A THEORETICAL COMPARISON OF RESNET AND DENSENET ARCHITECTURES ON THE SUBJECT OF SHORELINE EXTRACTION

MERT ILHAN ECEVIT
B.S., Management Information Systems, IŞIK UNIVERSITY, 2016

Submitted to the Graduate School of Science and Engineering
in partial fulfillment of the requirements for the degree of
Master of Science
in
Information Technologies

IŞIK UNIVERSITY
2020

IŞIK UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

A THEORETICAL COMPARISON OF RESNET AND DENSENET ARCHITECTURES ON THE SUBJECT OF SHORELINE EXTRACTION

MERT İLHAN ECEVİT

APPROVED BY:

APPROVAL DATE: ..../..../....

A THEORETICAL COMPARISON OF RESNET AND DENSENET ARCHITECTURES ON THE SUBJECT OF SHORELINE EXTRACTION

Abstract

Today's Deep Learning technologies provides numerous approaches on the subject of convolutional networks. These approaches serve researchers to train datasets and generate wanted results from these datasets. Each CNN architecture has its own strong points and weak sides. Because of this situation a comparison between these architectures is a valuable asset. Image processing is a method that is frequently used to process remotely sensed data in remote sensing studies.. Between current architectures, RESNET and DENSENET architectures are chosen to be used by Dr. Çavdaroğlu for her project on TÜBİTAK. The result of this comparison will be used in that project in order to apply most efficient architecture.

This thesis is written to draw outlines of RESNET and DENSENET and create a foresight for further projects which can be supported by this thesis. In order to achieve an accurate image recognition process in remote sensing domain, a preliminary research is requisite. As a research thesis this work serves the purpose of learning manner of works, performance indicators of RESNET and DENSENET convolutional networks. The result of this research will create a baseline for an academical project. At the other hand, comparison of these two convolutional network approaches provides information to decide which approach is more suitable for remote sensing projects depending upon the subject of the project. For future works on Remote Sensing this thesis work will serve a guideline and reason for preference.

The presented thesis work has been developed as the technical feasibility of the 3501 TÜBITAK Project named "Uydu Görüntülerinden Kıyı Sınırlarının Derin Öğrenme Yöntemleri ile Otomatik Çıkarımı", applied by Dr. G. Çiğdem Çavdaroğlu, and the thesis results will be applied within the scope of the project after the project acceptance.

Keywords: Convolutional neural network, deep learning, training algorithms, resnet, densenet, cnn architectures, comparison, remote sensing, shoreline extraction, LANDSAT-8, SENTINEL 2-A

RESNET VE DENSENET MİMARİLERİNİN KIYI ŞERİDİ ÇIKARIMI KONUSUNDA TEORİK BİR KARŞILAŞTIRMASI

Özet

Günümüzün Derin Öğrenme teknolojileri, evrişimsel ağlar konusunda bir çok yaklaşım sunmaktadır. Sunulan bu yaklaşımlar, veri kümelerini eğitmek ve bu veri kümelerinden istenen sonuçları üretmek için araştırmacılara hizmet eder. Her CNN mimarisinin kendine özgü güçlü noktaları ve zayıf yanları vardır. Bu durum nedeniyle, bu mimariler arasındaki bir karşılaştırma değerli bir varlıktır. Görüntü işleme, uzaktan algılama çalışmalarında, uzaktan algılanmış verinin işlenmesi amacıyla yaygın olarak kullanılan bir yöntemdir. Mevcut mimariler arasında RESNET ve DENSENET mimarileri Dr. Gülsüm Çiğdem Çavdaroğlu tarafından TÜBİTAK üzerindeki projesi için kullanılmak üzere seçilmiştir. Karşılaştırmanın sonucu o projede en verimli mimariyi uygulamak için kullanılacaktır.

Bu tez, RESNET ve DENSENET'in ana hatlarını çizmek ve bu tez tarafından desteklenebilecek diğer projeler için bir öngörü oluşturmak için yazılmıştır. Uzaktan algılama alanında doğru bir görüntü tanıma süreci elde etmek için bir ön araştırma gereklidir. Araştırma tezi olarak bu çalışma, RESNET ve DENSENET evrişim ağlarının performans göstergelerini, çalışma biçimini öğrenme amacına hizmet eder. Araştırmanın sonucu akademik bir proje için bir temel oluşturacaktır. Diğer yandan, bu evrişimsel ağ yaklaşımlarının karşılaştırılması, projenin konusuna bağlı olarak hangi yaklaşımın uzaktan algılama projeleri için daha uygun olduğuna karar vermek için bilgi sağlar. Uzaktan Algılama üzerine gelecekteki çalışmalar için bu tez çalışması bir rehberlik ve tercih sebebi sağlayacaktır.

Sunulan tez çalışması, Dr. Gülsüm Çiğdem Çavdaroğlu tarafından başvurulan "Uydu Görüntülerinden Kıyı Sınırlarının Derin Öğrenme Yöntemleri ile Otomatik Çıkarımı" isimli 3501 Tübitak Projesi'nin teknik fizibilitesi olarak geliştirilmiştir ve tez sonuçları proje kabulü sonrasında proje kapsamında uygulanacaktır.

Anahtar kelimeler: Evrişimsel ağlar, derin öğrenme, algoritma eğitimi, resnet, densenet, cnn mimarileri, karşılaştırma, uzaktan algılama, kıyı çizgisi çıkarımı, LANDSAT-8, SENTINEL 2-A

Acknowledgements

Thanks to my advisor Dr. Gülsüm Çiğdem ÇAVDAROĞLU for her consistent interest and her advices in the process of conduction of this work, although she was really busy. I also would like to thank my teachers Assoc. Prof. M. Taner ESKİL for his patience and guidance, and Dr. Zeynep TURGUT for her review and assessment.

This study is a technical feasibility for a project applied to The Scientific and Technological Research Council of Turkey (TÜBİTAK).

Project Name: "Uydu Görüntülerinden Kıyı Sınırlarının Derin Öğrenme Yöntemleri ile Otomatik Çıkarımı"

To my dearest instructor Assoc. Prof. Gülay ÜNEL, a wonderful and gentle soul I will always remember ...

Most hearty thanks to my beloved wife for her devotion in this step of my life ...

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Remote Sensing

The term remote sensing was first utilized within the 1960s to clarify the actual observations of the globe from a far and will be defined because the “acquisition of knowledge about the state and condition of an object through sensors that do not seem to be in physical contact with it” [1]. As a way of collecting data about areas or objects from distances via images from satellites, airplanes or any flying craft that’s capable of taking images of the subject, Remote Sensing is a science [1]. Because of various propagated signals, remote sensing makes gathering essential information from areas that’s dangerous or inaccessible possible. Therefore this science includes a major importance in many areas including military information collection, land-use classification and monitoring [2–5]. Number of high-resolution image data to be used for remote sensing is increasing greatly due to the technological advancements in sensors. [6].

Sensors can be named as passive or active. Passive sensors react to external sources and consumes natural energy that’s reflected or the radiation the Earth’s surface. However, active sensors have their own source for illumination and measuring reflected energy. Electromagnetic detection are often done by sensors, where the sun is that the most source of energy to earth. There is also energy radiated from Earth surface yet but it’ll be explained later. Earth comprises of soil, water, rocks, vegetation, climatic element and also human structures which also emit a part of that energy back to the satellite sensor. Instruments utilized in sensor can measure and record the energy leaving the surface and are transmitted back to the ground receiving station through digital telemetry. Measured raw digital data is being collected by the sensor and converted into refined data [2].

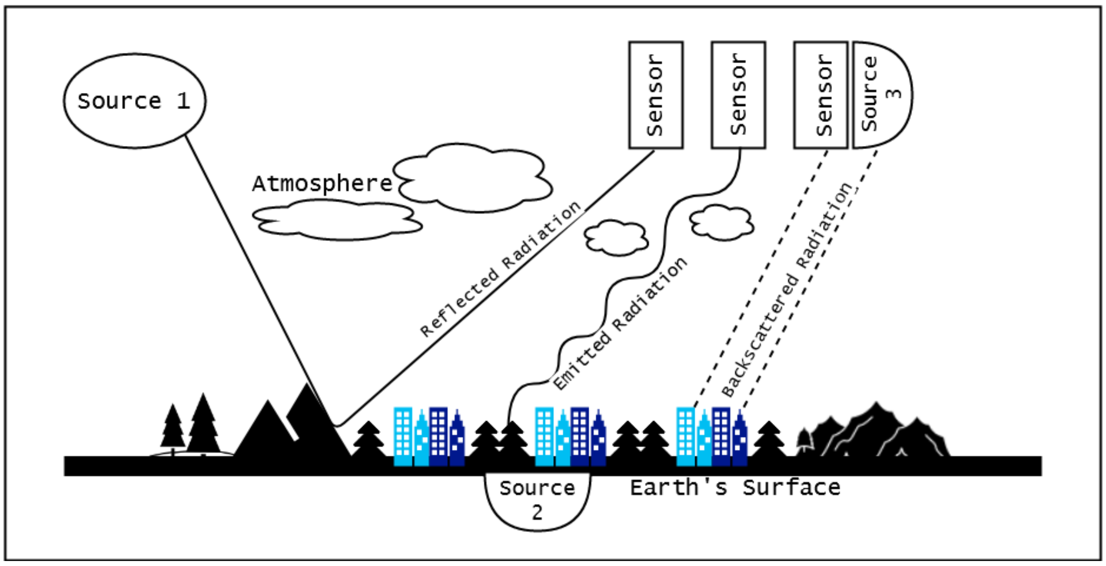


Figure 1.1: Energy transmissions back from earth to the satellite sensors [1].

Satellites also measure electromagnetic waves radiated from the earth, beyond reflected energy. (1.2) Much of this can be thermal radiation. Repeated waves are very short, say 1 to 2 micrometers (microns), but thermal waves are longer. Airplanes examplicate a way smaller band than satellites, but some type of films can record invisible infrared radiation [1]. This gives an opportunity of detecting infrared radiation from the surface with more accuracy.

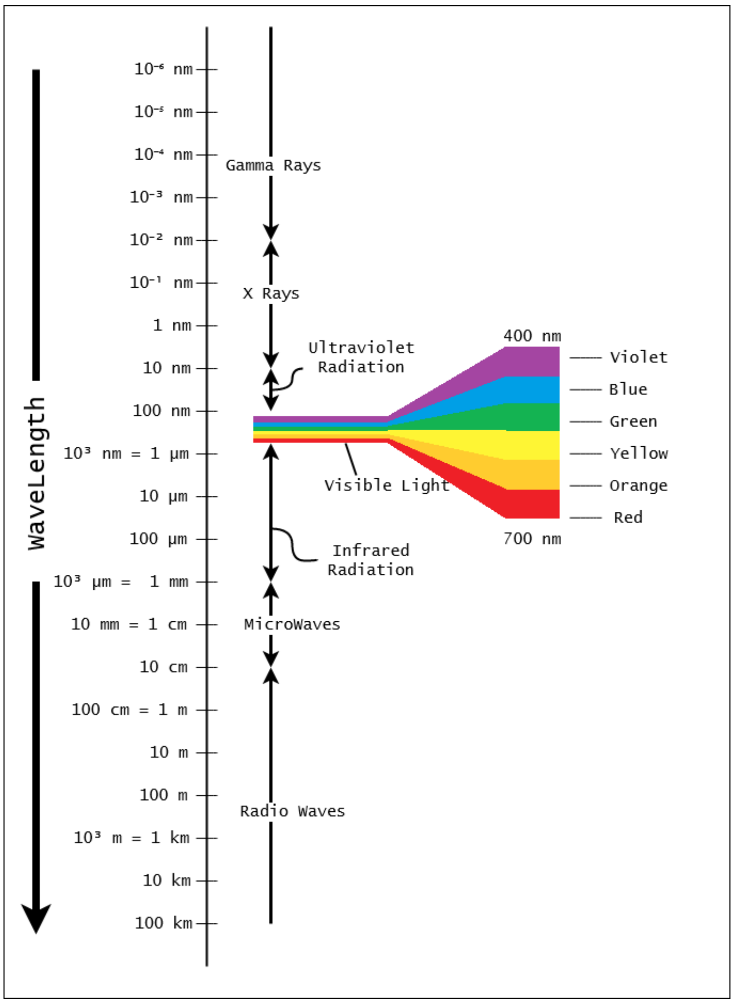


Figure 1.2: The electromagnetic spectrum [1].

1.2 Literature Domain

As technological domain this work and its supporting TÜBİTAK project these headlines are the most suitable:

1. Remote Sensing
    - Satellite Image Analysis
    - Shoreline Analysis

2. Image Processing
    - Processing of Large Sized Images
    - Object Recognition
    - Target Capture
3. Artificial Intelligence
    - Data Extraction
    - Machine Learning
    - Deep Learning
    - Feature Extraction
    - Transfer Learning
    - Data Quantity

These domain titles are in the area of interest of this work as well as in its scope. For each item, an explanation and definition can be found in individual titles. Short definitions can be found below:

Remote Sensing: Acquisition of knowledge about the state and condition of an object through sensors that do not seem to be in physical contact with it [1].

Satellite Image Analysis: Analyzing of images of the area of interest taken by an orbiting satellite from space. These analyses can consist of frequency band, object in image or geographical events.

Shoreline Analysis: Under the title of Remote Sensing, Shoreline Analysis can be explained as measuring shoreline quantitites, such as length, width or content, and providing a valuable information, caption about the interested shoreline.

Image Processing: This is a method for getting an enhanced image or extracting useful information from an image data. Input is an image data, output can be another image or features associated with the image.

Object Recognition: As a general term Object Recognition is a collection of related computer vision processes that involve identifying, naming or sensing object in digital images.

Artificial Intelligence: AI for short, is a sub title to computer science working on the simulation of intelligent, human-like, computer behaviors.

Data Extraction: A process of retrieving meaningful information from data sources.

Machine Learning: Constructing computer programs which the ability to automatically improve with experience.

Deep Learning: Deep Learning aims to give AIs aspects of human brain at processing data and decision making by creating patterns. As a function Deep learning can be taken as a subset of Machine learning.

Feature Extraction: This process' objective is to reduce the quantity of features in a dataset. It can be done by creating new features from existing ones.

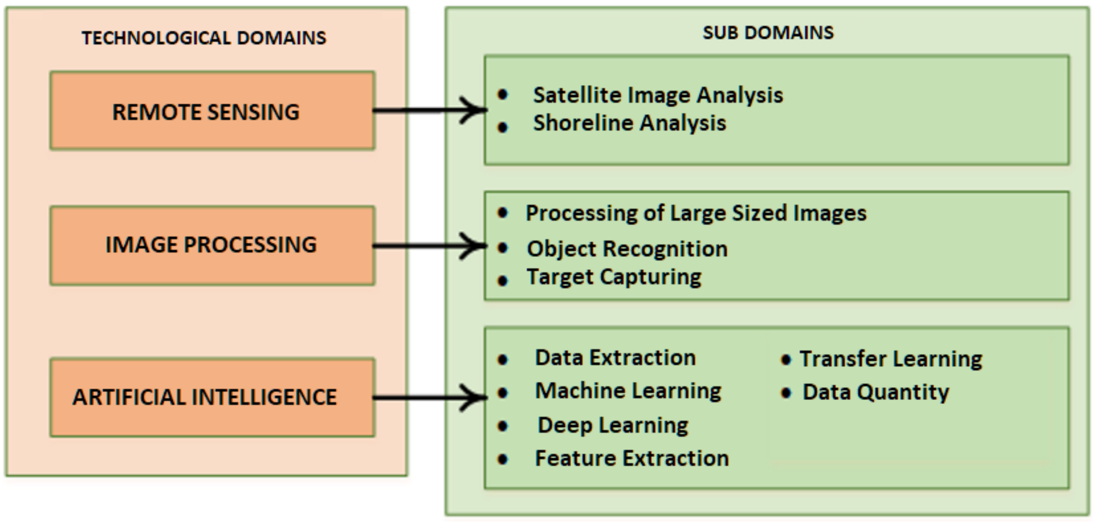


Figure 1.3: Technological Domain chart of this project.

Chapter 2

General Information About Shoreline Extraction.

2.1 Definitions on Shoreline

Shoreline is defined by Boak et al. as a physical line that divides land and water surfaces [7]. Fırat gives definitions of shoreline in Turkish Constitution as follows:

According to the coastal law numbered 3621, which was published in Turkey on 17.04.1990, and also the coastal law numbered 3830 regarding amendments in some articles of this law, the definitions made about the coastal line are as follows: (2.1)

Coastal Line: The line consisting of the points where the water touches the land in sea, natural and artificial lakes and streams except for flood situations.

Coastal Border Line: The natural boundary of sandy, gravel, rocky, stony, reed, marshy and similar areas within the sea, natural and artificial lakes and streams, where water movements are created in the direction of the land after the shoreline.

Shore: The land area at the edge of the water.

Shoreline: The area with a width of a minimum of 100 meters horizontally within the direction of land starting from the coastal line

Narrow Shore: Refers to the coastal line coinciding with the shoreline [8].

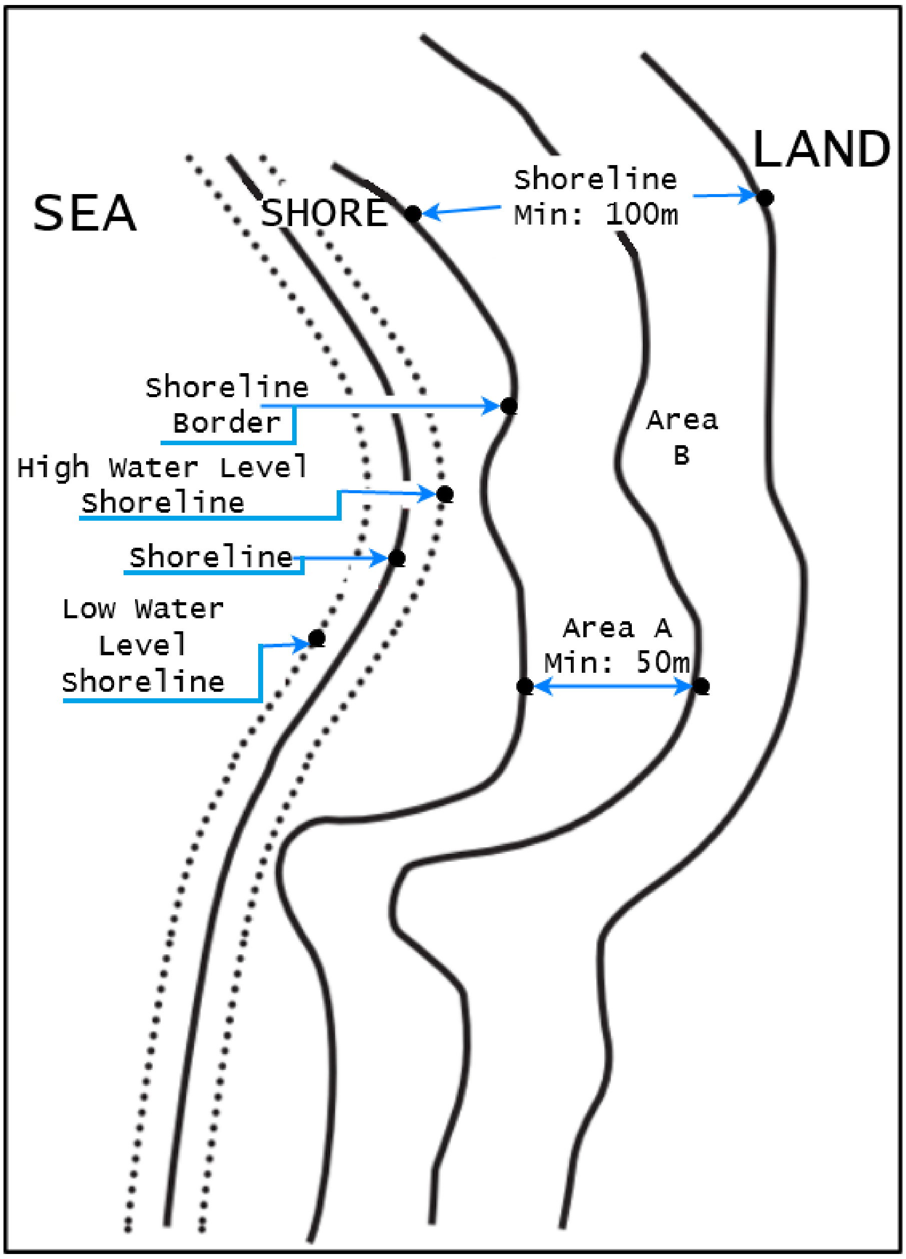


Figure 2.1: Definitions in the Coastal Law (Translated).

2.2 The Importance of Shoreline Detection

Determination of the coastal line and creating maps of coastal areas has critical importance for safe navigation, management of coastal resources, protection of the environment in coastal areas and healthy coastal planning. Changes occurring along the shoreline may affect the order of the coastal zone. Natural causes or humanity can cause these changes [9].

Coastal areas are one amongst the foremost threatened ecosystems. The rise of pollution from population growth, global warming, erosion from urbanization and extreme weather events make management of coastal areas important [10]. Sale et al. within the study conducted, predicted that in the of 91 percent of the coastal areas within the world are going to be adversely affected because of irregular planning by 2050 [11]. (2.2)

One of the foremost obvious consequences of global warming is that the water level rise. Flooding from rising water level will greatly affect low altitude areas. About 100 million people live within 1 meter of the average water level, and these people are going to be more at risk within the coming years. Some island countries and coasts that form the delta are threatened by rising water level [12].

In addition to tourism activities and aesthetic values, coastal areas play a very important role in promoting economically important fishing activities and in the cycle of basic life resources like nitrogen [13].

Coastal areas are presented between 27 most vital natural riches of earth by The International Geographic Data Committee (IGDC) [14].

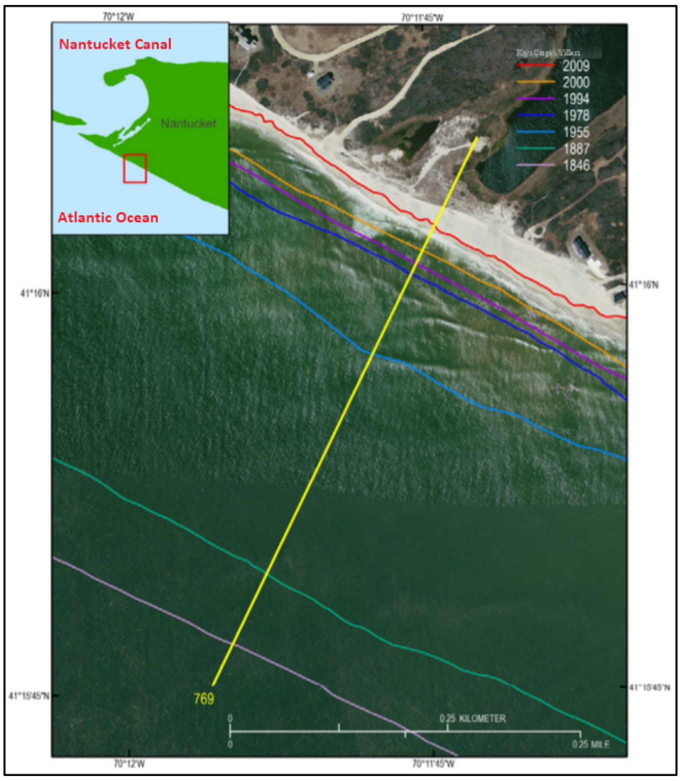


Figure 2.2: Change of shoreline by years [8].

2.3 Remote Sensing and Shoreline Extraction

The function of obtaining information about the object or area by analyzing the information obtained by a device 2.3 that's not in the area of contact with the object or area that's the subject of research is named as Remote Sensing [15].

Object extraction in remote sensing will be defined as image processing techniques accustomed to define and classify interrelationships between image regions [16]. By applying various image processing techniques to satellite images, it's possible to separate the objects of interest from other objects automatically [17].

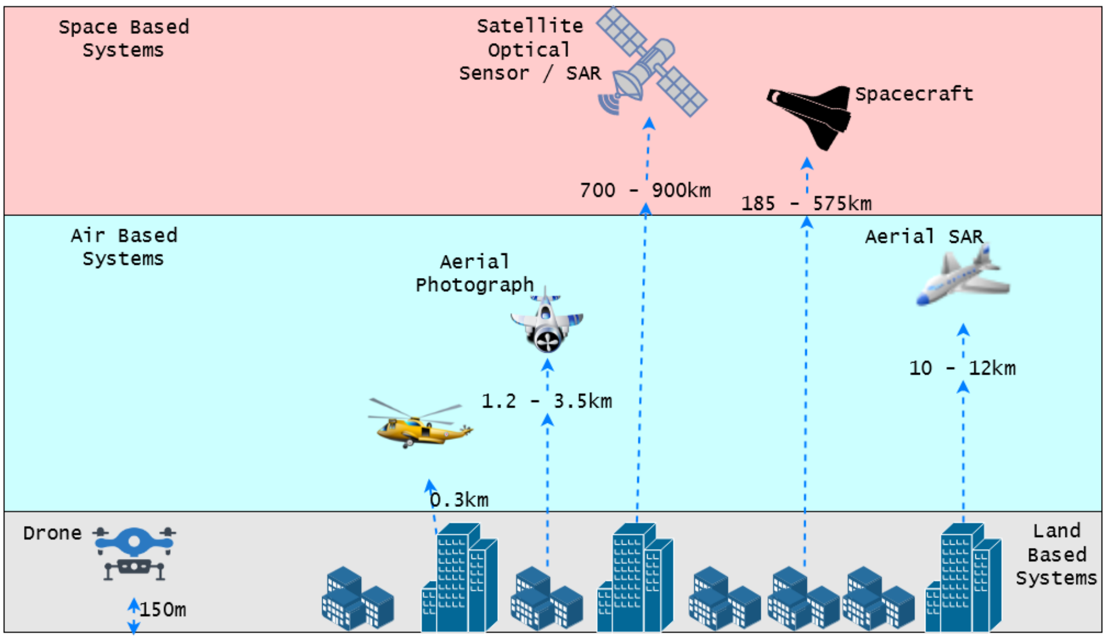


Figure 2.3: Platforms and sensors used in remote sensing.

Terrestrial measurement, photogrammetric methods, and remote sensing data can be utilized in studies for coastal area monitoring and coastal line extraction [18].

Tracking coastal lines using satellite images makes it possible to get instant and temporal data that cannot be obtained by terrestrial measurements. Current, accurate, temporal and reliable information about coastal areas may be obtained by using optical satellite images [19].

2.4 Shoreline Satellite Image Data Specifications

Shoreline Satellite Images are taken by satellites which are orbiting the globe. These images have different frequencies.

Label data that expresses the water and land classes of the training image which will be used to train the deep learning network is required. NDWI images of high resolution images obtained for the creation of label data were created. NDWI is a band proportioning method used to determine water properties using remote sensing images. NDWI images are calculated by the equation [8].

$$NDWI = (NIR - RED)/(NIR + RED) \tag{2.1}$$

NDWI values range from -1 to +1. Green areas tend to possess positive values, soil areas near zero, and water properties tend to possess negative NDWI values [8].

These types of Images have:

- High File Size

- Great count of pixels

- High Resolution

In the TÜBİTAK project of ”Uydu Görüntülerinden Kıyı Sınırlarının Derin Öğrenme Yöntemleri ile Otomatik Çıkarımı”, LANDSAT-8 and SENTINEL 2-A satellite images are determined to be used.

2.4.1 LANDSAT-8

LANDSAT-8 can produce output in 11 different bands with its two main sensors. The Operational Land Image(OLI) has 9 spectral bands (1 to 9) with 15, 30, 60 meter resolutions. Thermal Infrared Sensor(TIRS) includes 2 thermal bands (10 and 11) and provides 100 meter resolution each. The most popular band combinations can be seen at figures below(2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 2.10, 2.11, 2.12).

Bands	**Features**	**Spectral Witdh (µm)**	**Resolution (m/x)**
Band 1	Coastal / Aerosol	435-452	30
Band 2	Visible blue	452-512	30
Band 3	Visible green	533-590	30
Band 4	Visible red	636-673	30
Band 5	Near-infrared	851-879	30
Band 6	Short wavelength infrared	1560 - 1660	30
Band 7	Short wavelength infrared	2107-2294	60
Band 8	Panchromatic	503-676	15
Band 9	Cirrus	1363-1384	30
Band 10	Long wavelength infrared	10300-11300	100
Band 11	Long wavelength infrared	11500-12500	100

Table 2.1: Band variations of LANDSAT-8 images (m/x: meter / per pixel).

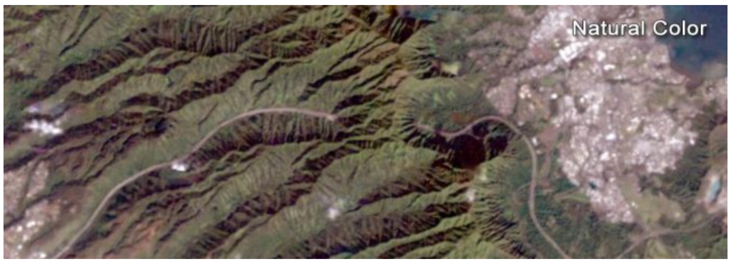


Figure 2.4: Natural Color (4,3,2) [20].

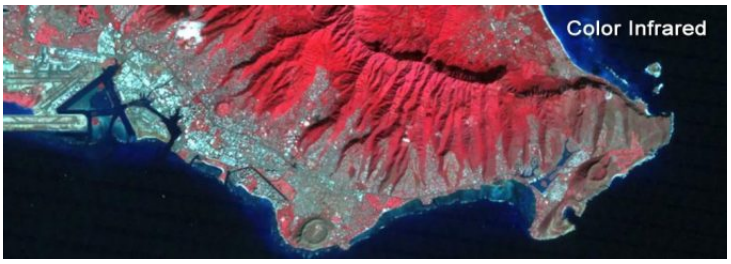


Figure 2.5: Color Infrared (5, 4, 3) [20].

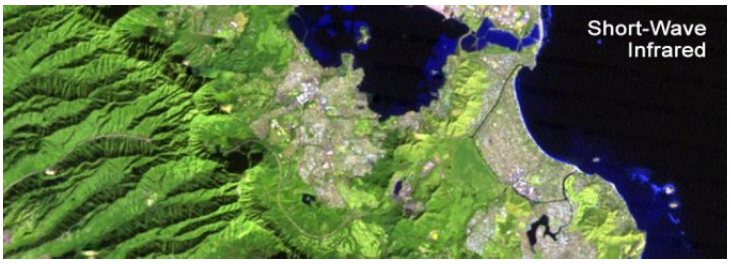


Figure 2.6: Short-Wave Infrared (7, 6, 4) [20].

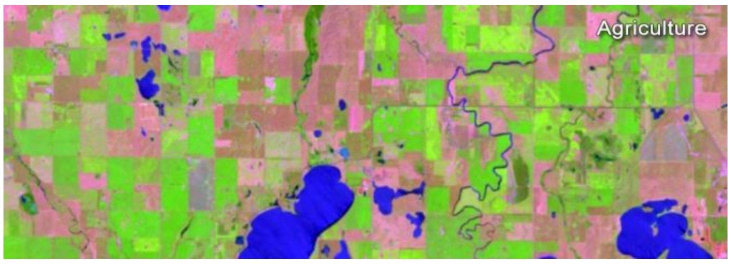


Figure 2.7: Agriculture (6, 5, 2) [20].

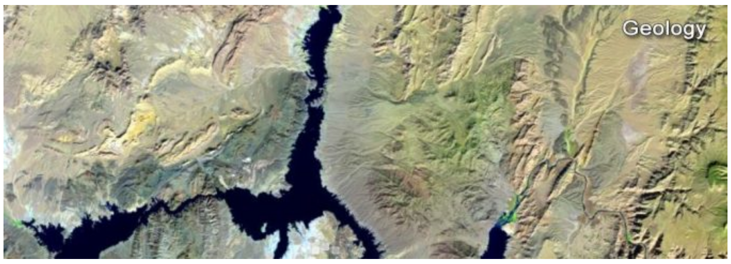


Figure 2.8: Geology (7, 6, 2) [20].

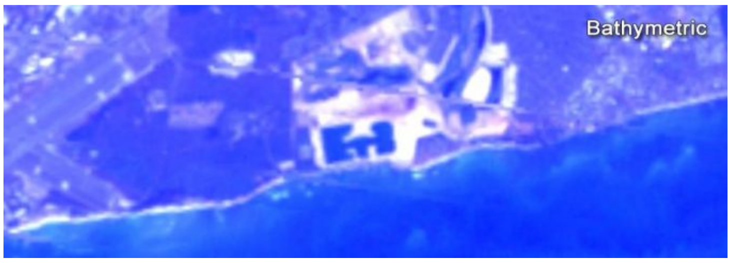


Figure 2.9: Bathymetric (4, 3, 1) [20].

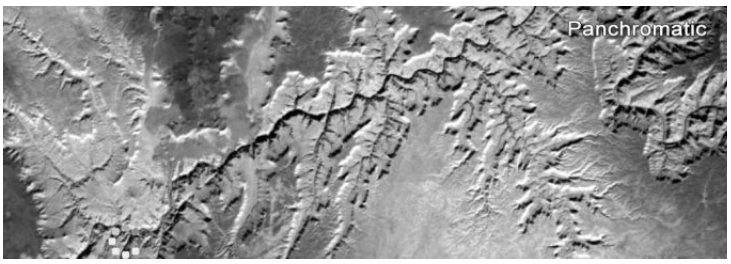


Figure 2.10: Panchromatic [20].

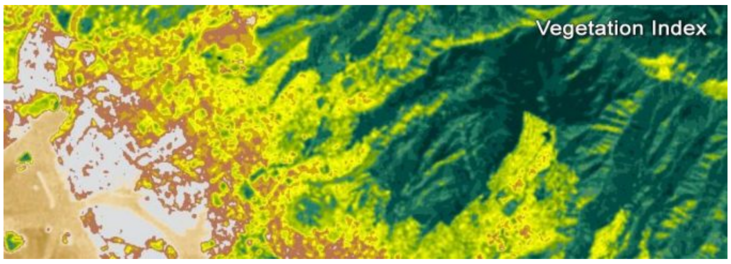


Figure 2.11: Vegetation Index [20].

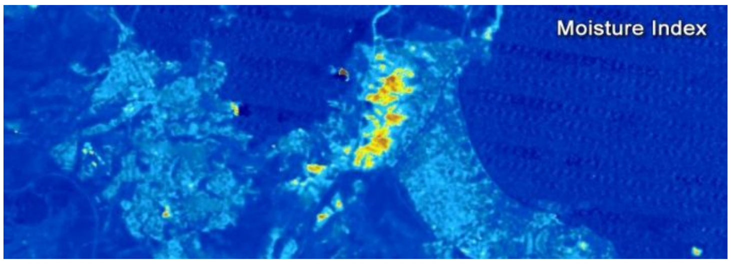


Figure 2.12: Moisture Index [20].

Also images of Istanbul that are generated in raster, blue-red-NIR and binary forms from LANDSAT-8 are added below(2.13, 2.14, 2.15).

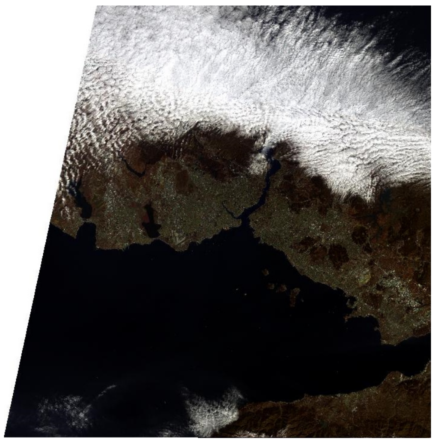


Figure 2.13: Raster Satellite Image of Istanbul.

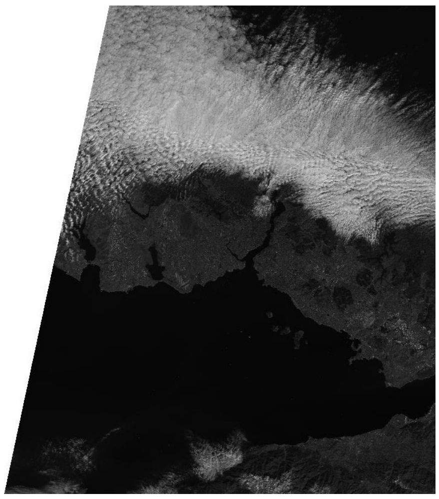


Figure 2.14: Satellite Image of Istanbul with blue, red and NIR bands.

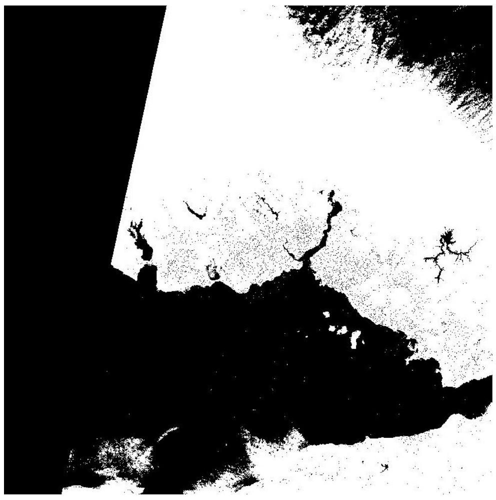


Figure 2.15: Binary Image. Satellite Image of Istanbul.

2.4.2 SENTINEL 2-A

Sentinel 2-A contains the Multispectral Imager (MSI) which can output 13 bands in total with 10,20 or 60 meters resolution in each pixel. At below common combinations of Sentinel 2-A band combinations are added(2.16, 2.17, 2.18, 2.19, 2.20, 2.21, 2.22).

Band	Resolution	Central Wavelength	Description
B1	60m	443nm	Ultra blue (Coastal and Aerosol)
B2	10m	490nm	Blue
B3	10m	560nm	Green
B4	10m	665nm	Red
B5	20m	705nm	Visible and Near Infrared (VNIR)
B6	20m	740nm	Visible and Near Infrared (VNIR)
B7	20m	783nm	Visible and Near Infrared (VNIR)
B8	10m	842nm	Visible and Near Infrared (VNIR)
B8a	20m	865nm	Visible and Near Infrared (VNIR)
B9	60m	940nm	Short Wave Infrared (SWIR)
B10	60m	1375nm	Short Wave Infrared (SWIR)
B11	20m	1610nm	Short Wave Infrared (SWIR)
B12	20m	2190nm	Short Wave Infrared (SWIR)

Table 2.2: Band variations of SENTINEL 2-A images.

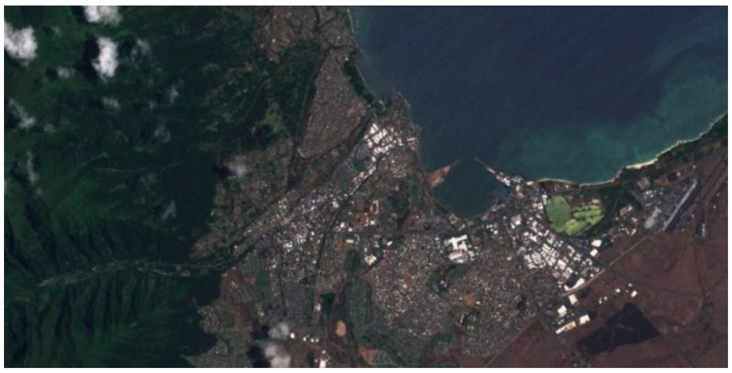


Figure 2.16: Natural Color (B4, B3, B2) [21].

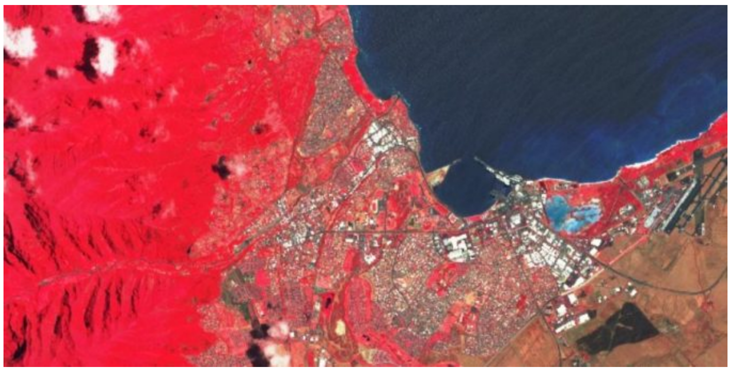


Figure 2.17: Color Infrared (B8, B4, B3) [21].

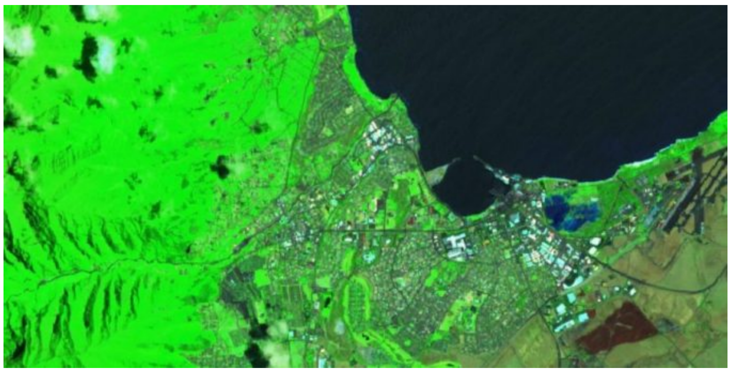


Figure 2.18: Short-Wave Infrared (B12, B8A, B4) [21].

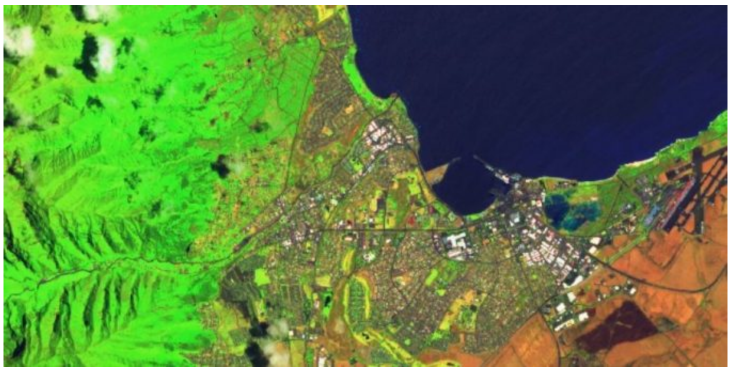


Figure 2.19: Agriculture (B11, B8, B2) [21].

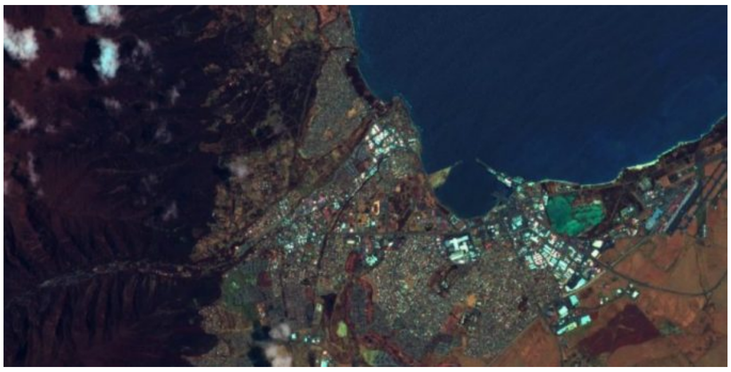


Figure 2.20: Geology (B12, B11, B2) [21].

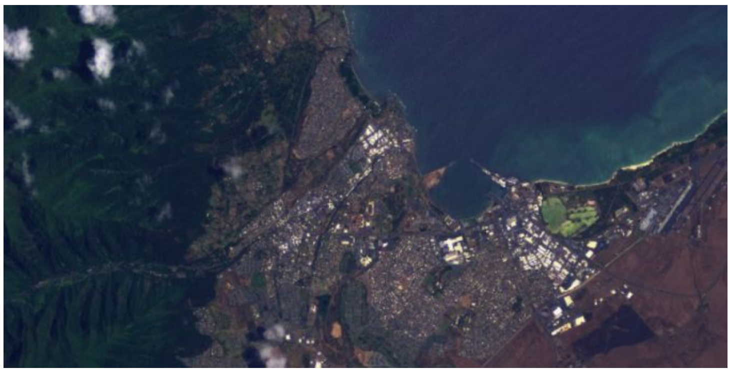


Figure 2.21: Bathymetric (B4, B3, B1) [21].

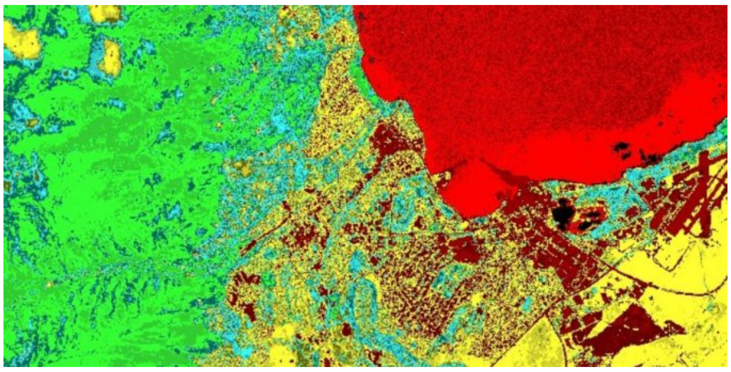


Figure 2.22: Vegetation Index (B8-B4)/(B8+B4) [21].

Chapter 3

Convolutional Neural Network

3.1 Convolutional Neural Network

With the technological advancement at sensing technologies, researchers have great numbers of data to process while engaged on their subject. These situations push AI near human like capabilities. Especially in latest years AI is cut off a monumental distance at the gap between human and machines. Researches are working on many areas with usage of AI. One amongst these fields is Computer Vision. Computer Vision field's main aim is to create possibility for machines to see world as human brain can do. Beyond that Computer Vision is able to do sensing that exceeds human capabilities. These advancements with Deep Learning created Convolutional Neural Network.

Terminologically, ConvNet/CNN short for Convolutional Neural Network is a Deep Learning algorithm which gets an image as input and determines the relevance of aspects or objects within the image.

Compared to rest of the classification algorithms, pre-processing function need in ConvNet is lesser. As the basic methods filters are developed by hand, ConvNet shows a promise to learn these filters if training is provided enough [7].

The main architecture belongs to Deep Learning is accepted as CNN. In CNN first several stages are Convolution and Pooling layers. In final stage there are Fully-Connected layer and Classification layer. After these numerous successive trainable layers, Deep Learning structure continues with a training layer. CNN gives an output to check with real/correct results. This comparison gives a mistake rate which is that the difference between generated output and targeted result. CNN can use various input file like image, sound video or other signals.

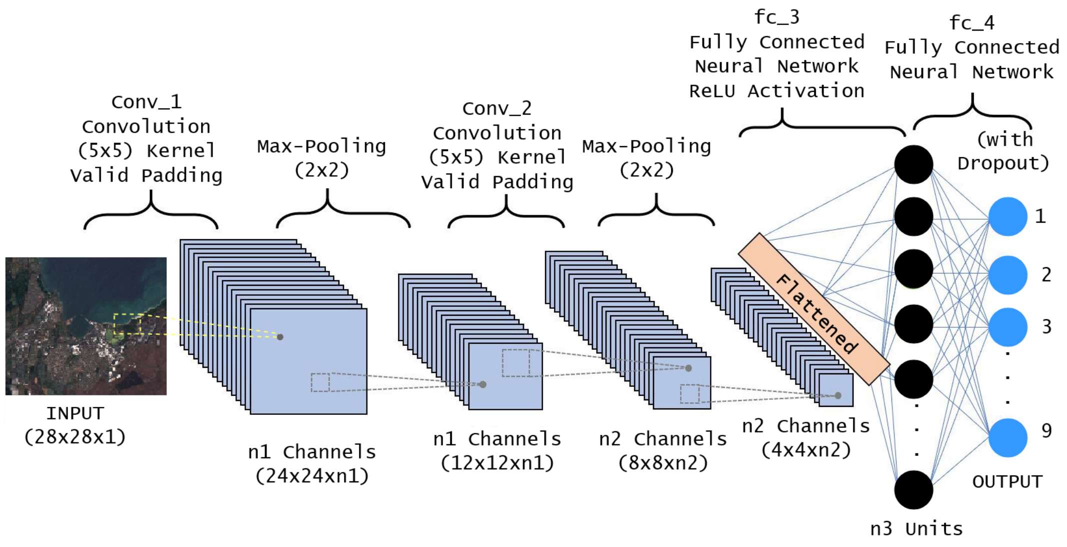


Figure 3.1: A CNN sequence to classify handwritten digits [7].

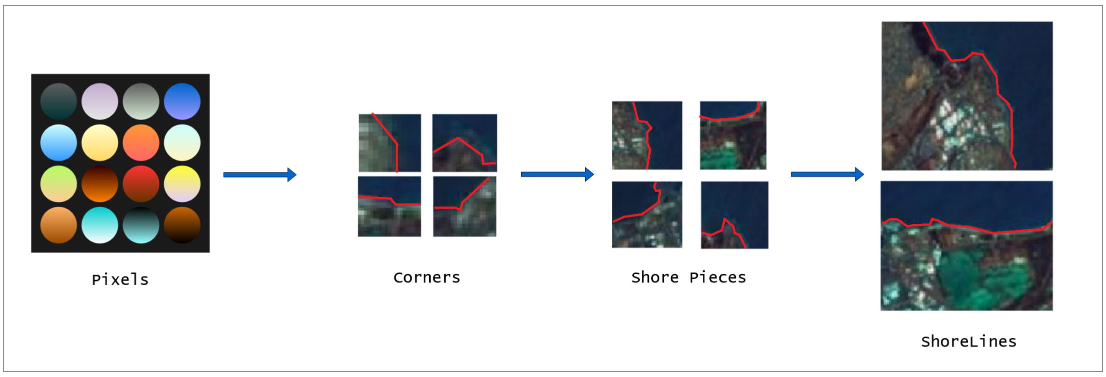


Figure 3.2: Different representations of object parts in different layers of CNN [22].

3.2 CNN Layers

3.2.1 Input Layer

From this layer data gets into CNN network. For accuracy of the designed model the scale of input data is very important. At the other hand size determines memory requirement and training time. If data size is chosen large it will create a high demand of memory and time respectively. When data size is tiny, training time could also be reduced but it will lower the deepness of the network and performance of it.

3.2.2 Convolution Layer

These filters can get different sizes as 2x2, 3x3, 1x3 – 3x1. Filters are applying convolution process to data from previous layer for producing an output. As process ends feature map is provided. While training with CNN, Filter coefficient is modified by each repetition. This manner network, for feature determination, can specify which data areas are important. For example; if input data is assumed as an RGB image with a 5x5 matrix. A 3x3 filter is circulated on input data. Process continues by going one digit down on the data as filter reaches matrix border. Filter coefficient multiplies with each color channel and result sums. This calculation gives the feature map. For every color channel filter coefficients are different and determined by analysts as suitable to their model design.

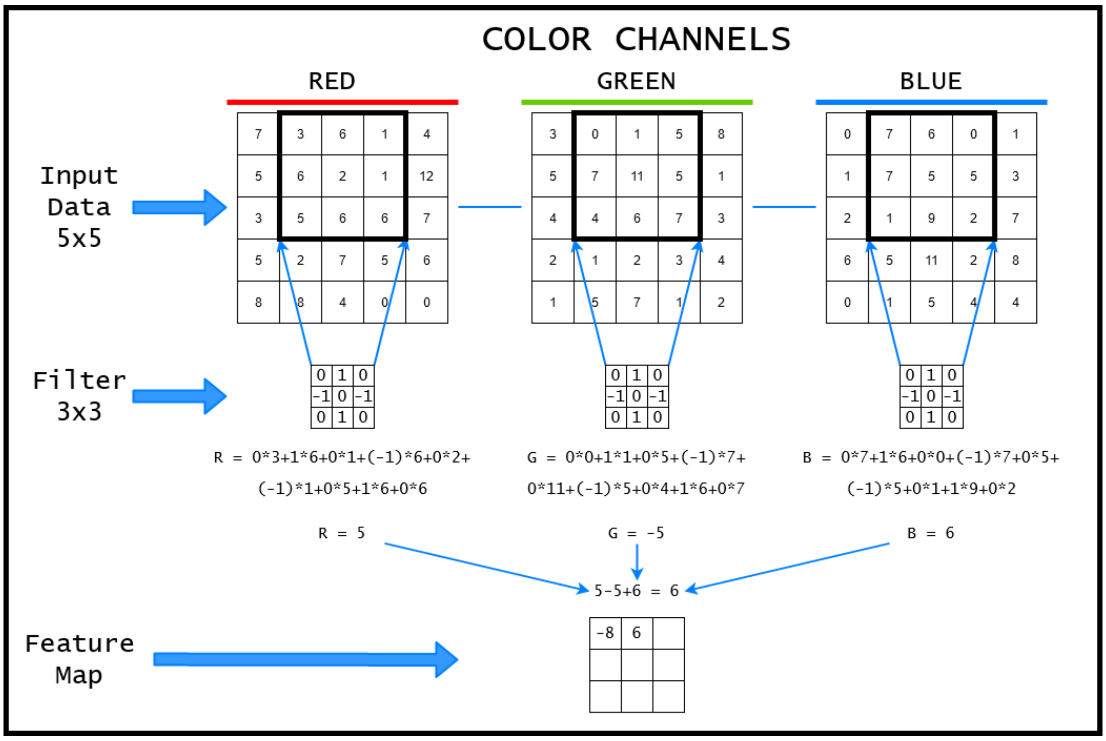

COLOR CHANNELS
RED
GREEN
BLUE
Input Data 5x5
Filter 3x3
Feature Map
R = 0*3+1*6+0*1+(-1)*6+0*2+(-1)*1+0*5+1*6+0*6
G = 0*0+1*0+5+(-1)*7+0*11+(-1)*5+0*4+1*6+0*7
B = 0*7+1*6+0*0+(-1)*7+0*5+(-1)*5+0*1+1*9+0*2
R = 5
G = -5
B = 6
5-5+6 = 6
-8
6

Figure 3.3: Convolution process [23].

3.2.3 Rectified Linear Units Layer (ReLu)

After Convolation Layer, ReLu is employed. ReLu is usually referred as a rectifier unit for the outputs of CNN Nodes. Its effect on input data to that is to alter negative values with 0. With usage of this layer network can train more swiftly.

3.2.4 Pooling Layer

Limited availability of feature map output from convolutional layers is due to these layers' saving function of the specific position of features within the input data. By this way with small changes at the position of the feature, a larger feature map can be achieved. This can be done by cropping, rotating, shift, or any minor manipulations to the input image.

Down Sampling is a general solution for this situation. When a lower resolution branch is formed from the input signal, it still consists of large structural elements which is not useful for the process. Down sampling is done by altering convolution stride among the image. Pooling layer is a more common and robust way for this task [24].

This process can be done by two ways, the utmost values across the pixels can be taken (maximum pooling) or the average of this values (average pooling). This layer causes loss of data because of reduced size. But the loss is beneficial for network due to two reasons. First, it creates a less calculation load and second it prevents system to memorize. Pooling is performed on each image by using the number of filters produced as a output of the convolution layer. In CNNs, pooling layer is optional and not utilized by some architectures [23].

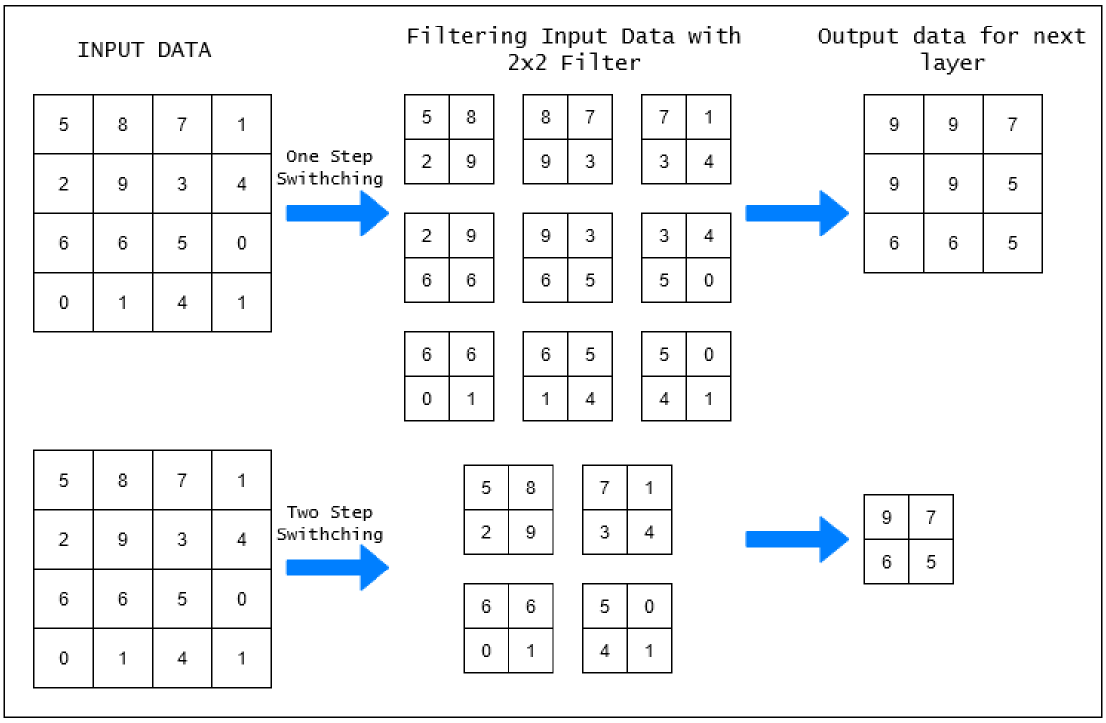


Figure 3.4: Maximum pooling with 2x2 Filter in 5x5 Input Data [23].

3.2.5 Fully Connected Layer

In order to utilize classification decision, the data which is broken down into features and analyzed independently is given into fully connected layer. It has three parts:

• **Fully connected input layer:** It turns data from previous layers into a single layer vector.

• **The primary fully connected layer:** Predicts correct label by applying weights to inputs from feature analysis.

• **Fully connected output layer:** Finalize probabilities for every label [25].

With this layer output of previous layers are being labeled and gets ready for classifications. As it is explained in(3.1).

Translation of a practical explanation from İnik et al. [23], This layer is bound to every aspect of the predecessor layer. The amount of this layer can change regarding the architecture. When the final layer's matrix size is selected as 25x25x256 = 160000x1 and also the size of the matrix in the fully connected layer is selected as 4096x1. A complete weight matrix of 160000x4096 is made. That is, each 160000 neurons are connected with 4096 neurons. As a result of this situation, this layer is named a fully connected layer [23].

To determine the foremost accurate weights, the fully connected part gives every neuron the most suitable labeling. Finally the comparison of max values are giving a classification decision.

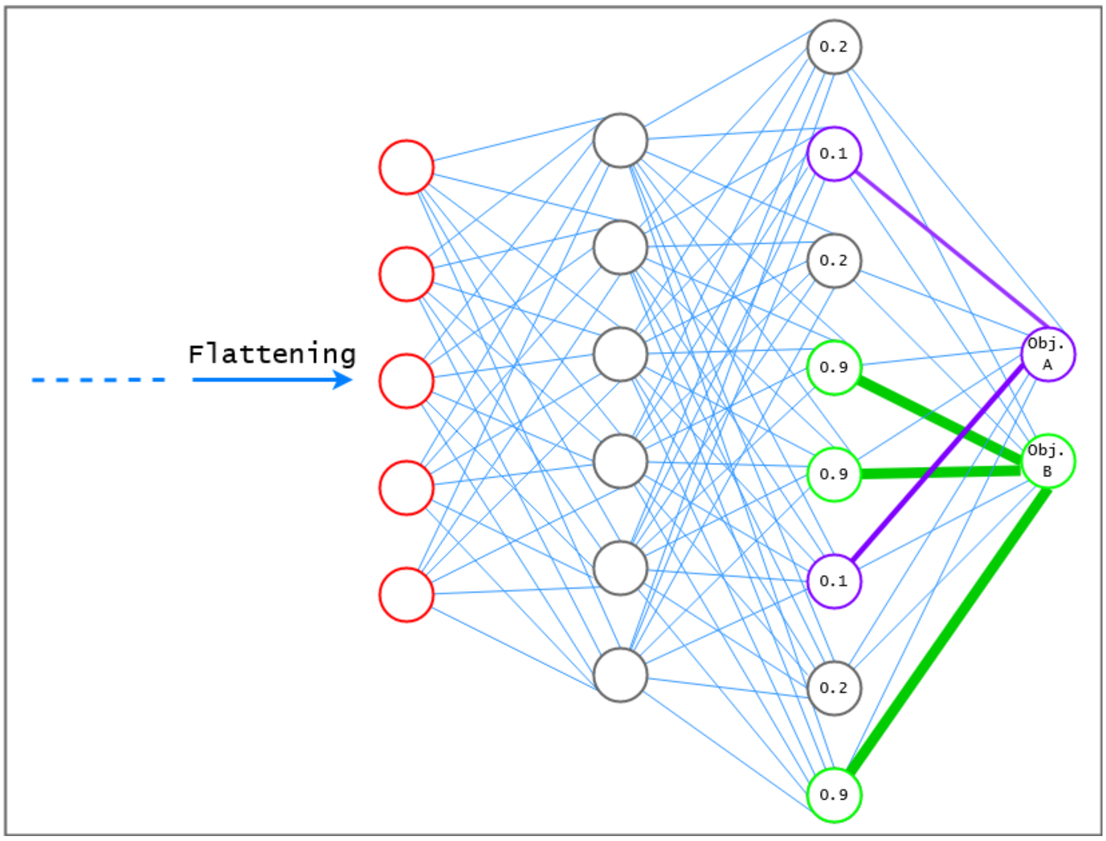


Figure 3.5: Basic illustration for fully connected layer's decision making [25].

3.2.6 Dropout Layer

This layer acts as a error reducing tool and not always used by analysts. Dropout layer gets input from fully connected layer and the algorithm sets values to a certain ratio (p = 0.5). Other values aren't set to 0 are connected to every other to effectively prevent the model from over-fitting. Dropout could be a powerful technique introduced in [26] for improving the generalization error of enormous neural networks [27].

Generally dropout is utilized on fully connected layers by deep learning models, but it can be done by utilizing dropout after maximum pooling layers to create a effective image noise augmentation [28].

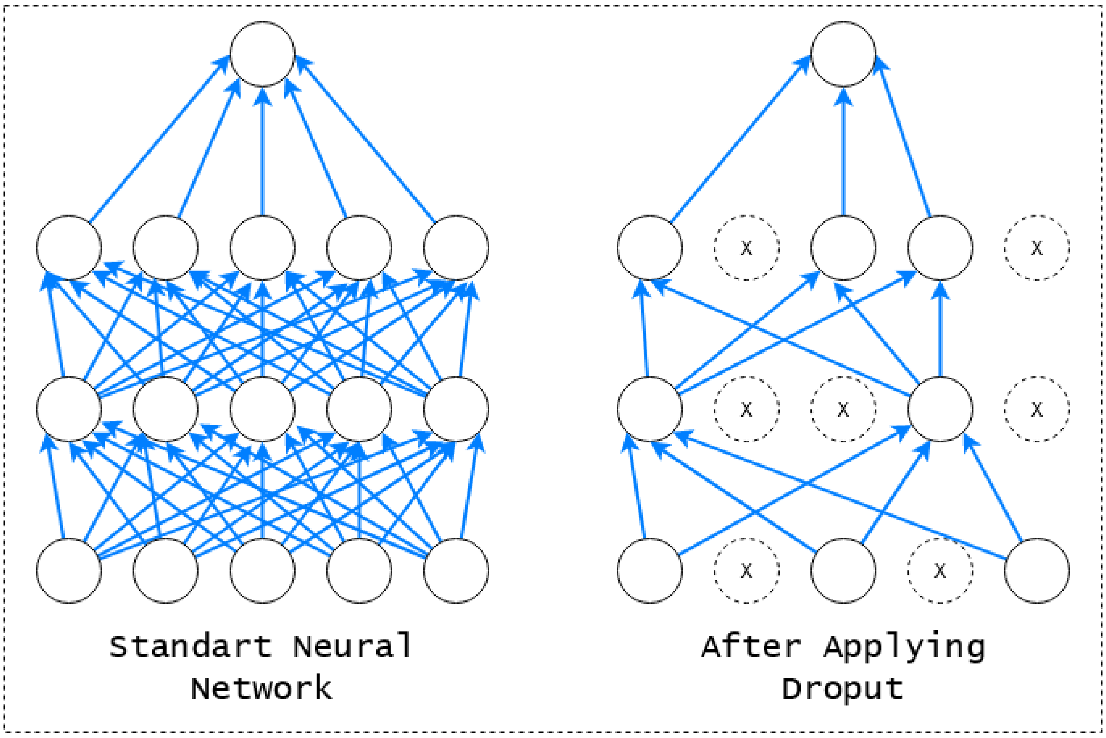


Figure 3.6: Left: A standard neural net with 2 hidden layers. Right: Application of dropout [28].

3.2.7 Classification Layer

Classification layer comes after fully connected layer. Classification process happens during this layer.

The quantity of objects for classification is equivalent to output value of this layer. As an example, if 15 different objects are going to be classified, the classification layer output value should be 15. If the output value is chosen as 4096 within the fully linked layer, a 4096x15 weight matrix is obtained for the classification layer according to this output value. Different classifiers are utilized in this layer. Because of its success rates Softmax classifier is chosen. In classification, 15 different objects produce a specific value in the range of 0-1. The output that produces a result near 1 is known to be the object the network predicted [23].

3.3 Architectural innovations in CNN

From 1989 to today different improvements for CNN architectures are made. These improvements had its main trust from processing unit restructure and producing blocks. There are seven class of CNNs can be seen on the figure 3.7.

Scheme of CNN architectures is represented in figure 3.7:

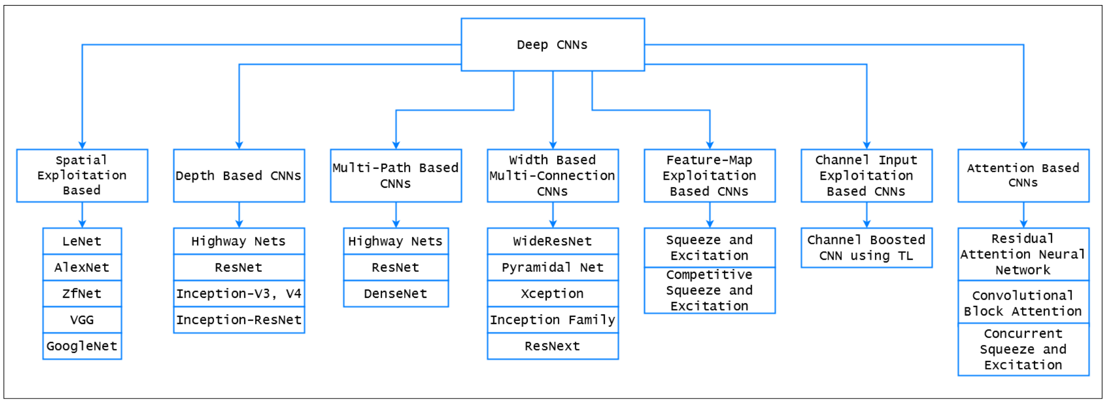

Deep CNNs
Spatial Exploitation Based
LeNet
AlexNet
ZfNet
VGG
GoogleNet
Depth Based CNNs
Highway Nets
ResNet
Inception-V3, V4
Inception-ResNet
Multi-Path Based CNNs
Highway Nets
ResNet
DenseNet
Width Based Multi-Connection CNNs
WideResNet
Pyramidal Net
Xception
Inception Family
ResNext
Feature-Map Exploitation Based CNNs
Squeeze and Excitation
Competitive Squeeze and Excitation
Channel Input Exploitation Based CNNs
Channel Boosted CNN using TL
Attention Based CNNs
Residual Attention Neural Network
Convolutional Block Attention
Concurrent Squeeze and Excitation

Figure 3.7: CNN architectures in seven different categories [29].

3.4 CNN Architectures - State-of-the-art

The most prominent architectures that are being used are the State of the art CNN architectures. Respectively Convolutional Layers, pooling layers and fully connected layers are utilized at the last stage of these architectures.

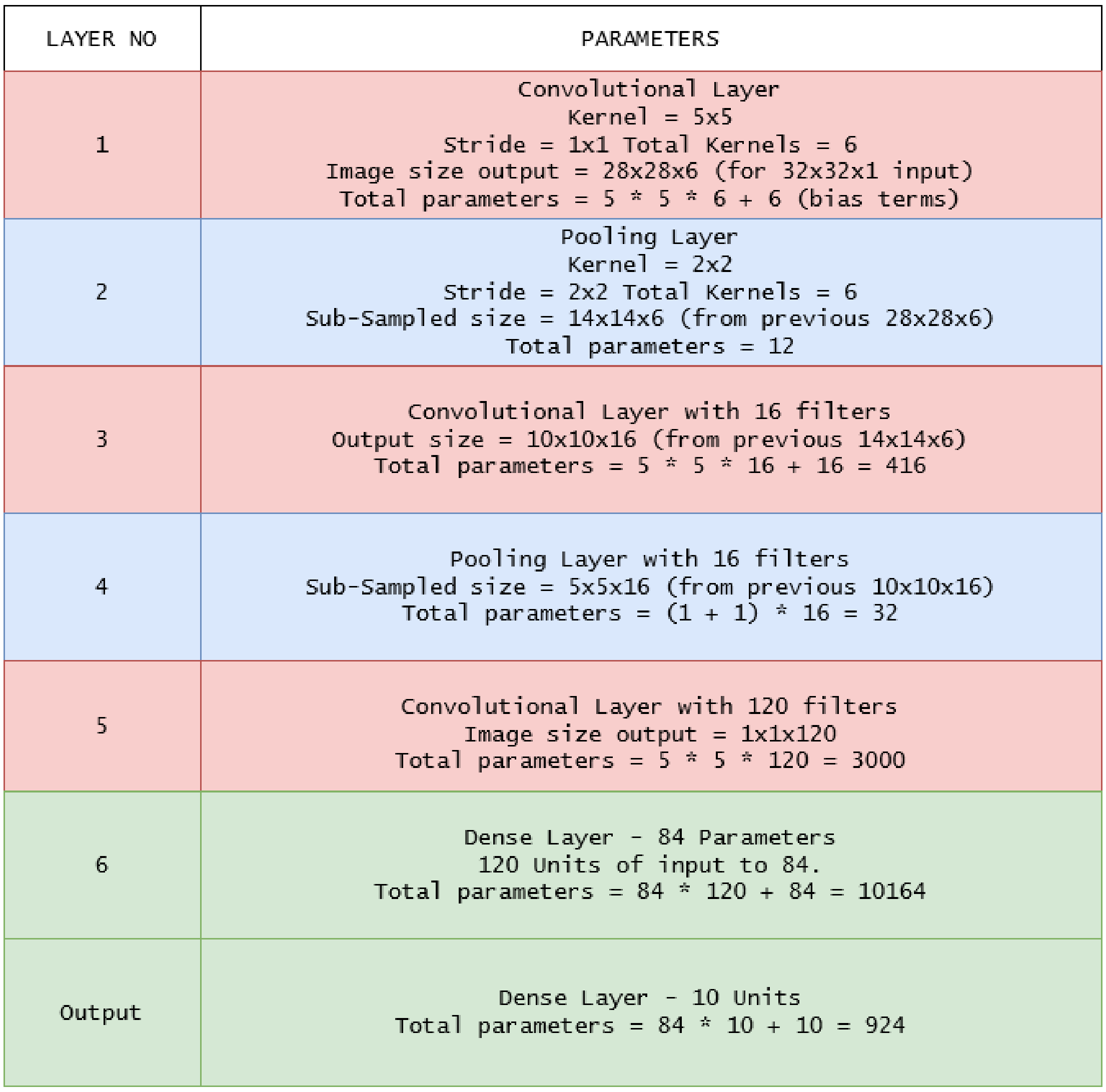
LeNet [30], AlexNet [31], VGG Net [32], NiN [33] and All Conv [34] are the examples to these architectures. There are other examples which can be shown as more efficient advanced alternatives to those architectures that are proposed including Residual Networks [35], DenseNet [36], GoogLeNet [37], Inception [35] and FractalNet [38].Components of convolution and pooling are nearly the identical among these architectures.

DCNN architectures, AlexNet, VGG, GoogLeNet, DenseNet and FractalNet, due to their performance on object recognition can be termed as the foremost popular architectures. Between those structures, GoogleNet and ResNet is specifically developed for larger data analysis scales, whereas the VGG network is taken into account as a common architecture. Part of these architectures are showing much more dense connectivity, like DenseNet. At the other hand Fractal Network can considered as an alternative to ResNet [39].

3.4.1 LeNet

LeNet was a difficult algorithm to implement until 2010 because of lesser computation power and memory scale, since it had been proposed within the 1990s [30]. To attain state-of-the-art accuracies LeCun, used back-propagations and tested on digit dataset generated by hand. LeNet-5 is his renowned architecture [30]. LeNet5 contains two convolution layers, two sub-sampling layers, two FC(Fully Connected) layers. An output layer is also included. Total number of weights are 431k and MACs count is 2.3M [39].

3.4.1.1 LeNet Architecture



LAYER NO	PARAMETERS
1	Convolutional Layer Kernel = 5x5 Stride = 1x1 Total Kernels = 6 Image size output = 28x28x6 (for 32x32x1 input) Total parameters = 5 * 5 * 6 + 6 (bias terms)
2	Pooling Layer Kernel = 2x2 Stride = 2x2 Total Kernels = 6 Sub-Sampled size = 14x14x6 (from previous 28x28x6) Total parameters = 12
3	Convolutional Layer with 16 filters Output size = 10x10x16 (from previous 14x14x6) Total parameters = 5 * 5 * 16 + 16 = 416
4	Pooling Layer with 16 filters Sub-Sampled size = 5x5x16 (from previous 10x10x16) Total parameters = (1 + 1) * 16 = 32
5	Convolutional Layer with 120 filters Image size output = 1x1x120 Total parameters = 5 * 5 * 120 = 3000
6	Dense Layer - 84 Parameters 120 Units of input to 84. Total parameters = 84 * 120 + 84 = 10164
Output	Dense Layer - 10 Units Total parameters = 84 * 10 + 10 = 924

Figure 3.8: LeNet Architecture Table [39].

This approach can not be scaled to larger images. Expect input layer there are 7 layers in this model [39]. Showing it layer by layer is easy because of its small architecture5.1.

3.4.2 AlexNet

In 2012 Alex Krizhevesky won the foremost difficult challenge for ILSVRC, short for ImageNet Large Scale Recognition Challenge, by proposing a higher scale CNN model compared to LeNet [30].

Accuracy rate achieved by AlexNet is the highest rate among all of the traditional functions. It absolutely created big leap forward for the domain of computer vision for image recognition and classification processes and it created a rapidly increasing interest in deep learning [39].

The study was published with the article "ImageNet Classification with Deep Convolutional Networks" [19] and 16227 quotations were made as of October 2017. With this architecture, computerized object identification error rate has been reduced from 26.2 percent to 15.4 percent. Figure (3.9) shows configuration of the architecture. The architecture is intended to classify 1000 objects.

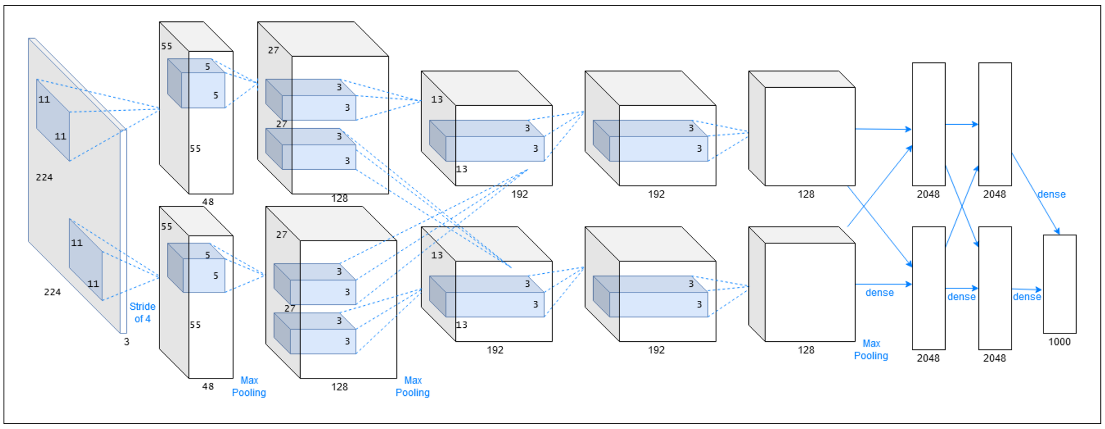


Figure 3.9: Illustration of Alex Net's architecture [23].

3.4.2.1 AlexNet Architecture

To train on two different GPUs at the same time, AlexNet is divided into two with 3 convolution layers and a pair of fully connected layers, as shown in Figure (3.9). Total number of parameters of AlexNet while processing the ImageNet dataset at primary layer: input sample number is 224x224x3, filters are adding a size of 11, stride is 4 and output is 55x55x96. First layer has 290400 neurons from the calculation of 55x55x96 and weight number at it is 364. Calculation comes 290400x364 = 105,705,600 as the parameter count for the primary convolution layer. This calculates to 61M is the total weight number and there are 724M MACs number in the entire network [39]

Various layers' configurations can be seen on Figure (3.10).

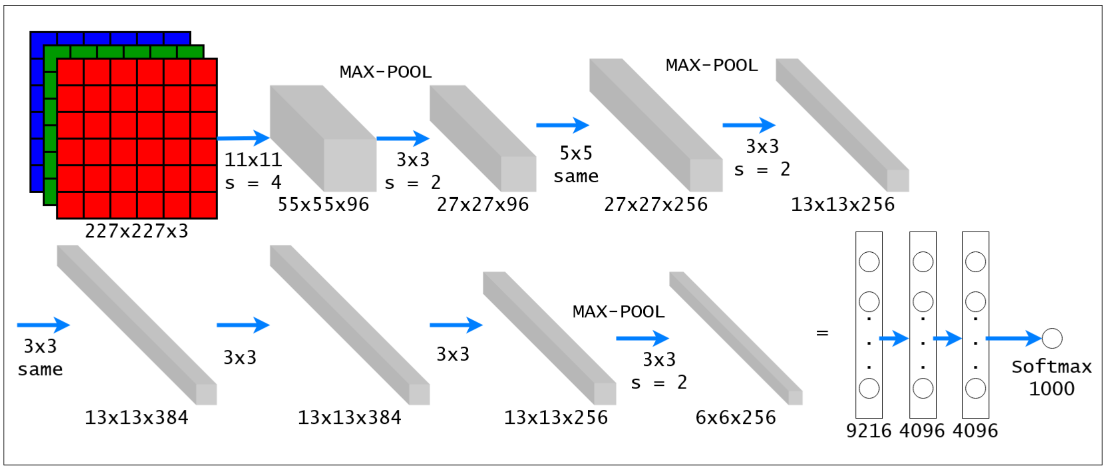


Figure 3.10: Layers of AlexNet architecture [40].

3.4.3 VGGNet

"VGG, short for Visual Geometry Group, has been the most effective at 2014 ILSVRC." [32] Depth of a network proved to be a very important component to achieve higher rate on recognition and classification accuracy among CNNs. VGG architectures uses ReLU activation function in its two convolutional layers. ReLU activation is utilized at an individual maximum pooling layer as well as numerous fully connected layers. At the end for classification a Softmax layer is utilized [32]. In VGG-E [32], filter size of convolution is 3x3 and the stride is 2. VGG-11 with 11, VGG-16 with 16 and VGG-19 with 19 layers are the three VGG-E models. [39].

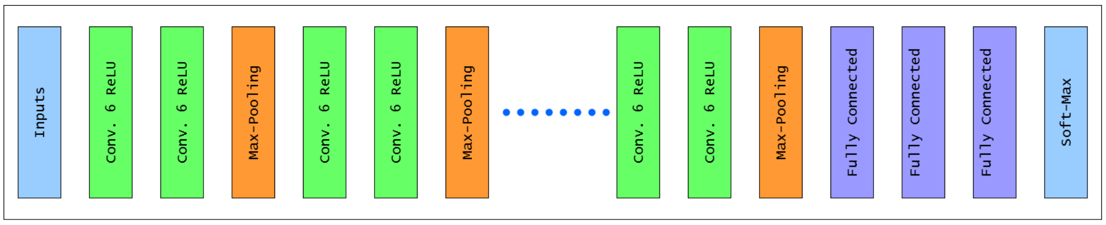


Figure 3.11: Building block of VGG network [39].

3.4.3.1 VGGNet Architecture

VGGNet has two main rules to follow:

1. For every Convolutional layer, configuration: kernel = 3×3, stride = 1×1, padding = same. Only different filter counts.

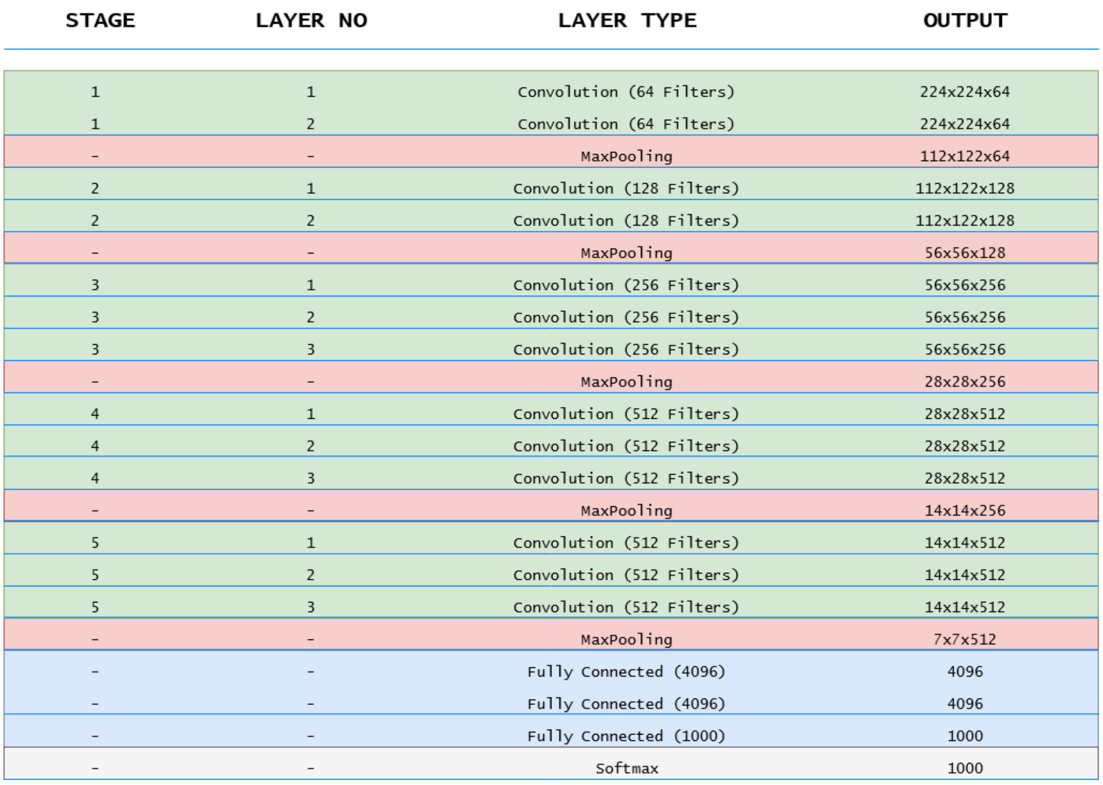
2. For every Maximum Pooling layer, configuration: windows = 2×2 and stride = 2×2. Image size cut down by 2.

If the image input was a RGB of 300x300 pixels. Input size equals to 300x300x3. The fully connected layers are adding the biggest portion of the params [39].

• The first Fully Connected layer contribution = 4096 * (7 * 7 * 512) + 4096 = 102,764,544

• The second Fully Connected layer contribution = 4096 * 4096 + 4096 = 16,781,312

• The third Fully Connected layer contribution = 4096 * 1000 + 4096 = 4,100,096 [39]



STAGE	LAYER NO	LAYER TYPE	OUTPUT
1	1	Convolution (64 Filters)	224x224x64
1	2	Convolution (64 Filters)	224x224x64
-	-	MaxPooling	112x122x64
2	1	Convolution (128 Filters)	112x122x128
2	2	Convolution (128 Filters)	112x122x128
-	-	MaxPooling	56x56x128
3	1	Convolution (256 Filters)	56x56x256
3	2	Convolution (256 Filters)	56x56x256
3	3	Convolution (256 Filters)	56x56x256
-	-	MaxPooling	28x28x256
4	1	Convolution (512 Filters)	28x28x512
4	2	Convolution (512 Filters)	28x28x512
4	3	Convolution (512 Filters)	28x28x512
-	-	MaxPooling	14x14x256
5	1	Convolution (512 Filters)	14x14x512
5	2	Convolution (512 Filters)	14x14x512
5	3	Convolution (512 Filters)	14x14x512
-	-	MaxPooling	7x7x512
-	-	Fully Connected (4096)	4096
-	-	Fully Connected (4096)	4096
-	-	Fully Connected (1000)	1000
-	-	Softmax	1000

Figure 3.12: VGGNet architecture on a table [39].

3.4.4 GoogLeNet

This architecture proves to be a complex architecture because of the Inception modules (3.13) in GoogLeNet [41]. The ImageNet 2014 competition the winner was GoogleNet.It has 22 layers and error rate of 5.7 percent. This architecture is one among the first CNN architectures to avoid stacking convolution and pooling layers in a consecutive structure. Additionally, this new model has a crucial place on memory and power usage. Because stacking all of the layers and adding a huge number of filters adds a calculation and memory cost. GoogLeNet modules are used in parallel to beat this case [23].

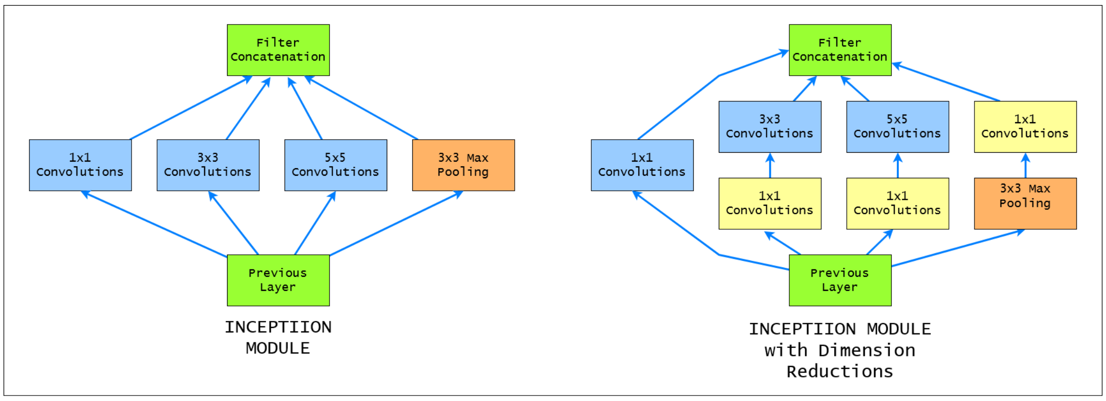


Figure 3.13: Inception Module [40].

Inception module purposes:

1. Abstract results from input of each layer. 3x3 layer will give different information from 5x5 layer.

2. Dimensionality reduction by using 1×1 convolutions [40].

3.4.4.1 GoogLeNet Architecture

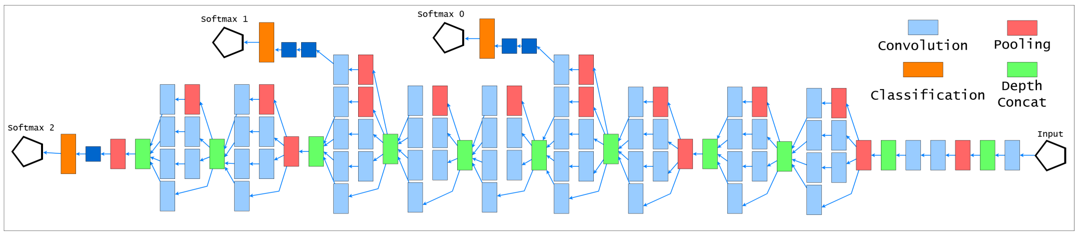


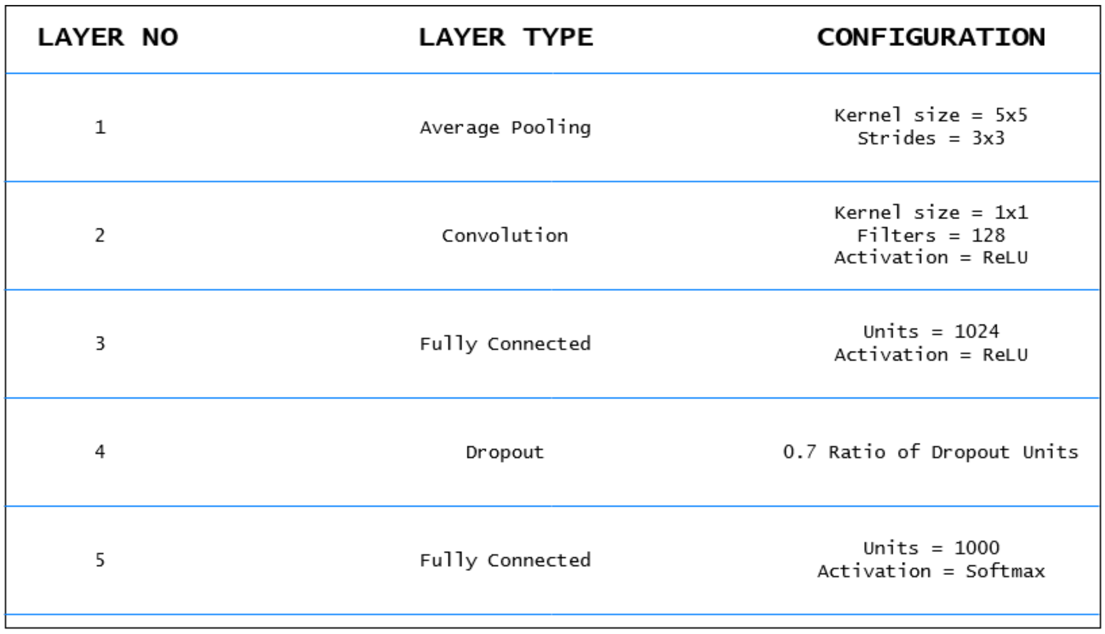
Figure 3.14: GoogLeNet network architecture [41].

These auxiliary classifiers [41] (colored orange (3.14)) can be explained as:

1. Lower stage discrimination: To create more probability of output, gradients from earlier staged layer are used to train lower layers in the network. By this way network can get discriminations about different objects earlier on.

2. Back propagated gradient signal increase: The gradients flowing back becomes smaller and smaller In deep neural networks. This causes the learning rate of sooner layers to be really low. Using classification layer sooner propagates an effective gradients signal to support the network.

3. Additional regularization: Classifiers that used sooner is regulizing the over-fitting effect at deeper layers of DNN's [40].



LAYER NO	LAYER TYPE	CONFIGURATION
1	Average Pooling	Kernel size = 5x5 Strides = 3x3

2	Convolution	Kernel size = 1x1
		Filters = 128
		Activation = ReLU

3	Fully Connected	Units = 1024
		Activation = ReLU

4	Dropout	0.7 Ratio of Dropout Units

5	Fully Connected	Units = 1000
		Activation = Softmax

Figure 3.15: Inception Module Table [40].

3.4.5 Residual Network (ResNet)

ResNet architecture is the winner of ILSVRC 2015 with its 3,6 percent error rate [23, 39]. Kaiming He wanted to solve the problem of vanishing gradient problem. He developed ultra-deep networks to achieve this [35]. ResNet has numbers of layers as 34, 50, 101, 152, 1202. Most well-liked variation of it is ResNet50 which consist of 49 convolutional layers with 1 FC(Fully Connected) layer. Total weight number is 25.5M and MACs number is 3.9M in the entire network [39].

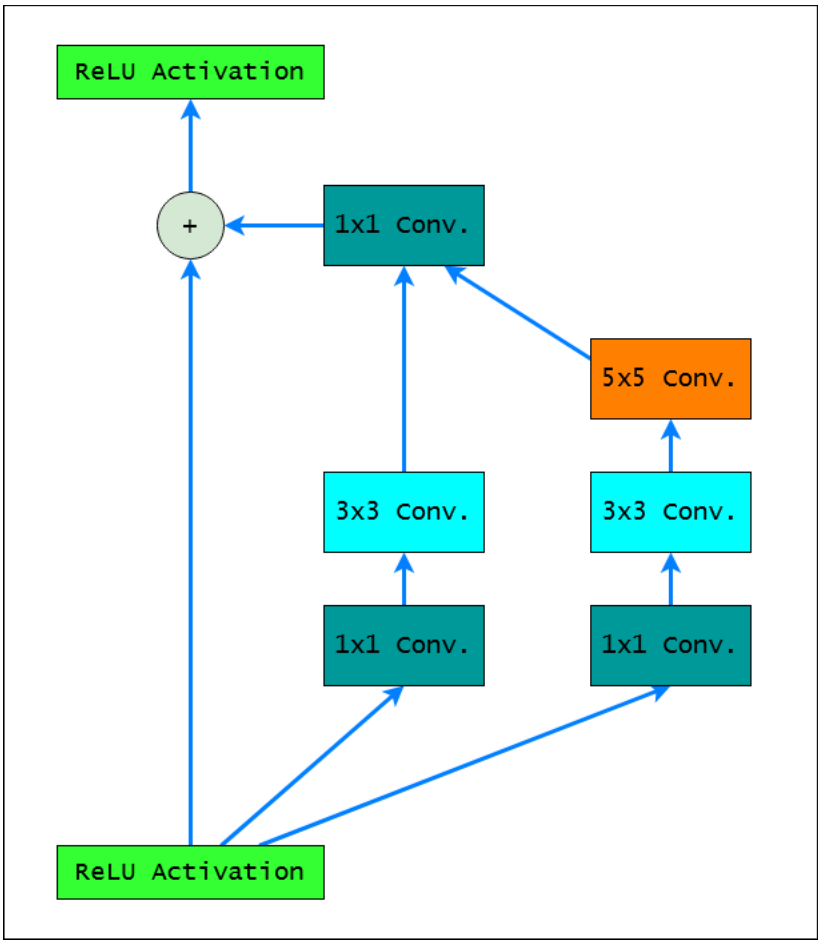


Figure 3.16: Residual unit block diagram [39].

Shahbaz in the web source explains that the network is introducing ”skip connections” which is a completely unique approach. The idea of this came out from an observation. General thought is that DNN are performing worse as more layers are added. But this is not the case. If a network’s performance is assumed as y with k layer, k+1 layers should give a minimum performance of y. The hypothesis of it is hard to learn direct mappings. So learning the difference between output of a layer and its input, learning the residual, is better than learning the mapping between them. This removes the situation of the numbness of DNN that caused by vanishing gradients. A shortcut is created by ”the skip connections” to previous layer gradients that skips numerous layers in between [40].

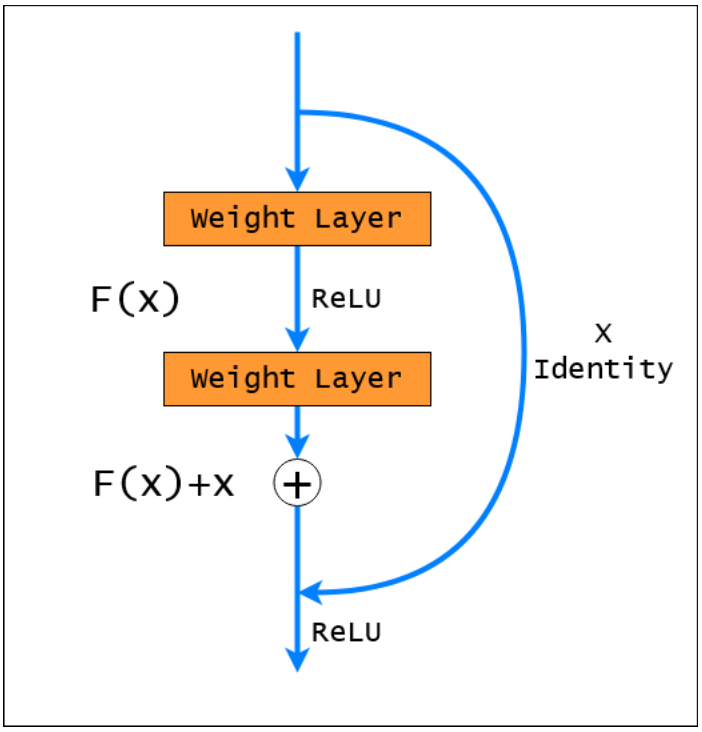


Figure 3.17: Block of residual learning [35].

3.4.5.1 ResNet Architecture

ResNet architecture can be seen as the figure (3.18, 3.19).

ResNet contains 3 sorts of bypass/shortcut connections for smaller dimensions of the input compared to output dimensions.

(A) Getting increased dimensions by adding an extra zero padding.

(B) More parameters are required for utilization of a shortcut for increasing dimensions, the other shortcuts are identity.

(C) Every shortcut is a projection. Extra parameters are on B.

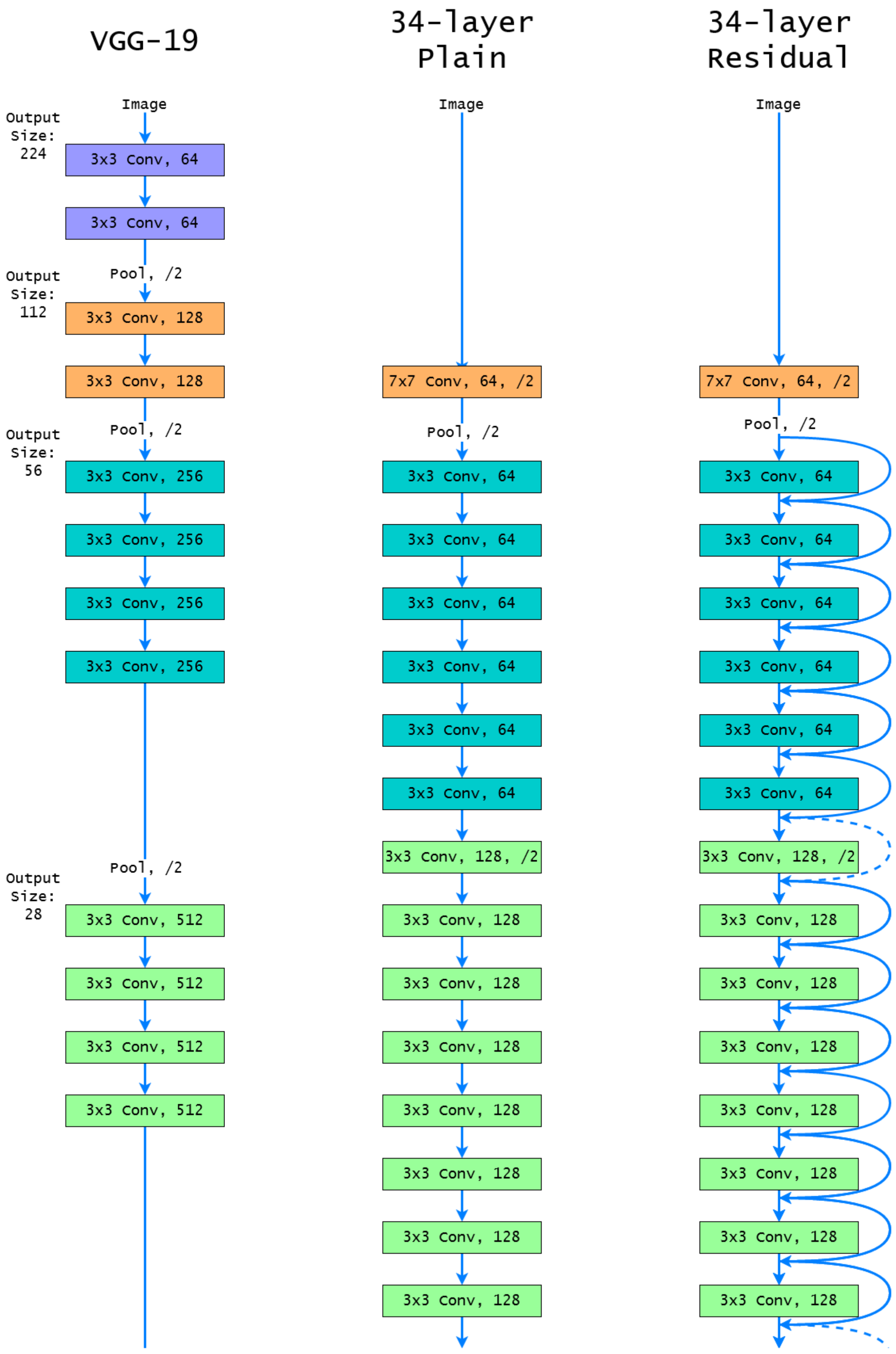


Figure 3.18: PART-1 of ResNet (left), Pure Network (middle), VGG-19 (right) [40].

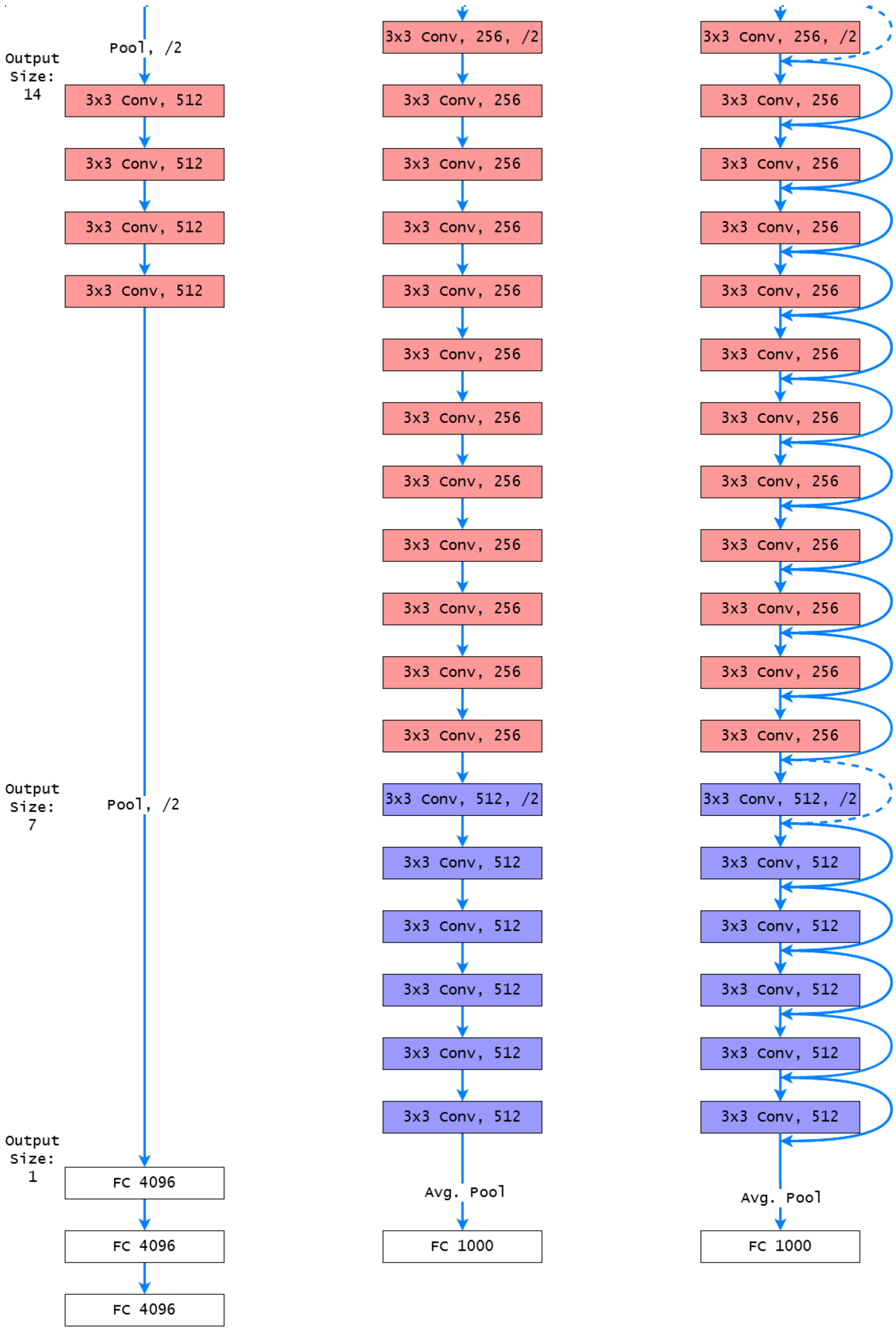


Figure 3.19: PART-2 of ResNet (left), Pure Network (middle), VGG-19 (right) [40].

Explanation of this architecture is given by Tsang says that for reducing parameter amount and not decreasing the network performance a lot, layers of 1x1 convolution layers is added to start and end points of the network [42]. (Figure: 3.20)

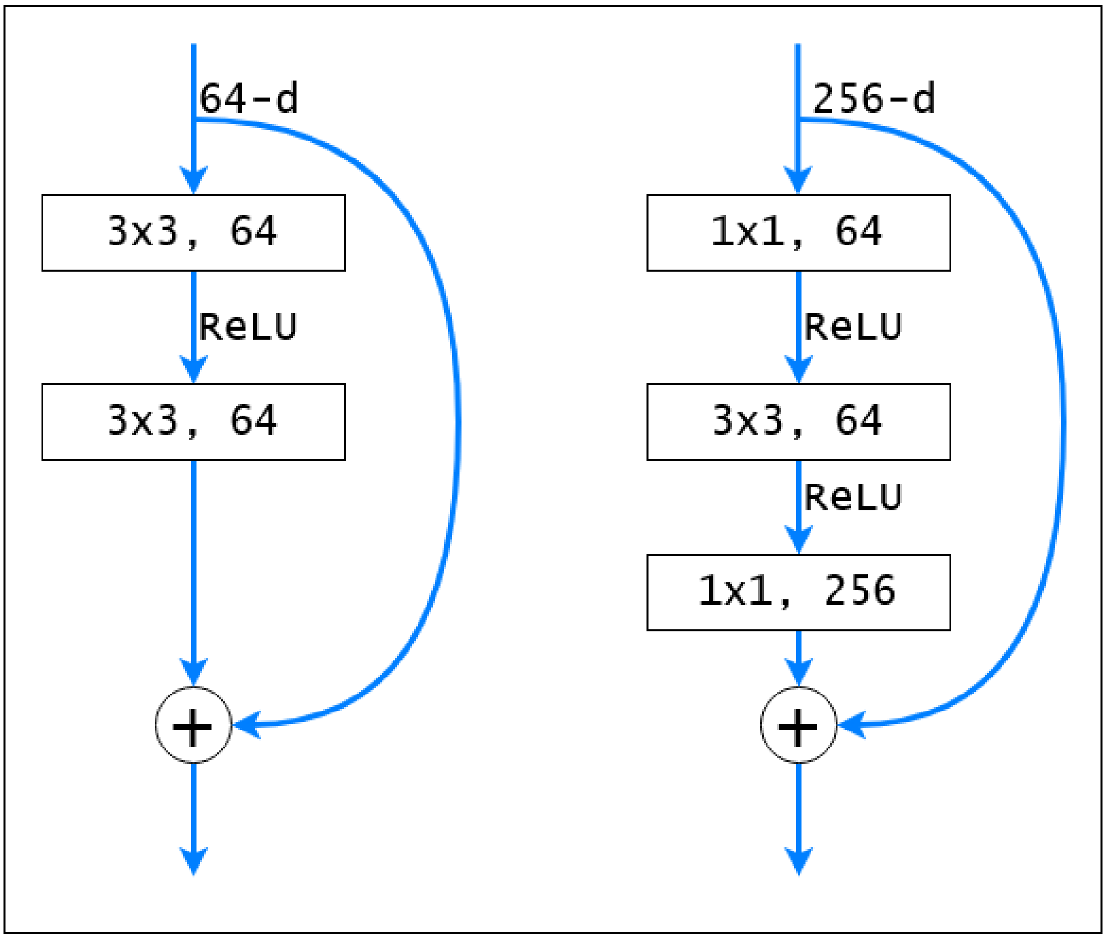
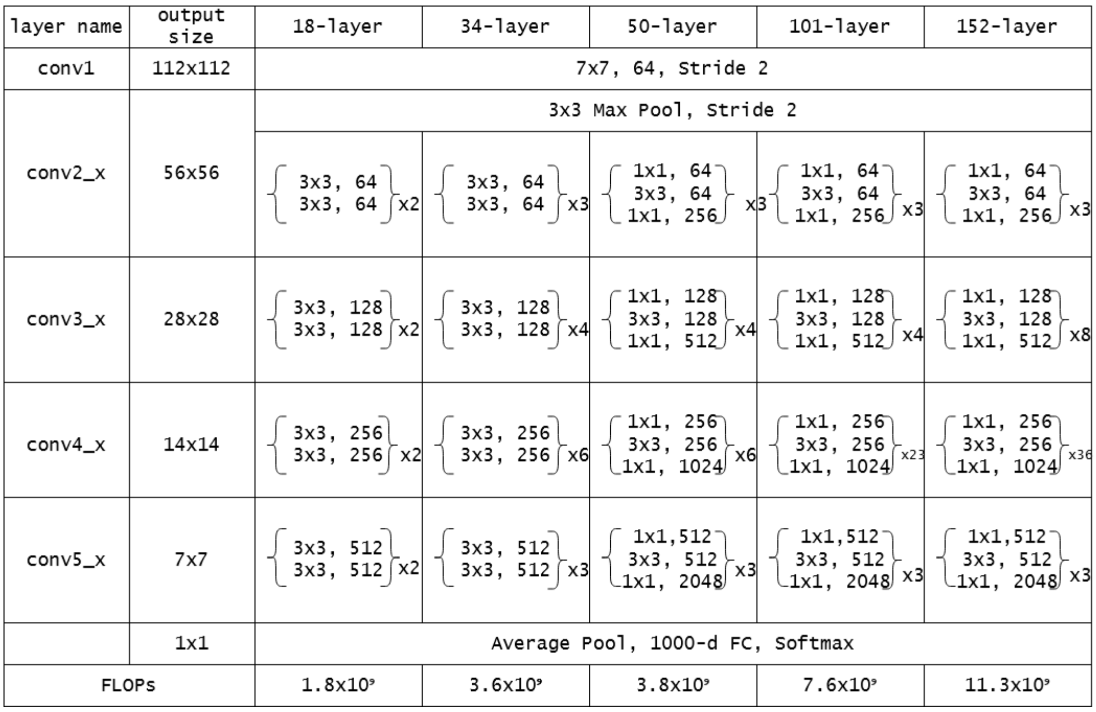


Figure 3.20: Fundamental Block (left) and Design of Bottleneck (right) [40].

34-layer ResNet altered to a 50-Layer Resnet in bottleneck design and ResNet-101 and ResNet-152 are deeper networks that uses bottleneck design [42]. Architecture for all networks is as below figure 3.21:



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112x112	7x7, 64, Stride 2				
conv2_x	56x56	3x3 Max Pool, Stride 2				
		$\begin{Bmatrix} 3x3, 64 \\ 3x3, 64 \end{Bmatrix}$ x2	$\begin{Bmatrix} 3x3, 64 \\ 3x3, 64 \end{Bmatrix}$ x3	$\begin{Bmatrix} 1x1, 64 \\ 3x3, 64 \\ 1x1, 256 \end{Bmatrix}$ x3	$\begin{Bmatrix} 1x1, 64 \\ 3x3, 64 \\ 1x1, 256 \end{Bmatrix}$ x3	$\begin{Bmatrix} 1x1, 64 \\ 3x3, 64 \\ 1x1, 256 \end{Bmatrix}$ x3
conv3_x	28x28	$\begin{Bmatrix} 3x3, 128 \\ 3x3, 128 \end{Bmatrix}$ x2	$\begin{Bmatrix} 3x3, 128 \\ 3x3, 128 \end{Bmatrix}$ x4	$\begin{Bmatrix} 1x1, 128 \\ 3x3, 128 \\ 1x1, 512 \end{Bmatrix}$ x4	$\begin{Bmatrix} 1x1, 128 \\ 3x3, 128 \\ 1x1, 512 \end{Bmatrix}$ x4	$\begin{Bmatrix} 1x1, 128 \\ 3x3, 128 \\ 1x1, 512 \end{Bmatrix}$ x8
conv4_x	14x14	$\begin{Bmatrix} 3x3, 256 \\ 3x3, 256 \end{Bmatrix}$ x2	$\begin{Bmatrix} 3x3, 256 \\ 3x3, 256 \end{Bmatrix}$ x6	$\begin{Bmatrix} 1x1, 256 \\ 3x3, 256 \\ 1x1, 1024 \end{Bmatrix}$ x6	$\begin{Bmatrix} 1x1, 256 \\ 3x3, 256 \\ 1x1, 1024 \end{Bmatrix}$ x23	$\begin{Bmatrix} 1x1, 256 \\ 3x3, 256 \\ 1x1, 1024 \end{Bmatrix}$ x36
conv5_x	7x7	$\begin{Bmatrix} 3x3, 512 \\ 3x3, 512 \end{Bmatrix}$ x2	$\begin{Bmatrix} 3x3, 512 \\ 3x3, 512 \end{Bmatrix}$ x3	$\begin{Bmatrix} 1x1,512 \\ 3x3, 512 \\ 1x1, 2048 \end{Bmatrix}$ x3	$\begin{Bmatrix} 1x1,512 \\ 3x3, 512 \\ 1x1, 2048 \end{Bmatrix}$ x3	$\begin{Bmatrix} 1x1,512 \\ 3x3, 512 \\ 1x1, 2048 \end{Bmatrix}$ x3
	1x1	Average Pool, 1000-d FC, Softmax				
FLOPs		$1.8x10^9$	$3.6x10^9$	$3.8x10^9$	$7.6x10^9$	$11.3x10^9$

Figure 3.21: Complete architecture for each network [42].

3.4.6 Densely Connected Network (DenseNet)

DenseNet comes together with CNN layers that are densely connected. In a dense block, its layer's outputs are connected [39]. It has been announced by Gao Huand et al. at 2017 [43].

By reducing network parameters, feature reuse is an efficient capability of DenseNet.

In DenseNet a number of dense blocks and transition block that are in the middle of side-by-side pair of dense black. (3.22)

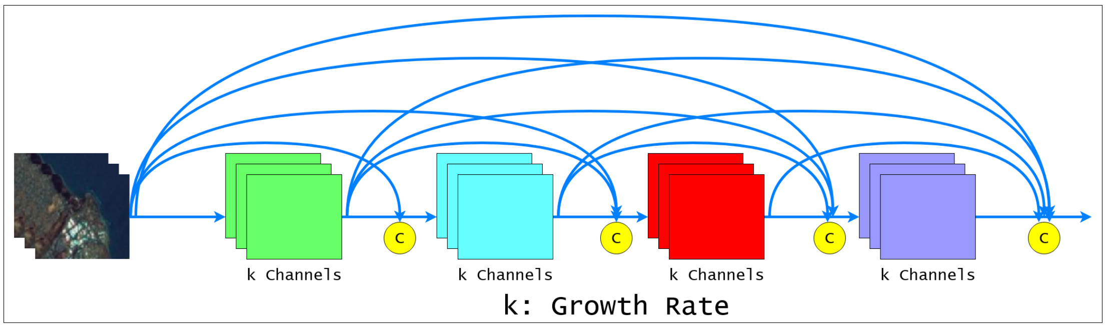


Figure 3.22: Dense Block - Growth Rate of k [42].

The network is compact thanks to supply of feature maps from previous layers to every layer which reduces quantity of channels. Rate of growth is additional number of channels for every layer.

3.4.6.1 DenseNet Architectue

There are 3 layers within the architecture of DenseNet. These layers are:

1. Basic DenseNet Composition Layer

Completing BN-ReLU-1x1 Conv before BN-ReLU-3x3 (3.23) is decreasing model complexity and size [42].

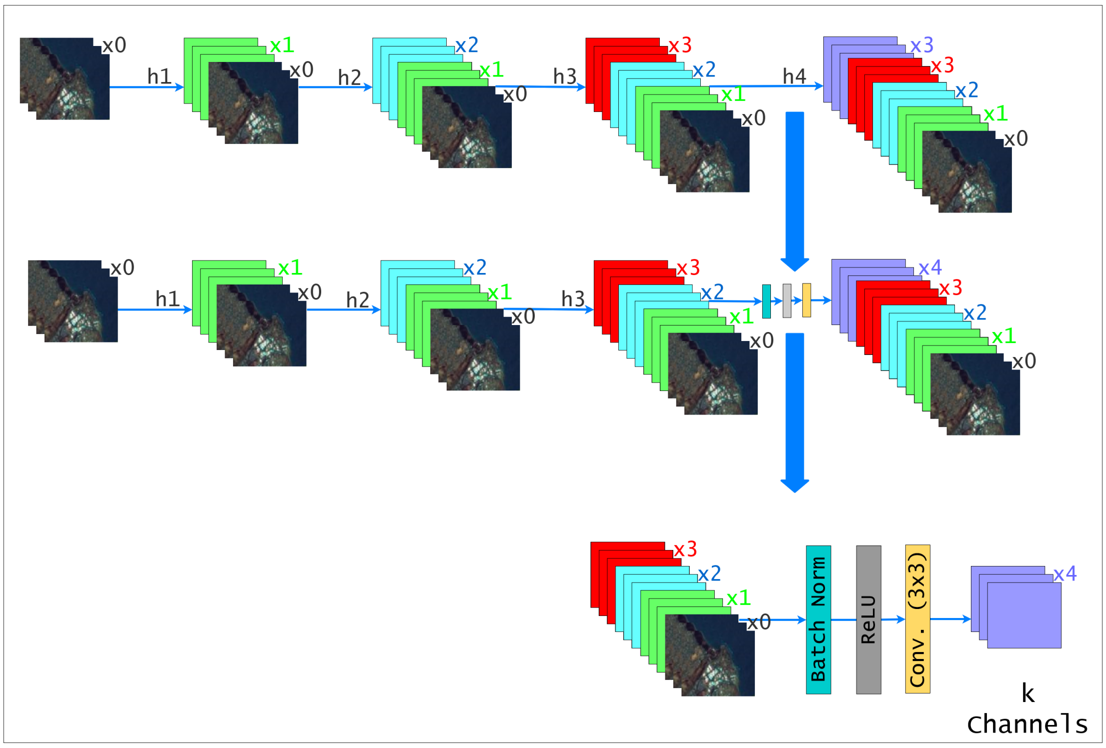


Figure 3.23: Composition Layer [42].

2. DenseNet-B (Bottleneck Layers)

Output feature of k channels with 3x3 ConV are through for each Pre-Activation Batch Norm (BN) and ReLU [42]. (3.24)

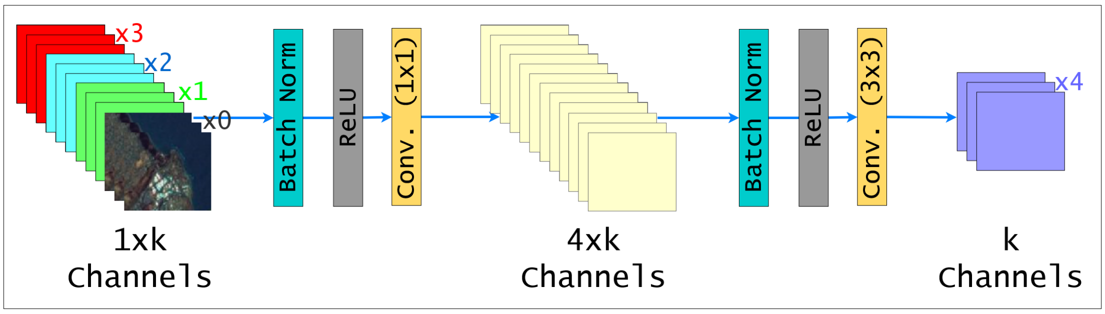


Figure 3.24: DenseNet-B [42].

3. Multiple Dense Blocks with Transition Layers

2x2 average pooling is utilized after 1x1 Conv in the place of transition layers among two contiguous dense blocks. [42] (3.25)

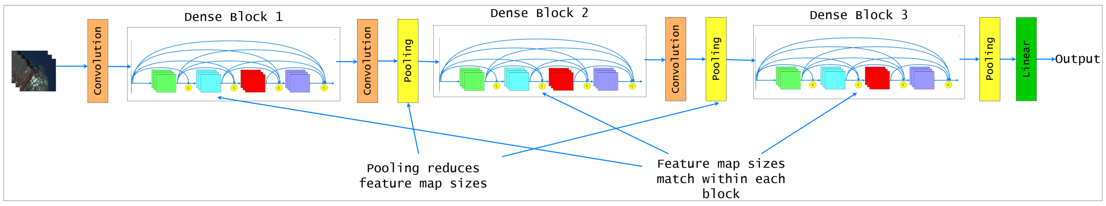


Figure 3.25: Multiple Dense Blocks [42].

To easily concatenate together, sizes of feature map are the same. Feature map sizes are kept identical within dense block.

With the attachment of softmax classifier after a global average pooling, dense blocks comes to an end [42].

Chapter 4

Examination of a Previous Research

This section will consist researches previously done in this field. We will use this section to narrow down our experiment on two or more models. By this way we will not be conducting previous researches and the result from this section will give us an idea to focus on which models for experimental comparison.

4.1 List of Compared DenseNet and ResNet Models

• DenseNet

o DenseNet-121

o DenseNet-161

o DenseNet-169

o DenseNet-201

• ResNet

o ResNet-18

o ResNet-34

o ResNet-50

o ResNet-101

o ResNet-152

4.2 Performance Indices

In order complete this section and reach to its purpose for this research, after various searchings and readings, we have decided that the work of Bianco et al. [44] is an excellent resource.

This work used PyTorch framework to train models or these models are collected from other works as trained and converted in PyTorch.

There will be several titles as performance indices as listed below:

1. Accuracy Rate

Validation set of ImageNet-1k is selected to determine estimated accuracy of the task.

2. Model Complexity

Measured by getting the quantity of parameters. Size is collected in terms of MB.

3. Memory Usage

Memory Allocated and memory required for the process of the batch.

4. Computational Complexity

Measured computational asset for every DNN model considered the FLOPs, short for floating-point operations

5. Inference Time

Bianco et al. [44] reports that for each DNN Model, measuring of inference time per image has been done in milliseconds. We will only use the results from NVIDIA Titan X Pascal GPU but there has been measurements with NVIDIA Jetson TX1 as well.

4.3 Comparison

4.3.1 Model Complexity VS Accuracy Rate VS Computational Complexity

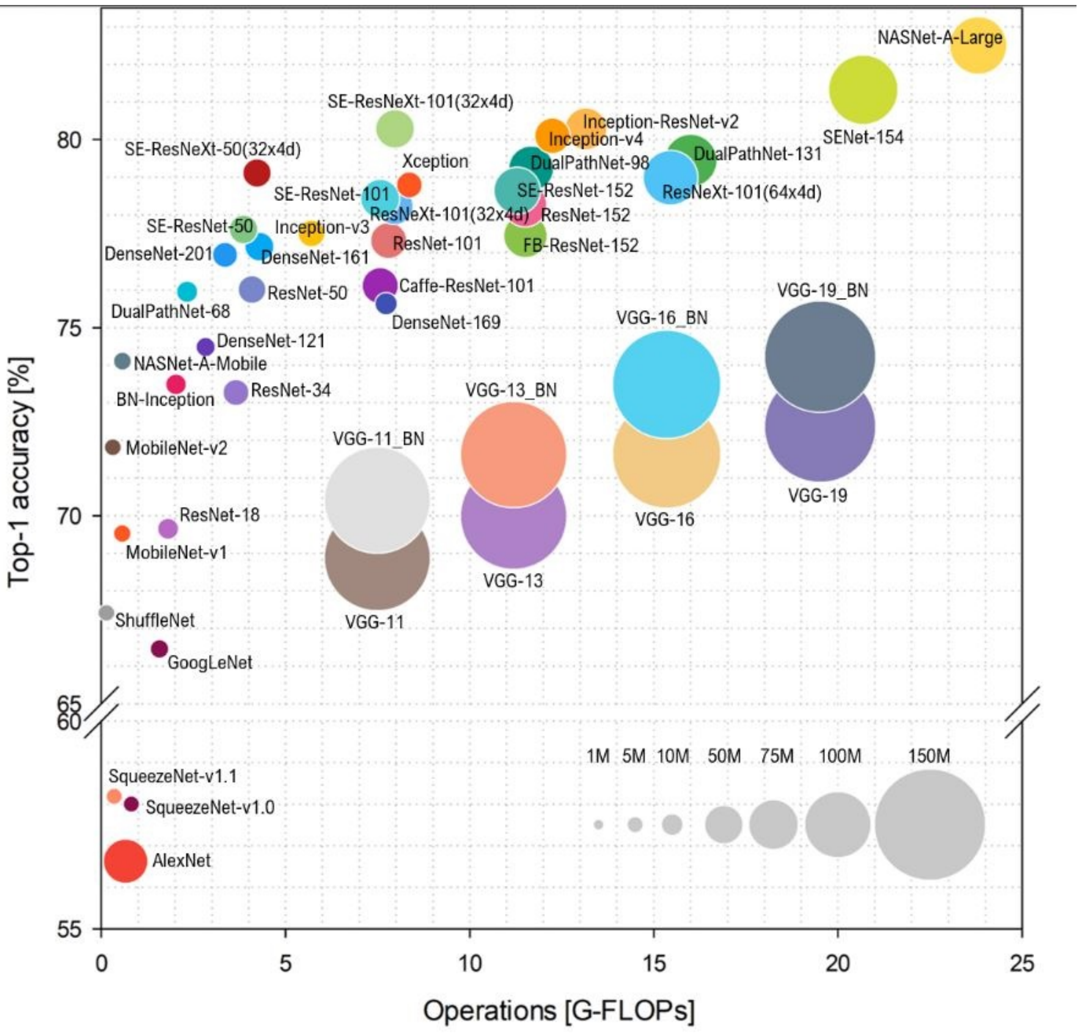


Figure 4.1: Computational complexity vs Accuracy [44].

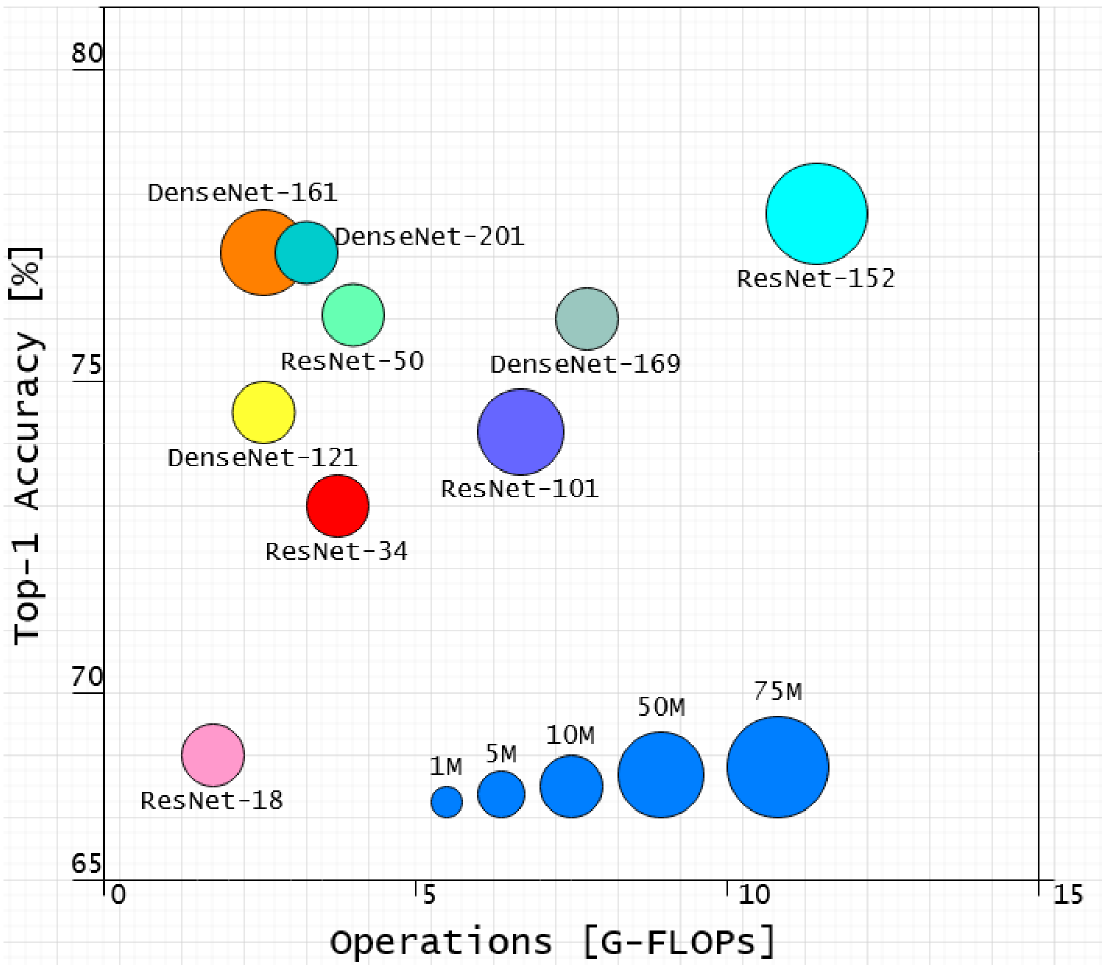


Figure 4.2: Computational complexity vs Accuracy (Simplified).

These accuracy values (4.2) are created with regard to computational complexity for a single forward pass. As in others ImageNet-1k is used [44].

Model	Accuracy	Operations[G-FLOPS]	Model Complexity
DenseNet-121	~74 percent	2.8	10M
DenseNet-161	~77 percent	2.8	50M
DenseNet-169	~76 percent	7.9	10M
DenseNet-201	~77 percent	3.2	10M
ResNet-18	~69 percent	1.8	10M
ResNet-34	~73 percent	3.6	10M
ResNet-50	~76 percent	4.0	10M
ResNet-101	~74 percent	7.9	50M
ResNet-152	~78 percent	11.5	75M

Table 4.1: Extraction from Figure(4.2).

In terms of Model Complexity, shoreline dataset have very high data sizes. This causes longer process times.

As in table-1 ResNet-152 have higher learnable parameter with 75M and highest accuracy with a percentage of nearly 78 percent.

4.3.2 Learning Power vs Accuracy Rate

The inefficiency of DNNs on full learning power usage is a known fact [44]. Despite this there are numerous papers that exploits this feature for producing DNN models in a compressed from with the accuracy of original models [41].

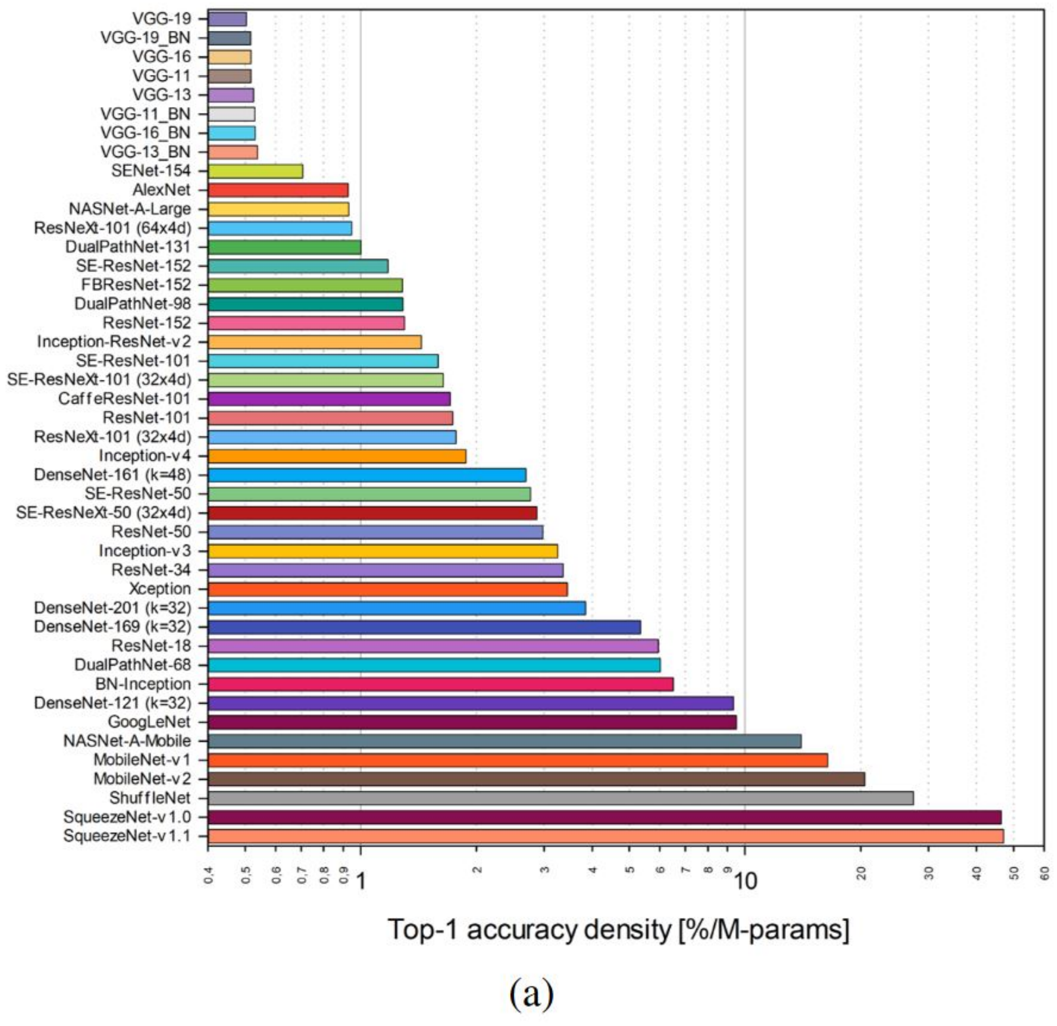
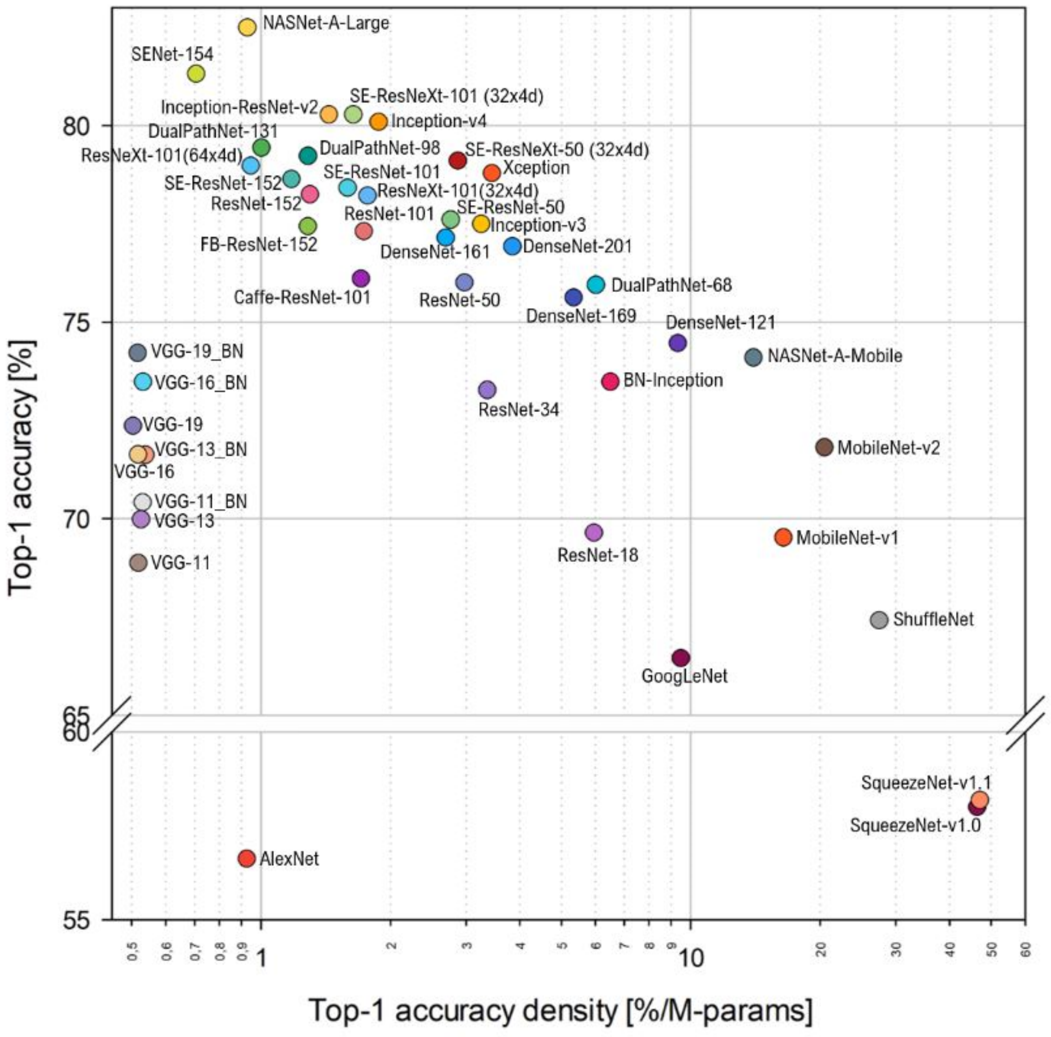


Figure 4.3: Accuracy density [44].



(b)

Figure 4.4: Accuracy vs. Accuracy density [44].

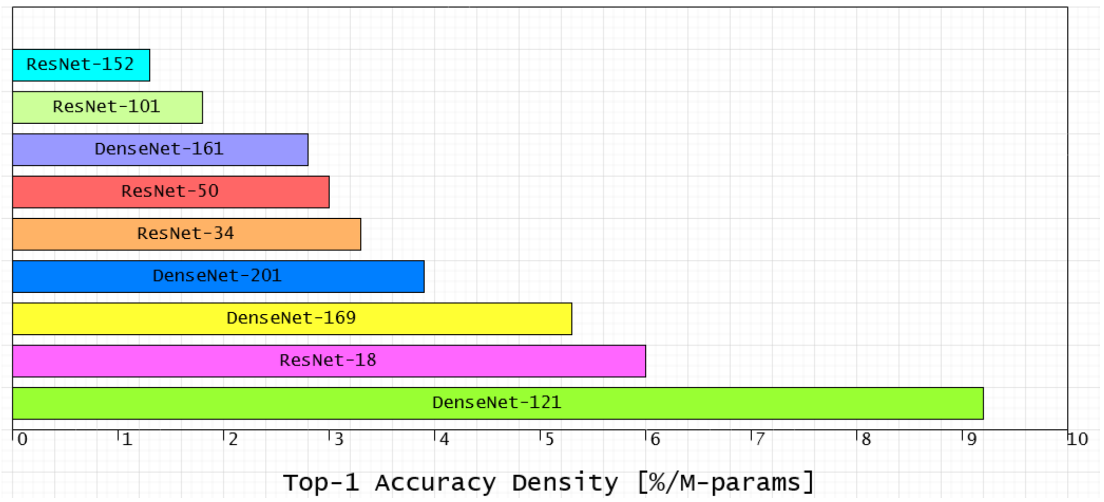


Figure 4.5: Accuracy density(Simplified).

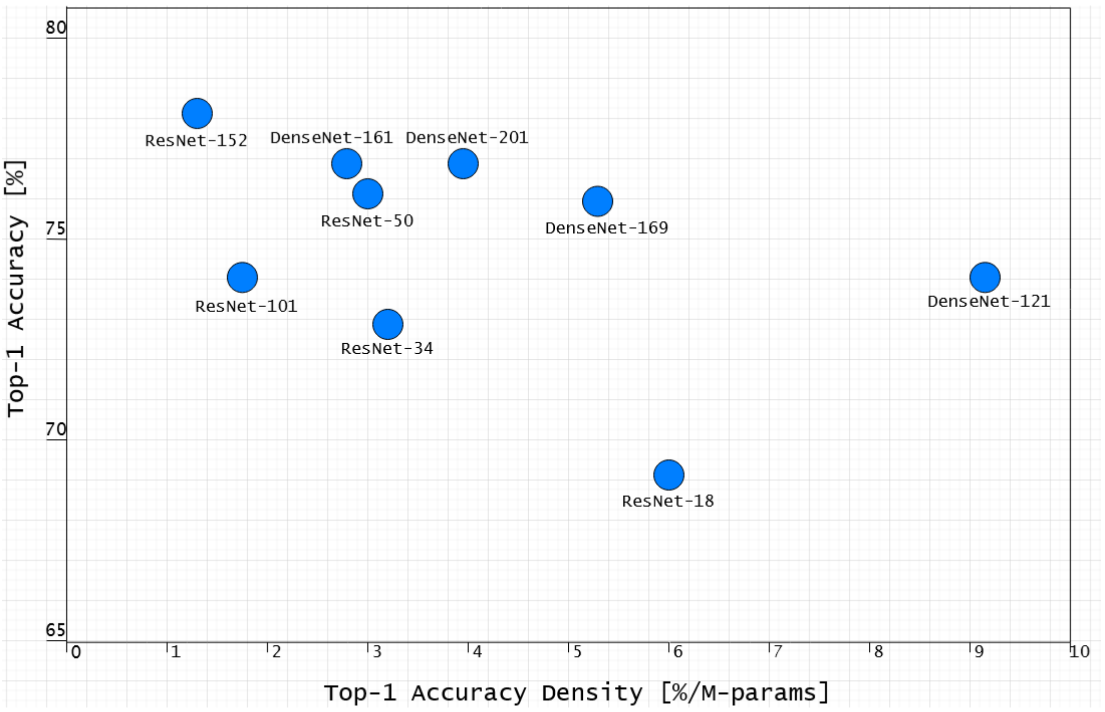


Figure 4.6: Accuracy vs. Accuracy density(Simplified).

Accuracy divided by the parameter quantity, this value's height is showing the efficiency.

Model	**Accuracy**	**Accuracy Density**
DenseNet-121	∼74 percent	9.2
DenseNet-161	∼77 percent	2.8
DenseNet-169	∼76 percent	5.3
DenseNet-201	∼77 percent	3.9
ResNet-18	∼69 percent	6.0
ResNet-34	∼73 percent	3.3
ResNet-50	∼76 percent	3.0
ResNet-101	∼74 percent	1.8
ResNet-152	∼78 percent	1.3

Table 4.2: Extraction from Figure(4.3) and (4.4).

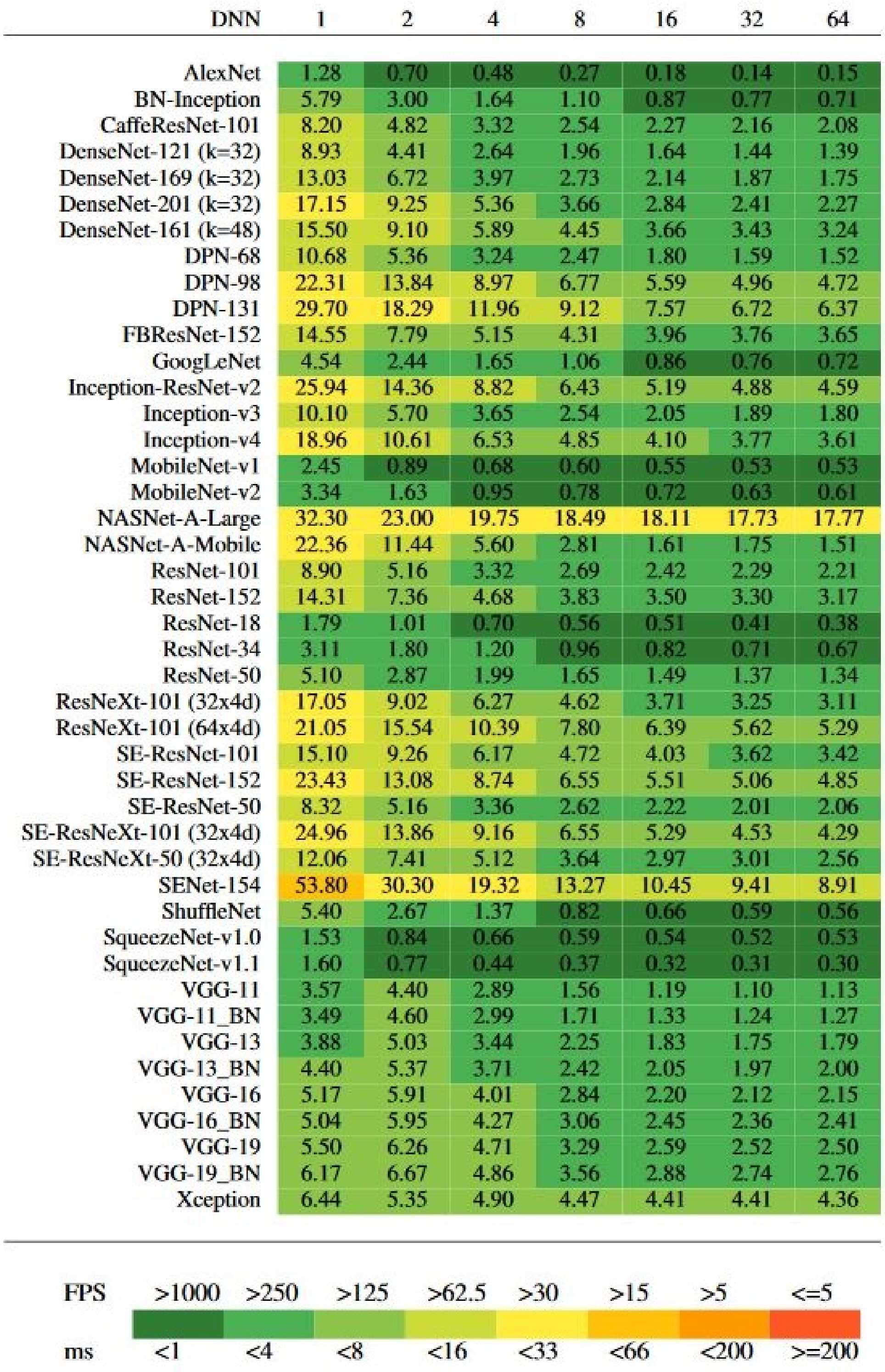
Between subjects that got the most accuracy rate (i.e. higher or equal to 77 percent), we are able to say that the model using its parameters more efficiently is ResNet-152.

4.3.3 Inference Time

Done in 10 runs. For batch sizes 1 to 64 average inference time for per image is recorded. In (4.7) are color coded for easy distinction in frames per second (FPS).

Model	**Batch Size 1**	**2**	**4**	**8**	**16**	**32**	**64**
DenseNet-121	8.93	4.41	2.64	1.96	1.64	1.44	1.39
DenseNet-161	15.50	9.10	5.89	4.45	3.66	3.43	3.24
DenseNet-169	13.03	6.72	3.97	2.73	2.14	1.87	1.75
DenseNet-201	17.15	9.25	5.36	3.66	2.84	2.41	2.27
ResNet-18	1.79	1.01	0.70	0.56	0.51	0.41	0.38
ResNet-34	3.11	1.80	1.20	0.96	0.82	0.71	0.67
ResNet-50	5.10	2.87	1.99	1.65	1.49	1.37	1.34
ResNet-101	8.90	5.16	3.32	2.69	2.42	2.29	2.21
ResNet-152	14.31	7.36	4.68	3.83	3.50	3.30	3.17

Table 4.3: Extraction from Figure(4.7).



DNN	1	2	4	8	16	32	64
AlexNet	1.28	0.70	0.48	0.27	0.18	0.14	0.15
BN-Inception	5.79	3.00	1.64	1.10	0.87	0.77	0.71
CaffeResNet-101	8.20	4.82	3.32	2.54	2.27	2.16	2.08
DenseNet-121 (k=32)	8.93	4.41	2.64	1.96	1.64	1.44	1.39
DenseNet-169 (k=32)	13.03	6.72	3.97	2.73	2.14	1.87	1.75
DenseNet-201 (k=32)	17.15	9.25	5.36	3.66	2.84	2.41	2.27
DenseNet-161 (k=48)	15.50	9.10	5.89	4.45	3.66	3.43	3.24
DPN-68	10.68	5.36	3.24	2.47	1.80	1.59	1.52
DPN-98	22.31	13.84	8.97	6.77	5.59	4.96	4.72
DPN-131	29.70	18.29	11.96	9.12	7.57	6.72	6.37
FBResNet-152	14.55	7.79	5.15	4.31	3.96	3.76	3.65
GoogLeNet	4.54	2.44	1.65	1.06	0.86	0.76	0.72
Inception-ResNet-v2	25.94	14.36	8.82	6.43	5.19	4.88	4.59
Inception-v3	10.10	5.70	3.65	2.54	2.05	1.89	1.80
Inception-v4	18.96	10.61	6.53	4.85	4.10	3.77	3.61
MobileNet-v1	2.45	0.89	0.68	0.60	0.55	0.53	0.53
MobileNet-v2	3.34	1.63	0.95	0.78	0.72	0.63	0.61
NASNet-A-Large	32.30	23.00	19.75	18.49	18.11	17.73	17.77
NASNet-A-Mobile	22.36	11.44	5.60	2.81	1.61	1.75	1.51
ResNet-101	8.90	5.16	3.32	2.69	2.42	2.29	2.21
ResNet-152	14.31	7.36	4.68	3.83	3.50	3.30	3.17
ResNet-18	1.79	1.01	0.70	0.56	0.51	0.41	0.38
ResNet-34	3.11	1.80	1.20	0.96	0.82	0.71	0.67
ResNet-50	5.10	2.87	1.99	1.65	1.49	1.37	1.34
ResNeXt-101 (32x4d)	17.05	9.02	6.27	4.62	3.71	3.25	3.11
ResNeXt-101 (64x4d)	21.05	15.54	10.39	7.80	6.39	5.62	5.29
SE-ResNet-101	15.10	9.26	6.17	4.72	4.03	3.62	3.42
SE-ResNet-152	23.43	13.08	8.74	6.55	5.51	5.06	4.85
SE-ResNet-50	8.32	5.16	3.36	2.62	2.22	2.01	2.06
SE-ResNeXt-101 (32x4d)	24.96	13.86	9.16	6.55	5.29	4.53	4.29
SE-ResNeXt-50 (32x4d)	12.06	7.41	5.12	3.64	2.97	3.01	2.56
SENet-154	53.80	30.30	19.32	13.27	10.45	9.41	8.91
ShuffleNet	5.40	2.67	1.37	0.82	0.66	0.59	0.56
SqueezeNet-v1.0	1.53	0.84	0.66	0.59	0.54	0.52	0.53
SqueezeNet-v1.1	1.60	0.77	0.44	0.37	0.32	0.31	0.30
VGG-11	3.57	4.40	2.89	1.56	1.19	1.10	1.13
VGG-11_BN	3.49	4.60	2.99	1.71	1.33	1.24	1.27
VGG-13	3.88	5.03	3.44	2.25	1.83	1.75	1.79
VGG-13_BN	4.40	5.37	3.71	2.42	2.05	1.97	2.00
VGG-16	5.17	5.91	4.01	2.84	2.20	2.12	2.15
VGG-16_BN	5.04	5.95	4.27	3.06	2.45	2.36	2.41
VGG-19	5.50	6.26	4.71	3.29	2.59	2.52	2.50
VGG-19_BN	6.17	6.67	4.86	3.56	2.88	2.74	2.76
Xception	6.44	5.35	4.90	4.47	4.41	4.41	4.36

FPS	>1000	>250	>125	>62.5	>30	>15	>5	<=5
	🟩	🟩	🟩	🟨	🟨	🟧	🟧	🟥
ms	<1	<4	<8	<16	<33	<66	<200	>=200

Figure 4.7: Inference time vs. batch size [44].

Due to its low computational complexity ResNet-18 is the fastest model on extraction table 4.3.

However we have to consider accuracy as well while deciding which model is efficient when inference time is taken into account, which will be considered on the next perfomance indice title 4.3.4.

4.3.4 Inference Time vs Accuracy-Rate

On the table 4.4 we are able to assess each model for its FPS performance with its accuracy. Higher accuracy with high fps is healthier. As it's clear that every model show great fps results. However an equilibrium point is required to choose which model is the best.

When accuracy and image per second considered ResNet-50 is the most efficient model as it can be seen on the table.

Model	**Accuracy**	**FPS**
DenseNet-121	$\sim$74 percent	>100
DenseNet-161	$\sim$77 percent	$\sim$70
DenseNet-169	$\sim$76 percent	>75
DenseNet-201	$\sim$77 percent	$\sim$60
ResNet-18	$\sim$69percent	$\sim$550
ResNet-34	$\sim$73 percent	>300
ResNet-50	$\sim$76 percent	$\sim$200
ResNet-101	$\sim$74 percent	>100
ResNet-152	$\sim$78 percent	$\sim$70

Table 4.4: Extraction from Figure(4.8).

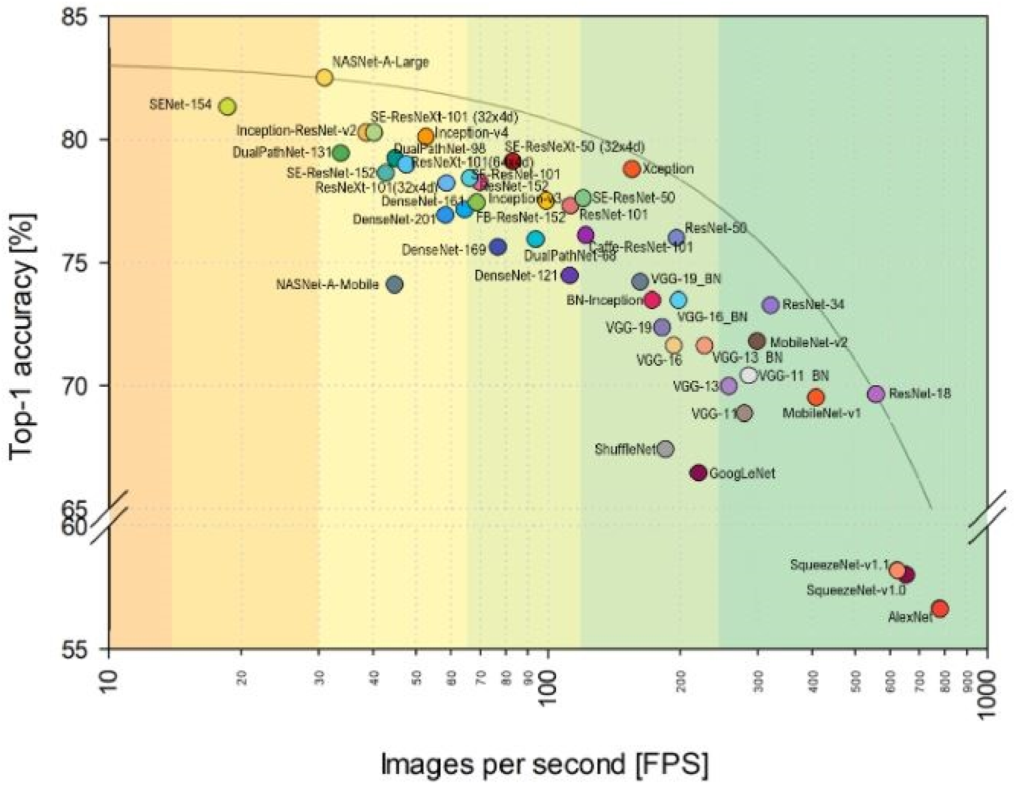


Figure 4.8: Analysis with accuracy and FPS [44].

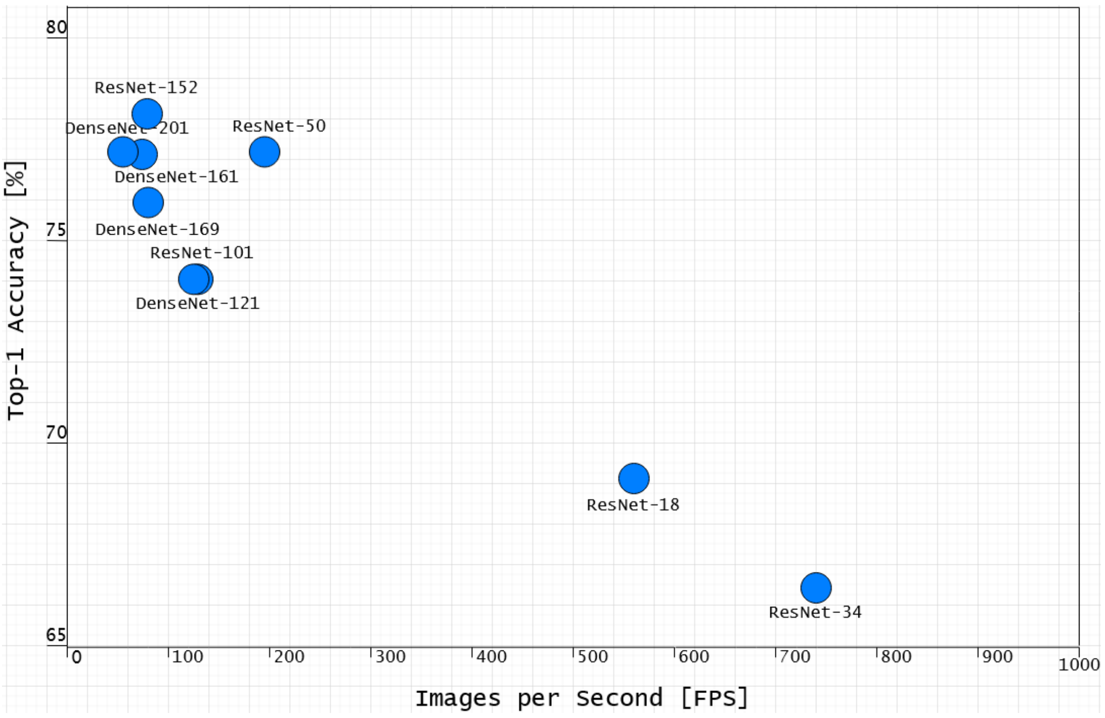


Figure 4.9: Analysis with accuracy and FPS(Simplified).

4.3.5 Usage of Memory

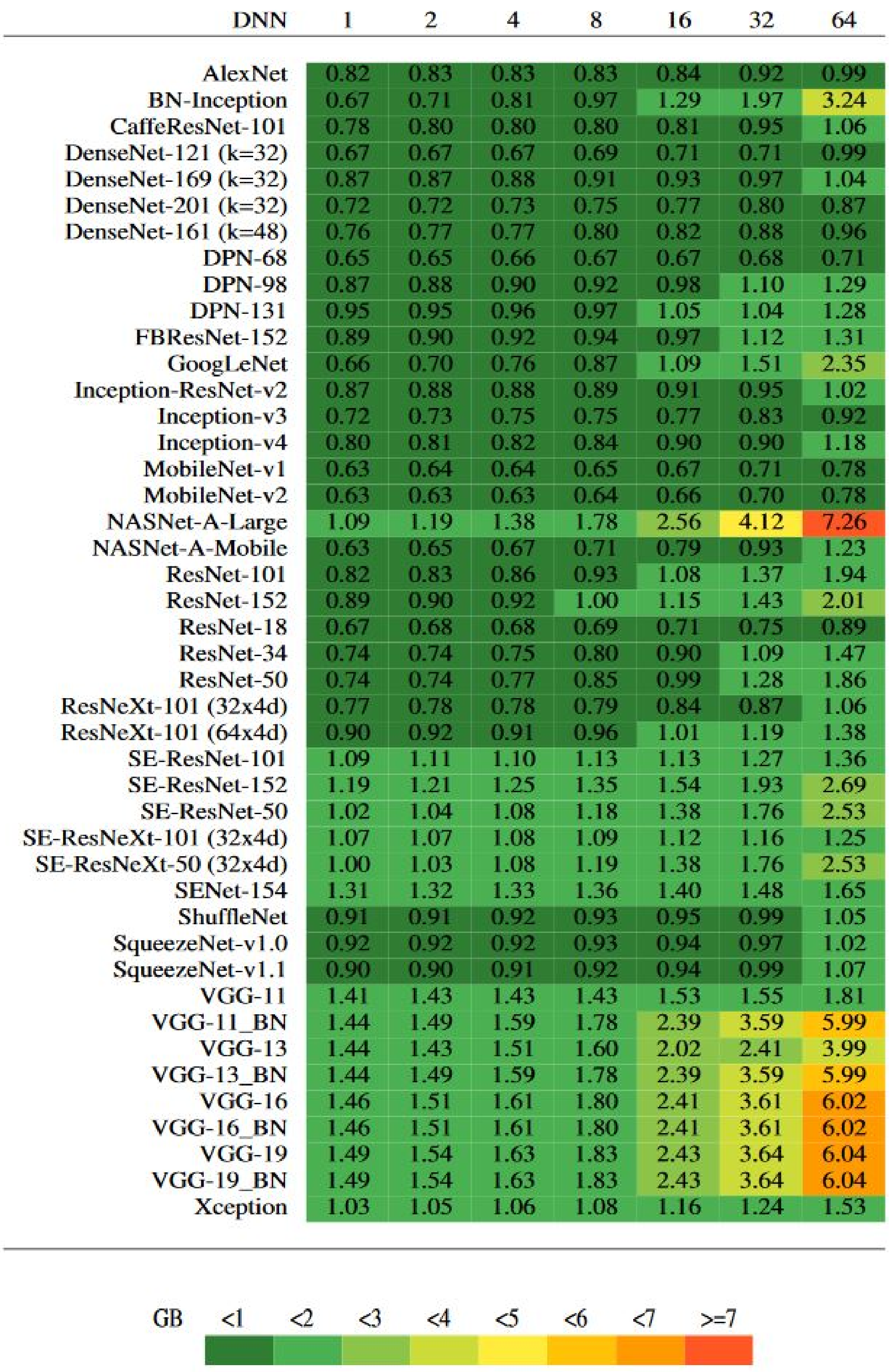
In Figure (4.7) GB correspond of the memory usage at each DNN models taken into account for numerous batch sizes on can be seen.

Model	**Batch Size 1**	**2**	**4**	**8**	**16**	**32**	**64**
DenseNet-121	0.67	0.71	0.81	0.97	1.29	1.97	3.24
DenseNet-161	0.76	0.77	0.77	0.80	0.82	0.88	0.96
DenseNet-169	0.87	0.87	0.88	0.91	0.93	0.97	1.04
DenseNet-201	0.72	0.72	0.73	0.75	0.77	0.80	0.87
ResNet-18	0.67	0.68	0.68	0.69	0.71	0.75	0.89
ResNet-34	0.74	0.74	0.75	0.80	0.90	1.09	1.47
ResNet-50	0.74	0.74	0.77	0.85	0.99	1.28	1.86
ResNet-101	0.82	0.83	0.86	0.93	1.08	1.37	1.94
ResNet-152	0.89	0.90	0.92	1.00	1.15	1.43	2.01

Table 4.5: Extraction from Figure(4.10).

Among our models in terms of memory usage for each batch size it is obvious that DenseNet-121 and ResNet-18 has lowest results on the table 4.5.

But to decide the most efficient model, model complexity and memory consumption must be considered. Next title will have this consideration, 4.3.6.



DNN	1	2	4	8	16	32	64
AlexNet	0.82	0.83	0.83	0.83	0.84	0.92	0.99
BN-Inception	0.67	0.71	0.81	0.97	1.29	1.97	3.24
CaffeResNet-101	0.78	0.80	0.80	0.80	0.81	0.95	1.06
DenseNet-121 (k=32)	0.67	0.67	0.67	0.69	0.71	0.71	0.99
DenseNet-169 (k=32)	0.87	0.87	0.88	0.91	0.93	0.97	1.04
DenseNet-201 (k=32)	0.72	0.72	0.73	0.75	0.77	0.80	0.87
DenseNet-161 (k=48)	0.76	0.77	0.77	0.80	0.82	0.88	0.96
DPN-68	0.65	0.65	0.66	0.67	0.67	0.68	0.71
DPN-98	0.87	0.88	0.90	0.92	0.98	1.10	1.29
DPN-131	0.95	0.95	0.96	0.97	1.05	1.04	1.28
FBResNet-152	0.89	0.90	0.92	0.94	0.97	1.12	1.31
GoogLeNet	0.66	0.70	0.76	0.87	1.09	1.51	2.35
Inception-ResNet-v2	0.87	0.88	0.88	0.89	0.91	0.95	1.02
Inception-v3	0.72	0.73	0.75	0.75	0.77	0.83	0.92
Inception-v4	0.80	0.81	0.82	0.84	0.90	0.90	1.18
MobileNet-v1	0.63	0.64	0.64	0.65	0.67	0.71	0.78
MobileNet-v2	0.63	0.63	0.63	0.64	0.66	0.70	0.78
NASNet-A-Large	1.09	1.19	1.38	1.78	2.56	4.12	7.26
NASNet-A-Mobile	0.63	0.65	0.67	0.71	0.79	0.93	1.23
ResNet-101	0.82	0.83	0.86	0.93	1.08	1.37	1.94
ResNet-152	0.89	0.90	0.92	1.00	1.15	1.43	2.01
ResNet-18	0.67	0.68	0.68	0.69	0.71	0.75	0.89
ResNet-34	0.74	0.74	0.75	0.80	0.90	1.09	1.47
ResNet-50	0.74	0.74	0.77	0.85	0.99	1.28	1.86
ResNeXt-101 (32x4d)	0.77	0.78	0.78	0.79	0.84	0.87	1.06
ResNeXt-101 (64x4d)	0.90	0.92	0.91	0.96	1.01	1.19	1.38
SE-ResNet-101	1.09	1.11	1.10	1.13	1.13	1.27	1.36
SE-ResNet-152	1.19	1.21	1.25	1.35	1.54	1.93	2.69
SE-ResNet-50	1.02	1.04	1.08	1.18	1.38	1.76	2.53
SE-ResNeXt-101 (32x4d)	1.07	1.07	1.08	1.09	1.12	1.16	1.25
SE-ResNeXt-50 (32x4d)	1.00	1.03	1.08	1.19	1.38	1.76	2.53
SENet-154	1.31	1.32	1.33	1.36	1.40	1.48	1.65
ShuffleNet	0.91	0.91	0.92	0.93	0.95	0.99	1.05
SqueezeNet-v1.0	0.92	0.92	0.92	0.93	0.94	0.97	1.02
SqueezeNet-v1.1	0.90	0.90	0.91	0.92	0.94	0.99	1.07
VGG-11	1.41	1.43	1.43	1.43	1.53	1.55	1.81
VGG-11_BN	1.44	1.49	1.59	1.78	2.39	3.59	5.99
VGG-13	1.44	1.43	1.51	1.60	2.02	2.41	3.99
VGG-13_BN	1.44	1.49	1.59	1.78	2.39	3.59	5.99
VGG-16	1.46	1.51	1.61	1.80	2.41	3.61	6.02
VGG-16_BN	1.46	1.51	1.61	1.80	2.41	3.61	6.02
VGG-19	1.49	1.54	1.63	1.83	2.43	3.64	6.04
VGG-19_BN	1.49	1.54	1.63	1.83	2.43	3.64	6.04
Xception	1.03	1.05	1.06	1.08	1.16	1.24	1.53

GB	<1	<2	<3	<4	<5	<6	<7	>=7
	🟩	🟩	🟩	🟨	🟨	🟧	🟧	🟥

Figure 4.10: Memory utilization [44].

4.3.6 Model Complexity vs Memory Usage

In Figure (4.11) Bianco et. al. review the connection between the one by one model parameter stationary allocation. There is a linear relationship with slope value 1.10 and 1.15. Interceptions are 910 and 639 as different values. Examinations proves that for estimating a total memory utilization, the model complexity can be used [44].

Model	GPU Memory Utilization(GB)	Parameters(MB)
DenseNet-121	0.67	~30
DenseNet-161	0.76	~110
DenseNet-169	0.87	~60
DenseNet-201	0.72	~90
ResNet-18	0.67	~40
ResNet-34	0.74	~90
ResNet-50	0.74	~100
ResNet-101	0.82	~180
ResNet-152	0.89	~220

Table 4.6: Extraction from Figure(4.11).

Since all models required memory lower than 1 GB. It is fair to decide most efficient model by its parameter value. Because todays technological advancement in computer components is making 1GB of GPU Memory highly accessible. By this comparison we can say that ResNet-152 is the most efficient model on the table (5.9).

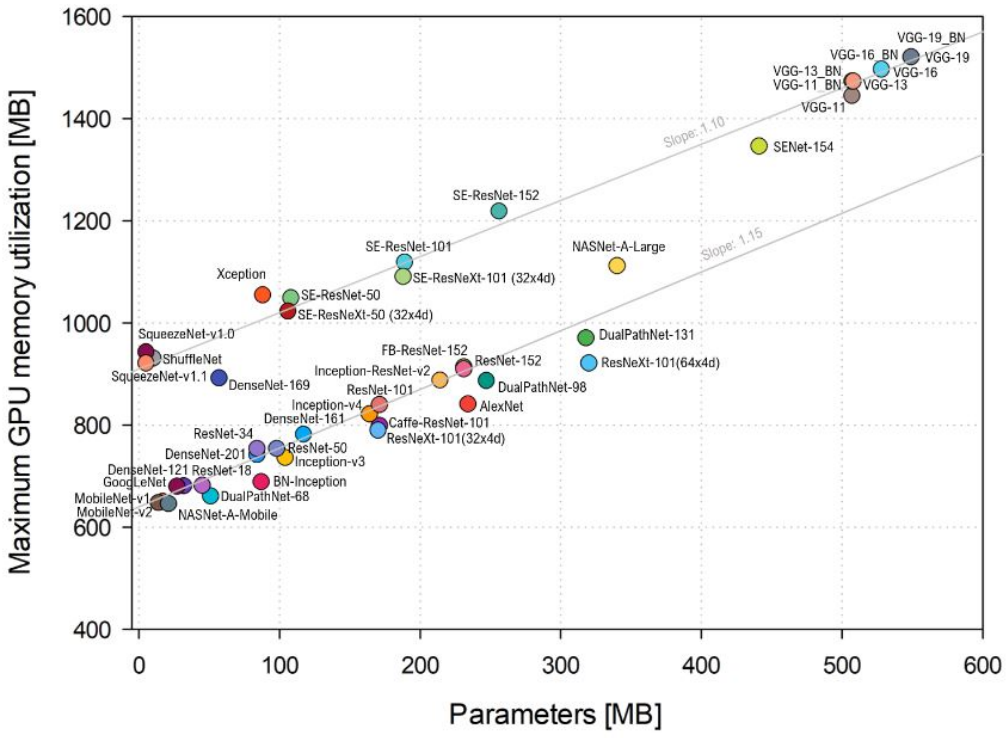


Figure 4.11: Model Complexity vs Memory Utilization [44].

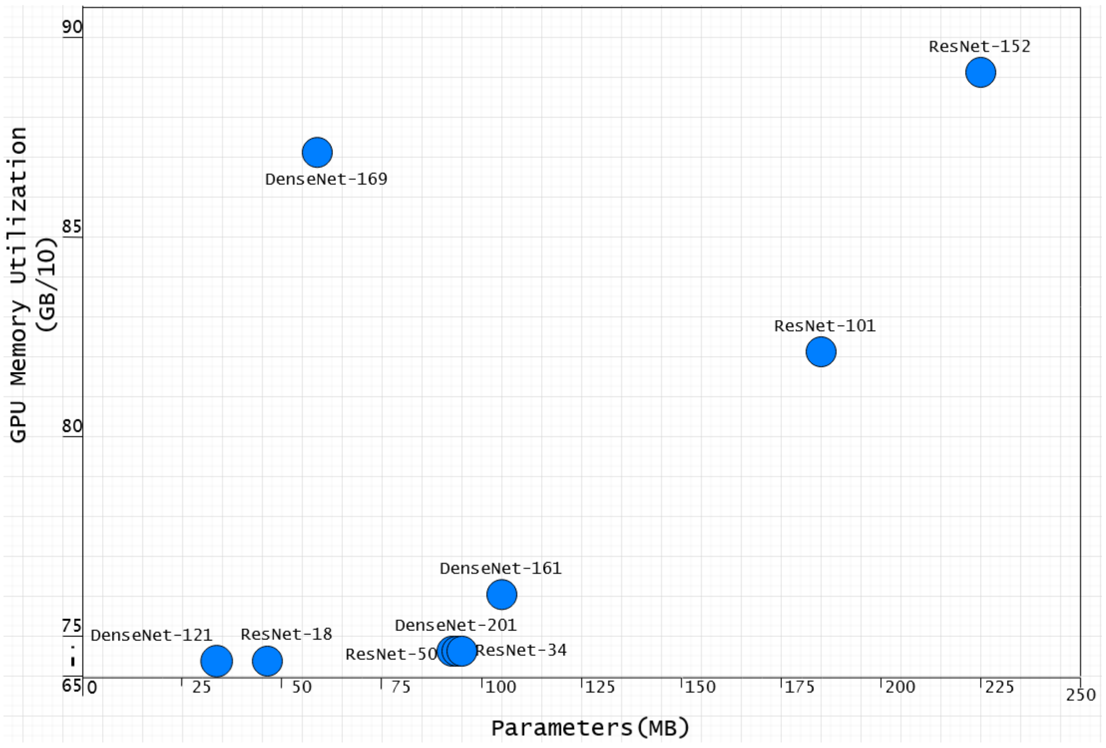


Figure 4.12: Model Complexity vs Memory Utilization(Simplified).

4.3.7 Best Model According to Given Constraints

TITAN XP			
<=0.7GB, @30FPS	**Accuracy**	**<=0.7GB, @60FPS**	**Accuracy**
DPN-68	75.95	DPN-68	75.95
DenseNet-121	**74.47**	**DenseNet-121**	**74.47**
NASNet-A-Mobile	74.10	NASNet-A-Mobile	73.48
BN-Inception	73.48	BN-Inception	71.81
MobileNet-v2	71.81	MobileNet-v2	71.81
<=1.0GB, @30FPS	**Accuracy**	**<=1.0GB, @60FPS**	**Accuracy**
Inception-ResNet-v2	80.28	Se-ResNeXt-50 (32x4d)	79.11
Inception-v4	80.10	**ResNet-152**	**78.25**
DPN-131	79.44	Inception-v3	77.50
DPN-98	79.23	FBResNet-152	77.44
Se-ResNeXt-50 (32x4d)	79.11	ResNet-101	77.31
<=1.4GB, @30FPS	**Accuracy**	**<=1.4GB, @60FPS**	**Accuracy**
NASNet-A-Large	82.50	Se-ResNeXt-50 (32x4d)	79.11
Inception-ResNet-v2	80.28	Xception	78.79
Se-ResNeXt-101 (32x4d)	80.28	SE-ResNet-101	
Inception-v4	80.10	**ResNet-152**	**78.25**
DPN-131	79.44	SE-ResNet-50	77.61

Table 4.7: Top 5 ResNet and DenseNet models satisfying memory consumption and inference speed constraints on the Titan Xp [44].

In the table (4.7) top ResNet and DenseNet models are labelled with red.

With higher FPS and Memory requirement there are one DenseNet model and three ResNet models. There are two places where ResNet-152 model took place.

In given constraints it is safe to say ResNet-152 and DenseNet-121 are the best models in given constraints between ResNet and DenseNet architecture among DNN models.

Chapter 5

Experiments and Results

5.1 Hardware and Environment

Deep Neural Network classification projects are requiring capable CPU or GPU to run on. In order to give a reference to hardware i will be using for practical comparison of ResNet-152 and DenseNet-121 architecture modelsi, a list of PC hardware can be found below.

- CPU: Intel Core i7-9700K 8-Core 4699Mhz
- GPU: Nvidia GeForce RTX 2060 6GB
- RAM: 32GB
- File Storage: 1TB

Nvidia GeForce RTX 2060 is a CUDA supported high-end graphics card. With compatible environment setup this graphics card will serve the purpose of this project well. Usage of CPU is also possible for DNN projects but GPU will work more efficient than CPU due to its neural network capabilities.

For the purpose of implementing models and creating a code that will be used to compile this project, TensorFlow and Keras platforms have used. At the other hand to take full advantage of the GPU and run the project on GPU, there is a requirement of Nvidia CUDA Deep Neural Network library. There is a list below that contains the details of the environment.

- OS: Ubuntu 20.04.1 LTS (Focal Fossa)
- TensorFlow v2.3.0

- Nvidia Driver v450.51.06
- CUDA v11.0
- cuDNN v7.6.5

Building this platform and libraries environment can be tricky and all platform versions must be compatible. Compatibility depends on the hardware and software specifications of the device.

5.1.1 Setting Environment Up

As it has been mentioned at the section above, setting environment and running a model on GPU in the system can be problematic. All drivers, libraries and frameworks must be compatible with each other. GPU hardware generation and CUDA versions must be compatible as well as these versions must be supported by the Tensorflow version we are using. In order the get a better understanding on these aspects, we can always check documentations on these packages. Below you can find current compatibility documentations published.

5.1.1.1 Compatibility

First of all we have to make sure that our GPU driver will work with our operating system. Ubuntu 20.04.1 LTS is an Linux x86-64 operating system and we are using a Nvidia RTX 2060 graphics card which is Turing Generation.

		Driver Version				
Hardware Generation	**Compute Capability**	**384.111+**	**410.48+**	**418.40+**	**440.33+**	**450.36+**
Ampere	8.0	No	No	No	No	Yes
Turing	7.5	No	Yes	Yes	Yes	Yes
Volta	7.x	Yes	Yes	Yes	Yes	Yes
Pascal	6.x	Yes	Yes	Yes	Yes	Yes
Maxwell	5.x	Yes	Yes	Yes	Yes	Yes
Kepler	3.x	Yes	Yes	Yes	Yes	Yes
Fermi	2.x	No	No	No	No	No

Table 5.1: Compute Capability Support - Nvidia.

CUDA Toolkit	**Linux x86_64 Driver Version**
CUDA 11.0 (11.0.171)	>= 450.36.06
CUDA 10.2 (10.2.89)	>= 440.33
CUDA 10.1 (10.1.105)	>= 418.39
CUDA 10.0 (10.0.130)	>= 410.48
CUDA 9.2 (9.2.88)	>= 396.26
CUDA 9.1 (9.1.85)	>= 390.46
CUDA 9.0 (9.0.76)	>= 384.81
CUDA 8.0 (8.0.61 GA2)	>= 375.26
CUDA 8.0 (8.0.44)	>= 367.48
CUDA 7.5 (7.5.16)	>= 352.31
CUDA 7.0 (7.0.28)	>= 346.46

Table 5.2: CUDA Application Compatibility Support Matrix - Nvidia.

As it can be seen on the table (5.1) Turing hardware generation has support for driver versions 384.111 and later.This means that we can use latest driver 450.36.06. On next table (5.2), Nvidia driver 450+ has support for linux x86-64 distros and this combination is only compatible with CUDA version 11.0. This means that this is the configuration we will use at the GPU side.

Now we should check which Tensorflow version we should use with this configuration.

Version	Python Version	Compiler	cuDNN	CUDA
tensorflow-2.2.0	3.5-3.8	GCC 7.3.1	7.6	10.1
tensorflow-2.1.0	2.7, 3.5-3.7	GCC 7.3.1	7.6	10.1
tensorflow-2.0.0	2.7, 3.3-3.7	GCC 7.3.1	7.4	10.0
tensorflow_gpu-1.14.0	2.7, 3.3-3.7	GCC 4.8	7.4	10.0
tensorflow_gpu-1.13.1	2.7, 3.3-3.7	GCC 4.8	7.4	9
tensorflow_gpu-1.12.0	2.7, 3.3-3.6	GCC 4.8	7	9
tensorflow_gpu-1.11.0	2.7, 3.3-3.6	GCC 4.8	7	9
tensorflow_gpu-1.10.0	2.7, 3.3-3.6	GCC 4.8	7	9
tensorflow_gpu-1.9.0	2.7, 3.3-3.6	GCC 4.8	7	9
tensorflow_gpu-1.8.0	2.7, 3.3-3.6	GCC 4.8	7	9
tensorflow_gpu-1.7.0	2.7, 3.3-3.6	GCC 4.8	7	9
tensorflow_gpu-1.6.0	2.7, 3.3-3.6	GCC 4.8	7	9
tensorflow_gpu-1.5.0	2.7, 3.3-3.6	GCC 4.8	7	8
tensorflow_gpu-1.4.0	2.7, 3.3-3.6	GCC 4.8	6	8
tensorflow_gpu-1.3.0	2.7, 3.3-3.6	GCC 4.8	6	8
tensorflow_gpu-1.2.0	2.7, 3.3-3.6	GCC 4.8	5.1	8
tensorflow_gpu-1.1.0	2.7, 3.3-3.6	GCC 4.8	5.1	8
tensorflow_gpu-1.0.0	2.7, 3.3-3.6	GCC 4.8	5.1	8

Table 5.3: Tensorflow version compatibility table - Tensorflow.

On the table above, latest version of Tensorflow is 2.2.0 and it is compatible with cuDNN 7.6 and CUDA 10.1. While this experiment on-going we found out that there is a new version release for Tensorflow which is 2.3.0. In order to be sure we tried both tensorflow versions and has been able to take advantage of GPU with Tensorflow 2.3.0. At the other hand we updated our cuDNN version to 7.6.5 for compatibility.

5.1.1.2 Installation

For this experiment we will use Python and rather than installing Tensorflow for our whole operting system, creating a Python virtual environment is a healthier option. To do this we need Python 3.4 and PIP. Usually latest distros of Ubuntu comes with PIP and we can directly step to creating virtual environment. We will run codes below in our Ubuntu Terminal.

```
#Installing python3-venv
apt-get install python3-venv -y

#Create and activate environment
python3 -m venv "environment_name"
source ./venv/bin/activate
workon environment_name

#Shell prompt should look something like this
(environment_name) root@ubuntu:~#

#Update PIP
(environment_name) root@ubuntu:~# pip install -U pip

#Update setuptools
(environment_name) root@ubuntu:~# pip install -U setuptools
```

Now we are ready to install Tensorflow and the code below will install latest version of Tensorflow to our python evironment.

```
pip install tensorflow
#You may require different versions of Tensorflow, so simply
add the version to the end of the command.(ex: tensorflow==1.15)
```

After this step we have to install Nvidia packages and drivers. The code below can be followed and run at the terminal.

```
# Adding NVIDIA package repositories
wget https://developer.download.nvidia.com/compute/cuda/repos/ \
        ubuntu1804/ x86_64/cuda-repo-ubuntu1804_10.1.243-1 \
        _amd64.deb
sudo apt-key adv --fetch-keys https://developer.download. \
        nvidia.com/compute/cuda/repos/ubuntu1804/ \
        x86_64/7fa2af80.pub
sudo dpkg -i cuda-repo-ubuntu1804_10.1.243-1_amd64.deb
sudo apt-get update
wget http://developer.download.nvidia.com/compute/ \
        machine-learning/repos/ubuntu1804/x86_64/ \
        nvidia-machine-learning \
        -repo-ubuntu1804_1.0.0-1_amd64.deb
sudo apt install ./nvidia-machine-learning-repo- \
        ubuntu1804_1.0.0-1_amd64.deb
sudo apt-get update

# Installation of NVIDIA driver
sudo apt-get install --no-install-recommends nvidia-driver-450
# A reboot is required after this command.

# Installing development and runtime libraries
sudo apt-get install --no-install-recommends \
    cuda-10-1 \
    libcudnn7=7.6.5.32-1+cuda11.0  \
    libcudnn7-dev=7.6.5.32-1+cuda11.0



# Installing TensorRT. Before that libcudnn7 must be installed.
sudo apt-get install -y
        --no-install-recommends libnvinfer6=6.0.1-1+cuda11.0
    libnvinfer-dev=6.0.1-1+cuda11.0 \
    libnvinfer-plugin6=6.0.1-1+cuda11.0
```

Before going further we shoud check if the installed versions are the same with the configuration we planned. We will use terminal again to check installed versions.

```
#Check Nvidia driver version
nvidia-smi

#Check CUDA version
nvcc --version

#Check cuDNN version
cat /usr/include/cudnn.h | grep CUDNN_MAJOR -A 2

#Check Tensorflow version
pip show tensorflow
```

If everything checks out that means our environment installation is done and we can further our experiment with preparing dataset and running our code for model training.

5.2 Dataset

This projects purpose is to compare the efficiency of DNN architecture models' rather than testing these models. Because of this the dataset we are gonna use is an average sized dataset when compared to datasets that are used generally. Dataset contains images which are extracted from satellite images. In total of 9040 images, 6780 is used for training dataset, 2160 for validation dataset and 100 images for testing dataset.

5.2.1 Dataset Preparation

This experiment requires satellite images, Landsat 8 to be exact. To get these images with their raw image band data, we can use United States Geological Survey(USGS) tools. USGS has a website where users can sign up by giving personal information and information like which area and how they will use data acquired. (`https://earthexplorer.usgs.gov/`)

By using USGS Earth Explorer tool we can select the area, satellite and type of the satellite dataset in their database. After we found the satellite image we want, it is possible to download GeoTIFF data product. (Usually this file has a size vary between 0.8-1.5GB) Since we have the satellite image data, we can divide our image to pieces and create a dataset that consist images that has shoreline and does not consist. In order to achieve this and work on this huge sized image data we are going to use QGIS software. QGIS software has tools which makes it possible to work on such raster images and make manipulations we require. An example to this process found at the images below. (5.1,5.2)

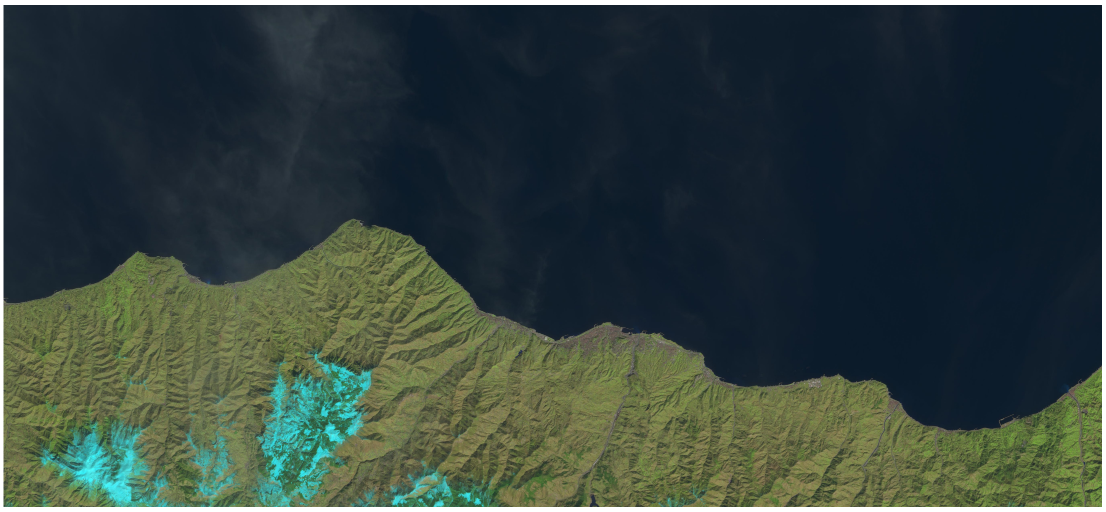


Figure 5.1: Example for satellite image.

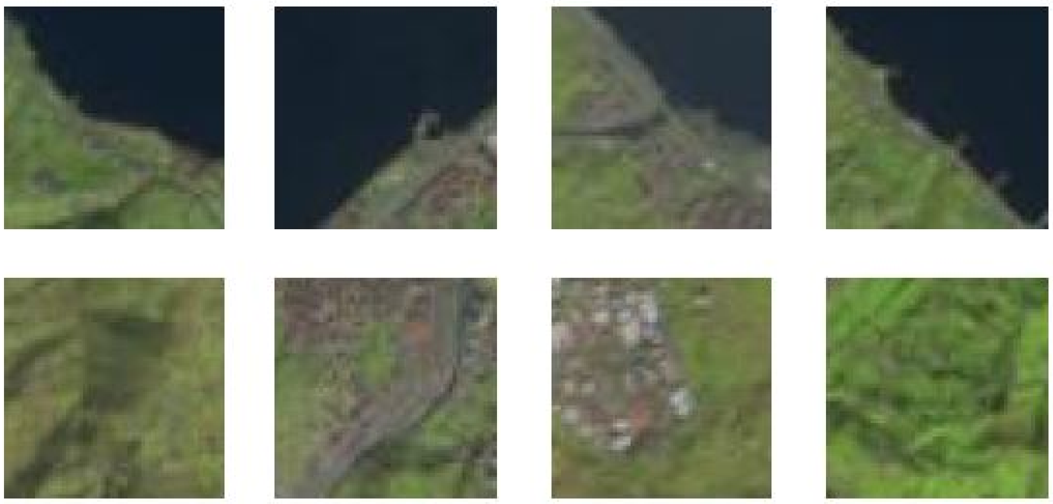


Figure 5.2: Images with and without a shoreline, exracted from satellite images.

We divided whole satellite image data to images which has 100x100. After this process we will have a dataset that consist high quantity of images. In order to increase data quantity in our dataset we can rotate all image by 90, 180 and 270 degrees. This will multiply the dataset we have by four. There are several ways we can conduct this process, in this process we prepared a python code. This code takes a folder of images, reads all image inside, rotates and finally saves images to a folder specified. An example for this can be found below.

```
rot_degree = "90"
int_rot_degree = int(a)

def main():
    outPath = 'path_to_folder'+rot_degree
    path = 'path_to_folder'

    # iterate through the names of contents of the folder
    for image_path in os.listdir(path):

        # create the full input path and read the file
        input_path = os.path.join(path, image_path)
        image_to_rotate = mpimg.imread(input_path)

        # rotate the image
        rotated = ndimage.rotate(image_to_rotate, int_rot_degree)

        # create full output path, 'example.jpg'
        # becomes 'rotate_example.jpg', save the file to disk
        fullpath = os.path.join
        (outPath, 'rotated_'+rot_degree+image_path)
        mpimg.imsave(fullpath, rotated)

if __name__ == '__main__':
    main()
```

For training purposes we divide our dataset to training dataset and validation dataset. Also we are going to exract some images for test dataset and our model will not process these images until testing phase. After this we will have our dataset at ready and continue with code implementation for creating our training model.

5.3 Model Creation

In this section there will be an explanation of the python code we use. First of all consideration of dataset augmentation, epoch count and batch size are important for comparing two models. These values must be equal for two models to make a healthy comparison. In our dataset we have 100x100 pixels size images, these images are manipulated to 50x50 size while preparing dataset for feeding to model. 50x50 pixels is fairly big for DNN.

One epoch means that our dataset will be passed across our DNN model. Passing whole dataset for one time is not effective. Epoch count will be 50 for our comparison run for each model.

Since we cannot feed complete dataset to our model at once, we have to divide our dataset in to batches. Batch size can be various. If batch size is low, model can be suffered underfitting and if batch size is too high there can be overfitting. Batch size 64 is decided to give an optimal batch size to the model.

- Image Size: 50x50
- Epoch Count: 50
- Batch Size: 64

For these comparisons we are going to use code below, here we are determining image width and height we want our data's size while it is being loaded, path to our dataset folders(training, validation, test), epoch count and batch size.

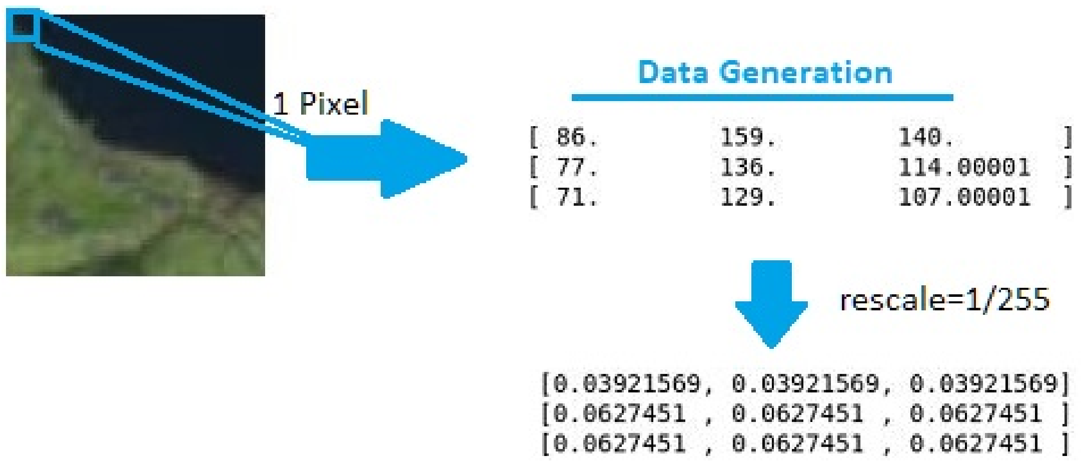
```
img_width, img_height = 50,50 #size for input image
input_depth = 3 #3: rgb image

#training dataset
train_data_dir = os.path.expanduser('path_to_foldern')
#validation dataset
validation_data_dir = os.path.expanduser('path_to_folder')
#testing dataset
test_data_dir = os.path.expanduser('path_to_folder')

epochs = 50
batch_size = 64
```

After these denifinitons we will define an image generator for Keras. RGB image data consists three variables. These variables has a value between 0 and 255. Each value represents weights of colors red,green and blue.(Ex: [253,142,043]). Our loaded image data will have matrices that consists these RGB values. We will use ImageDataGenerator function from Keras/Tensorflow to create these matrices from image data. This function will extract RBG values pixel by pixel. We will normalize these values to have an intensity between 0-1. ImageDataGenerator function can do this process by rescale input.(5.3) Code below shows this process.

```
train_datagen = ImageDataGenerator(rescale=1/255)
validation_datagen = ImageDataGenerator(rescale=1/255)
test_datagen = ImageDataGenerator(rescale=1/255)
```

From exracted and normalized data we will create our dataset. In this step we will flow our generated data to an object which will hold the data for our dataset. "Flow from directory" function from Keras/Tensorflow will handle this process and we will determine some values for further configuration. These values are path to dataset folder, color mode of the data, targeted size, batch size, will image be shuffled or not and class mode that will determine how we divided classes we have in our dataset.

```
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    color_mode='rgb',
    target_size=(img_width,img_height),
    batch_size=batch_size,
    class_mode='categorical')
validation_generator = validation_datagen.flow_from_directory(
    validation_data_dir,
    color_mode='rgb',
    target_size=(img_width,img_height),
    batch_size=batch_size,
    class_mode='categorical')
test_generator = test_datagen.flow_from_directory(
    test_data_dir,
    color_mode='rgb',
    target_size=(img_width,img_height),
    batch_size=1,
    class_mode='categorical',
    shuffle=False)
```



```
[ 86.         159.         140.        ]
[ 77.         136.         114.00001  ]
[ 71.         129.         107.00001  ]
```

```
[0.03921569, 0.03921569, 0.03921569]
[0.0627451 , 0.0627451 , 0.0627451 ]
[0.0627451 , 0.0627451 , 0.0627451 ]
```

Figure 5.3: Data generation process.

Since we have our dataset configured and loaded we can define our network model. This part can be done by defining each layer in our model but in this experiment we will use Keras/Tensorflow application. This application has predefined network models which can be called by one function. This function uses "ImageNet" weight by default and we will specify that there will be no weights. Also we will give a class attribute to tell our model how many classes that our model will work on. In our case we have 2 classes. Data that has shore and not. Code in order to call DenseNet-121 model:

```
model = tf.keras.applications.DenseNet121(
    weights=None,
     classes=2
)
```

Code in order to call ResNet-152 model:

```
model = tf.keras.applications.ResNet152(
    weights=None,
     classes=2
)
```

All models must be complied after definition. Compile function for network model takes variable inputs. For our experiment we will use loss function, optimizer algorithm and metrics. Loss function will determine how we calculate models error at each output layer. Since our experiment is on performance of the models we can decide on any loss function and optimizer algorithm since it will not effect the results for comparison. Deciding which loss function and optimizer to use to train our model in a better way is in the aspects of future work of this research. To compile our model we will use code below.

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])
```

With summary function we can see the summary of our model. This summary will show every layer in our models. Because of these models are DNN models and have too many layers, showing every layer in our DenseNet-121 and ResNet-152 will take 60 pages combined. These summary tables will not be shared in this document but can be found online.

```
model.summary()
```

5.3.1 Training the Model

Now our model is ready and complied. We can start training our model by fitting. Fitting function will take our configuration and dataset informations and start training our model by epoch count with our batch size. Traning can be started by the code given below.

```
model.fit(
    train_generator,

    steps_per_epoch=np.floor(train_generator.n/batch_size),
    epochs=epochs,

    validation_data=validation_generator,
    validation_steps=np.floor(validation_generator.n / batch_size)
```

Terminal will give an output that consist the current situation of our training process for each epoch and terminal will do this for each epoch until it is done processing the epoch count we have decided. Example of the output is given below.

```
.
.
.
Epoch 10/50
50/50 [==============================] - 2s 50ms/step -
loss: 0.2220 - accuracy: 0.9147
- val_loss: 1.0379 - val_accuracy: 0.7474
Epoch 11/50
50/50 [==============================] - 2s 49ms/step -
loss: 0.2024 - accuracy: 0.9269
- val_loss: 0.8641 - val_accuracy: 0.6849
Epoch 12/50
50/50 [==============================] - 2s 49ms/step -
 loss: 0.1811 - accuracy: 0.9262
- val_loss: 0.7022 - val_accuracy: 0.7960
```

```
Epoch 13/50
50/50 [==============================] - 2s 50ms/step -
loss: 0.1725 - accuracy: 0.9366
- val_loss: 0.7698 - val_accuracy: 0.7222
Epoch 14/50
50/50 [==============================] - 2s 50ms/step -
loss: 0.1693 - accuracy: 0.9378
- val_loss: 0.6771 - val_accuracy: 0.7891
Epoch 15/50
50/50 [==============================] - 3s 50ms/step -
loss: 0.1514 - accuracy: 0.9400
- val_loss: 0.3865 - val_accuracy: 0.8689
Epoch 16/50
.
.
.
```

After our training is done and we achieved targeted training results we can save our model to load it whenever need, we can use load and save functions. This way we will not need to train our model again every time we need it.

```
model.save('path_to_save/model_name.h5')

model.load('model_name.h5')
```

5.3.2 Testing the Model

Since we have trained and prepared a test dataset, we can try to run a testing on our model with our dataset. In this testing dataset there are 100 images which our model has never seen before. This will give an idea for our future works on this project. First we will open a file for writing and create probabilities object. With predict generator function we will fill our probabilities object, this object will consist two element arrays.

```
open("file_name.cvs","w")
probabilities = model.predict_generator(test_generator)
```

Now for each image in our testing dataset we will create prediction values and we can read all images in our dataset, write out predictions and plot image itself under them with code given.

```
for index, probability in enumerate(probabilities):
    image_path = test_data_dir + "/"
                     +test_generator.filenames[index]
    img = mpimg.imread(image_path)
    print(test_generator.filenames[index])
    with open("file_name.cvs","a") as fh:
        fh.write(
                str(probability[0]) + " for: " + image_path + "\n"
                )
    plt.imshow(img)
```

Finally we can print our plots and results by a condition. This condition will take prediction from probabilities array and check if first element in array is bigger than 0.5 or not. If value is bigger than 0.5 this means image is predicted by our model as it does not consist a shore. If not image is predicted as it consist a shore. Example prints and code can be seen below.

```
if probability[0] > 0.5:
        plt.title("%.2f" % (probability[0]*100) + "% notshore")
    else:
        plt.title("%.2f" % ((1-probability[0])*100) + "% shore")
    plt.show()
```

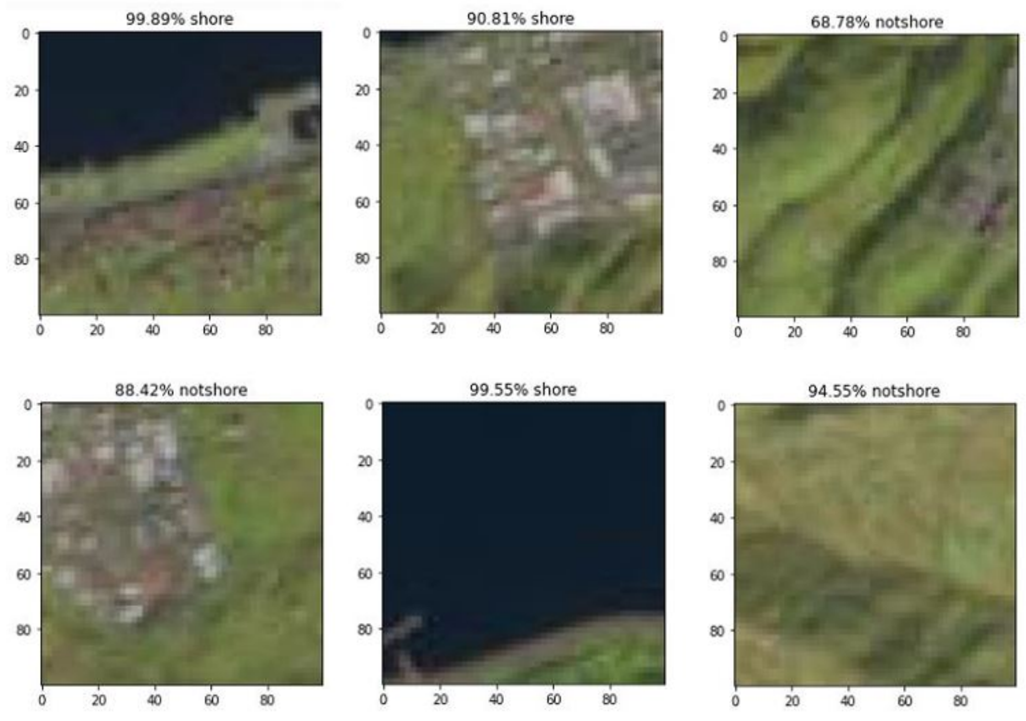


Figure 5.4: Example test results for ResNet-152 model(predictions above images).

5.4 Results and Comparison

For comparison memory utilization of GPU versus parameter count, training time, accuracy rate, loss rate and accuracy on testing dataset will be used.

5.4.1 GPU Memory Utilization vs Parameter Count

Comparing Parameter Count with GPU memory utilization show how efficient our model in terms of memory for each parameter. If model requires less memory for each parameter this means that the model is running more efficiently. As it can be seen on the table below ResNet-152 requires far less memory for its parameters than DenseNet-121.

Model	**GPU Mem. Util.**	**Parameter(millions)**	**Memory/Parameter**
ResNet-152	5206MB	∼ 58,3M	∼89,2MB
DenseNet-121	3683MB	∼ 7,03M	∼523.8MB

Table 5.4: Memory Utilization of GPU, Total Parameters and Memory divided by Parameter(per million) for each model.

5.4.2 Training Time

If model can train in a shorter time, this means that model will be more efficient at more training iterations. As it can be seen on the table below DenseNet-121 model is more in the aspect of time.

Model	**Training Time**
ResNet-152	∼530 seconds
DenseNet-121	∼250 seconds

Table 5.5: Training Time for each model.

5.4.3 Accuracy Rate

Model accuracy means how well our model is trained when it is validated with validation dataset. As it can be seen on the table below two models are close but ResNet-152 is training more accurately but DenseNet-121 model is giving very similar accuracy results. We can train each model with similar accuracy rates.

Model	**Accuracy**
ResNet-152	∼0.8733
DenseNet-121	∼0.8464

Table 5.6: Accuracy Rate for each model.

5.4.4 Loss Rate

Lower loss rate means that model has less error and is training more efficiently. As it can be seen on the table below ResNet-152 model can train more efficiently.

Model	**Loss**
ResNet-152	~0.3664
DenseNet-121	~0.8209

Table 5.7: Loss Rate for each model.

5.4.5 Accuracy on Testing Dataset.

When our models' predictions are checked and mistakes counted, it will give a result for actual testing. This result depends on the dataset ofcourse but result of this section will give us an idea which model was better at predicting on actual testing dataset. On the table below we can see that ResNet-152 has given better predictions. Accuracy rate between these models can be accepted as close or even similar.

Model	**Mistaken/Total Images.**
ResNet-152	4 / 100
DenseNet-121	6 / 100

Table 5.8: Accuracy on Testing Dataset.

5.4.6 Overall Comparison

Between our comparison terms ResNet152 architecture model is showing better efficiency in the area of memory consumption, accuracy, error rate(loss) and accuracy on testing dataset while DenseNet-121 has a better training time. Since ResNet-152 model has better results in four ares out of five, it is safe to say that ResNet-152 architecture model is a more efficient model.

Model	**Mem/Param**	**Train Time**	**Accuracy**	**Loss**	**Test Acc.**
ResNet-152	X		X	X	X
DenseNet-121		X	X		X

Table 5.9: Overall Comparison of each model.

Chapter 6

Conclusion

This research work is trying to answer the question of "Between ResNet and DenseNet architectures, which architecture and model is more efficient to use while training networks on satellite image datasets for shoreline detection?"

Previous researchs shows that if GPU Memory is low on the computer that model is running, especially if GPU memory is lower than or equal to 700MB it is best to use DenseNet Architecture. Precisely **DenseNet-121** model of DenseNet Architecture will act more efficiently than other architecture models. Between 700 and 1000MB of GPU Memory **ResNet-101** and **ResNet-152** models can be used for more efficient training environment. These models both are showing a closer efficiency. For higher GPU Memory capabilities between 1000 and 1400MB it is best to use **ResNet-152**.

But since current graphics cards have high memory than 1.4GB, our experiment is showing a more percise result. If GPU Memory is high and a shorter training time is required DenseNet-121 can give satisfactory results. In addition to that shoreline satellite image has bigger data size than standard image data. Due to its data character satellite datasets tend to consist huge datasets. This situation causes training a network with satellite images to require more memory and DenseNet architecture would require greater memory sizes. ResNet architecture would provide more efficient memory performance for large datasets.

Overall performance indices' results shows that **ResNet Architecture** shown more efficiency and performance.

This work is providing a solution for choosing right architecture model for working with satellite images by giving the answer of **“ResNet Architecturse are more efficient for large satellite datasets with its lower memory requirement for each parameter. Especially ResNet-152 architecture model.”**

Chapter 7

Future Works

This experimental research on ResNet and DenseNet architectures gives an idea for which architecture to use while training a model to extract shorelines from satellite images. As the conclusion of the research says architecture type to use for best performance depends on the hardware configuration.

In the future, as a part of TÜBİTAK project of ”Uydu görüntülerinden Kıyı Sınırlarının Derin Öğrenme Yöntemleri ile Otomatik Çıkarımı” satellite images of selected coastal areas will be used for training.

Given list shown next phase of this research in titles below:

- Preparing and producing training/test datasets from Sentinal 2-A Satellite images (Locations may vary)

- Testing performance of ResNet models and optimization of the model. Optimization process may consist altering the standard model. (Adding, altering and optimizing layers etc.)

- Acquiring and improving accuracy rate to a satisfactory point

References

[1] S. Manchala, "Remote Sensing", M.S. thesis, Computer Science, San Diego State University, San Diego, USA, 2017.

[2] C. Chen, B. Zhang, H. Su, W. Li, and L. Wang, “Land-use scene classification using multi-scale completed local binary patterns,” *SIViP*, vol. 7, no. 10, pp. 745–752, 2014.

[3] J. Li, P. Gamba, and A. Plaza, “A novel semi-supervised method for obtaining finer resolution urban extents exploiting coarser resolution maps,” *IEEE J. Sel. Top. Appl. Earth Obser. Remote Sens.*, vol. 7, no. 10, pp. 4276–4287, 2014.

[4] W. Li, C. Chen, H. Su, and Q. Du, “Local binary patterns and extreme learning machine for hyperspectral imagery classification,” *IEEE Trans. Geosci. Remote Sens.*, vol. 53, no. 7, pp. 3681–3693, 2017.

[5] G. Vosselman, M. Coenen, and F. Rottensteiner, “Contextual segment-based classification of airborne laser scanner data,” *ISPRS J. Photogramm. Remote Sens.*, no. 128, pp. 354–371, 2017.

[6] Zhang and Baochang, “One-Two-One Networks For Compression Artifacts Reduction In Remote Sensing,” *ISPRS Journal Of Photogrammetry And Remote Sensing*, no. 145, pp. 184–196, 2017.

[7] “A comprehensive guide to convolutional neural networks — the eli5 way,” https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53/, accessed: 2019-11-16.

[8] E. Fırat, "LANDSAT-8 Uydu Görüntülerinden Derin Öğrenme Algoritmaları Kullanılarak Kıyı Çizgisi Çıkarımı", M.S. thesis, Fen Bilimleri Enstitüsü, T.C. Yıldız Teknik Üniversitesi, İstanbul, TURKEY, 2018.

[9] S. B. El Kafrawy, M. E. Basiouny, E. A. Ghanem, and A. S. Taha, “Performance Evaluation of Shoreline Extraction Methods Based on Remote Sensing Data,” *Journal of Geography*, vol. 11, no. 4, pp. 1–18, 2017.

[10] L. I. Bendell and P. C. Wan, “Application of aerial photography in combination with GIS for coastal management at small spatial scales: a case study of shellfish aquaculture,” *Journal of Coastal Conservation*, vol. 15, no. 4, pp. 417–431, 2011.

[11] P. F. Sale, I. M. J. Butler, A. J. Hooten, K. P. Kritzer, K. C. Lindeman, Y. J. S. de Mitcheson, R. S. Steneck, and H. van Lavieren, “Stemming Decline of the Coastal Ocean: Rethinking Environmental Management,” *UNU-INWEH*, 2008.

[12] K. Zhang, B. C. Douglas, and S. P. Leatherman, “Global warming and coastal erosion,” *Climatic Change*, vol. 64, no. 1, pp. 41–58, 2004.

[13] J. T.D. and R. J.E., “Biogeochemistry of intertidal sediments,” *Cambridge Environmental Chemistry Series*, 1997.

[14] R. Li, K. Di, and R. Ma, “A Comparative Study of Shoreline Mapping Techniques,” *4th International Symposium on Computer Mapping and GIS for Coastal Zone Management*, 2001.

[15] T. Lillesand, R. W. Kiefer, and J. Chipman, *Remote Sensing and Image Interpretation.*, Fifth Edition, John Wiley and Sons, New Jersey, 2014.

[16] M. Baatz, U. Benz, S. Dehghani, M. Heynen, A. Höltje, P. Hofmann, I. Lingenfelder, M. M. M. Sohlbach, M. Weber, and G. Willhauck, “eCognition Professional User Guide 4,” *Defines Imaging*, 2000.

[17] R. A. Schowengerdt, *Remote Sensing: Models and Methods for Image Processing.*, Third Edition, Elsevier, 2006.

[18] R. Gens, “Remote sensing: models and methods for image processing,” *International Journal of Remote Sensing*, vol. 31, no. 7, pp. 1819–1836, 2010.

[19] T. Kutser, B. C. Paavel, C. Verpoorter, T. Kauer, and E. Vahtmäe, “Remote sensing of water quality in optically complex lakes,” *International Archives of the Photogrammetry*, no. XXXIX-B8, pp. 165–169, 2012.

[20] “Landsat 8 bands and combinations,” https://gisgeography.com/landsat-8-bands-combinations/, accessed: 2019-11-21.

[21] “Sentinel 2 bands and combinations,” https://gisgeography.com/sentinel-2-bands-combinations/, accessed: 2019-10-02.

[22] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations,” *ACM*, vol. 54, no. 10, 2009.

[23] İ.N.İ.K Özkan and E. Ülker, “Derin Öğrenme Ve Görüntü Analizinde Kullanılan Derin Öğrenme Modelleri,” *Gaziosmanpaşa Bilimsel Araştırma Dergisi*, vol. 3, no. 6, pp. 85–104, 2017.

[24] J. Brownlee, “A gentle introduction to pooling layers for convolutional neural networks,” https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/, accessed: 2019-09-08.

[25] “Fully connected layers in convolutional neural networks: The complete guide - missinglink.ai,” https://missinglink.ai/guides/convolutional-neural-networks/fully-connected-layers-convolutional-neural-networks-complete-guide/, accessed: 2019-10-11.

[26] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *Computer Science*, vol. 4, no. 3, pp. 212–223, 2012.

[27] S. Ding and G. Wang, “Research On Intrusion Detection Technology Based On Deep Learning,” *IOP Conference Series: Materials Science and Engineering*, vol. 646, 2019.

[28] N. Srivastava, “Dropout: A Simple Way To Prevent Neural Networks From Overfitting,” *Journal Of Machine Learning Research*, no. 15, 2014.

[29] A. Khan, “Survey Of The Recent Architectures Of Deep Convolutional Neural Networks,” *Artificial Intelligence Review*, arXiv: 1901.06032.

[30] Y. LeCun, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *NIPS*, pp. 1106–1114, 2012.

[32] K. Simonyan and A. Zisserman, “Very deep convolutional networks for largescale image recognition,” 2014, arXiv:1409.1556.

[33] M. Lin, Q. Chen, and S. Yan., "Network in network," 2013, arXiv:1312.4400.

[34] J. T. Springenberg, "Striving for simplicity: The all convolutional net," 2014, arXiv:1412.6806.

[35] H. Kaiming, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[36] G. Huang, Z. Liu, K. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," 2016, arXiv:1608.06993.

[37] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, arXiv:1502.03167.

[38] G. Larsson, M. Maire, and G. Shakhnarovich, "FractalNet: Ultra-Deep Neural Networks without Residuals," 2016, arXiv:1605.07648.

[39] Z. Alom, "Improved Deep Convolutional Neural Networks (DCNN) Approaches for Computer Vision and Bio-Medical Imaging," Ph.D. dissertation, Dept. Electrical and Computer Engineering, University of Dayton, Dayton, USA, 2018.

[40] F. Shahbaz, "Five powerful cnn architectures," https://medium.com/datadriveninvestor/five-powerful-cnn-architectures-b939c9ddd57b, accessed: 2019-09-07.

[41] C. Szegedy, "Rethinking the inception architecture for computer vision," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[42] S.-H. Tsang, "Review: Resnet — winner of ilsvrc 2015 (image classification, localization, detection)," https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5d8, accessed: 2019-09-07.

[43] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme Learning machine for regression and multiclass classification," *IEEE Trans. Syst. Man Cybern. Part B Cybern*, vol. 42, no. 2, pp. 513–529, 2012.

[44] S. Bianco, "Benchmark Analysis Of Representative Deep Neural Network Architectures," *IEEE Access*, vol. 6, pp. 64 270–64 277, 2018.