**DOKUZ EYLÜL UNIVERSITY**
**GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES**

# OBJECT RECOGNITION FROM DEPTH CAMERA IMAGES

**by**

**Mert ŞEN**

**March, 2020**

**İZMİR**

# OBJECT RECOGNITION FROM DEPTH CAMERA IMAGES

**A Thesis Submitted to the**
**Graduate School of Natural and Applied Sciences of Dokuz Eylül University**
**In Partial Fulfillment of the Requirements for the Degree of Master of Science in Electrical and Electronics Engineering Program**

**by**

**Mert ŞEN**

**March, 2020**
**İZMİR**

**M.Sc THESIS EXAMINATION RESULT FORM**

We have read the thesis entitled **"OBJECT RECOGNITION FROM DEPTH CAMERA IMAGES"** completed by **MERT ŞEN** under supervision of **ASST. PROF. DR. HATİCE DOĞAN** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Hatice DOĞAN

Supervisor

Assoc. Prof. Dr. Gülseser K Demir

(Jury Member)

Asst. Prof. Dr. Nalan Özkurt

(Jury Member)

Prof.Dr. Kadriye ERTEKİN

Director
Graduate School of Natural and Applied Sciences

**ACKNOWLEDGEMENTS**

I am grateful to my advisor, Asst. Prof. Dr. Hatice Doğan for her insight, advice and devotion of her precious time throughout the thesis project.

I would like to thank my family for all their support and patience in my life. I am specially very thankful to my mother, who has played a crucial role in all my success.

I would also like to thank my friends, Berk AĞIN, İbrahim DEMİR and Salih ÜNSAL for all their friendship, help and patience during my study.

Mert ŞEN

## OBJECT RECOGNITION FROM DEPTH CAMERA IMAGES


## ABSTRACT

Object recognition from RGB-D images that provide additional depth information is very important task in many real world robotics and computer vision applications. The Convolutional Neural Networks (CNNs) have widely used in numerous applications especially RGB-D object recognition. However, CNNs have several restrictions even though they have demonstrated outstanding performance on object recognition. Pooling layer of CNNs causes to information loss in the stage of feature extraction. In addition to this, CNN is very sensitive to environmental factors such as rotation and light intensity. Capsule networks proposed by Hinton have been developed to avoid from these problems. In the thesis, the performances of the Capsule networks are investigated on the RGB-D dataset. Also a two-layer hierarchical structure is proposed in which the depth images are used in the first layer and RGB images are used in the second layer. Two different hierarchical structures that consist of CNN and capsule networks are designed. The performances of the hierarchical CNN and capsule networks are evaluated on the Washington RGB-D dataset. According to the simulation results, the best performance has been achieved with hierarchical CNN.



**Keywords:** Capsule network, hierarchical cnn, rgb-d image, object recognition

## DERİNLİK KAMERASI GÖRÜNTÜLERİNDEN NESNE TANIMA

## ÖZ

Derinlik bilgisini sağlayan RGB-D görüntülerinden nesne tanıma, robotik ve bilgisayarlı görü gibi uygulama alanlarında çok önemli bir görevdir. Evrişimli sinir ağları, özellikle RGB-D nesne tanıma uygulamalarında yaygın olarak kullanılmaktadır. Evrişimli sinir ağları, nesne tanımada üstün performans göstermesine rağmen, belirli kısıtlara sahiptir. Evrişimli sinir ağlarında kullanılan örnekleme katmanı, öznitelik çıkarma aşamasında bilgi kaybına neden olmaktadır. Buna ek olarak, evrişimli sinir ağları, dönme ve ışık şiddeti gibi çevresel etkenlere karşı çok hassastır. Hinton tarafından önerilen kapsül ağları, bu tarz problemlerden kaçınmak için geliştirilmiştir. Bu tezde, kapsül ağlarının, RGB-D veri seti üzerindeki performansları incelenmiştir. Ayrıca, ilk katmanında derinlik görüntüleri, ikinci katmanında RGB görüntüleri olan iki katmanlı hiyerarşik bir yapı önerilmiştir. Evrişimli sinir ağları ve kapsül ağlarından oluşan iki farklı hiyerarşik yapı tasarlanmıştır. Evrişimli sinir ağları ve kapsül ağlarının performansları, Washington RGB-D veri seti üzerinde değerlendirilmiştir. Simülasyon sonuçlarına göre, en iyi performans hiyerarşik evrişimli sinir ağı ile elde edilmiştir.

**Anahtar kelimeler:** Kapsül ağı, hiyerarşik cnn, rgb-d görüntü, nesne tanıma

**CONTENTS**

**Page**

## LIST OF FIGURES

**Page**

## LIST OF TABLES

**Page**

# CHAPTER ONE

# INTRODUCTION

Object recognition is a task used to identify any objects in images or videos. The fundamental idea behind the object recognition is to extract features which are widely known as shape, size and color and apply to the model database in order to verify the object. Object recognition allows the machines to learn, identify and improve the visual perception, which is the importance of this technique. In recent years, object recognition from RGB-D images becomes hot topic since it plays crucial role in many real world robotics and machine vision applications.

An RGB-D image is a combination of RGB and corresponding grayscale image. Depth image is a single channel image whose pixels give distances between the image plane and the corresponding object in RGB image. Usage of RGB and grayscale depth images simultaneously makes the performance of object recognition better.

In recent years, many cameras that generate RGB and grayscale depth images have been developed by means of advancing technology. In parallel with these improvements, many researches including Convolutional Neural Networks which are widely used and have proven success in image processing have been done progressively. In the literature, different neural networks have been used to extract features from the images and these features have been fused in various ways. Cheng et al., 2016, proposed to use Semi-Supervised learning and deep CNNs to recognize the objects belonging to Washington RGB-D dataset. Two deep CNNs have been implemented for RGB and depth images separately and only 5% labelled images of the dataset have been used for training process. At the end of the training, all RGB and depth features extracted by the networks were applied to a classifier separately. The outputs of these classifiers contain the labels of images which are applied to the inputs of both networks. The extracted features are used to create fusion classifier. Eitel, Springenbeg, Spinello, Riedmiller & Burgard (2015) realized Supervised Learning technique on Washington RGB-D dataset. They have used two pre-trained deep CNNs in order to extract features from RGB and depth images. Afterwards,

all extracted features were used as inputs of a fusion classifier. The key point of the study is to encode the grayscale depth images as RGB image by the help of colourization methods. Besides, in order to use transfer learning, all images found in the dataset have been resized using a new method proposed by (Eitel et al., 2015). Schwarz, Schulz & Behnke (2015) develop a new appoach on Washington RGB-D dataset. They have worked on categorization of all objects, instance recognition and pose estimation. The whole images are masked in order to extract foreground. Furthermore, they have used two different pretrained CNNs to obtain features, and they are fed to a Support Vector Machine (SVM) to evaluate the object category, type of instance and pose. In the work of Cheng, Zhao, Huang & Tan (2014) use two different deep Convolutional neural networks as performed in (Cheng et al., 2016). Both networks are configured as semi-supervised and unsupervised learning. One of the main differences between the studies is the usage of multiple Recursive Neural Networks (RNNs) for the learning of the features from RGB and depth images. CNNs have learned the low level features while RNNs have obtained the high level features. Using two different network topologies for feature extraction makes the training process faster.

A novel method offered by Rahman, Tan, Xue & Lu (2017) states that three independent CNNs have been designed for feature learning. One of these networks is used for RGB images, the others are operated for depth images. In this technique, both networks encode the depth images as RGB image and use surface normals of that images. GoogleNet is used for feature learning of RGB images while CaffeNet is preferred for depth images due to its straightforward structure. In the study of Mocanu & Clapon (2018), object detection task has been performed on Sun RGB-D dataset via faster Recursive Convolutional Neural Network (RCNN) with a Region Proposal Net (RPN). For RGB and depth images, VGG has been used independently and then all extracted features are concatenated. These features are the inputs of both RPN and fast RCNN. The RPN contains convolutional and ReLU activation layers. This network is used to determine region of the object and estimate the object bounding. Besides, fast RCNN includes max pooling layer used for determining Region of Interest (RoI). Wang, Lu, Chen & Wu (2015) utilizes a fine-tuned convolutional neural network with images from ImageNet. They have used two different Caffe models to learn features from Washington RGB-D

dataset, extracted and concatenated features are applied to a linear SVM for classification. All missing depth values are filled by computing the mean of depth values in $5 \times 5$ region before the stage of training. In the work of Wei, Zhiguo, Yang & Zhiwen (2015), RGB-D object recognition has been performed using both convolutional neural network and Fisher vector which measures the similarity between objects. The features of RGB images have been extracted by VGG16 or VGG19, while the features of depth images are obtained by means of Fisher kernel. Afterwards, all features are combined in order to form the inputs of SVM used for classification. The critical point of the study is that RGB and depth images of Washington RGB-D dataset are processed without segmentation masks.

Another way to extract features is to combine various network topologies. In the study of Bui, Lech, Cheng, Neville & Burnett (2016), AlexNet, which is the most popular and widely used architecture, and a RNN are connected in cascade form. The classification is performed by RNN instead of fully-connected layers after AlexNet has extracted all features from the images found in Washington RGB-D dataset. On the other hand, feature extraction process do not need additional training time. In other work that uses Washington RGB-D dataset, Lai, Bo, Liefeng & Fox (2011a) propose Sparse Distance Learning to combine RGB and depth images for object recognition. The principle of Sparse Distance Learning is similar to artificial neural networks since the distance between an unknown image and a known set of images has been computed. This approach is very useful for instance recognition.

On the other hand, 2D and 3D Convolutional neural networks have been developed to perform object recognition. Zia, Yüksel, Yüret & Yemez (2017) have designed a hybrid 2D/3D CNNs to learn features from RGB-D images. 3D architecture encodes spatial and color information at the same time. For all RGB images, features have been obtained by means of pre-trained VGGNet and they are used to train a SVM. On the other hand, a three dimesional voxel whose height and width are the as original image is created for depth images. All depth information have been located on the voxel with respect to the distance computed using maximum and minimum depth values in the image. Furthermore, missing depth values are filled with interpolation method. Maturana & Scherer (2015) have implemented a 3D CNN, which is called as VoxNet. The VoxNet transforms the point

clouds coming from the RGB-D cameras into occupancy grid. After that, all images have been applied to VoxNet and fully connected layers for classification. Caglayan & Can (2018) exploits a volumetric demonstration with a 3D CNN. Due to noisy data coming from the sensor, they have applied denoising techniques. After that volumetric representations are generated from raw depth images. 3D CNN with input volume representation of $32 \times 32 \times 32$ has two convolutional layers, pooling and fully connected layers in order to make the generalization ability of the network better. In the training stage, both randomly mirroring and shifting have been performed for data augmentation. Nishi, Kurogi & Matsuo (2017) have investigated to classify the objects with respect to their size. The pixels of RGB images are transformed in order to compute the center of the object using appropriate depth pixels in 3D. In the topology of CNN, there are many convolution and pooling layers to ensure invariance to scaling. They have used four categories such as apple, bell pepper, orange and tomato from Washington RGB-D dataset. In addition to this, each object is enlarged by a finite ratio which changes from 0.8 to 1.2 and rotated between -10 and 10 degrees around z-axis.

The major problem dealing with large dataset and neural networks is directly related with the computation power. Uetz & Behnke (2009) overcome the problem by using NVIDIA CUDA (Compute Unified Device Architecture) technology to operate Locally-connected Neural Pyramid (LCNP) for extracting features. Unlike the traditional CNN structure, they have used Hierarchical structure and parallel computing allows the network to process images different cores. One of the crucial aspect of LCNP is that the network has no weight sharing among neurons. Therefore, object recognition task can be performed faster and efficiently. In the study carried out by Asif, Bennamoun & Sohel (2017), Washington RGB-D recognition and scene images have been analyzed via pre-trained CNNs. The network has learned the features from the point clouds of the objects. All features including RGB color, color gradients, surface normals and their orientations are used for robotic grasping.

In the literature, different versions of CNNs have been proposed for object recognition from RGB and depth images. Even though they have presented high performance on RGB-D images, CNNs have several disadvantages. CNNs require generally all points

of view of an object for recognition precisely and they are also sensitive to light intensity, occlusion and rotation. Furthermore, several layers of the network loses the spatial information in the step of feature extraction. To cope with these troubles, *Capsule Networks* are proposed by Hinton (Sabour, 2017). A capsule is described as a small group of neurons that encodes the spatial information. Each capsule has an activity vector that represents the specific parameters of the object. The length of this vector is equal to the probability of the object if it is present in the image. In addition, orientation of the vector gives information about the pose of the object.

In this thesis, a hierarchical Convolutional neural network that consists of two layers is proposed for object recognition from RGB and depth images. In this architecture, all classes of the dataset have been classified with a CNN structure in order to determine the object classes recognized with the best accuracy. After that, first layer of the hierarchical structure contains the object classes whose accuracies are higher than 90%. The second layer covers the remaining object classes. The depth images are used in the first layer and the RGB images are used in the second layer. The aim of the hierarchical classification is to decompose the classes to be classified easily and design a different classifier for the object classes which is hard to classify. This technique is also performed with capsule networks. The performance of the networks are evaluated on challenging Washington RGB-D dataset. The compelling issues of this dataset can be summarized as intraclass and interclass variations in Washington RGB-D dataset, various shape of cropped versions of both RGB and depth images, missing depth pixels due to sensor limitations, deformations and occlusion. Moreover, the performances of the capsule networks have been investigated on Washington RGB-D dataset.

This thesis is organized as follows: The general structure and limitations of CNNs are studied in Chapter II. The topology of Capsule Networks, dynamic routing algorithm are explained in Chapter III. Washington RGB-D object recognition dataset, preprocessing steps and all simulation results are presented in Chapter IV and conclusions are given in Chapter V.

# CHAPTER TWO

# CONVOLUTIONAL NETS

## 2.1 Convolutional Neural Networks

Convolutional neural network is a feed-forward, multi-layer deep neural network to analyze data. This type of networks show better performance than the classical methods especially on object recognition. Convolutional neural networks have been originated from a mathematical operation known as *convolution* defined as,

$$h[n] = \sum_{k=-\infty}^{\infty} f(k)g(n-k) \tag{2.1}$$

In the deep learning terminology, Eq.(2.1) states that $h[n]$ is the output of the convolutional layer, $f(k)$ is the input data to be applied to neural network and $g(n-k)$ is the filter to extract features. CNNs can be used with different learning methods. In *supervised learning*, all input and output data are always known. Algorithm predicts the output data iteratively using training data. The learning process continues until desired performance is achieved. In *unsupervised learning*, the network has unlabelled data. The learning process highly depends on the distribution of the data. In *semi-supervised learning*, small portion of the data is labelled. In the thesis, supervised learning has been performed. During the thesis, the following parameters must be carefully adjusted to improve the generalization performance of the network.

- *Epoch* is an integer number, which learning process operates on the set of training data. The value of this parameter can be selected with respect to the network complexity,

- *Batch* is a parameter which describes the number of data applied to the input of the network before updating of the internal weights of the network,

- *Learning Rate* is a hyper-parameter that defines the step size of an optimization algorithm to approach the minimum value of a loss function. The parameter affects the generalization performance of the network directly. It is no clear way to

determine the learning rate. However, large learning rates make the training phase unstable. The illustration between loss and learning rate can be seen below.

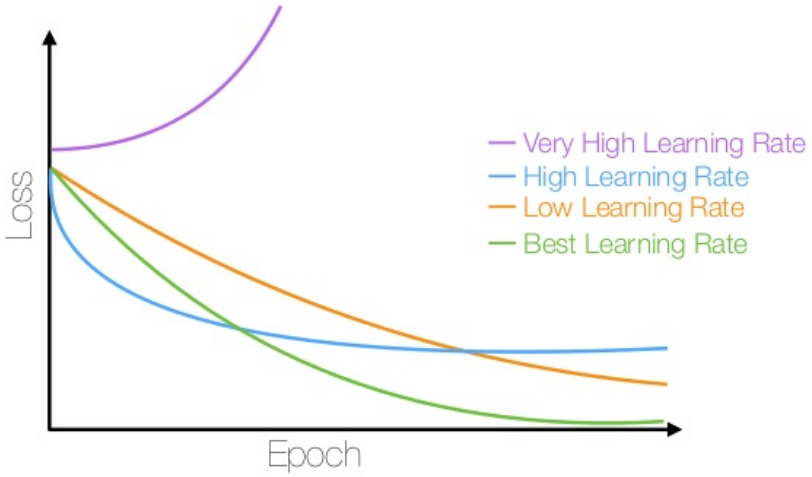



Figure 2.1 Relationship between learning rate and loss (Gonçalves, 2016)

A CNN consists of convolutional and activation layers, pooling layers and fully connected layers. The general structure of a CNN can be seen below. The operating principle of a CNN can be summarized as follows,

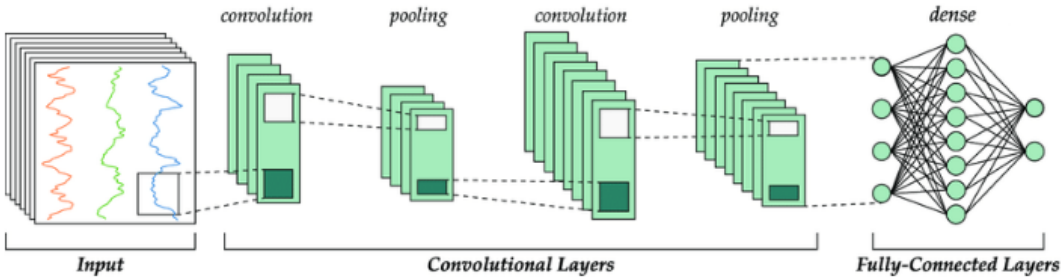



Figure 2.2 General overview of a CNN (Baldominos et al., 2018)

1. When an image is applied to the network, all these layers perform convolution operation for feature extraction,
2. All extracted features are transferred to activation and pooling layers,
3. Pooling layers reduce the dimension of the input data,
4. Down-sampled data are applied to fully-connected layers for training.

### 2.1.1 *Convolutional & Activation Layers*

A convolutional layer is applied to raw or pre-processed input data. The operation of the layer is to slide a kernel over the whole image. The simple operation of the layer can be seen in the following figure.

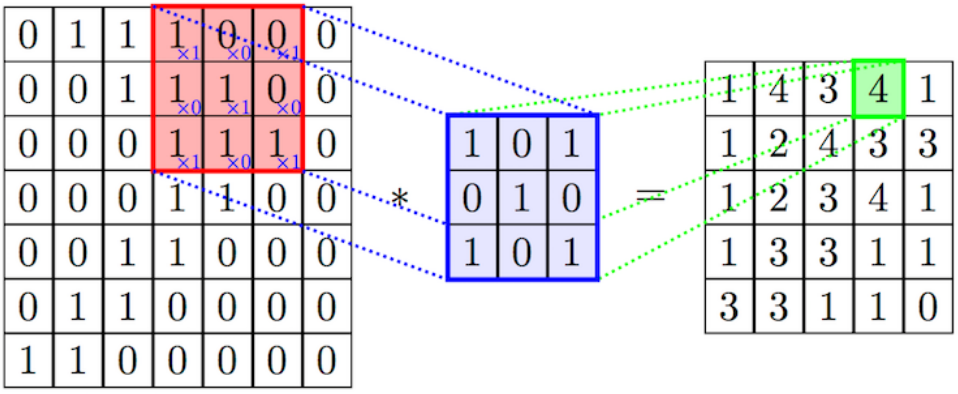



Figure 2.3 Convolution operation (Chatterjee, 2017)

The output of the convolutional layer can be computed as,

$$Output\ Shape = \left[\frac{W-K+2P}{S}\right]+1 \tag{2.2}$$

where W is the input shape, K is the shape of the kernel, P is the parameter used to determine whether the input data are zero-padded or not. Lastly, S is the stride number which determines the step size of the kernel (Chatterjee, 2017).

Activation layer is used to add non-linearity to the neural network. There are many different activation functions such as Sigmoid, Hyperbolic Tangent and ReLU (Rectified Linear Unit). Mathematically,

$$f_{sigmoid}(x) = \frac{1}{1+e^{-x}} \tag{2.3}$$

$$f_{tanh}(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{2.4}$$

$$f_{ReLU}(x) = max(0,x) \tag{2.5}$$

This layer solves the *vanishing gradient problem* which emerges from the selection of the activation. The problem states that the output of the network produces a small change even if big change occurs in the input of the network. In order to eliminate the very small values of the gradient, ReLU is used as an activation function in the thesis. Besides, ReLU makes the training process of the network faster.

### *2.1.2 Pooling Layer*

Pooling layer performs downsampling operation to produce a summary of all sub-region in an image. The layer reduces the spatial information of the input data to avoid from computational cost. Average or Max Pooling in the following figure can be widely used in the neural networks (Balodi, 2019).

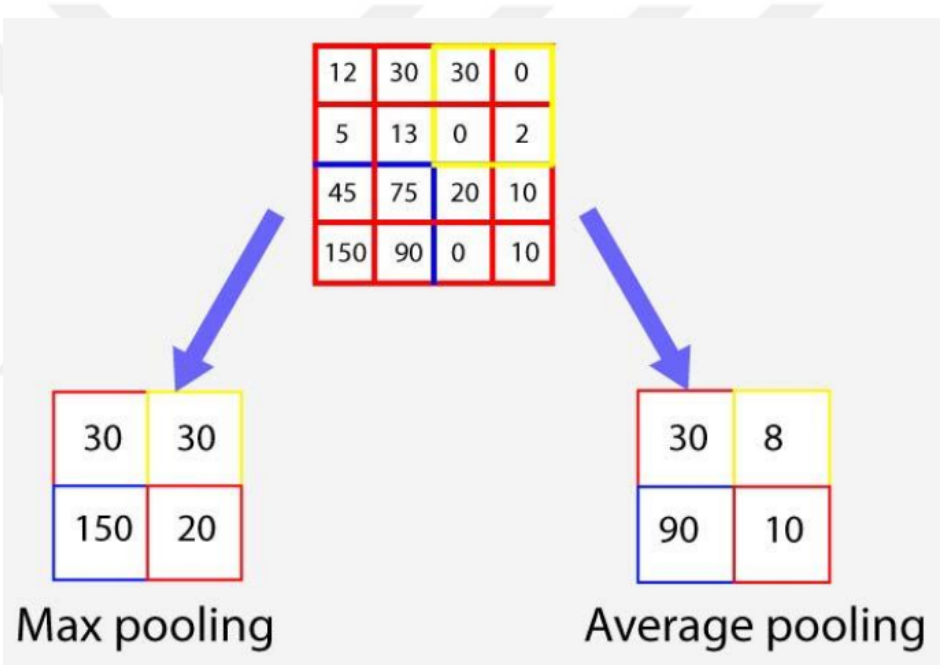



Figure 2.4 Popular pooling techniques (Balodi, 2019)

Furthermore, pooling layers also control the overfitting problem. Overfitting is an undesired phenomena in deep learning. Overfitting occurs when a neural network memorizes all training data including noise. Thus, the generalization ability of the network is influenced negatively. In addition to this, one of the most common situation is known as underfitting. In this case, the neural network has bad training performance and it can not generalize both training and test data. Graphical representation of both conditions is given below.

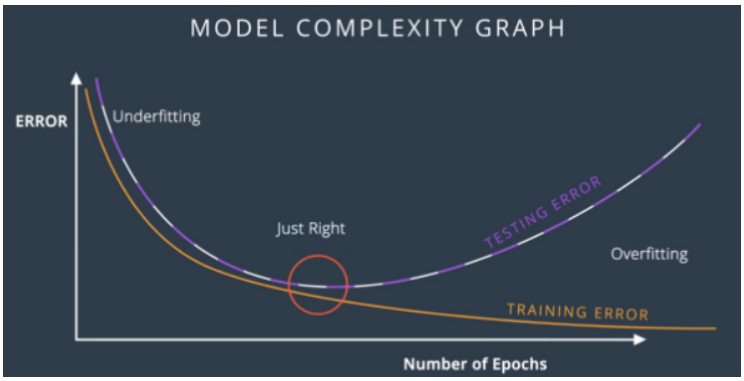



Figure 2.5 Overfitting and underfitting (Patel, 2019)

To prevent overfitting, the amount of training data can be increased or regularization and dropout methods are applied to the network for reducing model complexity. In addition to this, early stopping can be performed in training step. In order to cope with underfitting, number of features, epochs and the complexity of the model can be increased (Patel, 2019). In the thesis, Image Data Generator defined in Keras library is used to evaluate the classification performance of the network as data augmentation method. The well-known data augmentation methods are flipping, rotation, scaling, cropping and adding noise.

### *2.1.3 Fully Connected Layer*

This layer determines which features are mostly correspond to a particular class. In general, all features coming from the previous layers are applied to the input of MLP. An example of MLP can be seen in the following figure.

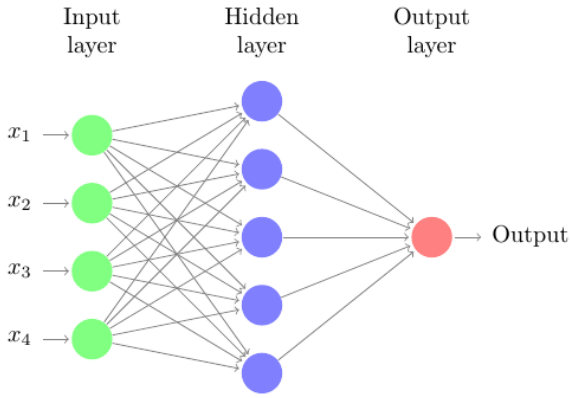



Figure 2.6 A fully connected MLP with three layers

## 2.2 Types of Convolutional Neural Networks

In the last decades, many different types of CNNs have been used to implement various tasks. Each CNN has different topology and number of parameters, which affects the performance of the neural network significantly. The popular topologies can be seen in Table 2.1.

Table 2.1 Number of model parameters of different convolutional nets

| Model | Model Parameter | Depth |
|---|---|---|
| Xception | 22.9M | 126 |
| VGG16 | 138.3M | 23 |
| VGG19 | 143.6M | 26 |
| InceptionV3 | 23.86M | 159 |
| MobileNet | 4.2M | 88 |
| ResNet50 | 25.6M | 168 |

The models in this table can be used for transfer learning which is an optimization technique to save time and to perform better. Transfer learning has two stages. In the pre-training stage, the network has been trained on a large dataset such as ImageNet, then all parameters of the network have been already learned. In the fine-tuning, a new dataset is applied to pre-trained network. If the dataset is very similar to one that used in pre-training, feature extraction can be performed with same weights. After that, it is better idea to train the fully-connected layers.

In the thesis, VGG16 has been used to perform the object recognition task and compare the classification performance to the performance of Capsule network.

## 2.3 VGG16 Architecture

VGG16 is a convolutional neural network created by Oxford University. The input shape of VGG16 is adjusted as $224 \times 224$ RGB image. In addition to this, the size of the kernel used in the convolutional layer is $3 \times 3$. VGG16 has three fully-connected layers and the first two layers have 4096 neurons. The neuron number of the last layer can be determined with respect to the object classes of the dataset. The general overview of VGG16 is depicted in the following figure.

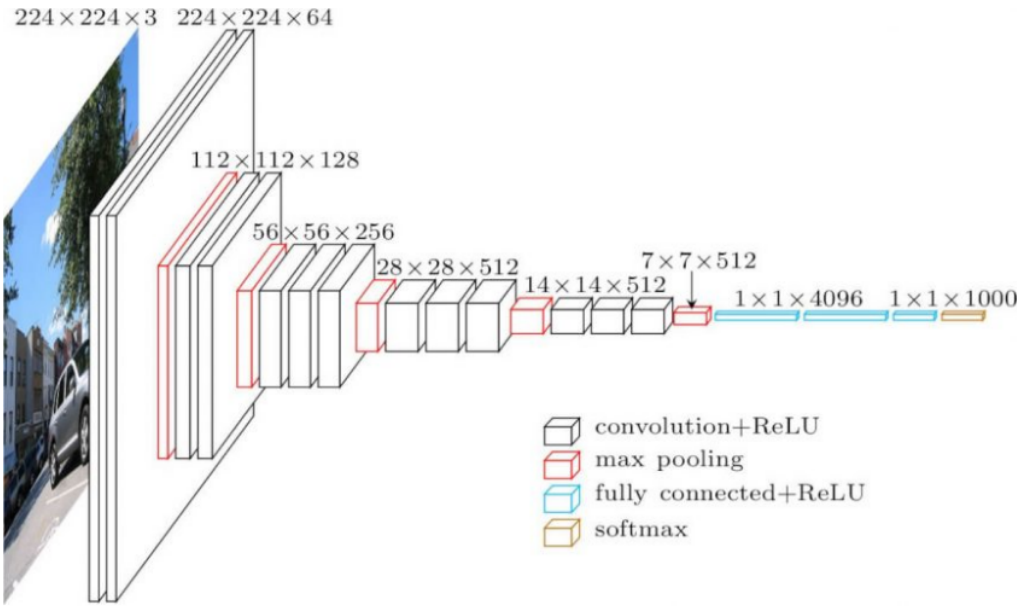



Figure 2.7 General structure of VGG16 (Thakur, 2019)

## 2.4 Limitations of Convolutional Neural Networks

Convolutional Neural Networks are very sensitive to the orientation of the object and light intensity in the environment. Simple and complex spatial relationship between object and environment can not be taken into the consideration. As seen in the following figure, light intensity on the different perspectives of an object in the image decreases the performance of the network (Pechyonkin, 2017). Therefore, more training data are required to improve the performance. However, this situation causes high computational cost, training time and powerful hardware.

Figure 2.8 Images with different orientations and light intensities (Pechyonkin, 2017)

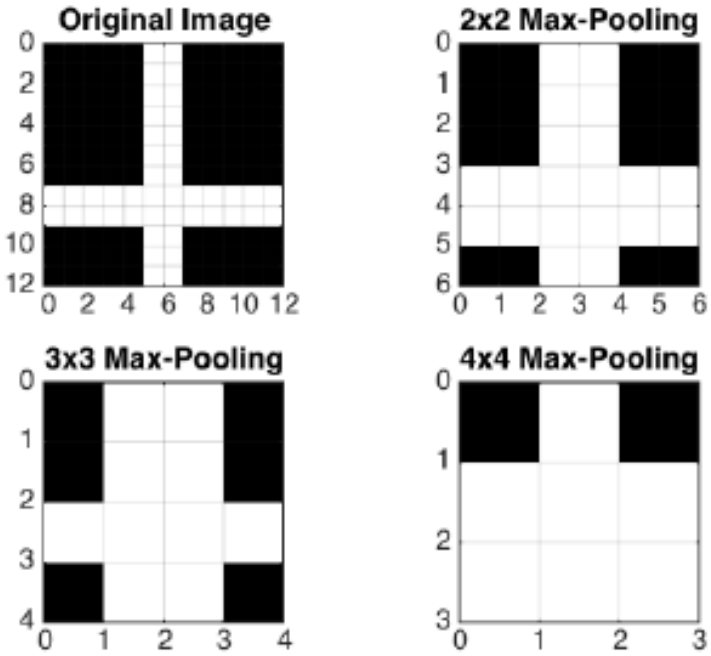



Figure 2.9 Disadvantage of a pooling layer (Thörnberg, 2015)

On the other hand, the major problem with CNNs is the pooling layers, since this layer loses crucial information in the image. Figure 2.9 represents that the information extracted from the image is reduced significantly when the dimension of the pooling layer is increased.

# CHAPTER THREE

# CAPSULE NETWORKS

## 3.1 Capsule Networks

Capsule networks show up to overcome the disadvantages of the convolutional neural networks. A capsule can be described as a small group of neurons. Input and output of a capsule are in the form of vector, which is opposed to artificial networks. Each capsule has an activity vector marked with red and blue in the following figure. Pose, deformation and velocity (also called as instantiation parameters) are indicated by this vector (Sabour, 2017).

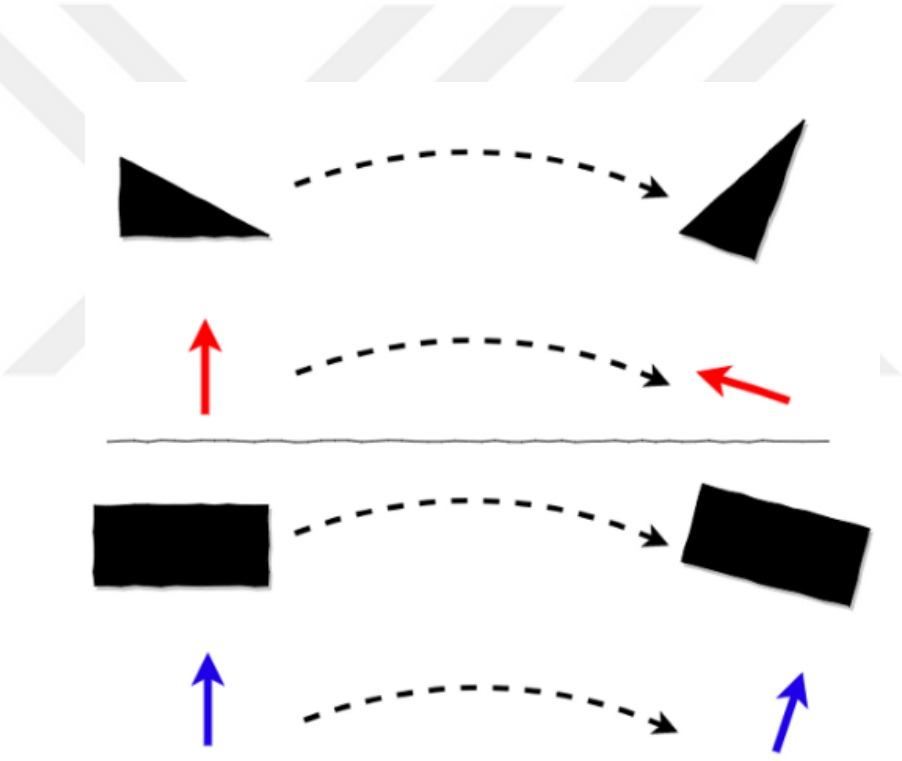


Figure 3.1 Activity vector of a capsule (Rimal, 2018)

The length of the activity vector corresponds to the probability which the object exists in the image and orientation of the vector points out the instantiation parameters. The following table represents the differences between a capsule and a traditional neuron. One of the crucial dissimilarities between these neurons is that capsule network performs affine transformation before weighted sum (Pechyonkin, 2017).

Table 3.1 Comparison of a capsule to a traditional neuron (Pechyonkin, 2017)

| Operation Type | Capsule | Traditional Neuron |
|---|---|---|
| Input Shape | $vector(\mathbf{u}_i)$ | $scalar(x_i)$ |
| Affine Transform | $\hat{\mathbf{u}}_{\mathbf{j}\|\mathbf{i}} = \mathbf{W}_{\mathbf{ij}}\mathbf{u}_{\mathbf{i}}$ | - |
| Weighting & Sum | $\mathbf{s_j} = \sum_i c_{ij}\hat{\mathbf{u}}_{\mathbf{j}\|\mathbf{i}}$ | $a_j = \sum_i w_i x_i$ |
| Non-linear Activation | $\mathbf{v_j} = \frac{\|\|\mathbf{s_j}\|\|^2}{1+\|\|\mathbf{s_j}\|\|^2}\frac{\mathbf{s_j}}{\|\|\mathbf{s_j}\|\|}$ | $h_j = f(a_j)$ |
| Output Shape | $vector(\mathbf{v}_j)$ | $scalar(h_j)$ |

## 3.2 Operating Principle of Capsule Network

The study offered by Hinton, Krizhevsky and Wang (2011) defines the main concept of the capsules by transforming auto-encoders. Each capsule learns how to recognize the objects under different circumstances such as light intensity, translational change. It is important to investigate operation of lower level capsule.

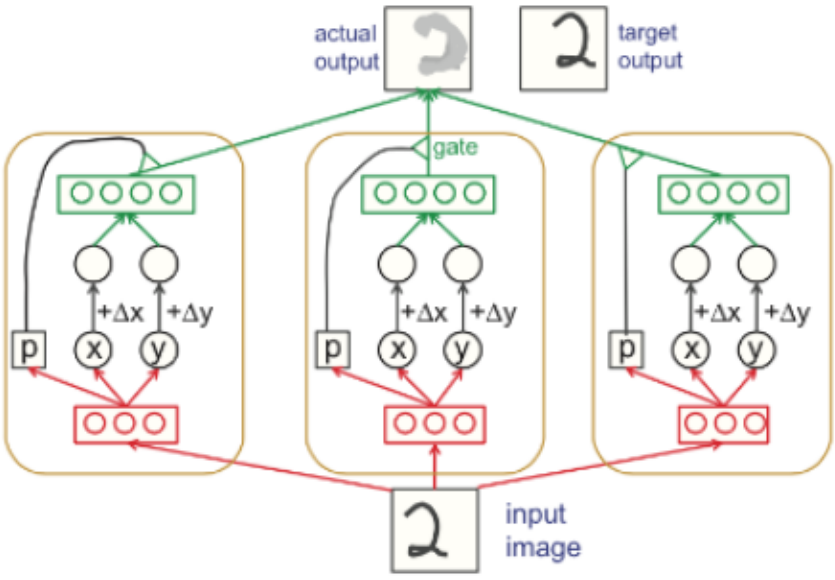



Figure 3.2 Basis of a capsule operation (Hinton et al., 2011)

The Figure 3.2 represents a deterministic and feed forward network with three capsules. Each capsule has three recognition units and four generation units. Recognition units can be considered as a hidden layer to calculate the probability of existence of the

object (p), position of the object (x and y). Generation units compute the the contribution of each capsule to an image to be transformed. This network contains different capsules interacting at the last layer for shifted image. Both an image and amount of desired shift (denoted as $\Delta x$ and $\Delta y$) have been used as inputs of the network and the network generates a shifted image. $x + \Delta x$ and $y + \Delta y$ are the inputs of generation units. If a capsule is not active, this means that there is no contribution to output image. When randomly-shifted input, output images and amount of shift are applied, capsules learn how to find the position of the object.

On the other hand, the pioneer study of internal operation of capsule networks performed by Sabour, Frosst and Hinton (2017) can be explained visually as follows,

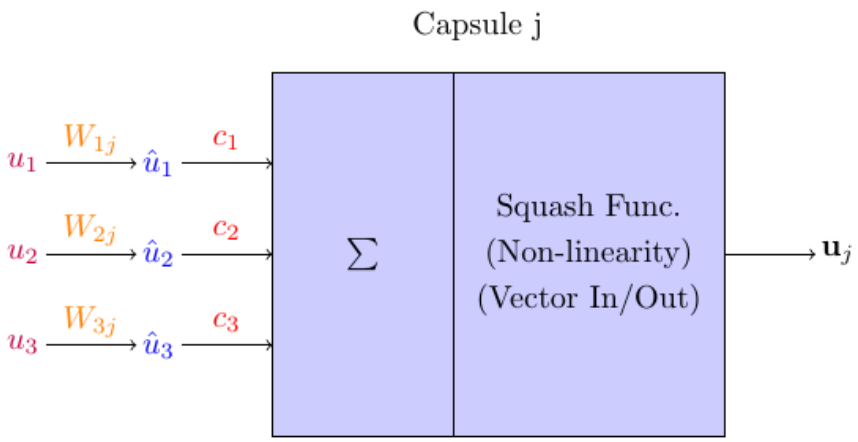



Figure 3.3 Internal operation of a capsule

- **Matrix Multiplication of Input Vectors :** The vectors $\mathbf{u_1}$, $\mathbf{u_2}$ and $\mathbf{u_3}$ coming from a lower level layer are considered as input vectors. Length of these vectors encodes the probability of detected objects whereas direction of the vectors encodes the internal state of the object. Then these vectors are multiplied by a weight matrix denoted as $\mathbf{W}$ in order to store important spatial and other relationships between lower level features and higher level features. In fact, the estimated position of the high level feature is found.

- **Scalar Weighting of Input Vectors :** In this case, lower level capsules send their outputs to higher level capsules. The critical point is to determine which input of the higher level capsule is connected to the lower level capsule. The weights $c_1$,

$c_2$ and $c_3$ are used for the connection between capsules and they are determined by using *routing algorithm*.

- **Sum of Weighted Input Vectors :** This step is the same as in a traditional neuron. Namely, it represents combination of vectors.

- **Squash: Vector-to-Vector Non-linearity :** This non-linear activation function takes an input as a vector and squashes it without changing its direction. After the squashing, the length can not exceed unity. To compute vector inputs and outputs of a capsule, it can be written mathematically as,

$$\mathbf{v_j} = \frac{||\mathbf{s_j}||^2}{1+||\mathbf{s_j}||^2} \frac{\mathbf{s_j}}{||\mathbf{s_j}||} \tag{3.1}$$

where $\mathbf{v_j}$ is the vector output of capsule $j$ and $\mathbf{s_j}$ is total input. The first term on the right-side of the Eq.(3.1) performs **additional squashing** and the second term applies unit scaling to input vector.

### *3.2.1 Routing-by-Agreement Technique*

In addition to Eq.(3.1), total input is given by,

$$\mathbf{s_j} = \sum_i \mathbf{c_{ij}} \hat{\mathbf{u}}_{\mathbf{j|i}}, \qquad \hat{\mathbf{u}}_{\mathbf{j|i}} = \mathbf{W_{ij}} \mathbf{u_i} \tag{3.2}$$

where $\hat{\mathbf{u}}_{\mathbf{j|i}}$ is the prediction vectors from capsules in the layer below, $\mathbf{u_i}$ is the output of a capsule in the layer below and $\mathbf{W_{ij}}$ is the weight matrix. The coupling coefficients $\mathbf{c_{ij}}$ are computed by the help of iterative dynamic routing process. The sum of the coefficients equals to one and they are determined by a routing softmax whose initial values $b_{ij}$ are the *log probabilities* that capsule $i$ should be coupled to capsule $j$. The coefficients can be found as,

$$c_{ij} = \frac{exp(b_{ij})}{\sum_k exp(b_{ik})} \tag{3.3}$$

Note that $c_{ij}$ is a non-negative scalar and for each lower level capsule i, the number of weights equals to the number of higher level capsules. On the other hand, **the agreement condition** is described as scalar product,

$$a_{ij} = \mathbf{v_j}.\hat{\mathbf{u}}_{\mathbf{j|i}} \tag{3.4}$$

### *3.2.2 Routing Algorithm*

The following figure shows the routing algorithm described by Hinton. Besides, operation of each step in the algorithm can be examined step by step (Pechyonkin, 2017).

**procedure** ROUTING($\hat{\boldsymbol{u}}_{j|i}, r, l$)
    for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$: $b_{ij} \leftarrow 0$.
    **for** $r$ iterations **do**
        for all capsule $i$ in layer $l$: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$
        for all capsule $j$ in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}$
        for all capsule $j$ in layer $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$
        for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
    **return** $\mathbf{v}_j$

Figure 3.4 Routing algorithm (Sabour et al., 2017)

- **Step 1 :** Algorithm takes the number of routing iteration (r), number of lower layer ($l$) and $\hat{\mathbf{u}}$ is the outputs of all capsules in that layer.
- **Step 2 :** At the beginning, $b_{ij}$ is set to 0. The temporary coefficient $b_{ij}$ is iteratively updated until the procedure is completed. Then, it is finally stored in $c_{ij}$.
- **Step 3 :** Repeat the dynamic routing algorithm with respect to the iteration number.
- **Step 4 :** Compute the values of the vector $c_i$ for all lower level capsules using Eq.(3.3). All coefficients $c_{ij}$ are equal in the first iteration since $b_{ij}$ = 0. It means that algorithm shows **maximum confusion and uncertainty.** In other words, the appropriate output of higher level capsule can not be determined by lower level capsule.
- **Step 5 :** After all computations of $c_{ij}$ for lower layer capsules, a linear combination of input vectors that are weighted by $c_{ij}$ is computed for higher level capsules. Then $\mathbf{s_j}$ is calculated by using Eq.(3.2).
- **Step 6 :** The vector computed in previous step is squashed for non-linearity. The direction of the vector is not changed and its length is not bigger than 1. Then, output vector $\mathbf{v_j}$ is produced using Eq.(3.1) for all higher level capsules.
- **Step 7 :** This step examines each input and updates the corresponding weight $b_{ij}$ for each higher level capsule $j$. The dot product looks at similarity between input

to the capsule and output from the capsule. This similarity is evaluated by the **dot product**.

### *3.2.3 Visual Representation of Updating Weights*

Assume that there are two higher level capsules whose outputs are denoted as $v_1$ and $v_2$. The red vectors represent input from one of the lower level capsules and the black vectors represent all the remaining inputs from other lower level capsules in the following figure.

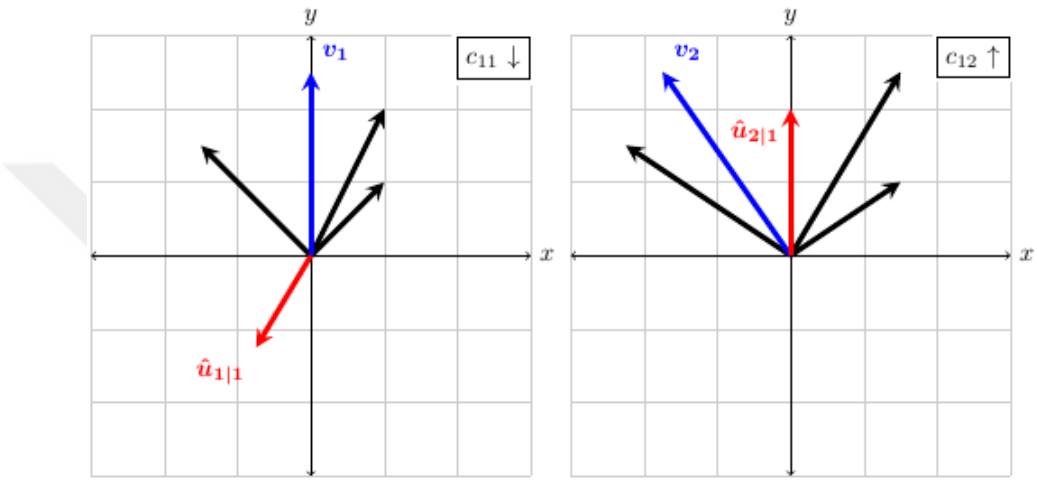



Figure 3.5 Similarity measure between input and output of capsules (Pechyonkin, 2017)

The left figure shows that $\hat{\mathbf{v}}_1$ and $\hat{u}_{1|1}$ are in opposite direction, which means that they are not similar. Then, dot product is negative and $c_{11}$ is decreased. The right figure represents that $\hat{\mathbf{v}}_2$ and $\hat{u}_{2|1}$ are in same direction, which means that they are similar. Then, $c_{12}$ is increased. The process is repeated for all higher level capsules.

## 3.3 CapsNet Architecture

Capsule network used in the thesis has a convolutional layer with $32 \times 32 \times 3$ input size for both RGB and depth images. Capsule networks have encoder and decoder parts as shown in Figure 3.6 and Figure 3.7.

In the encoder part, the convolution layer used for feature extraction has 256 filters, no padding and produces one-dimensional output. In addition to this, stride is selected as 1. Due to one dimensional output, there is no orientation. PrimaryCaps layer is another

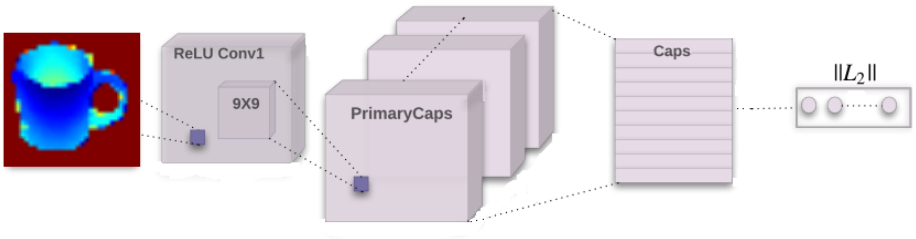



Figure 3.6 Encoder part of the capsule network (Sabour et al., 2017)

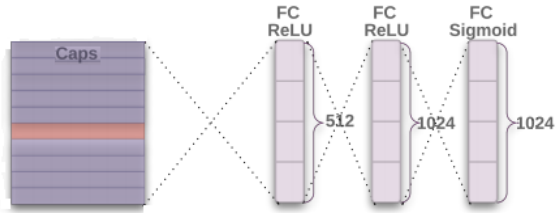



Figure 3.7 Decoder part of the capsule network (Sabour et al., 2017)

convolution layer that applies squashing. The number of primary capsule is selected as 16. Caps layer is used for classification and it calculates the classification losses. Note that total loss is the sum of individual capsule loss. The critical point is that the routing algorithm is performed between PrimaryCaps and Caps layers.

Decoder part uses the outputs of Caps layer to reconstruct the images. However, reconstruction of all images has not been studied in the thesis.

# CHAPTER FOUR

# APPLICATIONS & RESULTS

In this section, Washington RGB-D dataset and preprocessing steps are explained briefly. In the preprocessing step, a new method which is proposed by Eitel et al. (2015) has been used to resize the whole images in the dataset. In addition to this, interpolation technique proposed by Thörnberg (2015) is used to fill the missing depth pixels. Throughout the thesis, Python and its libraries (Keras and Tensorflow as backend) are used.

## 4.1 Washington RGB-D Dataset

A dataset is an organized set of information such as image, text or speech signals to be processed by a computer. The dataset has mainly divided into three subsets. They can be summarized as,

- *Training Set* uses the large portion of the whole dataset to train the neural network.
- *Validation Set* utilizes the small part of the dataset. It is generally created from the training set by the help of cross validation. In this step, internal parameters of the network are adjusted by using this set.
- *Test Set* contains unlabelled and unseen data which are used to evaluate the generalization performance of the network.

In the thesis, Washington RGB-D dataset which provides RGB and depth images is used.

A large-scale and hierarchical Washington RGB-D object dataset has been created via Microsoft Kinect cameara by Lai, Bo, Liefeng anf Fox (2011b). The whole dataset includes 300 distinct instances and they are organised as 51 object classes (categories). All images taken from the camera synchronously have 640×480 resolution. The whole dataset has RGB images, grayscale depth images, segmentation masks and location files for cropping images. Segmentation masks are also used to remove background from the

images to observe the evolution of the classification performance of the network. Object recognition can be performed in two different ways:

1. **Category Recognition :** In this categorization, the neural network has 51 classes. Some of the classes are shown in Figure 4.1. Each class consists of sub-classes of the same object. A sub-class from each category is used as test set for the network.

2. **Instance Recognition :** In this situation, the neural network has 300 classes. This means that each sub-class is considered as a new class. For example, apple category has 5 different sub-classes (instances), hence red apple and yellow apple are different classes that are shown in Figure 4.2.

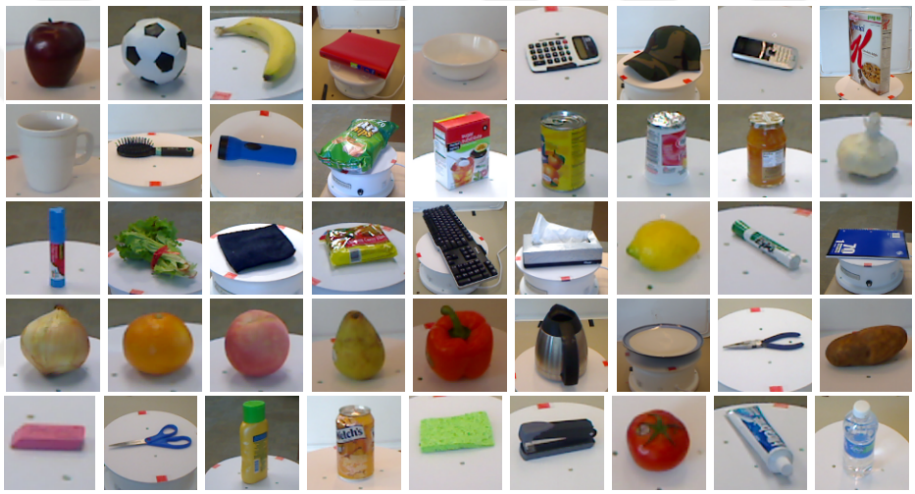


Figure 4.1 Different classes from the dataset (Lai et al., 2011b)

Figure 4.2 Instances from Apple object class with segmentation mask (Lai et al., 2011b)

In the thesis, the cropped versions of all images have been used in order to avoid from memory allocation problems. Besides, segmentation masks are applied to the depth images in order to eliminate unwanted depth pixels. Since the whole dataset is too large, it is downscaled. Before the downscaling operation, the sum of RGB and depth images is 440K approximately. The downscaled version of the dataset has 42k images separately. This study focuses on the category recognition with downscaled dataset. Training and test

sets are constructed by extracting an instance from each category (300 - 51 = 249). This process is as: Each class has different instances like in Figure 4.2 and the name of these instances is given as Apple_1, Apple_2, ... so on. Then, one of these instances is chosen as test class, the others are used as train set. This is also valid for all object classes. Shortly, training set has approximately 35k images, the remaining images are evaluated as test set. Performances of all networks have been evaluated via confusion matrix.

Confusion matrix is a simple table used to describe the performance of a classifier or network over test data whose true values are known. For binary classification, confusion matrix can be written as,

Table 4.1 An example of confusion matrix for binary classification

| | **Prediction (Class = 0)** | **Prediction (Class = 1)** |
|---|---|---|
| **Target (Class = 0)** | True Positive (**TP**) | False Negative (**FN**) |
| **Target (Class = 1)** | False Positive (**FP**) | True Negative (**TN**) |

Mathematical formulas for confusion matrix can be given as follows.

$$Sensitivity = True\ Positive\ Rate = \frac{TP}{TP+FN} \tag{4.1}$$

$$Specificity = True\ Negative\ Rate = \frac{TN}{TN+FP} \tag{4.2}$$

$$PPV = Positive\ Predictive\ Value = \frac{TP}{TP+FP} \tag{4.3}$$

$$NPV = Negative\ Predictive\ Value = \frac{TN}{TN+FN} \tag{4.4}$$

The overall accuracy of the classifier is computed as,

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{4.5}$$

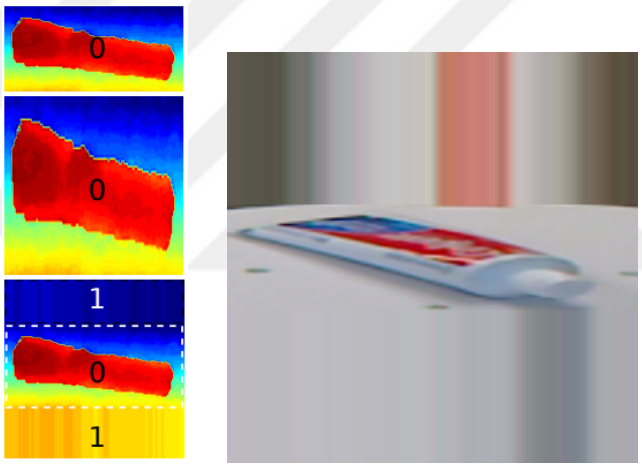
On the other hand, Washington RGB-D dataset is challenging dataset, because cropped images have different dimensions. Therefore, all images must be resized with respect to the input shape of the networks by keeping their aspect ratio constant. In addition, it is hard to discriminate the objects from grayscale depth images since shapes of the objects in different categories are very similar to each other.

## 4.2 Preprocessing Steps

All RGB and grayscale images of the dataset have been preprocessed in order to improve the performance of the neural network.

### *4.2.1 Resizing Images*

Due to the different dimensions of the cropped images, it is obligatory to resize RGB and depth images according to the input shape of the Capsule and Convolutional neural networks. In the thesis, the input shape of Capsule networks is 32×32, whereas the input size of CNNs is adjusted as 150 ×150. This operation has been performed for RGB and depth images. The critical point of resizing depth images is to encode grayscale depth images as RGB images. The main aim is to preserve the shape of the object as shown in below figure.




(a) Depth Image (b) RGB Image

Figure 4.3 Resizing images with original aspect ratio (Eitel et al., 2015)

This process can be explained in detail as,

1. Specify the input shape of the network,
2. Find the longest edge of the cropped image and compute the ratio between the longest edge and desired dimesion,
3. Using this ratio, resize the shortest edge of the image,
4. Split RGB images into separate color channels,

5. Extend the borders of the longest edge of the image in the direction of the shortest edge,

6. Merge the channels to create final image.

The flowchart of this operation has been used to resize training and test images for both capsule and convolutional neural networks.

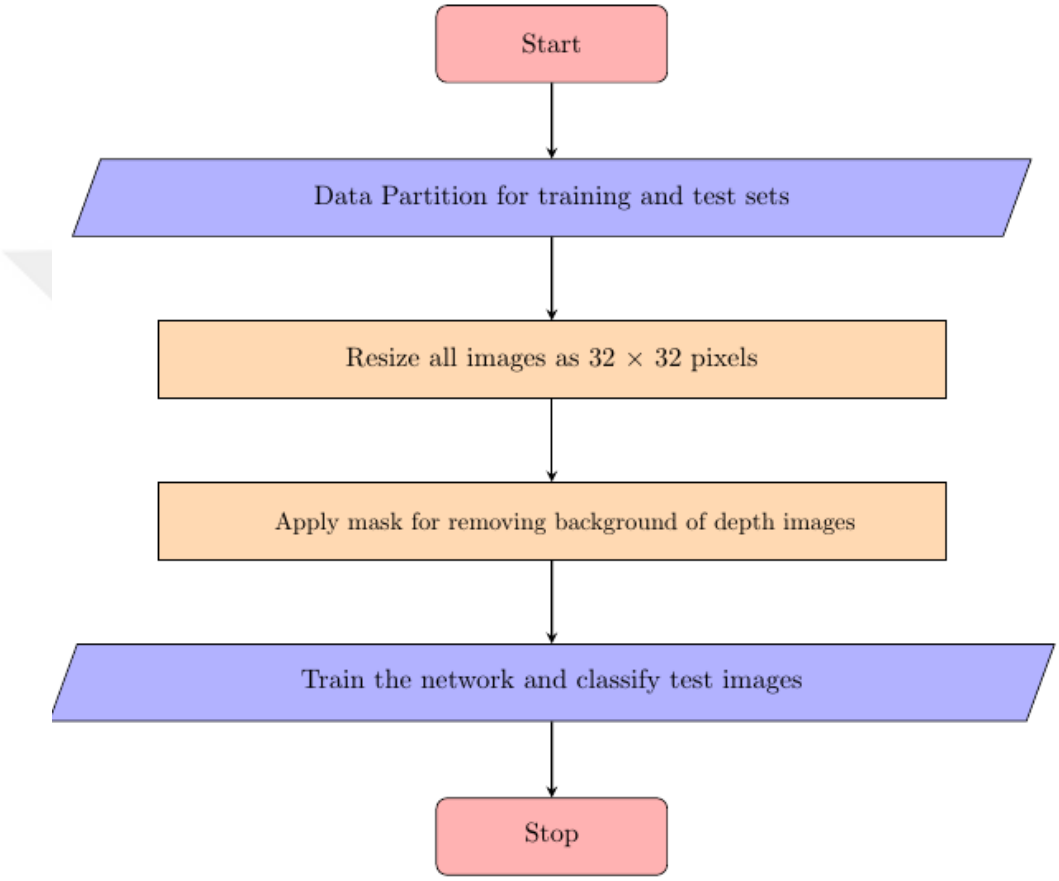



Figure 4.4 Flowchart of the resizing operation for depth images

### *4.2.2 Encoding Depth Images into RGB*

This step is applied only for grayscale depth images. In order to encode them into RGB, *jet colormap* is used. The result of the colorization technique can be seen below.

### *4.2.3 Interpolating Missing Depth Pixels*

RGB images with their backgrounds have been used to fill the missing depth values by interpolation. When performing the interpolation, the Euclidean distance among neigh-

(a) Grayscale Depth Image (b) Coloured Depth Image

Figure 4.5 Difference between grayscale and coloured depth images

bour RGB pixels is computed in order to find appropriate depth pixel (Thörnberg, 2015). Euclidean distance is defined as,

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{4.6}$$

The general procedure of this interpolation can be expressed in detail as,

1. Find the minimum depth value of original depth image and subtract the value from the whole images,
2. Normalize the depth image and set missing depth values in borders of the image to maximum depth value,
3. Determine the location of the missing pixel,
4. Search non-missing pixel values around the missing pixel in left, right, up and down directions. At the end of searching, these points form a rectangular matrix,
5. Convert the matrix into a vector by neglecting missing depth values and create a new vector to store non-missing depth values,
6. Find the RGB pixel values in each element of the vector and missing depth pixel,
7. Compute the Euclidean distances between RGB pixel value of missing depth image and each element of the vector,
8. Find the smallest distance and take the location of that RGB pixel. The value of this location corresponds to the pixel value of the missing depth.

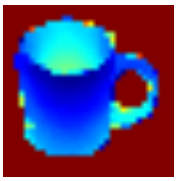


Figure 4.6 An example of masked and interpolated sample

This technique is one of the best interpolation methods, but it requires high computational cost. The flowchart of this operation can be seen below.

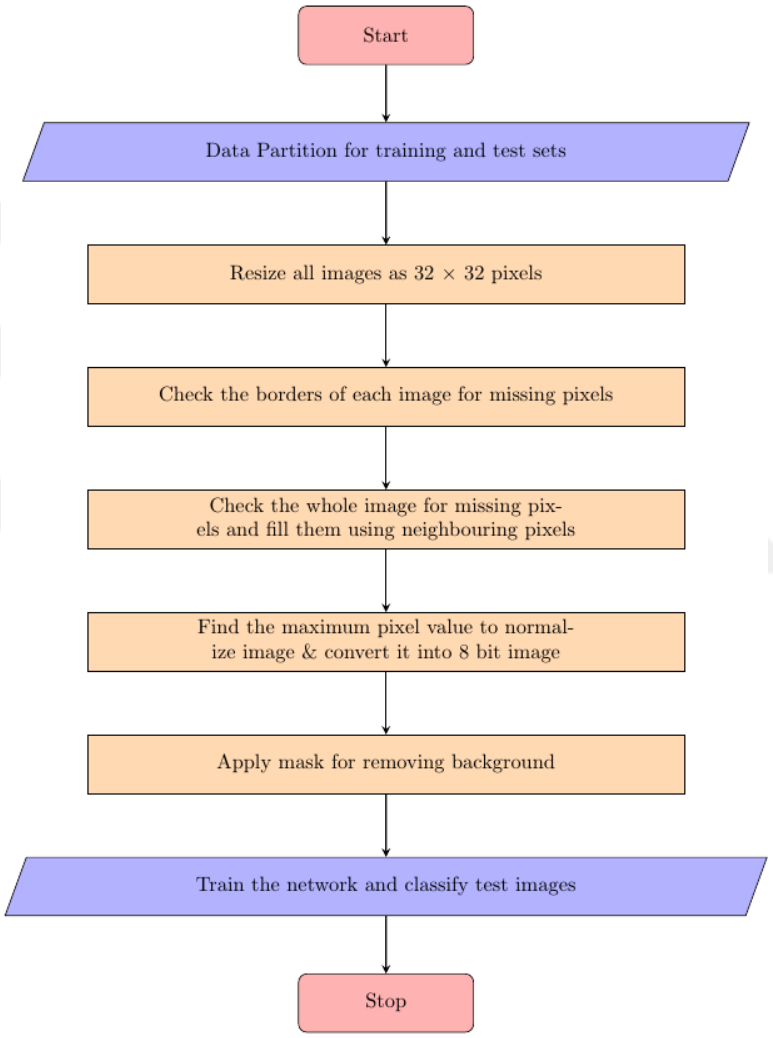



Figure 4.7 Flowchart of filling deficient depth values

## 4.3 Performance of VGG16

In the thesis, VGG16 architecture is used and the neurons in the last layer have been changed from 1000 to 51 since Washington RGB-D dataset has 51 distinct object classes. Additionally,the input shape of VGG16 has been changed from 224 × 224 to 150 × 150 in

order to prevent memory problem. After reconfiguration of VGG16, the model summary can be given in the following table. The table represents the output of each layer after reconfiguring VGG16.

Table 4.2 Base structure of VGG16 after reconfiguration

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_1 (InputLayer) | (None, 150, 150, 3) | 0 |
| block1_conv1 (Conv2D) | (None, 150, 150, 64) | 1792 |
| block1_conv2 (Conv2D) | (None, 150, 150, 64) | 36928 |
| block1_pool (MaxPooling2D) | (None, 75, 75, 64) | 0 |
| block2_conv1 (Conv2D) | (None, 75, 75, 128) | 73856 |
| block2_conv2 (Conv2D) | (None, 75, 75, 128) | 147584 |
| block2_pool (MaxPooling2D) | (None, 37, 37, 128) | 0 |
| block3_conv1 (Conv2D) | (None, 37, 37, 256) | 295168 |
| block3_conv2 (Conv2D) | (None, 37, 37, 256) | 590080 |
| block3_conv3 (Conv2D) | (None, 37, 37, 256) | 590080 |
| block3_pool (MaxPooling2D) | (None, 18, 18, 256) | 0 |
| block4_conv1 (Conv2D) | (None, 18, 18, 512) | 1180160 |
| block4_conv2 (Conv2D) | (None, 18, 18, 512) | 2359808 |
| block4_conv3 (Conv2D) | (None, 18, 18, 512) | 2359808 |
| block4_pool (MaxPooling2D) | (None, 9, 9, 512) | 0 |
| block5_conv1 (Conv2D) | (None, 9, 9, 512) | 2359808 |
| block5_conv2 (Conv2D) | (None, 9, 9, 512) | 2359808 |
| block5_conv3 (Conv2D) | (None, 9, 9, 512) | 2359808 |
| block5_pool (MaxPooling2D) | (None, 4, 4, 512) | 0 |

Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

On the other hand, the topology given above does not include the fully connected layers to be trained for Washington RGB-D dataset. The fully connected layers and change in the number of parameters can be depicted in the following table.

Table 4.3 Fully-connected layers of VGG16 after reconfiguration

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              33558528
_________________________________________________________________
dense_2 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dense_3 (Dense)              (None, 51)                208947
=================================================================
Total params: 65,263,475
Trainable params: 65,263,475
Non-trainable params: 0
_________________________________________________________________
```

### *4.3.1 Simulation Results of VGG16*

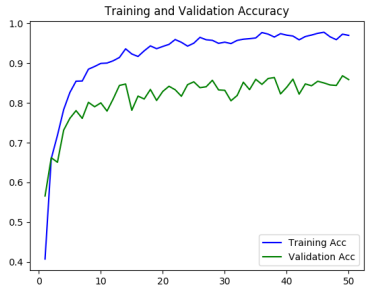
Different versions of simulations have been tested on Washington RGB-D dataset by using VGG16 and all obtained results are given in tabular form.

Table 4.4 VGG16 simulation results on Washington RGB-D dataset

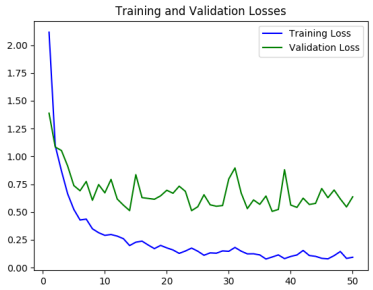
| **Image Type** | **Background Removal** | **Fusion** | **Data Augmentation** | **Test Accuracy (%)** |
|---|---|---|---|---|
| Coloured Depth | Yes | No | No | 82.4 |
| RGB | Yes | No | No | 55.24 |
| RGB | No | No | No | 75.88 |
| RGB | No | No | Yes | 78.24 |
| **RGB** & Coloured Depth | **No** & Yes | Yes | No | 83.33 |

In this table, depth images have been masked with segmentation masks to increase the accuracy of the object recognition. According to the results, the neural network shows better performance on RGB images without segmentation masks. In addition to this, horizontal flipping has been just performed on RGB images as data augmentation method. In fusion technique, RGB and depth images are applied to separate VGG16 networks to extract the features, which improves the performance of the network. After that, all features are concatenated in a unique vector and classified. In the study performed by Eitel et al. (2015), the accuracies of CNNs whose input shape is $227 \times 227 \times 3$ are 84.1% and 83.8% for RGB and depth images, respectively. In our architecture of VGG16, the accuracy of coloured depth images is 82.4% by using preprocessing images and changing

the input shape from $224 \times 224 \times 3$ to $150 \times 150 \times 3$. Besides, the training and validation performances of the simulated networks can be seen in the following figures.

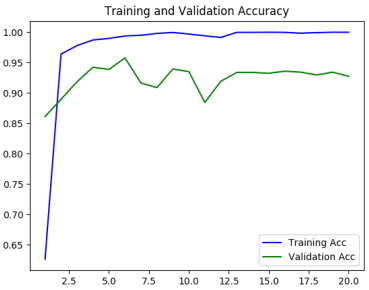



(a) Training and Validation Accuracy

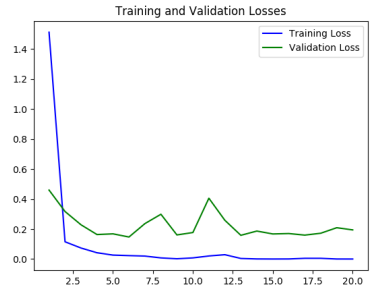



(b) Training and Validation Losses

Figure 4.8 Performance of VGG16 on coloured depth images

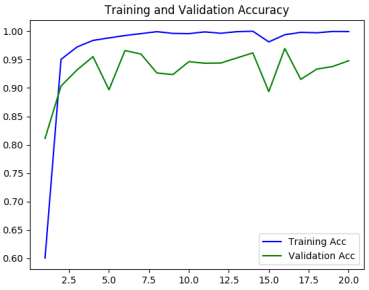



(a) Training and Validation Accuracy

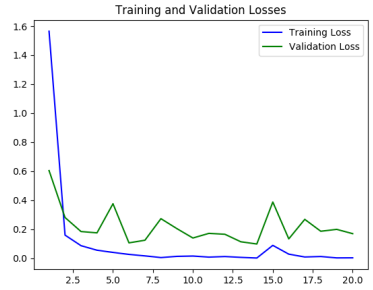



(b) Training and Validation Losses

Figure 4.9 Performance of VGG16 on non-masked RGB images




(a) Training and Validation Accuracy




(b) Training and Validation Losses

Figure 4.10 Performance of VGG16 on non-masked, augmented RGB images

When computing the test accuracy of the convolutional neural networks, weights which are saved at the best validation accuracy are used. All confusion matrices are given in Appendix 1 - Appendix 3. The result for depth images states that the neural network does

not classify round objects. On the other hand, data augmentation on non-masked RGB images improves the performance of the network.

On the other hand, 20 sub-classes of Washington RGB-D dataset which are selected with containing dissimilar classes are simulated to compare the performance of VGG16 to CapsNet. The main aim of the simulation is to compare the performance of both networks when the dataset does not include similar objects. All simulation results can be seen in the following table.

Table 4.5 VGG16 simulation results on 20 classes of Washington RGB-D dataset

| Image Type | Background Removal | Fusion | Data Augmentation | Test Accuracy (%) |
|---|---|---|---|---|
| Coloured Depth | Yes | No | No | 95.8 |
| RGB | Yes | No | No | 88.94 |

According to Table 4.4 and Table 4.5, fewer object classes improve the performance of VGG16. The confusion matrix of VGG16 on coloured depth images can be seen below.

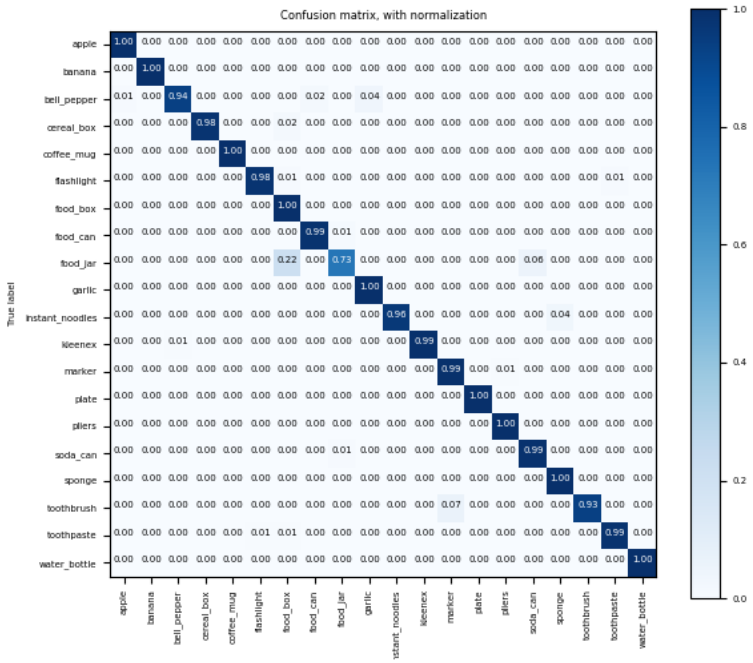



Figure 4.11 Confusion matrix of 20 classes on coloured depth images for VGG16

## 4.4 Performance of CapsNet

The input-output connection of the capsule network constructed for RGB-D dataset can be seen in the following table.

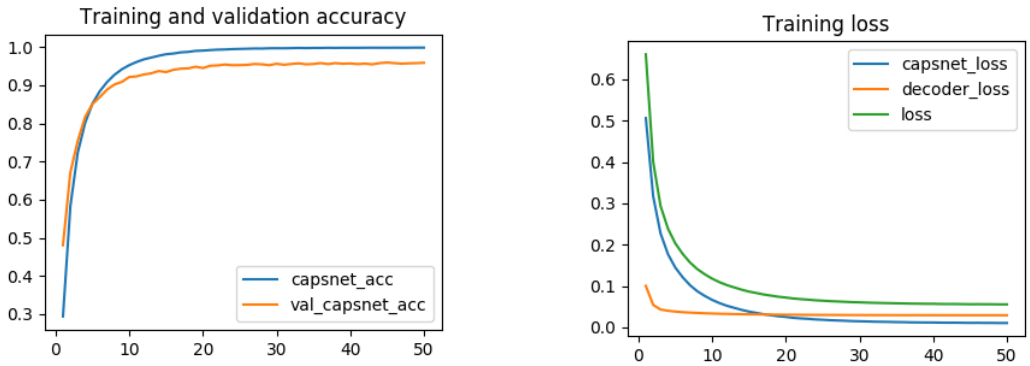
Table 4.6 Capsule network for Washington RGB-D dataset

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | (None, 32, 32, 3) | 0 | |
| conv1 (Conv2D) | (None, 24, 24, 256) | 62464 | input_1[0][0] |
| primarycap_conv2d (Conv2D) | (None, 8, 8, 512) | 10617344 | conv1[0][0] |
| primarycap_reshape (Reshape) | (None, 2048, 16) | 0 | primarycap_conv2d[0][0] |
| primarycap_squash (Lambda) | (None, 2048, 16) | 0 | primarycap_reshape[0][0] |
| caps (CapsuleLayer) | (None, 51, 16) | 26738688 | primarycap_squash[0][0] |
| input_2 (InputLayer) | (None, 51) | 0 | |
| mask_1 (Mask) | (None, 816) | 0 | caps[0][0]<br>input_2[0][0] |
| capsnet (Length) | (None, 51) | 0 | caps[0][0] |
| decoder (Sequential) | (None, 32, 32, 3) | 4092416 | mask_1[0][0] |

Total params: 41,510,912
Trainable params: 41,510,912
Non-trainable params: 0

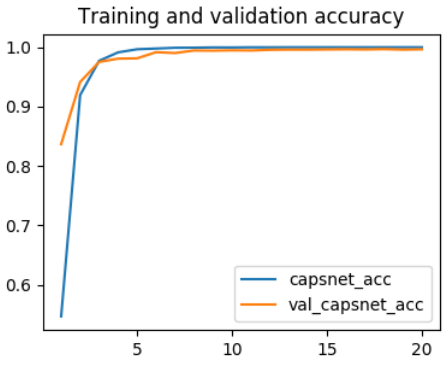
### *4.4.1 Simulation Results of CapsNet*

Different simulations have been performed on Washington RGB-D dataset. Besides, the performance of the capsule networks has been evaluated by changing the number of convolution layers and primary capsules. All simulations are realized on Google Colaboratory which provides Tesla K80 GPU. The training and validation performance of the simulated networks can be seen in the following figures.

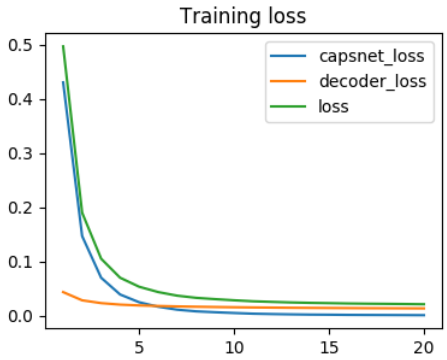



(a) Training and Validation Accuracy (b) Training and Validation Losses

Figure 4.12 Performance of CapsNet on coloured depth images




(a) Training and Validation Accuracy




(b) Training and Validation Losses

Figure 4.13 Performance of CapsNet on non-masked RGB images

Table 4.7 CapsNet simulation results on Washington RGB-D dataset

| Image Type | Background Removal | Conv Layer # | Data Augmentation | Primary Caps # | Test Accuracy (%) |
|---|---|---|---|---|---|
| Coloured Depth | Yes | 1 | No | 16 | 70.72 |
| Coloured Depth | Yes | 2 | No | 64 | 70.9 |
| RGB | Yes | 1 | No | 16 | 58.92 |
| RGB | No | 1 | No | 16 | 65.29 |
| Grayscale Depth | Yes | 2 | No | 64 | 67.95 |

As seen in Table 4.7, RGB images with background represent better performance. Furthermore, coloured depth images have better accuracy according to grayscale depth images. According to confusion matrices in Appendix 4 and Appendix 5, it is hard to discriminate the similar objects from each other for capsule networks.

(a) RGB image without mask

(b) RGB image with mask

Figure 4.14 Background removal on RGB images (Lai et al., 2011b)

As seen in simulation results given in Table 4.4 and Table 4.7, both VGG16 and CapsNet show better performance on RGB images without mask. The important change in the performances of the architectures is that background images have been treated as features to recognize objects. The rise of the performance of VGG16 on non-masked RGB images is much higher than that of CapsNet due to its topology.

As performed for VGG16, the same 20 sub-classes of Washington RGB-D dataset which are selected with containing dissimilar classes are simulated to compare the performance of CapsNet to VGG16. All simulation results can be seen in the following table.

Table 4.8 CapsNet simulation results on 20 classes of Washington RGB-D dataset

| Image Type | Background Removal | Fusion | Data Augmentation | Test Accuracy (%) |
|---|---|---|---|---|
| Coloured Depth | Yes | No | No | 94.13 |
| RGB | Yes | No | No | 80.97 |

According to Table 4.7 and Table 4.8, fewer object classes improve the performance of CapsNet. In addition to this, the recognition performance of CapsNet converges to the performance of VGG16 for both RGB and depth images when the dataset contains non-identical object classes. The confusion matrix of CapsNet on coloured depth images can be seen below.

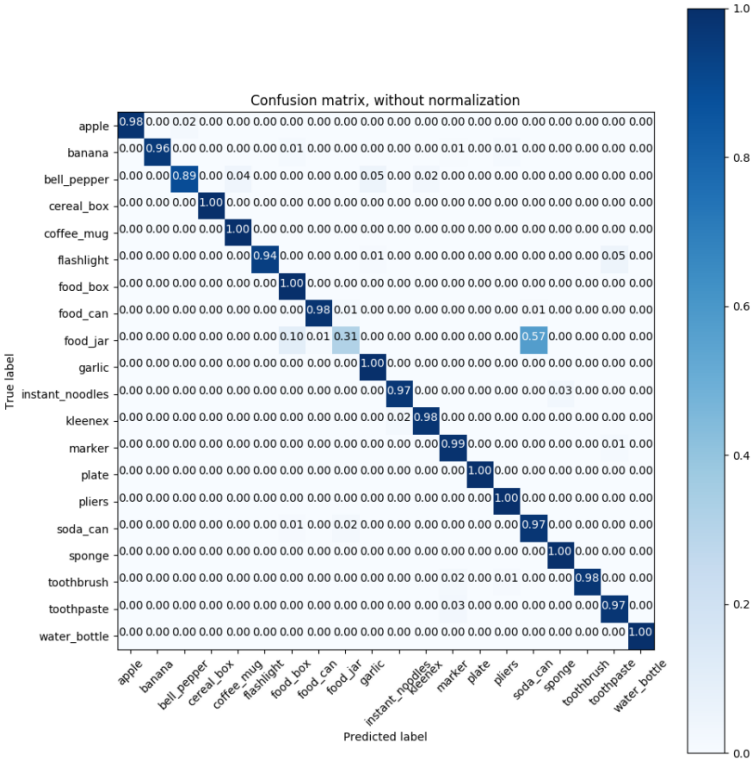
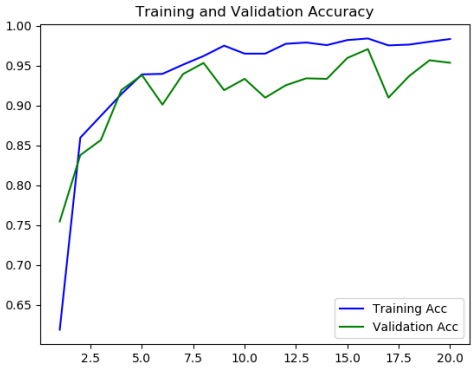



Figure 4.15 Confusion matrix of 20 classes on coloured depth images for CapsNet

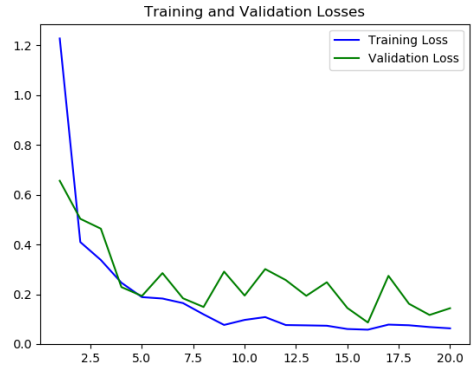
## 4.5 Proposed Hierarchical Structure

On the other hand, a hierarchical structure has been implemented instead of fusion operation. In this type of simulation, the confusion matrices which correspond to the

coloured depth and non-masked, no data augmented RGB images respectively in Table 4.4 are analysed in order to construct the hierarchical structure. For both type of images, the first layer of the hierarchical structure includes the object classes whose accuracy is greater than 90%. The remaining object classes are considered as the second layer of the structure. The critical point is that each layer of the hierarchical structure has different number of object categories. The training and validation performances of the hierarchical structure can be seen in the following figures.

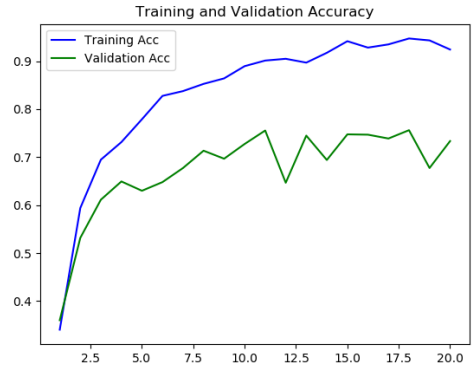



(a) Training and Validation Accuracy

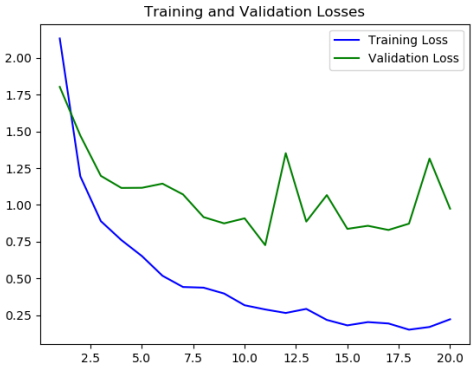



(b) Training and Validation Losses

Figure 4.16 Performance of first layer of VGG16 on coloured depth images



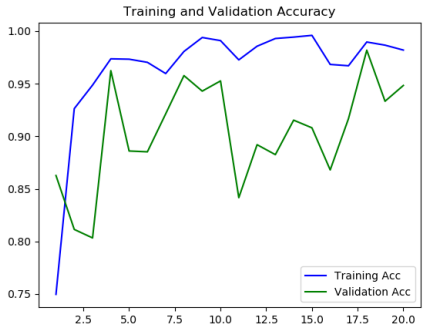

(a) Training and Validation Accuracy



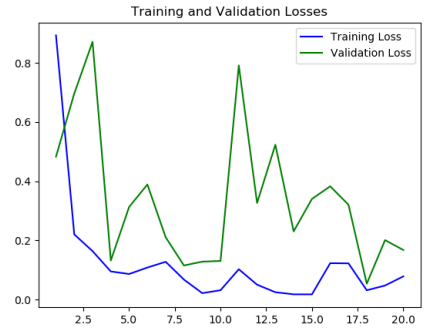

(b) Training and Validation Losses

Figure 4.17 Performance of second layer of VGG16 on coloured depth images

As seen in below and above figures, the first layer of the hierarchical VGG16 has better performance than the other layer on coloured depth images.

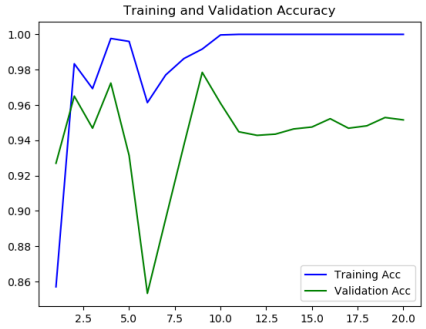



(a) Training and Validation Accuracy

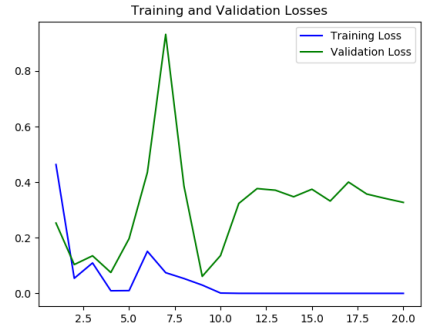



(b) Training and Validation Losses

Figure 4.18 Performance of first layer of VGG16 on non-masked RGB images




(a) Training and Validation Accuracy




(b) Training and Validation Losses

Figure 4.19 Performance of second layer of VGG16 on non-masked RGB images

Table 4.9 Simulation results for hierarchical VGG16 structure

| **Hierarchical Structure for Non-masked RGB Images** | | | | | |
|---|---|---|---|---|---|
| **Layer #** | **Test Image #** | **Correctly Classified #** | **Misclassified #** | **Class #** | **Test Accuracy (%)** |
| First Layer | 4770 | 4228 | 522 | 34 | 89.1 |
| Second Layer | 2263 | 1735 | 528 | 17 | 76.7 |
| **Total** | **7033** | **5963** | **1050** | **51** | **84.79** |
| **Hierarchical Structure for coloured depth Images** | | | | | |
| **Layer #** | **Test Image #** | **Correctly Classified #** | **Misclassified #** | **Class #** | **Test Accuracy (%)** |
| First Layer | 3478 | 3413 | 65 | 25 | 98.13 |
| Second Layer | 3555 | 2536 | 1019 | 26 | 71.33 |
| **Total** | **7033** | **5949** | **1084** | **51** | **84.59** |

As seen in above table, our hierarchical structure outperforms for RGB and depth images, separately. This hierarchical topology shows better performance with respect to

the results given in Table 4.4. Furthermore, confusion matrices of this structure can be seen in Appendices 6-9. In this hierarchical topology, the first layer consists of coloured depth images of object classes whose accuracies are higher than 90%. In addition, the second layer includes the remaining nonmasked RGB object classes. This procedure is valid for CapsNet and VGG16. All results of the hierarchical structure can be seen below.

Table 4.10 Simulation results for hierarchical VGG16 structure

| **Hierarchical Structure for VGG16** | | | | | |
|---|---|---|---|---|---|
| **Layer #** | **Test Image #** | **Correctly Classified #** | **Misclassified #** | **Class #** | **Test Accuracy (%)** |
| First Layer - Depth | 3478 | 3413 | 65 | 25 | 98.13 |
| Second Layer - RGB | 3555 | 2832 | 723 | 26 | 79.67 |
| **Total** | **7033** | **6245** | **788** | **51** | **88.8** |

Table 4.11 Simulation results for hierarchical CapsNet structure

| **Hierarchical Structure for CapsNet** | | | | | |
|---|---|---|---|---|---|
| **Layer #** | **Test Image #** | **Correctly Classified #** | **Misclassified #** | **Class #** | **Test Accuracy (%)** |
| First Layer - Depth | 3220 | 3148 | 72 | 23 | 97.76 |
| Second Layer - RGB | 3813 | 2447 | 1366 | 28 | 64.18 |
| **Total** | **7033** | **5595** | **1438** | **51** | **79.55** |

As seen in above results, the proposed hierarchical structure improves the performance of both networks when combining non-masked RGB images with coloured depth images. According to the results, the best performance has been achieved with the hierarchical VGG16..

# CHAPTER FIVE

# CONCLUSION

In this thesis, the object recognition performance of capsule networks has been evaluated on a dataset includes RGB and grayscale depth images. Two different capsule networks have been implemented in order to classify objects using Washington RGB-D dataset. The most important aspect between capsule network and convolutional neural networks is that capsule networks have used dynamic routing algorithms instead of pooling layers in order to preserve spatial relationship among features. The general architecture of the capsule network used in the thesis has at least one or two convolution layers with 16 primary capsules. Furthermore, VGG16 with an input shape of $150 \times 150$ and five convolution blocks have been constructed for performance comparison. For both networks, preprocessing step has played in a crucial role. The Washington RGB-D dataset is quite challenging since all images have different size. The resize operation has been performed by keeping the aspect ratio of the images constant. Due to noisy data coming from the sensor, all depth images have deficient pixels. Interpolation method proposed by Thörnberg (2015) used for this operation exploits RGB pixels to determine the values of non-missing depth pixels. After that, all grayscale depth images are coloured by jet colormap.

Simulation results have shown that both capsule and convolutional neural networks shows better performance when segmentation masks have not been applied to RGB images. Besides, when horizontal flipping is applied to RGB images for data augmentation, the performance of CNNs improves slightly. CNN has better performance than capsule network on both RGB and depth images. When the number of class is increased, the performance of the capsule network affects adversely. The generalization ability of the capsule network can not exceed the performance of CNNs. In addition to this, it is hard to discriminate similar objects for capsule network. When the number of classes is decreased and the dataset does not include similar objects, the performance of the capsule network approaches to the performance of CNNs.

The performance of the networks are directly associated with their topologies. As seen in the results, capsule network have only one convolution layer and 16 primary capsules while VGG16 has 5 convolutional layers and it is pre-trained with ImageNet. Normally, VGG16 has 65.2M parameters when it is used as fine-tuned, but capsule network with a convolutional layer has 41.5M parameters. Training time of capsule network takes only 5 hours on Google Colab, however it takes 13 hours approximately for VGG16 on GeForce GTX 780.

In the proposed technique, combination of non-masked RGB and colured depth images has been performed by means of a hierarchical architecture instead of fusion. This hierarchical topology has been implemented for VGG16 and capsule network. It is clear that hierarchical architectures of both networks make the generalization performance better. While the performance of the hierarchical capsule network is promising and the performance of the hierarchical VGG16 is comparable to the results given in the literature.

In this thesis, a hierarchical structure that consists of two layers has been proposed.

## REFERENCES


Achatz, S., & Conradt, J. (2016). State of the art of object recognition techniques. *Scientific Seminar, Neuroscientific System Theory,* 5-24.

Asif, U., Bennamoun, M., & Sohel, F. A. (2017). RGB-D object recognition and grasp detection using hierarchical cascaded forests. *IEEE Transactions on Robotics, 33*(3), 547-564.

Ayyüce Kızrak, M. (2018). *Yapay zekanın yeni ve çekici mimarisi kapsül ağına uygulamalı bir bakış.* Retrieved January, 18, 2019, from https://medium.com/@ayyucekizrak/yapay-zekan%C4%B1n- yeni-ve-%C3%A7ekici-mimarisi-kaps%C3%BCl-a%C4%9F%C4%B1nauygulamal%C4%B1-bir-bak%C4%B1%C5%9F-ef7310e3d847

Balodi, T. (2019). *Convolutional neural network (Cnn): graphical visualization with code explanation.* Retrieved January, 13, 2020, from https://www.analyticssteps.com/blogs/convolutional-neural-network-cnn-graphical-visualization-code-explanation

Baldominos, A., Saez, Y. & Isasi, P. (2018). Evolutionary design of convolutional neural networks for human activity recognition in sensor-rich environments. *Sensors, 18,* 1-24.

Bui, H. M., Lech, M., Cheng, E., Neville, K. & Burnett, I. S. (2016). Object recognition using deep convolutional features transformed by a recursive network structure. *IEEE Access, 4,* 10059-10066.

Caglayan, A., & Can, A. B. (2018). Volumetric object recognition using 3-D CNNs on depth data. *IEEE Access, 6*, 20058-20066.

Chatterjee, S. (2017). *Different kinds of convolutional filters.* Retrieved January, 18, 2019, from https://www.saama.com/blog/different-kinds-convolutional-filters/

Cheng, Y., Zhao, X., Cai, R., Li, Z., Huang, K., & Rui, Y. (2016). Semi-supervised multimodal deep learning for RGB-D object recognition. *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16),* 3345-3351

Cheng, Y., Zhao, X., Huang, K., & Tan, T. (2014). Semi-supervised learning for RGB-D object recognition. *IEEE* $22^{th}$*International Conference on Pattern Recognition (ICPR),* 1-6.

Eitel, A., Springenberg, J. T., Spinello, L., Riedmiller, M., & Burgard, W. (2015). Multimodal deep learning for robust object recognition. *IEEE International Conference on Intelligent Robots and Systems (IROS),* 1-7.

Gonçalves, B. (2016). *A practical introduction to machine(s) learning.* Retrieved January, 12, 2020, from https://www.slideshare.net/bgoncalves/a-practical-introduction-to-machines-learning

Hinton, G., Krizhevsky, A., & Wang, S. D. (2011). Transforming auto-Encoders. *International Conference on Artificial Neural Networks,* 1-8.

Lai, K., Bo, L., Ren, X., & Fox, D. (2011a). Sparse distance learning for object recognition combining RGB and depth information. *IEEE International Conference on Robotics and Automation (ICRA),* 1-7.

Lai, K., Bo, L., Ren, X., & Fox, D. (2011b). A large-scale hierarchical multi-view RGB-D object dataset. *IEEE International Conference on Robotics and Automation (ICRA),* 1-8.

Maturana, D., & Scherer, S. (2015). VoxNet: A 3D convolutional neural network for real-time object recognition. *IEEE International Conference on Intelligent Robots and*

*Systems (IROS),* 922-928.

Mocanu, I., & Clapon, C. (2018). Multimodal convolutional neural network for object detection using RGB-D images. 41$^{st}$ *International Conference on Telecommunications and Signal Processing (TSP),* 307-310.

Nishi, T., Kurogi, S., & Matsuo, K. (2017). Grading fruits and vegetables using RGB-D images and convolutional neural network. *IEEE Symposium Series on Computational Intelligence (SSCI),* 1-6.

Patel, K. (2019). *Overfitting vs underfitting in neural network and comparison of error rate with complexity graph.* Retrieved February, 01, 2019, from https://towardsdatascience.com/overfitting-vs-underfitting-ddc80c2fc00d

Pechyonkin, M. (2017). *Understanding Hinton's capsule networks. part I: intuition.* Retrieved December, 22, 2019, from https://medium.com/ai%C2%B3-theory-practice-business/ understanding-hintons- capsule-networks-part-i-intuition-b4b559d1159b

Rahman, M. M., Tan, Y., Xue, J., & Lu, K. (2017). RGB-D object recognition with multimodal deep convolutional neural network. *IEEE International Conference on Multimedia and Expo (ICME),* 991-996.

Rimal, K. (2018). *Understanding capsule network architecture.* Retrieved December, 15, 2019, from https://software.intel.com/en-us/articles/understanding-capsule-network-architecture

Sabour, S., Frosst, N., & Hinton, G. (2017). Dynamic routing between capsules. 31$^{st}$ *Conference on Neural Information Processing Systems (NIPS),* 1-11.

Schwarz, M., Schulz, H., & Behnke, S. (2015). RGB-D object recognition and pose estimation based on pre-trained convolutional neural network features. *IEEE International Conference on Robotics and Automation (ICRA),* 1-7.

Thakur, R. (2019). *Step by step VGG16 implementation in Keras for beginners.* Retrieved December, 22, 2019, from https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c

Thörnberg, J. (2015). *Combining RGB and depth images for robust object detection using convolutional neural networks.* Master Thesis, KTH Royal Institute Of Technology, Stockholm.

Uetz, R., & Behnke, S. (2009). Large-scale object recognition with CUDA-accelerated hierarchical neural networks. *IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS),* 536-541.

Wang, J., Lu, J., Chen, W., & Wu, X. (2015). Convolutional neural network for 3D object recognition based on RGB-D dataset. *IEEE* $10^{th}$ *Conference on Industrial Electronics and Applications (ICIEA),* 34-39.

Wei, L., Zhiguo, C., Yang, X., & Zhiwen, F. (2015). Hybrid RGB-D object recognition using convolutional neural network and fisher vector. *Chinese Automation Congress (CAC),* 506-511.

Zia, S., Yüksel, B., Yüret, D., & Yemez, Y. (2017). RGB-D object recognition using deep convolutional neural networks. *IEEE International Conference on Computer Vision Workshops,* 887-894.

# APPENDICES

## Appendix 1: Performance of VGG16 for coloured depth images

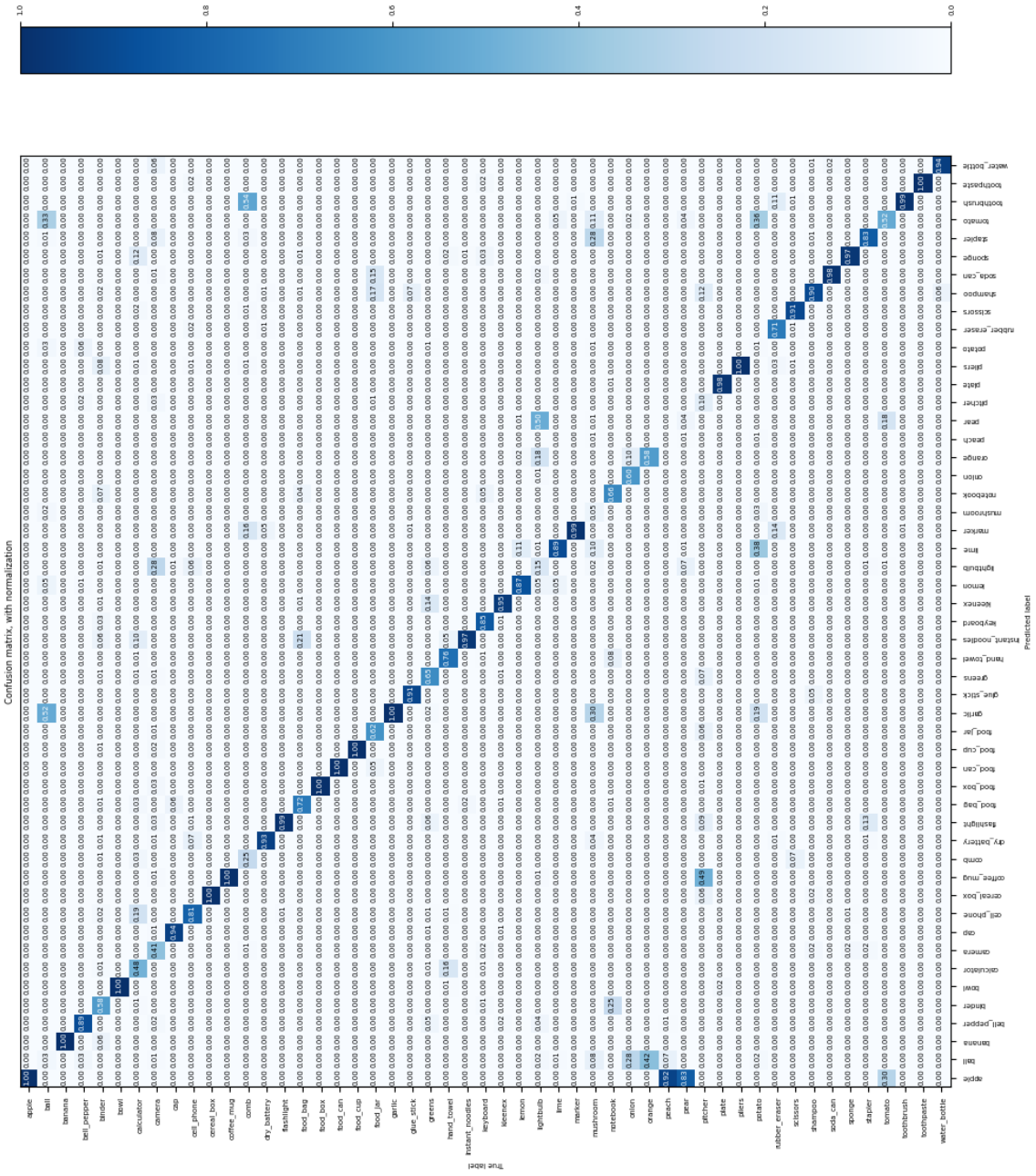



Figure A.1 Confusion matrix of VGG16 for coloured depth images

## Appendix 2: Performance of VGG16 for nonmasked RGB images

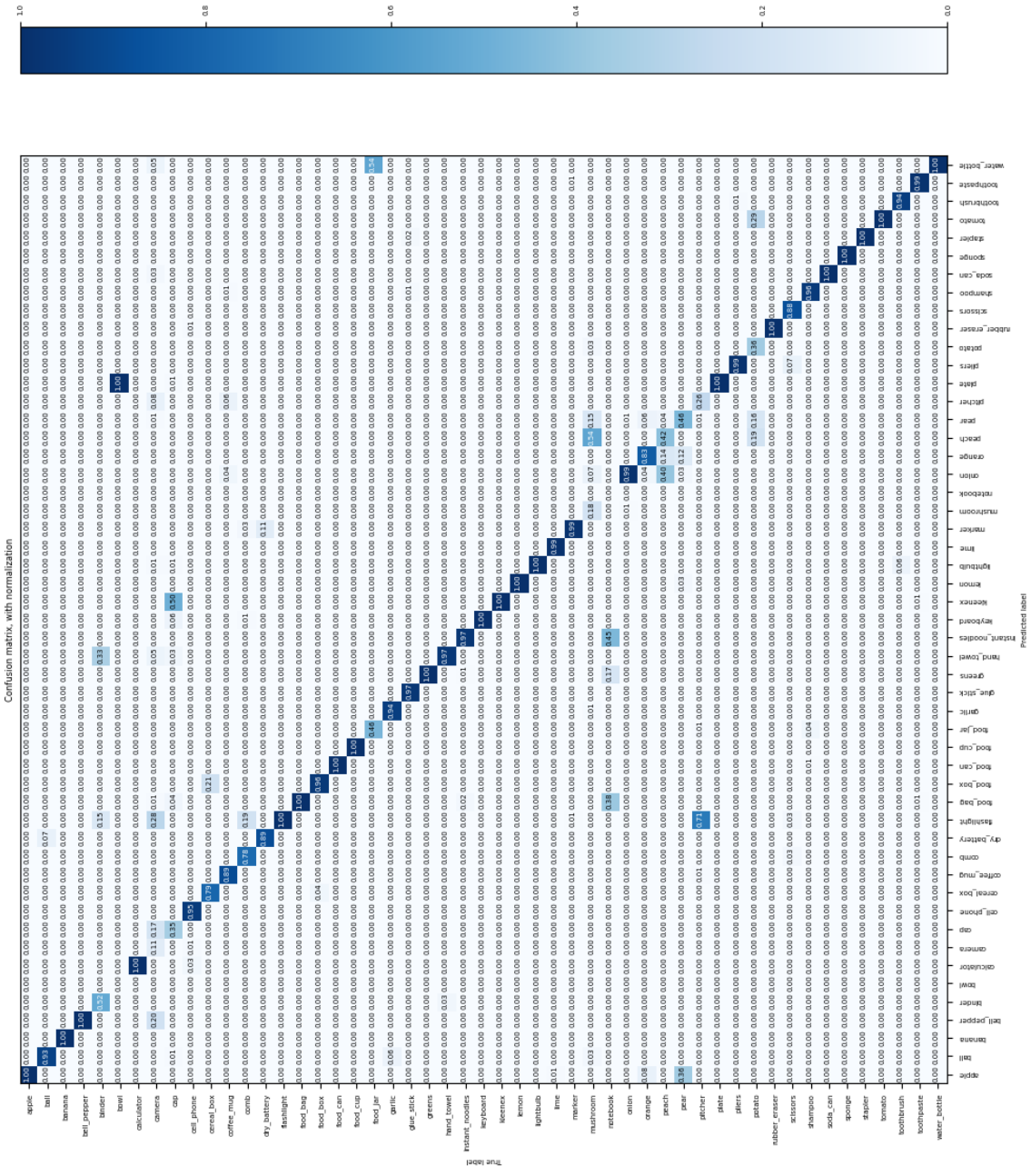



Figure A.2 Confusion matrix of VGG16 for nonmasked RGB images

## Appendix 3: Performance of VGG16 for augmented RGB images

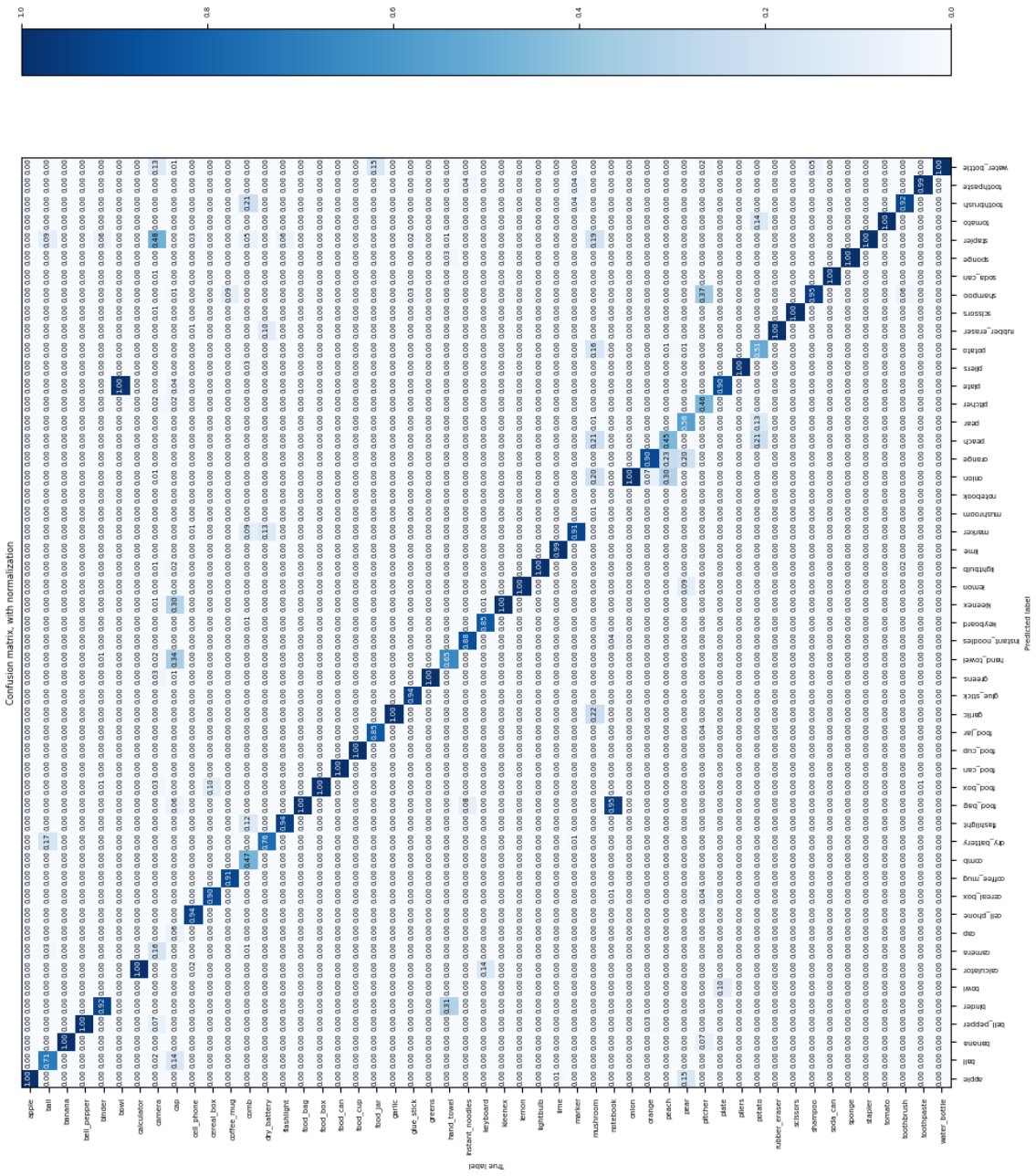



Figure A.3 Confusion matrix of VGG16 for nonmasked, augmented RGB images

## Appendix 4: Performance of CapsNet for coloured depth images

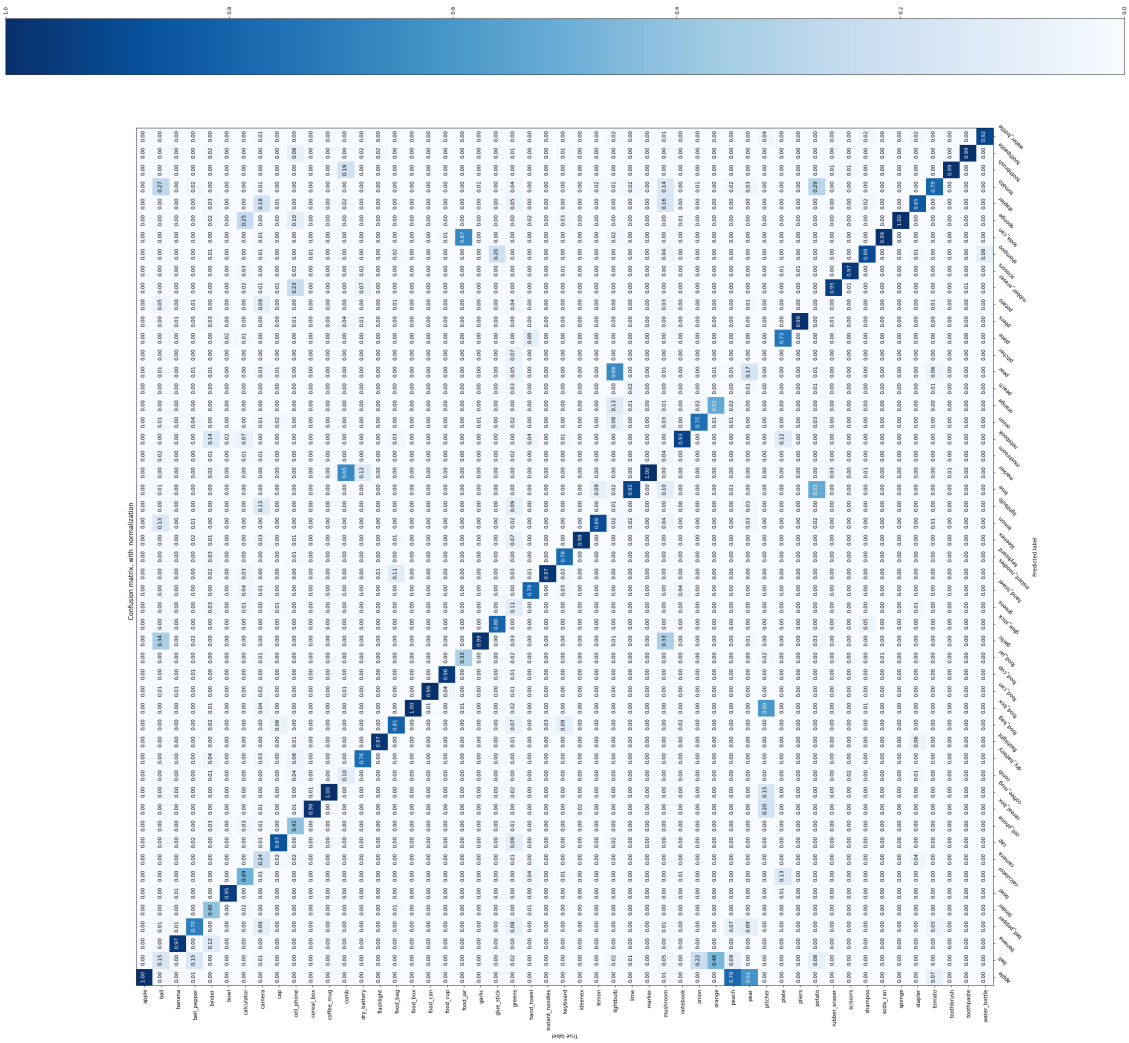



Figure A.4 Confusion matrix of CapsNet for coloured depth images

## Appendix 5: Performance of CapsNet for nonmasked RGB images

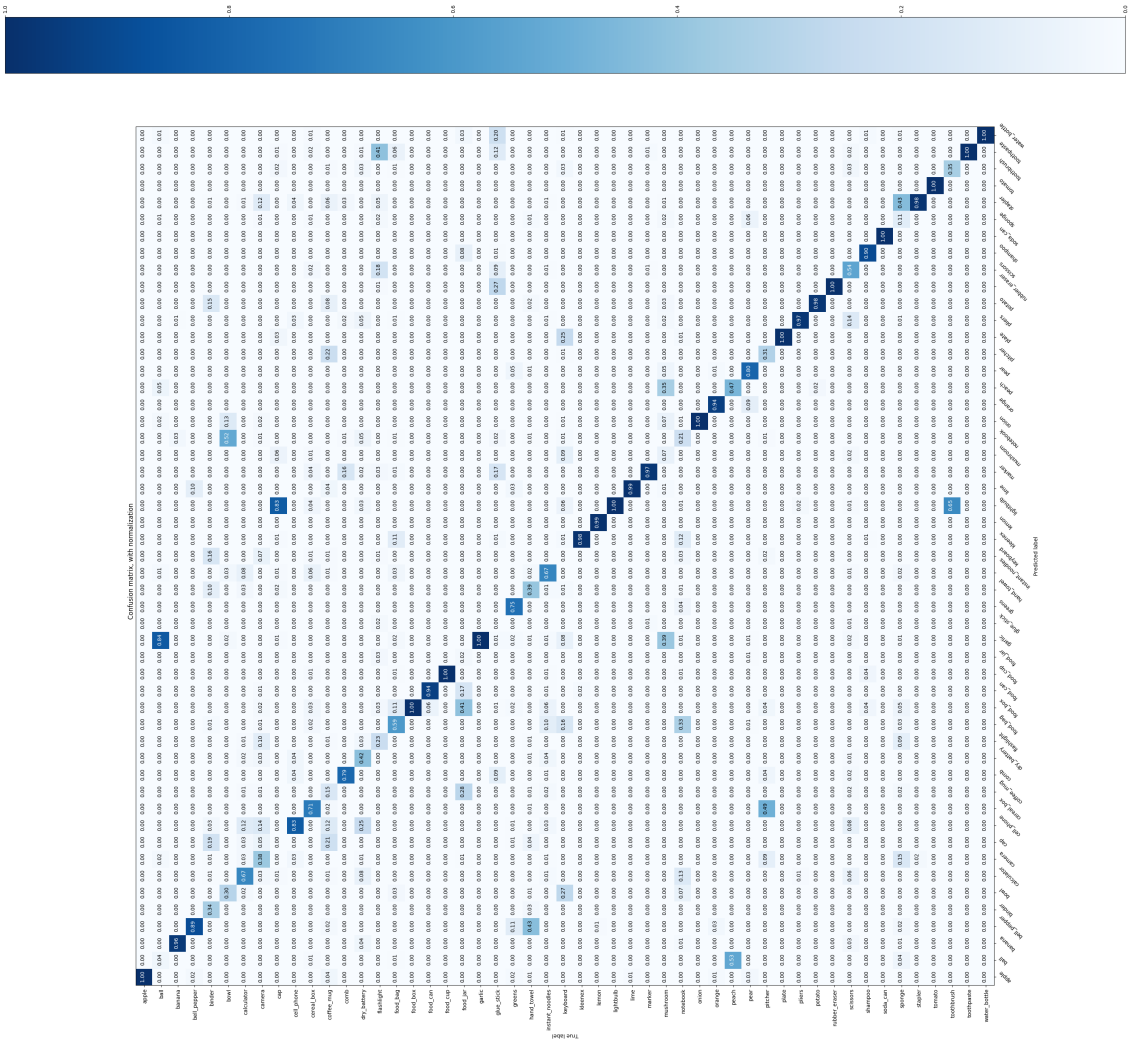



Figure A.5 Confusion matrix of CapsNet for nonmasked RGB images

## Appendix 6: Performance of $1^{st}$ layer of Hierarchical VGG16 for RGB images

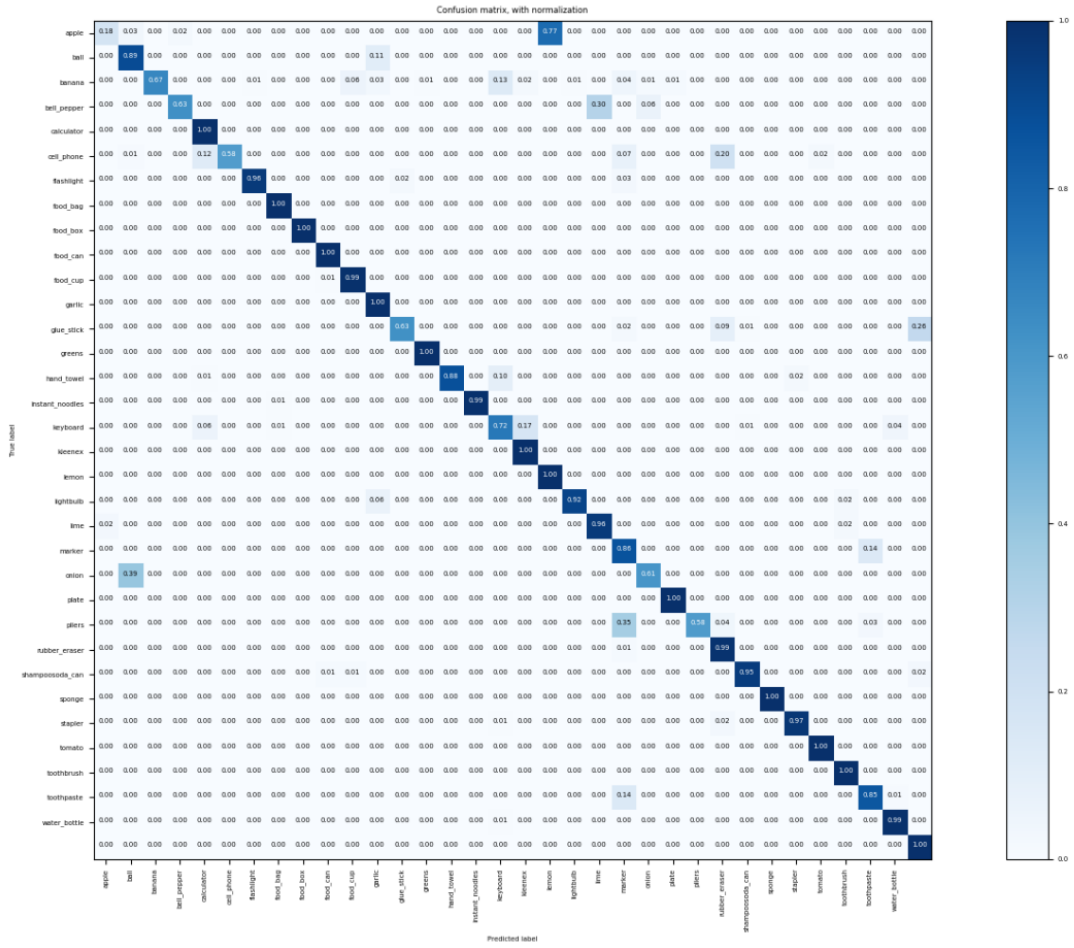



Figure A.6 Performance of $1^{st}$ layer of Hierarchical VGG16 for nonmasked RGB images

## Appendix 7: Performance of $2^{nd}$ layer of Hierarchical VGG16 for RGB images

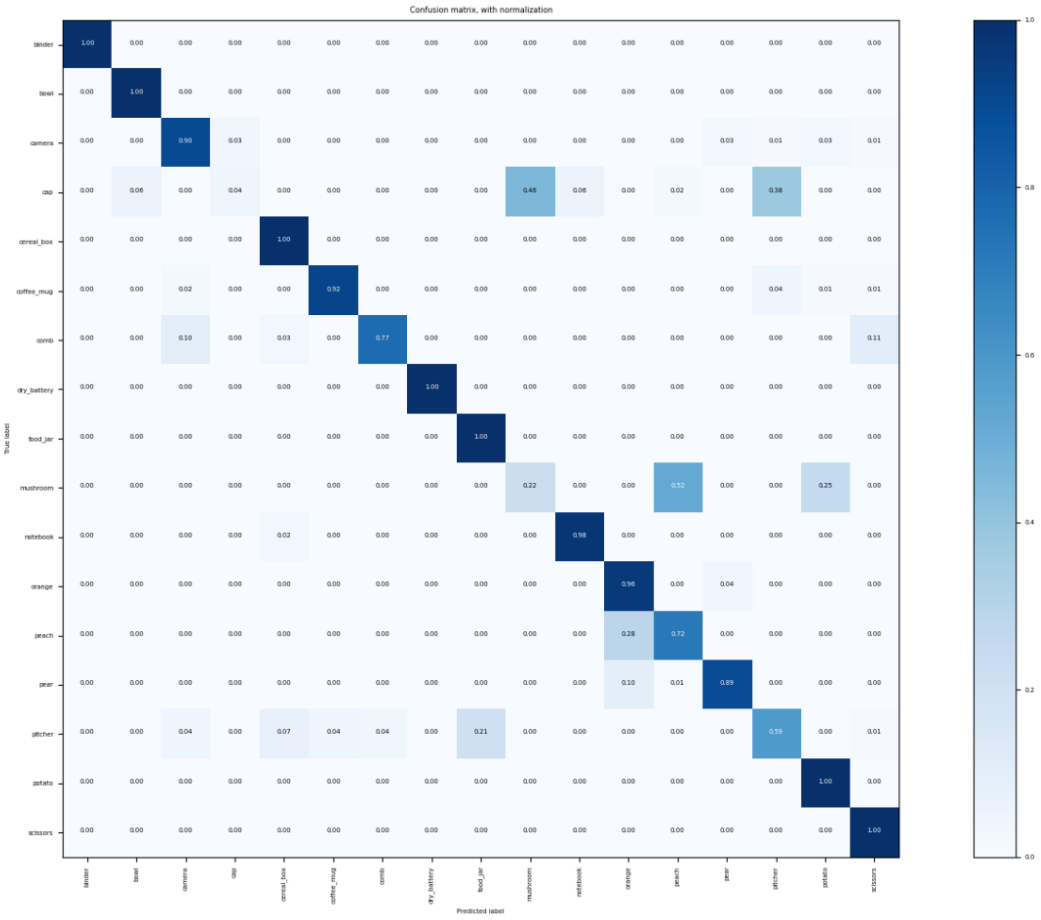



Figure A.7 Performance of $2^{st}$ layer of Hierarchical VGG16 for nonmasked RGB images

## Appendix 8: Performance of $1^{st}$ layer of Hierarchical VGG16 for depth images

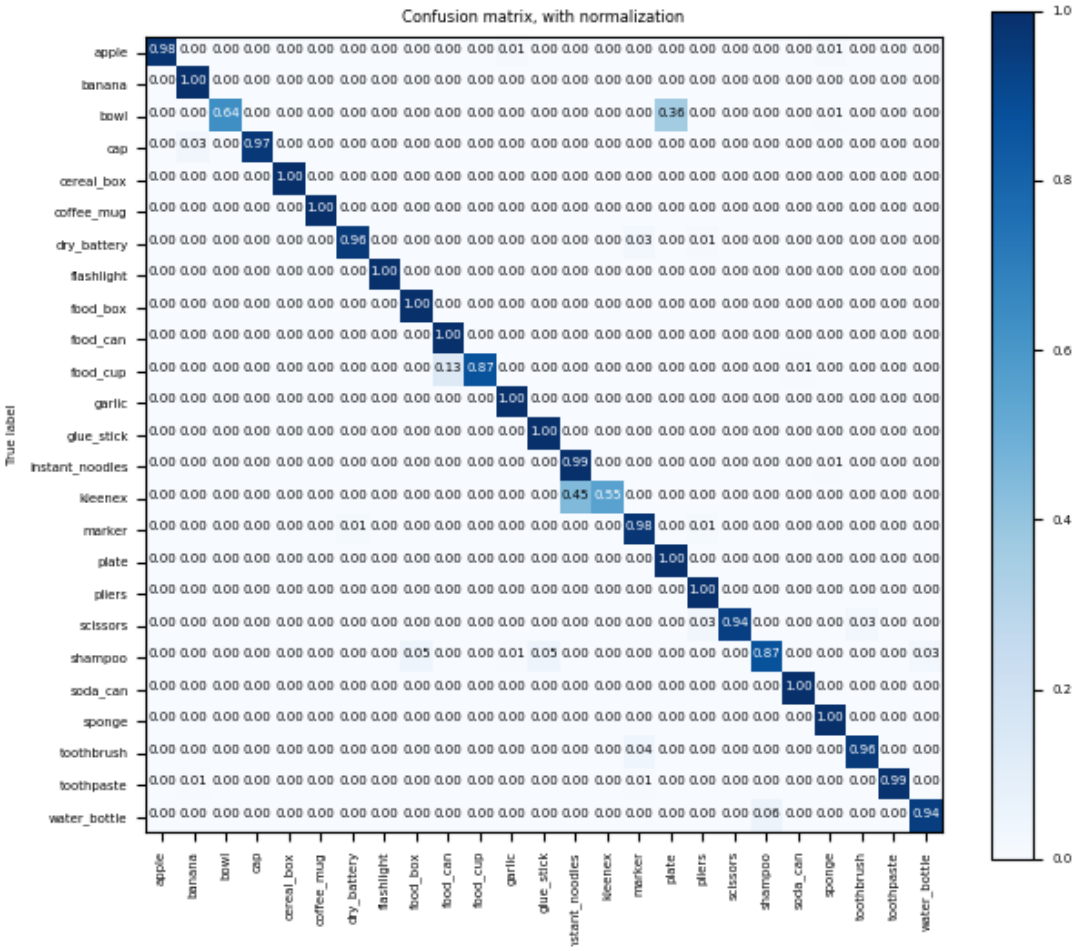



Figure A.8 Performance of $1^{st}$ layer of Hierarchical VGG16 for coloured depth images

## Appendix 9: Performance of $2^{nd}$ layer of Hierarchical VGG16 for depth images

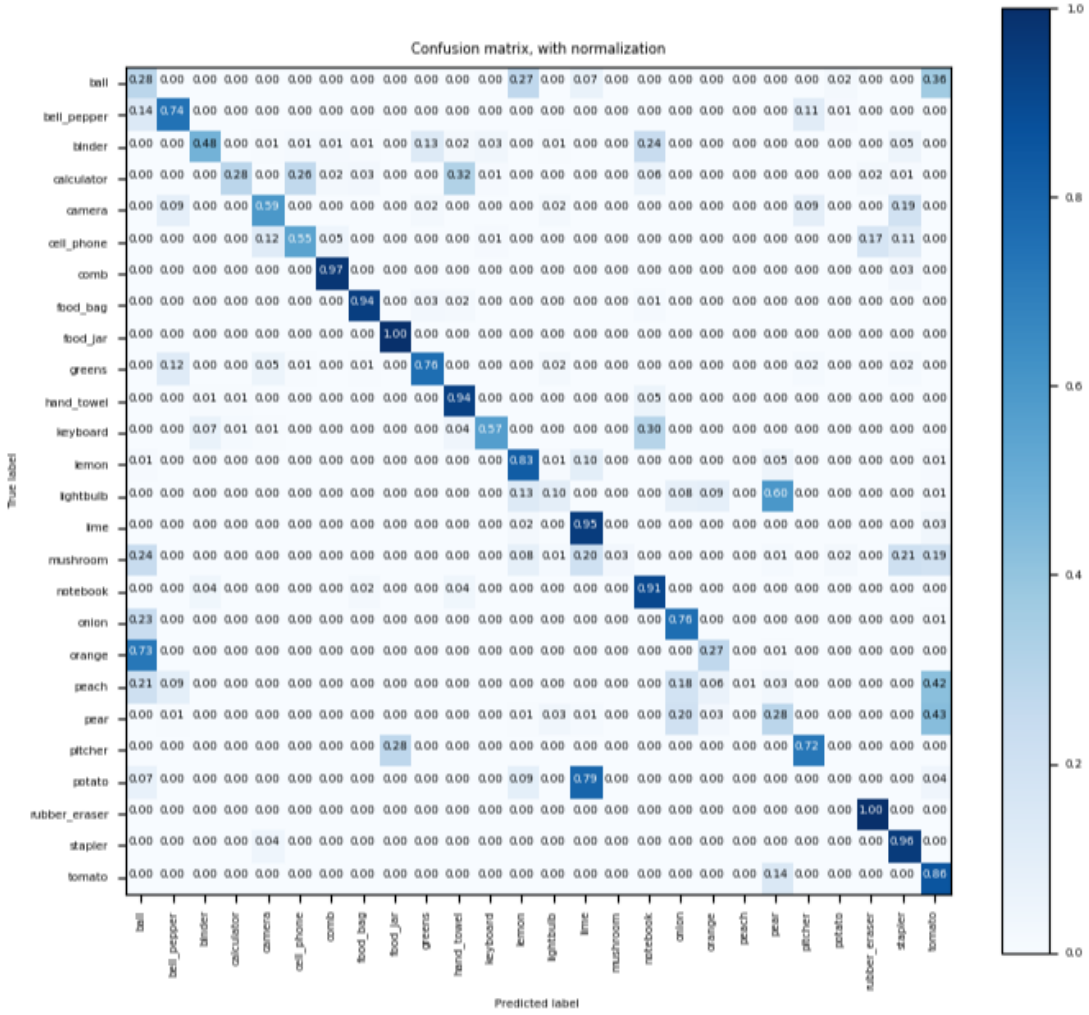



Figure A.9 Performance of $2^{st}$ layer of Hierarchical VGG16 for coloured depth images