THE REPUBLIC OF TURKEY

BAHÇEŞEHİR UNIVERSITY

STEREO VISION UTILIZING DEEP LEARNING

Master's Thesis

ABDULLAH NAZHAT ABDULLAH

İSTANBUL, 2020

THE REPUBLIC OF TURKEY

BAHÇEŞEHİR UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

COMPUTER ENGINEERING GRADUATE PROGRAM

STEREO VISION UTILIZING DEEP LEARNING

Master's Thesis

ABDULLAH NAZHAT ABDULLAH

Thesis Supervisor: ASSIST. PROF. DR. TARKAN AYDIN

İSTANBUL, 2020

THE REPUBLIC OF TURKEY
BAHCESEHIR UNIVERSITY

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
COMPUTER ENGINEERING

Title of the Master's Thesis	:	Stereo Vision Utilizing Deep Learning
Name/Last Name of the Student	:	Abdullah Nazhat Abdullah ABDULLAH
Date of Thesis Defence	:	10.01.2020

The thesis has been approved by the Graduate School of Natural and Applied Sciences.

Assist. Prof. Dr. Yücel Batu SALMAN
Graduate School Director

I certify that this thesis meets all the requirements as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Tarkan AYDIN
Program Coordinator

This is to certify that we have read this thesis and we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Science.

Examining Committee Members	Signatures
Thesis Supervisor	
Assist. Prof. Dr. Tarkan AYDIN	-----------------------------------
Member	
Assist. Prof. Dr. Cemal Okan ŞAKAR	-----------------------------------
Member	
Assist. Prof. Dr. Ulaş VURAL	-----------------------------------

ACKNOWLEDGEMENTS

A great amount of time and effort have gone into the work of this thesis. I would like to show my appreciation to all individuals who have encouraged and supported me throughout my work of this thesis during its performing and until its completion.

I would like to express my sincere appreciation to my advisor, Asst. Prof. Dr. Tarkan AYDIN, for his guidance and helpful support.

I would like also to thank my parents, teachers, friends and colleagues for supporting and encouraging me during my research.

İstanbul, 2020 Abdullah ABDULLAH

ABSTRACT

STEREO VISION UTILIZING DEEP LEARNING

Abdullah ABDULLAH
Computer Engineering Graduate Program
Thesis Supervisor: Assist. Prof. Dr. Tarkan AYDIN
January 2020, 77 pages

This thesis fixates on the problem of stereo vision based depth estimation as it represents one of the most challenging topics in computer vision research.

Recently, deep learning methods gained wide spread adoption among computer vision researchers and specialists, thus the thesis continues on this effort with the design and implementation of a deep learning architecture that have compactness and ease of training as the main target.

The Deep Neural Network architecture chosen to the task is a Fully Convolutional Encoder-Decoder and training data are stereo pair images with disparity maps as the output labels, several modifications are proposed in this design on the most recent proposals in the field and the new compact networks are trained and tested.

Keywords: Computer Vision, Machine Learning, Deep Neural Networks, Encoder-Decoder, Disparity estimation

ÖZET

GÖRÜNTÜ SINIFLANDIRMA İÇİN CNN MİMARLIKLARDA PERFORMANS GELİŞTİRME

Abdullah ABDULLAH
Bilgisayar Mühendisliği Yüksek Lisans Programı
Tez Danışmanı: Dr. Öğr. Üyesi Tarkan AYDIN
Ocak 2020, 77 sayfa

Bu tez, bilgisayarlı görme araştırmalarındaki en zorlu konulardan birini temsil ettiği için stereo görüş tabanlı derinlik tahmini sorununa odaklanmaktadır.

Son zamanlarda, derin öğrenme yöntemleri bilgisayar vizyonu araştırmacıları ve uzmanları arasında yaygın bir şekilde benimsenmiştir, bu nedenle ana hedef olarak kompaktlık ve eğitim kolaylığı olan derin bir öğrenme mimarisinin tasarımı ve uygulanması ile bu çabayı sürdürüyoruz.

Göreve seçilen Derin Sinir Ağı mimarisi Tamamen Konvolüsyonel Enkoder-Kod Çözücüdür ve eğitim verileri çıktı etiketleri olarak eşitlik haritalarına sahip stereo çift görüntülerdir, bu tasarımda alandaki en son tekliflerde ve yeni kompaktta birkaç değişiklik önerilmiştir ağlar eğitilir ve test edilir.

Anahtar Kelimeler: Bilgisayar Görüşü, Makine Öğrenme, Derin Sinir Ağı, Encoder-Decoder, Eşitsizlik Tahmini

CONTENTS

TABLES

FIGURES

ABBREVIATIONS

ANN	:	Artificial Neural Networks
ADAM	:	Adaptive Momentum estimation
BN	:	Batch Normalization
CPU	:	Central Processing Unit
CV	:	Computer Vision
CNN	:	Convolutional Neural Networks
DL	:	Deep Learning
DNN	:	Deep Neural Network
GPU	:	Graphics Processing Unit
ML	:	Machine Learning
MLP	:	Multi-layer Perceptron
NLP	:	Natural Language Processing
RAM	:	Random Access Memory
ReLU	:	Rectified Linear Unit
RMSPROP	:	Root Mean Square Propagation
RNN	:	Recurrent Neural Network
SGD	:	Stochastic Gradient Descent

1. INTRODUCTION

Stereo Vision is the hallmark of human perceptual capabilities, it enables a person to estimate depth and identify an object's distance by combining information from two eyes which represent the stereo input pair and deducing from them the all the useful information for fast and effective inference that allow us to perform feats such as avoiding collisions while walking or catching a flying ball.

These abilities that seem simple at first glance when required to be performed by computerized systems demands from specialists a great deal of diligence and design, extensive research has been allocated to the task of Stereo Vision and Stereo Matching, many proposals have been admitted that addressed this problem with good results albeit requiring large computational resources and noticeable delays in output.

The main aim of this thesis is to design a disparity mapping Deep Neural Network with the proprieties of small parameter size, ease of training and fast compute time by processing left and right stereo pairs as inputs and performing transformations on them to predict disparity values. These features are desired in systems that require fast execution with the best possible output fidelity, a prime example of such applications is the field of Robotic navigation.

The thesis starts by defining the theory of stereo vision and deep learning providing the foundations on which it proceeds to present past systems and designs given for solving the problem. It takes from that to specify the implementation for the proposed system of this work detailing design choices, procedures and modifications. The thesis is finalized with conclusion notes on the proposed implementation as well as the obtained test results.

2. THEORY AND LITERATURE RIVIEW

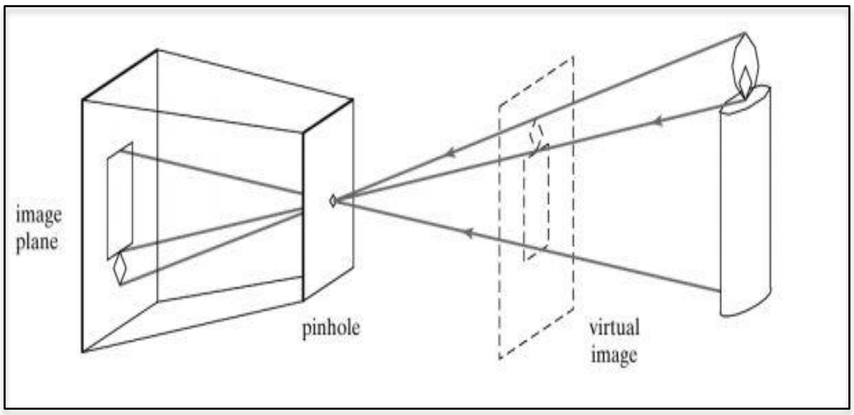
2.1 STEREO VISION

This part of the thesis presents the theory of Stereo vision starting with a model of how cameras operate, camera parameters intrinsic and extrinsic, stereo camera systems and lastly the stereo matching problem with a general theory of the proposed solutions.

2.1.1 Camera Model

The camera model of choice to describe the process of picture capture is the pinhole camera model (Szeliski 2010). It is a projection model that specifies the relationship between 3D points in the real world and 2D points on the image plane by tracing the path of light rays through the pinhole apparatus, the incoming rays converge on a single point called the camera centre from there these rays diverge onto the specific points that comprises the projected image.

Figure 2.1: The Pinhole Camera Model



2.1.2 Camera Intrinsic Parameters

Given a point in the real world with the three dimensional coordinates P = (X , Y, Z) the image plane projection of that point is given by the coordinates p = (x , y).

The equations for finding x and y (Szeliski 2010) are formulated as:

$$\mathrm{x} = f\,\frac{\mathbf{X}}{\mathbf{Z}}\ , \mathrm{y} = f\,\frac{\mathbf{Y}}{\mathbf{Z}} \qquad \textbf{(2.1)}$$

Where f represents the camera's focal length.

This transformation is unique but it is noted that performing the inverse is not, also noted that both the camera and the point of interest share the same coordinate system with the origin being the camera centre C.

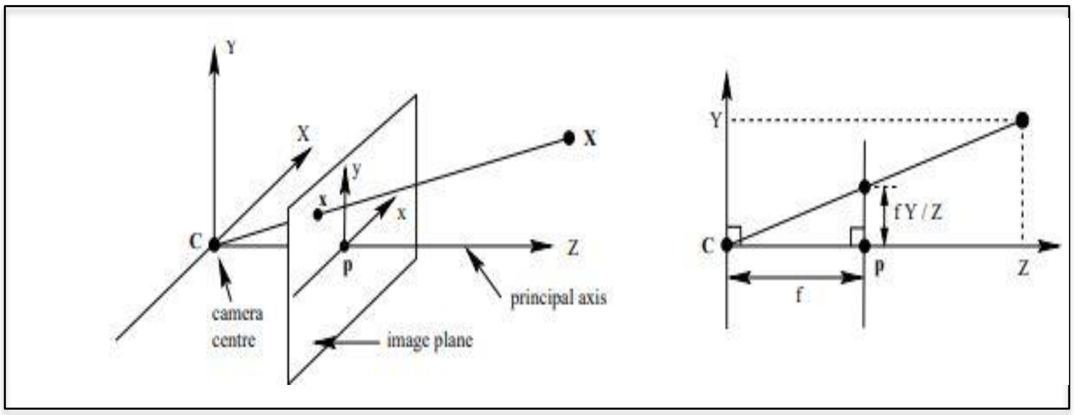
As for each image plane point there is a full line that could correspond to it thus it is deduced that capturing image projections with a single camera results in a loss of depth information.

The pinhole camera model (Szeliski 2010) can be defined in a matrix formulation as in the following:

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = k\ I \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \qquad \textbf{(2.2)}$$

Figure 2.2: Image Projection



The matrix k presented above is an approximation, physical digital cameras have active pixels not as mathematical points but with dimensions of width and height.
Manufacturing these pixels is not a perfect process and resulting pixels could have non-identical width and height dimensions as well as skew in positioning thus requiring extra parameters to accommodate these variations.

To account for aspect ratio difference between x and y dimensions of pixels the focal length is represented by two different parameters fx and fy.
If the image plane do not intersect directly with the principal axis offset parameters are introduced represented by cx and cy.
Skew in the elements placements are also accounted and represented by the parameter s.

The full k matrix with full form (Szeliski 2010) is presented as follows:

$$k = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

2.1.3 Camera Extrinsic Parameters

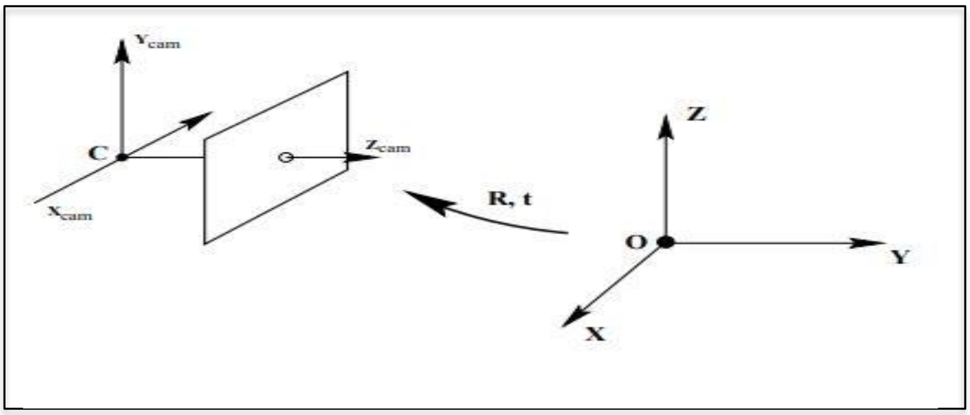
The coincidence of the origin point in the coordinate system that is related to the camera and the origin point in the coordinate system related to the object of interest is a highly idealized situation, mostly the camera will be in a different coordinate system with a different origin than the objects whose image being captured and this requires the introduction of extra parameters that calculate the transformations between the camera coordinate system and the coordinate system of real world objects.

These parameters are referred to as the camera extrinsic parameters (Szeliski 2010) and they are expressed in the following matrix form:

$$[R_{3\times3} \quad | \quad T_{3\times1}] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \qquad \textbf{(2.4)}$$

Where T is the matrix that encodes translations between camera origin and world origin while R is a matrix that encodes relative rotations between the two coordinate systems.

Figure 2.3: Camera to real world translations and rotations



With the extrinsic parameters matrix equation (2.2) can be modified to account for translations and rotations between camera and real world coordinates resulting in the following form:

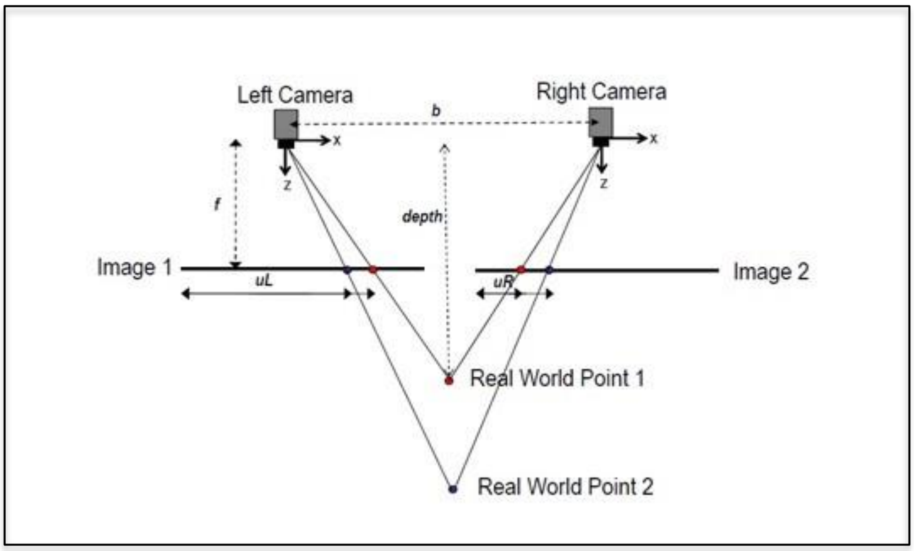
$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$= k \ [R_{3\times3} \quad | \quad T_{3\times1}] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.5}$$

2.1.4 Stereo Camera Systems

As the process of projection from world coordinates to image plane results in loss of explicit depth information the general concept of retrieving depth from two pictures taken for a single scene is now presented, this process in referred to as stereo vision.
Given two cameras with sensors that are aligned and coplanar to each other and having a certain separation distance in the plane of alignment, the rows of the images will correspond to the same lines on each sensor.
Having a point representing an object of interest on a certain location in one image plane such as the left camera will correspond to a horizontally shifted position on the image plane of the right camera, the measure of the shift is referred to as the disparity (Szeliski 2010), a near object to the two camera set will have a large disparity and vice versa thus establishing a relation between disparity measure and depth.

Figure 2.4: Disparity and depth form two cameras

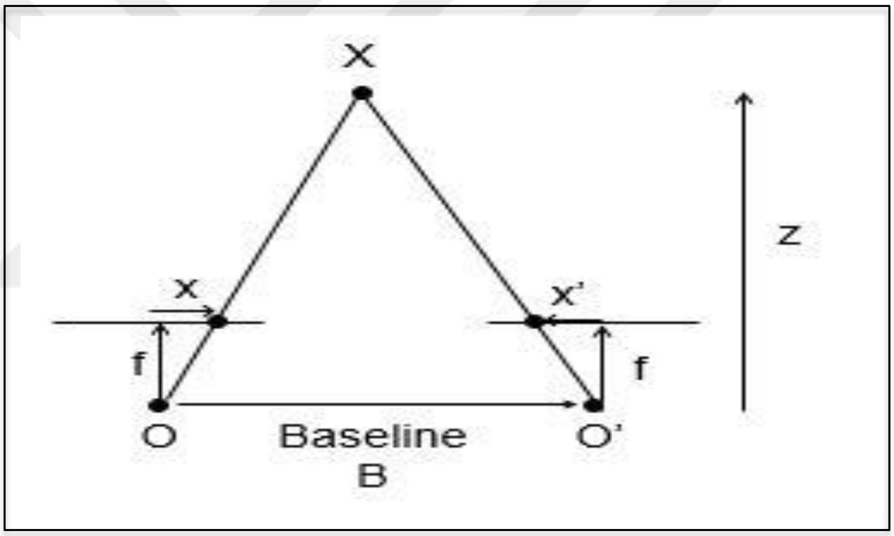


The two cameras that can be used to obtain a disparity measure must be identical as well as calibrated to have similar parameters, for two cameras with the same focal length f and having a separation distance also known as the baseline B the disparity (Szeliski 2010) can be defined for a certain point with left image plane location x and right image plane location x' as follows:

$$d = x - x' = B\frac{f}{z} \tag{2.6}$$

Where d is the disparity and Z is the real world depth.

Figure 2.5: Depth disparity relation

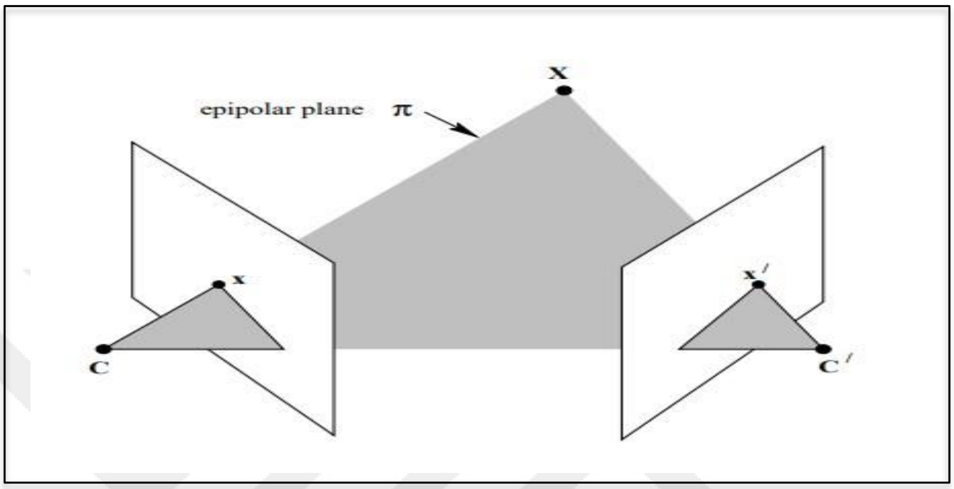


The relation between depth and disparity can be furthermore generalized to the case of non-alignment between the stereo cameras with the concept of Epipolar geometry (Szeliski 2010).

An Epipole is the point of intersection of the baseline with the image planes, considering that the baseline connects the optical centre of one camera to the optical centre of the other camera it can be deduced that an Epipole of a certain image plane is the projection of the optical centre of the other.

Using the two Epipoles and a real world point a plane embedded in three dimensions can be defined denoted as the Epipolar plane, the intersection of the Epipolar plane with each image plane defines two corresponding lines called the Epipolar lines.

Figure 2.6: Epipolar geometry



For a certain real world point X there is a line that connects it to each image centre and these lines bound the Epipolar plane and the intersection of each line with its corresponding image plane is the projection of the real world point on that image plane, the Epipole of a certain image plane is the intersection of all possible Epipolar lines of that image plane.

This geometric constraint can be used to deduce that the two corresponding image projections of a certain point must be located on the two corresponding Epipolar lines despite not being on the same horizontal line on the images due to camera misalignment.

With the Epipolar geometry specified the problem of disparity is a problem of locating the matching points that are situated on the corresponding Epipolar lines this requires correlating the object’s or part of the object’s pixels between those two lines given that the two cameras are calibrated and the relative transformation between them is at acceptable accuracy bounds.

This procedure of specifying the projection of an object or a point and searching for its corresponding position on the corresponding image plane with the constraint that it is on the corresponding Epipolar line is referred to as the procedure of stereo matching.

2.1.5 Stereo Matching

Given that the intrinsic and extrinsic parameters of the two cameras are known the projections in each image plane can be transformed into a normalized image coordinates thus the process of searching corresponding pixels of interest becomes a one dimensional search problem as the Epipolar lines for rectified stereo pair are located on the same rows.

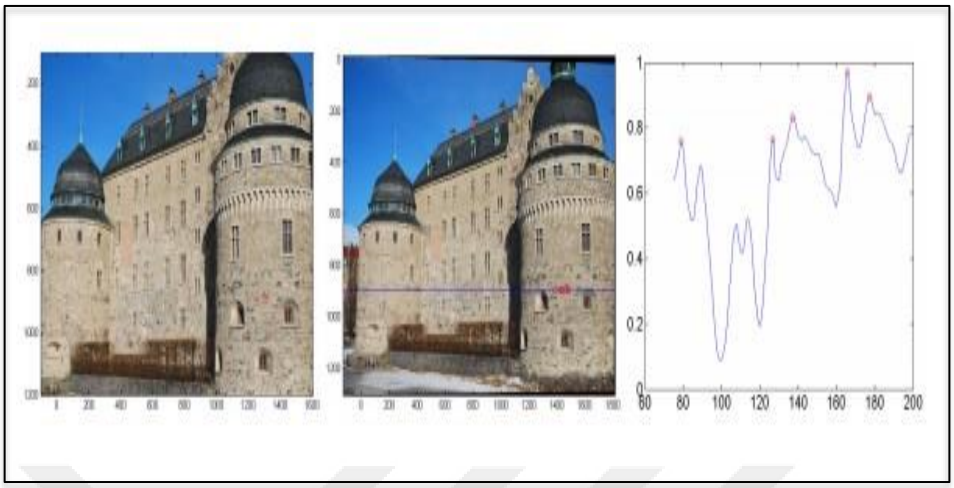
Performing the stereo matching procedure (Szeliski 2010) on each pixel results in a map known as the disparity map which is used to obtain the desired depth information to be used in later applications.

To build the disparity map a quantitative measure must be defined to find the correspondence, this measure is then required generally to be a similarity measure and is known as the matching cost.

The simplest matching cost is difference between pixel intensities with minimum difference being the result of correspondence search, this suffers from many drawbacks most notably the presence of multiple candidates for the matching pixel as well as minimal differences of illumination or presence of occlusions can result in the correct pixel having non minimal difference and not being included in the search results.

Practically other similarity measures are used such as sum of square distance or normalized correlation which are more resilient to noise and more effective as well as using windows to take into account the pixel’s neighbourhood in the process of correspondence.

Figure 2.7: Matching cost



More broadly techniques used in obtaining matching cost can be divided in two categories: local techniques and global techniques.

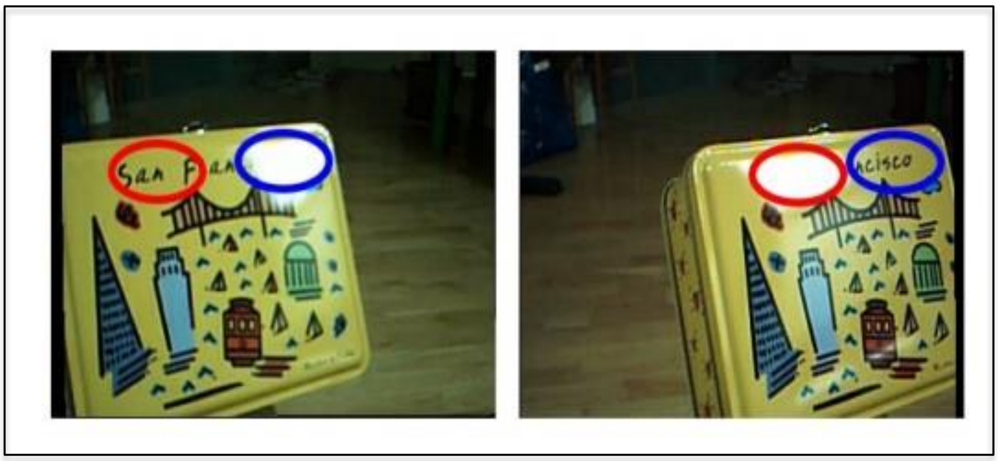
Local techniques as mentioned above use the pixel and its neighbourhood to obtain a matching cost measure and search for correspondence candidates while global techniques apply energy function minimization over the whole stereo pair.

The process of stereo matching faces several technical difficulties that add to its inherent complexity and reduce the quality of the obtained disparity map.

Such technical difficulties include:

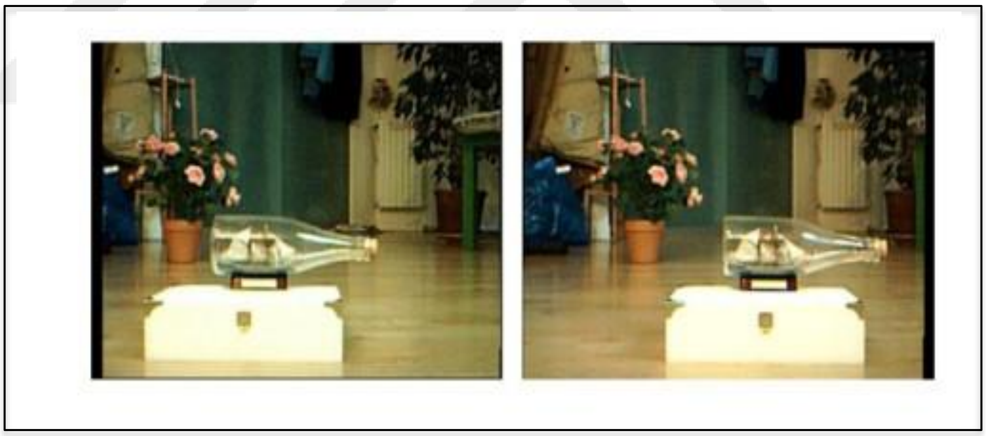
a. **Specular reflection and non-Lambertian surfaces**: where light reflected from objects has different intensities from different points of view or angles this effect is most noticeable from highly non-random surfaces called non-Lambertian surfaces leading to incorrect estimates on disparity.

Figure 2.8: Specular Reflection



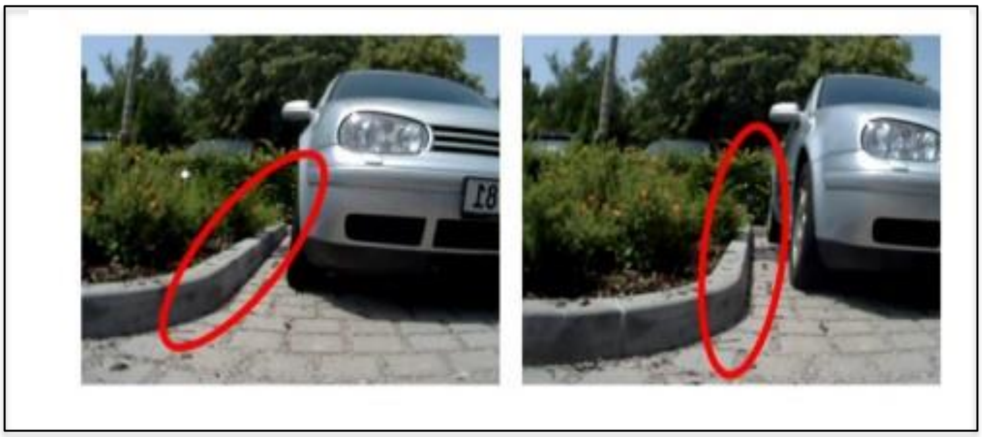
b. **Transparencies in objects**: leading to light from behind them to pass through and adversely affecting disparity estimates.

Figure 2.9: Transparencies



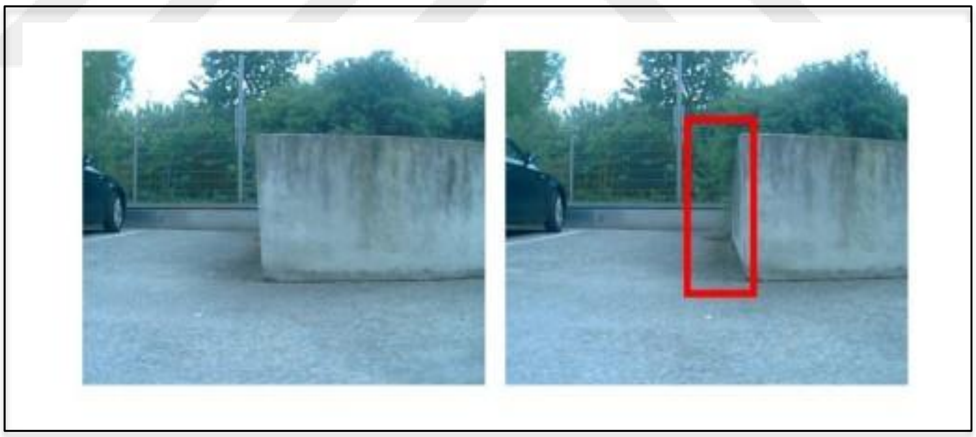
c. **Perspective change**: that leads to some objects to appear foreshortened.

Figure 2.10: Perspective change



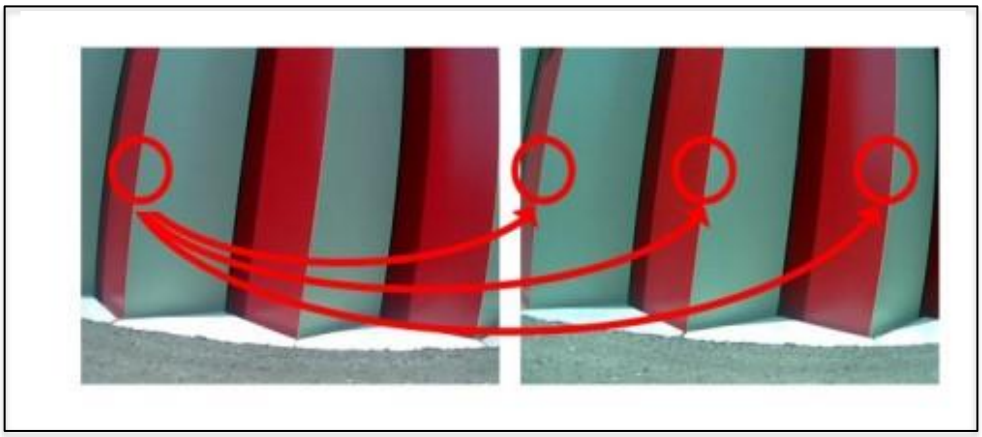
d. **Occlusions and discontinuities**: where full objects or parts of them are hidden from a certain view by another object while a discontinuity is a sharp change of pixel values due to certain geometries of the object.

Figure 2.11: Occlusions



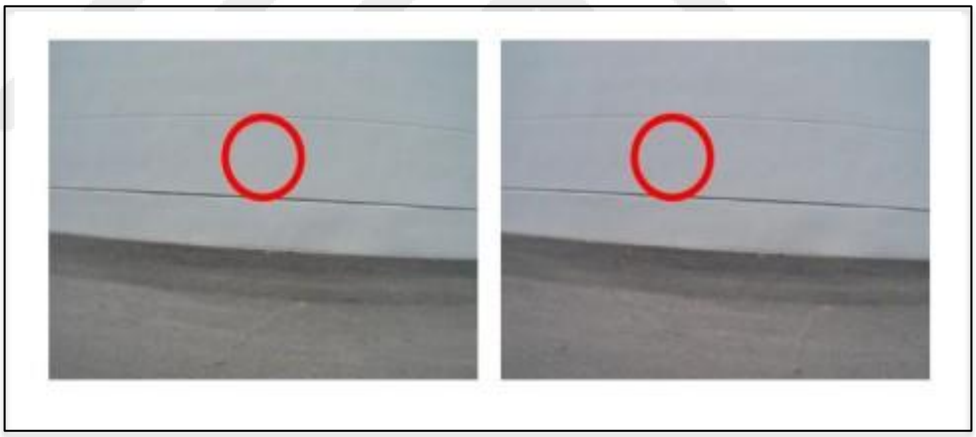
e. **Repetition of certain objects or patterns**: leading to difficulty in establishing single candidates from the cost measure.

Figure 2.12: Repetitive Objects



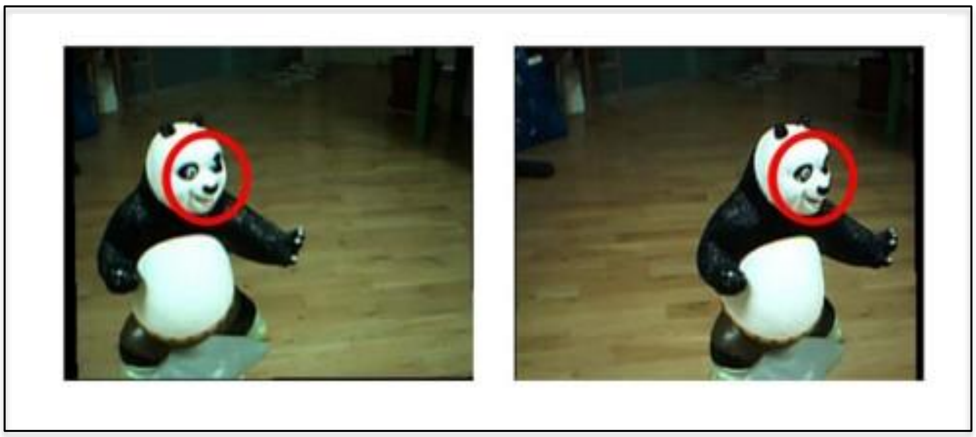
f. **Textureless surfaces**: results in difficulty of establishing correspondence due to similarity of pixel values

Figure 2.13: Textureless Surface



g. **Distortions and noise**: such as photometric distortions, faulty pixels and thermal noise.

Figure 2.14: Distortions



2.1 DEEP LEARNING

In this part a general theoretical introduction to deep learning systems is given starting with a definition of Neural Networks, Activation Functions, Deep learning, Convolutional Neural Networks and finally the Encoder-Decoder method.

2.2.1 Neural Networks

The human brain is a powerful information processing system capable of performing a variety of high complexity tasks such as speech recognition, motion planning as well as a plethora of vision related operations.

Artificial Neural Networks (ANN) are networked models that took inspiration from the human brain and was adapted for the construction of many practical algorithms that initiated a new era of computation.

Computationally ANN’s can be considered as a distributed, high connectivity and parallelized system with learnable parameters thus not requiring programming and therefore providing large saving of effort.

The Perceptron (Rosenblatt 1958) is the elementary building unit used in constructing Neural Networks, this unit receives inputs that may come from the data source or from other Perceptrons in the network.

Each input denoted by the vector xj where j = 1,… , d has a specific “weight” value associated to it denoted by wj and the most general form for calculating the output y is by taking the sum of the weighted inputs:

$$y = \sum_{j=1}^{d} w_j x_j + w_0 \quad \textbf{(2.7)}$$

W_0 is the bias value which represents an intercept that generalizes the model and its value is set to +1.

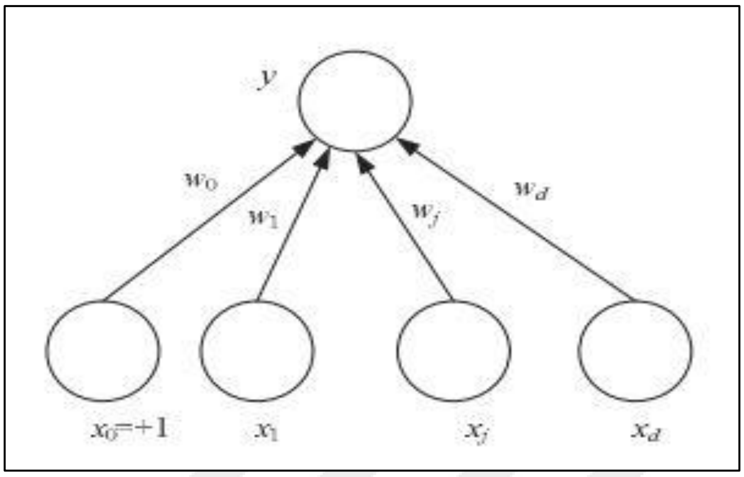
The dot product form of the above equation is presented as:

$$y = W^T X \quad \textbf{(2.8)}$$

Where $W = [w_0, w_1, \ldots, w_d]^T$ and $X = [1, x_1, \ldots, x_d]^T$

For the case of a single input the equation reduces to a line equation with slope w and intercept w_0 thus a perceptron thus its output is a linear fit.
With many inputs the equation generalizes to a multivariate fitting and the line becomes a hyper-plane in the multidimensional space of the inputs.

Figure 2.15: The Perceptron



To obtain a linear discriminant from the output of a perceptron a threshold function can be implemented as:

$$s(x) = \begin{cases} 1, & x > 0 \\ 0, & otherwise \end{cases} \tag{2.9}$$

So that the class choice reduces to:

$$choose = \begin{cases} c_1, & s(W^T X) > 0 \\ c_2, & otherwise \end{cases} \tag{2.10}$$

Generally a probability measure is needed thus a sigmoid function is used on the output of the perceptron as:

$$y = sigmoid(W^T X) = \frac{1}{1+\exp-[W^T X]} \tag{2.11}$$

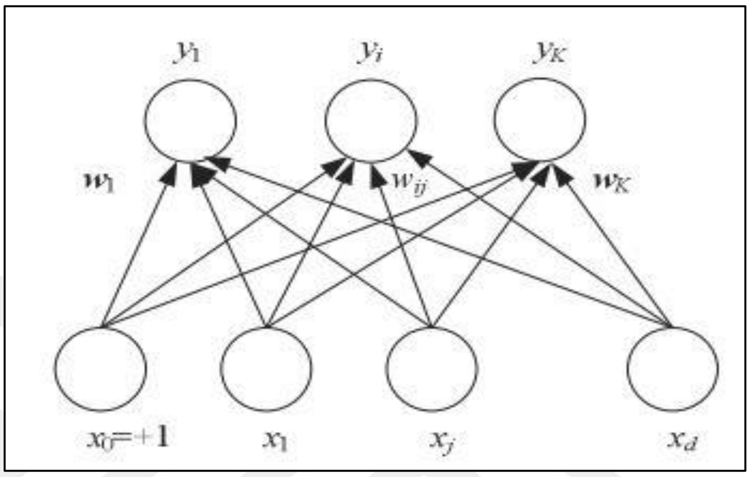
A perceptron can have multiple outputs instead of a single output this in particular useful for discriminating multiple classes of object with linear hyperplanes.
Given that the output are represented by the vector yi running from 1 to k the general equation of the perceptron is modified to:

$$y_i = \sum_{j=1}^{d} w_{ij} x_j + w_{i0} \tag{2.12}$$

Or in the dot product form as:

$$y_i = W_i^T X \tag{2.13}$$

Figure 2.16: Multi Output Perceptron (Multi-Class)



In the simple terms the desired class is the one with maximum output yi but as with the single output case a probability measure is more preferred and this is calculated using the softmax function as:

$$y_i = \frac{\exp\,[W_i^T X]}{\sum_k \exp\,[W_k^T X]} \tag{2.14}$$

Where the index k runs through all the outputs representing the list of classes and the softmax function provides the posterior probability of each class.

After defining the perceptron as a linear model describing a hyperplane in the input space of the data then a fitting must be imposed on that hyperplane in order to minimize error on prediction, this process is termed “training” of the perceptron.

Generally to train a perceptron a data vector termed an “instance” is drawn from the given data set to be fitted and the properties of this data vector known as the “features” are presented to the model as inputs to obtain the “predicted” value which then is used to obtain an error measure provided that the data has the actual desired output value, the connections of the perceptron represented by its “weights” are then updated according to that error measure.

If the error measure is a differentiable function then gradient descent can be used to minimize it and this process of gradual change of parameters with each instance is called Stochastic Gradient Descent (SGD) (Kiefer and Wolfowitz 1952).

The update rule for the model weights with SGD for an instance of index t with input features xit , label rt and prediction yt is given by:

$$\triangle w_j^t = \eta(r^t - y^t)x_j^t \tag{2.15}$$

Where η is the learning factor and Δw_i^t is the weight update of connection *i*

Generally the update rule can be expressed as:

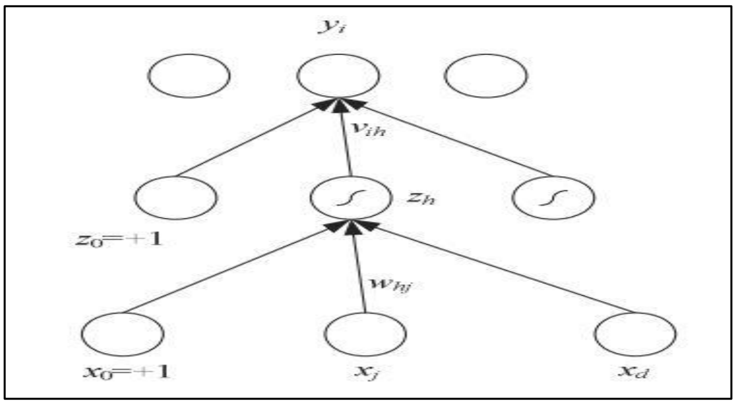
Update =Learning Factor · (Desired Output – Predicted Output) · Input

The general algorithm for (SGD) is given as follows:

```
For i = 1,…,K
    For j = 0,…, d
        $w_{ij} \leftarrow rand(-0.01,0.01)$
Repeat
    For all ($x^t$,$r^t$) ∈ X in random order
        For i = 1,…,K
            $o_i \leftarrow 0$
            For j = 0,…,d
                $o_i \leftarrow o_i + w_{ij}\ x_j^t$
        For i = 1,…,K
            $y_i \leftarrow exp(o_i) / \sum_k exp\ (o_k)$
        For i = 1,…,K
            For j = 0,…,d
                $w_{ij} \leftarrow w_{ij} + \eta(r_i^t - y_i^t)x_j^t$
Until convergence
```

A single layer Perceptron can only implement linear discriminants thus it will not be able to solve nonlinear problems which necessitates the use of a nonlinear estimator.
This nonlinearity can be introduced with adding extra “hidden” layer between the inputs of the perceptron and the output calculation, this layer has units with weights and bias as in the previous manner and its calculations are passed into activation functions.
The activation values pass forwardly from hidden units to the output therefore this type of perceptron represents a “feed forward” model and the general name for a perceptron with a hidden layer is a Multi-Layer Perceptron (MLP).
It is notable that if the hidden units are linear the output will be a linear combination of linear models and the hidden layer will be limited in capability and unable to model nonlinear problems thus several nonlinear differentiable “activation functions” are applied in the hidden units such as the sigmoid activation functions and hyperbolic tangent or “tanh” activation function, these activation functions perform a nonlinear transformation on the input vector mapping it from the D-dimensional space of input features to H-dimensional space where H spans the number of hidden units and providing nonlinear basis to be combined by the output layer calculation.

Figure 2.17: Multi-Layer Perceptron (MLP)



For the value of each hidden layer activation zh with h indexing the hidden units and running from 1 to H the calculation of the sigmoid activation is given by:

$$z_h = \frac{1}{1+exp-[\sum_{j=1}^{d} w_{hj}x_j+w_{h0}]} \quad \textbf{(2.16)}$$

If v_{ih} is the hidden to output connection weights then the output of a one hidden layer MLP is given by:

$$y_i = \sum_{h=1}^{H} v_{ih}z_h + v_{i0} \quad \textbf{(2.17)}$$

MLP's are trained using SGD as with the single layer Perceptron but since there is a hidden layer with activation function the gradient calculation will be modified into a form of chain rule of error propagation from output through the hidden units as in the following equation:

$$\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial z_h}\frac{\partial z_h}{\partial w_{hj}} \quad \textbf{(2.18)}$$

The process of calculating error gradient by chain rule for MLP is termed "Backpropagatoin" (Rumelhart, Hinton and Williams 1986) as the error is propagated inversely through the network from a layer to the previous.

The general algorithm for Backpropagation is given as follows:

```
Initialize all $v_{ih}$ and $w_{hj}$ to $rand(-0.01,0.01)$
Repeat
    For all $(x^t,r^t) \in X$ in random order
        For h = 1,...,H
            $z_h \leftarrow sigmoid(\boldsymbol{w}_h^T\boldsymbol{x}^t)$
        For i = 1,...,K
            $y_i = \boldsymbol{v}_i^T\boldsymbol{z}$
        For i = 1,...,K
            $\triangle v_i = \eta(r_i^t - y_i^t)\boldsymbol{z}$
        For h = 1,...,H
            $\triangle w_h = \eta(\sum_k(r_i^t - y_i^t)v_{ih})z_h(1 - z_h)\boldsymbol{x}^t$
       For i = 1,...,K
            $v_i \leftarrow v_i + \triangle v_i$
       For h = 1,...,H
            $w_h \leftarrow w_h + \triangle w_h$
```

Until convergence

A Multi-Layer Perceptron (MLP) can have more than one hidden layer and can be designed with several hidden layers that provide multiple hierarchical mappings which are used to extract the most relevant information representation to achieve best performance results thus another way to think about ANN's is to consider them as hierarchically connected representation machines.

Each layer can have another design factor which is the number of units "nodes" inside each layer giving designs with wide layers or narrow layers and in that light it is considerable to mention that an MLP with one hidden layer and arbitrary number of hidden units is able to learn the representation of any nonlinear function of its inputs, this property of MLP is called the Universal Approximation Theorem (Hornik et al. 1989) where the aforementioned MLP is a Universal Approximator of the data representing function.

Generally by increasing the number of layers in the network better and more robust practical results can be achieved and this design parameter is referred to as the depth of the network, with sufficiently large number of layers more rich representations can be learned capable of performing on very high dimensional data from a variety of sources.

These networks started to be widely preferred and their design and development became a rich and active field with the name of Deep Neural Networks (DNN).

2.2.2 Deep Learning and Deep Neural Networks

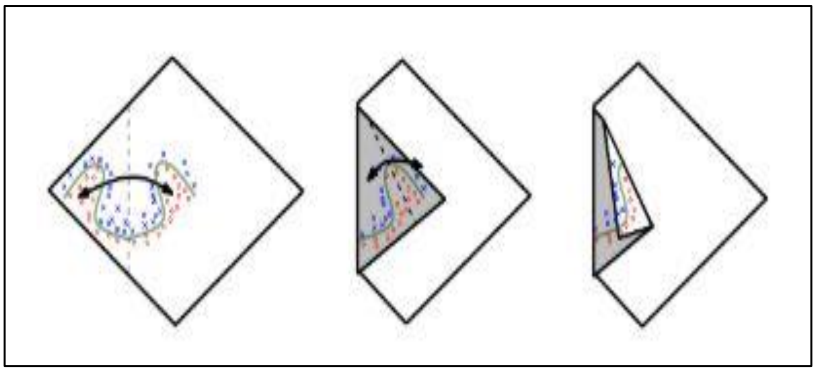
A Multi-layer Perceptron with increasing number of layers is referred to as a Deep Neural Network with the general field being referred to as Deep Learning (DL) which has a focus on the research of high "depth" hierarchical representation learning systems.

This move into deep models was driven by the limitation traditional machine learning (ML) approaches faced which required deliberate data manipulations and preprocessing to obtain relevant features to be provided as an input to "shallow" models and obtain good practical performance, that resulted in a requirement of analysing data with a great deal of domain expertise in order to extract suitable input vector choices and with exponential increase in data size the effort becomes exponential added to that the variety of problems that are hard to formalize effectively such as recognition of spoken words or recognition of faces in images.

Deep Learning techniques as mentioned provide compounded representational learning capable of finding sophisticated mappings from input space and when applied on raw data shows the ability to extract the most relevant features by the process of compounding simple concepts into more complex concepts.

A geometric picture of the efficacy of Deep Learning comes from considering the mapping process as defining symmetry lines where the problem space can be compacted into a less complex representation and with many layers this process is repeated with sufficiency for recovering the representation suitable for the required task.

Figure 2.18: The Geometric representation of Deep Learning



Deep Learning is now a valuable tool in many science and engineering tasks, most notably are computer vision (CV), Robotics, audio processing, natural language processing (NLP), search engines and finance.

Deep Learning architectures can be trained in both supervised and unsupervised learning settings as well as reinforcement learning.

In supervised learning all the data vectors are provided with the actual desired output value also known as the “ground truth” and the process of learning is performed with the calculating the SGD in a Backpropagation algorithm.

While some data sets can have all its samples with labelled output ground truth other data sets can have only a subset of it labelled and training in this setting is referred to as semi-supervised learning.

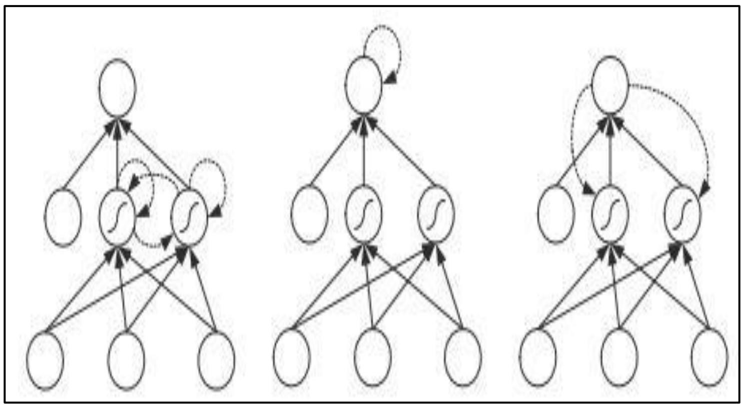
Unsupervised learning is a setting where no data vector is labelled and the model must learn from the implicit correlations within the data as well as some measures that can be applied to the error signal to obtain performance indicators.

Reinforcement learning is a setting where the networked is given a logical or numeric feedback value after completing several tasks to gauge the model performance and perform adjustments on the network values which defines a certain “policy” on the tasks given in the setting.

Deep Learning networks can be architected in two broad categories either as illustrated with prediction computation going from input through the layers until the output computation and this is termed as Feed Forward” architecture.

Or in addition to the feed forward connections the architecture can have connections directing from a unit to itself or to units in previous layers in a recursive manner, these connections provide a mechanism similar to memory and the architecture is referred to as Recurrent Neural Networks (RNN).

Figure 2.19: Recurrent Neural Networks (RNN)



In the feed forward architecture all the units in hidden layers can be connected to all the units in the next hidden layers, this architecture is called “fully connected”.
To reduce computational burden especially in image processing another architectural choice is employed that takes advantage of the mathematical operation of convolution and the design known as Convolutional Neural Networks (CNN) gained a wide spread adoption in computer vision and image processing tasks due to its notable result improvements and comparative ease of implementation.

2.2.3 Activation Functions

Most tasks utilize Deep Learning algorithms are high complexity tasks with nonlinear data distributions, this as illustrated required that the hidden units in Deep Neural Networks apply nonlinear transformations on the weighted summation results of its learnable parameters which are known as activation functions.
Many activation functions were proposed for Neural Network models each with its specific transformation that achieves certain requirements or adheres to certain considerations practical or theoretic with the most important quality of being a differential function in order to be compatible with the process of backpropagation.

Some of these functions include:

a. Sigmoid: the Sigmoid activation function also known as the logistic function or squashing function is one of the most implemented activation function in feedforward architectures.

   The Sigmoid function is defined on the reals with positive derivative and good degree of smoothness, the Sigmoid function is given by:

$$f(x) = \frac{1}{(1+e^{-x})} \tag{2.19}$$

   When Sigmoid is used in output layer of a model it provides probability on predicted output, this activation function has the benefit of easy to understand and implement but there are drawbacks of slowing convergence and gradient saturation.

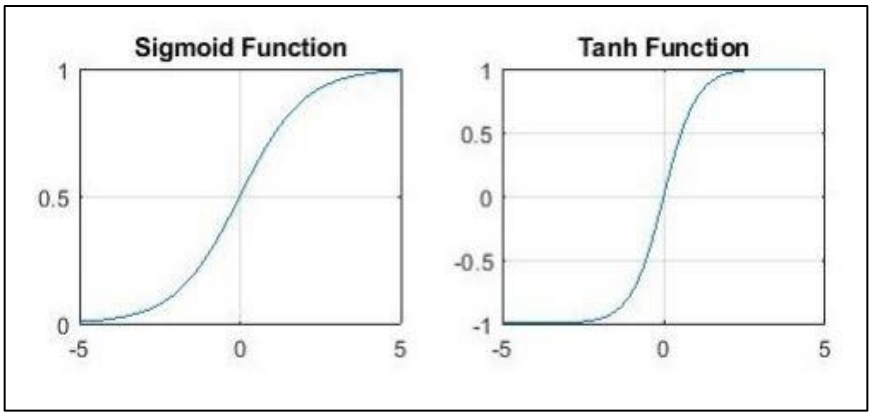
b. Hyperbolic Tangent (tanh): this activation function is more smooth than the Sigmoid and have outputs ranging from -1 to 1, it provides better performance than Sigmoid in deep models and it is given by the equation:

$$f(x) = \frac{(e^{x}-e^{-x})}{(e^{x}+e^{-x})} \tag{2.20}$$

   While the tanh activation function has zero centred outputs increasing compatibility with the backpropagation process it has a vanishing gradient drawback.

   The Hyperbolic Tangent is mostly used in recurrent architectures and natural language processing tasks.

Figure 2.20: Sigmoid and Tanh activation Functions



c. Softmax: this function provides probability measure on several outputs and is applied on the output layer of models with multi-class classification.

   The equation describing the Softmax function is given by:

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.21}$$

   Where i is the index of a specific class or output and j spans the class list or outputs.

   This activation function is similar to Sigmoid as both are probability calculations where the Sigmoid is a binary classification related while the Softmax is multivariate classification related and since both use divisions and exponentials in the implementation the general behaviour is highly similar.

d. Rectified Linear Unit (ReLU): the (ReLU) function (Nair and Hinton 2010) is a widely used and very successful activation due do its performance gains and robustness against drawbacks that affected other activation functions.

   This activation function is much faster as well as preserving linearity on a large portion of the function span aiding in the gradient calculations and as its name suggesting it applies a rectification on values less than zero eliminating the vanishing gradient issue.

   The equation describing the ReLU activation is given by:

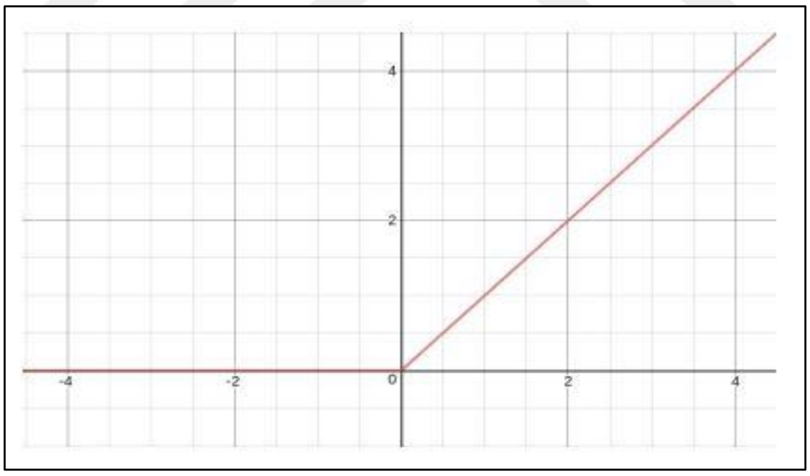
$$f(x) = \max(0, x) = \begin{cases} x, & if\ x \geq 0 \\ 0, & if\ x < 0 \end{cases} \qquad \textbf{(2.22)}$$

This activation functions with its simplicity requiring no calculation of divisions of exponentials greatly improve the speed of the model while also introducing squashing and sparse activation but the main drawback is the propensity to overfitting and producing zero gradient resulting in either rigidity in model or the issue of dead neuron units.

Another version that augments the positive portion of the function with negative slope is also implemented which keeps the units "alive" with sustainable weight update and is referred to as LeakyReLU (Maas, Hannun and Ng 2013).

Rectified Linear Unit activation function is used extensively with Convolutional Neural Networks (CNN).

Figure 2.21: ReLU Activation Function



In addition to the mentioned activation functions several other modifications and enhancements have been implemented that address the illustrated drawbacks or introduce enhancements in speed, some of these include Sigmoid-Weighted Linear Unit (SiLU) (Elfwing 2017) and derivative of Sigmoid-Weighted Linear Unit (dSiLU) (Elfwing 2017).

Other functions have been proposed which offer nuanced approaches to address the difficulties of nonlinear activation such as (Swish) (Ramachandran et al. 2017) which is a hybrid of the input and its sigmoid, (Maxout) activation functions) (Goodfellow et al. 2013) that extends the learning process to the activation function in addition to the traditional connection parameters by a process of piecewise approximation to nonlinear functions which are optimized by gradient during training to provide the best activation that is dictated by the performance.

2.2.4 Convolutional Neural Networks

Convolutional Neural Networks (CNN) (LeCun et al. 1989) are a specialized architecture of the feed forward category adapted to data that can be expressed in grid like form such as images which can be thought of as a 2D grid thus instead of having the fully connected network converge on a suitable connectivity that is most performing on such data structure, the structure itself can drive design choices to increase model efficiency.

CNN's represent also a prominent case of biology and neuroscience influencing design of successful algorithms in which the study of the human visual cortex specifically the V1 area of the vision system known as the primary visual cortex provided the information central to the formulation of the convolutional architecture.

Several properties of the V1 system are present in the general scheme of CNN primarily V1 is arranged as a two dimensional spatial map with certain parts of the retina corresponding to similar spatially located regions in V1 and this is employed in CNN as it extracts two dimensional maps from data.

V1 shows groups of simple cells having linear activations of only small regions with spatial locality and CNN employ this property of local receptive fields.

There exist in V1 complex cells that capture features with an important property of invariance on shifting the position of the specific feature in the map, this feature is also present in CNN architecture with the operation of pooling units.

The main mathematical operation used in the transformation performed by CNN's is the Convolution operation which has the general form:

$$S(t) = \int x(a)w(t-a)da \tag{2.23}$$

The convolution operation can be denoted by an asterisk symbol with the equation adopting the form:

$$S(t) = (x * w)(t) \qquad \textbf{(2.24)}$$

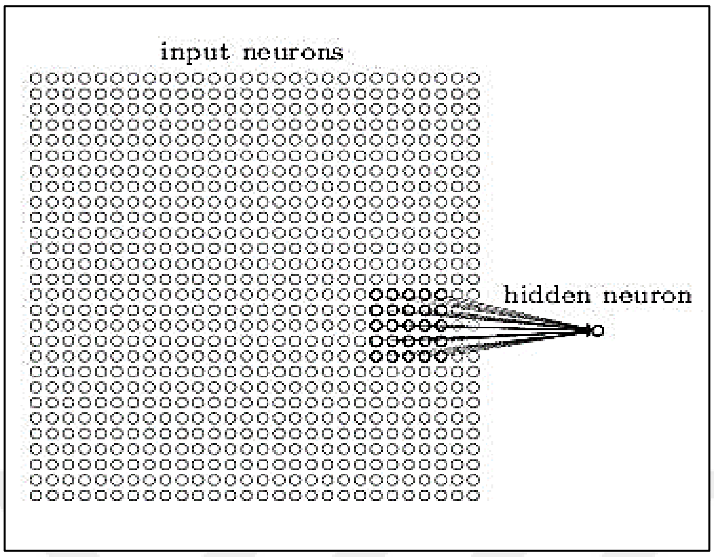
The variable x is taken to be the input of the operation while w is referred to as the kernel of the convolution or by the term feature map.
Considering that in Deep Learning the data processed by CNN’s are discrete valued usually with multiple indices “grids” and running only on finite ranges the more practical form of convolution operation is written as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \qquad \textbf{(2.25)}$$

Where i,j are indices of the input and m,n are kernel indices referred to as kernel size.

As mentioned in this thesis Convolutional Neural Networks (CNN) are motivated by design considerations that mimic the brain’s primary visual cortex these considerations are summarized in three principles: local receptive field, weight sharing and pooling.
The fully connected Neural Network contains an connectivity parameter for each input pixel to each unit in the subsequent hidden layer while Convolutional Neural Networks in contrast adopt a principle of sparse connectivity by implementing the concept of local receptive fields, this is achieved by having a kernel that is significantly smaller than the input image dimensions reducing the parameter requirement since the features to be extracted are highly local such as lines and edges it is considerably inefficient and greatly difficult to train a fully connected network to detect these features.

Figure 2.22: Local Receptive Field

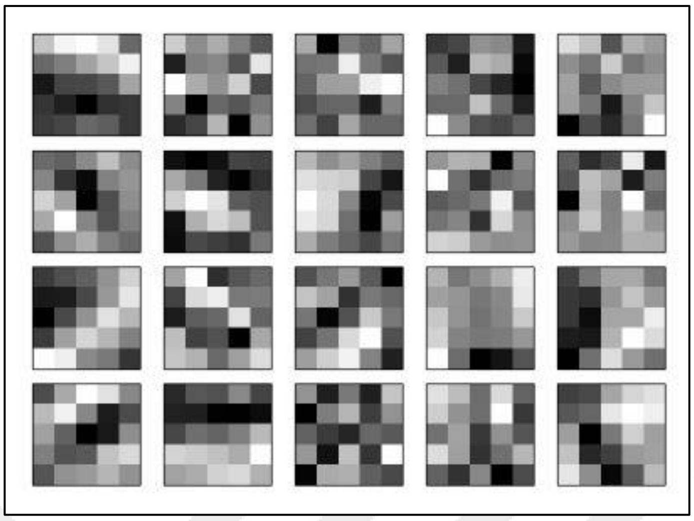


Each hidden neuron in CNN’s is connected to the input by a kernel that has a small number of learnable parameters improving the efficiency as well as the convergence characteristics of the network, this local receptive field of a certain kernel connecting a specific hidden neuron is shifted on input compared to the previous hidden neuron with a number of pixels referred to as the stride of the CNN thus it is seen that the result of a convolutional layer is also a grid with each region correlated to a corresponding portion of the input.

Another principle contributing to improving performance of CNN’s is weight sharing which is a process where the kernel parameters of some or all hidden neurons in the subsequent hidden layer are kept the same, this reduces the storage requirement for the resulting model and since the output of learning in such a process is a grid of units with parameters that activate the artificial neuron on the presence of a certain pattern such as a specifically oriented line of edge the layer is thus called a feature map as it maps the input to another representation on the presence of a certain feature.

For each stage in a CNN several of these feature maps can be computed and the choice of the number of feature maps in a certain stage is decided by the designer.

Figure 2.23: Feature Maps



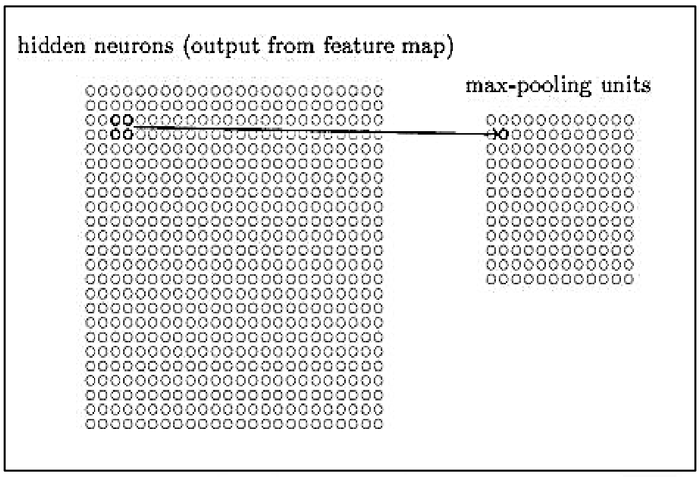
The outputted feature maps has a size comparable to the previous layer and implementing the full CNN in this manner will result in a redundancy of the representation although the aim of the network is to detect certain objects or patterns to be used by a later classifier or another transformation.

Pooling is a design principle of Convolutional Networks that acts as an information simplification process and when applied after each convolutional stage it reduces redundancy by performing what is analogous to a statistical summarization.

As an illustration “Max Pooling” is the most used pooling mechanism by practitioners and this layer outputs the maximum value that coincides in a certain rectangular region other types of pooling include “Average Pooling” which outputs the average value of the rectangular window and “L2 Pooling” that calculates the square root of sum of squares for the window values.

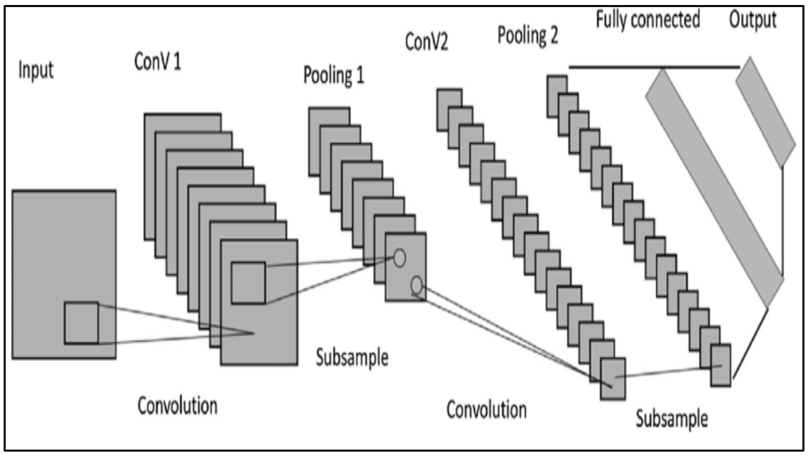
Performing the Pooling Process also known as “Sub-sampling” Takes a statistical summary of the mapping in feature maps, this besides filtration of the information redundancy provides also an extraction of the targeted features regardless to the localization or orientation of that feature in the input and providing an important property of invariance in which small perturbation in the target object’s position still do not affect the capability of the model to detect its presence.

Figure 2.24: Max Pooling Layer



After performing several stages of convolutional, activation and pooling layers to the specification of the desired architecture generally a fully connected network is used as the end receptor of the feature maps in order to perform classification or regression tasks depending on the application.

Figure 2.25: CNN General Architecture



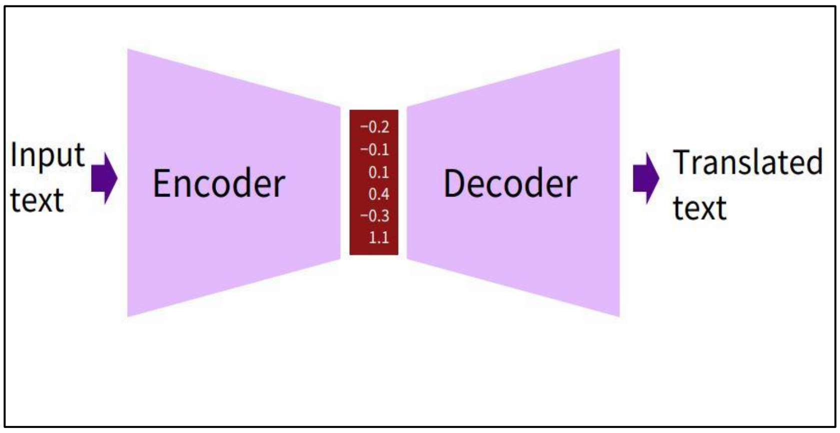
On image processing applications several CNN architectures have been implemented such as LeNet (LeCun 1998),AlexNet (Krizhevsky, Sutskever and Hinton 2012), VGGNet (Simonyan and Zisserman 2015), Inception (Szegedy et. al. 2014) and ResNet (He et al. 2015a).

These architectures among others in addition to the general design principles mentioned in this section apply several other modifications accompanied by the use of deeper models trained with massive compute capability in data centres on large bodies of data provided by the current data explosion sourced from social networks.

2.2.5 Encoder-Decoder Architecture

In Neural Networks data features are subjected to transformations in order to extract relevant information, this property of embedding the data vector from input space to another representation is the design principle behind the
Encoder-Decoder architecture (Ronneberger et al., 2015) which takes the input such as a text in vector form and performs an Encoding into a contextual representation that minimizes redundant information and this representation is then passed into a Decoding process that returns the embedding into a from comparable to the input or into a specific space that the application requires as an example the contextual embedding of the text vector can be mapped into a translated text vector of another language.

Figure 2.26: Encoder-Decoder Architecture

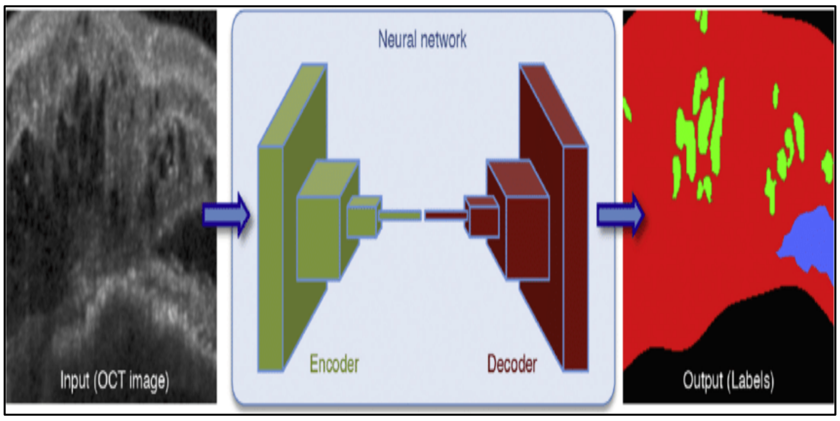


Encoder-decoder architectures can be implemented with fully connected Neural Networks either in feed forward or recurrent variants which are applied in machine translation tasks but this design principle is also compatible with the CNN implementations in which it is applied to solve problems of extracting relevant representation from images to contextual embedding space for tasks such as Semantic Segmentation, Object Detection and Stereo Vision.

In CNN's the encoder applies successive restriction on feature map size to extract the most informative kernels representing the embedding then the decoder performs the suitable transformation to map these kernels to the output space an example on this is "De-noising Auto Encoders" (Bengio et al., 2013) that are a form of unsupervised learning where the input and target are the same image and the Encoder-Decoder maps to a form of the input with noise removed as the information extraction attenuates the non-representative parts of the image and decoder stages reconstruct only the representative information back into an image.

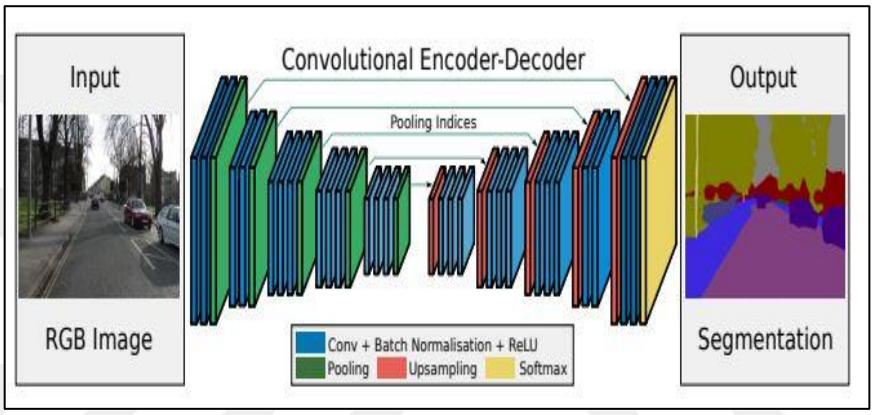
The same technique can be used in a supervised context where the input images are to be mapped to an output image with some modifications or transformations where some portions to be highlighted or subtracted or another grid representation is needed for a specific application an example for such an implementation is in the medical sector where images taken by radiology methods can be used in conjunction with expert knowledge in labelling them for the identification of healthy and tumorous tissue, an Encoder-Decoder CNN can be then trained to perform this process by the mentioned information extraction and re-representation to automatically label radiology images.

Figure 2.27: Encoder-Decoder Application Example



In Computer Vision tasks several issues still impose a challenge such as variability of lighting, variability of object's pose, occlusions and contextual as well as temporal dynamics, Encoder-Decoder Convolutional Networks perform transformations from inputs to representational embedding and reconstruction from that embedding thus it is suitable for it to be utilized to address the issues of Computer Vision applications and to illustrate such an adoption (Segnet) (Badrinarayanan 2015) is an example of an Encoder-Decoder CNN that performs End-to-End Deep Learning with the use case of semantic segmentation.

Figure 2.28: Encoder-Decoder CNN for Computer Vision



2.3 LITERATURE RIVIEW

In this part a review of the literature and related work to the aim of the thesis is presented.

The problem of stereo vision and stereo matching being one of the most challenging tasks in the field garnered the attention and effort of the researchers throughout the years, classical stereo matching solutions where based on hand crafted algorithms either through local methods that calculate correspondence between small image patches by minimizing a cost function such as sum of square distance (SSD) (Hannah 1974) and Normalized Cross Correlation (NCC) (Lewis 1995) or global approaches that use some assumptions on the full image to obtain a disparity estimate such as Semi-Global Matching (SGM)(Hirschmueller 2005) and Markov Random Field (MRF)(Zhang and Seitz 2007).

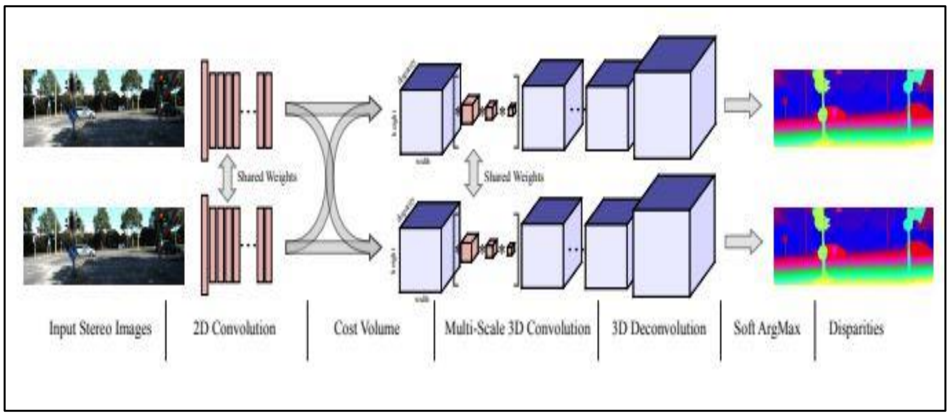
Recently Deep Learning methods and implementations were on the rise in solving the general stereo vision tasks such as optical flow, object reconstruction, orientation and most importantly depth estimation.

For depth and disparity mapping two main approaches were implemented either by mimicking the local hand crafted methods and their operations or methods that utilize the full capability of Neural Networks to extract semantic based depth estimation on full images.

Local patch methods compute the estimate from a certain image patch and a corresponding patch after applying a CNN or an MLP to convert the patch data into feature maps, examples of these approaches are MC-CNN (Zbontar and LeCun 2015) and Content-CNN (Luo, Schwing and Urtasun 2016) these approaches are still computationally expensive and not trainable end to end as several calculations must be done to obtain the desired disparity map including calculating cost aggregation, left to right consistency checking as well as interpolation.

End to end disparity estimation using Deep Learning rely on the ability of deep CNN's to extract the relevant information to obtain the depth estimate, these approaches use a CNN architecture analogous to autoencoders that consist of an encoding stage of both left and right images followed by a decoder stage to the target map. Examples of these approaches are DispNet (Mayer, Ilg, Hausser, Fischer, Cremers, Dosovitskiy and Brox 2016) and GC-Net (Kendall, Martirosyan, Dasgupta, Henry, Kennedy, Bachrach and Bry 2017) in which the left and right images of the rectified stereo pair are passed into a Siamese CNN encoder that contain convolutional layers with weight sharing between the left and right paths, GC-Net also use a Cost Volume calculation using 3D convolution.

Figure 2.29: GC-Net architecture



Since these methods contain a large number of layers with large number of parameters in addition to the computational burden during training there is also the issue of vanishing gradients and to address that GC-Net and DispNet use connections from lower layers to high level layers and in the case of DispNet an auxiliary loss function is also employed to prevent gradient decay.

DenseMapNet takes another take on the issue that is based on the DenseNet (Huang, Liu, Weinberger and van der Maaten 2017) architecture in which previous layer inputs are concatenated with the next layer inputs to address the gradient vanishing and another feature of DenseMapNet (Atienza 2018) design is the emphasis on obtaining an efficient implementation with small number of parameters preventing over fitting on small data sets.

A method that is widely used in CNN computer vision implementations to reduce parameter size is to convert the regular convolutional layers into Depthwise Separable Convolution layers this greatly reduce the computational requirements for training the architecture leading to more efficient designs.

3. DATA AND METHOD

This part presents the data sets chosen for the work and the Deep Learning proposed implementations.

3.1 DATA SETS

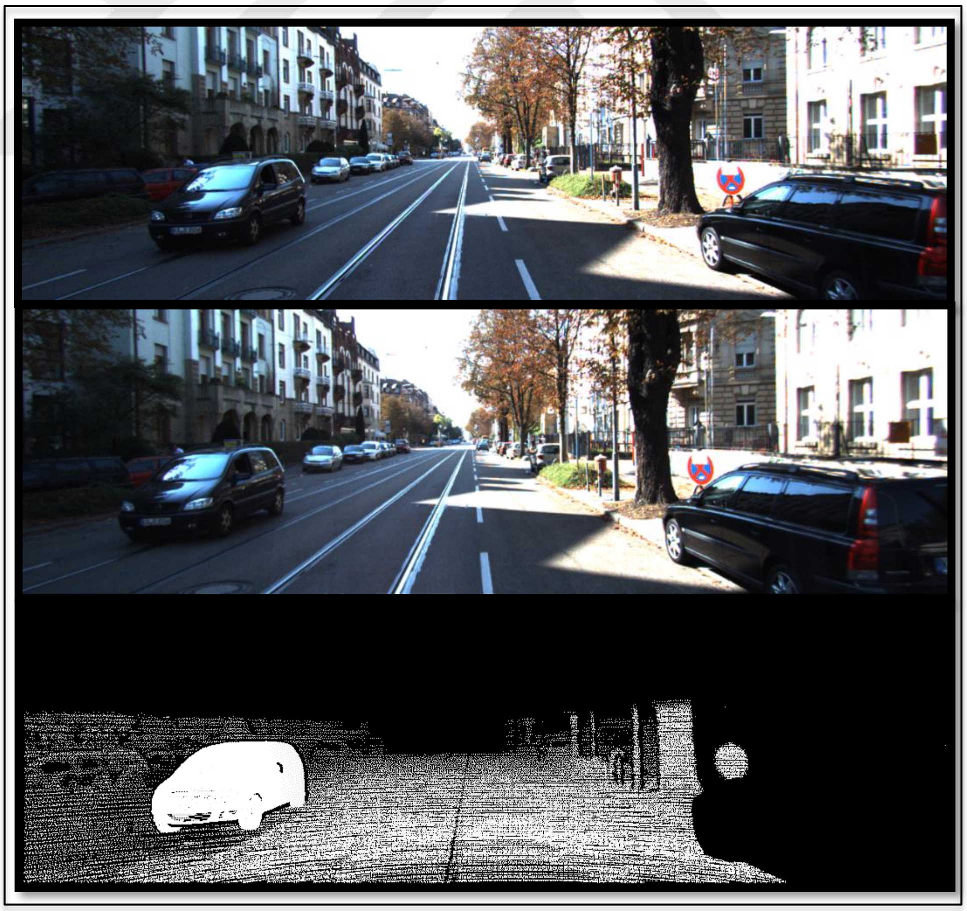
Machine learning in general and Deep Learning approaches in particular rely heavily on the availability of appropriately large, contextually relevant data sets with suitable variety in order for the learning system to capture an effective model and obtain proper results.
Since in this work Deep Learning is utilized for solving the stereo matching task specificity must be taken in choosing the data sets to train and evaluate the proposed architectures.
To this consideration two data sets have been chosen in this work, the KITTI data set (Geiger et al. 2013) and MPI-Sintel data set (Butler et al. 2012) in which both have been used comparatively in training and evaluation aims to illuminate on the effect of the data on model performance.
The choice is based on the relevancy of these sets to the stereo vision research and the aim of this thesis which studying Deep Learning solutions for depth estimation as the use of the mentioned data sets is numerous in the literature and research particular to the task. These two data sets offer contrasting takes on the source of data as KITTI is an approach based on camera plus Light Detection and Ranging (LiDAR) for capturing the stereo pair and producing the ground truth data while MPI-Sintel is a synthetic data set in which a Computer Generated Imagery (CGI) sequence is prepared in a stereo format along with the ground truth data.

The KITTI 2015 data set: This data set contains 200 stereo image pairs with resolution of 1241 by 376 pixels both Gray scale and RGB taken from a stereo rig platform mounted on a moving vehicle, ground truth is established using a LiDAR system and the data is collected from real world scenes of street recordings.

The data provide information collected from physical situations in traffic heavy streets which is preferable particularly for applications in autonomous vehicle research and the use of laser ranging provides precision ground truth but the number of samples provided is not large and the ground truth contain sparse values.

Data set have been divided into 150 image pairs for training, 50 image pairs for validation and the data set provides 200 image pairs non labelled that can be used for test purposes.

Figure 3.1: KITTI 2015 Data Set Example: left stereo image (above), right stereo image (middle), ground truth (below)

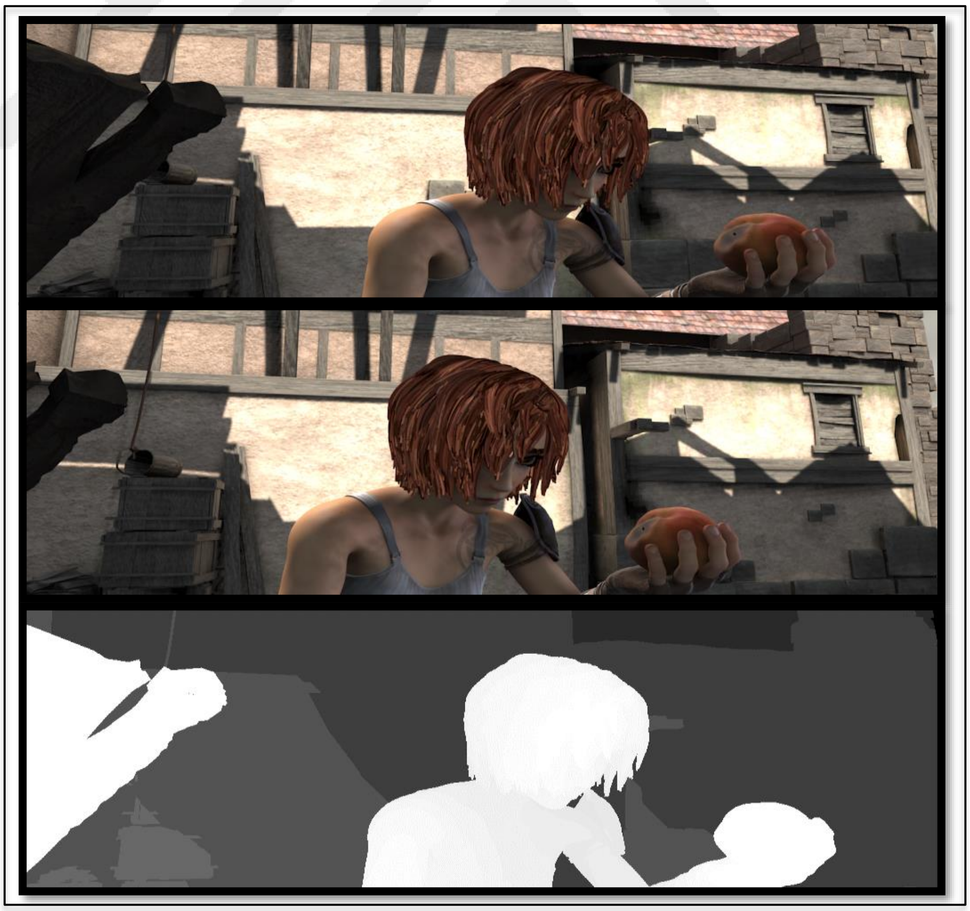


The MPI-Sintel data set: This data set contains 1064 stereo image pairs with resolution of 1024 by 436 pixels taken from the synthetic animation named Sintel the stereo pairs are in RGB format and the generated ground truth is in Gray scale.

The data contains a variety of scenes with high complexity in non-rigid motion, specular reflections, blur and atmosphere effects and being a larger data set gives an advantage in using it for machine learning based applications with the added benefit of non-sparse information dense ground truth that is a very important characteristic to be present in data used for training a Deep Learning solution.

This data set is divided into 900 image pairs for training, 100 image pairs for validation and 64 pairs were reserved for testing purposes.

Figure 3.2: MPI-Sintel Data Set Example: left stereo image (above), right stereo image (middle), ground truth (below)



The selection of the data sets was to have a thorough investigation for the effect of the data both in size and context on how the proposed architectures will perform in training, validation and testing and which design parameters are most sensitive to the data characteristics.

It is taken into account the computational resources available to produce this work as very large data sets will incur a computational burden taking in mind that several architectural choices are to be comparatively studied in experimental implementations and the chosen data sets can provide solid basis for evaluation while large size data sets that require extensive resources can be used for scaling and generalizing the results obtained in this thesis.

3.2 SUPERVISED LEARNING IMPLEMENTATION OF PRETRAINED ENCODER-DECODER NETWORK

The first design consideration partaken in this work is to investigate the effectiveness of a pretrained Neural Network architecture in the task of stereo vision depth estimation.

Many Deep Learning Neural Network predesigned and pretrained architectures are available and in this thesis two architectures have been chosen to be implemented: VGG and Inception.

The VGG pretrained architecture: this convolutional architecture is categorized as a very deep design with up to 19 layers and the main aim of its development is large scale image classification, it uses a unified convolutional kernel parameter of 3 by 3 for all of its convolutional layers and the number of feature maps starts from 64 in the first CNN layer up to 512 in the last layer, number of parameters is 14.4 million parameter.

The pretraining is performed on ImageNet data set (Deng et al. 2009) with 1.3 million training images, it is imperative to evaluate how such a pretrained system with classification task at hand can be useful for the task of depth estimation.

The architectural design choices in VGG are the usage of highly deep layer size and unified parameter selection for kernels in each feature map for each CNN layer thus its performance makes explicit the informational content and variability of the data set it uses to train the weight parameters.

The VGG network is used in the encoder stage with only the convolutional layers omitting the fully connected layers which represent a classifier process.

Since the system is designed for stereo input pair the encoder contains two VGG branches that are joined by a concatenation of their feature maps which are subjected to a Batch Normalization (BN) process (Ioffe and Szegedy 2015) to reduce over fitting.

The decoder stage is comprised of four convolution layers of successively reduced feature size from 64 kernels down to 3 for the output convolution that comprise the 3 channels of target image.

The following table gives a full view of the layers used in the model:

Table 3.1: Pretrained Supervised Model with VGG Encoder

Layer(type)	**Output Shape**	**Parameters Number**	**Connected to**
input_1 (Input layer)	(None,750,750,3)	0	*
input_2 (Input layer)	(None,750,750,3)	0	*
vgg16 (Model)	Multiple	14714688	input_1[0][0] input_2[0][0]
batchnormalization_1 (BatchNormalization)	(None,23, 23,512)	2048	vgg16[1][0]
batchnormalization_2 (BatchNormalization)	(None,23, 23,512)	2048	vgg16[2][0]
concatenate_1 (Concatenate)	(None,23, 23,1024)	0	batchnormalization_1[0][0] batchnormalization_2[0][0]
conv2d_1 (Conv2D)	(None,23, 23,64)	589888	concatenate_1[0][0]
upsampling2d_1 (UpSampling2D)	(None,92, 92,64)	0	conv2d_1[0][0]

conv2d_2 (Conv2D)	(None,92, 92,32)	18464	upsampling2d_1 [0][0]
upsampling2d_2 (UpSampling2D)	(None,368, 368,32)	0	conv2d_2[0][0]
conv2d_3 (Conv2D)	(None,368, 368,16)	4624	upsampling2d_2 [0][0]
upsampling2d_3 (UpSampling2D)	(None,736, 736,16)	0	conv2d_3[0][0]
conv2d_3 (Conv2D)	(None,736, 736,3)	435	upsampling2d_3 [0][0]

The weight of VGG model are shared for the branches to reduce computational cost and only upper layers were selected to be trainable for tuning with the decoder stage thus the number of trainable parameters for the design is 5,335,075 in addition to that batch normalization introduces parameters that do not enter in training process giving the model a total number of parameters of 15,332,195.

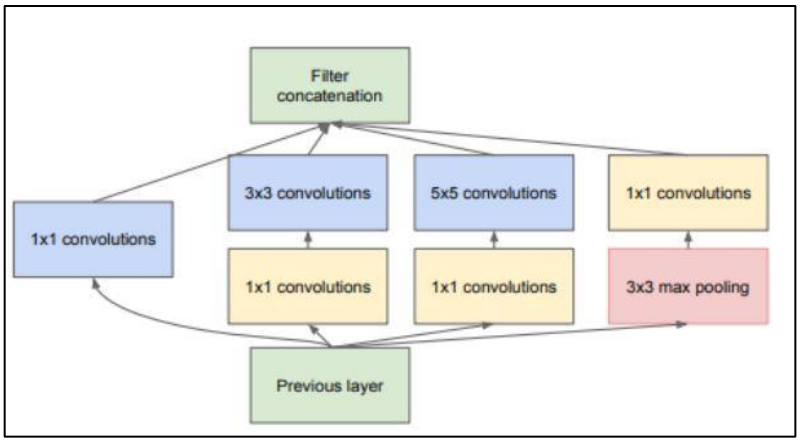
The Network is trained on the provided system for 1000 epochs on each of the two data sets and outputs were taken on the available testing image pairs.

The loss function used is binary cross entropy with an (RMSPROP) optimizer having learning rate of 10^{-3} and decay rate of 10^{-6}.

The inception pretrained architecture: a convolutional architecture that is based on the concept of sparse connectivity and clustering of connections between modules of convolution that capture the optimal local statistics.

The modules are stacked on top of each other with each cluster connected to the previous cluster in a hierarchical structure, feature maps of this architecture use kernel sized of 1 by 1, 3 by 3 and 5 by 5. This modularity facilitates increasing the number of units without major upsurge in number of parameters in the network thus conserving computational resources

Figure 3.3: Inception Module



The typical implementation of inception modules is LeNet architecture which has 22 inception layers and with counting the inner layers of the modules the number is up to 100 convolutional layers making this architecture a highly deep design providing large learning capacity. In this work the Inception convolutional network with no fully connected classifier and only upper layer training allowed is used for each branch of the encoder stage taking input from the stereo image pair and the two branches of the encoder are connected with a concatenation process, the decoder stage is identical to the decoder used for the pretrained VGG design.

The following table gives full view of the layers used in the model:

Table 3.2: Pretrained Supervised Model with Inception Encoder

Layer(type)	**Output Shape**	**Parameters Number**	**Connected to**
input_1 (Input layer)	(None,750,750,3)	0	*
input_2 (Input layer)	(None,750,750,3)	0	*
Inception_v3(Model)	Multiple	21802784	input_1[0][0] input_2[0][0]
concatenate_1 (Concatenate)	(None,22, 22,4096)	0	Inception_v3[1][0] Inception_v3[2][0]

conv2d_1 (Conv2D)	(None,22, 22,64)	2359360	concatenate_1[0][0]
upsampling2d_1 (UpSampling2D)	(None,44, 44,64)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None,44, 44,32)	18464	upsampling2d_1 [0][0]
upsampling2d_2 (UpSampling2D)	(None,176, 176,32)	0	conv2d_2[0][0]
conv2d_3 (Conv2D)	(None,176, 176,16)	4624	upsampling2d_2 [0][0]
upsampling2d_3 (UpSampling2D)	(None,704, 704,16)	0	conv2d_3[0][0]
conv2d_3 (Conv2D)	(None,704, 704,3)	435	upsampling2d_3 [0][0]

Number of trainable parameters for this design is 18,598,819, total number of parameters is 24,185,667.

The Network is trained on the provided system for 1000 epochs on each of the two data sets and outputs were taken on the available testing image pairs.

The loss function used is binary cross entropy with an (RMSPROP) optimizer having learning rate of 10^{-6} and nor decay rate employed.

It is noticeable that the Inception implementation has higher number of parameters compared to the VGG implementation and since sparse connectivity and batch normalization are built in features of the Inception module no extra batch normalization layers were added to the implementation in the Inception case.

3.3 SUPERVISED LEARINING IMPLEMENTATION OF ENCODER-DECODER NETWORK

The second design consideration assumed in this thesis is to formulate non-trained architectures specific to the task of stereo vision based on the latest approaches and suggestions in literature.
Focussed on the principles of efficiency and ease of training, Depth-wise Separable Convolution (V. Vanhoucke 2014) is chosen as the main operation in the construction of the architectures as it provides a considerable decrease in the parameter size of the models leading to a reduction in computational requirements and overfitting thus making the same choice of convolution operations adopted in Xception architecture (Chollet 2016).
Two architectures are proposed in this work for the non-trained architectures: parameter reduced DenseMapNet and a Siamese Network (Bromley et al., 1993) with encoder local skip connections combined with correlation extraction merge of stereo branches.

Parameter reduced DenseMapNet: this model is a modification of the DenseMapNet architecture from the use of typical convolution operation to the use of the depth-wise separable convolution.
This architecture uses a DenseNetwork inspired structure where each layer or module of several layers is connected not just to the feedforward input but also to the inputs of the previous layer or module with the aim of providing paths for information to flow in between layers to aid in gradient calculations and preventing vanishing gradients.
As an example of mapping type encoder-decoder architecture DenseMapNet consists of two sections a Correspondence Network that learns to perform stereo matching on the stereo image pair and a Disparity Network which applies the resulting disparity on the reference image by performing a decoding of the obtained representation of correspondence and convolving the output to the reference image that is provided as a feature map from the initial layers of the architecture. The network contains 18 convolutional units with up to 63 layers and the proposed implementation of this work that utilizes the depth-wise separable convolution method has a parameter size of 697,512 parameter. The model is trained for 1000 epochs on both of the provided data sets and results are obtained for evaluation on the available test image pairs. The loss function

used is binary cross entropy with an (RMSPROP) optimizer having learning rate of 10^{-3} and decay rate of 10^{-6}.

The following table gives an overview of the layers used in the model "for the full view see. Appendix-1, Table-1":

Table 3.3: Supervised Model with Parameter reduced DenseMapNet

Layer(type)	**Output Shape**	**Parameters Number**	**Connected to**
input_1 (Input layer)	(None,748,744,3)	0	*
input_2 (Input layer)	(None,748,744,3)	0	*
concatenate_1 (Concatenate)	(None,748,744,6)	0	input_1[0][0] input_2[0][0]
separable_conv2d_3 (SeparableConvolution)	(None, 748, 744, 128)	1046	concatenate_1[0][0]
Continu.			
disparity_output (Activation)	(None, 748, 744, 3)	0	separable_conv2d_14[0][0]

Siamese Network with encoder local skip connections and correlation merge: this model is based on the skipping connection mechanism (Ronneberger, Fischer and Brox 2015) which typically connects the encoder layers to the decoder layers, it is proposed within this work in addition to this design criterion the implementation of a skipping connection within the encoder to increase information flow from the initial layers to layers deep in the encoder thus providing in addition to amplifying the representation relevancy of the deep layers a procedure to facilitate a more robust gradient calculation .

Since the application is in stereo vision the encoder consists of two branches that are identical in parameters and use the method of weight sharing between the encoder branches to achieve a "Siamese" network layout, further reduction of parameter requirement is performed with the application of depth-wise separable convolution operation.

Another design choice that is proposed for this model is to merge the Siamese branches with a process of correlation calculation inspired by the brain (Welchman 2016), the process consists of a multiplication merge layer followed by a convolution operation layer in which the kernel is characterized by an asymmetric window with preference for the horizontal direction provided that the inputs are rectified, this choice provides lower number of feature maps that contain correlations of the two branches and with the preference for the horizontal direction the computational requirement for learning the matching procedure can be further reduced.

The network contains 51 layers and the number of parameters is 453,686.

The model is trained for 1000 epochs on both of the provided data sets and results are obtained for evaluation on the available test image pairs.

The loss function used is binary cross entropy with an (RMSPROP) optimizer having learning rate of 10^{-3} and decay rate of 10^{-6}.

The following table gives an overview of the layers used in the model "for the full view see. Appendix-1, Table-2":

Table 3.4: Supervised Model with Siamese Network

Layer(type)	**Output Shape**	**Parameters Number**	**Connected to**
input_1 (Input layer)	(None, 672, 672, 3)	0	*
input_2 (Input layer)	(None, 672, 672, 3)	0	*
separable_conv2d_1 (SeparableConvolution)	(None, 672, 672, 32)	275	input_1[0][0] input_2[0][0]
batch_normalization_1 (BatchNormalization)	(None, 672, 672, 32)	128	separable_conv2d_1 [0][0] separable_conv2d_1 [1][0]

activation_1 (Activation)	(None, 672, 672, 32)	0	batch_normalization_1[0][0] batch_normalization_1[1][0]
Continu.			
separable_conv2d_11 (SeparableConvolution)	(None, 672, 672, 3)	3843	concatenate_7[0][0]

3.4 UNSUPERVISED LEARNING IMPLEMENTATION OF ENCODER-DECODER NETWORK

The third design consideration in this thesis is to apply the unsupervised learning approach to the task of stereo vision (Wang et al. 2018) (Jiang et al. 2018) (Zhan et al. 2019).

Unsupervised learning is the technique of training the machine learning algorithm without providing the labelled targets and rely on other indicators to optimize the performance of the model, denoising auto-encoders represent an example where the image itself is to be used as an input and a measure of performance.

For stereo vision one can rely on the availability of left and right image pair and take advantage of the spatial consistency between them by considering a modification of the Auto Encoder architecture in which the left image is chosen as a reconstruction target and the neural network apply learned transformations on the right image taken as an input to reduce the reconstruction loss and the internal decoder model outputs will be forced to learn the disparity mapping through this modified loss criterion.

Other approaches rely on temporal consistency between image frames in videos which also can be used to train a depth inference encoder Neural Network.

It is chosen in this work to implement the stereo consistency approach and two architectures are proposed: a modified DenseMapNet that is included within unsupervised loss process and Symmetric Encoder-Decoder no skip connections.

DenseMapNet based Encoder-decoder: this architecture repurposes the design of DenseMapNet to accommodate a “self-supervised” learning mechanism by using the left image as a reference for a consistency based unsupervised loss thus omitting the convolutional path for it in the architecture which applies a single convolutional operation on the left image and feed to the last stage to apply depth estimation mapping.
The right image convolutional path which represents the core operations of DenseMapNet model is utilized as the target model to be trained to capture the depth mapping procedure and similar to the supervised approach all convolutions are performed using the depth-wise separable technique to reduce overall parameter requirement.

While the complete system used in training phase is using the stereo image pair the trained Encoder-Decoder resulting from applying the unsupervised approach only requires a single image representing the right image form the pair, this is an advantage of this methodology as it only requires the full stereo camera systems for capturing training data while deployment setting only requires mono images.
Another advantage of the unsupervised technique and in particular the employed self-supervision by consistency measures is that it relies on the actual physical consideration of spatial or temporal dependability between multiple views or multiple successive captures of localized scenes to enforce a performance measure on the training process reducing the work or hardware resources to obtain ground truth samples.
The network contains 63 layers and the number of parameters is 753,166.
The model is trained for 1000 epochs on both of the provided data sets and results are obtained for evaluation on the available test image pairs.
The optimizer used is (ADAM) optimizer.
The following table gives an overview of the layers used in the model “for the full view see. Appendix-1, Table-3”:

Table 3.5: Unsupervised Model with DenseMapNet Encoder-decoder

Layer(type)	Output Shape	Parameters Number	Connected to
input_1 (Input layer)	(None, 672, 672, 3)	0	*
input_2 (Input layer)	(None, 672, 672, 3)	0	*
model_1 (Model)	(None, 672, 672, 3)	753166	input_1[0][0]
custom_layer_8 (CustomLayer)	(None, 672, 672, 3)	0	input_1[0][0] input_2[0][0] model_1[1][0]
Conitnu.			
disparity_output (Activation)	(None, 672, 672, 3)	0	separable_conv2d_32 [0][0]

Symmetric Encoder-decoder: this architecture is designed with the aim of applying the unsupervised approach to a minimalistic Symmetric Encoder-Decoder architecture.

The main consideration is to have a straightforward skip connections free architecture with symmetry of parameter selection for the convolutional kernels, this choice can then make explicit the effect of self-supervised learning mechanism on the flow of the weight values during training and its susceptibility to overfitting as well as the overall ability to extract relevant representations from provided unlabelled data due to consistency loss functions.

In this design the depth-wise separable technique is also used to reduce parameter requirement with the goal that the reconstruction loss captures the transformations required for producing the stereo matching mechanism.

The network contains 33 layers and the number of parameters is 661,782.

The model is trained for 1000 epochs on both of the provided data sets and results are obtained for evaluation on the available test image pairs.

The optimizer used is (ADAM) optimizer.

The following table gives an overview of the layers used in the model “for the full view see. Appendix-1, Table-4”:

Table 3.6: Unsupervised Model with Symmetric Encoder-decoder

Layer(type)	**Output Shape**	**Parameters Number**	**Connected to**
input_1 (Input layer)	(None, 672, 672, 3)	0	*
input_2 (Input layer)	(None, 672, 672, 3)	0	*
model_1 (Model)	(None, 672, 672, 3)	661782	input_1[0][0]
custom_layer_8 (CustomLayer)	(None, 672, 672, 3)	0	input_1[0][0] input_2[0][0] model_1[1][0]
Continu.			
separable_conv2d_9 (SeparableConvolution)	(None, 672, 672, 3)	771	up_sampling2d_4 [0][0]

4. EXPERIMENTAL RESUTLS

This part presents the hardware system available to implement and run the proposed architectures and the results obtained from the execution of the implementations.

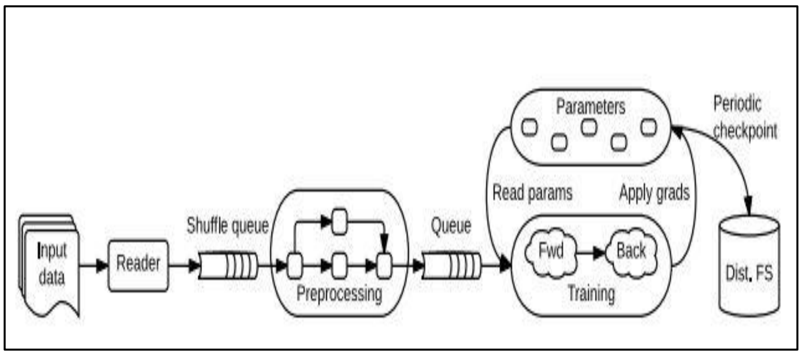
4.1 HARDWARE AND SOFTWARE

The hardware system used for thesis work is a desktop build with Intel Core i9 9900k, 32GB of system RAM, Nvidia RTX 2080 Ti GPU with 11GB of RAM,Cuda toolkit version 10.1.167 with cudnn version 7.6 and the operating system used is UBUNTU 18.04.3 LTS.
The programming platform chosen to perform the implementation is Google Tensorflow with Keras API in the Python programming language.

Python programming language: a high level, interpreted, dynamically typed and object oriented programming language designed to be modular and open source allowing for rapid development of code bases and due to having clear and readable syntax it provides low cost on debugging and maintenance.
Python version used is computed using Anaconda package management environment 3.7.3 of version 4.7.5

Tensorflow: a machine learning platform designed for large scale and data intensive computations, it represents the state, operations and transformations on states with the methodology of a Dataflow graph.
Nodes of the dataflow graph can be assigned to different machines in a cluster and to different computational devices in a single machine, the devices include multicore CPUs, GPUs, and application specific chips such as Tensor Processing Units (TPUs).
Such a platform provides developers and researchers with the flexibility to perform a variety of learning and optimization projects and increases sharing and collaboration.
Tensorflow version used is 1.14

Figure 4.1: Tensorflow Dataflow Graph Schematic



Keras API: a Python based high level API for programming Neural Networks developed for fast and seamless prototyping and evaluation with capability of running on both CPU and GPU.
Keras emphasizes on modularity and readability and provides support for a wide class of architectures including recurrent and convolutional Neural Networks.
Keras version used is 2.2.4

4.2 RESULTS

This part presents the obtained results of the proposed architectures from performing training and evaluation with the designated data sets executed on the hardware system provided.
All designs were trained for 1000 epochs with accuracy and loss curves obtained for training and validation on each epoch from training and validation sets respectively, image results are obtained from evaluating on the test data sets.

Pretrained supervised learning Encoder-Decoder architectures:

The following figure shows the training and validation accuracies attained for the Inception pretrained supervised learning Encoder-decoder:

Figure 4.2: Training and Validation Accuracy for Inception Pretrained Supervised Learning Model on KITTI Data Set

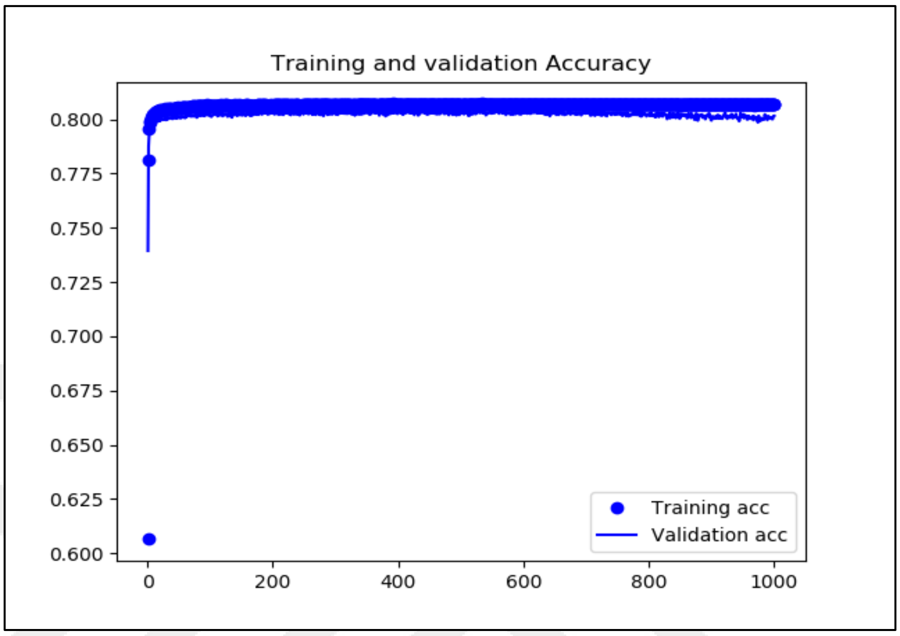
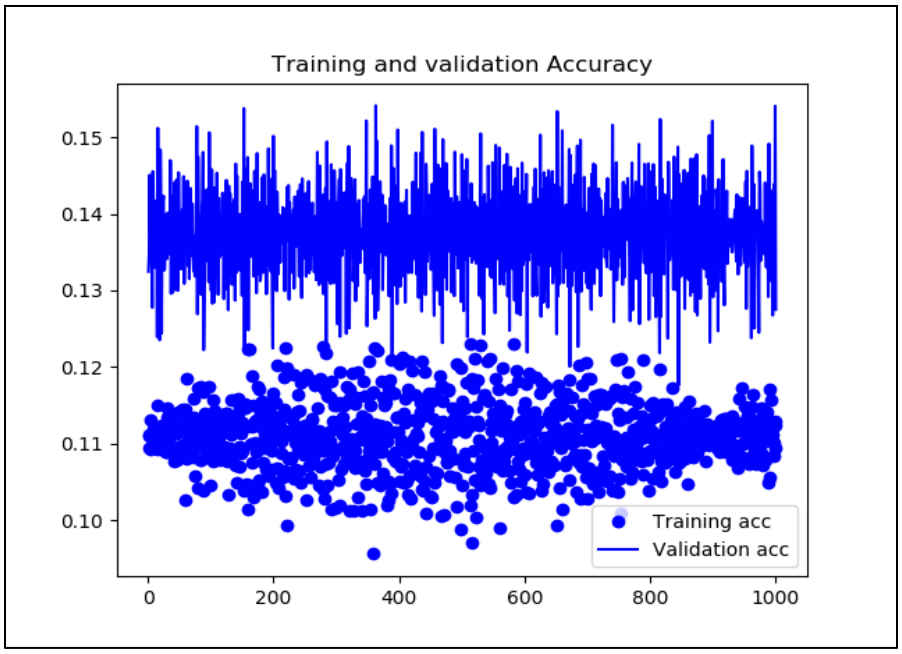


Figure 4.3: Training and Validation Accuracy for Inception Pretrained Supervised Learning Model on MPI-Sintel Data Set



The following figure shows the training and validation losses attained for the Inception pretrained supervised learning Encoder-decoder:

Figure 4.4: Training and Validation Loss for Inception Pretrained Supervised Learning Model on KITTI Data Set

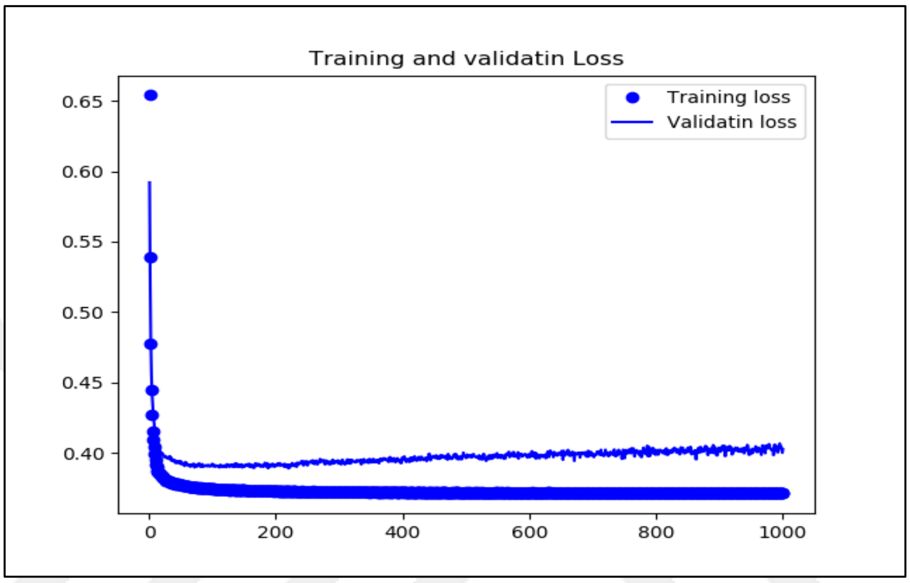
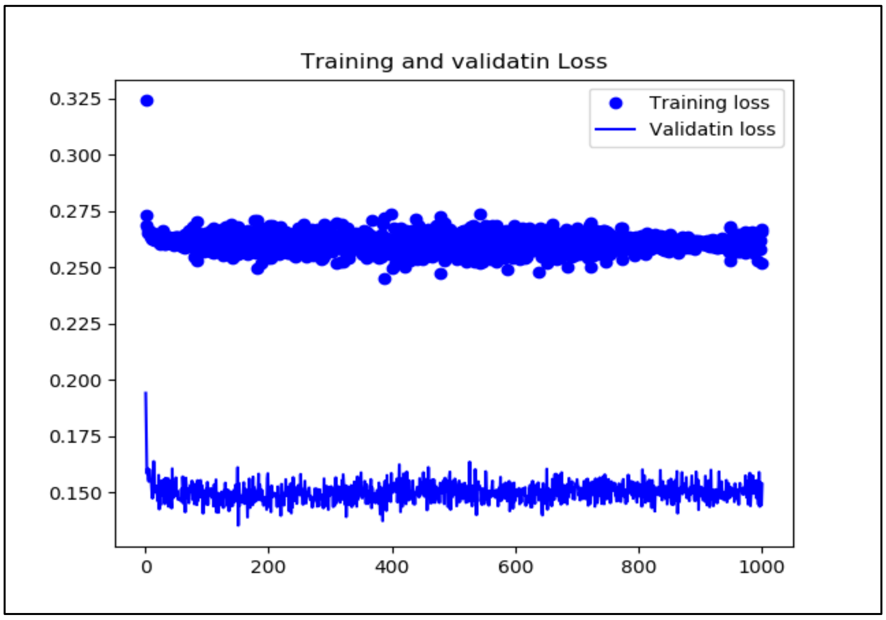


Figure 4.5: Training and Validation Loss for Inception Pretrained Supervised Learning Model on MPI-Sintel Data Set



The following figure shows the evaluation image result attained for the Inception pretrained supervised learning Encoder-decoder:

Figure 4.6: Evaluation Output for Inception Pretrained Superviesed Learning Model on KITTI Data Set

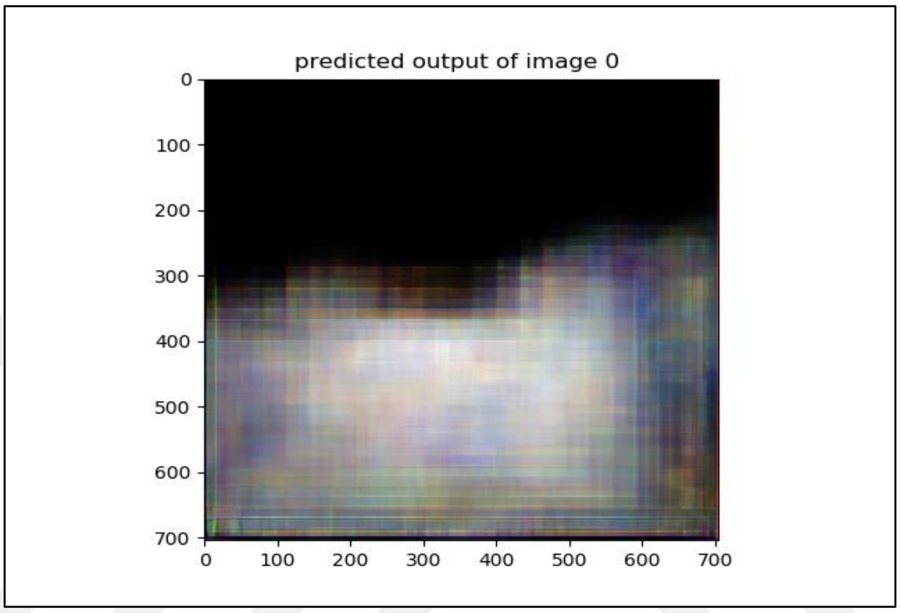
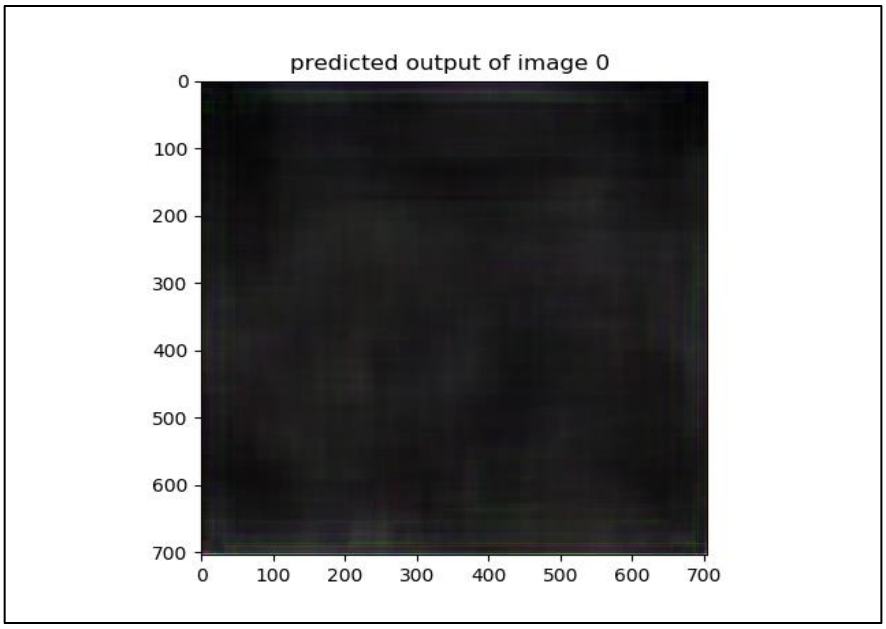


Figure 4.7: Evaluation Output for Inception Pretrained Superviesed Learning Model on MPI-Sintel Data Set



The following figure shows the training and validation accuracies attained for the VGG pretrained supervised learning Encoder-decoder:

Figure 4.8: Training and Validation Accuracy for VGG Pretrained Supervised Learning Model on KITTI Data Set

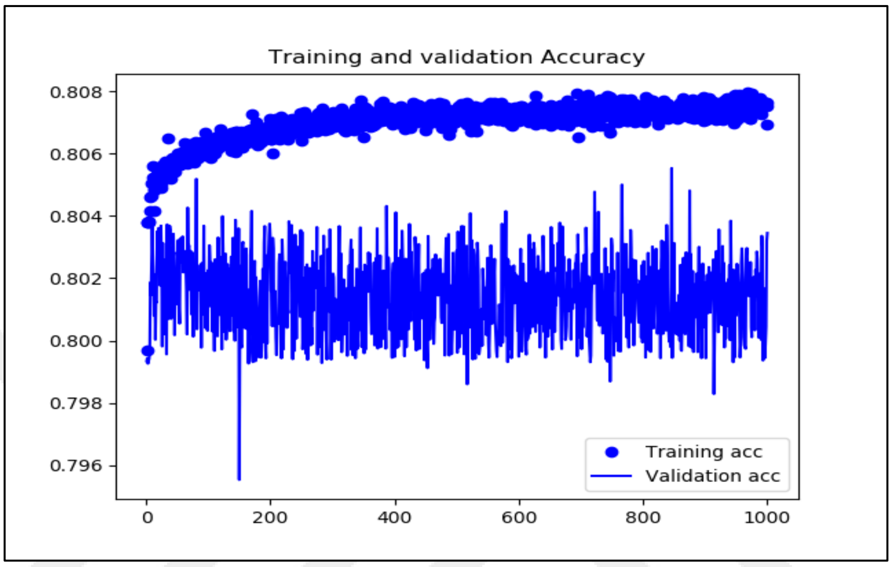
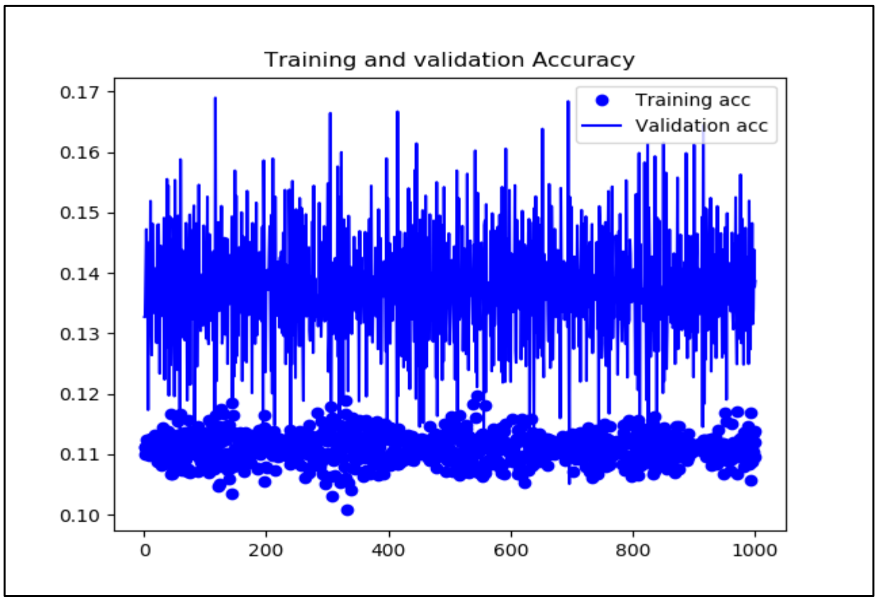


Figure 4.9: Training and Validation Accuracy for VGG Pretrained Supervised Learning Model on MPI-Sintel Data Set



The following figure shows the training and validation losses attained for the VGG pretrained supervised learning Encoder-decoder:

Figure 4.10: Training and Validation Loss for VGG Pretrained Supervised Learning Model on KITTI Data Set

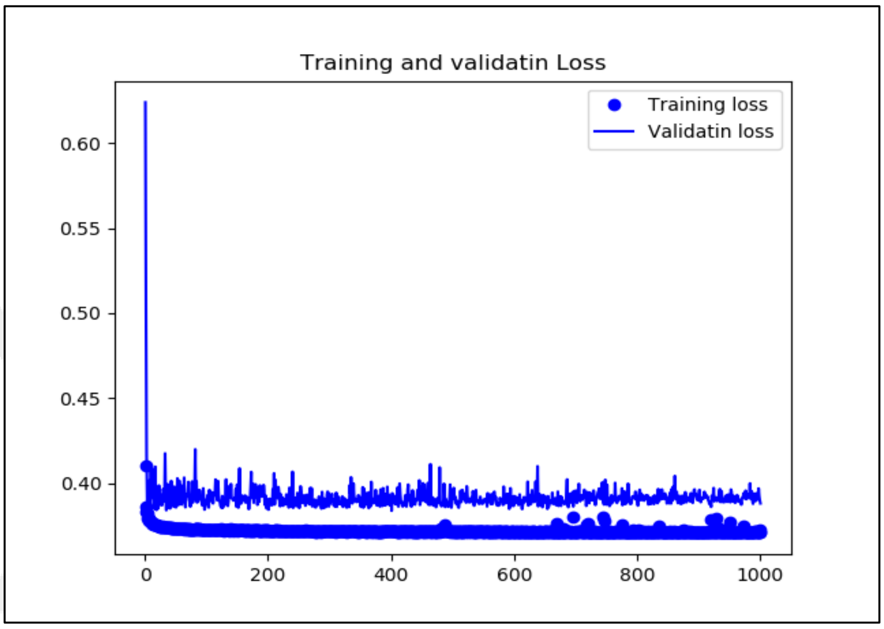
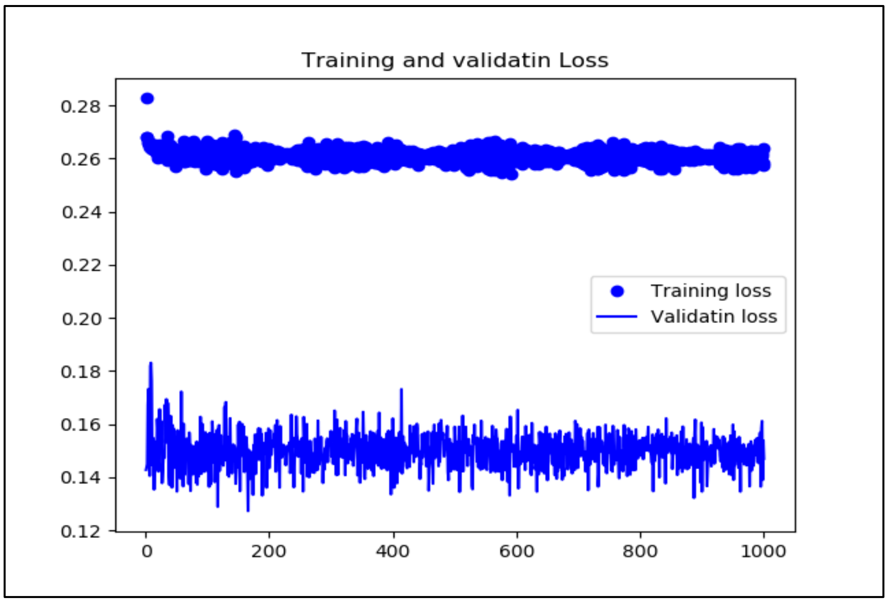


Figure 4.11: Training and Validation Loss for VGG Pretrained Supervised Learning Model on MPI-Sintel Data Set



The following figure shows the evaluation disparity map result attained for the VGG pretrained supervised learning Encoder-decoder:

Figure 4.12: Evaluation Output for VGG Pretrained Superviesed Learning Model on KITTI Data Set

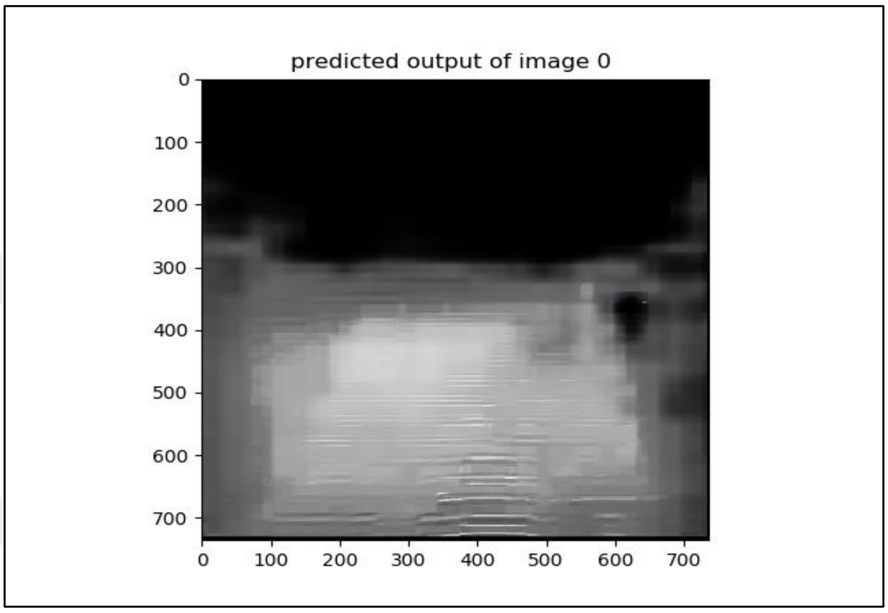
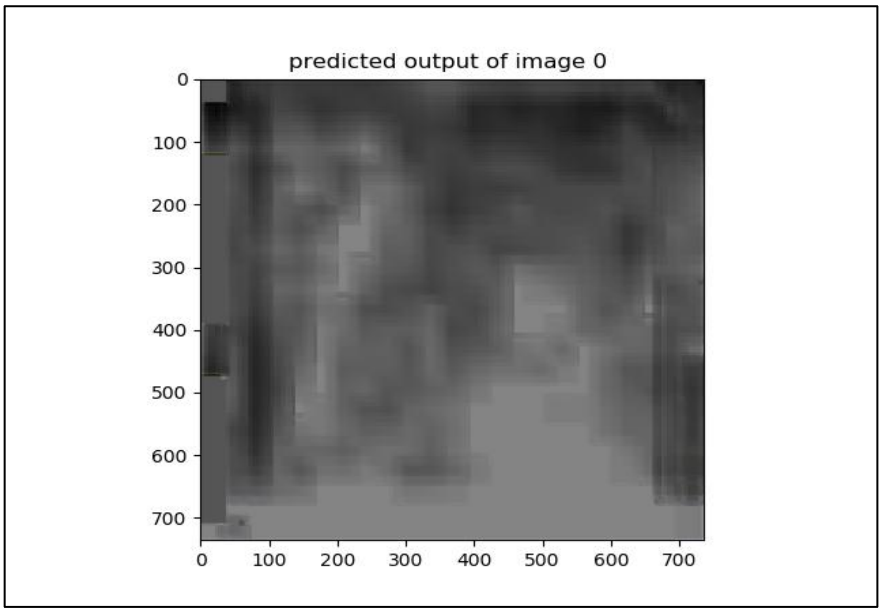


Figure 4.13: Evaluation Output for VGG Pretrained Superviesed Learning Model on MPI-Sintel Data Set



Supervised learning Encoder-Decoder architectures:

The following figure shows the training and validation accuracies attained for the parameter reduced DenseMapNet supervised learning Encoder-decoder:

Figure 4.14: Training and Validation Accuracy for DenseMapNet Supervised Learning Model on KITTI Data Set

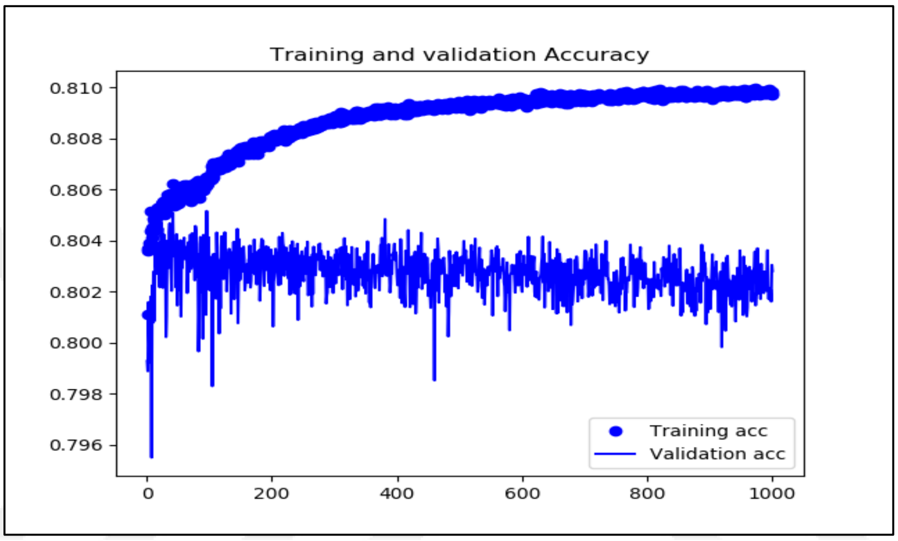
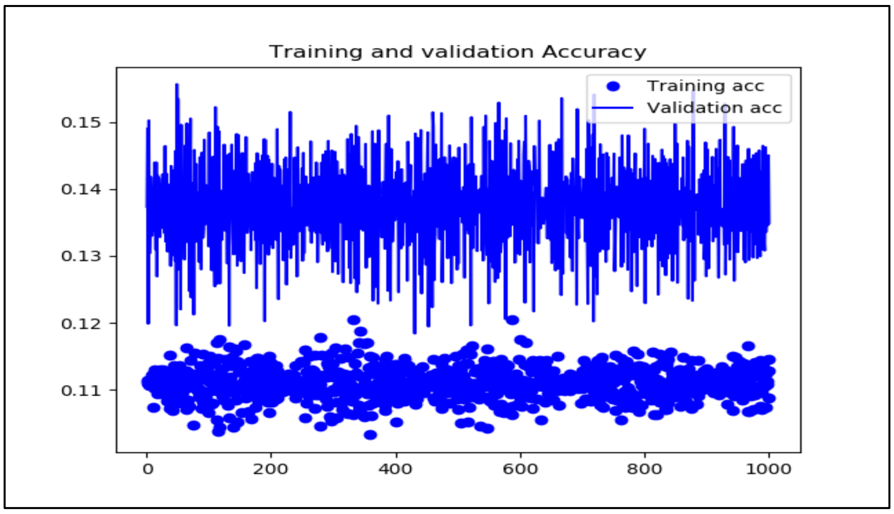


Figure 4.15: Training and Validation Accuracy for DenseMapNet Pretrained Supervised Learning Model on MPI-Sintel Data Set



The following figure shows the training and validation losses attained for the parameter reduced DenseMapNet supervised learning Encoder-decoder:

Figure 4.16: Training and Validation Loss for DenseMapNet Supervised Learning Model on KITTI Data Set

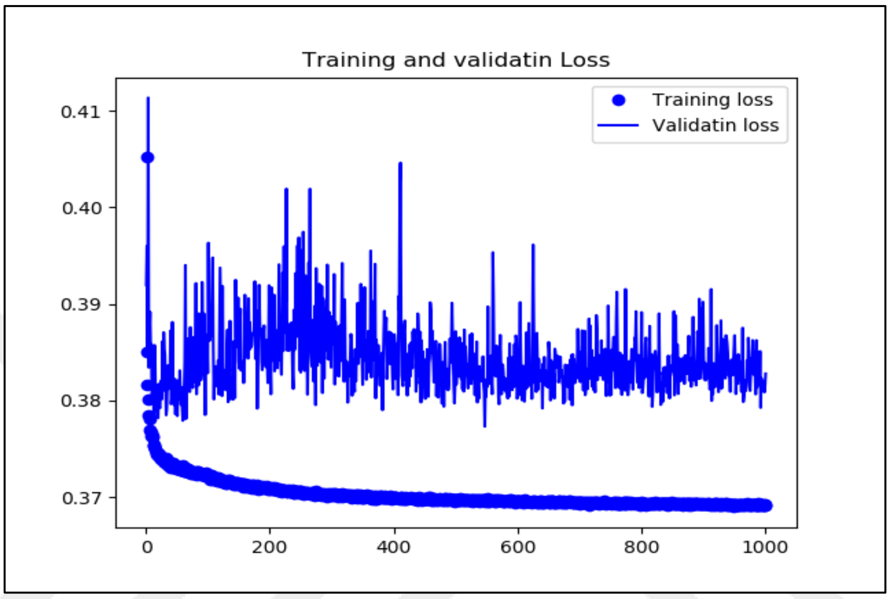
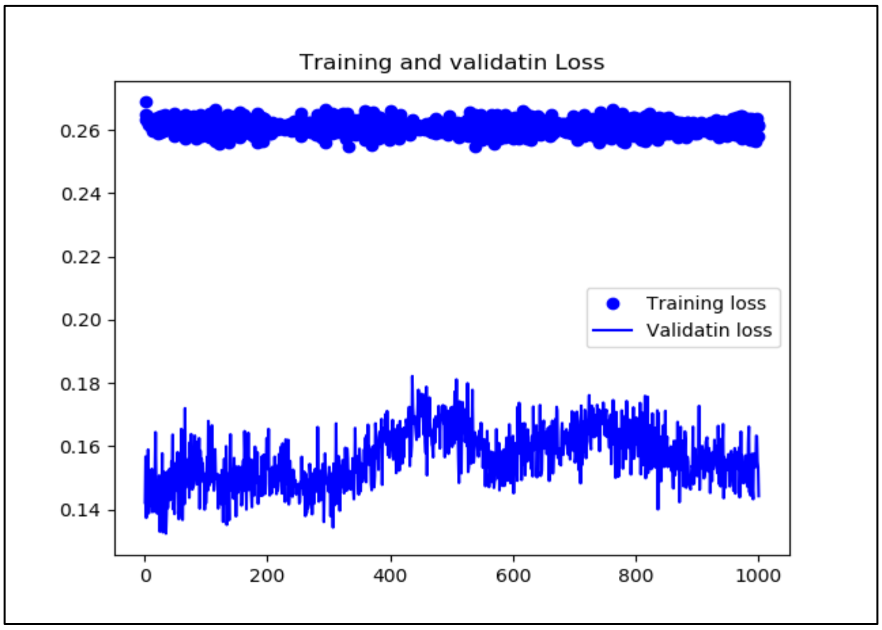


Figure 4.17: Training and Validation Loss for DenseMapNet Supervised Learning Model on MPI-Sintel Data Set



The following figure shows the evaluation image result attained for the parameter reduced DenseMapNet supervised learning Encoder-decoder:

Figure 4.18: Evaluation Output for DenseMapNet Superviesed Learning Model on KITTI Data Set

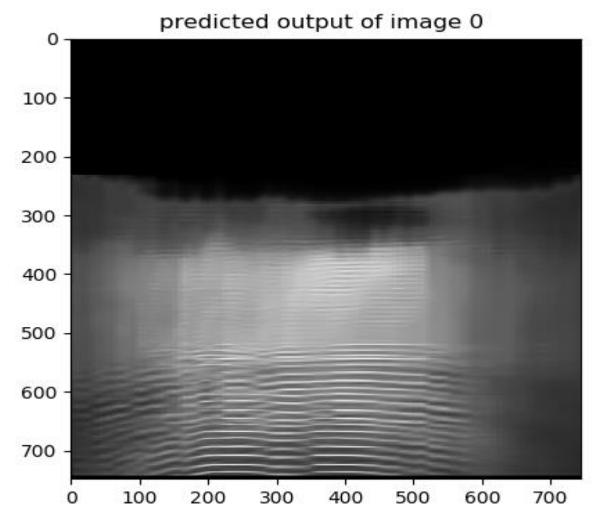
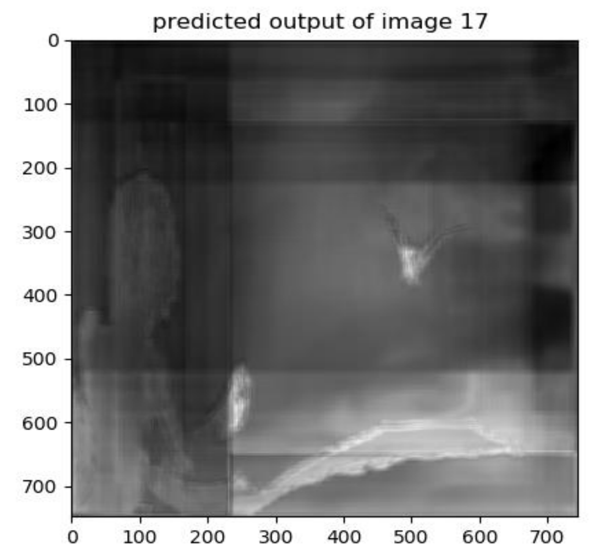


Figure 4.19: Evaluation Output for DenseMapNet Superviesed Learning Model on MPI-Sintel Data Set



The following figure shows the training and validation accuracies attained for the Siamese Network with encoder local skip connections and correlation merge supervised learning Encoder-decoder:

Figure 4.20: Training and Validation Accuracy for Siamese Network Supervised Learning Model on KITTI Data Set

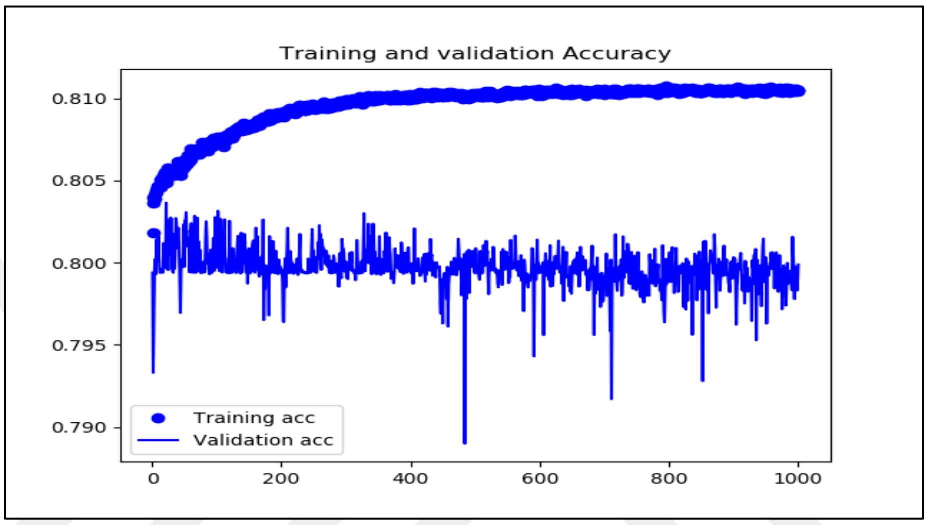
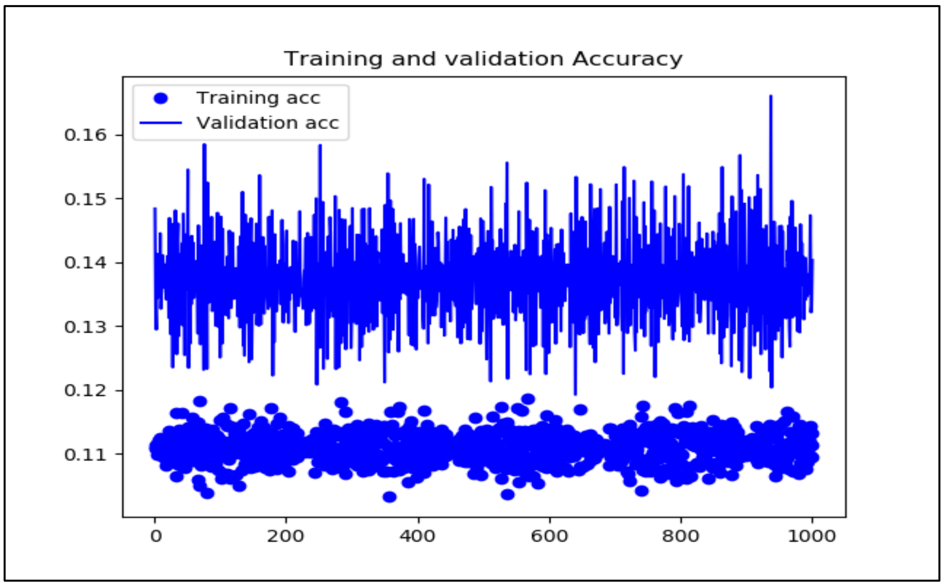


Figure 4.21: Training and Validation Accuracy for Siamese Network Supervised Learning Model on MPI-Sintel Data Set



The following figure shows the training and validation losses attained for Siamese Network with encoder local skip connections and correlation merge supervised learning Encoder-decoder:

Figure 4.22: Training and Validation Loss for Siamese Network Supervised Learning Model on KITTI Data Set

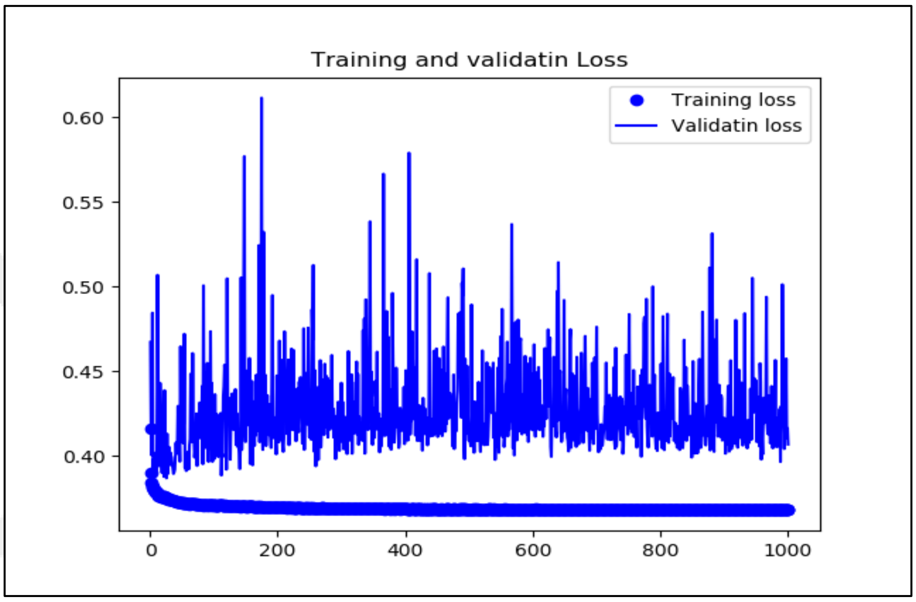
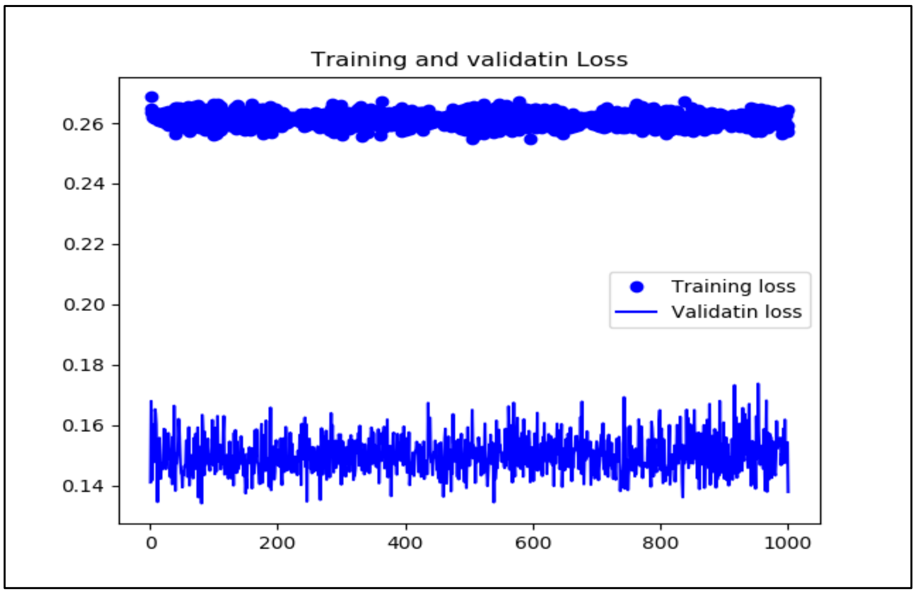


Figure 4.23: Training and Validation Accuracy for Siamese Network Supervised Learning Model on MPI-Sintel Data Set



The following figure shows the evaluation disparity map result attained for the Siamese Network with encoder local skip connections and correlation merge supervised learning Encoder-decoder:

Figure 4.24: Evaluation Output for Siamese Network Superviesed Learning Model on KITTI Data Set

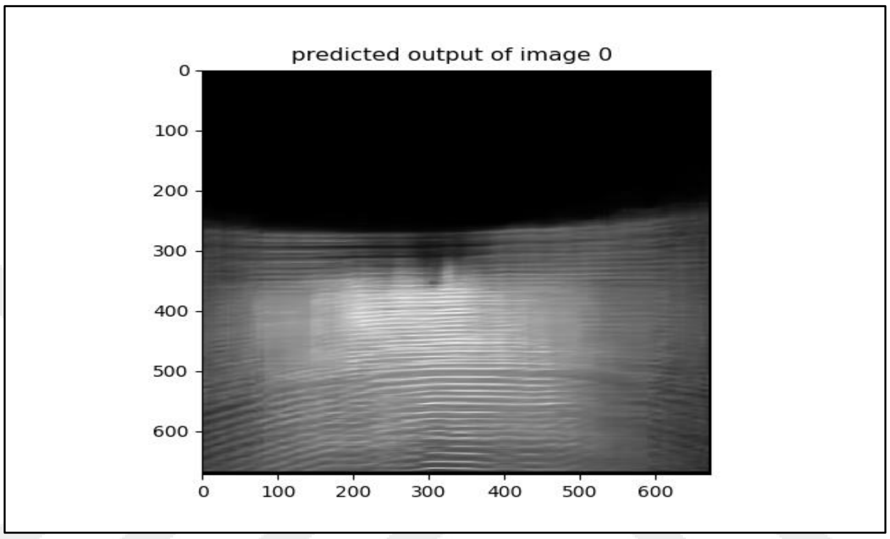
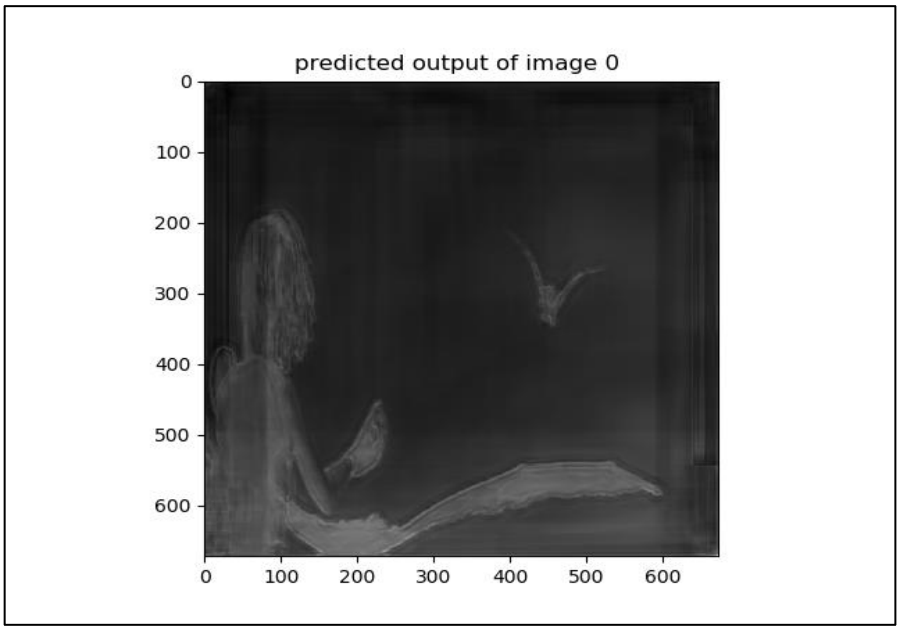


Figure 4.25: Evaluation Output for Siamese Network Supervised Learning Model on MPI-Sintel Data Set



Unsupervised learning Encoder-Decoder architectures:

In the unsupervised technique no labelled images are provided to the network and the measure for optimization is the left image to right image consistency that acts as a reconstruction loss and directs the evolution of the weights.

Accuracy metric is omitted in the unsupervised case as ground truth images are not needed in this implementation and the reconstruction losses of training and validation are recorded.

The following figure shows the training and validation reconstruction losses attained for the DenseMapNet based unsupervised learning Encoder-decoder:

Figure 4.26: Training and Validation Reconstruction Loss for DenseMapNet Unsupervised Learning Model on KITTI Data Set

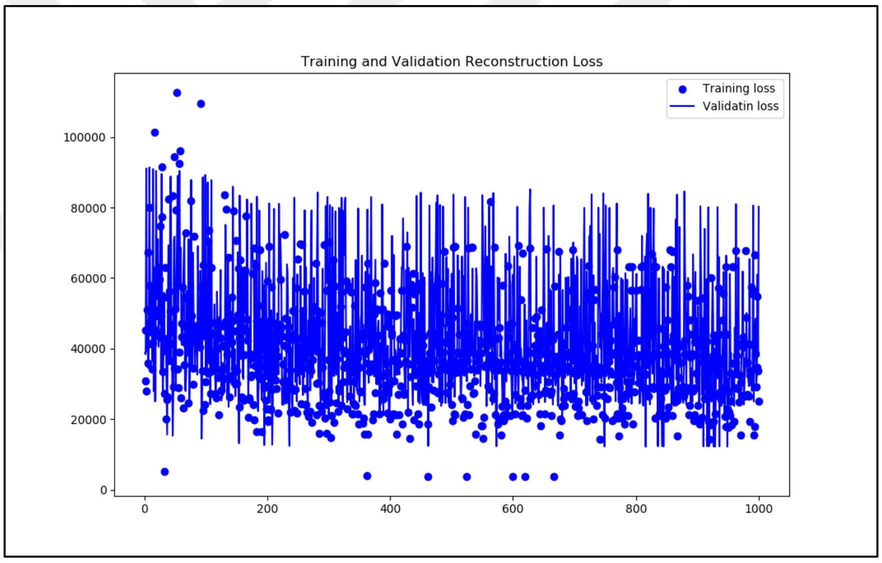
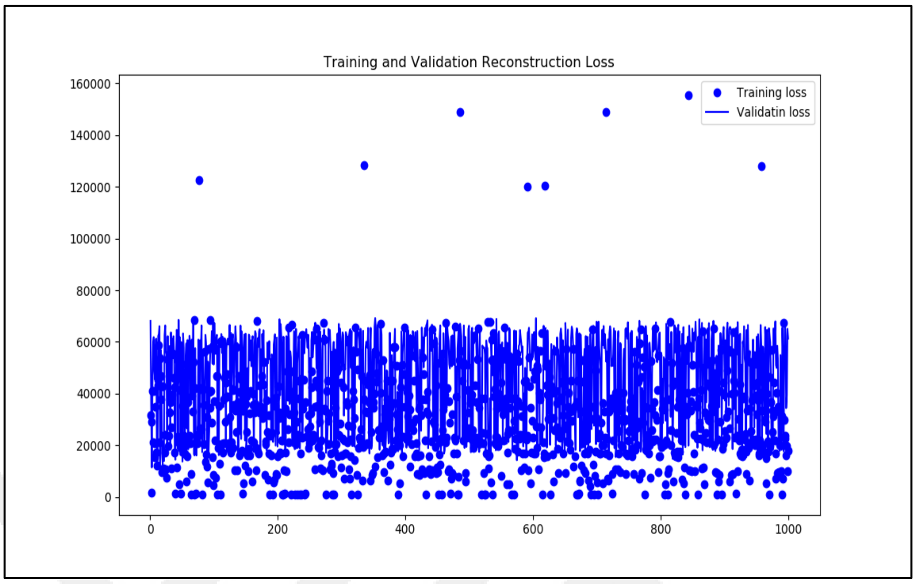


Figure 4.27: Training and Validation Reconstruction Loss for DenseMapNet Unsupervised Learning Model on MPI-Sintel Data Set



The following figure shows the evaluation image result attained for the DenseMapNet based unsupervised learning Encoder-decoder:

Figure 4.28: Evaluation Output for DenseMapNet Unsupervised Learning Model on KITTI Data Set

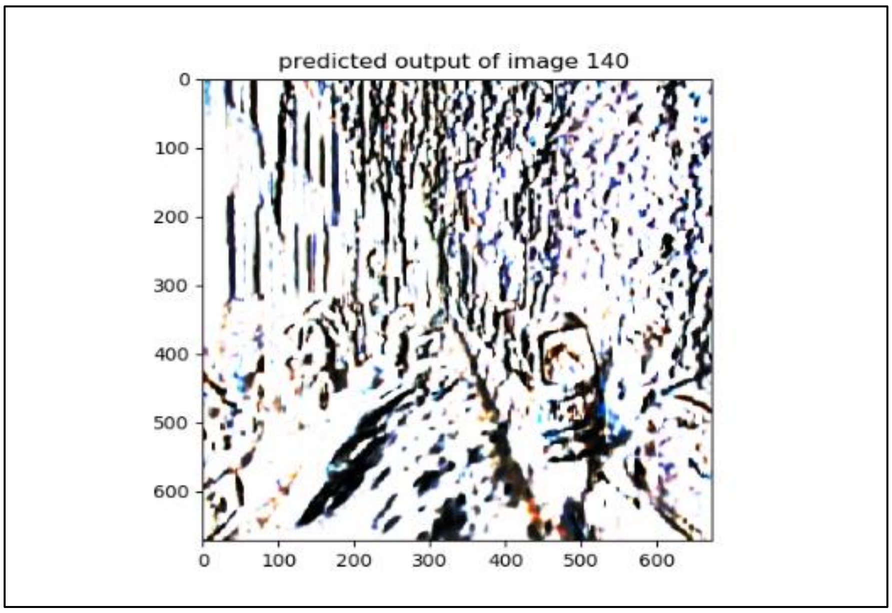
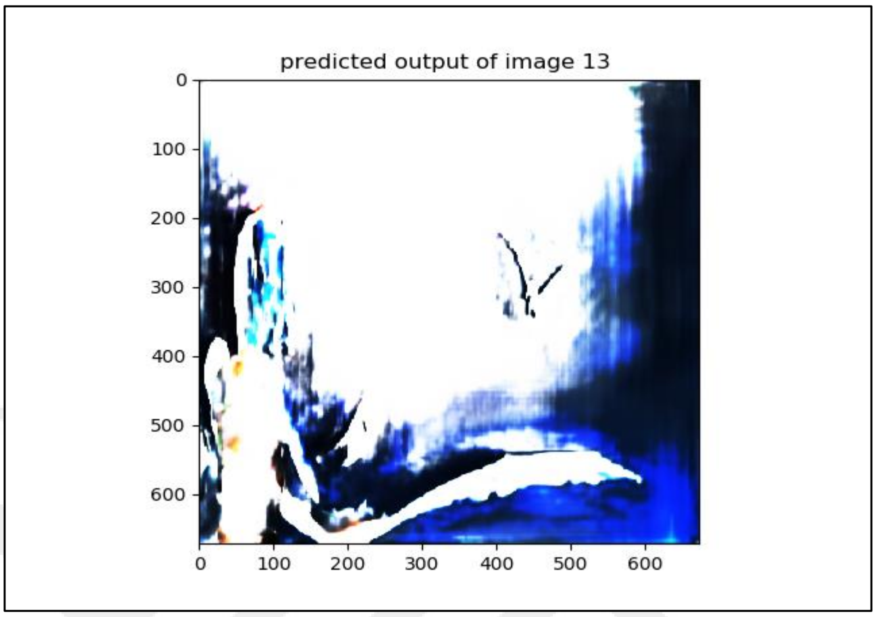


Figure 4.29: Evaluation Output for DenseMapNet Unsupervised Learning Model on MPI-Sintel Data Set



The following figure shows the training and validation reconstruction losses attained for the Symmetric Encoder-decoder based unsupervised learning Encoder-decoder:

Figure 4.30: Training and Validation Reconstruction Loss for Symmetric Network Unsupervised Learning Model on KITTI Data Set

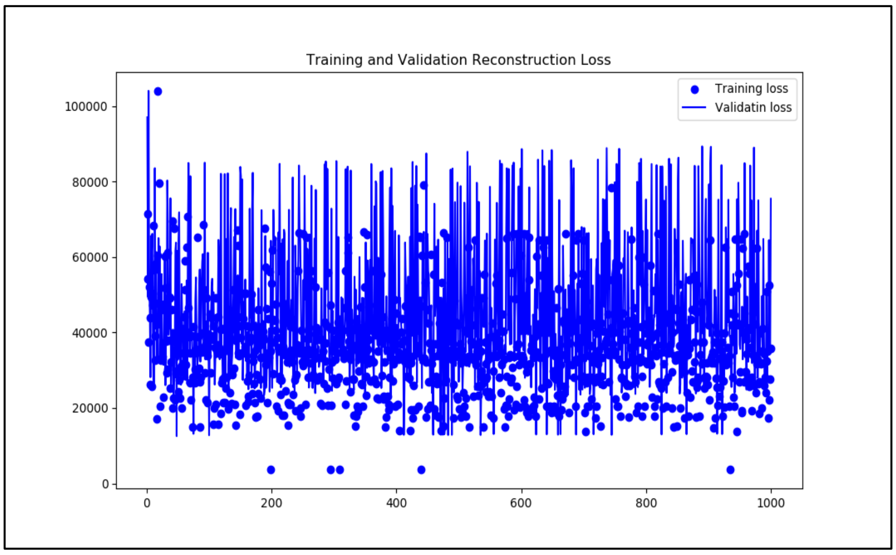
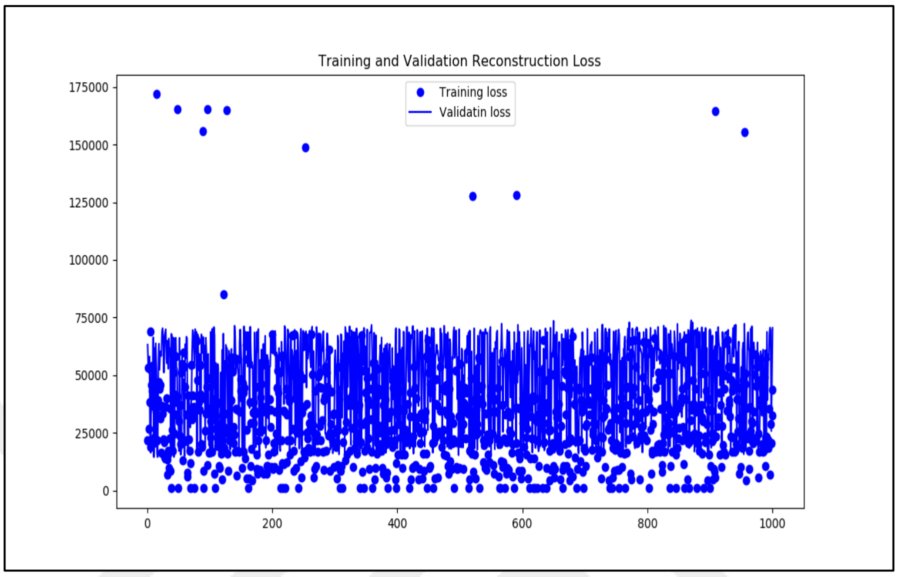


Figure 4.31: Training and Validation Reconstruction Loss for Symmetric Network Unsupervised Learning Model on MPI-Sintel Data Set



The following figure shows the evaluation disparity map result attained for the Symmetric Encoder-decoder based unsupervised learning Encoder-decoder:

Figure 4.32: Evaluation Output for Symmetric Network Unsupervised Learning Model on KITTI Data Set

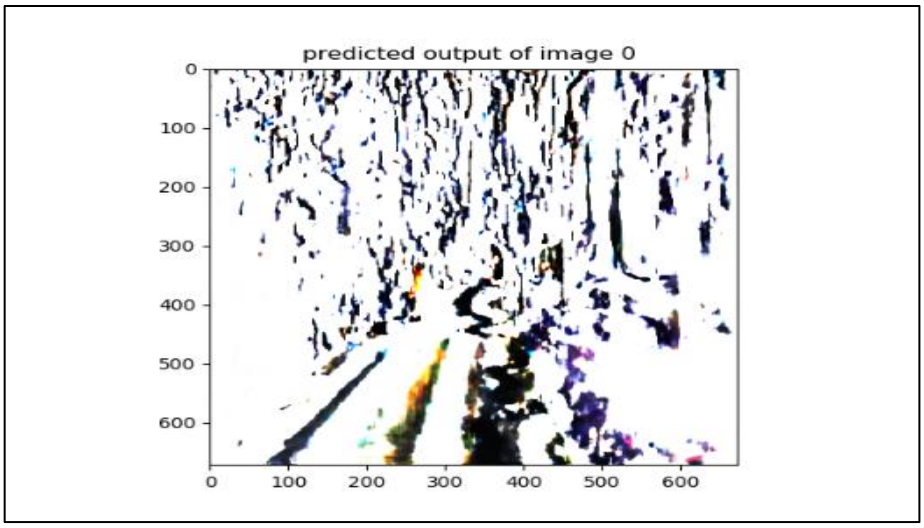
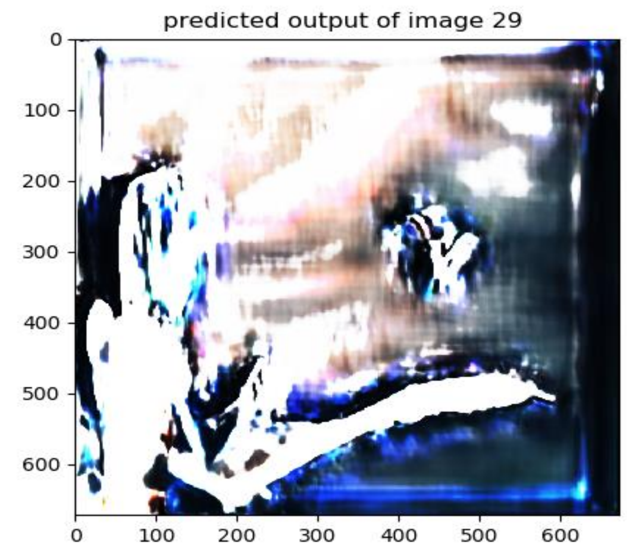


Figure 4.33: Evaluation Output for Symmetric Network Unsupervised Learning Model on MPI-Sintel Data Set



5. DISCUSSION

This thesis aims at formulating compact, efficient and specific Deep Learning models for the task of stereo vision by training different architecture designs that implement the latest techniques presented in literature with the proposed modifications.

It is observed from the implementation process that the hardware requirements for performing Deep Learning for computer vision and specifically CNN architectures are demanding in computational requirements most notably on the specifications of the hardware accelerators represented by the GPU, from the onset of the implementation the libraries and platforms employed required resent iteration of such hardware which are capable of supporting the compatible versions such as supporting Nvidia Cuda acceleration library of version 10 which is at the present an essential feature for execution of the implementations. Specific to the models is the GPU RAM consumption which further restricts the hardware choice as the general models with conventional convolution required RAM availability of more than 10 GB for preferable parameter sizes with some literature work operating with hardware envelop of multi GPU and multi CPU thus the main objective was to choose compact models and further reduce parameter requirements.

From the obtained results it is observed a strong dependence of the training and validation behaviour as well as output depth map detail and performance on the data set specification and characteristics, the main factor identified is the size of the data set as KITTI data set provide a lower number of example stereo pairs to the architecture to train on compared to MPI-Sintel which noticeably reflects on the detail provided by the output depth maps from architectures trained on these data sets respectively, from the training and validation accuracy and loss curves KIITI data set display the typical signs of overfitting within the first 100 epochs which is recorded on all supervised architectures and explainable by the low sample size of KITTI data set.

MPI-Sintel training and validation curves display a loss and accuracy saturation within a very small number of epochs but provide output performance that is more semantically relevant to the stereo vision task, this can be understood from the second factor identified that is data set characteristics and it is seen that MPI-Sintel is a synthetic data set having a lower complexity in relations of data points compared to KITTI data set that is generated from real world imaging therefore MPI-Sintel data set allows the architectures to capture the information within the first epochs while having the learning process balanced by the larger number of provided examples and the specifically constructed scenes with labelled ground truth images which prevent typical overfitting. In contrast KITTI data set outputs show lower semantic relevancy and the designs output depth maps that has a tendency to produce the salient features that are inherently present in LiDAR generated ground truth which is the presence of scan line repeated patterns which the networks overfit on due to optimization flow, also noticeable the high contrast of the produced features between up and down the detected horizon line due to the sharply increasing vanishing values of large distances typical to LiDAR data. Architecture wise the outputs obtained from pretrained supervised designs display weaker information capture and less semantic relevancy in the generated depth maps for which this can be explained with two factors:

Firstly, these designs are pretrained with a classification task on data sets containing millions of samples oriented for such a task thus the representation encoded in the weight values of these pretrained architectures are optimized for scale invariant representations that maximize relevant object detectability and classification this runs in contrast to the task of recovering information that defines scale.

Secondly, these architectures are large on the layer depth parameter making them less compact and more susceptible to overfitting on small data sets similar to the ones generally available to stereo vision task despite that the implementation only tunes the last layers of the pretrained system as an encoder to match the weights obtained for the decoder system.

The Inception based implementation provides larger contrast in the outputs this is attributed to Inception employing in its subunits the process of regularization through batch normalization and the presence of fan out structures that enhance the gradient flow in the backpropagation phase of training to overcome vanishing gradients.

The VGG based design displays greater detail for the generated depth maps as the VGG system captures greater detail due to less propensity to overfitting given by the smaller number of weight parameters allowing the optimizer to evolve the weight values to greater extent.

Based on the previous observations this work partakes into designing non-pretrained CNN Encoder-Decoder architectures that are specific to the task of generating stereo vision depth maps and constructed with compactness and low parameter size.

For the DenseMapNet based supervised learning architecture the observation of improvement in output detail with reducing parameter size relative to data size is implemented by employing the depth wise separable convolution technique and this design choice is validated by the generated output maps as it displays relevant depth annotation on semantic features and objects based on depth and this is most evident with training on MPI-Sintel data set. It is observed for this architecture a box artefact effects that introduce regions with variance in response activation to depth semantics this is due to the architecture structure that utilize a process of successive layer to layer input concatenation to alleviate vanishing gradient issue but introduce information mixing resulting in reconstruction artefacts. For the Siamese based architecture with skipping connection the design choice is to apply symmetry for the two stereo branches in kernel parameters and the layer structure of the Encoder-Decoder, moreover it is employed in this work a weight sharing mechanism in between the branches of the encoder system to enhance this symmetry with identical gradient calculations and reduction in parameter requirement which is further enhanced by using depth wise separable convolution.

A skipping connection mechanism is also introduced in two modes:

A typical encoder to decoder skipping connections connecting the feature extraction layers of the encoder with the decoder layers symmetric to it in kernel parameter size enhancing information flow.

A local to Encoder skip connection that connects the first information representations generated by the early encoding layers and introduce it the last encoding layers to further reduce information loss.

Another design choice is to merge the branches with a correlation extraction method that reduces parameters and emphasise on generating correspondence representations from the embedded information recovered from each branch.

It is observed an enhancement of generated depth maps compared to the DensemapNet architecture with more pronounced semantic depth identification and absence of box artefact behaviour present in the DenseMapNet design. Additionally the supervised learning Siamese based architecture extends the use of batch normalization to enhance the optimizer performance.

Considering that data sets provided with properly annotated ground truth images for the task of stereo vision are not prevalent relative to other computer vision tasks such as classification, Unsupervised Learning provides an opportunity to gain performance improvements while simplifying the data gathering process which requires either costly and complex Lidar based measurement systems or designing specific computer generated imagery of characteristics specific to obtain relevant depth ground truth.

Incorporated in the work the technique of self-supervised learning by customizing the Encoder-Decoder to train on KITTI and MPI-Sintel date sets with omitted labels for modified DenseMapNet architecture and Symmetric architecture with omitted skip connections.

It is observed that the applied technique of imposing a reconstruction loss acquired through the consistency in spatial information of scenes between left and right images produces different behaviours on the extracted depth maps.

While still suffering from overfitting effects on KITTI data set the consistency based measure manages to extract object outline information as displayed on the obtained outputs signifying an increase in feature embedding in the networks.

On MPI-Sintel the semantic depth information are more prominent on the outputs in accordance with the supervised method although displaying different artefact effects that shows a misidentification of highly illuminated regions as low depth and this activation is conserved in both the DenseMapNet and Symmetric architectures indicating that the artefact is a characteristic of the consistency measure and not specific to network design details.

6. CONCLUSIONS AND FUTURE WORK

In this part conclusions acquired and prospects for future work are presented.

6.1 CONCLUSIONS

It is concluded from this work presented in this thesis that the task of obtaining stereo vision depth maps utilizing deep learning shows a strong dependency on data size and data characteristics and most data sets publicly available for this task are oriented for the traditional stereo vision techniques in comparison with other tasks in computer vision such as classification and object detection where data sets are much larger with up to millions of samples, this has most impact in the supervised learning implementations for stereo vision. Another strong dependency is availability of hardware capability that is an essential requirement with deep convolutional architectures, mainline designs that are pretrained on large data sets with data centre level hardware provided to high corporations such as Inception and VGG are concluded to be not suitable for the stereo vision task due to different end goal criteria as well as the difficulty of fully retraining such very large designs within readily available hardware specifications. It is also concluded that for efficient Encoder-Decoder CNN architectures that are trained in a supervised learning setting for the task of generating stereo vision depth maps a Siamese design that emphasizes symmetry on the left and right encoder branches in both kernel parameter size and sharing of left and right weight values in addition to skip connection mechanism for information flow show an enhanced performance in extraction of semantically annotated depth information that the successively linked architecture of DenseNet inspired designs that suffer from reconstruction artefacts. Identified by this work the procedure of the depth wise separable convolution as a desirable Enhancements on parameter requirements resulting in more efficient implementations that critical to meet the hardware envelop provided to this work. Other enhancements are using batch normalization for mitigating the vanishing gradient effects and the use of correlation based merge of left and right feature maps. For the unsupervised learning setting this work concluded that it is applicable for the stereo vision task by implementing a self-learning reconstruction loss on right to left images on both a DenseMapNet architecture and a

Symmetric network architecture, this is very desirable as data sets for this learning setting can be constructed with less requirement and effort.
It is identified a conserved artefact effect in this setting which is high activation in depth maps for strongly illuminated section in the input stereo pair.

6.2 FUTURE WORK

Further future work venues can be identified based on this thesis, exploration of unsupervised and self-supervised architectures is still in its early stages and extending the consistency measure based reconstruction loss which can be taken to incorporate full simulation of the physical parameters such as light propagation characteristics or object movement profiles, alternatively the reconstruction loss can be specified to the outputs of highly precise although computationally intensive traditional depth estimation algorithms.
Data collection and generation in large sized and specific features for stereo vision is also a high recommendation in the supervised learning approach.
Another venue for research is to investigate hardware implementation of the recommended efficient architectures for fast real time applications such as Robotic navigation.

REFERENCES

Books

Alpaydin, E., 2010, *Introduction to Machine Learning*, 2nd edn. London: The MIT Press.

Bishop, C., 2006, *Pattern Recognition and Machine Learnin*g. Springer.

Kriesel, D., 2005, *A Brief Introduction to Neural Networks.*

Szeliski, R., 2010, *Computer Vision: Algorithms and Applications.* Springer.

Periodicals

Agarap, A., 2019. *Deep Learning using Rectified Linear Units (ReLU)*. [online] Available at: https://arxiv.org/pdf/1803.08375.pdf [accessed 15 July 2019]

Atienza, R., 2018. *Fast Disparity Estimation using Dense Networks.* [online] Available at: https://arxiv.org/pdf/1805.07499.pdf [accessed 6 May 2019]

Badrinarayanan,V. et. al., 2017. *SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation*. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) [online] Available at: https://www.repository.cam.ac.uk/bitstream/handle/1810/271007/07803544.pdf?sequence=7 [accessed 22 November 2019]

Bosch, M. et. al., 2019. *Semantic Stereo for Incidental Satellite Images*. 2019 IEEE Winter Conference on Applications of Computer Vision (WACV) [online] Available at: https://arxiv.org/ftp/arxiv/papers/1811/1811.08739.pdf [accessed 6 May 2019]

Butler,D. et. al., 2012. *A Naturalistic Open Source Movie for Optical Flow Evaluation.* European Conference on Computer Vision ECCV 2012 [online] Available at: https://homes.cs.washington.edu/~djbutler/papers/ButlerECCV2012.pdf [accessed 5 April 2019]

Chollet, F., 2017. *Xception: Deep Learning with Depthwise Separable Convolutions*. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* [online] Available at: http://openaccess.thecvf.com/content_cvpr_2017/papers/Chollet_Xception_Deep_Learning_CVPR_2017_paper.pdf [accessed 10 July 2019]

Deng, J., Dong, W., Socher, R., Li, L., 2009. *ImageNet: A Large-Scale Hierarchical Image Database*. IEEE *Computer Vision and Pattern Recognition (CVPR)* [online] Available at: http://www.image-net.org/papers/imagenet_cvpr09.pdf [accessed 6 May 2019]

Geiger,A. et. al.,2013. *Vision meets robotics: The KITTI dataset* [online] Available at: http://ww.cvlibs.net/publications/Geiger2013IJRR.pdf [accessed 5 April 2019]

Guo,J. et.al.,2018. *Network Decoupling: From Regular to Depthwise Separable Convolutions*. BMVC 2018 [online] Available at: http://bmvc2018.org/contents/papers/0849.pdf [accessed 10 July 2019]

Ioffe, S., Szegedy, C., 2015. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. *ICML 2015* [online], Available at: https://arxiv.org/pdf/1502.03167.pdf [accessed 10 July 2019]

Ji, M. et. al., 2017, '*SurfaceNet: An End-to-End 3D Neural Network for Multiview Stereopsis*. 2017 IEEE International Conference on Computer Vision (ICCV) [online] Available at: https://arxiv.org/pdf/1708.01749.pdf [accessed 15 July 2019]

Jiang,H. et. al., 2018. *Self-Supervised Relative Depth Learning for Urban Scene Understanding*. European Conference on Computer Vision ECCV 2018 [online] Available at: https://arxiv.org/pdf/1712.04850.pdf [accessed 14 August 2019]

Jiao, J. et. al. ,,2014. *Local Stereo Matching with Improved Matching Cost and Disparity Refinement*. *IEEE MultiMedia*, [online] Available at: https://jianbojiao.com/pdfs/mm.pdf [accessed 10 July 2019]

Kendall,A. et. al., 2017. *End-to-End Learning of Geometry and Context for Deep Stereo Regression*. 2017 IEEE International Conference on Computer Vision (ICCV) [online] Available at: https://arxiv.org/pdf/1703.04309.pdf [accessed 10 July 2019]

Kingma, D.,Welling,M.,2014. *Auto-Encoding Variational Bayes*. *ICLR 2014* [online] Available at: https://arxiv.org/pdf/1312.6114.pdf [accessed 10 July 2019]

Koch, G. et. al., 2015. *Siamese Neural Networks for One-Shot Image Recognition*. [online] Available at: https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf [accessed 12 July 2019]

Luo, W. et.al. , 2016. *Efficient Deep Learning for Stereo Matching*. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* [online] Available at: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Luo_Efficient_Deep_Learning_CVPR_2016_paper.pdf
[accessed 10 July 2019]

Mazaheri, G. et. al., 2019. *A Skip Connection Architecture for Localization of Image Manipulations*. CVPR Workshops 2019. [online] Available at:

http://openaccess.thecvf.com/content_CVPRW_2019/papers/Media%20Forensics/Mazaheri_A_Skip_Connection_Architecture_for_Localization_of_Image_Manipulations_CVPRW_2019_paper.pdf [accessed 12 July 2019]

Nwankpa, C. et. al.,2018. *Activation Functions: Comparison of Trends in Practice and Research for Deep Learning*. [online] Available at: https://arxiv.org/pdf/1811.03378.pdf [accessed 15 July 2019]

Ramachandran, P. et. al., 2017. *Swish: a Self-Gated Activation Function*. [online] Available at: https://pdfs.semanticscholar.org/4f57/f486adea0bf95c252620a4e8af39232ef8bc.pdf [accessed 10 July 2019]

Szegedy, C. et. al., 2014. *Going Deeper with Convolutions. IEEE* [online] Available at: https://arxiv.org/pdf/1409.4842.pdf [accessed 6 May 2019]

Tardon, et.al. 2011. *Markov Random Fields in the Context of Stereo Vision* [online] Available at: https://www.researchgate.net/publication/221909970_Markov_Random_Fields_in_the_Context_of_Stereo_Vision [accessed 10 July 2019]

Trucco, Emanuele, Roberto, Vito, Tinonin, Corbatto, 1970. *SSD Disparity Estimation for Dynamic Stereo*. [online] Available at: https://www.researchgate.net/publication/2600153_SSD_Disparity_Estimation_for_Dynamic_Stereo [accessed 10 July 2019]

Wang, F. et. al.,2018. *Self-Supervised Learning of Depth and Camera Motion from 360◦ Videos*' [online] Available at: https://arxiv.org/pdf/1811.05304.pdf [accessed 14 August 2019]

Wei, Hu, Hao, Zhou, Hua-wei, Lau, 2019. *Characterizing Rock Facies Using Machine Learning Algorithm Based on a Convolutional Neural Network and Data Padding Strategy*. Pure and Applied Geophysics. 1-13. 10.1007/s00024-019-02152-0. [online] Available at: https://www.researchgate.net/publication/331991539_Characterizing_Rock_Facies_Using_Machine_Learning_Algorithm_Based_on_a_Convolutional_Neural_Network_and_Data_Padding_Strategy [accessed 10 July 2019]

Welchman,A.,2016 *The Human Brain in Depth: How We See in 3D*', The Annual Review of Vision Science 2016. [online] Available at: https://www.repository.cam.ac.uk/bitstream/handle/1810/271007/07803544.pdf?sequence=7 [accessed 15 June 2019]

Yasrab,R. et. al.,2017. *An Encoder-Decoder Based Convolution Neural Network (CNN) for Future Advanced Driver Assistance System (ADAS)*. Applied Sciences [online] Available at: https://pdfs.semanticscholar.org/0510/00dd99635b74862b4b33f6dd84aca34f7471.pdf?_ga=2.120936589.429826482.1576776997-284257306.1530051971 [accessed 10 July 2019]

Ye, J.,Sung,W.,2019, '*Understanding Geometry of Encoder-Decoder CNNs*',[online], Available at: https://arxiv.org/pdf/1901.07647.pdf [accessed 10 July 2019]

Zhang, Y. et. al.,2018. *ActiveStereoNet: End-to-End Self-Supervised Learning for Active Stereo Systems*. European Conference on Computer Vision ECCV 2018 [online] Available at: http://openaccess.thecvf.com/content_ECCV_2018/papers/Yinda_Zhang_Active_Stereo_Net_ECCV_2018_paper.pdf [accessed 14 August 2019]

Other Publications

Ewbank, T., 2017. *Efficient and precise stereoscopic vision for humanoid robots*. [online] Available at: https://matheo.uliege.be/bitstream/2268.2/3144/5/master-thesis-Tom-Ewbank.pdf [accessed 1 May 2019]

Goodfellow,I. et.al., 2016. *Deep Learning ',MIT press 2016*. [online] Available at: http://www.deeplearningbook.org [accessed 1 May 2019]

Kar, A., 2017. *Learning to Reconstruct 3D Objects*. [online] Available at: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-199.pdf [accessed 1 May 2019]

Nilsen,M.,2015. *Neural Networks and Deep Learning*. Determination press 2015 [online], Available at: http://neuralnetworksanddeeplearning.com [accessed 1 May 2019]

Pisapia, R., 2016. *Disparity map extraction for a low cost 3D sensor*. [online] Available at: https://members.loria.fr/SATabbone/Disparity%20map%20extraction%20for%20a%20low%20cost%203D%20sensor%20-%20Pisapia%20Roberto.pdf [accessed 1 May 2019]

Poggi, M., 2017. *Deep Learning for stereo matching and related tasks*. [online] Available at: http://vision.disi.unibo.it/~mpoggi/talks/Deep_learning_stereo.pdf [accessed 10 July 2019]

Stigborn, P., 2018. *Generating 3D-objects using neural networks*. [online] Available at: http://www.diva-portal.org/smash/get/diva2:1218064/FULLTEXT01.pdf [accessed 1 May 2019]

APPENDICES

Appendix A.1 Table

Table-1: Supervised Model with Parameter reduced DenseMapNet

Layer(type)	**Output Shape**	**Parameters Number**	**Connected to**
input_1 (Input layer)	(None,748,744,3)	0	*
input_2 (Input layer)	(None,748,744,3)	0	*
concatenate_1 (Concatenate)	(None,748,744,6)	0	input_1[0][0] input_2[0][0]
separable_conv2d_3 (SeparableConvolution)	(None, 748, 744, 128)	1046	concatenate_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 93, 93, 128)	0	separable_conv2d_3 [0][0]
batch_normalization_1 (BatchNormalization)	(None, 93, 93, 128)	512	max_pooling2d_1[0] [0]
downsampled_stereo (Activation)	(None, 93, 93, 128)	0	batch_normalization _1[0][0]
separable_conv2d_4 (SeparableConvolution)	(None, 93, 93, 64)	11456	downsampled_stereo [0][0]
separable_conv2d_5 (SeparableConvolution)	(None, 93, 93, 64)	11456	downsampled_stereo [0][0]
dropout_1 (Dropout)	(None, 93, 93, 64)	0	separable_conv2d_4 [0][0]
separable_conv2d_6 (SeparableConvolution)	(None, 93, 93, 64)	11456	downsampled_stereo [0][0]
dropout_2 (Dropout)	(None, 93, 93, 64)	0	separable_conv2d_5 [0][0]

concatenate_2 (Concatenate)	(None, 93, 93, 192)	0	dropout_1[0][0] downsampled_stere [0][0]
separable_conv2d_7 (SeparableConvolution)	(None, 93, 93, 64)	11456	downsampled_stereo [0][0]
dropout_3 (Dropout)	(None, 93, 93, 64)	0	separable_conv2d_6 [0][0]
concatenate_3 (Concatenate)	(None, 93, 93, 256)	0	dropout_2[0][0] concatenate_2[0][0]
separable_conv2d_1 (SeparableConvolution)	(None, 748, 744, 128)	587	input_1[0][0]
dropout_4 (Dropout)	(None, 93, 93, 64)	0	separable_conv2d_7 [0][0]
concatenate_4 (Concatenate)	(None, 93, 93, 320)	0	dropout_3[0][0] concatenate_3[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 93, 93, 128)	0	separable_conv2d_1 [0][0]
concatenate_5 (Concatenate)	(None, 93, 93, 384)	0	dropout_4[0][0] concatenate_4[0][0]
concatenate_6 (Concatenate)	(None, 93, 93, 512)	0	max_pooling2d_2 [0][0] concatenate_5 [0][0]
batch_normalization_2 (BatchNormalization)	(None, 93, 93, 512)	2048	concatenate_6[0][0]
activation_1 (Activation)	(None, 93, 93, 512)	0	batch_normalization _2[0][0]
separable_conv2d_8 (SeparableConvolution)	(None, 93, 93, 64)	33344	activation_1[0][0]
batch_normalization_3 (BatchNormalization)	(None, 93, 93, 64)	256	separable_conv2d_8 [0][0]

activation_2 (Activation)	(None, 93, 93, 64)	0	batch_normalization _3[0][0]
conv2d_1 (Conv2D)	(None, 93, 93, 64)	102464	activation_2[0][0]
dropout_5 (Dropout)	(None, 93, 93, 64)	0	conv2d_1[0][0]
concatenate_7 (Concatenate)	(None, 93, 93, 576)	0	concatenate_6[0][0] dropout_5[0][0]
batch_normalization_4 (BatchNormalization)	(None, 93, 93, 576)	2304	concatenate_7[0][0]
activation_3 (Activation)	(None, 93, 93, 576)	0	batch_normalization _4[0][0]
separable_conv2d_9 (SeparableConvolution)	(None, 93, 93, 64)	37504	activation_3[0][0]
batch_normalization_5 (BatchNormalization)	(None, 93, 93, 64)	256	separable_conv2d_9 [0][0]
activation_4 (Activation)	(None, 93, 93, 64)	0	batch_normalization _5[0][0]
conv2d_2 (Conv2D)	(None, 93, 93, 64)	102464	activation_4[0][0]
dropout_6 (Dropout)	(None, 93, 93, 64)	0	conv2d_2[0][0]
concatenate_8 (Concatenate)	(None, 93, 93, 640)	0	concatenate_7[0][0] dropout_6[0][0]
batch_normalization_6 (BatchNormalization)	(None, 93, 93, 640)	2560	concatenate_8[0][0]
activation_5 (Activation)	(None, 93, 93, 640)	0	batch_normalization _6[0][0]
separable_conv2d_10 (SeparableConvolution)	(None, 93, 93, 64)	41664	activation_5[0][0]
batch_normalization_7 (BatchNormalization)	(None, 93, 93, 64)	256	separable_conv2d_1 0[0][0]
activation_6 (Activation)	(None, 93, 93, 64)	0	batch_normalization _7[0][0]
conv2d_3 (Conv2D)	(None, 93, 93, 64)	102464	activation_6[0][0]

dropout_7 (Dropout)	(None, 93, 93, 64)	0	conv2d_3[0][0]
concatenate_9 (Concatenate)	(None, 93, 93, 704)	0	concatenate_8[0][0] dropout_7[0][0]
batch_normalization_8 (BatchNormalization)	(None, 93, 93, 704)	2816	concatenate_9[0][0]
activation_7 (Activation)	(None, 93, 93, 704)	0	batch_normalization _8[0][0]
separable_conv2d_11 (SeparableConvolution)	(None, 93, 93, 64)	45824	activation_7[0][0]
batch_normalization_9 (BatchNormalization)	(None, 93, 93, 64)	256	separable_conv2d_1 1[0][0]
activation_8 (Activation)	(None, 93, 93, 64)	0	batch_normalization _9[0][0]
conv2d_4 (Conv2D)	(None, 93, 93, 64)	102464	activation_8[0][0]
dropout_8 (Dropout)	(None, 93, 93, 64)	0	conv2d_4[0][0]
upsampled_disparity (Concatenate)	(None, 93, 93, 768)	0	concatenate_9[0][0] dropout_8[0][0]
batch_normalization_10 (BatchNormalization)	(None, 93, 93, 768)	3072	upsampled_disparity [0][0]
activation_9 (Activation)	(None, 93, 93, 768)	0	batch_normalization _10[0][0]
separable_conv2d_12 (SeparableConvolution)	(None, 93, 93, 64)	49984	activation_9[0][0]
up_sampling2d_1 (UpSampling2D)	(None, 744, 744, 64)	0	separable_conv2d_1 2[0][0]
zero_padding2d_1 (ZeroPadding2D	(None, 748, 744, 64)	0	up_sampling2d_1 [0] [0]
separable_conv2d_2 (SeparableConvolution)	(None, 748, 744, 1)	79	input_1[0][0]
concatenate_10 (Concatenate)	(None, 748, 744, 65)	0	zero_padding2d_1 [0][0]

			separable_conv2d_2 [0][0]
batch_normalization_11 (BatchNormalization)	(None, 748, 744, 65)	260	concatenate_10[0][0]
activation_10 (Activation)	(None, 748, 744, 65)	0	batch_normalization _11[0][0]
separable_conv2d_13 (SeparableConvolution)	(None, 748, 744, 64)	5849	activation_10[0][0]
concatenate_11 (Concatenate)	(None, 748, 744, 129)	0	concatenate_10 [0][0] separable_conv2d_1 3[0][0]
batch_normalization_12 (BatchNormalization)	(None, 748, 744, 129)	516	concatenate_11[0][0]
activation_11 (Activation)	(None, 748, 744, 129)	0	batch_normalization _12[0][0]
separable_conv2d_14 (SeparableConvolution)	(None, 748, 744, 3)	10839	activation_11[0][0]
disparity_output (Activation)	(None, 748, 744, 3)	0	separable_conv2d_1 4[0][0]

Table-2: Supervised Model with Siamese Network

Layer(type)	Output Shape	Parameters Number	Connected to
input_1 (Input layer)	(None, 672, 672, 3)	0	*
input_2 (Input layer)	(None, 672, 672, 3)	0	*
separable_conv2d_1 (SeparableConvolution)	(None, 672, 672, 32)	275	input_1[0][0] input_2[0][0]
batch_normalization_1 (BatchNormalization)	(None, 672, 672, 32)	128	separable_conv2d_1 [0][0] separable_conv2d_1 [1][0]
activation_1 (Activation)	(None, 672, 672, 32)	0	batch_normalization_1[0][0] batch_normalization_1[1][0]
max_pooling2d_1 (MaxPooling2D)	(None, 336, 336, 32)	0	activation_1[0][0] activation_1[1][0]
separable_conv2d_2 (SeparableConvolution)	(None, 336, 336, 64)	3680	max_pooling2d_1 [0][0] max_pooling2d_1 [1][0]
batch_normalization_2 (BatchNormalization)	(None, 336, 336, 64)	256	separable_conv2d_2 [0][0] separable_conv2d_2 [1][0]
activation_2 (Activation)	(None, 336, 336, 64)	0	batch_normalization_2[0][0]

			batch_normalization_2[1][0]
max_pooling2d_2 (MaxPooling2D)	(None, 168, 168, 64)	0	activation_2[0][0] activation_2[1][0]
separable_conv2d_3 (SeparableConvolution)	(None, 168, 168, 128)	9920	max_pooling2d_2 [0][0] max_pooling2d_2 [1][0]
batch_normalization_3 (BatchNormalization)	(None, 168, 168, 128)	512	separable_conv2d_3 [0][0] separable_conv2d_3 [1][0]
activation_3 (Activation)	(None, 168, 168, 128)	0	batch_normalization_3 [0][0] batch_normalization_3[1][0]
max_pooling2d_5 (MaxPooling2D)	(None, 84, 84, 32)	0	activation_1[0][0] activation_1[1][0]
max_pooling2d_3 (MaxPooling2D)	(None, 84, 84, 128)	0	activation_3[0][0] activation_3[1][0]
concatenate_1 (Concatenate)	(None, 84, 84, 160)	0	max_pooling2d_5 [0][0] max_pooling2d_3 [0][0]
concatenate_2 (Concatenate)	(None, 84, 84, 160)	0	max_pooling2d_5 [1][0] max_pooling2d_3 [1][0]
separable_conv2d_4 (SeparableConvolution)	(None, 84, 84, 256)	42656	concatenate_1[0][0] concatenate_2[0][0]

batch_normalization_4 (BatchNormalization)	(None, 84, 84, 256)	1024	separable_conv2d_4 [0][0] separable_conv2d_4 [1][0]
activation_4 (Activation)	(None, 84, 84, 256)	0	batch_normalization_4[0][0] batch_normalization_4[1][0]
max_pooling2d_4 (MaxPooling2D)	(None, 42, 42, 256)	0	activation_4[0][0] activation_4[1][0]
multiply_1 (Multiply)	(None, 42, 42, 256)	0	max_pooling2d_4 [0][0] max_pooling2d_4 [1][0]
separable_conv2d_5 (SeparableConvolution)	(None, 42, 42, 256)	66816	multiply_1[0][0]
batch_normalization_5 (BatchNormalization)	(None, 42, 42, 256)	1024	separable_conv2d_5 [0][0]
activation_5 (Activation)	(None, 42, 42, 256)	0	batch_normalization_5[0][0]
max_pooling2d_6 (MaxPooling2D)	(None, 21, 21, 256)	0	activation_5[0][0]
separable_conv2d_6 (SeparableConvolution)	(None, 21, 21, 256)	66816	max_pooling2d_6 [0][0]
batch_normalization_6 (BatchNormalization)	(None, 21, 21, 256)	1024	separable_conv2d_6 [0][0]
activation_6 (Activation)	(None, 21, 21, 256)	0	batch_normalization_6[0][0]
up_sampling2d_1 (UpSampling2D)	(None, 42, 42, 256)	0	activation_6[0][0]

concatenate_3 (Concatenate)	(None, 42, 42, 512)	0	up_sampling2d_1 [0][0] activation_5[0][0]
separable_conv2d_7 (SeparableConvolution)	(None, 42, 42, 256)	135936	concatenate_3[0][0]
batch_normalization_7 (BatchNormalization)	(None, 42, 42, 256)	1024	separable_conv2d_ 7 [0][0]
activation_7 (Activation)	(None, 42, 42, 256)	0	batch_normalizatio n_7[0][0]
up_sampling2d_2 (UpSampling2D)	(None, 84, 84, 256)	0	activation_7[0][0]
concatenate_4 (Concatenate)	(None, 84, 84, 512)	0	up_sampling2d_2 [0][0] activation_4[0][0]
separable_conv2d_8 (SeparableConvolution)	(None, 84, 84, 128)	78464	concatenate_4[0][0]
batch_normalization_8 (BatchNormalization)	(None, 84, 84, 128)	512	separable_conv2d_ 8 [0][0]
activation_8 (Activation)	(None, 84, 84, 128)	0	batch_normalizatio n_8[0][0]
up_sampling2d_3 (UpSampling2D)	(None, 168, 168, 128)	0	activation_8[0][0]
concatenate_5 (Concatenate)	(None, 168, 168, 256)	0	up_sampling2d_3 [0][0] activation_3[0][0]
separable_conv2d_9 (SeparableConvolution)	(None, 168, 168, 64)	28992	concatenate_5[0][0]
batch_normalization_9 (BatchNormalization)	(None, 168, 168, 64)	256	separable_conv2d_ 9 [0][0]
activation_9 (Activation)	(None, 168, 168, 64)	0	batch_normalizatio n_9[0][0]

up_sampling2d_4 (UpSampling2D)	(None, 336, 336, 64)	0	activation_9[0][0]
concatenate_6 (Concatenate)	(None, 336, 336, 128)	0	up_sampling2d_4 [0][0] activation_2[0][0]
separable_conv2d_10 (SeparableConvolution)	(None, 336, 336, 32)	10400	concatenate_6[0][0]
batch_normalization_10 (BatchNormalization)	(None, 336, 336, 32)	128	separable_conv2d_10[0][0]
activation_10 (Activation)	(None, 336, 336, 32)	0	batch_normalization_10[0][0]
up_sampling2d_5 (UpSampling2D)	(None, 672, 672, 32)	0	activation_10[0][0]
up_sampling2d_6 (UpSampling2D)	(None, 672, 672, 256)	0	activation_6[0][0]
concatenate_7 (Concatenate)	(None, 672, 672, 320)	0	up_sampling2d_5 [0][0] activation_1[0][0] up_sampling2d_6 [0][0]
separable_conv2d_11 (SeparableConvolution)	(None, 672, 672, 3)	3843	concatenate_7[0][0]

Table-3: Unsupervised Model with DenseMapNet Encoder-decoder

Layer(type)	Output Shape	Parameters Number	Connected to
input_1 (Input layer)	(None, 672, 672, 3)	0	*
input_2 (Input layer)	(None, 672, 672, 3)	0	*
model_1 (Model)	(None, 672, 672, 3)	753166	input_1[0][0]
custom_layer_8 (CustomLayer)	(None, 672, 672, 3)	0	input_1[0][0] input_2[0][0] model_1[1][0]
model_1 (Model)			
input_1(InputLayer)	(None, 672, 672, 3)	0	*
separable_conv2d_17 (SeparableConvolution)	(None, 672, 672, 128	587	input_1[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 84, 84, 128)	0	separable_conv2d_17 [0][0]
batch_normalization_13 (BatchNormalization)	(None, 84, 84, 128)	512	max_pooling2d_3 [0][0]
downsampled_stereo (Activation)	(None, 84, 84, 128)	0	batch_normalization_ 13[0][0]
separable_conv2d_18 (SeparableConvolution)	(None, 84, 84, 128)	19712	downsampled_ stereo [0][0]
separable_conv2d_19 (SeparableConvolution)	(None, 84, 84, 128)	19712	downsampled_stereo [0][0]

dropout_9 (Dropout)	(None, 84, 84, 128)	0	separable_conv2d_18 [0][0]
separable_conv2d_20 (SeparableConvolution)	(None, 84, 84, 128)	19712	downsampled_stereo [0][0]
dropout_10 (Dropout)	(None, 84, 84, 128)	0	separable_conv2d_19 [0][0]
concatenate_10 (Concatenate)	(None, 84, 84, 256)	0	dropout_9[0][0] downsampled_stereo [0][0]
separable_conv2d_21 (SeparableConvolution)	(None, 84, 84, 128)	19712	downsampled_stereo [0][0]
dropout_11 (Dropout)	(None, 84, 84, 128)	0	separable_conv2d_20 [0][0]
concatenate_11 (Concatenate)	(None, 84, 84, 384)	0	dropout_10[0][0] concatenate_10[0][0]
dropout_12 (Dropout)	(None, 84, 84, 128)	0	separable_conv2d_21 [0][0]
concatenate_12 (Concatenate)	(None, 84, 84, 512)	0	dropout_11[0][0] concatenate_11[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 84, 84, 128)	0	separable_conv2d_17 [0][0]
concatenate_13 (Concatenate)	(None, 84, 84, 640)	0	dropout_12[0][0] concatenate_12[0][0]
concatenate_14 (Concatenate)	(None, 84, 84, 768)	0	max_pooling2d_4 [0][0] concatenate_13[0][0]
batch_normalization_14 (BatchNormalization)	(None, 84, 84, 768)	3072	concatenate_14[0][0]
activation_12 (Activation)	(None, 84, 84, 768)	0	batch_normalization_14[0][0]

separable_conv2d_22 (SeparableConvolution)	(None, 84, 84, 128)	99200	activation_12[0][0]
batch_normalization_15 (BatchNormalization)	(None, 84, 84, 128)	512	separable_conv2d_22 [0][0]
activation_13 (Activation)	(None, 84, 84, 128)	0	batch_normalization_ 15[0][0]
separable_conv2d_23 (SeparableConvolution)	(None, 84, 84, 64)	11456	activation_13[0][0]
dropout_13 (Dropout)	(None, 84, 84, 64)	0	separable_conv2d_23 [0][0]
concatenate_15 (Concatenate)	(None, 84, 84, 832)	0	concatenate_14[0][0] dropout_13[0][0]
batch_normalization_16 (BatchNormalization)	(None, 84, 84, 832)	3328	concatenate_15[0][0]
activation_14 (Activation)	(None, 84, 84, 832)	0	batch_normalization_ 16[0][0]
separable_conv2d_24 (SeparableConvolution)	(None, 84, 84, 128)	107456	activation_14[0][0]
batch_normalization_17 (BatchNormalization)	(None, 84, 84, 128)	512	separable_conv2d_24 [0][0]
activation_15 (Activation)	(None, 84, 84, 128)	0	batch_normalization_ 17[0][0]
separable_conv2d_25 (SeparableConvolution)	(None, 84, 84, 64)	11456	activation_15[0][0]
dropout_14 (Dropout)	(None, 84, 84, 64)	0	separable_conv2d_25 [0][0]
concatenate_16 (Concatenate)	(None, 84, 84, 896)	0	concatenate_15[0][0] dropout_14[0][0]
batch_normalization_18 (BatchNormalization)	(None, 84, 84, 896)	3584	concatenate_16[0][0]

activation_16 (Activation)	(None, 84, 84, 896)	0	batch_normalization_18[0][0]
separable_conv2d_26 (SeparableConvolution)	(None, 84, 84, 128)	115712	activation_16[0][0]
batch_normalization_19 (BatchNormalization)	(None, 84, 84, 128)	512	separable_conv2d_26 [0][0]
activation_17 (Activation)	(None, 84, 84, 128)	0	batch_normalization_19[0][0]
separable_conv2d_27 (SeparableConvolution)	(None, 84, 84, 64)	11456	activation_17[0][0]
dropout_15 (Dropout)	(None, 84, 84, 64)	0	separable_conv2d_27 [0][0]
concatenate_17 (Concatenate)	(None, 84, 84, 960)	0	concatenate_16[0][0] dropout_15[0][0]
batch_normalization_20 (BatchNormalization)	(None, 84, 84, 960)	3840	concatenate_17[0][0]
activation_18 (Activation)	(None, 84, 84, 960)	0	batch_normalization_20[0][0]
separable_conv2d_28 (SeparableConvolution)	(None, 84, 84, 128)	123968	activation_18[0][0]
batch_normalization_21 (BatchNormalization)	(None, 84, 84, 128)	512	separable_conv2d_28 [0][0]
activation_19 (Activation)	(None, 84, 84, 128)	0	batch_normalization_21[0][0]
separable_conv2d_29 (SeparableConvolution)	(None, 84, 84, 64)	11456	activation_19[0][0]
dropout_16 (Dropout)	(None, 84, 84, 64)	0	separable_conv2d_29 [0][0]
upsampled_disparity (Concatenate)	(None, 84, 84, 1024)	0	concatenate_17[0][0] dropout_16[0][0]

batch_normalization_22 (SeparableConvolution)	(None, 84, 84, 1024)	4096	upsampled_disparity [0][0]
activation_20 (Activation)	(None, 84, 84, 1024)	0	batch_normalization_ 22[0][0]
separable_conv2d_30 (SeparableConvolution)	(None, 84, 84, 128)	132224	activation_20[0][0]
up_sampling2d_2 (UpSampling2D)	(None, 672, 672, 128)	0	separable_conv2d_30 [0][0]
batch_normalization_23 (BatchNormalization)	(None, 672, 672, 128)	512	up_sampling2d_2 [0][0]
activation_21 (Activation)	(None, 672, 672, 128)	0	batch_normalization_ 23[0][0]
separable_conv2d_31 (SeparableConvolution)	(None, 672, 672, 64)	11456	activation_21[0][0]
concatenate_18 (Concatenate)	(None, 672, 672, 192)	0	up_sampling2d_2 [0][0] separable_conv2d_31 [0][0]
batch_normalization_24 (SeparableConvolution)	(None, 672, 672, 192)	768	concatenate_18[0][0]
activation_22 (Activation)	(None, 672, 672, 192)	0	batch_normalization_ 24[0][0]
separable_conv2d_32 (SeparableConvolution)	(None, 672, 672, 3)	16131	activation_22[0][0]
disparity_output (Activation)	(None, 672, 672, 3)	0	separable_conv2d_32 [0][0]

Table-4: Unsupervised Model with Symmetric Encoder-decoder

Layer(type)	**Output Shape**	**Parameters Number**	**Connected to**
input_1 (Input layer)	(None, 672, 672, 3)	0	*
input_2 (Input layer)	(None, 672, 672, 3)	0	*
model_1 (Model)	(None, 672, 672, 3)	661782	input_1[0][0]
custom_layer_8 (CustomLayer)	(None, 672, 672, 3)	0	input_1[0][0] input_2[0][0] model_1[1][0]
model_1(Model)			
input_1 (InputLayer)	(None, 672, 672, 3)	0	*
separable_conv2d_1 (SeparableConvolution)	(None, 672, 672, 64)	403	input_1 [0][0]
batch_normalization_1 (BatchNormalization)	(None, 672, 672, 64)	256	separable_conv2d_1 [0][0]
activation_1 (Activation)	(None, 672, 672, 64)	0	batch_normalization _1
max_pooling2d_1 (MaxPooling2D)	(None, 336, 336, 64)	0	activation_1[0][0]
separable_conv2d_2 (SeparableConvolution)	(None, 336, 336, 128)	11456	max_pooling2d_1 [0][0]
batch_normalization_2 (BatchNormalization)	(None, 336, 336, 128)	512	separable_conv2d_2 [0][0]
activation_2 (Activation)	(None, 336, 336, 128)	0	batch_normalization _2[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 168, 168, 128)	0	activation_2[0][0]

separable_conv2d_3 (SeparableConvolution)	(None, 168, 168, 256)	36224	max_pooling2d_2 [0][0]
batch_normalization_3 (BatchNormalization)	(None, 168, 168, 256)	1024	separable_conv2d_3 [0][0]
activation_3 (Activation)	(None, 168, 168, 256)	0	batch_normalization _3[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 84, 84, 256)	0	activation_3[0][0]
separable_conv2d_4 (SeparableConvolution)	(None, 84, 84, 512)	133888	max_pooling2d_3 [0][0]
batch_normalization_4 (BatchNormalization)	(None, 84, 84, 512)	2048	separable_conv2d_4 [0][0]
activation_4 (Activation)	(None, 84, 84, 512)	0	batch_normalization _4[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 42, 42, 512)	0	activation_4[0][0]
separable_conv2d_5 (SeparableConvolution)	(None, 42, 42, 512)	267264	max_pooling2d_4 [0][0]
batch_normalization_5 (BatchNormalization)	(None, 42, 42, 512)	2048	separable_conv2d_5 [0][0]
activation_5 (Activation)	(None, 42, 42, 512)	0	batch_normalization _5[0][0]
up_sampling2d_1 (UpSampling2D)	(None, 84, 84, 512)	0	activation_5[0][0]
separable_conv2d_6 (SeparableConvolution)	(None, 84, 84, 256)	144128	up_sampling2d_1 [0][0]
batch_normalization_6 (BatchNormalization)	(None, 84, 84, 256)	1024	separable_conv2d_6 [0][0]
activation_6 (Activation)	(None, 84, 84, 256)	0	batch_normalization _6[0][0]

up_sampling2d_2 (UpSampling2D)	(None, 168, 168, 256)	0	activation_6[0][0]
separable_conv2d_7 (SeparableConvolution)	(None, 168, 168, 128)	45440	up_sampling2d_2 [0][0]
batch_normalization_7 (BatchNormalization)	(None, 168, 168, 128)	512	separable_conv2d_7 [0][0]
activation_7 (Activation)	(None, 168, 168, 128)	0	batch_normalization _7[0][0]
up_sampling2d_3 (UpSampling2D)	(None, 336, 336, 128)	0	activation_7[0][0]
separable_conv2d_8 (SeparableConvolution)	(None, 336, 336, 64)	14528	up_sampling2d_3 [0][0]
batch_normalization_8 (BatchNormalization)	(None, 336, 336, 64)	256	separable_conv2d_8 [0][0]
activation_8 (Activation)	(None, 336, 336, 64)	0	batch_normalization _8[0][0]
up_sampling2d_4 (UpSampling2D)	(None, 672, 672, 64)	0	activation_8[0][0]
separable_conv2d_9 (SeparableConvolution)	(None, 672, 672, 3)	771	up_sampling2d_4 [0][0]