Academic year 2019-2020

University of Piraeus

Department of Digital Systems

Postgraduate Programme in Big Data & Analytics

Medical Image Analysis With Machine Learning Techniques And Deep Learning Neural Networks

MSc Thesis

Author:
George Giannoulopoulos
Registration Number: ME1708

Supervisor:
Prof. Ilias Maglogiannis

Abstract

Skin cancer and especially melanoma, a type of skin cancer, is one of the most dangerous types of cancer. Globally, in 2012, it newly occurred in 232,000 people. In 2015 there were 3.1 million with active disease which resulted in 59,800 deaths. Although melanoma is a lethal disease, early detection can decrease its lethality percentage by a lot. Thus, a reliable automated detection application can prove invaluable in a diagnostic center in order to enhance the diagnostic ability of a dermatologist. Following the explosion of computational resources in recent years, various machine learning techniques have been developed in the computer vision field, techniques that can be used for medical image classification.

In this master thesis, the recent development of computer vision machine learning algorithms will be examined and the performance of a subset of those will be studied on the skin lesion image classification task. More specifically, the performance of deep learning algorithms on image classification will be presented and tested. A specific family of convolutional neural networks called EfficientNets will be trained on the ISIC challenge dataset of 2019, which contains 25,331 dermoscopy images of skin lesion images that belong to 8 classes. The performance of the resulting trained neural networks will be then examined on predicting the correct class of skin lesion dermoscopy images.

Finally, an application based on the Flask python mini framework, a model view controller framework, will be developed as a proof of concept of a MVP (Minimum viable product) that can be used in a production environment of a diagnostic center. The application will use the best previously trained efficientNet model and diagnose the uploaded by the user images.

Keywords

Deep Learning, Neural Networks, EfficientNet, Skin lesion classification, Machine Learning, Image classification, Medical imaging, Skin cancer, Melanoma classification, Dermoscopy

Acknowledgements

I would like to thank Prof. Ilias Maglogiannis of the Dept. of Digital Systems of University of Piraeus for supervising my research and providing the guidance and resources needed for completing this thesis. I would also like to thank Sr Teaching Fellow Konstantinos Moutselos for his help and insights, as well as, all the Big data & Analytics teaching faculty of the Digital Systems Department. Last but not least, I want to thank my family and friends for their love and support.

Contents

List of Figures

List of Abbreviations

BCC	Basal cell carcinoma
SCC	Squamous cell carcinoma
AI	Artificial Intelligence
ML	Machine Learning
SVM	Support vector machines
SVR	Support Vector Regressor
ANN	Artificial Neural Networks
CV	Computer vision
HOG	Histogram of oriented gradients
CNN	Convolutional neural network
FCN	Fully Connected Neural Network
MNIST	Modified National Institute of Standards and Technology
SGD	Stochastic Gradient Descent
ISIC	International Skin Imaging Collaboration
FLOPS	Floating point operations per second
NAS	Neural architecture search
CPU	Central processing unit
GPU	Graphics processing unit
TPU	Tensor processing unit
DF	Dermatofibroma
REST	Representational state transfer
API	Application programming interface
MVC	Model View Controller
HTTP	Hypertext Transfer Protocol

Chapter 1

Introduction

1.1. Rationale

The ability to record massive amounts of data in the healthcare sector has allowed health practitioners to access enormous volumes of information about individual patients but the sheer size of the data forbids a human to make sense of it. As a response to that challenge, machine learning techniques are being used to detect patterns, in all that raw data and produce insights [40]. The applications of machine learning algorithms are spread across most of the healthcare fields and while software recommendations and alerts have been around for some time the differences are critical. So far, the recommendation algorithm was hard coded into the software and had a rigid nature, the decision criteria most of the time were based on an external research that could not be representative of the actual use cases as environment and populations could differ. On the other hand machine learning algorithms can be refined by the anonymised data of the health facility that uses the recommendation module, providing an increased accuracy on the produced diagnosis [41]. The tasks that machine learning is applied are numerous and versatile. Identifying potential drug combinations, assisting the health practitioners diagnosis using the available symptoms data or classifying medical images to the correct disease classes are some of the challenges machine learning is tasked to conquer.

The development of information systems for the automated diagnosis of medical images constitutes a field of ever-growing scientific research in the last decade. Digital medical images are present in the majority of diagnostic labs, providing an easy way of manipulation through various information systems. The digital processing of medical images by multiple feature extraction techniques can lead to the accumulation of a number of features in a way that is reliable and replicable. The analysis of biomedical images through the values of extracted features, is a process that can be carried out by machine learning algorithms (through the use of

various classifiers) and ultimately augment the decision making of health practitioners by providing automated diagnosis. The information gain of such systems is huge, as it enhances the timely and reliable identification of important patient cases. Systems like that, can be incorporated in local information systems at diagnostic centers but can also be a part of a telehealth information system.

One field of medicine that relies heavily on digital medical biomedical images in order to produce a diagnosis, is dermatology. In dermatology the correct classification of skin lesions through medical images can greatly affect the well being of the patient. For this reason, the performance of machine learning classifiers will be examined on the task of diagnosing dermatoscopic images and in particular the performance of a category of classifiers called neural networks. Targeted architectures of neural networks that handle image class prediction (Convolutional neural networks) will be presented and tested on the same problem in order to measure their performance. After the development of the deep learning convolutional neural networks, their results on a minimum viable product application will be examined as to see whether or not they are able to be incorporated in a diagnostic center even if its in a simulation environment.

1.2. Structure of the thesis

To begin with, the problem that this thesis is focused at is presented in chapter two. Various details along with the importance of finding a solution are highlighted in order to motivate the reader. To continue with, because the analysis of medical images through machine learning techniques is a combination of multiple computer science fields, chapter three contains all the relevant background theory that is needed in order to understand the various algorithm approaches. In the beginning a brief introduction to the field of machine learning in general is presented and then a closer look on a particular set of algorithms called neural networks. Next, the field of computer vision is examined and the chapter ends with an introduction to convolutional neural networks, a subset of neural networks that are focused on computer vision.

In chapter four, a more detailed presentation of the convolution network architectures that are going to be used is performed by studying a few predominant algorithms and their

applications. Chapter five contains the experimental setup and results of the selected approaches and in chapter six the conclusion to those results is presented. Lastly, the bibliography that was used during the authoring of this thesis is displayed.

Chapter 2

Skin Lesion Analysis Towards Melanoma Detection

2.1. The Problem

Skin lesions are a common disorder of the body's biggest organ, the skin. The lesions most of the time are harmless just constituting a visual defect not requiring treatment but there are cases that, left untreated, can be fatal. The identification and classification so far is done by dermatologists, where by evaluating a number of characteristics, diagnose the lesion. The diagnosis most of the time is a simple procedure for a dermatologist, considering the amount of cases he has diagnosed before. Nevertheless, there are patients who display skin lesions that are extremely hard to diagnose and need to undertake a biopsy. The correct diagnosis is of utmost importance as if a potentially fatal lesion is falsely diagnosed then precious time is wasted and the percent of successful treatment reduces dramatically.

Among the different skin lesions types, skin cancer lesions are cases that require additional effort in order to diagnose and treat. Skin cancer is the most common occurring malignancy in the general population on a global scope and the majority of the skin cancer cases are basal cell carcinoma (BCC), Squamous cell carcinoma (SCC) and Melanoma. More than 5.4 million cases of nonmelanoma skin cancer were treated in over 3.3 million people in the U.S. in 2012 [1]. The most common form of skin cancer is BCC, an estimated 4.3 million cases diagnosed in the U.S. each year, while SCC appears in over one million cases in the U.S. each year and results in 15,000 deaths. Melanoma on the other hand is a much rarer occurrence but far more lethal than the other types of skin cancer. An estimated 192,310 cases of melanoma will be

diagnosed in the U.S. in 2019, with 7,230 of those proving fatal. Of those, 4,740 will be men and 2,490 will be women. Although the mortality is significant, when detected early, melanoma survival exceeds 95%.

Melanoma occurrence is affected by the environment as well as the patient's organism. Environmental factors include ultraviolet exposure which mainly originate from sunlight but can be produced by tanning beds also. The ultraviolet rays damage the DNA of the genes that regulate the skin cells growth and ultimately lead to the formation of skin cancer [2]. Additionally, sunburns have been linked to melanoma development in the torso and more specifically, scientists have found a correlation between childhood sunburns and torso melanoma. Various aspects of the patient's organism also play an important role in the development of melanoma. The existence of numerous moles and being fair skinned also increase the risk of forming melanoma as their UV tolerance is reduced compared to the darker pigmented skin. Additionally, being older and a male are also factors that increase the occurrence of melanoma.

2.2. Skin Lesions

There are over 35 types of skin lesions split into two main categories of primary and secondary skin lesions. Primary lesions are skin abnormalities that are present at birth or appear during the life of a person, while secondary lesions are the result of irritation or manipulation of a primary lesion. In the confines of this particular thesis the number of skin lesions that are going to be present on the experiments is eight (Figure 1). Some of the most important recognized by the ISIC organization [30] skin lesion categories are:

1. Melanoma
2. Melanocytic nevus
3. Basal cell carcinoma
4. Actinic keratosis / Bowen's disease (intraepithelial carcinoma)
5. Benign keratosis (solar lentigo / seborrheic keratosis / lichen planus-like keratosis)
6. Dermatofibroma
7. Vascular lesion

8. Squamous cell carcinoma

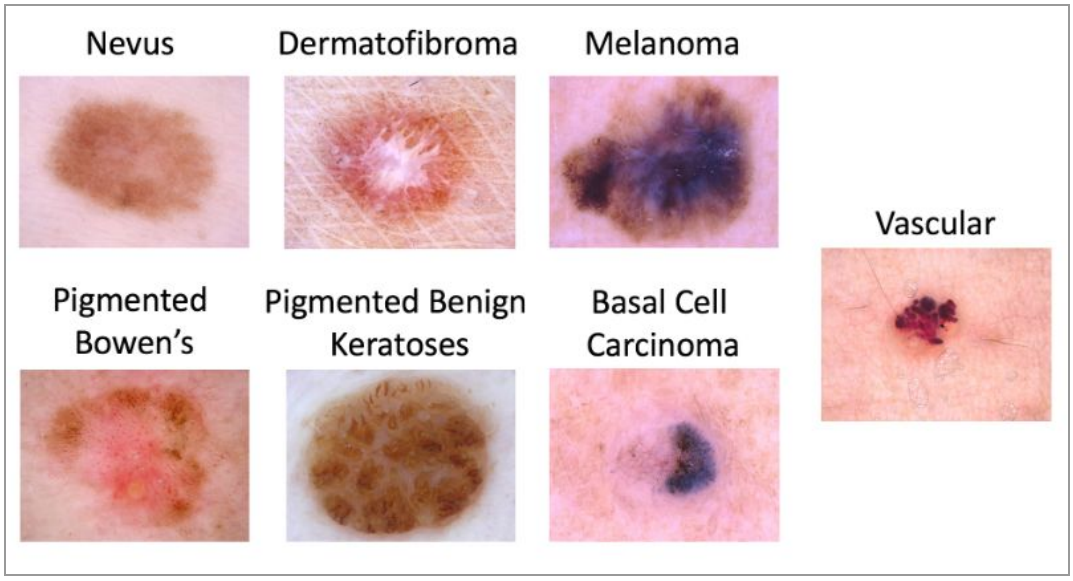


Figure 1. Various types of skin lesions

The majority of these types of skin lesions are benign, a setting that corresponds to the real world conditions where the benign incidencies are far more than the malignant ones.

2.3. Dermoscopy Images

In order to diagnose the different skin lesion types a dermatologist is employed as his initial diagnosis is going to determine the further examinations and treatment at last. As a result, an expert visual inspection of the abnormality is due by the dermatologist, usually employing high resolution cameras and algorithms for image analysis. Many centers have begun their own research efforts on an automated image analysis but a centralized system across institutions has yet to be implemented.

Dermoscopy is an imaging technique that adds a protocol on gathering the high resolution images of skin lesions. The various dermoscopy imaging devices operate on some values that are shared among them. The dermoscopy images that are captured from such a device are of high resolution comparable to a high end photography camera, are abundantly illuminated and most of the time there is a standard distance that the image is captured in order to assist the image comparison. Dermoscopy uses surface microscopy, that is also referred to as

epiluminoscopy or epiluminescent microscopy, and allows the dermatologist to look into the layers, color patterns and changes deep within the mole, rather than just glancing at the surface [3]. The increased depth penetration is achieved by eliminating the surface reflection of the skin that ultimately leads to enhanced optical resolution of the skin lesion. Research has shown that dermoscopy, when used by expert dermatologists, provides improved diagnostic accuracy than standard photography.

It is therefore critical to develop an accurate and reliable system for analyzing the dermoscopy images in order to aid the dermatologist process or even replace his services when needed. Towards that course, novel machine learning algorithms are employed and tested in order to analyze the images. A category of promising machine learning algorithms that perform well and show considerable accuracy results are neural networks. And it is this kind of algorithms that are going to be developed and used in this master thesis.

Chapter 3

Background Theory

3.1. Machine Learning

Artificial intelligence projects have been present for decades and one of the first approaches were hard coding logical inference rules using formal languages, an approach usually referred to as knowledge base. This kind of projects didn't have any particular success mostly because of the rigid nature of the software and the increasing complexity of rules that need to be coded in order to describe real world scenarios. The difficulties faced by systems relying on hard-coded knowledge suggest that AI systems need the ability to acquire their own knowledge, by extracting patterns from raw data. This capability is known as machine learning.

The machine learning (ML) term was first used by Arthur Samuel in 1959 where he used that term to describe a chess program (Samuel Checkers-playing Program) that was one of the

first self-learning programs [4]. Later, Tom M. Mitchell provided the first formal definition for ML: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E" [5]. What that translates to is a program able to get better at its task just by trying the same task over and over again.

In a non ML program, its architecture consists of an input step, a logic step and an output step. During the input step the programmer feeds the program with data and that data is fed in the logic module. The logic module is hardcoded by the programmer and performs the various calculations needed in order to produce the correct data that are forwarded to the output step.
So in traditional programming the logic that transforms the data is known beforehand so the programmer can provide the pipeline with the correct directions. In Machine learning the pipeline differs. There is also an input and an output step but the logic module does not have the directions hardcoded, instead the logic that produces the output data evolves as more data are provided in the input step.

There are four categories that machine learning applications can be split into: Supervised, Unsupervised , Semi-Supervised and Reinforcement learning.

3.1.1. Supervised Learning

In Supervised learning the dataset that is used as input consists of the independent variables of each example (features) and a dependent variable (target variable) or label for classification problems. The task of a supervised machine learning program is to produce the correct label from the features for each example. The main algorithm for these applications is to feed the data to the machine learning algorithm that predicts the label from the features for each example. After the prediction, the program calculates through a cost function the error in predicting the correct label. The final step is to perform adjustments to the parameters of the machine learning algorithm in order to reduce the value of the error. That procedure is then repeated until a satisfactory error is presented.The two applications of supervised learning are classification problems and regression problems.

In classification problems the examples correspond to a specific class. The class needs to be discreet and needs to be present in every example. The task that the program needs to complete is to predict the correct label or class for each example. The dataset can consist of binary class examples or multiclass examples. Some widely used classification algorithms are Support vector machines (SVMs), Logistic Regression, Naive Bayes Classifier and Neural Networks.

In regression tasks the target variable of each example is a continuous number that needs to be approximated as close as possible. In applications like that the output of the logic module of the program is a number. The goal of these programs, as in classification problems, is to minimize the error between the ground truth number (target value) and the predicted number. Some famous regression machine learning algorithms are Linear Regression, Support Vector Regressor (SVR) and Lasso Regression.

3.1.2. Unsupervised Learning

Unsupervised learning refers to those tasks that the target variable is not known beforehand and thus the program cannot be guided by the programmer (supervisor). Unsupervised machine learning is used to discover patterns in unlabelled data. Through those patterns the algorithms can cluster the data in groups that have similarities or even pinpoint data that work as outliers (anomaly detection). The main types of problems unsupervised machine learning is used are clustering, anomaly detection and association rule learning problems.

Clustering refers to the ability to group data together based on a similarity score between them. The most famous clustering algorithm is the k-means algorithm which uses a squared Euclidean distance formula as a measure of similarity (there other formulas that can be used for distance measurement based on the problem at hand). The product of such an algorithm is a set of k centroids that each one corresponds to the center of each cluster. The data are split into the various clusters based on their distance from each centroid. There are also other clustering algorithms that can be categorized as centroid-based, density-based, distribution-based and hierarchical.

Unsupervised Anomaly Detection is based on two assumptions, the first is that most of the data is statistically similar and the second is that only a small percentage presents strong variations and is in fact anomalous. Based on these assumptions, the data is clustered and the examples that are outliers are considered anomalous.

Association rule learning is the process of identifying the hidden relationships between the features of the dataset. The main concept of association rule learning is finding if/then relationships, where by the existence of certain features the system can predict the existence of others. Apriori algorithm is an association rule learning algorithm that finds the different associations in a dataset.

3.1.3. Semi-Supervised Learning

The third category of machine learning algorithms are semi supervised learning, a type of algorithms that use concepts from both supervised and unsupervised learning. The machine learning models in that category train on both labeled and unlabeled data with labeled data being a very small percentage of the dataset as a whole. Labeled data are used to help identify that there are different groups or classes in the data and what those classes might be. The rest of the data that are unlabeled are then used to find those different classes and even find additional. The profit from such models is great as there is not always access to labeled data and even then the size of them is minimal compared to unlabeled. The process of labeling data is time demanding and expensive so through semi supervised machine learning techniques the unlabeled data can produce insights [6].

3.1.4. Reinforcement Learning

In reinforcement learning problems the model, called agent, needs to train itself on a sequence of actions that get rewarded or punished. As such there is not a correct target variable that needs to be predicted but rather to come up with the optimal set of actions to maximise reward. These kinds of algorithms are mainly used in AI for video games, robotics and industrial robotics as well as dialog agents for speech and text. All these applications highlight the key

deployment of reinforcement learning models which is self learning by experience and behaviour [7].

3.2. Neural Networks

Neural networks or Artificial Neural Networks (ANN) are a set of machine learning algorithms that mostly fall into the supervised learning models category. They are used for both classification and regression problems but mostly for classification tasks. Neural networks are comprised of multiple neurons, a basic calculation unit that performs a single task and produces a number. That number can be zero or one, in case that the output of a neuron is one, the neuron is considered to be activated. That functionality and representation of this basic calculation unit resembles the biological neuron cells and that is where neural networks got their name.

The first neuron was called Perceptron and was conceived in the 1950s and 1960s by the scientist Frank Rosenblatt, inspired by earlier work by Warren McCulloch and Walter Pitts. Perceptron takes several binary inputs x_1, x_2, …, x_n and produces a single binary output as shown in figure 2 [8].

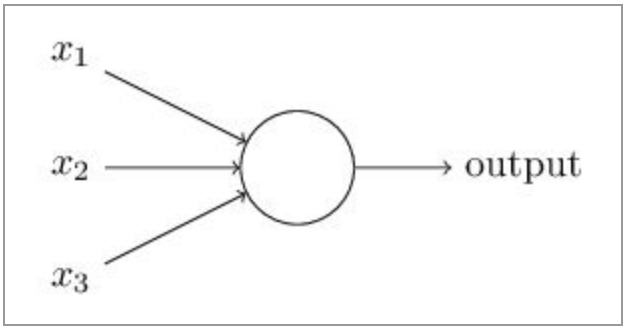


Figure 2. The first neural network neuron, the Perceptron

These inputs x are multiplied by coefficients called weights, which are real numbers that correspond to the importance of each input value, and are summed when used as an input for a perceptron neuron. The output of the neuron is then calculated to be either zero or one if the sum is greater or lower than a threshold value (Figure 3).

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

Figure 3. Activation check of the perceptron neuron that calculates the output of the node

Those basic calculations are enough to construct a perceptron and are sufficient for a single decision problem. For example if that perceptron was tasked to decide whether a meal was satisfactory or not, where one represents that it was and zero that it wasn't, then the inputs could be if the meal was adequate, if the meal was tasty and maybe the appearance of the meal itself. The next step would be to decide the importance weights for each input and the threshold value that tunes the ending result. Finally, if the output of the perceptron was wrong, by adjusting the values of the weights and threshold, the perceptron would be trained on that task and perform as it is supposed to.

The graphic representation of the activation function that corresponds to the conditional check of being greater than the threshold is called a step function and can be seen below in figure 4.

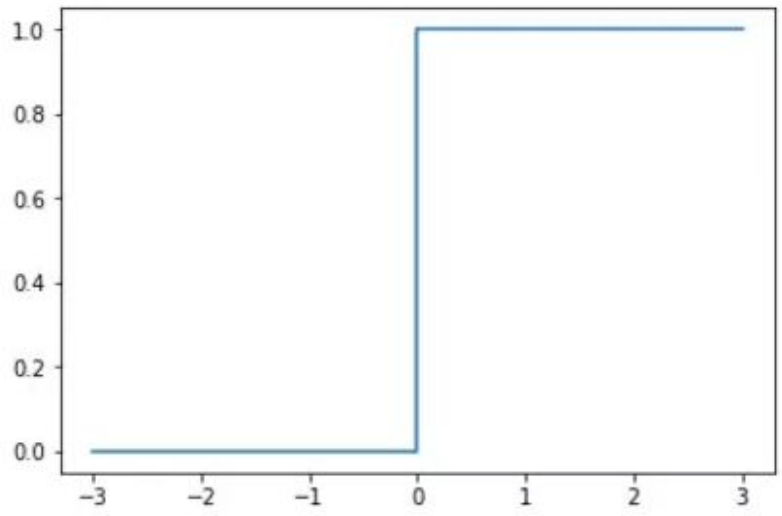


Figure 4. The curve of a step function

By moving the threshold variable to the first part of the inequality and by assigning a new variable called bias that equals with the negative of the threshold (b ≡ -threshold), the functions are transformed and the step function is centered around the y axis (Figure 5).

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Figure 5. The activation function with the bias variable

The meaning of the bias is to control how difficult a neuron can be activated, so a high bias value can make sure the neuron will be almost always activated and a negative value can hinder the activation of a neuron.

Perceptron has some disadvantages that originate from the way its activation function works. For example, a small change in the values of weights and biases can alter the output value from 0 to 1 and thus change the meaning of the task at hand. To combat that behaviour, the sigmoid neuron was developed. A sigmoid neuron has a sigmoid function as an activation function that can handle continuous number inputs and produce a continuous number output(Figure 6).

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}} \qquad z \equiv w \cdot x + b$$

Figure 6. A sigmoid activation function

There are many activation functions but most of them present the same sigmoid graphical representation. The figure for the logistic function is shown in figure 7.

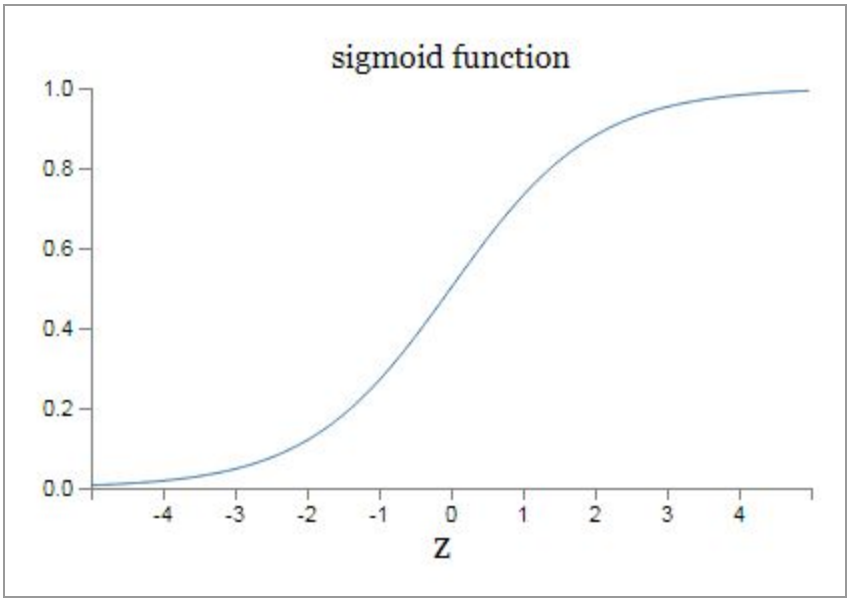


Figure 7. The curve that describes a logistic function

By combining neurons split into layers a neural network can be constructed based on the preferred architecture. A neural network has an input layer, where the input neurons reside and are handling the initial data inputs. After that layer there can be multiple layers of neurons or a single one, that layers are called hidden layers. Finally, an output layer is always present that produces the output values of the network.

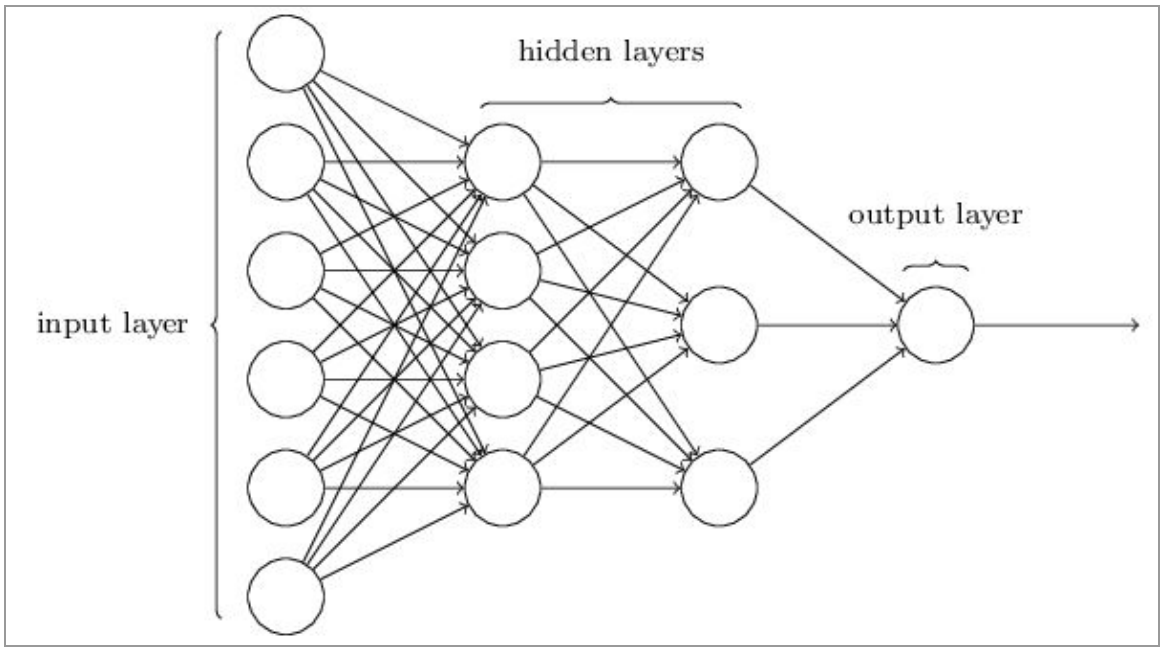


Figure 8. The architecture of a fully connected neural network

In figure 8 there is presented a fully connected feed-forward neural network. The term fully connected refers to the fact that the output of each neuron is used as an input all the neurons of the layer that follows and the term feed-forward shows the data flow in which case the flow is from left to right. In order to train the network there needs to be a measure of how well the network is performing on the task. That functionality provides the cost function for each network. The cost function measures how much is the deviation of the produced output than the correct output (ground truth). There are many cost functions that perform well on different architectures but one of the first and still popular is the quadratic cost function (Figure 9) [8].

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

Figure 9. The quadratic cost function

Where b and w are the vectors of all weights and biases of the network, n is the total number of training inputs x, a is the output of the network and y(x) are the ground truth values. With the introduction of cost functions there is now a measure of the network performance. When the value of the cost function is small then the output is not far off the ground truth and when the value is great then the network underperforms. In order to maximize the accuracy of such a network, the cost function needs to produce the minimum values possible. One of the most popular techniques to find all the weights and biases that minimize the cost function is with gradient descent.

Gradient descent is a technique used in calculus to determine the local and global minimums in multi variable problems by using partial derivatives. The idea is to calculate the slope in each point and then take a step towards the opposite direction in order to descend to the local or global minimum. For example the surface that can represent a problem for two variables can be seen in figure 10. The goal of the technique is to find those values of the two variables that correspond to the lowest point (minimum) of the surface by taking small steps towards that point [9].

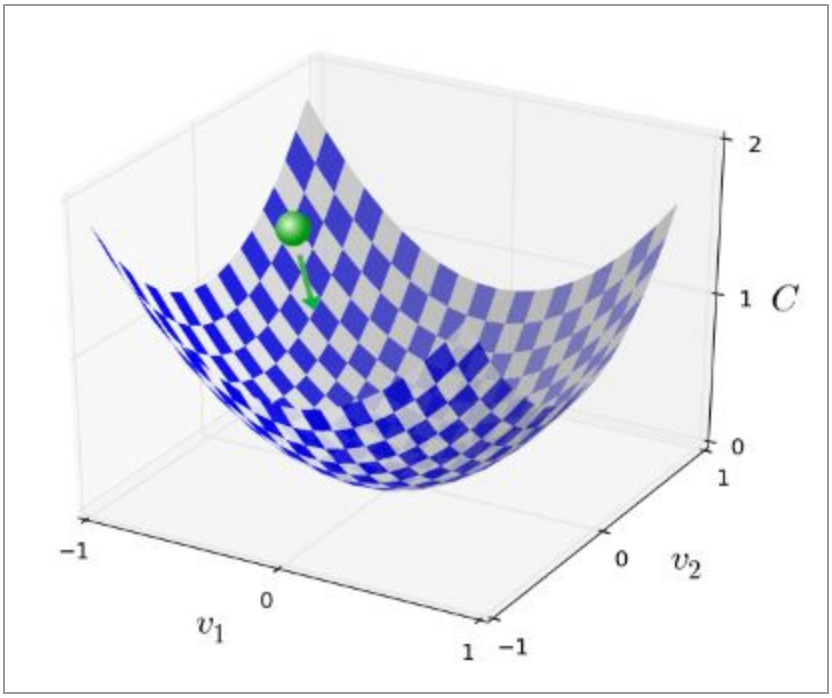


Figure 10. The error surface that describes a two variable problem

In order to calculate the contribution of each weight and bias on the cost function and finally measure the amount of the adjustment needed to reduce the value of the cost function an algorithm called backpropagation is used. The backpropagation algorithm was originally introduced in the 1970s, but its importance wasn't fully appreciated until a famous 1986 paper by David Rumelhart, Geoffrey Hinton, and Ronald Williams. Backpropagation aims to minimize the cost function by adjusting network's weights and biases [10]. The level of adjustment is determined by the gradient of the cost function with respect to those parameters. The backpropagation algorithm gets triggered after each forward pass through the network and performs a backward pass adjusting the values of weights and biases. The amount of adjustment is the difference between the old value minus the derivative multiplied with the learning rate parameter (Figure 11).

$$New\ weight = old\ weight — Derivative * learning\ rate$$

Figure 11. The weight and bias update on each pass

Where the learning rate parameter tunes how much the weights and biases need to be "punished" in order for the network to perform better. Usually a small learning rate is desired so that the steps towards convergence are smooth and steady.

3.3. Computer Vision

Computer vision is a multi-disciple field that focuses on replicating the capabilities of the human vision system in order to enable computers to identify and process objects in digital images and videos. Its task is to extract meaningful information from images or video frames in order to accurately infer the contents of those images. Computer vision (or CV) is different than image processing in a way that image processing involves the manipulation of pixel data and not the content inference but it is in many cases an important initial step before running various computer vision algorithms. The main problems that computer vision is tasked to solve are [11]:

- Object Classification: Which category of objects are shown in the image
- Object Identification: What type of object is shown
- Object Verification: Does the object exists
- Object Detection: Where are the objects
- Object Landmark Detection: Where are the key points of the object
- Object Segmentation: Which pixels belong to each object class
- Object Instance Segmentation: Which pixels belong to each object
- Object Recognition: What objects are contained and where are they

The principles of computer vision are similar to machine learning tasks, the first step is to find a suitable representation of the content of a digital image so that features can be extracted and then used as an input for the algorithm. The next step is to train the algorithm on those features and finally produce a prediction on the content. Traditionally, there have been multiple techniques for feature extraction such as eigenfaces, which was a novel way of identifying faces through images by projecting the images to a set of eigenvectors [12]. The eigenvectors were extracted through principal component analysis which computes a new set of axes(eigenvectors) that each corresponds to facial features. By calculating the eigenvalues of images projected on those axes the algorithm can perform face recognition (Figure 12).

Figure 12. The features that were produced through the eigenfaces technique

Another popular technique that is used in the field of computer vision is histogram of oriented gradients. The histogram of oriented gradients (HOG) feature descriptor is the process of mapping every change on the value of each pixel in concern with the adjusting ones [13]. So in a grayscale image where every pixel value ranges from 0 (black) to 255 (white) the vertical and horizontal gradients for each pixel are calculated using the grad operator (partial derivatives). By dividing the image into small (usually 8x8 pixels) cells and blocks of 4x4 cells. Each cell has a fixed number of gradient orientation bins. Each pixel in the cell votes for a gradient orientation bin with a vote proportional to the gradient magnitude at that pixel (Figure 13).

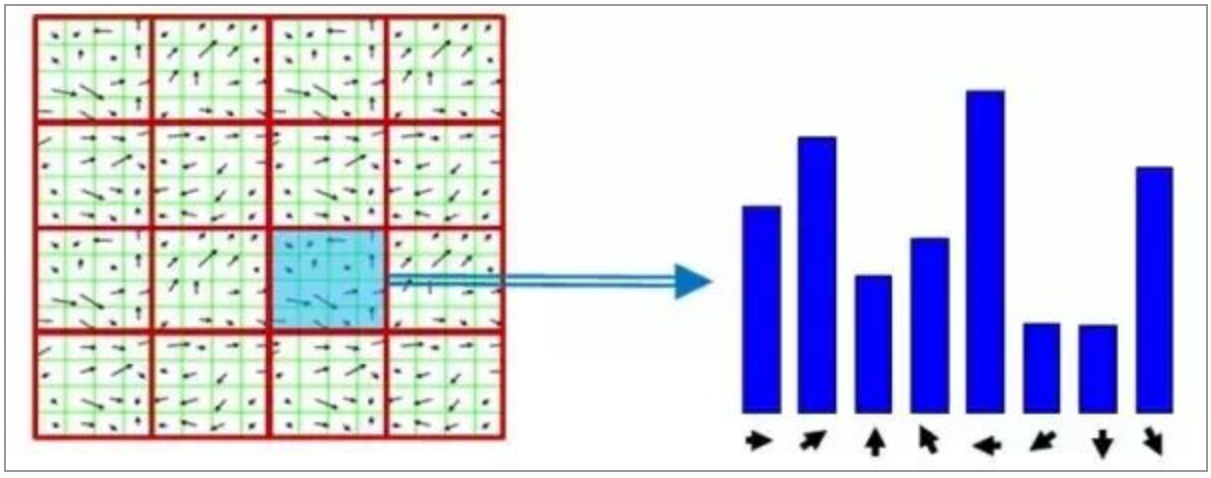


Figure 13. The final gradient orientation voting for a selected cell

By calculating the gradients through a HOG algorithm the produced feature vector can be considered as a collection of information about the structure of the image (Figure 14).

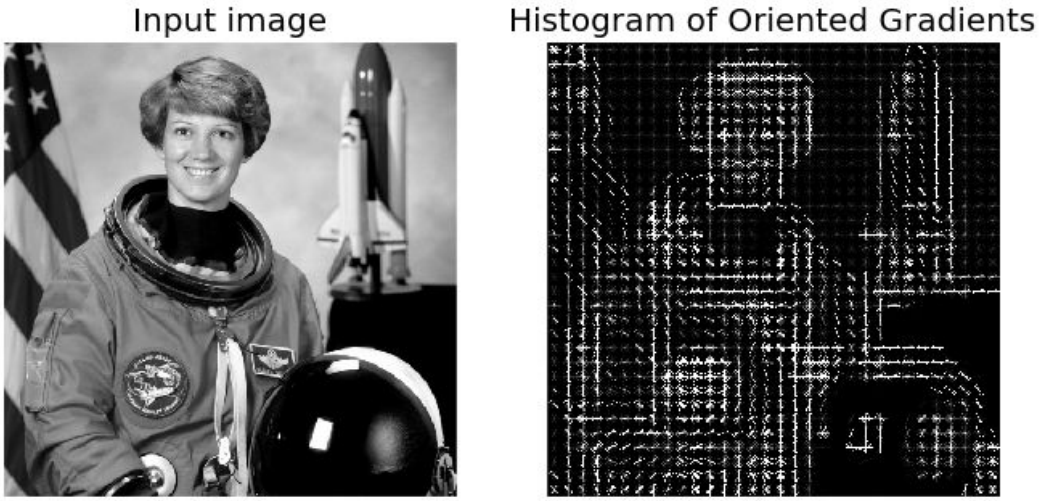


Figure 14. The produced feature map of the image through the HOG algorithm

After obtaining the feature vector then by using a classifier like Support vector machines (SVMs) for example the classifier can be trained on those structures and predict the class of an image or perform object detection by checking the existence or absence of them.

3.4. Convolutional Neural Networks

Convolutional neural networks are a deep ANN variant that aims to reduce the computational strain that accompanies deep learning in computer vision tasks. The first version of a CNN was first presented in 1980 by Dr. Kunihiko Fukushima who proposed a neural net architecture called Neocognitron [14]. That multilayered ANN in each layer had to perform feature extraction on a limited region of the input image much like the receptive field that CNNs use. That way Neocognitron was able to recognise edges and structures and perform robust visual pattern recognition. Later, in 1989 Yann LeCun proposed a pioneering version of the Convolutional Neural Network that was based on a Neocognitron-like architecture and used the

backpropagation training algorithm [15]. From then on, CNNs have been used on almost all computer vision problems as their performance in many tasks is comparable to that of a human.

The basic ideas that govern all CNN architectures are three: local receptive fields, shared weights and biases, and pooling. In the next sections a brief description on how these basic concepts work is presented [8].

3.4.1. Local Receptive Fields

For a Fully Connected Neural Network (FCN) that is used for image recognition the input layer would consist of neurons that correspond to the pixels of the image. Each neuron in the input layer will then connect to each neuron in the first hidden layer and so on. In CNNs, again the input layer consists of the same number of neurons that are the pixel values of the image but the connections to the first hidden layer differ. A small window of pixels is used to connect to each hidden layer neuron that is called local receptive field (Figure 15). The idea is for each hidden neuron to learn to analyze its local receptive field.

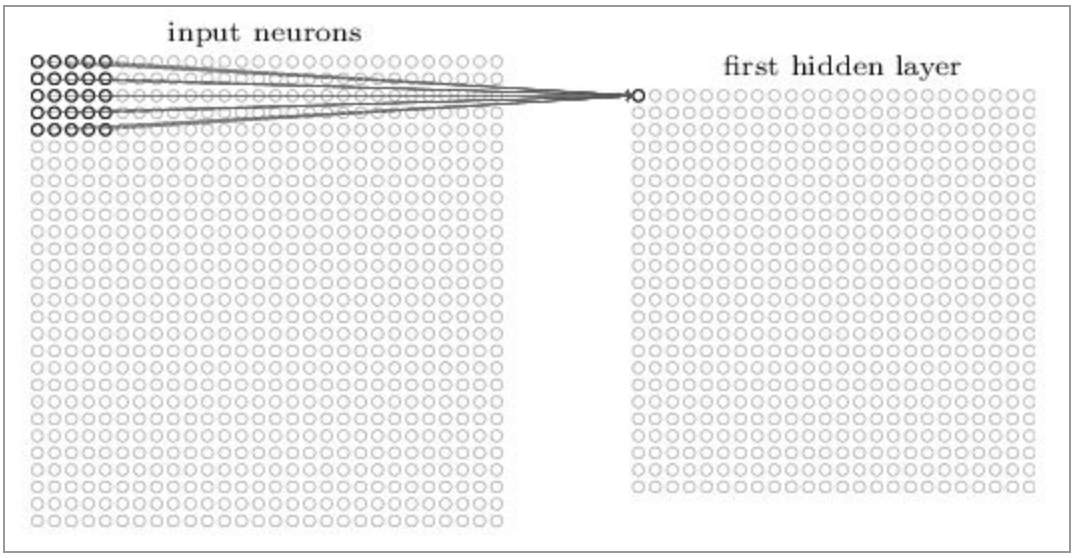


Figure 15. The connections of a single local receptive field

The size of the local receptive field is tunable and is one of the hyper parameters that can be adjusted in any CNN architecture. After the first connection the receptive field is moved to scan all the input pixels by a fixed value called stride length. In the end all the hidden neurons will correspond to a connection to a local receptive field of the input layer. The concept of local

receptive field is to extract local features by analyzing a small region and then by combining these simple structures to produce complex patterns.

3.4.2. Shared Weights and Biases

Each hidden neuron has a single connection to a local receptive field with as many weights as the pixels the local receptive field contains. These weights are the same or shared for every scan of the image with that particular local receptive field. The neurons that connect to the first scan are called a feature map. The concept is to scan the image with a receptive field that by using constant weights, detects a set feature for example an edge. The weights for that feature map are called shared weights and the bias that activates the neuron is called shared bias and they define a kernel or a filter as it is mentioned in relevant literature.

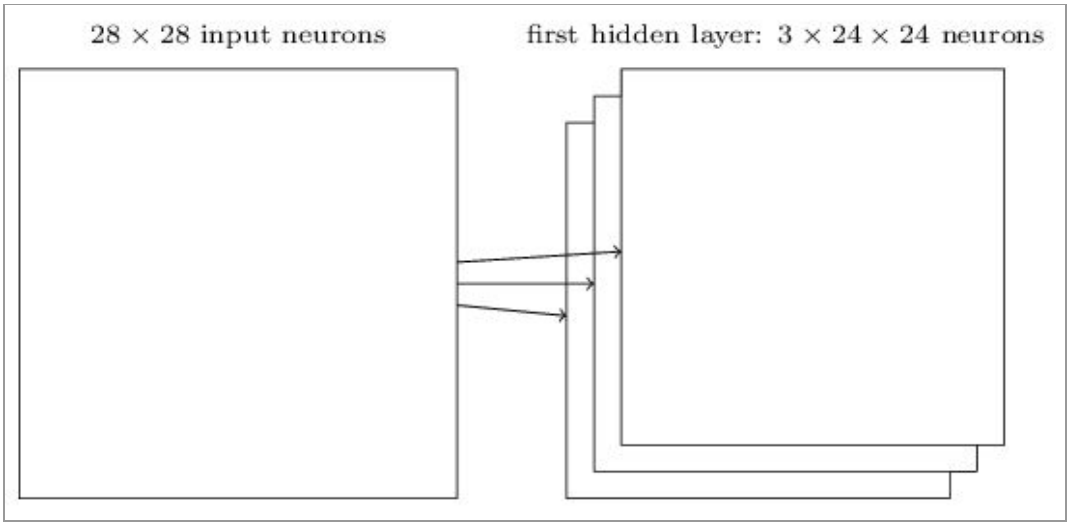


Figure 16. The three feature maps produced through the local receptive field scanning

In figure 16, there are 3 feature maps shown. Each feature map is defined by a set of 5×5 shared weights, and a single shared bias. The result is that the network can detect 3 different kinds of features, with each feature being detectable across the entire image. In recent CNN architectures the number of feature maps is great, resulting in advanced feature detection for the input image. In an early CNN architecture called LeNet-5 the number of feature maps that was used was six each associated with a 5x5 local receptive field [16]. The task that the CNN was used was the digit classification of the Modified National Institute of Standards and Technology

(MNIST) handwritten digit dataset [38]. Some feature maps (or kernels or filters) that were produced by the CNN are shown below.

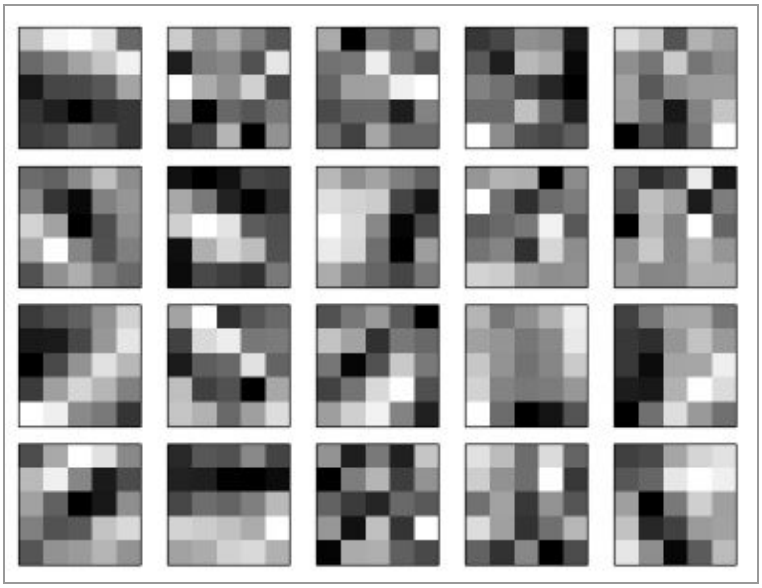


Figure 17. The structures that are detected by the feature maps

These 20 images, shown in figure 17, correspond to 20 different filters that detect specific structures. The whiter pixels show smaller weight values that respond less to the input pixel values and darker pixels means that the weight values are bigger and thus are sensitive to the input pixel values. An advantage of using convolutional layers that comprise of feature maps is that the parameters that are needed to be calculated are much fewer that using a fully connected layer. For each map the only parameters that make it up are the shared weights and a single shared bias.

3.4.3. Pooling Layers

The last basic concept of convolutional neural networks are pooling layers. Pooling layers follow in the architecture the convolutional layers. Their purpose is to simplify the output information of the convolutional layers. Pooling layers are made up with condense feature maps that are produced by scanning the feature maps of the convolutional layers. Each unit of the pooling layer examines a small region of the feature map, for example 2x2, and applies a function that sums up the information contained in that region. So for example, a max pooling layer will keep the maximum value of that region and discard the others (Figure 18). An L2

pooling layer will calculate the square root of the sum of the squares of the activations in that region [8].

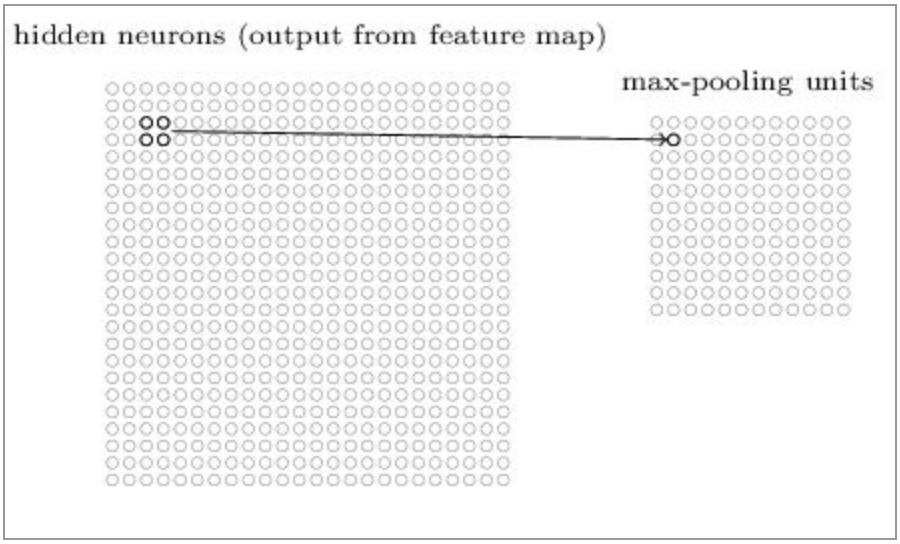


Figure 18. The connections of a max pooling layer that preserved the highest value only

The above procedure is applied on all feature maps and a pooling map is produced for each one of them. The resulting pooling maps contain the information of where a feature is relative to the other features and discards redundant information. The information discarded corresponds to parameters that are not important for the task and so the neural network shrinks in computational load.

These basic concepts form convolutional neural networks and by tuning the hyperparameters or inserting more layers various architectures can be produced. Usually for classification purposes, as an output layer, one or more fully connected layers are used that consume the outputs of the last pooling layer and produce the classification results (Figure 19).

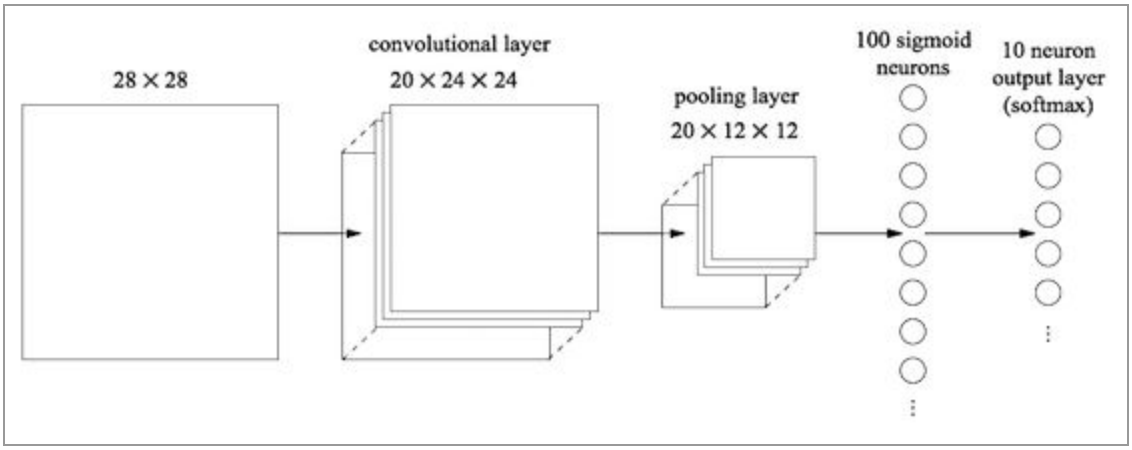


Figure 19. A convolutional neural network architecture that is uses the last fully connected layer and the softmax layer to perform classification on a 10 class task

3.5. Hyperparameters of CNNs

While trying to achieve optimal accuracy and performance of a neural network model, the machine learning engineer has the option to tune various parameters that determine how the network will train or even its architecture. These parameters are called hyperparameters and are extremely important in the overall performance of every ANN. There are some rules on deciding what is the optimal value of some of the hyperparameters but the process of hyperparameter tuning has a significantly experimental aspect.

3.5.1 Learning rate

One of the most important hyperparameters in ANNs and consequently in convolutional neural networks is the learning rate of the architecture optimizer. The learning rate hyperparameter refers to the amount of adjustment that is going to be applied to the weights of the network after each backpropagation pass (Figure 20).

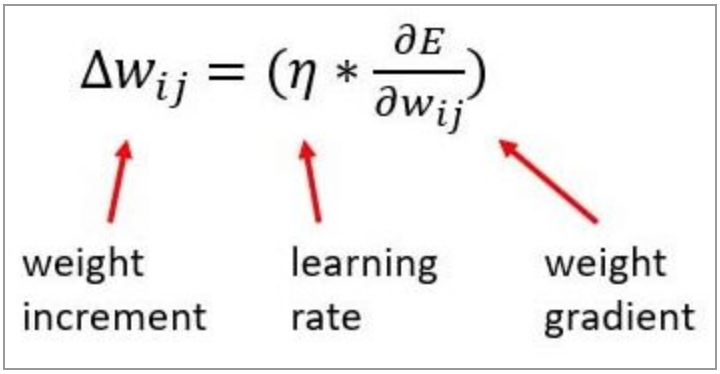


Figure 20. The weight update that gets applied in each pass

A small learning rate value in a gradient descent algorithm will require many iterations to find the global minimum as the step towards the optimal will be minimal. Even then, the minimum might not be a global minimum but a local one and by taking small steps in a specific direction might prove ineffective to emerge from the local minimum [17]. Furthermore, a high learning value will require much fewer iterations but the problem is that it might pass the minimum and never really converge or even diverge (Figure 21).

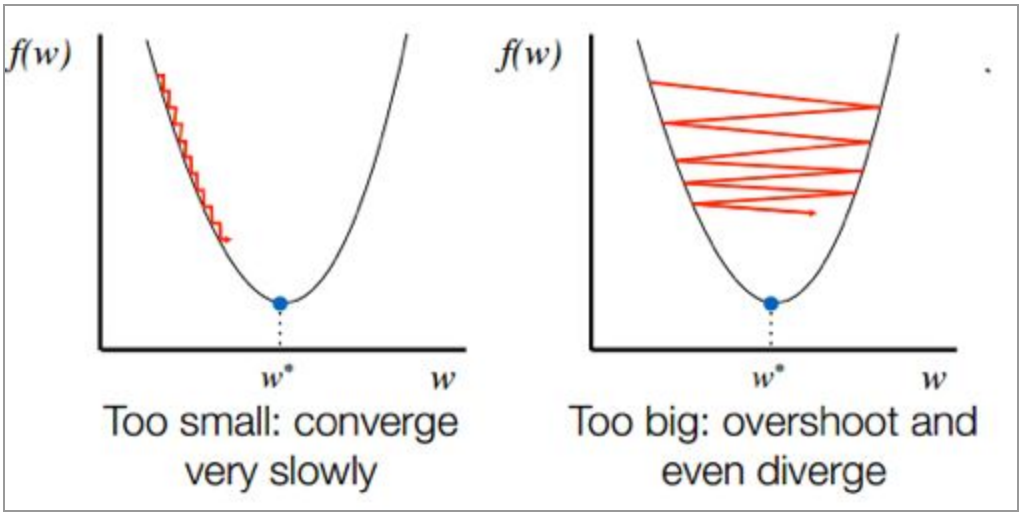


Figure 21. The effects of a badly tuned learning rate on convergence

3.5.2 Mini batch size

When training an ANN the engineer has the option to use a single example from the dataset in each pass or a set of examples. A common optimizer technique that can be used with all the variations on the number of examples is Stochastic Gradient Descent (SGD), that uses a

predefined number of examples for training in each pass. SGD replaces the gradient descent technique that uses the entire dataset (batch training) and reduces by a lot the computational cost of training in turn of a smaller convergence rate. When used with a single example or a set of examples (mini batch), SGD randomly chooses a subset of the dataset and trains the network on the isolated data. The main benefit of using a mini batch technique is the ability of the optimizer to produce noisy process updates and ultimately avoid local minimums and actually converge [18].

3.5.3 Dropout

Dropout is a generalization technique used in neural networks during the training phase in order to help avoid the over fitting of the model to the dataset. The main concept of dropout is to ignore a random subset of the neurons during training and not apply weight adjustment to them. By avoiding neurons the units become less dependent on their connecting neighbors and value the existence of more robust features that are not interdependent. Consequently, the model can be thought of as a juncture of slightly different architecture models that all are tasked with the same problem by producing a robust set of features (Figure 22) [19]. Dropout is applied by specifying the dropout hyperparameter value that ranges between zero and one. That value corresponds to the probability to drop a random set of neurons and is not applied at all after the training has finished, for example when using the trained model for prediction.

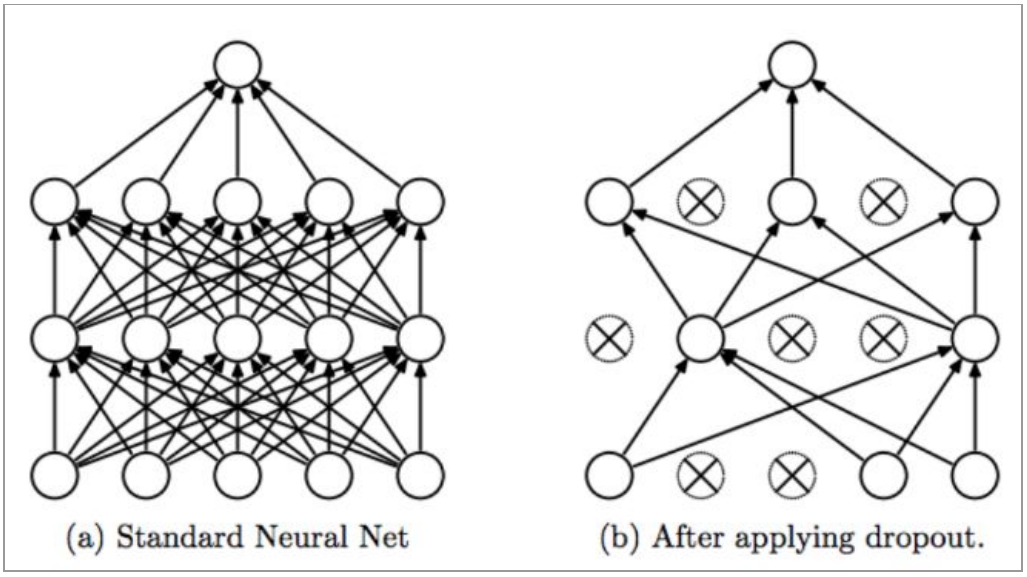


Figure 22. a) The architecture of a fully connected neural network during training without dropout, b) The architecture of the network that undergoes training when applying dropout.

Chapter 4

Neural Network Architectures

4.1. Related Work

The experiments that are going to be presented are focused on an annual recurring challenge called the ISIC (International Skin Imaging Collaboration) challenge. The purpose of that challenge is to develop novel automatic techniques of correctly classifying a number of skin lesions in order to improve the diagnosis capabilities of the medical staff. The ISIC Archive is the largest repository of quality controlled and publicly available digital images of skin lesions and the collection is ever growing. The images are screened to ensure satisfactory quality and are then examined by melanoma experts in order to verify the credibility of the metadata that accompany each image. The metadata contain mainly demographic information and diagnosis

details. A more thorough examination of the subset of the ISIC dataset that is going to be used is presented in chapter 5.1.

Through the ISIC challenge initiative and the nature of the problem itself, there has been great interest in tackling the skin lesion classification task. The promising gain from an automated technology that addresses that problem is huge in both resources as well as diagnostic speed and accuracy. A field of machine learning that has shown a great deal of potential is that of deep learning especially in recent years. In the paper '*Deep learning outperformed 136 of 157 dermatologists in a head-to-head dermoscopic melanoma image classification task*' that was published in April of 2019 [20], that potential is clearly portrayed. During their experiments the research team trained a convolutional neural network called ResNet50 on skin lesion image data and compared the results of the network with the results of domain expert diagnostic doctors.

The ResNet50 is a deep residual convolutional neural net that is comprised of 50 layers and 25.6 million parameters in total. The ResNet family networks were the winners of the Imagenet challenge of 2015 and are considered one of the breakthroughs of deep learning in computer vision [21]. Their main innovation was the introduction of a technique that addresses the vanishing gradients as well as the exploding ones in very deep neural networks. That technique is called skip connection (residuals) which connects the output of one layer with the input of the previous one. Skip connections allow the network to learn deviations from the identity layer, hence the term residual, residual here referring to difference from the identity (Figure 23).

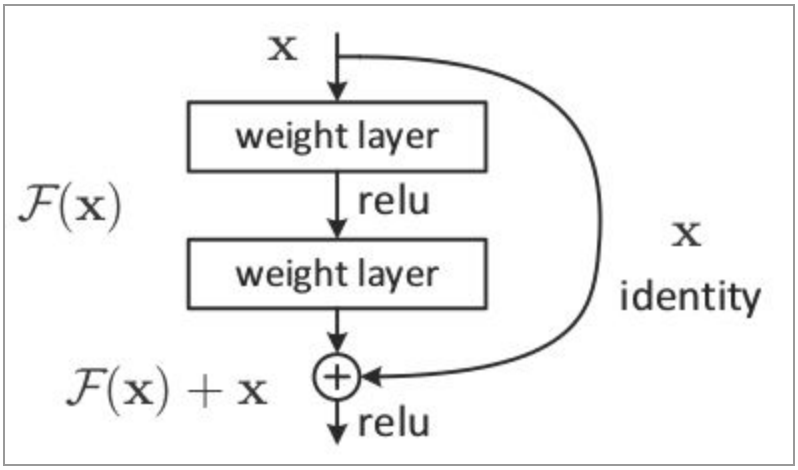


Figure 23. A residual learning building block

Through the usage of these residual building blocks the team was able to construct very deep networks that avoided the degradation problem of the gradients by ensuring that the next layers would not perform worse than the previous ones. The resulting network architectures (Figure 24) performed excellent on the ImageNet challenge and won the 2015 contest with a top-1 error of 19.38% and a top-5 error of 4.49%.

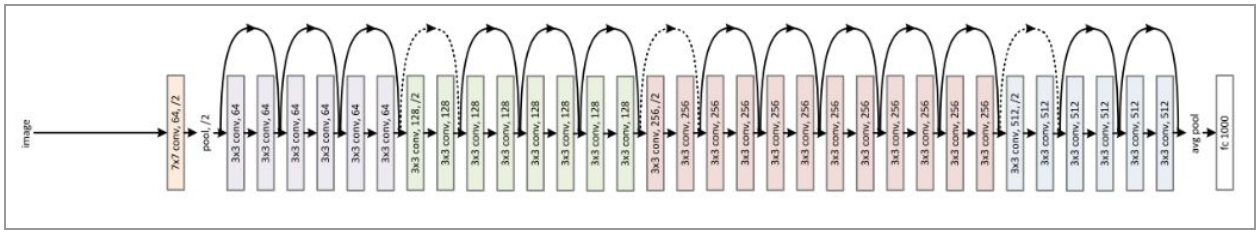


Figure 24. The architecture and building blocks of the ResNet 152

With the ResNet-50 architecture pretrained on the ImageNet dataset the research team trained the network on the ISIC challenge 2018 dataset which contained a total of 2,169 melanomas and 18,566 atypical nevi. The testing dataset that was reserved, was presented to 157 dermatologists from 12 German university hospitals of all levels of training including a small subsample of resident physicians for prediction. The algorithm had a sensitivity of 76% and a specificity of 81.7%, on average. Compared with the results of the resident physicians, who achieved a mean sensitivity of 67.7% and a mean specificity of 65.8% on the test set, the mean

specificity of the CNN was better by 15.9 percentage points at approximately the same sensitivity.

Another approach that was presented in the paper of April 2017 called '*Skin lesion classification from dermoscopic images using deep learning techniques*', was the use of a deep neural network, by the standards of 2014, called VGG [22].

VGG was developed by the Visual Geometry Group of Oxford University and is comprised of either 16 or 19 layers. That particular CNN, while only containing 16 or 19 layers, is a resource draining network because of the fully connected layers it contains. The resulting number of parameters that need to be trained is ~134-144 million a fact that deters its usage in many cases. The architecture of a VGG neural network can be seen in figure 25.

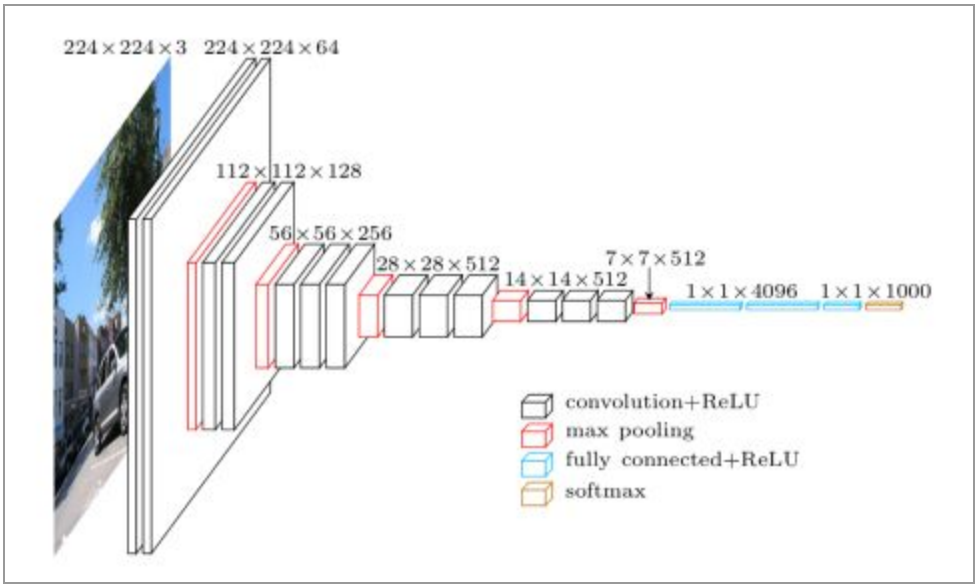


Figure 25. The architecture of a VGG neural network

For their experiments on the performance of VGG on dermoscopic images the research team decided to use three methods (M1) training the CNN from scratch, (M2) using the transfer learning paradigm to leverage features from a VGGNet pre-trained on the ImageNet dataset and (M3) keeping the transfer learning paradigm and fine-tuning the CNNs architecture. Again the dataset that was used, was provided by the ISIC challenge and was composed of 346 training images and 150 testing images. The classes that those images belonged to were again two, that of malignant and benign skin lesions. For the training purposes, because of the small number of

example images for training a deep convolutional network, data augmentation was used in order to increase their number through rotation, horizontal flipping, zooming and scaling. The results on the testing set for all three methods that were explored are presented in figure 26.

	Loss	Accuracy	Sensitivity	Precision
M1	0.6743	66.00%	0.5799	0.6777
M2	1.0306	68.67%	0.3311	0.4958
M3	**0.4337**	**81.33%**	**0.7866**	**0.7974**

Figure 26. The results of the VGG network on the testing set of the ISIC dataset

Deep learning can be also used along with other machine learning techniques in order to utilize the strengths of multiple algorithms. In the paper '*Skin Lesion Classification Using Hybrid Deep Neural Networks*' that was published in February of 2017 such a workflow is presented [23]. The main concept of that research was the use of multiple deep learning networks along with support vector machines (SVMs) in order to tackle the problem of skin lesion classification. Three convolutional neural networks were used namely AlexNet, VGG16 and ResNet-18 for feature extraction on the ISIC dataset. By using those three networks pre trained on the ImageNet dataset, the research team extracted the captures features of each network and attached a support vector machine classifier on the output of each network. As a final step these three architectures were used as an ensemble of networks (Figure 27) and by voting the resulting predictions were able to generalize better.

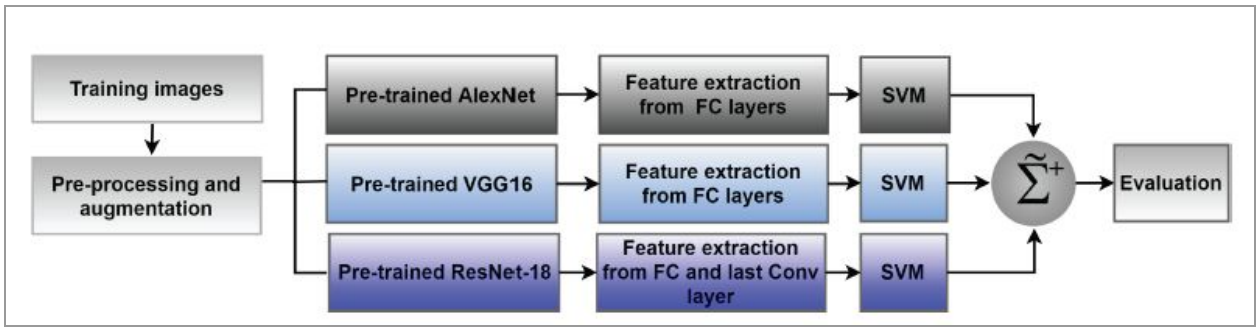


Figure 27. The flowchart of the hybrid ensemble of networks

The ISIC 2017 dataset that was used, was composed of three classes (malignant melanoma, seborrheic keratosis and benign nevi) and 2,037 images across all classes in total. The requested results from the 2017 challenge dictated that the performance was measured on two cases malignant melanoma vs. all and seborrheic keratosis vs. all classifications. The results of the hybrid ensemble of networks can be seen in figure 28.

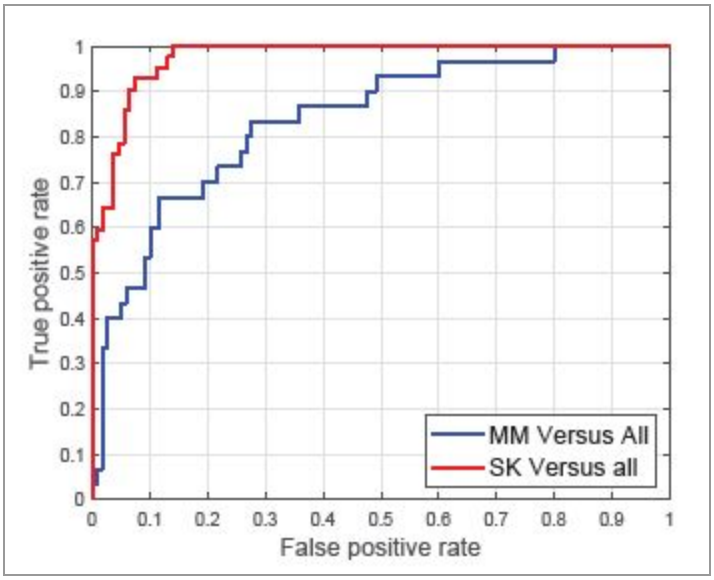


Figure 28. ROC curve of the best performing approach

4.2. EfficientNet

4.2.1. Compound Scaling

The artificial neural network that is going to be used in this master's thesis is based on the EfficientNet architecture and its main principles. EfficientNet was first presented in the paper "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks" by Mingxing Tan and Quoc V. Le in 2019 [24]. The goal of the paper was to address the problem of scaling the neural network architectures in order to increase accuracy.

Convolutional neural networks can be scaled up or down by tweaking three architecture parameters: the depth, the width and resolution. Depth refers to the number of hidden layers and

can be adjusted to fit the problem at hand. For example the ResNet-18 can be scaled up to ResNet-200 by increasing the number of hidden layers [21]. The number of the hidden layers can be thought of as its capacity to construct complex structures from simple ones. Width refers to the number of channels(nodes) that comprise the layers of a convolutional network and can be thought of as the number of structures that the network looks for while scanning the images. The last architecture parameter that can be scaled up or down is the resolution of the input images where by feeding the network with increased number of pixels you provide to the CNN increased information. The network architect can increase one, two or even all three of these parameters in order to manually tune the performance of the CNN but that process is tedious work and prone to errors. The work presented in the efficientNet paper proposes a way that uniformly scales all dimensions of depth/width/resolution using a simple yet highly effective compound coefficient (Figure 29, 30). The main reason that the dimensions of the network need to be exactly what the problem needs to be solved is the computational cost that accompanies an increase in a dimension.

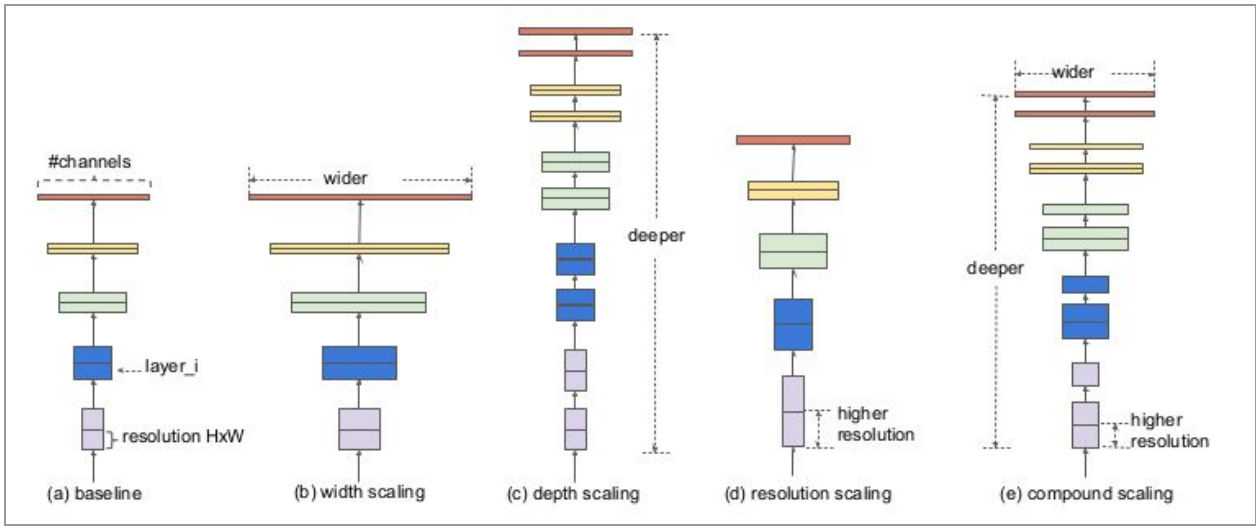


Figure 29. a)The baseline network, b)The network after increasing its width, c)The network after increasing its depth, d)The network that accepts higher resolution images, e)The baseline network that is expanded through compound scaling

The proposed compound scaling method is a set of rules to scale the three dimensions using a compound coefficient φ (Figure 30). Through that coefficient the dimensions are scaled in a uniform way. The user specified coefficient φ represents how many more resources are available for model scaling while α, β , γ denote how to assign these extra resources to the dimensions of the network. The constants α, β ,γ can be determined by a small grid search, where by trying combinations of them and evaluating the network an optimal one is chosen.

$$\begin{aligned} \text{depth: } & d = \alpha^{\phi} \\ \text{width: } & w = \beta^{\phi} \\ \text{resolution: } & r = \gamma^{\phi} \\ \text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned}$$

Figure 30. The formula that calculates how much the dimensions must change

4.2.2. EfficientNet B0

The EfficientNet paper also introduces a new baseline artificial neural network architecture called EfficientNet B0. The EfficientNet B0 is the ancestor of a family of neural networks that are produced through compound scaling and are named EfficientNet B1 to B7 [24]. The baseline network uses as a building block the mobile inverted bottleneck MBConv (Figure 31) and presents a similarity with the network MnasNet but is a bit bigger due to the larger FLOPS target (400M). EfficientNet B0 was chosen to be based on the mobile neural network architectures due to the fact it would be then scaled up to form the whole EfficientNet family. The MnasNet that provided the MBConv building block is a mobile neural network that was constructed through the neural architecture search (NAS) auto machine learning technique [25].

Neural architecture search is an auto machine learning technique with increased popularity lately, that is tasked with finding the optimal neural network architecture for a

selected problem. The purpose of NAS can be best described by Google CEO Sundar Pichai, who wrote that, "designing neural nets is extremely time intensive, and requires an expertise that limits its use to a smaller community of scientists and engineers. That's why we've created an approach called AutoML, showing that it's possible for neural nets to design neural nets."
Neural architecture search has three main dimensions [26]:

- The search space defines the type(s) of ANN that can be designed and optimized.
- The search strategy defines the approach used to explore the search space.
- The performance estimation strategy evaluates the performance of a possible ANN from its design (without constructing and training it).

For the construction of MnasNet the search space of the NAS technique was tuned to a factorized hierarchical search space that could construct different architecture building cells in order to present diversity and find the optimal combination. Along with the tuning of the search space of the NAS, search strategy and performance estimation are tweaked in order to minimize the latency of the resulting network and not FLOPS since the latter is not a clear indication on the speed of the network in mobile devices.

From the MnasNet architecture, the building block used in EfficientNet was MBConv6 that denotes mobile inverted bottleneck convolution and DWConv denotes depthwise convolution while k3x3/k5x5 denotes kernel size (Figure 31).

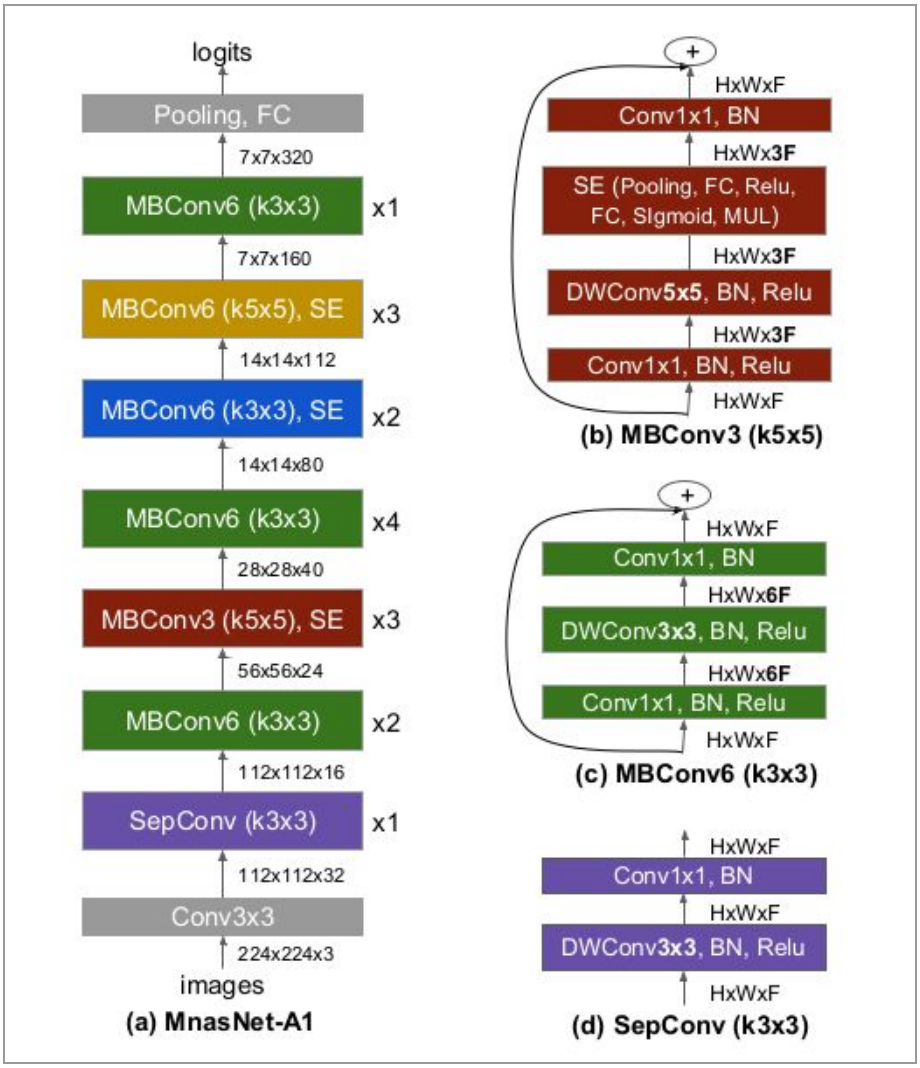


Figure 31. a)The MnasNet architecture that uses as building blocks the b,c and d cells

Depthwise convolutions were introduced in MobileNetV1 and are a type of factorized convolutions that reduce the computational cost as compared to standard convolutions [27]. Suppose an input volume of D_u x D_u x M is transformed to an output volume of D_v x D_v x N. The first set of filters are comprised of M single-channel filters, mapping the input volume to D_v x D_v x M on a per-channel basis. This stage known as depth-wise convolutions, resembles the intermediate tensor and achieves the spatial filtering component. In order to construct new features from those already captured by the input volume, we require a linear combination. To do so, 1x1 kernels are used along the depth of the intermediate tensor(point-wise convolution). N such 1x1 filters are used, resulting in the desired output volume of D_v x D_v x N (Figure 32) [28].

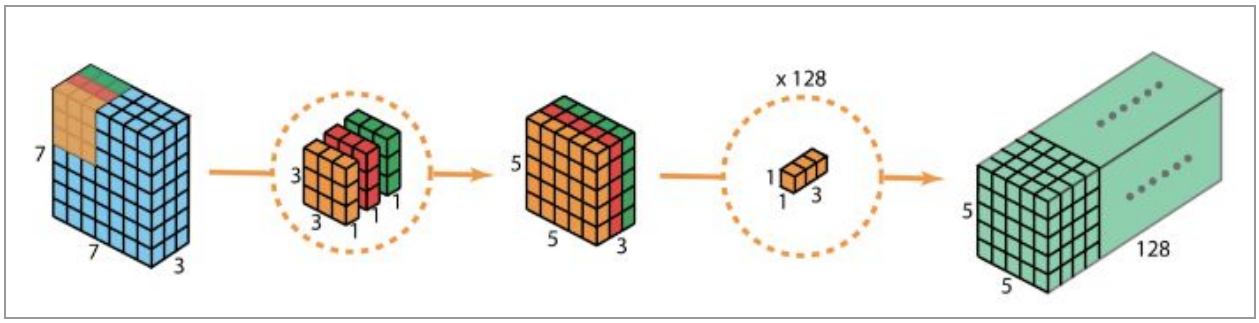


Figure 32. Depthwise convolution process

The building block of MBConv6 follows the inverted residual block architecture by swapping the classic wide -> narrow -> wide steps to narrow -> wide -> narrow. The first step is a 1x1 convolution that increases the depth while reducing the width of the network followed by a depthwise convolution and the last step is again a 1x1 convolution that squeezes the network again in order to match the initial channel number. In order to construct deeper architectures and avoid the problem of vanishing gradients a skip connection is used between the input and the output of the building block (Figure 33).

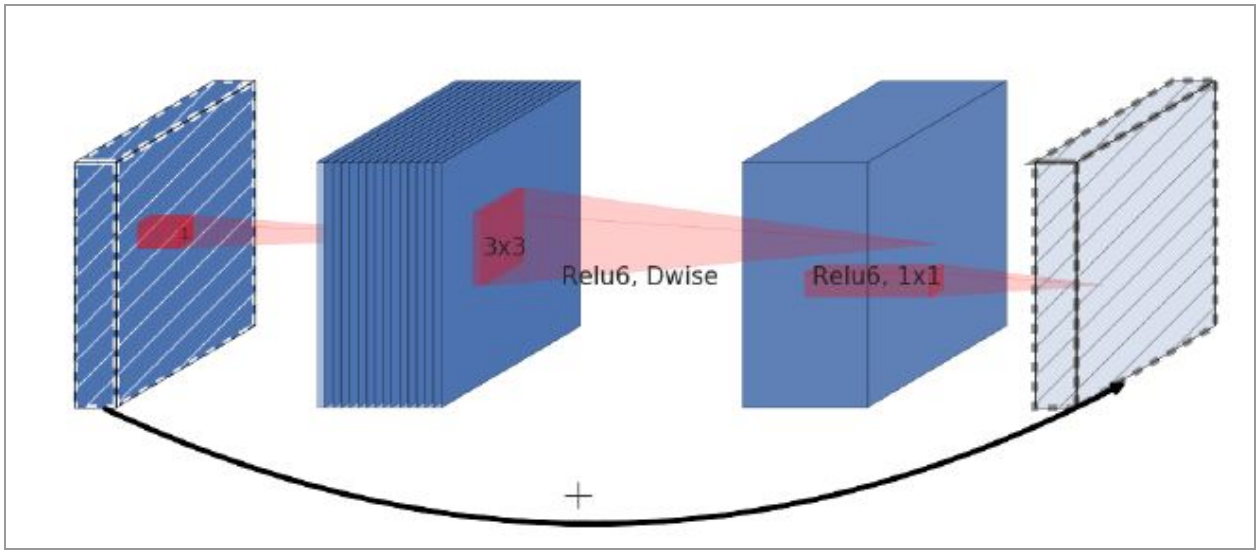


Figure 33. The MBConv6 process that is used in its building block

The Relu6 is an activation function that is based on the rectified linear unit function that has gained popularity due to the fact that addresses the vanishing gradients problem. The vanishing gradients refer to the fact that while backpropagating the gradient error in deep networks the error gets smaller and smaller and ultimately does not affect the initial layers. A Relu activation function can decrease that effect but also presents a flaw of having a range of [0, infinity). The solution to that attribute comes in the form of Relu6 that caps the range on the value 6 (Figure 34) [29].

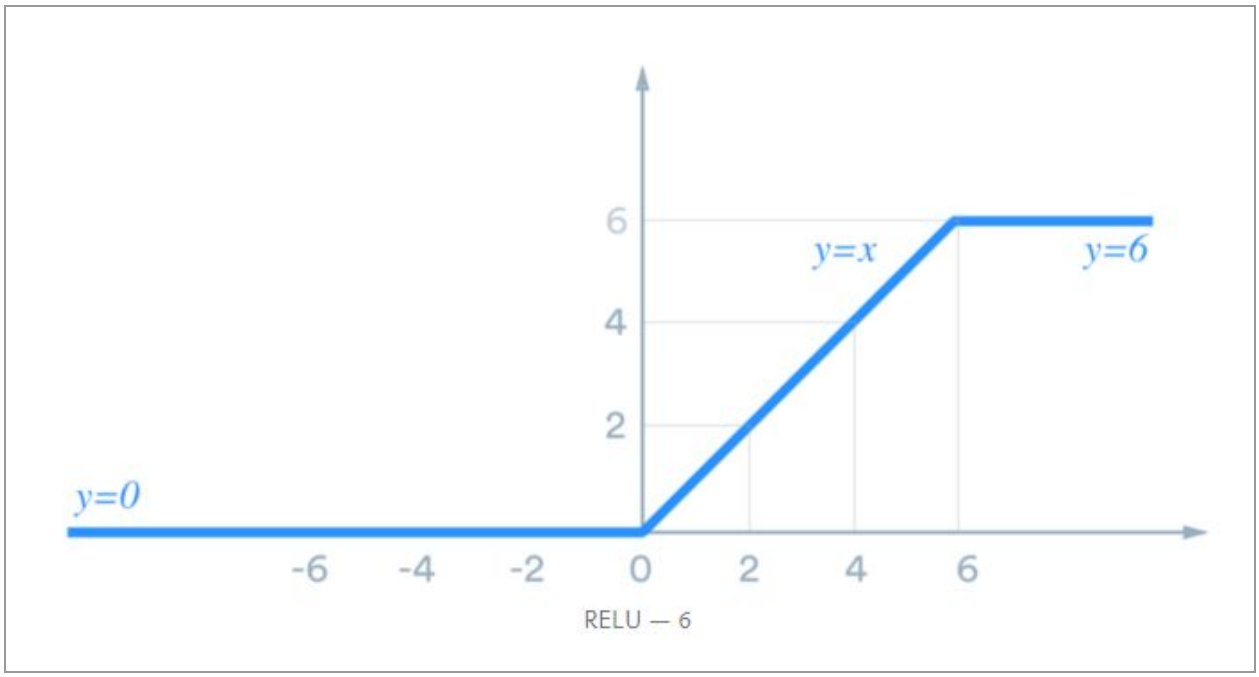


Figure 34. The curve that describes the Relu6 activation function

Ultimately the resulting architecture of the baseline EfficientNet B0 that is used to produce the EfficientNet family is comprised of the layers shown in figure 35.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Figure 35. The architecture of the baseline neural network EfficientNet B0

4.2.3. EfficientNet Family

Through the compound scaling technique a new set of neural networks are constructed that make up the EfficientNet family. These artificial neural networks because they use as a baseline network the EfficientNet B0, that is a network with architecture building blocks of mobile networks, and because they use the compound scaling method to expand their dimensions present great accuracy with much smaller number of parameters. When compared to the top neural architectures the efficientNet family produces similarly accurate results while keeping the computational cost at a minimum. In figure 36 a comparison with some popular architectures and their performance on the imagenet dataset can be seen. From the figure, one can notice the exceptional accuracy results that the efficientNet family produces and also notice at the bottom right corner the much smaller number of the parameters that directly translate to decreased computational cost.

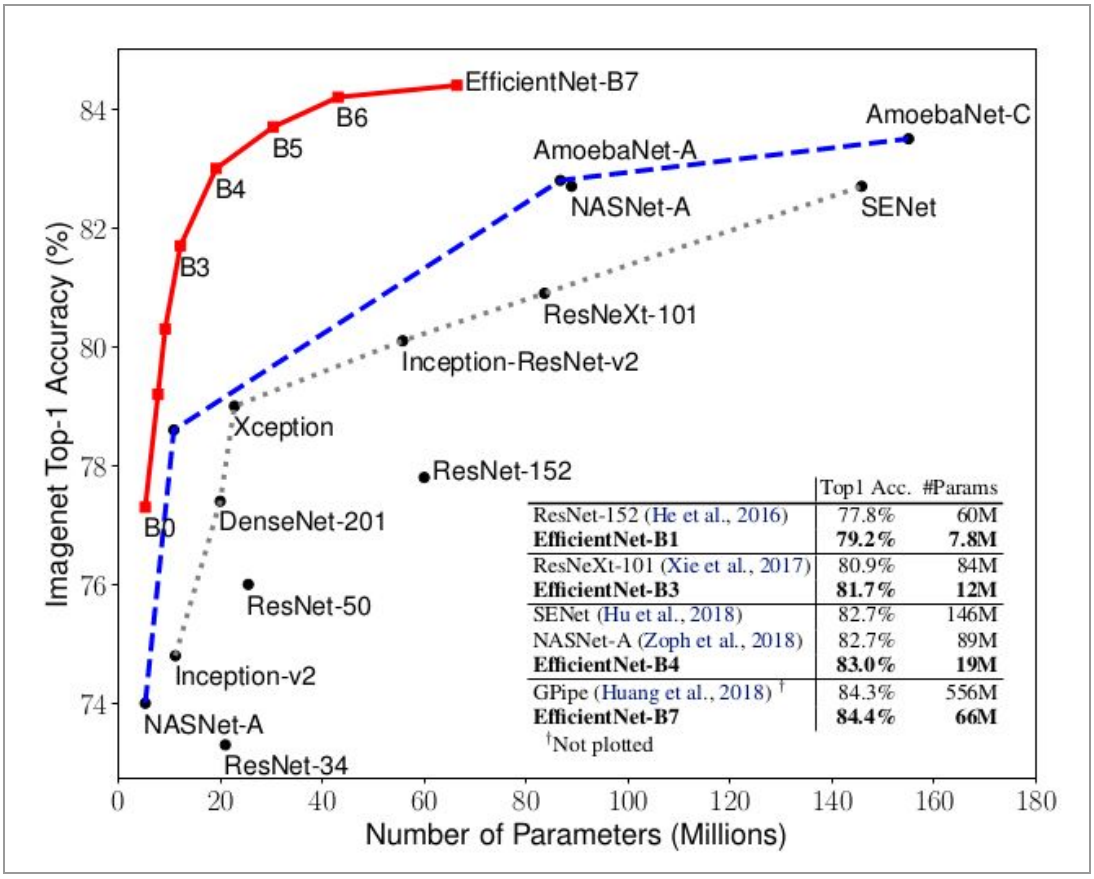


Figure 36. Accuracy results of the EfficientNet family networks on the imagenet dataset compared to some popular neural network architectures

Chapter 5

Configuring the ANN and running the Experiments

5.1. The Dataset

ISIC Challenge provides a dataset to train the machine learning algorithms that are going to be competing and finally test their accuracy. The dataset consists of 25,331 dermoscopic

images of varying resolutions and the images belong to 8 distinct classes [30]. The classes of the dataset are:

1. Melanoma
2. Melanocytic nevus
3. Basal cell carcinoma
4. Actinic keratosis
5. Benign keratosis
6. Dermatofibroma
7. Vascular lesion
8. Squamous cell carcinoma

Among those classes, the number of the images that belong to each one is not constant. Consequently, the dataset is highly imbalanced and this is something that affects the produced accuracy of the machine learning models. Specifically, the number of images for each class are the following:

Skin Lesion	Count
Melanoma	4,522
Melanocytic nevus	12,865
Basal cell carcinoma	3,323
Actinic keratosis	867
Benign keratosis	2,624
Dermatofibroma	239
Vascular lesion	253
Squamous cell carcinoma	628

The images are all contained in a folder with each one of them having a unique id filename. Along with the image folder, a csv is provided that contains the class of each id. As a first step the images are split into different folders that each of them corresponds to one class. The procedure is carried out by scanning the images and then finding by the filename the actual class and copying that file to the correct folder (Figure 37).

```
# Split the ISIC Data to class folders

import os
import pandas as pd
from shutil import copyfile


parent_dir = '../../Desktop/ISIC Data/'
with os.scandir(parent_dir + 'ISIC_2019_Training_Input/ISIC_2019_Training_Input/') as entries:
  training_gt = pd.read_csv(parent_dir + "ISIC_2019_Training_GroundTruth.csv")

  # print(training_gt.iloc[0,0])
  for entry in entries:
      filename = parent_dir+'ISIC_2019_Training_Input/ISIC_2019_Training_Input/'+entry.name

      # Get the row from the dataframe that corresponds to the image
      row = training_gt.loc[training_gt['image'] == entry.name.split('.')[0]]

      classname = str(row.columns[(row == 1).iloc[0]][0])

      # Create the class folders
      os.makedirs(parent_dir+'Class_folders/'+ classname, exist_ok=True)
      copyfile(filename, parent_dir+'Class_folders/'+ classname +"/" + entry.name)
```

Figure 37. The code that is responsible for splitting the images into class folders based on their class that is marked in the ground truth csv (Split_Data.py)

The resulting structure is a set of 8 folders with the images from that class in each folder. This task is needed in order to use the flow_from_directory method of the image generator class that exists in the Keras framework [31].

5.2. Transfer Learning

Due to the fact that the convolutional neural networks demand a lot of resources to train, a common technique to save time and resources is to use transfer learning. Transfer learning denotes using a pre trained neural model on a dataset other than the problem at hand. The benefits are that the initial weights of the network are not random but actually refer to a structure simple or complex. After importing such a network, the training can be isolated to the last few layers or on all of them by adjusting them to the current task. The performance of pretrained networks is phenomenal and is a common practice among machine learning engineers when the data are limited or when the resources needed to train the network are great.

5.3. Training the network

The training of a convolutional network can be done on a medium range pc if the task is not that demanding or the network is not that complex. In most cases a high end pc is needed in order to meet the computational costs of training the CNN. Additionally, there is also the option of using the CPU, for the calculations during the training, or the GPU. The CPU in most cases is slower than the GPU because the calculations are not complex but great in number. For that reason, a processing unit that is built for speed like the GPU outperforms the CPU. Recently, a new processing unit that is targeted towards neural network training has been developed by Google and is called TPU. That unit is built specifically for machine learning purposes and outperforms both the CPU and the GPU.

For the needs of this thesis, the training of the neural network was realized on the Google colaboratory platform. Colab (short for colaboratory) is a cloud platform that gives you access to a python kernel, either in version three or two, for you to run your code [32]. The layout is similar to the jupyter notebook framework for python that splits runnable cells and cells using the markup language. The main benefit of using Colab for the neural network training is the access it gives you to a GPU and a TPU if you choose to. The GPU you can use is a Tesla K80 and also 12Gb of RAM, the only restriction is that the maximum duration of using the GPU is capped at 12 hours of continuous load.

The first step is to import the efficientNet B0 model from the efficientnet.keras repository using the pretrained model on the imagenet dataset [33]. The model comprises of the pretrained weights and all its layers except the top ones. After importing the efficientNet B0 a GlobalMaxPooling2D layer is added in order to convert the 4D tensor to 2D that results in much fewer parameters and after that layer a fully convolutional layer with 8 nodes is added that will output the class probabilities. The activation function on the fully connected layer is a softmax function that is a function that takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers. The final output is a set of normalized probabilities one for each class. The final neural network architecture can be seen in figure 38.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
efficientnet-b0 (Model)      (None, 7, 7, 1280)        4049564
_________________________________________________________________
gap (GlobalMaxPooling2D)     (None, 1280)              0
_________________________________________________________________
dropout_out (Dropout)        (None, 1280)              0
_________________________________________________________________
fc_out (Dense)               (None, 8)                 10248
=================================================================
Total params: 4,059,812
Trainable params: 10,248
Non-trainable params: 4,049,564
```

Figure 38. The total parameters of the network to be trained along with the number of trainable and non trainable parameters

5.3.1 Training the last two layers

After constructing the model architecture, we set the layers of the base model to be untrainable so the training can take place on the last layers that were added. This technique is quite common where the pretrained base model is left untouched and the last layers are optimized for the specific task. The neural network is trained with the execution of the compile method of the model class from the keras framework that sets the optimizer and the fit_generator method that loads the data to start the training. In order to use the EfficientNet B0 model the images need to be resized so they can be used as an input for the neural network. That task is carried out by executing the center_crop_resize function that is provided in the utilities section of the efficientnet module (Figure 39).

```
from efficientnet.keras import center_crop_and_resize
import cv2
import os

parent_dir = '../../Desktop/ISIC Data/Class_folders/VASC/'
output_dir = '../../Desktop/ISIC Data/Class_folders_456/VASC/'
with os.scandir(parent_dir) as entries:
    for entry in entries:
        filename = parent_dir+entry.name
        img = cv2.imread(filename)
        resized_image = center_crop_and_resize(img, 456)
        cv2.imwrite(output_dir + entry.name, resized_image)
```

Figure 39. The code that is responsible for resizing the images to the correct efficientNet input dimensions (Resize_Data.py)

The hyperparameters that were tuned in order to reach optimal accuracy, were dropout, batch size, epochs, optimizer and learning rate (Figure 40).

224x224	batch size	epochs	optimizer	learning rate	dropout rate	loss	accuracy	val_loss	val_accuracy
	32	20	rmsprop	2.00E-05	0.2	2.0486	0.5341	2.6094	0.5181
	32	5	adam	0.001	0.2	1.7627	0.5632	1.9575	0.4987
	32	20	adam	0.001	0.2	1.6488	0.5746	2.3844	0.5057
	128	20	Nadam	0.002	0.2	2.8871	0.5473	3.2152	0.5064

Figure 40. The resulted accuracy and loss for different set of hyperparameter values

The most accurate model for the task was trained with hyperparameter values of:

- Dropout: 0.2
- Batch size: 32
- Epochs: 20
- Optimizer: Adam optimizer with a learning rate of 0.001

The model class of the keras framework also exposes a history method that holds information of the training process of the model and grants access to various metrics and information. The accuracy and loss of the model during training is shown in figure 41.

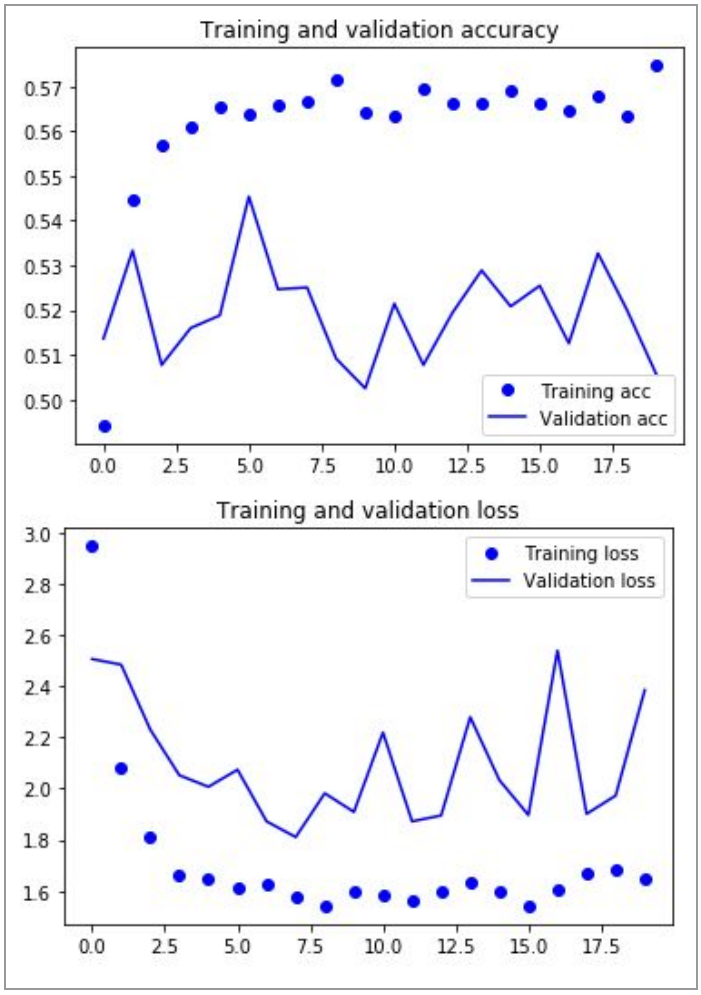


Figure 41. The accuracy and loss changes during training of the best performing EfficientNet B0 model

The validation accuracy suggests that the training is not optimal and there is not a satisfactory convergence. In an attempt to improve the accuracy of the model, the initial dataset was modified so it can be balanced. Data augmentation can help increase the number of the images in classes that are restricted in number. Data augmentation refers to those transformations that are applied to an image and produce a family of images that are different for the purposes of training a machine learning algorithm. Some transformations affect the color scheme of the image like hue, saturation, brightness, contrast. While others, affect the geometrical attributes of the image like rotation, zoom, crop, perspective [34]. For this experiment, geometrical data augmentations were prefered. From the training class folders the class with the lower number of

images was Dermatofibroma (DF) with 161 images. The transformations that were applied were rotation on 90,180,270 degrees angle and horizontal flipping on each rotation (Figure 42).

```
import os
import cv2


input_path = '../../Desktop/ISIC Data/Class_folders_456/Train_456/VASC/'
output_path = '../../Desktop/ISIC Data/Class_folders_456/Train_456_balanced/VASC/'
angles = [0, 90, 180, 270]

with os.scandir(input_path) as entries:
    for entry in entries:
        # Read image from the disk.
        img = cv2.imread(input_path+entry.name)
        for angle in angles:
            # getRotationMatrix2D creates a matrix needed for transformation.
            # We want matrix for rotation w.r.t center to the angle degrees without scaling.
            M = cv2.getRotationMatrix2D((456 / 2, 456 / 2), angle, 1)
            image = cv2.warpAffine(img, M, (456, 456))
            # Save the flipped and the not flipped image for each angle
            for i in range(0, 2):
                output_filename = entry.name.split(".")[0] + "_" + str(angle) + "_" + str(i) + "." \
                                  + entry.name.split(".")[1]
                if i == 1:
                    image = cv2.flip(image, 1)
                # Write image back to disk.
                cv2.imwrite(output_path + output_filename, image)
```

Figure 42. The code that is responsible for the data augmentation on the dataset

The result was 1,528 images. After applying those transformations on all classes that needed augmentation the dataset was balanced with a constant number of 1,528 images across each class. Note that the validation data must not be produced from augmented data but rather comprise of entirely different images so that the validation results are reliable. The results of training again the efficientNet B0 model on the balanced dataset can be seen in figure 43.

224x224	batch size	epochs	optimizer	learning rate	dropout rate	loss	accuracy	val_loss	val_accuracy
	32	20	adam	0.001	0.2	1.6379	0.527	1.9354	0.4923
	32	50	adam	0.001	0.2	1.5943	0.5357	1.8144	0.3959

Figure 43. The resulted accuracy and loss for different set of hyperparameter values on the balanced dataset

The accuracy and loss for the 50 epochs model can be seen in figure 44.

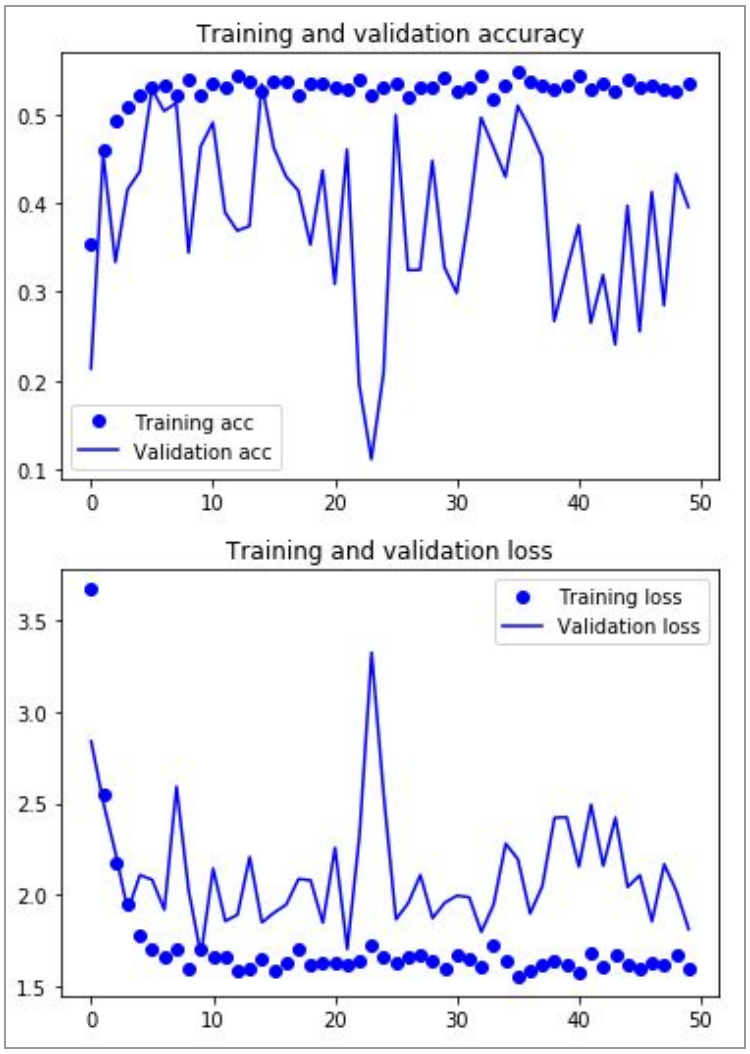


Figure 44. The accuracy and loss changes during training of the 50 epochs EfficientNet B0 model

For the next experiment, the neural network that is going to be trained is the EfficientNet B5 network of the same family. The purpose of this experiment is to see if the higher resolution images will help improve the accuracy of the model. The EfficientNet B5 model uses as an input 456x456 resolution images and is wider and deeper than the B0 by the magnitudes that are presented in figure 45.

```
def efficientnet_params(model_name):
    """ Map EfficientNet model name to parameter coefficients. """
    params_dict = {
        # Coefficients:   width,depth,res,dropout
        'efficientnet-b0': (1.0, 1.0, 224, 0.2),
        'efficientnet-b1': (1.0, 1.1, 240, 0.2),
        'efficientnet-b2': (1.1, 1.2, 260, 0.3),
        'efficientnet-b3': (1.2, 1.4, 300, 0.3),
        'efficientnet-b4': (1.4, 1.8, 380, 0.4),
        'efficientnet-b5': (1.6, 2.2, 456, 0.4),
        'efficientnet-b6': (1.8, 2.6, 528, 0.5),
        'efficientnet-b7': (2.0, 3.1, 600, 0.5),
    }
    return params_dict[model_name]
```

Figure 45. The scaling of the width, depth and resolution of each network of the EfficientNet family

Again after importing the pretrained EfficientNet B5 model without its last layers, a GlobalMaxPooling2D and a fully connected layer that predicts 8 classes are added. The final architecture of the B5 model can be seen in figure 46.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
efficientnet-b5 (Model)      (None, 15, 15, 2048)      28513520
_________________________________________________________________
gap (GlobalMaxPooling2D)     (None, 2048)              0
_________________________________________________________________
dropout_out (Dropout)        (None, 2048)              0
_________________________________________________________________
fc_out (Dense)               (None, 8)                 16392
=================================================================
Total params: 28,529,912
Trainable params: 28,357,176
Non-trainable params: 172,736
```

Figure 46. The architecture of the EfficientNet B5 model to be trained

After setting the base model to be untrainable and the last 2 layers to trainable, the training of the network is performed by running the functions that were used for the EfficientNet B0 model training. The top performing EfficientNet B5 models can be seen in figure 47.

456x456	batch size	epochs	optimizer	learning rate	dropout rate	loss	accuracy	val_loss	val_accuracy
	48	20	adam	0.001	0.4	8.0534	0.3324	12.4923	0.0661
	64	20	adam	0.001	0.4	3.593	0.376	7.9447	0.0754

Figure 47. The top performing EfficientNet B5 models

The accuracy and loss graphs for both training and validation for the last model can be seen in figure 48.

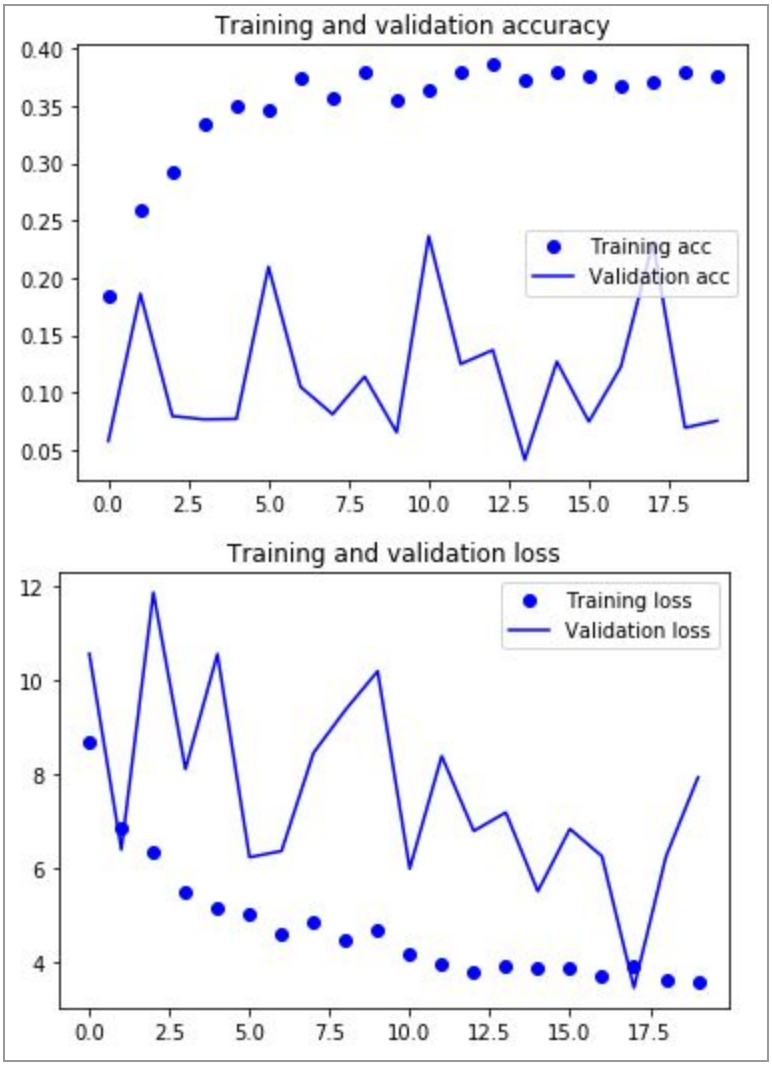
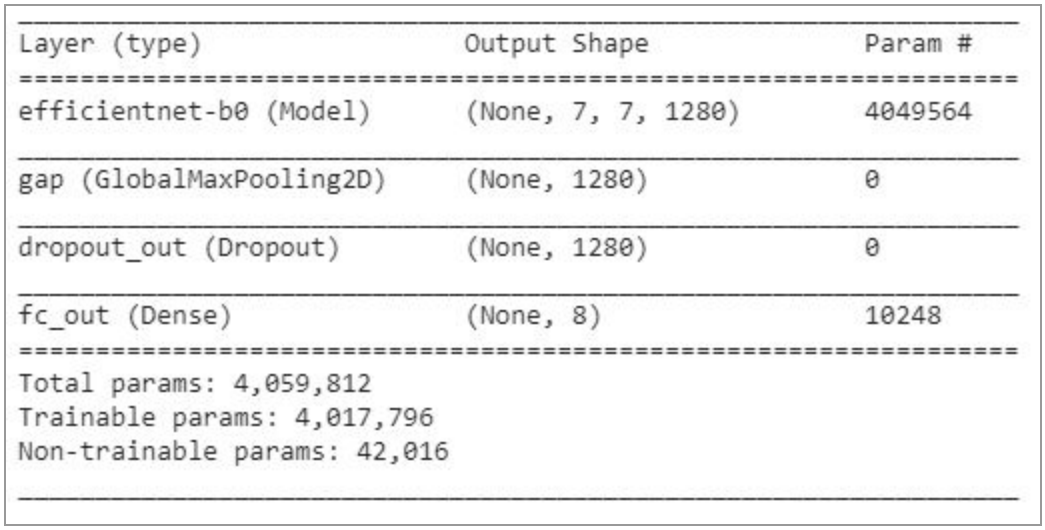


Figure 48. The accuracy and loss graphs for the top EfficientNet B5 model

5.3.2 Training the whole network

In this section the performance of the EfficientNet B0 will be studied when the whole network is available for weight adjustment during training. The process is exactly the same except for the step that sets the base model to untrainable (conv_base.trainable = False). By

setting the whole network available for training the number of trainable parameters increases from 10,248 to 4,017,796 (Figure 49)



Layer (type)	Output Shape	Param #
efficientnet-b0 (Model)	(None, 7, 7, 1280)	4049564
gap (GlobalMaxPooling2D)	(None, 1280)	0
dropout_out (Dropout)	(None, 1280)	0
fc_out (Dense)	(None, 8)	10248

Total params: 4,059,812
Trainable params: 4,017,796
Non-trainable params: 42,016

Figure 49. The architecture and number of parameters of the EfficientNet B0 model that will be trained in its entirety

Additionally, because of the increase in the number of parameters, the neural network is harder to train and as a result takes longer to do so. To keep the progress that was made during training and not lose it in case of a hardware failure, model checkpoint is introduced. Model checkpoint refers to the periodically saving of a model during training. That task is performed through a callback function that is passed as an argument in the fit_generator() function. First, the behaviour of the model checkpoint is configured by setting the save path, the metric to monitor in order to overwrite the model if that metric is improved and lastly the period of saving(number of epochs). By saving the model periodically, the training of a model can be resumed even if the training takes hours, days or even weeks.

The results for the EfficientNet B0 network that was fully trained on the balanced data are shown in figure 50 and 51.

224x224	batch size	epochs	optimizer	learning rate	dropout rate	loss	accuracy	val_loss	val_accuracy
	64	50	adam	0.001	0.2	0.102	0.9646	2.7564	0.4868

Figure 50. The resulted accuracy and loss for the fully trained EfficientNet B0 model

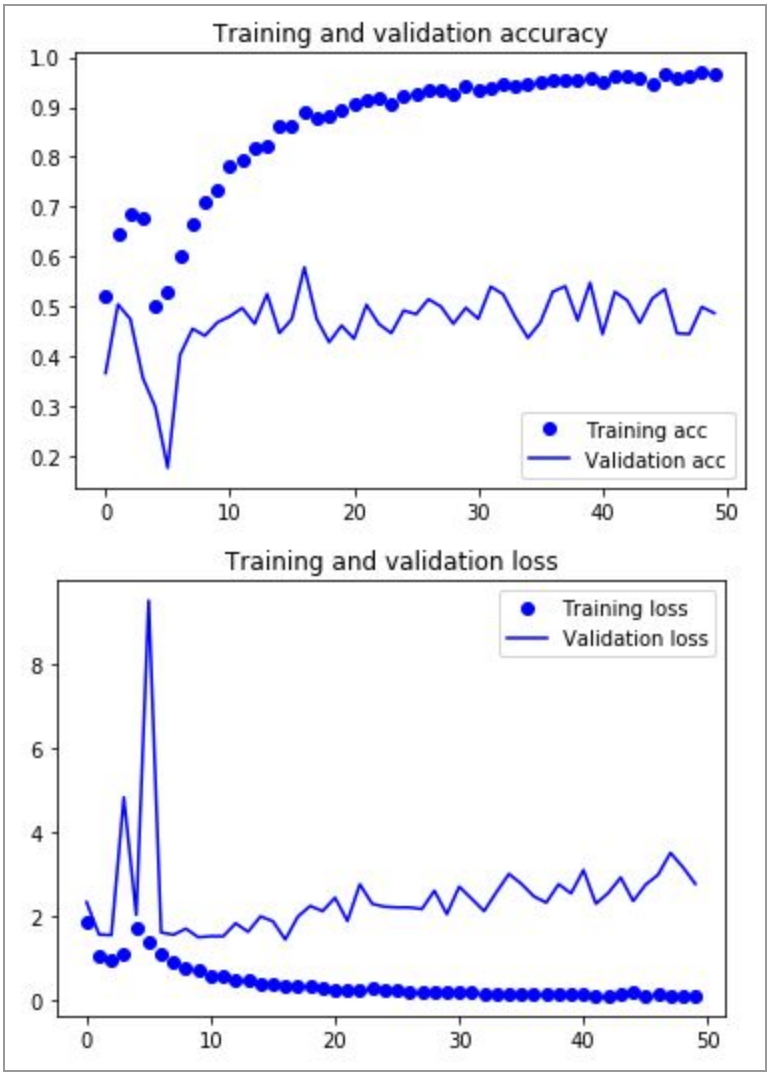


Figure 51. The resulted accuracy and loss curves for the fully trained EfficientNet B0 model

The produced graphs show increased accuracy on the training set but steady accuracy on the validation set with an increasing trend on the validation loss. These facts point to an overfitting behaviour additionally, the validation set is not balanced but rather imbalanced with the initial imbalanced training set distribution. The next experiment will check if the imbalanced validation set affects the scores of the model. To do so, the validation dataset will be converted to a balanced dataset with the number of images of the smallest class which is 48 images per class.

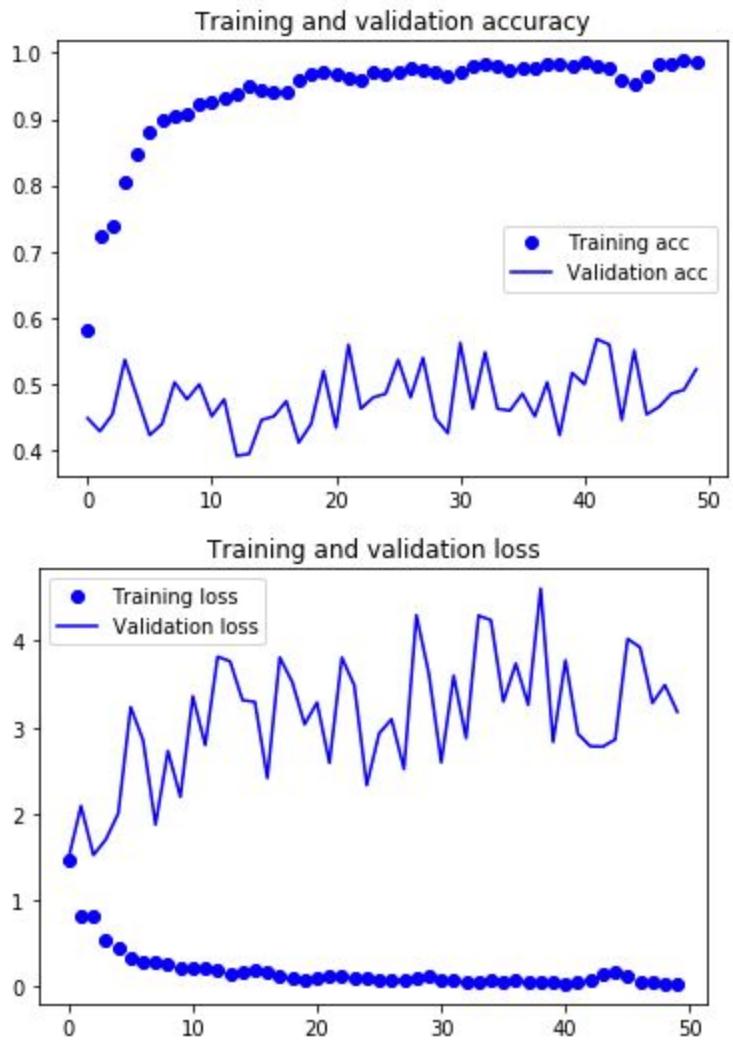


Figure 52. The resulted accuracy and loss curves for the fully trained EfficientNet B0 model with a balanced validation set

The resulting graphs (Figure 52) show that the network can again learn the training images by overfitting to them but the accuracy and loss on the validation set is not satisfactory (Figure 53).

224x224	batch size	epochs	optimizer	learning rate	dropout rate	loss	accuracy	val_loss	val_accuracy
	32	50	adam	0.001	0.2	0.0431	0.9862	3.1785	0.5227

Figure 53. The resulted accuracy and loss for the fully trained EfficientNet B0 model

The small size of the validation set (48 images per class), while being balanced, is something that presents a harder test on the network as the efficientNet has only 48 chances to be correct. That

fact suggests that the validation set needs to be a set percentage of the training set of the magnitude of ~ 0.15-0.25% to produce some viable results.

The next experiment will be with the initial imbalanced training dataset and an imbalanced validation dataset. The validation set is constructed by the ImageDatagenerator method of the Keras framework with a percentage of 0.2 and a stratified sampling. The resulting graphs are shown in figure 54.

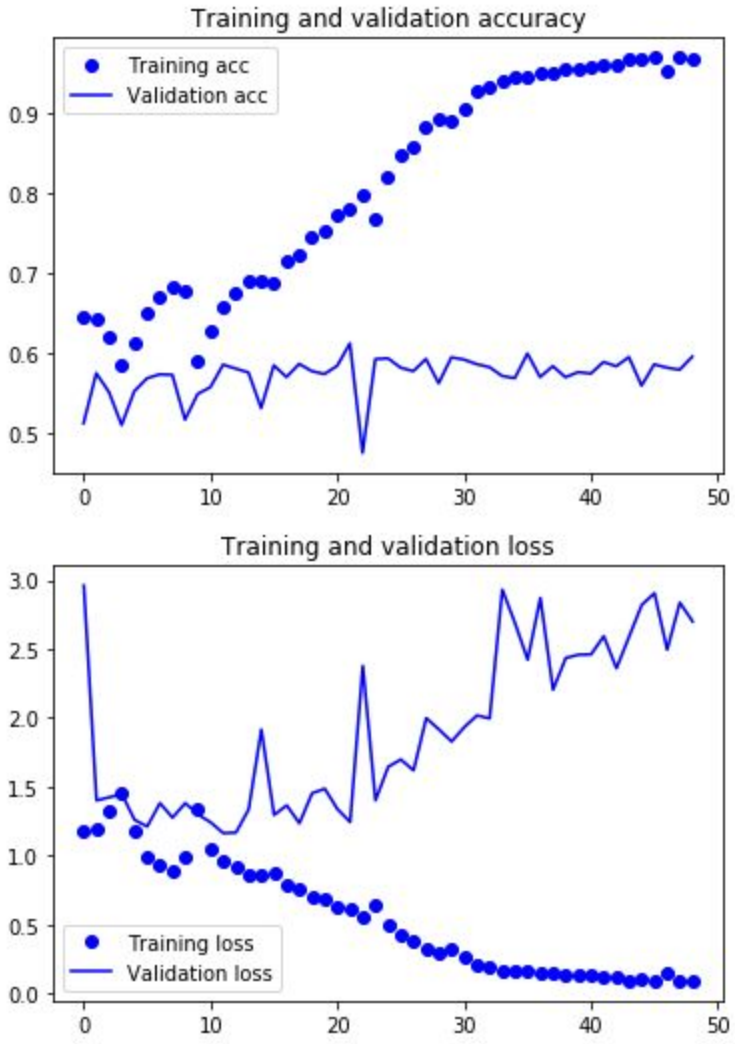


Figure 54. The resulted accuracy and loss curves for the fully trained EfficientNet B0 model

In this experiment the validation loss decreases as the epochs pass and the accuracy increases slightly until the model starts to overfit again. The final results of the accuracy and loss for the model are shown in figure 55.

224x224	batch size	epochs	optimizer	learning rate	dropout rate	loss	accuracy	val_loss	val_accuracy
	32	49	adam	0.001	0.2	0.0943	0.9686	2.7	0.5951

Figure 55. The resulted accuracy and loss for the fully trained EfficientNet B0 model

The next experiment will use the same initial imbalanced dataset with a 0.2 validation set, additionally the weights of the classes are used in the fit_generator method so that the model learns to pay attention to the smaller classes. The weights are produced through the class_weight method of the sklearn.utils module. After calculating the weights array, the array is converted to a dictionary so it can be inserted as an argument to the fit_generator method. The resulting graphs for the accuracy and loss of the training process are shown in figure 56.

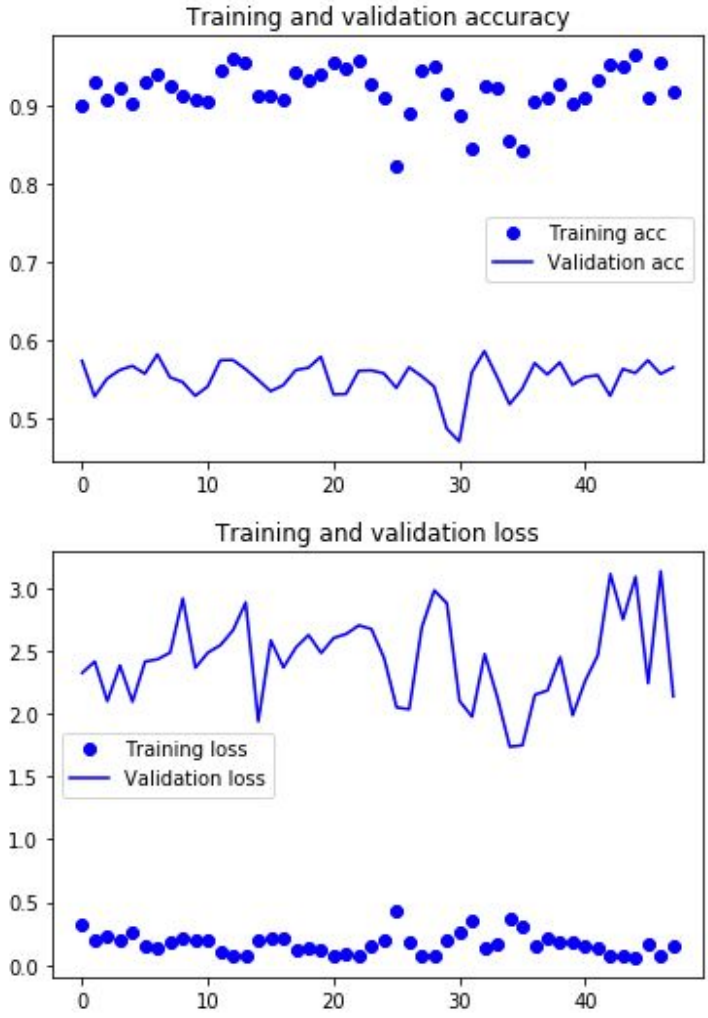


Figure 56. The resulted accuracy and loss curves for the fully trained EfficientNet B0 model

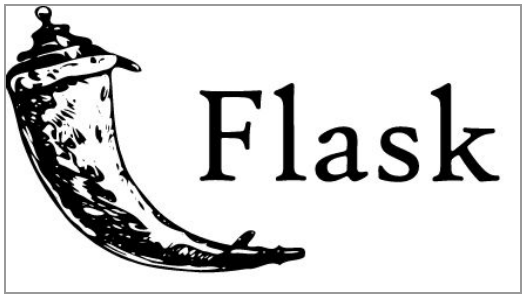
As expected the training accuracy and loss on the training set are worse than the previous experiments as the less represented classes are hurting the results of the model. The final results on the accuracy and loss of the model are shown in figure 57.

224x224	batch size	epochs	optimizer	learning rate	dropout rate	loss	accuracy	val_loss	val_accuracy
	32	48	adam	0.001	0.2	0.1588	0.9174	2.1375	0.5646

Figure 57. The resulted accuracy and loss for the fully trained EfficientNet B0 model

5.4. Skin lesion classification application (Mole Shaman)

In order to test the resulting model on a production environment and confirm its viability as an assistive diagnostic tool, a web app was developed that consumed the produced model and exposes a REST API endpoint to the user.



The framework that was used to develop the web app was the flask mini framework. Flask is a python framework following the Model View Controller (MVC) architecture that creates a model for the data entities, a controller that handles the API endpoint creation and a view that produces frontend functionalities for the user [35].

When the app loads a tensorflow graph is constructed in order to save it in a variable and reuse the same graph for each iteration of the model. That procedure is necessary so that the app knows which model to use. The main python file that starts the app is the server.py file, through that file the flask application is loaded and then the model gets loaded in the graph through the functions that are in the efficientnetB0_model.py. Specifically, the function load_EfficientnetB0() is executed when the application first loads so that the pre trained efficientNet B0 model, which is in a h5 format, gets initiated for prediction. The model is loaded with the use of the load_model() method of the keras framework.

The initial template the user is served is the home page html with the help of a GET HTTP method in the server.py file (Figure 58).

Figure 58. The homepage of the application(Mole Shaman) that prompts the user to choose an image file from the local filesystem and submit it for classifying

After rendering the home page the user has the option of uploading from the local filesystem an image of the skin lesion he wishes to diagnose (Figure 59). The application checks for nonexistent files and also for file extensions that do not correspond to actual image files. If the checks fail then an error page is presented to the user so he can try again. If the checks pass the application uses a POST HTTP method and executes the predict() function of the server.py file.

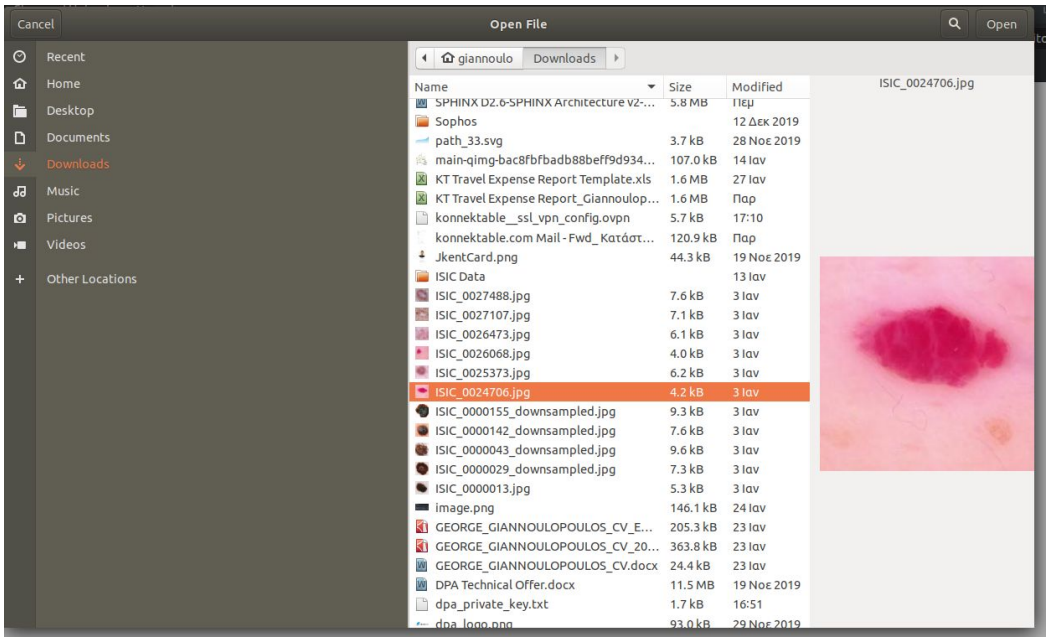


Figure 59. The image selection screen when the user presses the choose button in order to upload an image

The predict() function in turn calls the predict_class() function of the efficientnetB0_model.py file, which through the use of keras preprocessing method, resizes the

uploaded image to the dimensions that the model uses so it can be used as an input from the efficientNet model. Pixel values are then turned into floating points and are converted into a numpy array. After using that array as an input to the efficientNet model, the resulting probabilities for each class are produced and returned to the main python file (server.py). A template for displaying the results is constructed to render the results in a readable way (Figure 60).

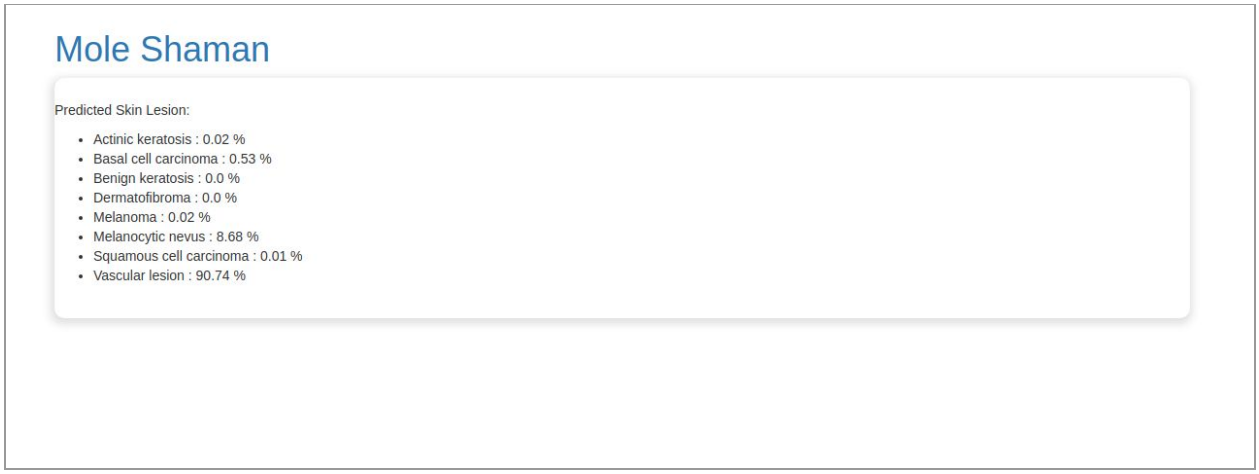


Figure 60. The multi-class predictions of the neural network model for the uploaded image

Chapter 6

Conclusion

6.1. Results

The top resulting trained EfficientNet B0 networks present a validation accuracy of 0.5851 - 0.5951 (Figure 61) which is significantly higher by the random classifier that has an accuracy of 1/k = 0.125, where k is the number of classes, but still the accuracy is not high enough to be used in a diagnosis production environment.

224x224	batch size	epochs	optimizer	learning rate	dropout rate	loss	accuracy	val_loss	val_accuracy
	32	49	adam	0.001	0.2	0.0943	0.9686	2.7	0.5951
	32	2	adam	0.001	0.2	0.9599	0.6577	1.162	0.5851

Figure 61. The top performing trained EfficientNet B0 models

Both models are trained on the initial imbalanced training dataset and are validated on the imbalanced validation dataset that was constructed with stratified sampling from the ImageDataGenerator class of the Keras framework. The first model is trained for the full 49 epochs which results into the model overfitting to the training data, that is why its accuracy is 96.6%, but not being able to generalise in a satisfactory manner on the validation dataset. The second model is produced through early stopping on the same experiment with the model not yet being overfitted. The validation accuracy of that model is lower than the fully trained model but the validation loss is considerably lower which means that the model is more confident in its predictions.

The resulting accuracy and loss of the transfer learning experiments that were executed show that the EfficientNet B0 model presents increased performance, on this particular task, when all the weights are fine tuned and not just the last layers of the network. This behaviour is to be expected, because of the differences of the ImageNet dataset that the model was trained on initially and the ISIC dataset. The structures that a convolutional neural network captures through the ImageNet dataset relate to an object classification problem while in skin lesion classification the structures ultimately need to approach the seven-point system (Figure 62) that the dermatologists use to classify a skin lesion.

The scoring seven-point system was adopted by the Cancer Research campaign in Scotland in the 1980s to help dermatologists diagnose early stage melanoma [39]. Early on, each feature of the seven-point system was scored with 1 thus making the resulting sum total to 7. The Scotish group recommended that lesions that had a total of 3 needed to be screened by a specialists and lesions that had a total of 4 or more were melanomas with a 90% accuracy. Later on in 1989, the seven-point system was modified to include features of major importance and minor ones with a weighted score to help the diagnosis. Again, a lesion of a total score of 3 or more needed an expert opinion in order to correctly classify the skin lesion.

ELM criterion	Score
Major criteria	
1. Atypical pigment network	2
2. Blue-whitish veil	2
3. Atypical vascular pattern	2
Minor criteria	
4. Irregular streaks	1
5. Irregular pigmentation	1
6. Irregular dots/globules	1
7. Regression structures	1

Figure 62. The seven point scale dermatologists use in order to classify a skin lesion as a melanoma lesion

Dataset balancing, that was performed in various experiments, was proven to be ineffective in increasing the performance of the EfficientNet B0 model. That fact can be attributed to the relative small size of the dataset for this particular problem as the classes of the image prediction are great (8) and the differences between them are minimal. Consequently, when trying to balance the classes the two options were subsampling the majority classes or augment the minority ones which led to very few examples in the first case and almost duplicate examples in the second case.

Lastly, the experiment with using transfer learning on the EfficientNet B5 model did not present increased performance which probably can be attributed to the fact that the training was done on the last layers only and not the network in its entirety.

6.2. Future Work

The next steps towards increasing the accuracy of the classifier would be the training of bigger neural networks in the EfficientNet family as the increased image resolution of the input images could help expose the microstructures that the network needs to capture. So far the

experiment on the EfficientNet B5 model showed that the network needs to be fine tuned as a whole in order to test its accuracy which is a resource consuming task but it might yield better results.

Additionally, another technique used for increasing the accuracy of the classification is the forming of an ensemble of neural networks, either only efficientNets or an ensemble of various architectures that all together vote for the most accurate prediction. By examining the leaderboard of the ISIC challenge, this option seems to be an effective one as the top scorers use an ensemble of networks. An ensemble of weak classifiers can provide the final verdict the correct structure predictions that are needed for a correct classification, structures that are detected by different networks.

Lastly, various improvements on the dataset can increase the classification accuracy. Improvements like increased dataset size by collecting additional high resolution dermoscopic images from diagnostic centers and hospitals or by combining patient metadata with the images. Apart from increased size, the dataset needs to be comprised of images that follow a capture protocol in order to simplify the problem. A standardized process of capturing the images by setting the same brightness conditions and distance needs to be applied on all training data as well as testing data. Lastly, some data preprocessing can also be performed on the dataset in order to remove structures that are not needed for classification like hair or skin that does not belong to the skin lesion. The removal of skin that does not belong to the skin lesion could be performed by training a network that predicts the boundaries of the lesion (e.g. Mask-RCNN [37]) and subtracting the excess pixels from the original input image.

References

1. Skin Cancer Facts & Statistics - The Skin Cancer Foundation [Internet]. The Skin Cancer Foundation. 2020 [cited 16 February 2020]. Available from: https://www.skincancer.org/skin-cancer-information/skin-cancer-facts

2. Melanoma Skin Cancer Risk Factors | Melanoma Risk Factors [Internet]. Cancer.org. 2020 [cited 16 February 2020]. Available from: https://www.cancer.org/cancer/melanoma-skin-cancer/causes-risks-prevention/risk-factors.html

3. Dermoscopy | DermNet NZ [Internet]. Dermnetnz.org. 2020 [cited 16 February 2020]. Available from: https://dermnetnz.org/topics/dermoscopy/

4. Samuel AL. Some studies in machine learning using the game of checkers. II—Recent progress. IBM Journal of research and development. 1967 Nov;11(6):601-17.

5. Mitchell T. Machine learning. New York: McGraw-Hill; 2013.

6. Zhu X, Goldberg AB. Introduction to semi-supervised learning. Synthesis lectures on artificial intelligence and machine learning. 2009 Jun 8;3(1):1-30.

7. Sutton RS, Barto AG. Introduction to reinforcement learning. Cambridge: MIT press; 1998 Mar 1.

8. Nielsen M. Neural Networks and Deep Learning [Internet]. Neuralnetworksanddeeplearning.com. 2020 [cited 16 February 2020]. Available from: http://neuralnetworksanddeeplearning.com/

9. Ruder S. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747. 2016 Sep 15.

10. Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. nature. 1986 Oct;323(6088):533-6.

11. Szeliski R. Computer vision: algorithms and applications. Springer Science & Business Media; 2010 Sep 30.

12. Turk M, Pentland A. Face recognition using eigenfaces. InProceedings. 1991 IEEE computer society conference on computer vision and pattern recognition 1991 Jan 1 (pp. 586-587).

13. Dalal N, Triggs B. Histograms of oriented gradients for human detection. In2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05) 2005 Jun 25 (Vol. 1, pp. 886-893). IEEE.

14. Fukushima K. Neocognitron: A hierarchical neural network capable of visual pattern recognition. Neural networks. 1988 Jan 1;1(2):119-30.

15. LeCun Y, Boser B, Denker JS, Henderson D, Howard RE, Hubbard W, Jackel LD. Backpropagation applied to handwritten zip code recognition. Neural computation. 1989 Dec;1(4):541-51.

16. LeCun Y. LeNet-5, convolutional neural networks. URL: http://yann. lecun. com/exdb/lenet. 2015;20:5.

17. Bengio Y. Practical recommendations for gradient-based training of deep architectures. InNeural networks: Tricks of the trade 2012 (pp. 437-478). Springer, Berlin, Heidelberg.

18. Li M, Zhang T, Chen Y, Smola AJ. Efficient mini-batch training for stochastic optimization. InProceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining 2014 Aug 24 (pp. 661-670).

19. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research. 2014 Jan 1;15(1):1929-58.

20. Brinker TJ, Hekler A, Enk AH, Klode J, Hauschild A, Berking C, Schilling B, Haferkamp S, Schadendorf D, Holland-Letz T, Utikal JS. Deep learning outperformed 136 of 157 dermatologists in a head-to-head dermoscopic melanoma image classification task. European Journal of Cancer. 2019 May 1;113:47-54.

21. He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. InProceedings of the IEEE conference on computer vision and pattern recognition 2016 (pp. 770-778).

22. Lopez AR, Giro-i-Nieto X, Burdick J, Marques O. Skin lesion classification from dermoscopic images using deep learning techniques. In2017 13th IASTED international conference on biomedical engineering (BioMed) 2017 Feb 20 (pp. 49-54). IEEE.

23. Mahbod A, Schaefer G, Wang C, Ecker R, Ellinge I. Skin lesion classification using hybrid deep neural networks. InICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2019 May 12 (pp. 1229-1233). IEEE.

24. Tan M, Le QV. Efficientnet: Rethinking model scaling for convolutional neural networks. arXiv preprint arXiv:1905.11946. 2019 May 28.

25. Tan M, Chen B, Pang R, Vasudevan V, Sandler M, Howard A, Le QV. Mnasnet: Platform-aware neural architecture search for mobile. InProceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2019 (pp. 2820-2828).

26. Zoph B, Le QV. Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578. 2016 Nov 5.

27. Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861. 2017 Apr 17.

28. A Comprehensive Introduction to Different Types of Convolutions in Deep Learning [Internet]. Medium. 2020 [cited 16 February 2020]. Available from:

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

29. Krizhevsky A, Hinton G. Convolutional deep belief networks on cifar-10. Unpublished manuscript. 2010 Aug;40(7):1-9.

30. ISIC 2019 [Internet]. Challenge2019.isic-archive.com. 2020 [cited 16 February 2020]. Available from: https://challenge2019.isic-archive.com/

31. Home - Keras Documentation [Internet]. Keras.io. 2020 [cited 16 February 2020]. Available from: https://keras.io/

32. Colaboratory – Google [Internet]. Research.google.com. 2020 [cited 16 February 2020]. Available from: https://research.google.com/colaboratory/faq.html

33. ImageNet [Internet]. Image-net.org. 2020 [cited 16 February 2020]. Available from: http://www.image-net.org/

34. Perez L, Wang J. The effectiveness of data augmentation in image classification using deep learning. arXiv preprint arXiv:1712.04621. 2017 Dec 13.

35. Welcome to Flask — Flask Documentation (1.1.x) [Internet]. Flask.palletsprojects.com. 2020 [cited 16 February 2020]. Available from: https://flask.palletsprojects.com/en/1.1.x/

36. Giannoulo/Skin-lesion-classifier [Internet]. GitHub. 2020 [cited 22 February 2020]. Available from: https://github.com/Giannoulo/Skin-lesion-classifier

37. He K, Gkioxari G, Dollár P, Girshick R. Mask r-cnn. InProceedings of the IEEE international conference on computer vision 2017 (pp. 2961-2969).

38. MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges [Internet]. Yann.lecun.com. 2020 [cited 28 February 2020]. Available from: http://yann.lecun.com/exdb/mnist/

39. Bishop J, Gore M. Melanoma. New York, NY: John Wiley & Sons; 2008.

40. Pankajavalli P, Karthick G. Incorporating the internet of things in healthcare applications and wearable devices.

41. Ahuja A. The impact of artificial intelligence in medicine on the future role of the physician. PeerJ. 2019;7:e7702.