV)*, pp. 667–676, 2017.
- [167] A. Dosovitskiy, J. T. Springenberg, M. Riedmiller, and T. Brox, “Discriminative unsupervised feature learning with convolutional neural networks,” in *Proc. Advances in Neural Inform. Proc. Systems (NIPS)*, pp. 766–774, 2014.
- [168] P. Agrawal, J. Carreira, and J. Malik, “Learning to see by moving,” in *Proc. IEEE Int. Conf. Comput. Vision (ICCV)*, pp. 37–45, 2015.
- [169] M. Steiger, J. Bernard, S. Thum, S. Mittelstädt, M. Hutter, D. Keim, and J. Kohlhammer, “Explorative analysis of 2D color maps,” in *Proc. Int. Conf. Comput. Graph., Visualization and Comput. Vision*, pp. 151–160, 2015.
- [170] S. Chopra, R. Hadsell, and Y. LeCun, “Learning a similarity metric discriminatively, with application to face verification,” in *Proc. IEEE Conf. Comput. Vision and Pattern Recognition (CVPR)*, pp. 539–546, 2005.
- [171] H. Yun, P. Raman, and S. Vishwanathan, “Ranking via robust binary classification,” in *Proc. Advances in Neural Inform. Proc. Systems (NIPS)*, pp. 2582–2590, 2014.

## CV

Name: Burak Benligiray  
Foreign Language: English  
Birth date and place: 20/10/1989, Eskisehir  
E-mail: burakbenligiray@eskisehir.edu.tr

### Education

- PhD (2014–2019), Eskisehir Technical University, Department of Electrical and Electronics Engineering
- MSc (2011–2014), Anadolu University, Department of Electrical and Electronics Engineering
- BSc (2006–2011), Middle East Technical University, Department of Electrical and Electronics Engineering

### Employment

- Research Assistant (2018–), Eskisehir Technical University, Department of Electrical and Electronics Engineering
- Research Assistant (2014–2018), Anadolu University, Department of Electrical and Electronics Engineering

### Publications

- B. Benligiray, C. Topal, and C. Akinlar, “STag: A stable fiducial marker system,” *Image and Vision Computing*, vol. 89, pp. 158–169, 2019
- B. Benligiray, D. Connor, A. Tate, *et al.*, “Honeycomb: An ecosystem hub for decentralized oracle networks,” *White paper*, 2019. [Online]. Available: <https://www.clcg.io/whitepaper/>
- H. Cevikalp, B. Benligiray, and O. N. Gerek, “Semi-supervised robust deep neural networks for multi-label classification,” in *Proc. IEEE Conf. Computer Vision and Pattern Recognition Workshops*, 2019
- H. Saribas, B. Uzun, B. Benligiray, *et al.*, “A hybrid method for tracking of objects by UAVs,” in *Proc. IEEE Conf. Computer Vision and Pattern Recognition Workshops*, 2019
- B. Benligiray and Ö. N. Gerek, “Deep gradient operator,” in *Proc. Int. Conf. Advanced Technologies*, 2019
- O. E. Yetgin, B. Benligiray, and O. N. Gerek, “Power line recognition from aerial images with deep learning,” *IEEE Trans. Aerospace and Electron. Syst.*, 2018

- B. Benligiray and O. N. Gerek, “Visualization of power lines recognized in aerial images using deep learning,” in *Proc. IEEE Signal Process. and Commun. Appl. Conf. (SIU)*, 2018
- B. Benligiray, “Correcting writing errors in Turkish with a character-level neural language model,” in *Proc. IEEE Signal Process. and Commun. Appl. Conf. (SIU)*, 2018
- C. Gacav, B. Benligiray, K. Özkan, *et al.*, “Facial expression recognition with FHOG features,” in *Proc. IEEE Signal Process. and Commun. Appl. Conf. (SIU)*, 2018
- M. Ç. Uysal, T. Karapinar, B. Benligiray, *et al.*, “Dataset augmentation for accurate object detection,” in *Proc. IEEE Signal Process. and Commun. Appl. Conf. (SIU)*, 2018
- B. Benligiray, C. Topal, and C. Akinlar, “SliceType: Fast gaze typing with a merging keyboard,” *J. Multimodal User Interfaces*, 2018
- C. Gacav, B. Benligiray, and C. Topal, “Greedy search for descriptive spatial face features,” in *Proc. IEEE Int. Conf. Acoust., Speech and Signal Process. (ICASSP)*, 2017, pp. 1497–1501
- H. I. Cakir, B. Benligiray, and C. Topal, “Combining feature-based and model-based approaches for robust ellipse detection,” in *Proc. Eur. Signal Process. Conf. (EUSIPCO)*, 2016
- B. Benligiray and C. Topal, “Blind rectification of radial distortion by line straightness,” in *Proc. Eur. Signal Process. Conf. (EUSIPCO)*, 2016
- C. Gacav, B. Benligiray, and C. Topal, “Sequential forward feature selection for facial expression recognition,” in *Proc. IEEE Signal Process. and Commun. Appl. Conf. (SIU)*, 2016
- B. Benligiray and H. C. Akakin, “HEp-2 cell classification using a deep neural network trained for natural image classification,” in *Proc. IEEE Signal Process. and Commun. Appl. Conf. (SIU)*, 2016
- N. Kazak, M. Koc, B. Benligiray, *et al.*, “A comparison of classification methods for local binary patterns,” in *Proc. IEEE Signal Process. and Commun. Appl. Conf. (SIU)*, 2016
- B. Benligiray and C. Topal, “Lens distortion rectification using triangulation based interpolation,” in *Proc. Int. Symp. on Visual Computing (ISVC)*, 2015
- B. Benligiray, H. I. Cakir, C. Topal, *et al.*, “Counting Turkish coins with a calibrated camera,” in *Proc. Int. Conf. Image Anal. and Process. (ICIAP)*, 2015
- B. Benligiray, H. I. Cakir, C. Topal, *et al.*, “Coin recognition based on geometric features,” in *Proc. IEEE Signal Process. and Commun. Appl. Conf. (SIU)*, 2015

- B. Benligiray, “A line detection based fiducial marker detection algorithm,” Master’s thesis, Anadolu University, Department of Electrical and Electronics Engineering, 2014
- C. Topal, K. Özkan, B. Benligiray, *et al.*, “A robust CSS corner detector based on the turning angle curvature of image gradients,” in *Proc. IEEE Int. Conf. Acoust., Speech and Signal Process. (ICASSP)*, 2013
- B. Benligiray, C. Topal, C. Akinlar, *et al.*, “Quantification of projective distortion for fiducial markers,” in *Proc. IEEE Signal Process. and Commun. Appl. Conf. (SIU)*, 2013
- B. Benligiray, C. Topal, and C. Akinlar, “Realtime circular object detection based on the statistical compactness estimation,” in *Proc. IEEE Signal Process. and Commun. Appl. Conf. (SIU)*, 2012
- B. Benligiray, C. Topal, and C. Akinlar, “Video-based lane detection using a fast vanishing point estimation method,” in *Proc. IEEE Int. Symp. on Multimedia (ISM)*, 2012
- C. Topal, B. Benligiray, and C. Akinlar, “On the efficiency issues of virtual keyboard design,” in *Proc. IEEE Int. Conf. Virtual Environments Human-Comput. Interfaces and Measurement Syst. (VECIMS)*, 2012