Final Year Project

Skin cancer classification using deep learning

Author:
Aliahmed JUNEAD

Supervisor:
Dr Nikolay NIKOLAEV

A thesis submitted in fulfilment of the requirements for BSc Computer Science Degree

May 6, 2022

Declaration of Authorship

I, Aliahmed JUNEAD, declare that this thesis titled, “Skin cancer classification using deep learning” and the work presented in it are my own. I confirm that:

- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed: Aliahmed JUNEAD

Date: May 6, 2022

Abstract

BSc Computer Science Degree

Benign and Malignant skin cancer classification & detection using deep learning

By Aliahmed JUNEAD

Skin cancer accounts for 2% of all cancers worldwide, and 20% of the U.S. may develop skin cancer by 70. Tumours appearing in the outer layers of skin can be cancerous or noncancerous. Early diagnosis is a critical step in determining the correct procedure for survival. Oncologists are qualified to identify types of skin disorders. However, misdiagnosis occurs when harmful skin conditions are mistaken for lesser ones, e.g. eczema. AI is increasingly explored in medicine, prevailing in diagnosis & achieving high accuracy with limited errors. AI models can learn diagnostic skills surpassing humans using vast amounts of data. The project aims to provide a methodology to classify nine skin lesions with a CNN algorithm using, conducting informative architectural experimentation, using data created by International Skin Imaging Collaboration (ISIC), which consists of over 2000 images belonging to 9 different categories, 2 of which are cancerous.

Acknowledgements

First and foremost I am utmost abundantly grateful to Allah (glory be to him), and all praise belongs to him. I would like to thank my supportive parents for their ongoing support, inspiration and important life lessons I have learnt about resilience which have given me tools to embark on this degree with courage. Finally, I would like to thank Nikolay NIKOLAEV, for being able to share some of his extensive knowledge and experience as a form of guidance for my project.

Contents

Keywords and abbreviations

Convolutional neural networks, Deep learning, neural networks, Artificial Intelligence, skin cancer, machine learning, malignant, benign, skin lesions.

CNN	*Convolutional Neural Network*
ANN	*Artificial Neural Network*
DL	*Deep learning*
AI	*Artificial Intelligence*
ML	*Machine Learning*
MLP	*Multi-layer Perceptrons*
FC	*Fully Connected*
ReLu	*Rectifier Linear Unit*

1. Introduction

1.1. Overview

The project aims to employ a DL model using CNN architecture and conduct experimentation, classifying distinct skin lesions into nine classifications. The objective is to conduct methodological investigations, analysing the effects of tuning techniques on the performance of the CNN on unseen data.

1.2. Motivation

Rapid growth in AI due to extensive research expanded into different sectors such as healthcare & medicine. With AI being increasingly experimented with and introduced into medicine, many AI services outdo human specialists. An autonomous diagnosis of pneumonia, *"CheXNet"*, was developed in 2017, using 100,000 X-ray chest images of 14 diseases. Compared to 4 radiologists, the CNN proved to be more accurate [1]. Effective categorisation of medical images in modern medicine is critical, hence the need for autonomous medical image classification. CNN models have performance levels that rival human expertise [2]. As a result, this initiative falls within the category of medicine. It serves as an example of how AI may be used to meet a need, albeit on a smaller scale, but with significant promise.

1.3. Project Scope

The CNN model will be using a dataset of 9 classes of skin lesions, this would be the most efficient form of training a CNN model, having a large dataset would be computationally inefficient. Although there are countless types of skin lesions and cancers, including a dataset with a greater number of classes would be a way to further the project.

1.4. Tools

Python

An object-oriented programming language, Python, is commonly used in data science, used for ML and AI due to its access to libraries and frameworks for AI, such as Keras, Numpy, Scikit-learn and Tensorflow.

Tensorflow

Developed by Google, Tensorflow is an AI library with a dense repository of ML tools, allowing for building NN's without defining and specifying intricacies such as derivatives of backpropagation.

Keras

An open-source API with Python interface, it runs on top of TensorFlow library, consisting of different types of layers that enable the freedom to build NN's.

Jupyter notebook

An online coding platform that allows for coding executable Python code whilst combining with markdown text.

1.5. Results

The CNN led to the accuracy of 90.1% when evaluated on validation data, and testing it on a test set resulted in 73.7%.

1.6. Report overview

Chapter 2 discusses **background** knowledge about the project, current state of medicine & background information about NN such as CNN's and MLP's.

Chapter 3 discusses the architecture of the CNN model, describing the choices for structural decisions with a methodical approach.

Chapter 4 will **implement** the methodology, discuss findings from experimentations and explain **results**.

Chapter 5 is a **conclusion** of the report and comprehensively explains the project's achievements in this project.

2. Background review

The report section will fulfil the necessary background research I conducted about the topics covered. The section will break down; skin cancer & diagnosis, AI background research, and components of the CNN algorithm.

2.1. Skin cancer

2.1.1. What is skin cancer?

The skin, the body's largest organ, provides vital protection against injury, regulating body temperature. Doctors in the U.S. diagnose 3 million people with skin cancer each year, making it common [3]. It occurs in the epidermis of the skin, the outermost layer, whereby abnormal cells grow at an alarming rate [4]. Tumours are either *benign* or *malignant,* noncancerous & cancerous, respectively. Benign tumours do not spread, whereas Malignant tumours proliferate, destroying body tissue as it spreads [5].

Figure 2.1 Melanoma (left, malignant), Nevus (right, benign)[71].

2.1.2. Diagnosis

Diagnosis gives understanding of how likely treatment is to be successful [6]. *Staging* describes determining the structural position of disease [7]. AJCC TNM staging is a numerical 6 stage staging system used for staging Melanoma skin cancer (a malignant and most harmful type of skin cancer) [8]. The SEER database has three categories of stages: *Localised, regional, and distant* [6]. Nonetheless, they achieve the same principle of determining how severe the disease is at the point of diagnosis.

2.1.3. Misdiagnosis in medical practices

Oncologists & dermatologists diagnose skin deformities into respective categories. However, misdiagnosis occurs, and severe conditions may become diagnosed as something less harmful. 48.2% of misdiagnoses occurred at a paediatric tertiary care centre at Khon Kaen University [9]. Furthermore, Singh H *et al.* estimated that 5.08% of diagnosis errors occur every year, affecting 12 million adults in the US[10]. Additionally, Academic, community and training paediatricians complete a survey on frequency of diagnostic errors. Over half of participants said diagnostic errors occur at least once or twice a year that may lead to harm in

patients [11]. In the United States, 47% of graduating dermatologists find it uneasy diagnosing skin diseases of patients with darker skin [42][70].

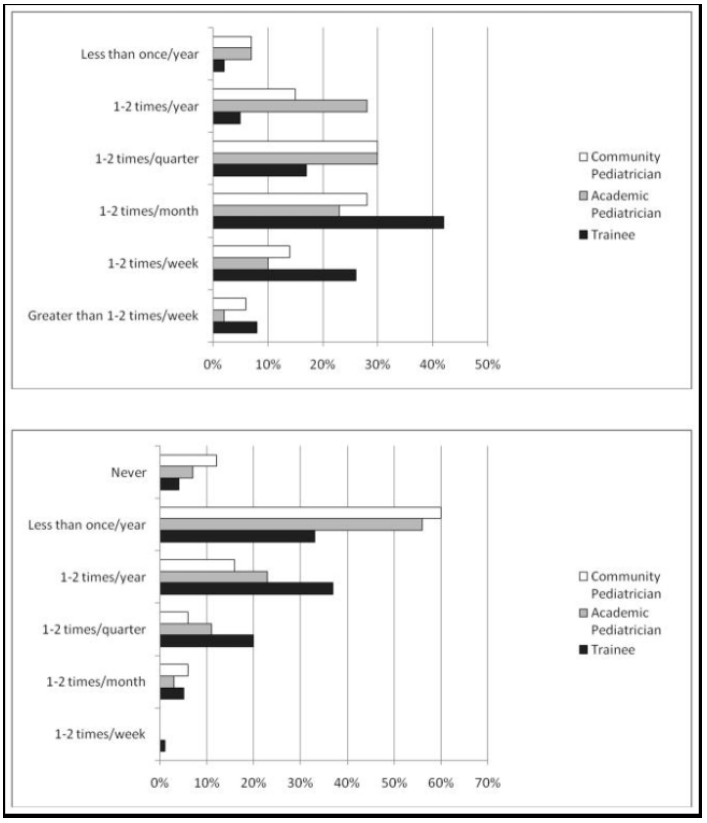


Figure 2.2 frequency of diagnostic errors that caused patients to harm [11]

2.1.4. AI and healthcare

We are beginning to see more applications of AI in healthcare & medicine. Some examples include using DL to form a novel method of early diagnosis for Alzheimer's disease using data collated from Alzheimer's MRI images [12].

Availability & easy access to large amounts of data accumulated online has allowed for highly accurate autonomous diagnosis, which is collected in healthcare [13]. Opportunities provided by EPR (electronic patient records) and medical datasets pose challenges to data sharing, resulting in possible negative biases and ethical issues with public data being sourced from social media and other media on the internet [14][15].

2.2. AI, ML & DL

Pioneered in the 1950s as an effort to automate tasks performed by humans, *artificial intelligence* is defined as the computational ability to achieve goals, which is what is automated [16]. Early chess programmes, for example, had hard-coded rules, exemplifying symbolic AI [17].

ML is a subset of AI; it approaches symbolic AI differently. It investigates the capability of having a computer learn rules autonomously, without providing data and the outcome to return rules required to achieve the outcome. As a result, human experts determine the features to comprehend the differences between data inputs [18].

DL is about a NN with multiple layers that emphasise learning from layers of increasingly meaningful representations. Deep learning models, as opposed to ML, can learn from hundreds of successive layers [19]. DL algorithms are now used for complex problems such as image classification and speech recognition.

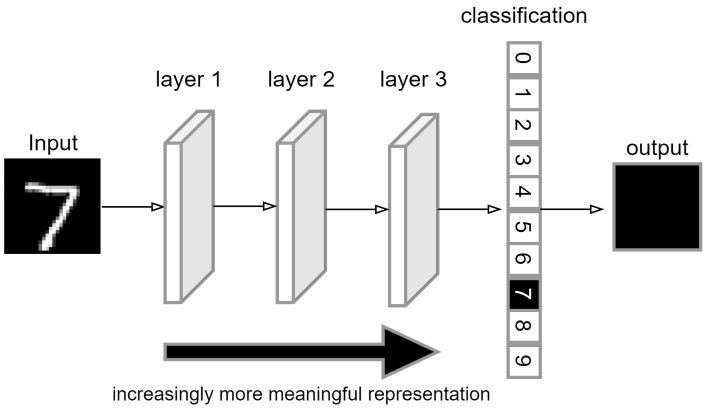


Figure 2.3 a deep neural network showing layers for handwritten digit classification

Each layer represents more meaningful representations that help to classify the image into correct outcomes.

2.3. MLP (Multilayer perceptron)

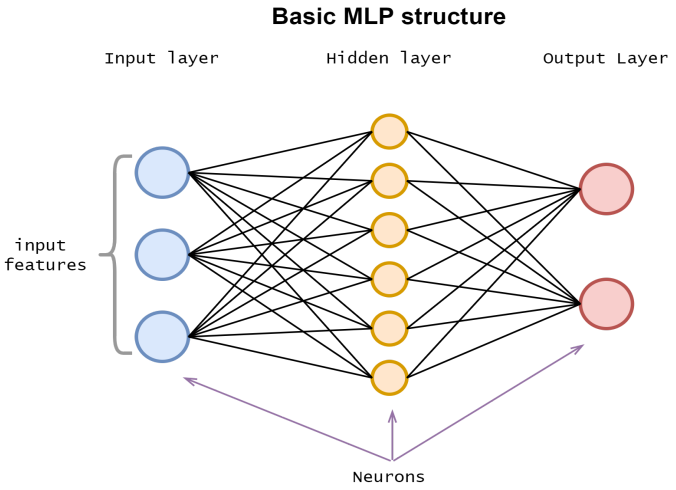


Figure 2.4 MLP architecture

Multilayer Perceptron, a feedforward NN, is an algorithm used in computer vision tasks. This NN consists of input, output, and hidden layers, taking the imagined input data as a vector. The most basic structure of an MLP consists of simply an input, output and one hidden layer between them [73][74]. Like CNN, its learning is done by backpropagation, although not as deep and is far shallower, hence having fewer learning capabilities.

The primary difference between MLPs and CNNs is that the latter is more equipped for image classification of detailed images. MLP, on the other hand, is better suited to less convoluted images. This is because CNN uses tensor inputs, which allow it to understand spatial relationships between pixels in an image [72], whilst MLPs take vectorised inputs. Information flows through an MLP via functional and error signals. The forward pass describes how a functional signal moves input in a forward direction from the input to hidden layers while processing the input with an activation function [75].

2.4. CNN

CNN is an architecture commonly used in computer vision tasks, specifically image classifications. It is excellent for feature extraction of images where convolutions are applied to input data; digital filters perform convolutional operations [2]. Therefore, its powerful analysis capabilities for detecting patterns of abstraction from images is an important reason I opted for a CNN architecture for this project [24][25].

CNN's comprise an input, an output, and hidden layers, but convolutional layers set them apart. Convolutional layers include three-dimensional output volume, sample number, and shape [26]. F. Chollet explains that fully-connected layers learn global patterns from given

inputs, such as patterns of all pixels [19]. Hence, CNNs are effective at image classification and capable of learning unique patterns in different 'locations' of an abstract visual problem [19].

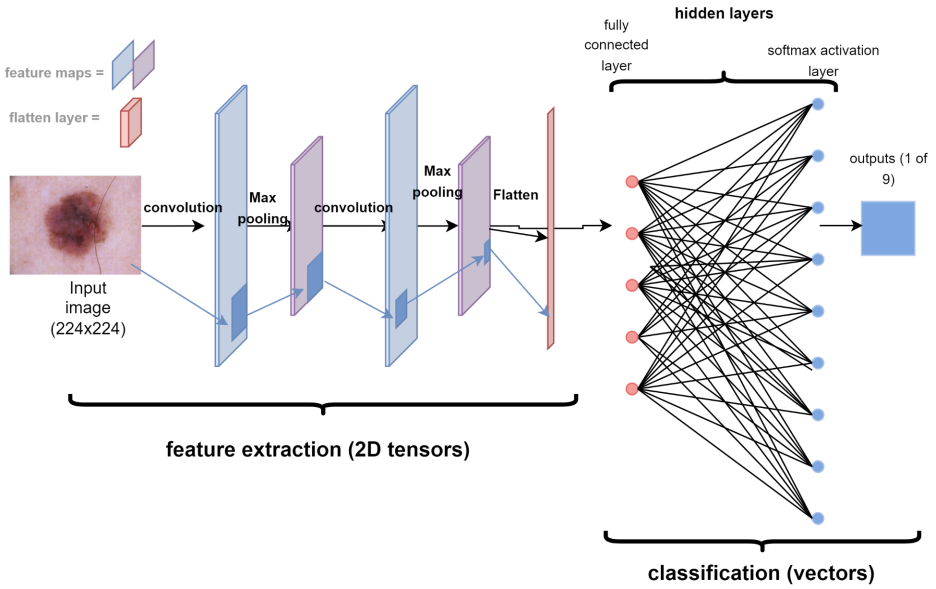


Figure 2.5 CNN architecture

2.4.1. Convolutional layer

Convolutional layers work by applying a filter to an input, extracting information from the input image[60]. The filter, made up of kernels, is applied to all parts of the input image, producing a tensor of feature maps [25]. A kernel is a matrix (2D) of weights which multiplies with the input layer to output extracted features, whereas filters are composed of multiple kernels (3D)[18][91]. In image classification problems, convolutions would be applied to a 2D image, outputting a 3D tensor of shape (height x width x channels). Convolutional layers learn local patterns, locating the same pattern anywhere else on an image, taking into account spatial hierarchical features of an image, specific features of an image, and understanding the arrangement of pixels with context[19][27]. Convolutional layers are not densely connected, unlike hidden layers, allowing for high-dimensional inputs such as images to be processed along the network [126].

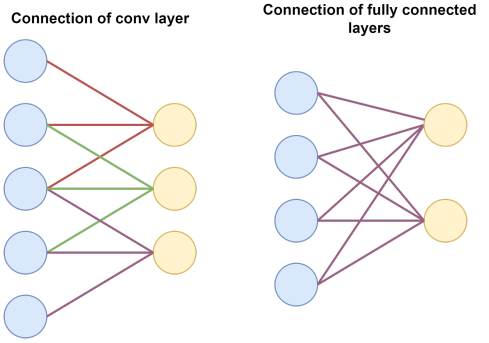


Figure 2.6 nodes of convolutional layer or not densely connected unlike dense layers

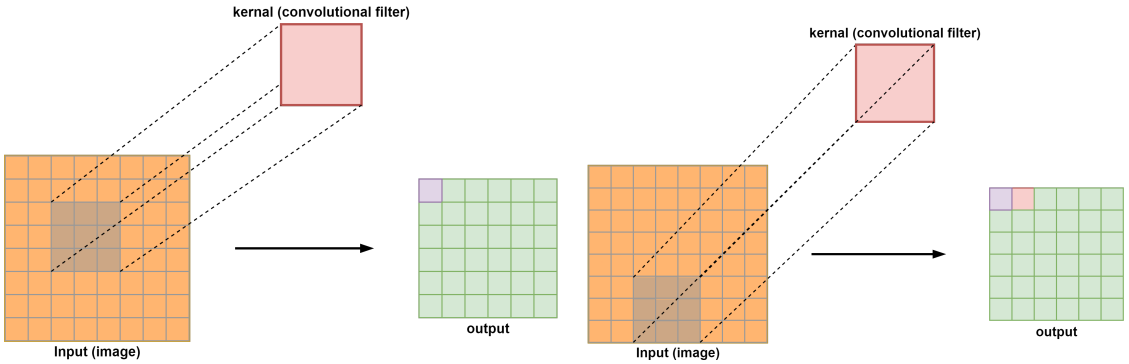


Figure 2.7 2D convolution: 1st feature map of convolution using 3x3 filter

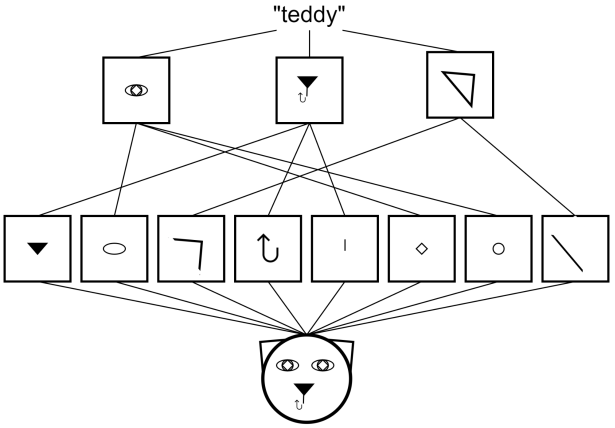


Figure 2.8 spatial hierarchy of an image of a teddy, split into various features

2.4.2. Loss function (cost functions)

The purpose is to calculate the error between the *'actual output'* and NN's predicted output [82]. The type of loss function depends on the nature of the ML problem. For multi-class problems such as problems classifying two or more classes, cross-entropy would be used. In

regression ML problems, MeanSquaredError (MSE) is commonly used. However, regression problems predict continuous outputs, whereas classification predicts distinct classes. Thus there are types of loss functions available for both types of problems.

Cross-Entropy Loss

Cross-entropy is the loss function used for multi-class & binary classification problems. Using cross-entropy loss trains a CNN to output the probability of classes, calculated by finding the difference between the predicted outcome and the actual outcome of inputs[52]. The two formulas are a cross-entropy loss for multi-class and binary classes [122].

Multi-class cross-entropy:

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{1}$$

Binary cross-entropy:

$$-(y \log(p) + (1 - y) \log(1 - p)) \tag{2}$$

M - number of classes

og - the natural log

y - binary indicator (0 or 1) if class ***c*** is the correct classification for observation ***o***

p - predicted probability observation ***o*** is of class ***c***

[122]

2.4.3. Activation functions

Activation functions assist in learning complex patterns from data and simply transforming inputs into outputs [56][59]. It computes a "weighted sum" of its input and adds a bias [57], introducing non-linearity to the learning step and is present in layers except the output. A network can successfully approximate functions that do not follow linearity or predict classes of a function that is divided by a nonlinear decision boundary. Non-linearity is significant in a network, without it; a multi-layered NNs would behave like a single-layer perceptron [84][85].

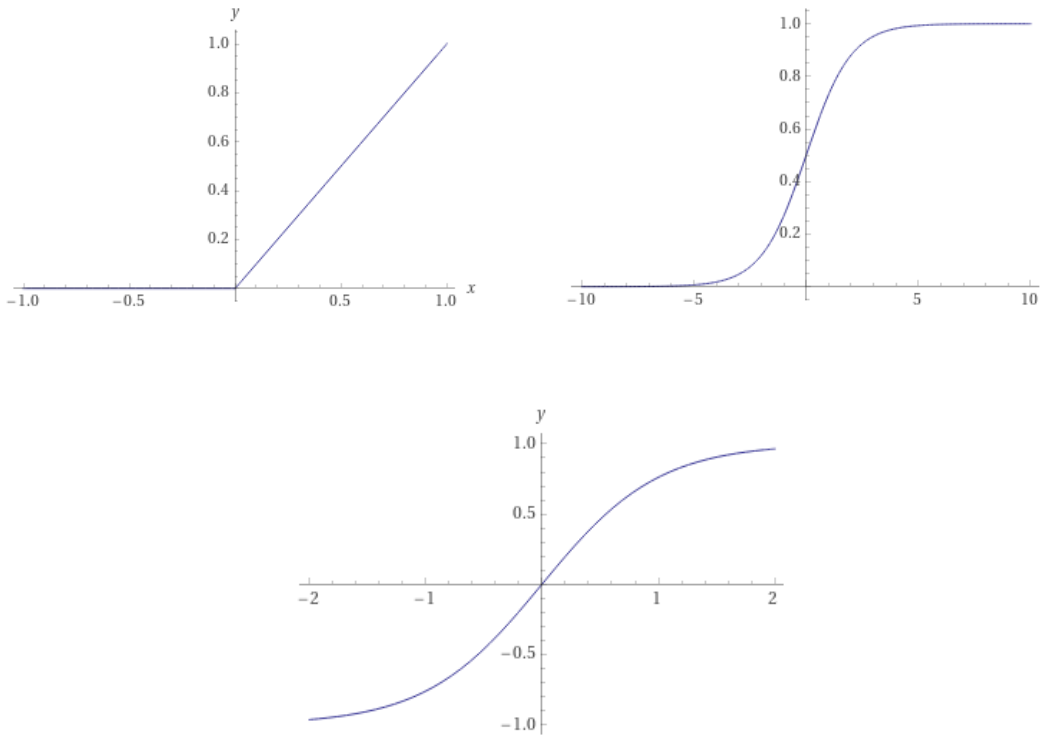


Figure 2.9 a graph of a ReLu (top left), Sigmoid (top-right) and Tanh (bottom)

ReLu activation function

ReLu, a non-linear activation function. It is computationally cheap, with less rigorous maths, making it more straightforward and reducing running time [58]. ReLus computational efficiency handles negative values of neurons as if they are positives, hence making any negatives a zero [87]. ReLU helps improve networks, speeding up training due to its representation sparsity, allowing activation of hidden layers in neural networks to contain one or more true zero values [86].

It is mathematically represented as

$$f(x) = max(0, x) \tag{3}$$

Sigmoid activation function

The larger the input, the closer the output to 1. Likewise, the smaller it is, the closer it will be to 0[57][62][119].

$$S(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

This function is used in ML models when a probability is predicted as an output, hence can also be used in the output layer. The probability of anything is between 0 and 1 [61]. The

probability of anything is equal to or greater than 0 and less than or equal to 1. Mathematically presented as (where $P(A)$ means ' probability of anything'):

$$0 \leq P(A) \leq 1 \tag{5}$$

Henceforth, sigmoid has a range of 0 to 1, making it useful for providing an output for probability prediction [62], making it ideal for binomial distribution problems, which are binary class problems.

TanH activation function

Similarly to the Sigmoid function, tanh (hyperbolic tangent) follows a similar shape, but the output ranges from -1 to 1.

$$T(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{6}$$

The advantages are that it produces a "zero-centred output" supporting the backpropagation process and is widely used in speech recognition [63]. Zero-centred allows output values to be mapped as strongly negative, strongly positive, or neutral (-1,0,1). Hidden layers use this activation to centre the data, making learning for subsequent layers easier [62].

Softmax activation function

Softmax is used exclusively for the output layer of models that predict the probability of multinomial distribution. Multinomial distribution in ML is the probability of an outcome or class. The softmax function returns the probability of each class, and it will provide the probability of each class, summing to a value of 1 [88][112]. Sigmoid and Softmax differ; the former is used in binary classification, and the latter is better suited for multivariate classification [63].

$$f(S)_i = \frac{e^{s_i}}{\sum_{j=1}^{K} e^{s_j}} \tag{7}$$

$f(S)_i$ = softmax the input vector

e^{s_i} = standard exponential of input vector, the input vector may come in positive, zero, or negative values (such as *-0,56, 5.2, 0*), these are not valid probabilistically. Therefore softmax value would correct it.

$\sum_{j=1}^{K} e^{s_j}$ = summation, ascertains that the output values will be all a sum of 1.

K = number of classes
e^{s_j} = standard exponential of output vector

2.4.4. Optimisers

The optimiser's role is to ensure that loss functions achieve their global minima and concentrate on speeding up the optimisation process by reducing the number of function evaluations required to find local minima, which modifies the learning rate of a NN, using back propagation to calculate gradients.

Gradient descent

Gradient descent is a first-order optimisation algorithm used to find the local minimum and maximum of a function [49]. Mathematically, "gradient" is a vector in reference to directions that have the largest value at one point [50]. It is a way to measure gradient to see the change in weights concerning change in error 51.

$$b = a - \gamma \nabla f(a) \tag{8}$$

$b = newposition$
$a = currentposition$
$\Delta f(a) = directionofsteepestdescent$

This algorithm is used to minimise loss of a network, and the gradient of the loss function provides the direction in which the function has the steepest rate of success; the higher the gradient, the faster the model learns.[51][82]

RMSprop

RMSprop (root mean squared propagation) is an altered version of AdaGrad optimisation. AdaGrad is an extension of Gradient Descent, taking a cumulative sum of all squared gradients of previous steps, hence having different learning rates for each parameter[116]. RMSprop works similarly but uses a decaying partial gradient, as opposed to using an aggregate of gradients like AdaGrad, maintaining a decaying average of gradients [78][117].

Adam

An adaptive learning rate optimisation algorithm is also an extension of gradient descent, Adam involves a combination of momentum and RMSprop. A Technique used in SGD

(Stochastic Gradient descent), momentum accumulates an exponentially weighted average of gradients which converges towards the minima at a fast pace. Adam takes both methods to reach a global minimum efficiently.[101][118]

2.4.5. Forward propagation

When an input is fed into a network, hidden layers process the input and pass it through the network to the following layers in a forward direction. This is known as a feed-forward network, where information about the input treacles to the output layer. The input is passed through the network as neurons, processed by activations before moving to the next layer. [124]

Each activation is denoted by $a^{(layer\ +\ 1)}$ = $activation^{(layer)}$, W = weight of previous layer activation, b = bias, x is the activation function used (e.g. relu) . The formula calculates the activation of each neuron for layer 2 of any generic network. [124]

$$a^1 = x(Wa^0 + b) \tag{9}$$

Once the neurons have been computed to the last layer, this activates *backwards-pass,* where the back propagation algorithm will update weights and biases on neurons from the last layer backwards.

2.4.6. Backpropagation

Backpropagation helps calculate the gradient of parameters of a network of a loss function regarding all weights in the network [29]. In CNN, said parameters are kernels and biases, and the gradients come from parameters of FC layers using the chain rule. Ultimately, the backpropagation algorithm is used to train the algorithm implicitly, as the optimiser uses the gradients the algorithm calculates to train the network [48].

Backpropagation starts with the output layer, updating the weights & biases of neurons in the network; weights and biases are connected to each neuron [124]. The chain rule is used to calculate partial derivatives; a shortened form of the entire mathematical notation which calculates derivatives of loss functions using Leibniz's notation [30,120]:

$$\frac{dZ}{dX} = \frac{dZ}{dY} \cdot \frac{dY}{dX} \tag{10}$$

2.4.7. Accuracy

Accuracy is a metric used to calculate how often a network is likely to make correct predictions. A simple formula for accuracy:

$$\frac{correctpredictions}{totalpredictions} \tag{11}$$

The accuracy is calculated by looking at the number of true-positives (correct predictions of positive class), true-negatives (correct predictions of negative class), false-positives (incorrect prediction of positive class) and false-negatives (incorrect prediction of negative class). TP and TN are correct predictions, TP, FP, TN, FN are all predictions made [125].

$$accuracy = \frac{TP+TN}{TP+FP+TN+FN} \tag{12}$$

2.5. Controversies and ethics

AI skin cancer detection models may be less accurate for darker complexions [31] due to absence of data; most skin cancer datasets are of overwhelming Caucasian samples only [32]. Without a diverse dataset, algorithms will not work on a diverse public [43][44]. The lack of representative data also resulted in AI classifying black models as gorillas and deeming them as less attractive [45][46]. Moreover, research on disparities finds that melanoma survival rates were significantly worse for black patients [34][35][36].

Such racial disparities are attributed to factors, such as lesions are in acral locations of the body of black patients [37], resulting in greater frequency of misdiagnoses [38]. The field of dermatology has had decades of clinical research focused on people with light skin, omitting minorities [39][47]. Black skin is used more in textbooks to depict sexually transmitted diseases [41]. Lipoff found that upto 18% is represented in dermatology textbooks [40][41].

2.6. Previous works

In 2018 researchers developed a CNN model using 100,000 images of cancerous skin samples, achieving accuracy in detecting skin cancer greater than experienced dermatologists,

which was achieved after broad hyperparameter regularisation [65]. Similarly, MobileNet learnt over 3000 images using sampling and pre-processing methods, and achieved an accuracy of 94% [64].

Moreover, researchers in Germany developed an autonomous diagnosis CNN model, detecting melanoma. The CNN outperformed 58 dermatologists; experts ranging from 17 countries, although 13 (22.4%) showed slightly higher diagnostic performance than the CNN [66].

Y. N. Fu'adah *et al.* from Telkom University in Indonesia created an autonomous cancer classification system that was trained on 3000 images and validated on 1000 images. The model was ultimately able to reach an accuracy of 99%. The system was able to classify four types of skin lesions: Dermatofibroma, Nevus pigmentosus, Squamous cell carcinoma, and Melanoma. The researchers had used three types of optimisers to see which one was best for the accuracy and found that the Adam optimiser was best [67].

3. Methodology

This chapter will describe the methodical approach taken to contriving a CNN model which classifies types of skin lesions, implemented using the *universal workflow method* [22][76] for machine learning. In particular, this chapter will give a thorough explanation of choices and selections made to construct a CNN image classification algorithm.

3.1. MLP model

In this instance, the MLP will handle a multi-class classification problem; for the output layer, It will be softmax activation & categorical-cross entropy loss function.

Dataset & data preprocessing

The dataset for the MLP model is the MNIST dataset, consisting of 70,000 (28x28) greyscale images of 10 numerical digits from 0 to 9. It allows for experimenting with learning techniques while spending as little time as possible on data preparation [108]. The training data is used for training the model, and the testing data is used to evaluate the models performance, giving an output validation accuracy and loss of the model.

The MNIST dataset is loaded from tensorflows accessible datasets library:

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

The data is then vectorised; as each image is of shape 28x28, it will be formed into a single dimension of 784 (28x28=784):

```
dimension_new_data = np.prod(train_images.shape[1:])
train_data = train_images.reshape(train_images.shape[0], dimension_new_data)
test_data = test_images.reshape(test_images.shape[0], dimension_new_data)
```

Using the to_categorical function converts labels from integers to one-hot encoded categorical labels (such as: [1,0,0,0,0,0,0,0,0,0],[0,1,0,0,0,0,0,0,0,0]...):

```
train_labels_cat = to_categorical(train_labels)
test_labels_cat = to_categorical(test_labels)
```

3.2. CNN Model

3.2.1. The Dataset

The dataset, formed by International Skin Imaging Collaboration (ISIC), is distributed on Kaggle [71]. Images are sorted & divided according to the label it belongs to. The ISIC dataset has *2357* images labelled into nine different skin lesions: 2239 images belonging to training subset, 188 and 444 belonging to test and validation subsets respectively.

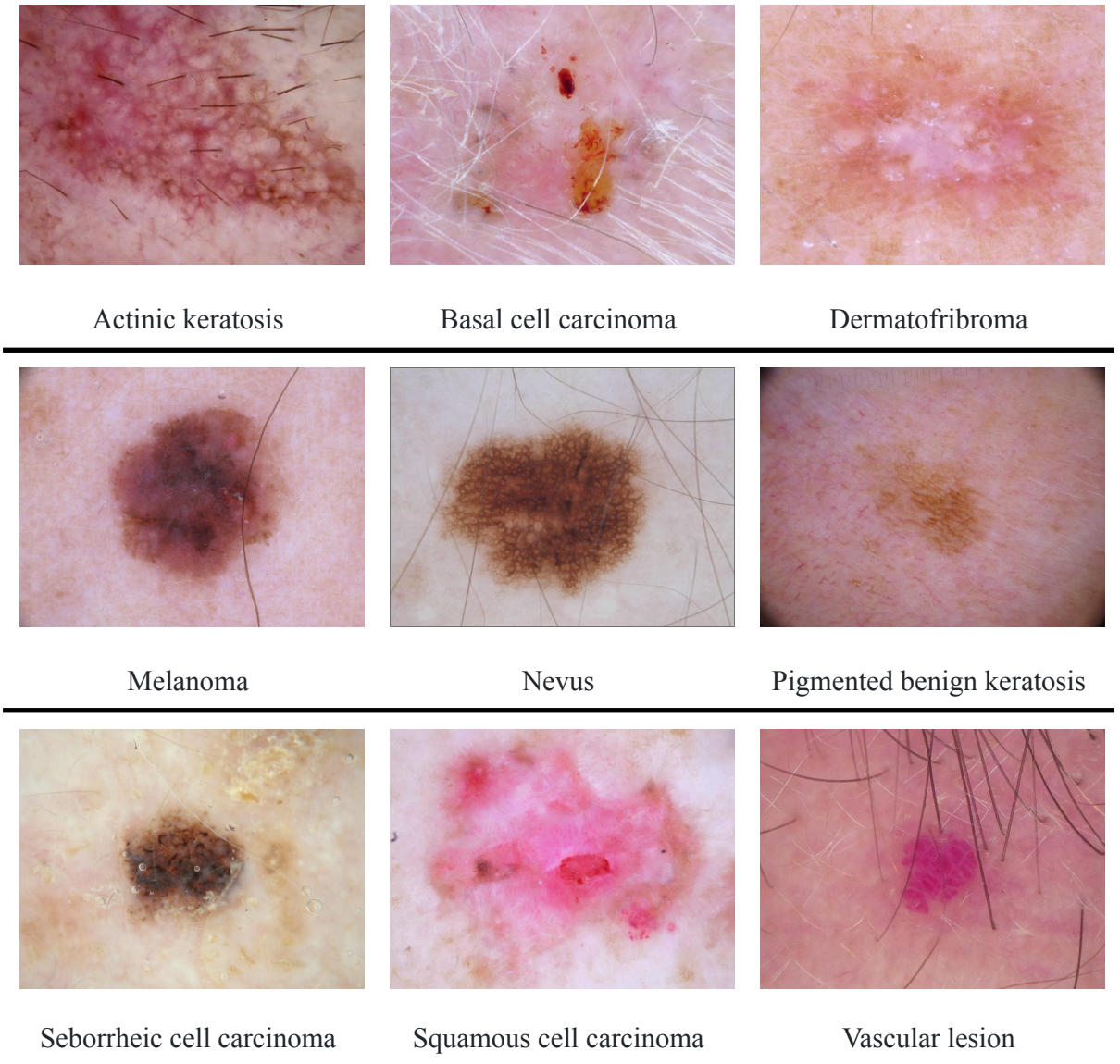


Figure 3.1 images of skin lesions from the ISIC dataset[71]

3.2.2. Defining the problem

The type of problem the CNN will aim to solve is an image classification for skin lesions; hence it is a multi-class classification problem. It is vital to distinguish between “multi-class”

and “multi-label” classification problems. Multi-class classification is where data samples belong to one class only, whereas multi-label classification is where samples may belong to multiple labels. The samples used for training are from 9 different classes of skin lesions. The objective is to use ML regularisation techniques to train a model that will be able to classify images.

3.2.3. Monitoring progress

To monitor progress of the two models (CNN & MLP), the accuracy and degree of overfitting will be observed. The accuracy of a model is describing how well the model is doing at classifying images into the correct labels Keras provides metric data per epoch. Epochs define how many times the algorithm will work through the training set [80]. overfitting is when the algorithm is Learning perfect patterns of training data, “overlearning” features of samples. The likelihood that unseen data will have corresponding idiosyncratic characteristics that are found in training data is low, hence the algorithm will not be able to classify new data accurately.

Overfitting is identified from analysing learning curves and observing validation accuracy and training accuracy after training and validating a network on training and validation data respectively. “*Learning curves (LCs) are deemed effective tools for monitoring the performance”* [104]. When validation loss & accuracy denature correspondingly after a certain epoch, this is considered overfitting; lack of generalisation on new data[69].

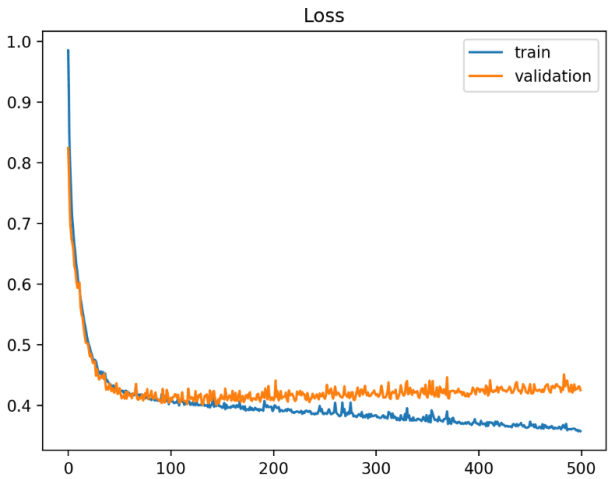


Figure 3.2 training & validation loss of an overfitting model [121]

.

3.2.4. Evaluation protocol

Evaluation protocols validate a NN's training to understand if it generalises well on unseen data. Types of evaluation protocols are implemented in ML: *cross-validation & hold out validation.*

Holdout validation is whereby data is split into training, test, and validation subsets. The training set is what an ML model would be trained on, and the testing set is what a NN would be tested on as it is "unseen data" [54].

Cross-validation is where datasets are divided into *k* numbers of sections known as *folds* [55]. Each fold is used as testing data at some point, while the rest is as training data [54].

The proposed CNN will have a training data split where 80% of data is used for actual training, and 20% will be a validation set. The model will not be evaluated on training data because the goal of ML is to make good predictions from unseen data (appendix **figure 6.5**).

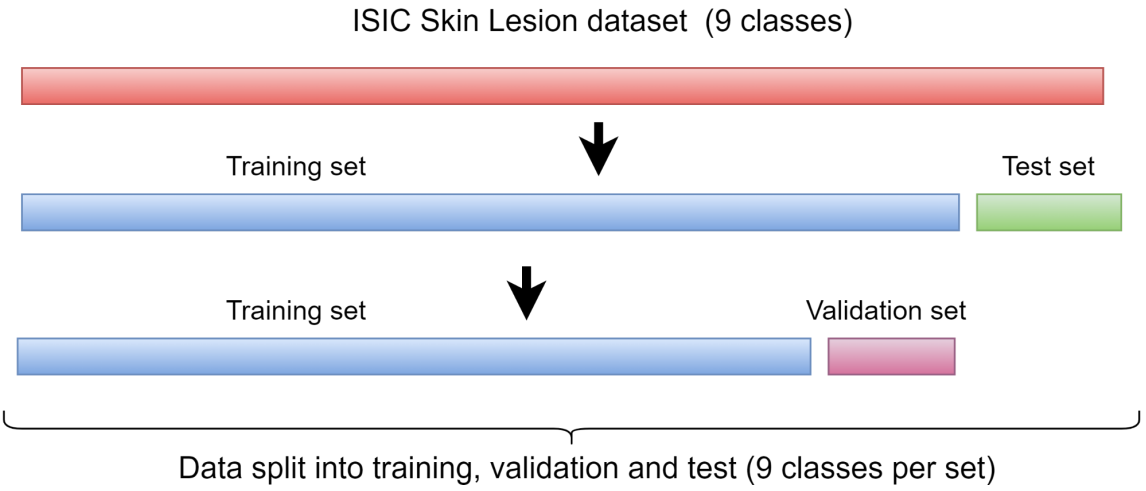


Figure 3.3 ISIC dataset split into subsets

3.2.5. Data preparation

The images are of varying sizes, all larger than 500 by 500 pixels; thus, images will be regularised into 224 by 224 pixels. This is the most appropriate size that will not diminish quality whilst also presenting the actual skin lesion. Although smaller dimensions is less straining on the runtime of training CNN's, in this case going any smaller sizes begins to diminish the image, with the actual skin lesion not present or cut off to the point it becomes redundant.

ImageDataGenerator

Keras has preprocessing tools such as *ImageDataGenerator*, which automatically converts image files on storage devices into batches of preprocessed tensors with real-time data augmentation, forming the training, test and validation generator. The samples are rescaled by size 1./255; a pixel has a value ranging from 0 to 255; as that is an extensive range, rescaling it will set it to 0 to 1, making it easier to learn representations. 20% of the training data is used for validation.

The samples per batch are set to 128, and each image is of shape, Height x width x colour channels. Therefore, the shape of each image would be [224 x 224 x 3] (3 represents RGB channels). The class mode was selected as 'categorical', as the multi-class problem consists of 9 categories.

ImageDataGenerator also allows for data augmentation techniques, which is modifying a dataset with additional information, such as brightness & geometric transformations[115]. The purpose of this is to diversify a smaller dataset to allow for more information for the model to learn from to make more accurate predictions. (appendix **figure 6.4** for code implementation)

Flow_from_directory Method

The dataset is already sorted into respective classes, and the file directories are appropriately grouped and labelled on the local machine. Flow-from-directory method loads data automatically from the directories (appendix **figure 6.5** for code implementation).

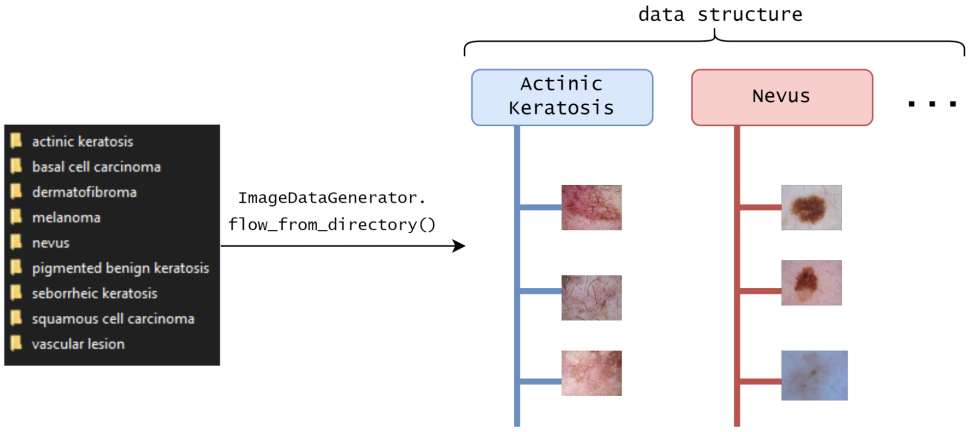


Figure 3.4 flow_from_directory() method, structuring data before the data is split

3.2.6. CNN Model architecture

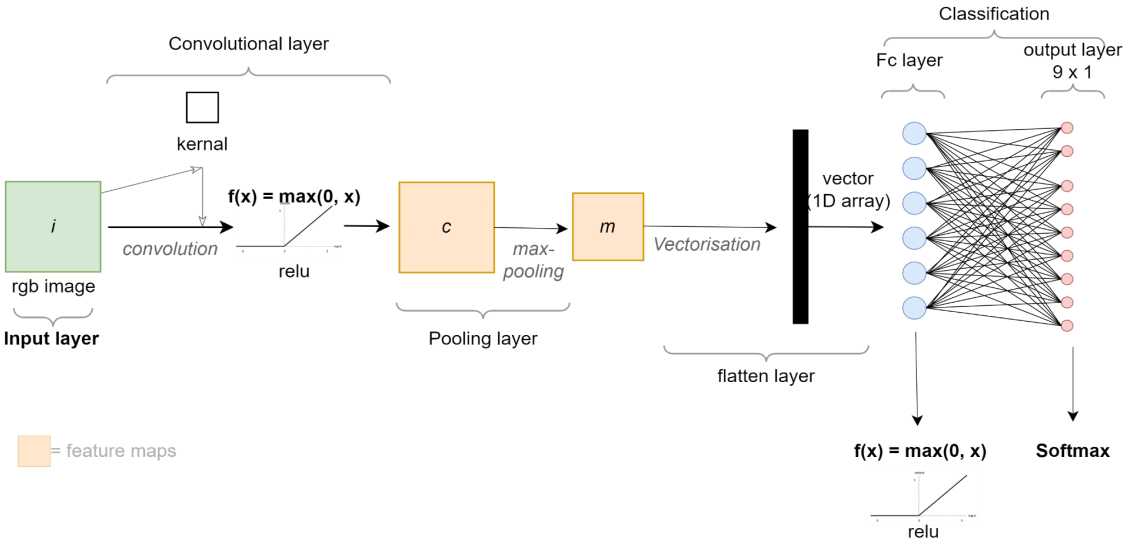


Figure 3.5 Baseline CNN Model architecture of project

Initial configuration of the CNN:

- convolutional input layer, with 3x3 Kernels - relu activation
- Max-Pooling layer
- Flatten layer
- FC layer - relu activation
- Softmax output layer
- Optimiser rmsprop

This is only for **gaining statistical power** hence why a shallow architecture is being considered, however progressively, the model will change due to regularisation experiments

Regularisation experiments (chapter 4):

- Developing a model to gain statistical power
- Developing a model that overfits
- Filter configurations of convolutional layers
- Layer depth
- Adam & RMSprop optimiser
- Adding Dropout
- Learning rate adjustments
- Weight regularisation (l1-l2)
- Increasing unit size of dense layer

Architectural choices

Convolutional layer

Chapter 2 *discusses* the convolutional operation and its filters for feature extraction of an input image, outputting it onto a feature map. Extracted features become increasingly meaningful as it progresses through the network, becoming progressively unique. Specifically, convolutional operation deals with the redundancy of features with its weight sharing [92]. Keras offers several types of convolutional layers, such as Conv2D & Conv3D. I will be using conv2d as the images are 2D with (Height x Width) as dimensions. In 2D convolutions, the filter moves in *x* and *y* directions [92]. The output of Conv2D will result in 3D tensors of shape (height x width x channels). The conv2D layer will have a 3x3 kernel dimension, as this is a standard size used across CNN algorithms, along with 5x5. Setting the kernel size too large could result in prolonged training time. Moreover, odd-sized filters are preferred as they symmetrically divide a pixel layer or feature map in a way that will fit the output pixel layer [123].

Pooling layer

The **max-pooling** (MaxPooling2D = keras class) operation down-samples maximum elements from parts of a feature map selected by the filter after convolutions [77]. It takes the most distinguishing features of a particular region and negates redundant information, preserving the dominant features of every pooling operation [68]. Reducing dimensions of feature maps allows for reducing parameters learned, hence making feature maps computationally efficient. Average pooling takes the average value of the pixels of an image input, which may distort prominent features of an image. It smoothes out images which may cause sharp features to be ignored [94][114], thus the medical skin lesion images will need intricate features to be retained.

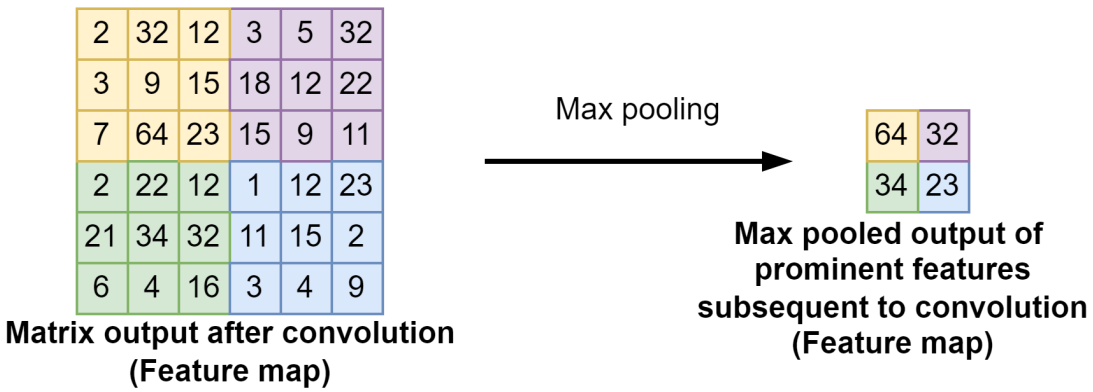


Figure 3.6 Max pooling takes the "Max" feature from a feature map matrix

Flatten layer

After the flatten layers, the successive layers are FC layers that take 1-dimensional tensors (vectors) of data; however, before this, the data is processed as a 3D tensor. Consequently, **flatten layers** "flatten" the output for preceding layers creating a 1-dimensional feature vector [79].

FC layer

Once the output of the pooling operation or the convolutional operation is flattened into a vector array, it becomes the input for FC layers. At this part of the network, it behaves more like a traditional MLP; the same hidden layer calculation occurs [95]. The inputs are connected to all of the activation units in the next layer (the softmax layer). FC layers connect every neuron to the following FC layer; hence every detail of pixel location of the initial input at the start is relayed to the output layer, allowing for the decision to be made. The final convolutional layer's receptive field does not cover the entire spatial dimension of the image; the features generated by it correspond to a portion of the input image. As a result, a few FC layers are required in such a scenario [105].

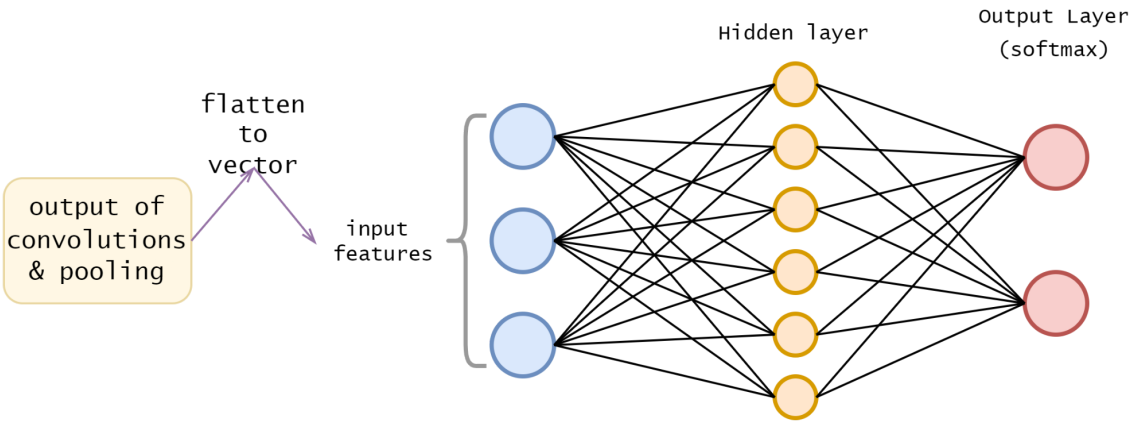


Figure 3.7 hidden layer and output layer (FC layers) part of the classification process of the network

Hidden layer activation function - ReLu

For the architecture of the custom CNN model, the **relu activation functions** would apply to convolutional layers and any FC hidden layers in the network. Goodfellow states that "in modern neural networks, the default recommendation is to use the rectified linear unit or ReLU" [83]. As mentioned in **Chapter 2**, ReLu brings computational benefits due to how it handles non-zero values of neurons. S. Zou et al. conducted several experiments using different hidden layer activation functions. They found that ReLu performed the best for

multivariate image classification problems compared to Sigmoid and TanH [89]. K. Jarrett et al. studied how non-linearity affected the feature extraction process and accuracy of object recognition models, concluding that ReLu is the "single most important factor in improving the performance of a recognition system."[90]

Output layer - Softmax activation

The output layer activation function of choice would be the softmax activation layer. Softmax calculates probabilities of categorical distribution of the classes [112] hence, the reason for the final output layer of CNN model.

Loss function - Categorical Cross-Entropy

Keras has loss functions such as *binary cross-entropy, categorical cross-entropy, and sparse categorical cross-entropy.* The CNN model will be using the categorical cross-entropy loss function; when labels are one-hot encoded. As there are 9 classes of skin lesions, categorical cross-entropy would measure the probability of 9 classes predicted from the image samples and then calculate the loss of an example by computing the sum [53]. the one-hot encoded labels would be as follows [96]:

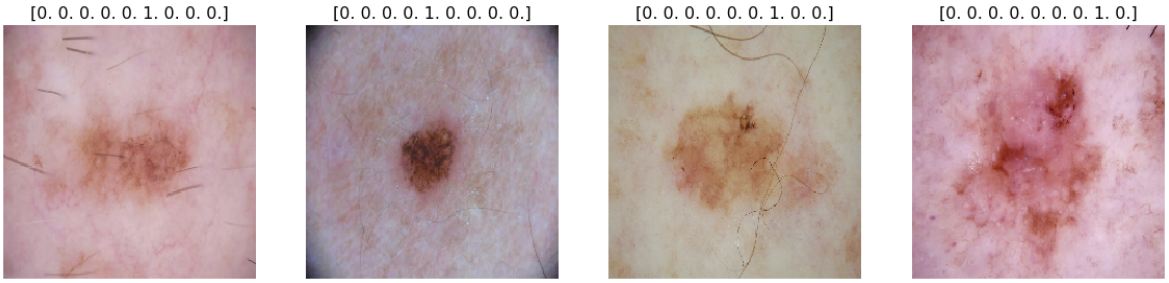


Figure 3.8 skin lesion images one-hot encoded

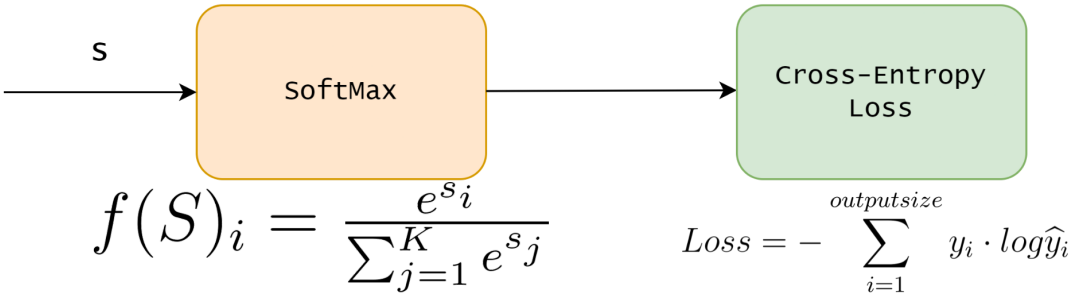


Figure 3.9 softmax activation (labelled softmax) & cross-entropy loss (labelled Cross-Entropy Loss)

4. Implementation & Results

This chapter will implement various structural concepts discussed in the methodology, showcasing various architectural building blocks of a CNN and an MLP Network. An MLP is constructed to show the basic functionality of an NN. Moreover, this section shows the technical experimentations performed on the CNN to build an optimal model.

**For some sections, graphical representations would be of the highlighted sections of tables. With sections with multiple graphs, there will be clear indication of what data is represented.*

4.1. MLP

The MLP model was constructed with a shallow architecture; input layer, an output layer and an FC layer. All experiments are conducted on 20 epochs. (appendix)

4.1.1. Experimentation

Statistical power

The first experimentation will be configuring the unit size of input and hidden layer of the MLP model. To begin with, the model will have an input layer, a single hidden layer, and an output layer (3 layers). The layered density of the model will be experimented on later. The model gained statistical power (baseline of MNIST is 0.1) from the unit configuration of 8-8; 8 for the input layer and 8 for the hidden layer. Achieving an accuracy of over 79%, validation accuracy is closely following the training accuracy, showing no signs of overfitting.

Unit size (input layer - hidden layer)	**Val-Acc**	**Val-loss**
8-8	0.7961	0.6972

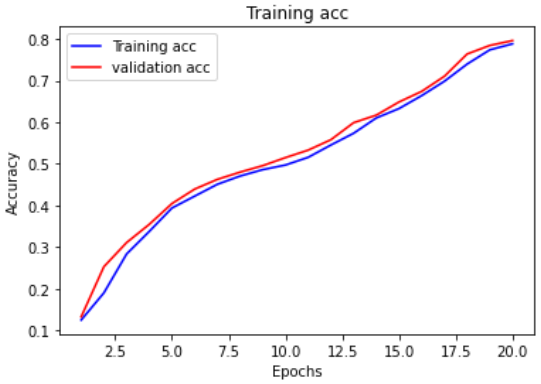


Figure 4.1 accuracy

Overfitting model

Upon gaining statistical power, the MLP must be sufficiently robust, ensuring it has enough layers or layer units to fit & model the problem at hand properly. For instance, a network may have statistical power with just one hidden layer with 8 units; however, this would not be sufficient to deal with a multiclass problem well. Chollet states that an ML model should have enough power to overfit initially "figure out where the border lies" [19]. Henceforth, increasing input and hidden layer units to 128 resulted in the model overfitting. When analysing the positing of validation data, it tends to degrade around 2.3 epochs, as the validation accuracy becomes less than the training, and the validation loss increases.

Unit size (Input layer-hidden layer)	**Val Acc**	**Val Loss**
128-128	0.9651	0.3326

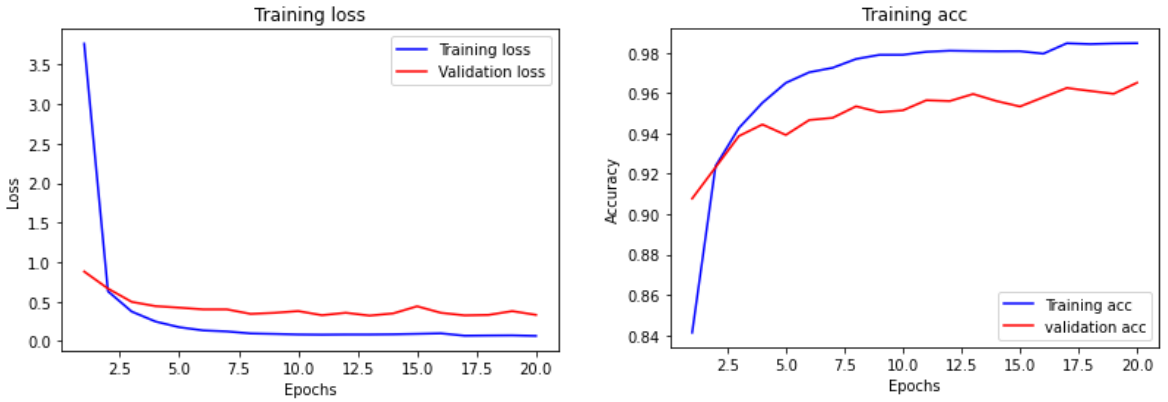


Figure 4.2 accuracy & loss

Unit experimentation

This is where the MLP model begins to be regularised. Unit experimentation is where the units of layers are adjusted to find an optimal size. The filter configuration of 128-128 for the input layer & hidden layer, respectively, showed the model to overfit. This allows for narrowing down to more optimal unit size. There was only one hidden layer for each experiment conducted, hence unit sizes were configured for input and hidden layers.

From trying unit size configurations of; 8-8, 16-16, 64-64, and 128-128, the most optimal was 64-64, overfitting less aggressively than 128-128 (figure 4.2); the validation loss tends to follow the training loss much more closely.

Unit size (input layer - hidden layer)	Val Acc	Val Loss
128-128	0.9651	0.3326
64-64	0.9540	0.2404
32-32	0.9427	0.2418
16-16	0.8721	0.4653
8-8	0.7961	0.6972

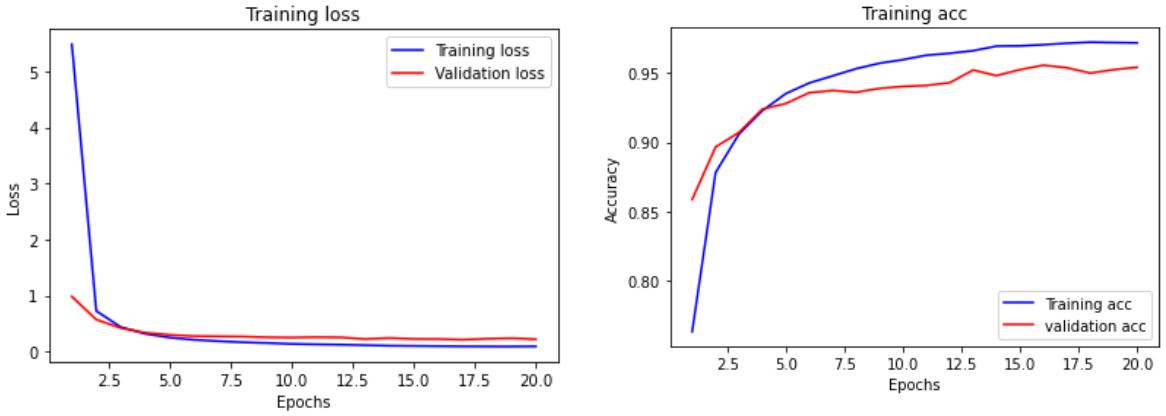


Figure 4.3 accuracy & loss (**appendix-6.16**)

Layer depth

Prior to upcoming experiments, the layer depth consisted of input, output and a hidden layer (3 layers). The number of hidden layers was experimented with to assess optimality. Increasing layer depth to 5 increased accuracy slightly; going beyond causes the model to grossly overfit; denaturing the model's prowess on unseen data.

Layer depth (hidden layers)	Val Acc	Val Loss
3(1)	0.9540	0.2404
4(2)	0.9562	0.2251
5(3)	0.9634	0.1646
6(4)	0.9667	0.1625

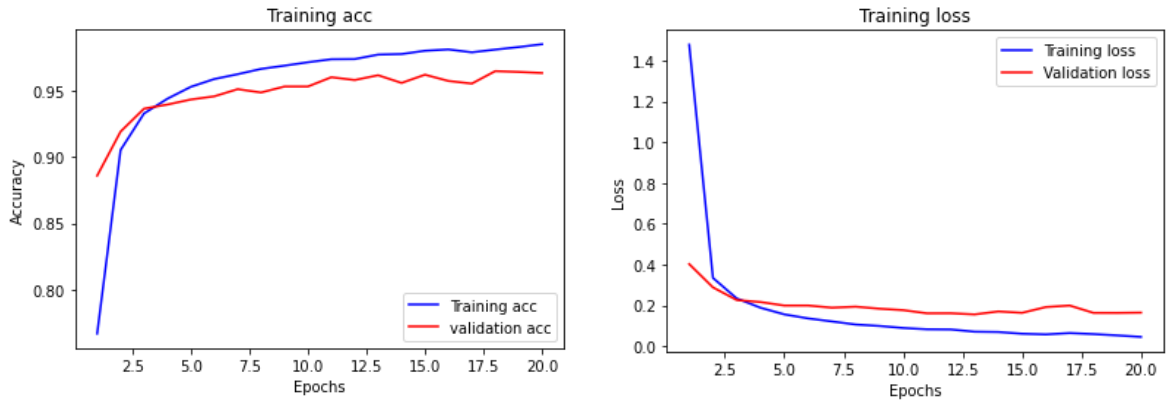


Figure 4.4 accuracy & loss

Dropout layer

Adding a dropout layer is to remedy overfitting. From the experiments done, 0.1 was the best value for the model. Anything more outstanding leads to a slight reduction in accuracy & poor generalisation, which is apparent when analysing the learning curves of each experiment, compared to other experiments (figure 4.2-4.4).

Dropout layer	Val Acc	Val Loss
0.5	0.9408	0.2240
0.3	0.9584	0.1469
0.1	0.9667	0.1290

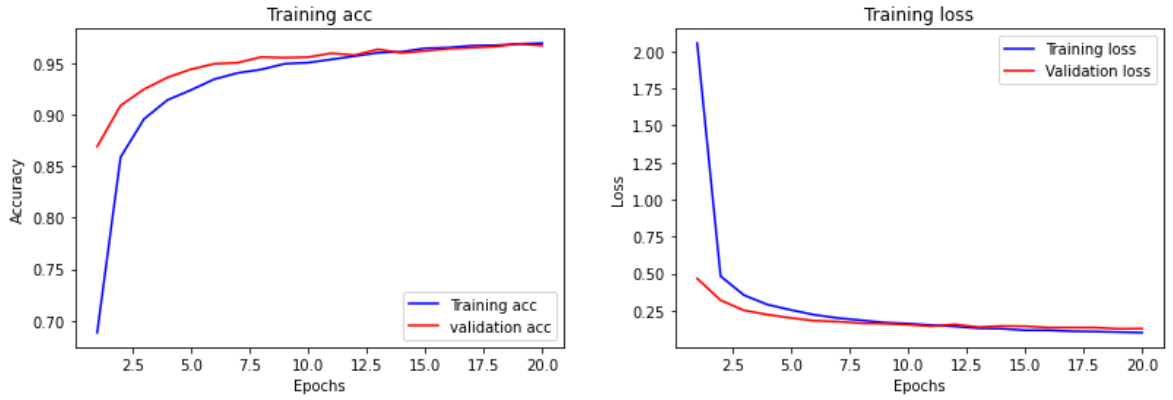


Figure 4.5 accuracy & loss

4.1.2. Evaluating model

The most optimal model from the experimentation is a model of 5 layers, with unit configuration of 64 for the input layer and following FC layers, and a softmax layer and dropout layer. After training the model, it is evaluated on test data to see how the model will perform on unseen data. The model maintained an accuracy of 96.5%.

```
313/313 [==============================] - 0s 891us/step - loss: 0.1269 - accuracy: 0.9653
Loss = 0.12693554162979126, accuracy = 0.9653000235557556
```

Figure 4.7 evaluation on test data for mlp model

4.2. CNN

4.2.1. Model with statistical power

In **chapter 3**, I had established that the common-sense baseline of the model would be 0.11 (the class possibility is of $\frac{1}{9}$); hence a model that would beat that would be a model that has statistical power. To begin with, the architecture of the model will be the most basic level of CNN architecture, consisting of a single conv2d layer, maxPooling2D layer, and the output Softmax classification layer. Moreover, the optimiser used will be rmsprop until **optimisation** experimentation, exploring other experiments. If there is an actual effect to identify, statistical power is the likelihood of discovering it [20][21].

Exp no.	Epoch	Conv layers	Filter size	Val-Acc	Val-loss
1	10	1	1	0.4910	1.5008

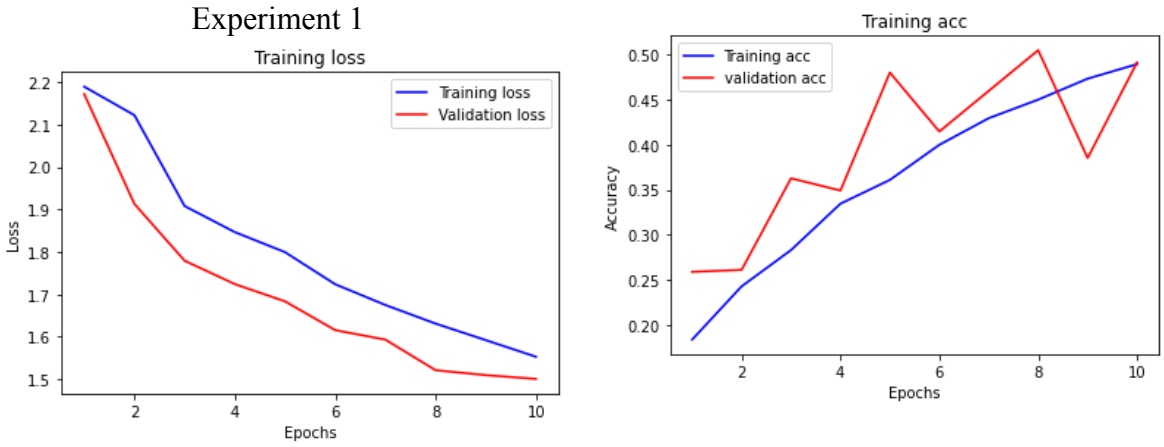


Figure 4.8 accuracy & loss **(appendix 6.6-6.7)**

The CNN was trained for 10 epochs & managed to gain statistical power achieving over 61% accuracy on training data but with also a high amount of loss. The model does not properly fit as the validation data show some distance between them, indicating poor fitting.

```
layers.Conv2D(1, 3, activation='relu', input_shape=(224,224,3)),
```

The highlighted portion of the conv2d layer is the filter argument; an integer value that represents the number of output filters, determining output space dimensions for the following layers. The integer next to filter size is the kernel size, which represents the *heightxWidth* of the kernel (in this case 3x3).

4.2.2. Model that overfits

After constructing a basic model that can gain statistical power, the CNN will need enough power to overfit. The previous model overfit; however, following experiments will increase filter size and add one more convolutional layer to ensure the model is adequately powerful. There needs to be enough layers and parameters that ensure the model is overfitting. Therefore, an additional convolutional layer will be introduced with a larger filter size for these experimentations. (see section ***4.1.1*** for reasoning for developing a model to overfit).

The filter size configurations were 64-128 (2 convolutional layers). Moreover, there will also be one hidden layer prior to the output layer. I will be running this on 20 epochs as this will give a range of results to analyse.

Exp no.	**Epoch**	**Conv layers**	**Filter size**	**Val-Acc**	**Val-loss**
2	20	2	64-128	0.8356	0.4526

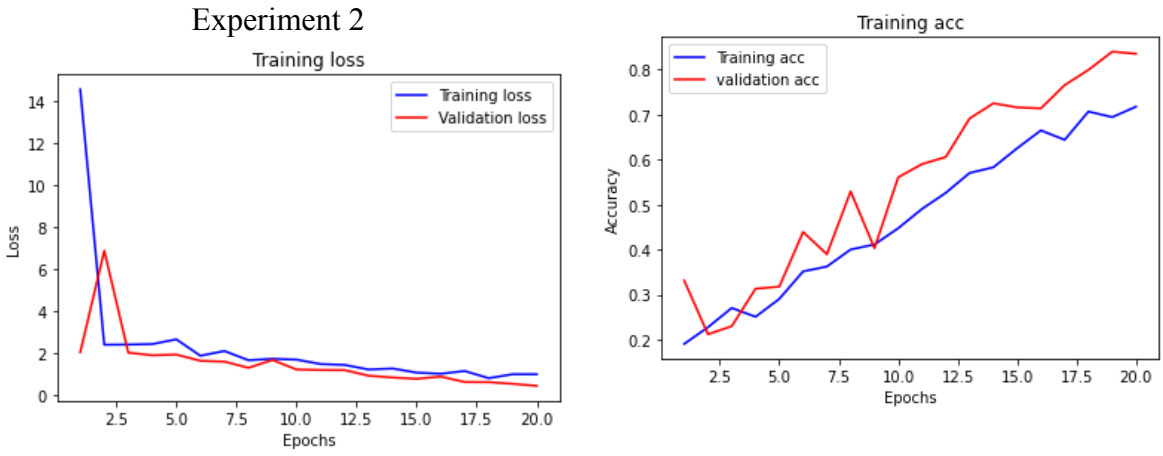


Figure 4.9 accuracy & loss of model (**appendix-6.8-6.10**)

Figure 4.9 shows the accuracy and loss of the model after changes were made. Overfitting occurs around 2.5 epochs. The loss of the model shows that validation loss is consistently less than training loss per epoch. Nonetheless, the model is performing well so far, achieving an accuracy of 83.6%.

4.2.3. Regularising & tuning hyperparameters

This section will look at repeatedly configuring, tuning, & training the model until the most optimal model is found [21]. This section will iteratively undergo alterations [23][33] to filter size, optimisers, layer depth, dropout, learning rate & weight regularisation.

Filter configuration of convolutional layers

The previous experiments had an element of filter size tuning. However, that was intended to gain statistical power & scale up to develop a model that overfits.

The size of convolution filters in CNNs determines the size of the receptive field from which features are extracted [97]. Hence, adjusting filter size improves the model. It is common for a convolutional layer to learn from 32 to 512 filters, even above and beyond those [98].

Exp no.	Epoch	Conv layers	Filter size	Val Acc
1	10	1	1	0.4910
2	20	2	64-128	0.8356
3	20	2	32-64	0.8536
4	20	2	16-32	0.7883
5	20	2	8-16	0.6622

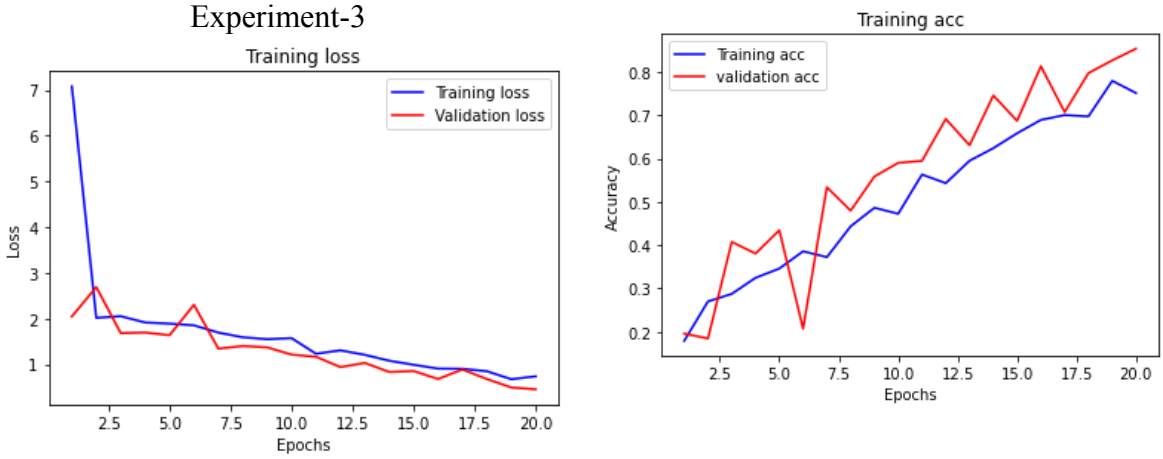


Figure 4.10 accuracy & loss of model (**appendix-6.11-6.12**)

Increasing the model to configured filter size of 32-64 led to 85.4% accuracy, appearing to overfit less aggressively when analysing the loss curve. A promising sign of generalisation occurs when the training & validation loss is reduced alongside each other without any divergence. Going any lower or higher than these configured filter sizes resulted in more jaggedness of curves, showing signs of overfitting. The model will explicitly learn training

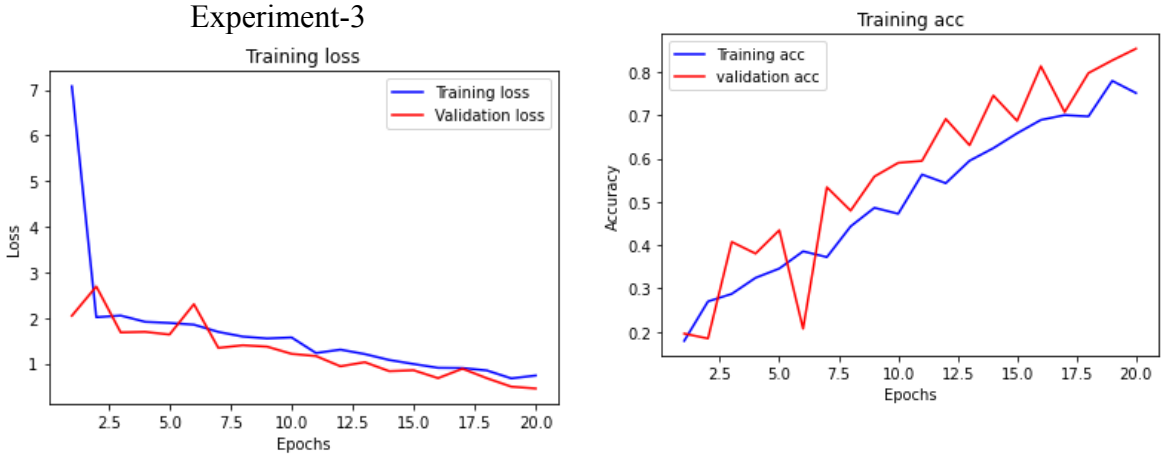
data patterns when increasing filter size, possibly ignoring generic features. However, the validation is erratic and is not as smooth; hence, considering this filter configuration (32-64), the next set of experiments adjusted layer depth.

Adjusting convolutional layer depth

Abstractions become increasingly more complex per convolution as the output feature map decreases, retaining more prominent original input features. Therefore, convolutional layer depth is vital to experiment with as it will show the network's optimality depending on the number of convolutions.

Exp no.	Epoch	Conv layers[filter size]	Layer depth	Val Acc
3	20	2 [32-64]	7	0.8536
6	20	3 [32-64-64]	9	0.5878
7	20	3 [32-64-128]	9	0.6126
8	20	6 [32-32-64-64-128-128]	15	0.4414
9	20	3 [16-32-64]	9	0.7342
10	20	4 [32-32-64-64]	9	0.7140

Increasing the depth of layers has a diminishing effect on the model's accuracy. The best accuracy is 85.4% of experiment 3 when there are seven layers: 2 convolutional, two max-pooling, and one fully-connected, and output layers. Going beyond seven diminishes the accuracy; there becomes a limit with adding more layers.



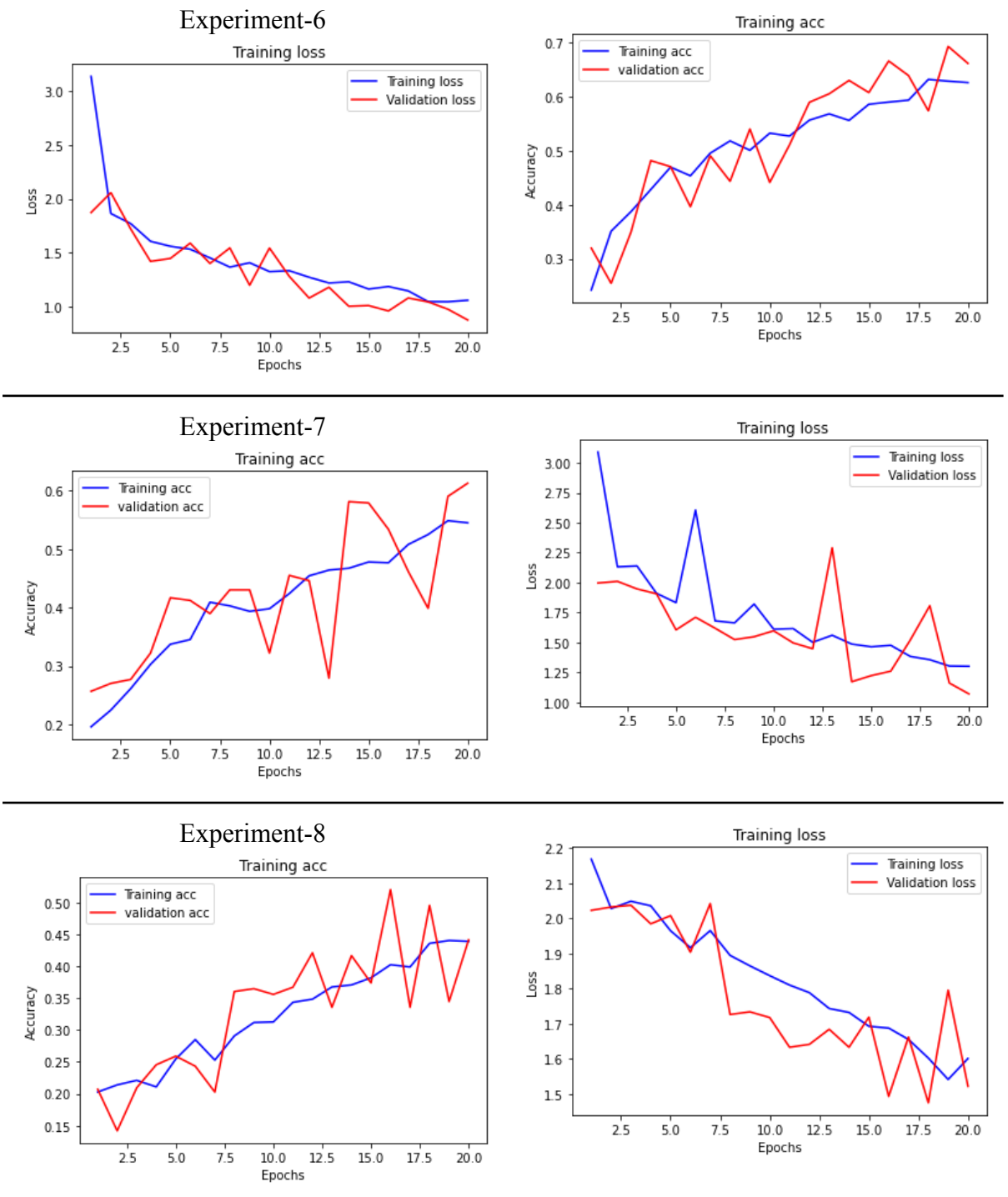


Figure 4.10 accuracy & loss (**appendix-6.13)**

Optimiser

Up until this point, all experiments were using RMSprop optimiser. The following series of experimentations will be looking to experiment with both rmsprop and adam optimiser to compare the results.

Exp no.	Epoch	Conv layers[filter size]	Optimiser	Val Acc
11	20	2 [32-64]	rmsprop	0.7973
12	20	2 [32-64]	adam	0.8153
13	20	2 [32-64]	rmsprop	0.7703
14	35	2 [32-64]	adam	0.9505
15	35	2 [32-64]	rmsprop	0.9122

Adam optimisation gained higher accuracy than rmsprop. The Adam optimiser dealt with overfitting better than rmsprop, which had more irregular forms of overfitting, but the Adam optimiser dealt quite well with it. Analysing the accuracy of both experiment-13 & 14, rmsprop shows more jaggedness, which could be a sign of poor generalisation. Based on these results, Adam is the more optimal.

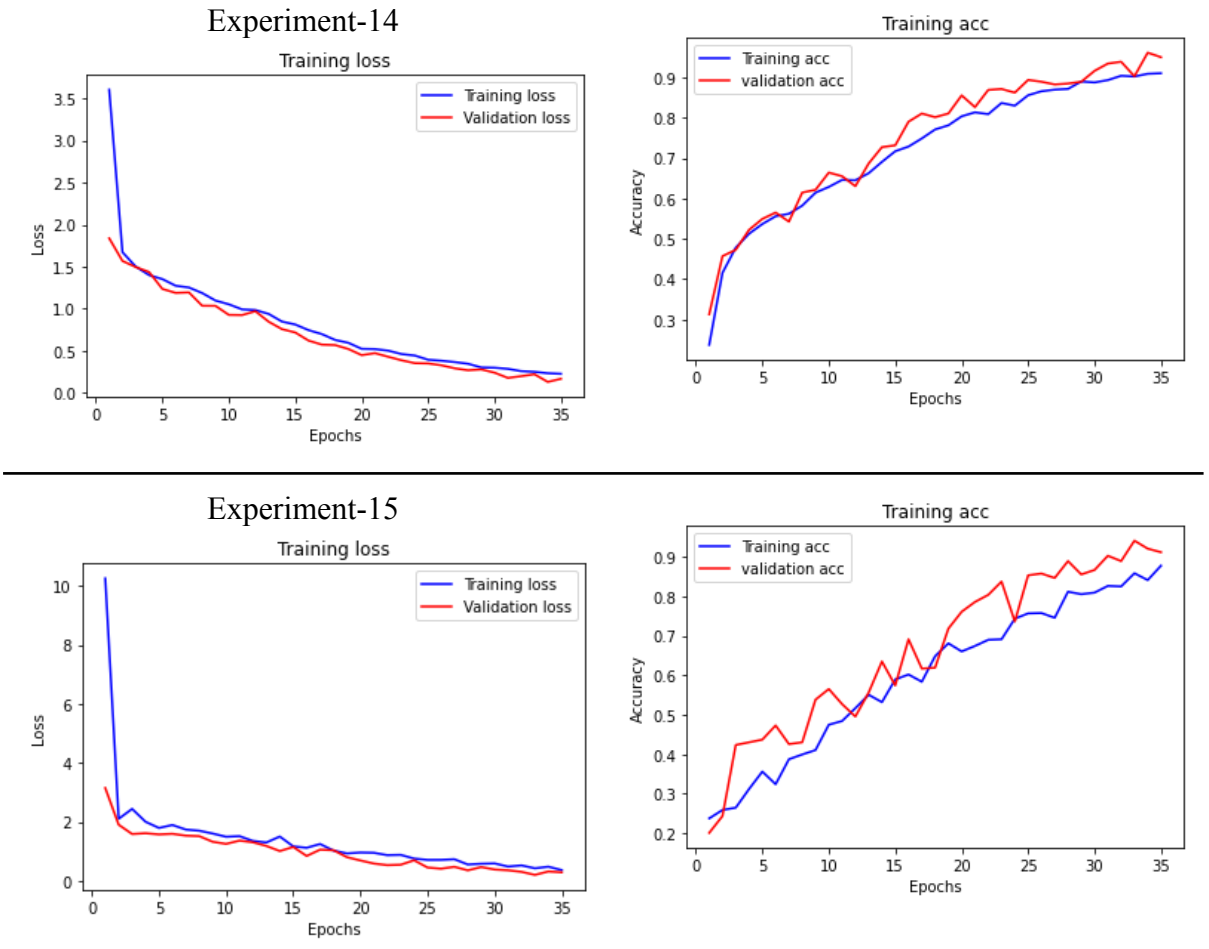


Figure 4.11 accuracy & loss (**appendix-6.14)**

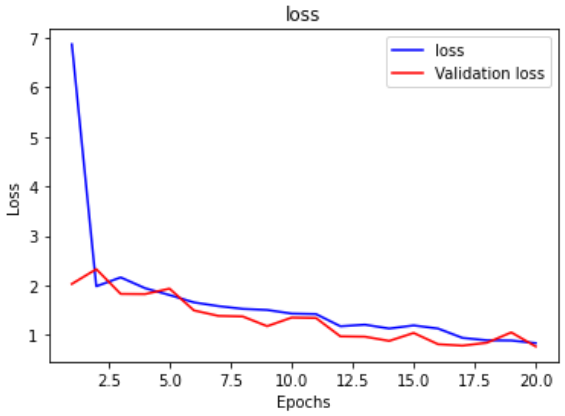
Adding dropout

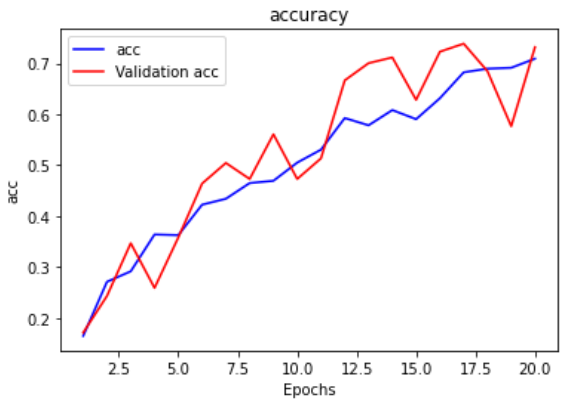
Coadaptation is when neurons learn to rectify the errors committed by other neurons upon training data. As a result, the network becomes adept at fitting training data [99]. Dropout is only used for training to handle random data fluctuations, which temporarily drops units depending on their retention probability, mitigating overfitting [99][100].

Exp no.	Epoch	Conv layers [filter size]	Dropout	Val-Acc	Val-loss
16	20	2 [32-64]	1 (0.5)	0.6937	1.2299
17	20	2 [32-64]	1 (0.3)	0.8243	0.5653
18	20	2 [32-64]	1 (0.7)	0.5833	1.4379
19	20	2 [32-64]	1 (0.2)	0.8041	0.6143
20	20	2 [32-64]	2 (0.2)	0.5459	1.1898

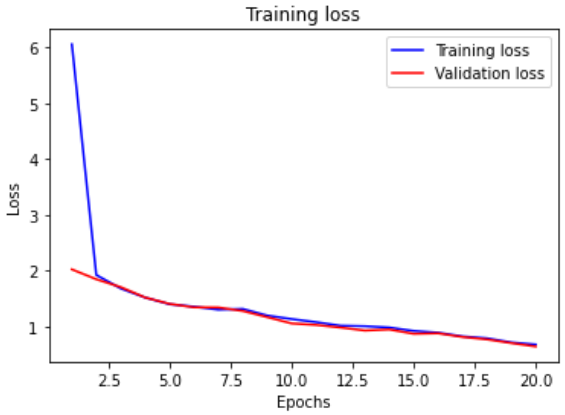
Initially, the dropout layer was added after the first convolutional layer only, and from experimentation, the best value for dropout was found. After which, dropout is applied to both convolutional layers to see its effect. More than one dropout layer affects the model's learning (compare experiment-17 & 20). The numerical value *and rate* of dropout layers ranged from 0.2 to 0.7. Experiment-18 showed vital signs of overfitting, whilst the ones lesser had mitigated overfitting well. Experiment-19 and 17 had similar results, showing that a value of 0.2-0.3 is most optimal for the model, as the validation data begins to fit the model far nicer than before. As experiment-17 had greater accuracy, further experimentation will be continued with the same configurations.

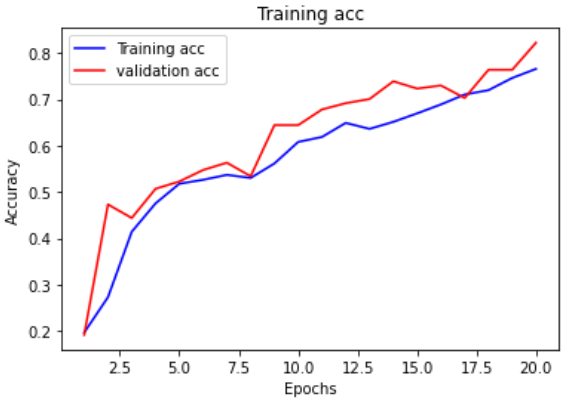
Experiment-16



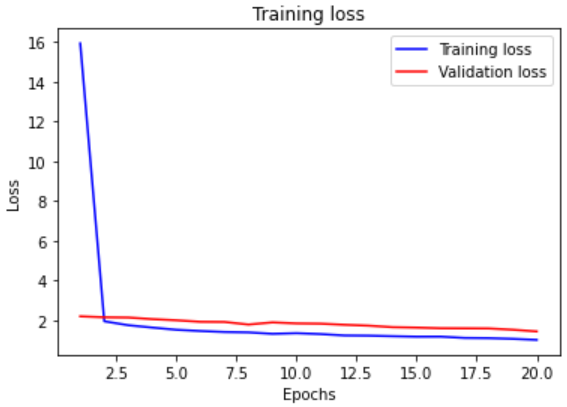


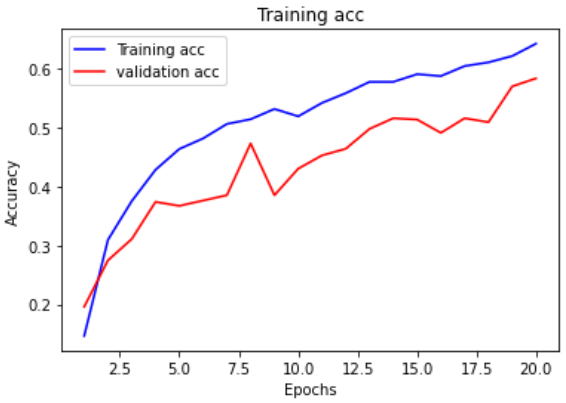
Experiment-17



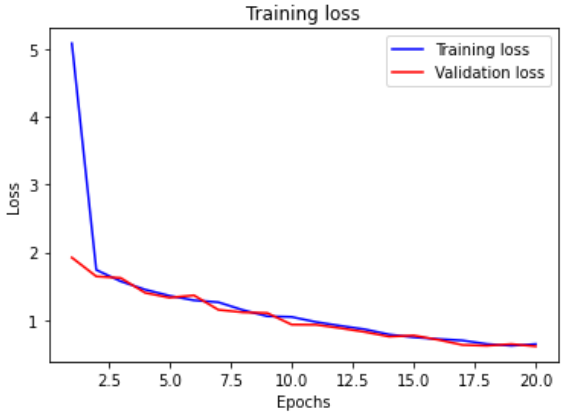


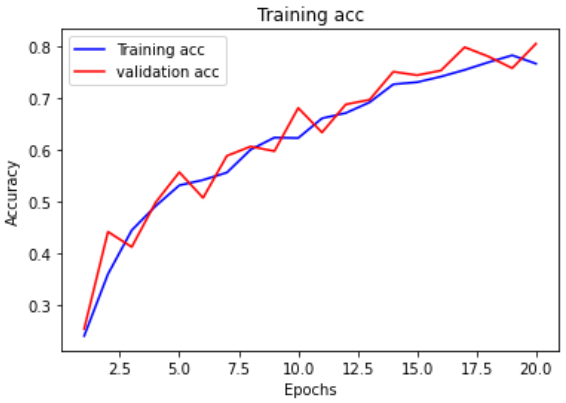
Experiment-18





Experiment-19





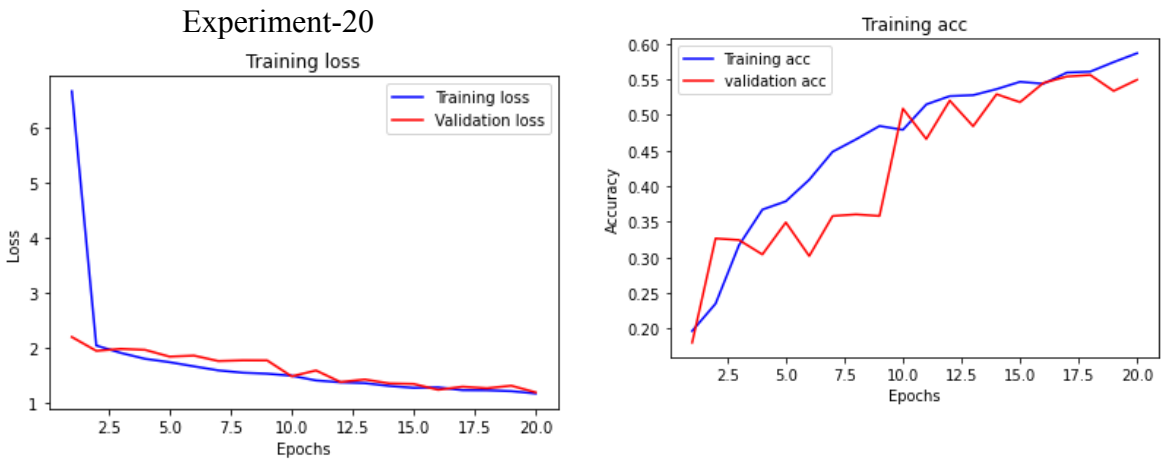


Figure 4.12 accuracy & loss

Learning rate

So far, current configuration of optimal CNN is:

Conv layers(filter size)	Max-pool	dropout	optimiser	Dense layer(units)	Output layer
2(32-64)	2	1 (0.3)	adam(0.001)	1(64)	softmax

Learning rate governs how much the model changes each time the model weights are changed in response to the predicted error [102]. The default learning rate of Adam is 0.001. A small value will result in training taking too long or failing to complete, whereas a more significant value will result in learning too quickly and not finding optimal features.

Exp no.	**Epoch**	**Learning rate**	**Val Acc**	**Val loss**
17	20	0.001	0.8221	0.6469
21	20	0.01	0.2072	2.0115
22	20	0.0001	0.5450	1.2958

Adjusting the learning rate saw no progress; going above and below the default resulted in poor accuracy. Experiment-22 overfitted early, whereas experiment-21 resulted in prolonged training and did not complete. As a result, The default learning rate was most optimal.

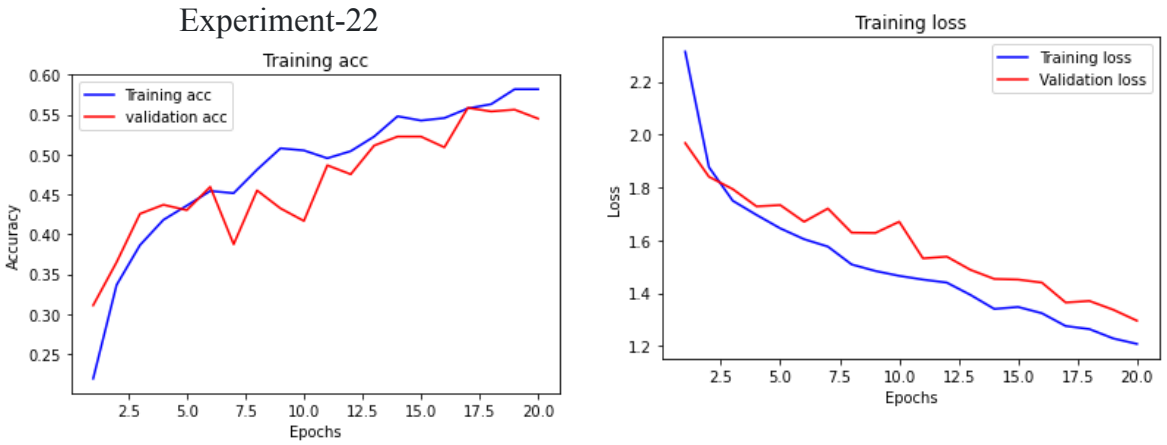


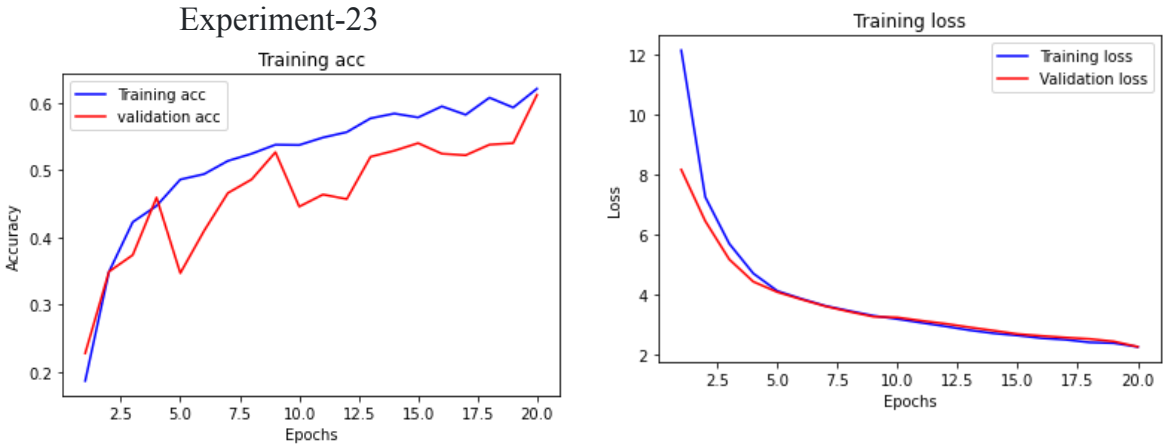
Figure 4.13 accuracy and loss of experiment-22

Weight-regularisation

Regularisation is a technique to help a model generalise better, which can help it to perform well with unseen data. There are two types of weight regulariser: l1 & l2. L1 results in more lightweight while L2 penalises larger weight creating more nuance [106].

L1 regulariser

Exp no.	Epoch	L1	Val Acc	Val loss
23	20	0.01	0.6126	2.2536
24	20	0.001	0.6577	1.2759
25	20	0.005	0.5338	2.3169



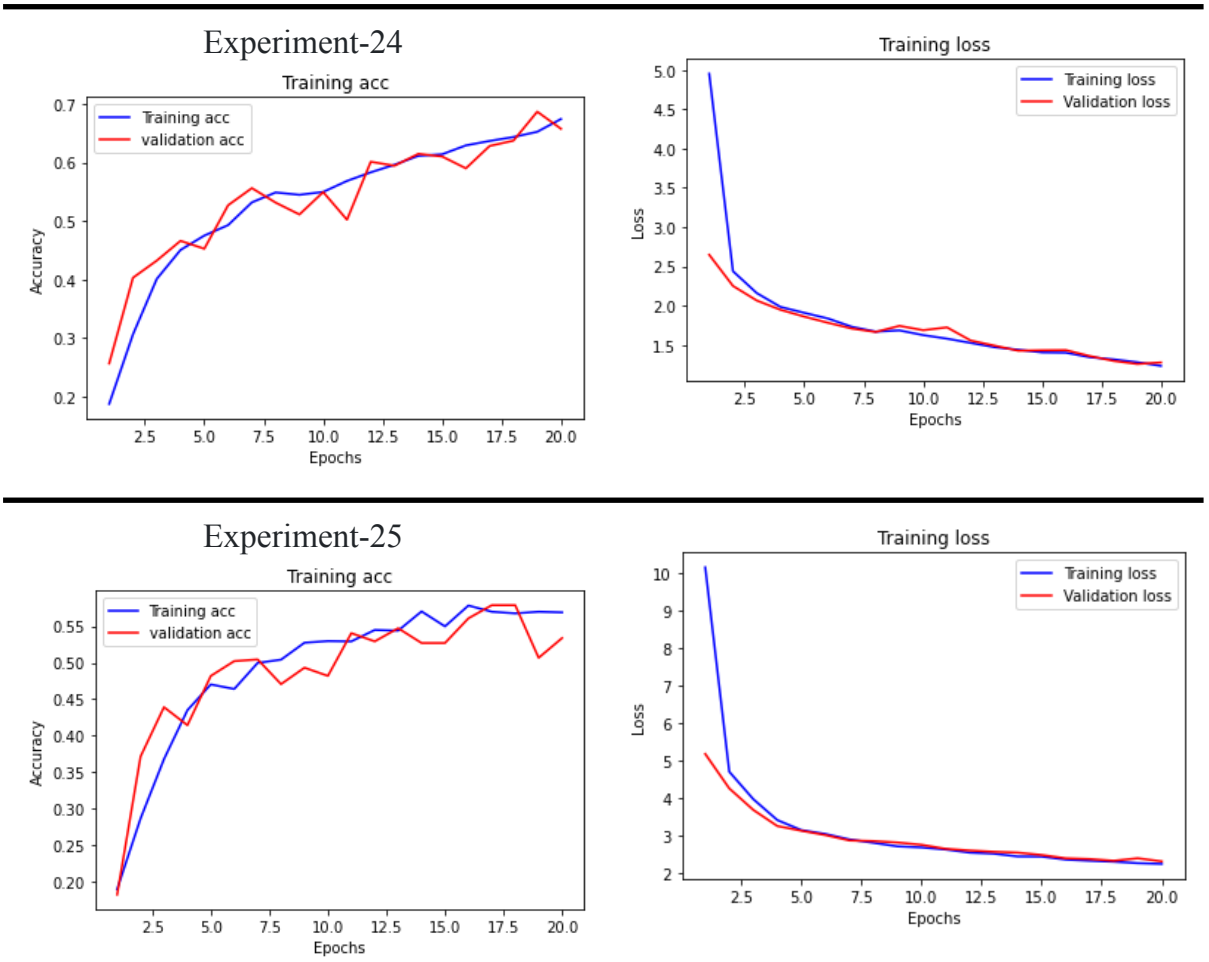


Figure 4.14 accuracy and loss

L2 regulariser

Exp no.	Epoch	L2	Val Acc	Val loss
26	20	0.001	0.8041	0.7098
27	20	0.0001	0.7568	0.9369
28	20	0.005	0.6757	1.0647

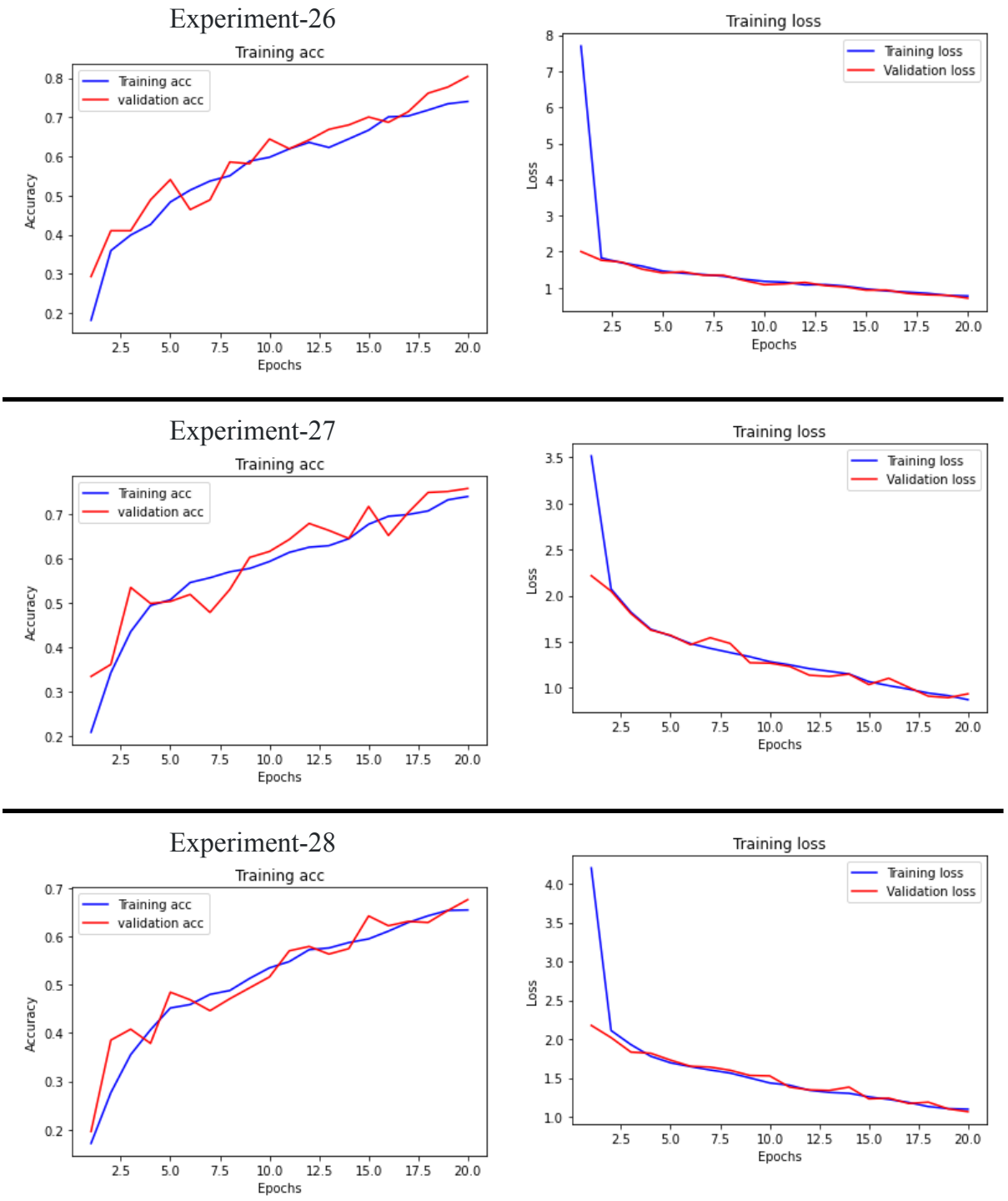


Figure 4.15 accuracy and loss

The network did not benefit from weight regularisation. The recommended sizes are of logarithmic values, and a demonstration showed that values around 0.0001 were quite beneficial [107]. However, from the experiments conducted for both l1 and l2, there seems to be lacking evidence of improvement, especially with l1. Overall they do not generalise as well as in previous experiments, which is evident when validation accuracy for all experiments tends to dip below the training accuracy.

4.2.4. Final model

The most optimal model before final experiment:

Exp no.	Conv layers(filter size)	Max-pool	dropout	optimiser	Dense layer(units)	val-acc	val-loss
17 (20 epochs)	2(32-64)	2	1 (0.3)	adam(0.001)	1(64)	0.8243	0.5653

The experiments had a penultimate Dense layer of unit size 64, hence as a final experiment, the unit is changed to 128 to see what effect it has on accuracy:

Conv layers(filter size)	Max-pool	dropout	optimiser	Dense layer(units)	val-acc	val-loss
2(32-64)	2	1 (0.3)	adam(0.001)	1(**128**)	0.9009	0.3436

Making this change has yielded greater accuracy. The model does not overfit and has generalised with validation data well, as validation loss does not go above training loss whilst accuracy is consistent. Shallow architecture necessitates many neurons in the FC layers [105].

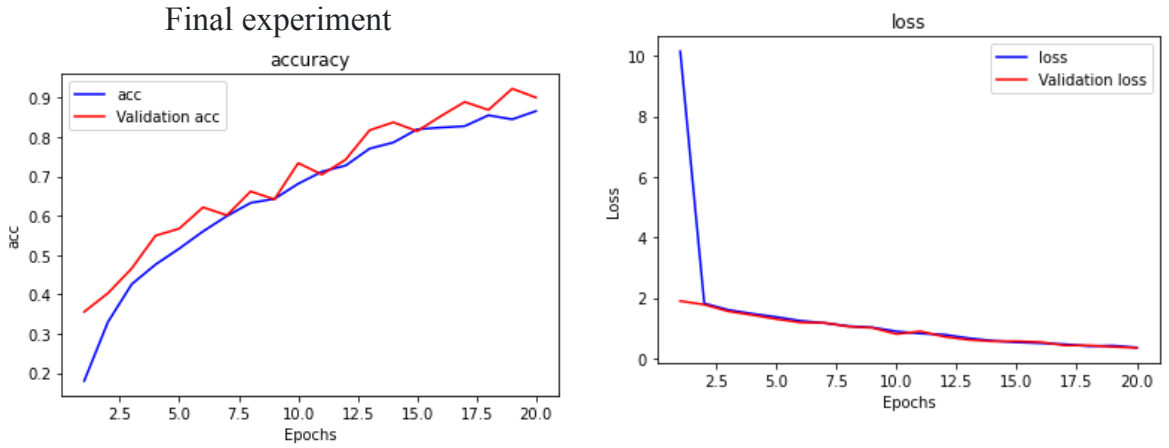


Figure 4.16 accuracy and loss of experiment-29

This made the model the most optimal after meticulous hyperparameter tuning. Therefore, testing the model is the next step, which will give an idea of how the model will handle unseen data, given that the model has generalised well to validation data.

Accuracy of model

The CNN model achieved 73.7% of accuracy when it was tested on unseen test data, the accuracy was calculated using categorical accuracy as specified in chapter 3,indicating instances of predicted labels were correctly predicted & instances of actual labels being predicted.[109]

Epoch	Accuracy	Loss
20	0.7373	0.6737

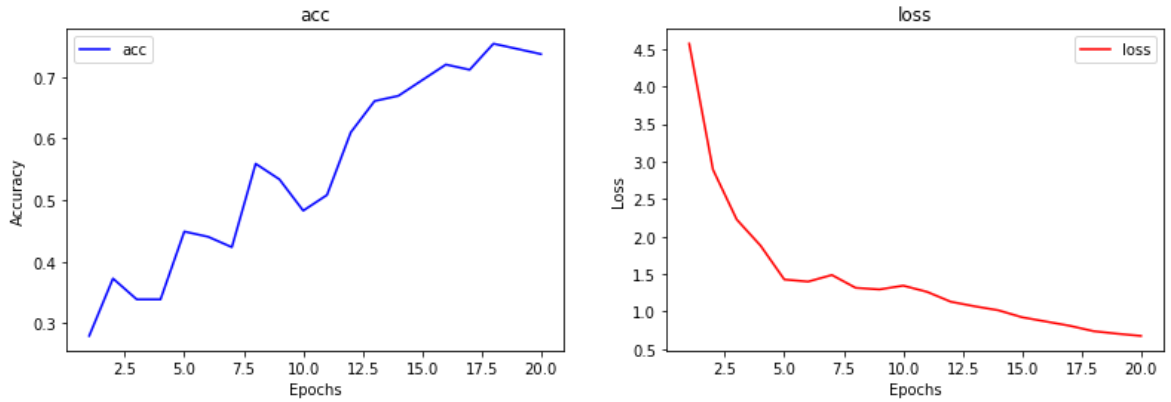


Figure 4.17 accuracy and loss

4.2.5. Discussion

Experiments were done to create the best model following the universal workflow method. An optimal network is defined by how well it generalises with training data, ultimately determining how the model will deal with unseen data, e.g. validation data. As a result, this led to an optimal CNN model; 32-64 configured for filter sizes of 2 convolutional layers with a layer depth of eight. 0.3 is the best for the dropout layer, reducing overfitting. Experimentation and studies show Adam optimiser with the default learning rate best for the model in terms of overfitting and generalisation. Ultimately, evaluating the CNN led to the accuracy of 90.1% and testing it on a test set resulted in 73.7%.

The MLP model, trained on the MNIST dataset, had an accuracy of 95.6%. Although convolutions are better for spatial information of pixels of an image and learning local patterns, the MLP model performed better with the MNIST dataset as opposed to the CNN model on the ISIC skin lesion dataset. An explanation as to why there might be the difference could be due to the fact that MNIST has more samples (70,000 total) while ISIC has over 2000. Having more data allows for a network to learn more features. Another factor would be that the ISIC dataset is an imbalanced dataset, which may cause the network to learn biased or “skewed” data of certain classes better than others.

4.2.6. Limitations and Future works

The project has changed significantly since the specification and prototype. Initially, the model would be deployed on a web application; however, it is essential to develop a model successfully. Hosting the model would result in less time for meticulous experimentation, testing and training. However, further refinement of the model can allow for it to be ready for deployment as a web application or application. Moreover, the CNN can also be adapted as an object detection model giving live results.

The dataset used for the CNN is an imbalance, which was a limitation. As a result, the model would learn features from the majority class better than others, a problem of having a shortage of data [68]; as a result, F1-score may have been a fairer metric to use. The findings are consistent with recent research and literature on the subject; the larger the dataset, the more features the model can learn [113]. When it comes to the effect of data on accuracy, Google researchers discovered that there are no "better algorithms. We have more data" [110].

Finding an appropriate dataset was difficult, as a large dataset requires sizable computational power to run deep NN; multi-class skin cancer datasets. Moreover, finding a multi-classification skin dataset was difficult; there are not many available and the ones that would be too large. The model can be improved if a more extensive, more balanced dataset includes varying skin types. Alternatively, using cross-validation as an evaluation protocol would also be beneficial. *K-fold cross-validation* was attempted. However, far too many errors did not improve the model (appendix ***6.15***).

In hindsight, a computer with far greater RAM would be more efficient in running CNN training. The risk of possible failure and extremely long training times were expected, and the challenge became increasingly tricky because the GPU is resilient to dense vector inputs [111]. The model would take hours to complete during the training, mainly when epochs are set to over 20-25. Figure 4.18 shows 86% of the memory of the local machine being occupied whilst training.

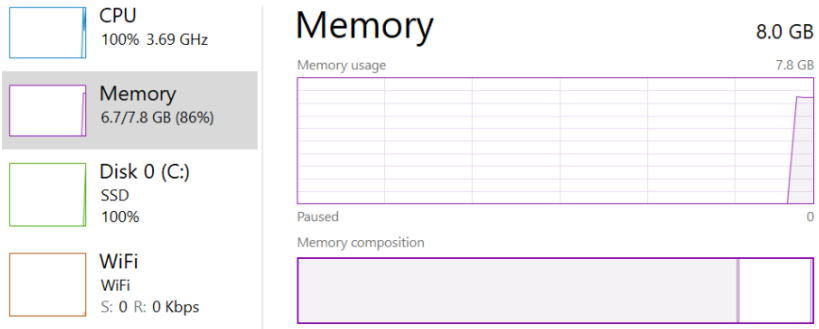


Figure 4.18 RAM of local machine whilst training CNN

5. Conclusion

This project aimed to build a CNN classification model for autonomous skin lesion diagnosis of 9 skin lesions found in the ISIC dataset, which was preprocessed and split into training, test and validation subsets. Moreover, a simple MLP was constructed to demonstrate trial and error investigation and exhibit techniques to mitigate overfitting.

The most optimal CNN structure consists of: 2 convolutional layers of filter sizes 32 and 64, max-pooling layers following the convolutional layers, a flatten layer to vectorise the output of convolutions, a dense layer of unit size 128 and a softmax output layer. The most optimal model was contrived from doing thorough experimentation, to which I find that Adam optimiser performs far better than RMSprop when classifying skin lesions. The model achieved 90.1% accuracy when validated validation data and had achieved 73.7% on solely unseen test data. Moreover, evaluating the MLP on test data resulted in 95.6% accuracy. Based on the results, the model proved reasonable accuracy, although not at the level where it can be confidently deployed into healthcare units. With further research, quality & quantity of data, and working around computational limits, this project has the potential to reach such heights.

6. Appendix

```
PATH = '../Binary classification of skin cancer/data 9 class/'

train_file = os.path.join(PATH, 'Train')
test_file = os.path.join(PATH, 'Test')
```

Figure 6.1 Retrieving file from correct file path

```
train_actinic_file = os.path.join(train_file, 'actinic keratosis')
train_basal_file = os.path.join(train_file, 'basal cell carcinoma')
train_dermato_file = os.path.join(train_file, 'dermatofibroma')
train_melanoma_file = os.path.join(train_file, 'melanoma')
train_nevus_file = os.path.join(train_file, 'nevus')
train_pigment_file = os.path.join(train_file, 'pigmented benign keratosis')
train_seborr_file = os.path.join(train_file, 'seborrheic keratosis')
train_squamous_file = os.path.join(train_file, 'squamous cell carcinoma')
train_vascular_file = os.path.join(train_file, 'vascular lesion')
```

Figure 6.2 Retrieving training data from correct file path

```
test_actinic_file = os.path.join(test_file, 'actinic keratosis')
test_basal_file = os.path.join(test_file, 'basal cell carcinoma')
test_dermato_file = os.path.join(test_file, 'dermatofibroma')
test_melanoma_file = os.path.join(test_file, 'melanoma')
test_nevus_file = os.path.join(test_file, 'nevus')
test_pigment_file = os.path.join(test_file, 'pigmented benign keratosis')
test_seborr_file = os.path.join(test_file, 'seborrheic keratosis')
test_squamous_file = os.path.join(test_file, 'squamous cell carcinoma')
test_vascular_file = os.path.join(test_file, 'vascular lesion')
```

Figure 6.3 Retrieving test data correct from correct file path

```
train_img_gen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
    brightness_range=[0.8,1.2],
    horizontal_flip=True

)
test_img_gen = ImageDataGenerator(rescale=1./255)
val_data_gen = ImageDataGenerator(rescale=1./255)
```

Figure 6.4 data preprocessing, vectorising data into tensors

```
train_data_generator = train_img_gen.flow_from_directory(batch_size=128,
                                                         directory=train_file,
                                                         shuffle=True,
                                                         target_size=(224, 224),
                                                         class_mode='categorical')

test_data_generator = test_img_gen.flow_from_directory(batch_size=128,
                                                       directory=test_file,
                                                       target_size=(224, 224),
                                                       class_mode='categorical')

validation_generator = train_img_gen.flow_from_directory(directory=train_file,
                                                       batch_size=128,
                                                        shuffle=False,

                                                        target_size=(224, 224),
                                                      class_mode='categorical',
                                                       subset='validation'
                                                      )
```

Figure 6.5 data preprocessing, imageDataGerator method creating data generators for train, test and validation sets

CNN-Baseline

```
model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_42 (Conv2D)          (None, 222, 222, 1)       28

 max_pooling2d_42 (MaxPoolin  (None, 111, 111, 1)      0
 g2D)

 flatten_14 (Flatten)        (None, 12321)             0

 dense_25 (Dense)            (None, 9)                 110898

=================================================================
Total params: 110,926
Trainable params: 110,926
Non-trainable params: 0
_________________________________________________________________
```

Figure 6.6 baseline model summary (***4.2.1***)

Exp no.	epochs	Val. acc	Val. loss	No. conv	filters	Layer depth	optimiser	overfit/ underfit	Optimal epoch
1	10	0.4910	1.5008	1	1	4	Rmsprop = 0.001	Overfit 2 epochs	8

Figure 6.7 table for section ***4.2.1***

CNN-Model that overfits

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 222, 222, 64)      1792

 max_pooling2d_4 (MaxPooling  (None, 111, 111, 64)     0
 2D)

 conv2d_5 (Conv2D)           (None, 109, 109, 128)     73856

 max_pooling2d_5 (MaxPooling  (None, 54, 54, 128)      0
 2D)

 flatten_2 (Flatten)         (None, 373248)            0

 dense_4 (Dense)             (None, 64)                23887936

 dense_5 (Dense)             (None, 9)                 585

=================================================================
Total params: 23,964,169
Trainable params: 23,964,169
Non-trainable params: 0
_________________________________________________________________
```

Figure 6.8 model summary for (***4.2.2***)

Exp no.	epochs	Val. acc	Val. loss	No. conv	filters	Layer depth	optimiser	overfit/ underfit	Optimal epoch
2	20	0.8356	0.4379	2	64-128	7	Rmsprop = 0.001	Overfit 2.5 epochs	20

Figure 6.9 table for (***4.2.2***)

Exp no.	epochs	Val. acc	Val. loss	No. conv	filters	Layer depth	optimiser	overfit/ underfit	Optimal epoch
1	10	0.4910	1.5008	1	1	4	Rmsprop = 0.001	Overfit 2 epochs	8
2	20	0.8356	0.4379	2	64-128	7	Rmsprop = 0.001	Overfit 2.5 epochs	20

Figure 6.10 table for (***4.2.3***)

CNN Experiments

CNN-filter size

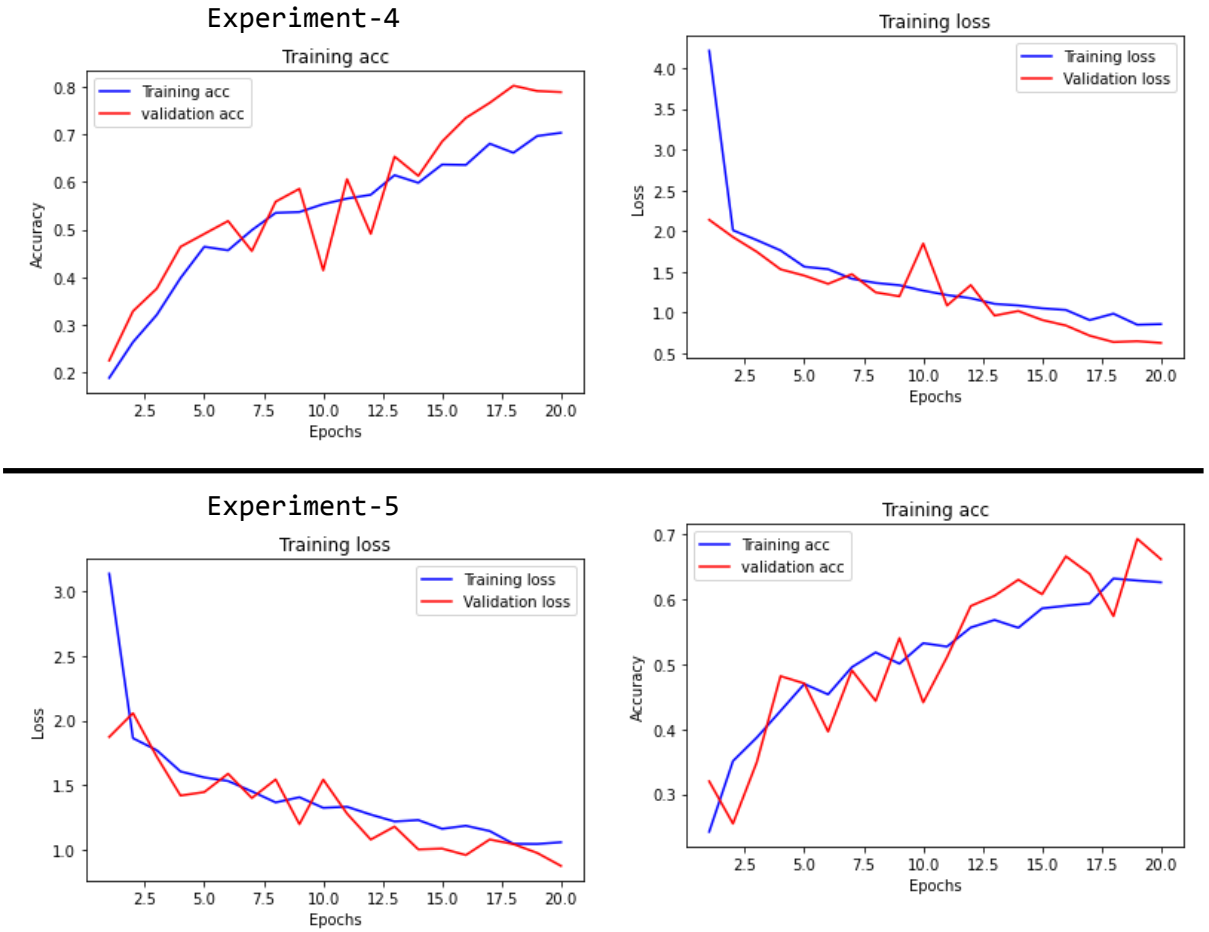


Figure 6.11 accuracy & loss of model section ***4.2.3 filter size***

epochs	Val. acc	Val. loss	No. conv	filters	Layer depth	optimiser	overfit/ underfit	Optimal epoch
20	0.8536	0.4526	2	32-64	7	Rmsprop = 0.001	Overfit 6 epochs	20

Figure 6.12 table for section ***4.2.3***

CNN-layer depth

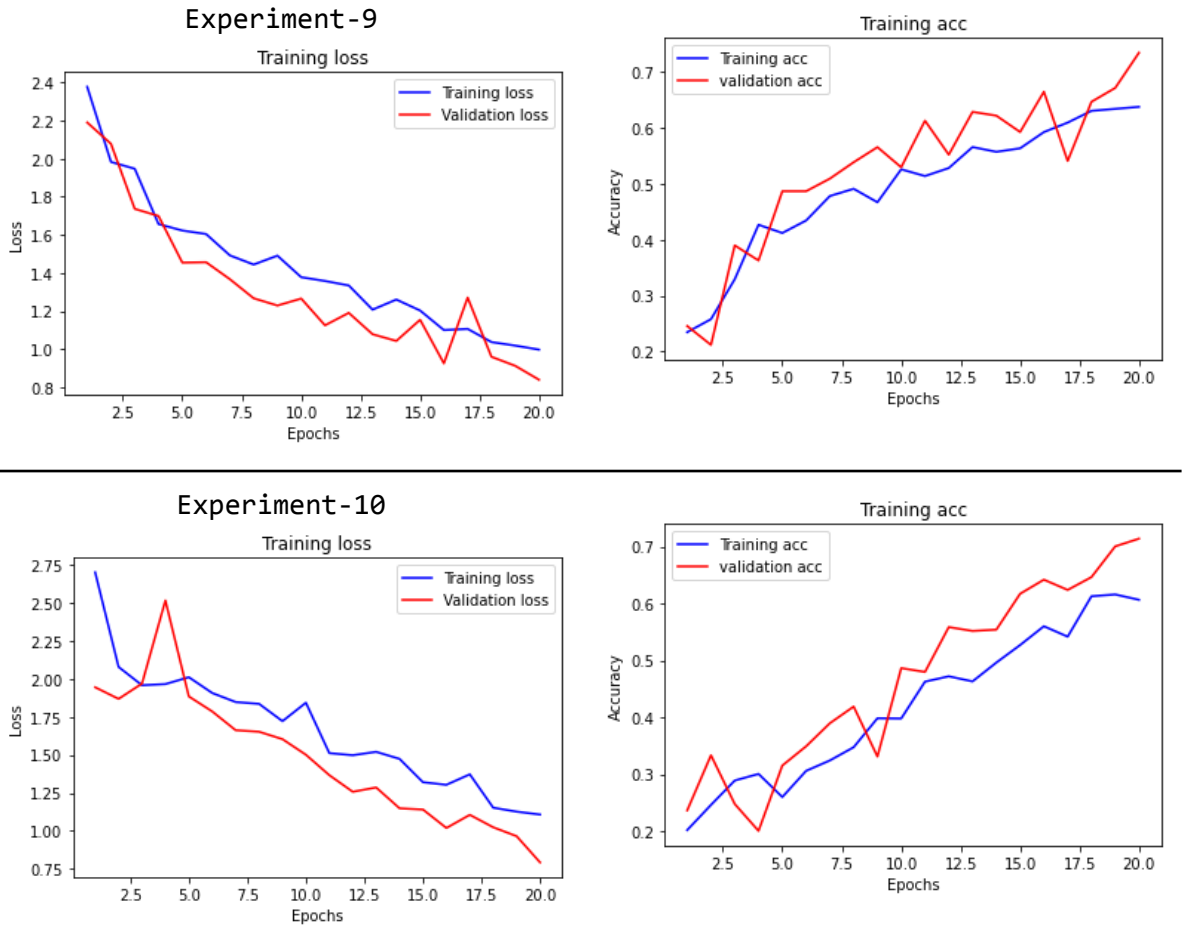
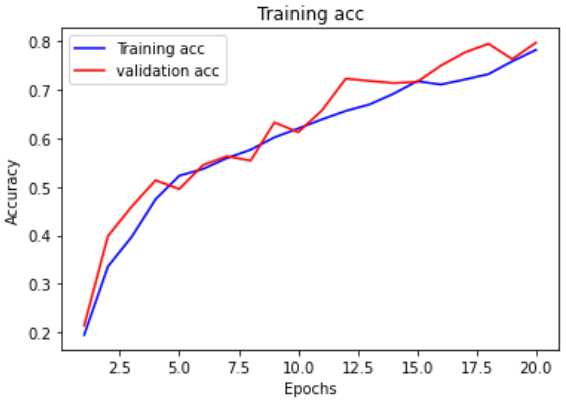


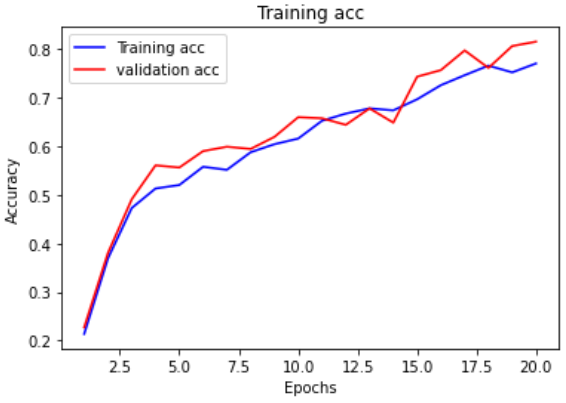
Figure 6.13 accuracy & loss of model **section *4.2.3 layer depth***

CNN-Optimiser experiments

Experiment-11



Experiment-12



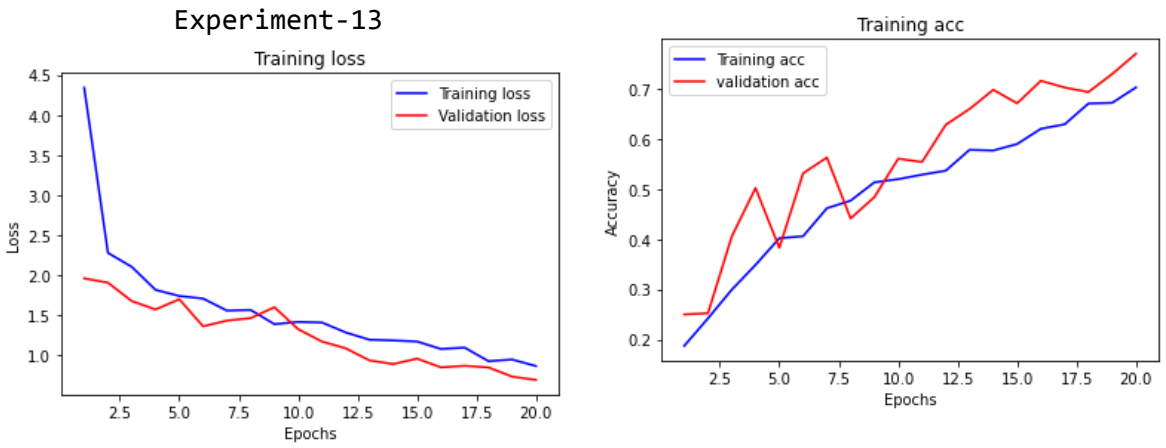


Figure 6.14 optimiser experiments section ***4.2.3 Optimiser***

CNN-K-fold cross validation

```
kfold = KFold(n_splits=num_folds, shuffle=True)
fold_no = 1

for train, test in kfold.split(train_data_generator):

  # Define the model architecture
#first conv relu should be bigger
  model = Sequential([

    layers.Conv2D(32, 3, activation='relu', input_shape=(224,224,3)),
    layers.MaxPooling2D(pool_size=(2,2)),
    layers.Conv2D(64, 3, activation='relu', input_shape=(224,224,3)),
    layers.MaxPooling2D(pool_size=(2,2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(9, activation='softmax')
])
  # Compile the model
  model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
print('------------------------------------------------------------------------')
print(f'Training for fold {fold_no} ...')

history = model.fit(
    train_data_generator,
    batch_size =20,
    epochs=2,
)
  # Generate generalization metrics
scores = model.evaluate(validation_generator, verbose=0)
print('Score for fold {fold_no}: {model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')
acc_per_fold.append(scores[1] * 100)
loss_per_fold.append(scores[0])

  # Increase fold number
fold_no = fold_no + 1

print('------------------------------------------------------------------------')
print('Score per fold')
for i in range(0, len(acc_per_fold)):
    print('------------------------------------------------------------------------')
    print(f'> Fold {i+1} - Loss: {loss_per_fold[i]} - Accuracy: {acc_per_fold[i]}%')
print('------------------------------------------------------------------------')
print('Average scores for all folds:')
print(f'> Accuracy: {np.mean(acc_per_fold)} (+- {np.std(acc_per_fold)})')
print(f'> Loss: {np.mean(loss_per_fold)}')
```

```
------------------------------------------------------------------------
Training for fold 1 ...
Epoch 1/2
18/18 [==============================] - 114s 6s/step - loss: 3.6904 - accuracy: 0.2117
Epoch 2/2
18/18 [==============================] - 93s 5s/step - loss: 1.9824 - accuracy: 0.3015
Score for fold {fold_no}: {model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%
------------------------------------------------------------------------
Score per fold
------------------------------------------------------------------------
> Fold 1 - Loss: 1.7441356182098389 - Accuracy: 33.33333432674408%
------------------------------------------------------------------------
Average scores for all folds:
> Accuracy: 33.33333432674408 (+- 0.0)
> Loss: 1.7441356182098389
------------------------------------------------------------------------
```

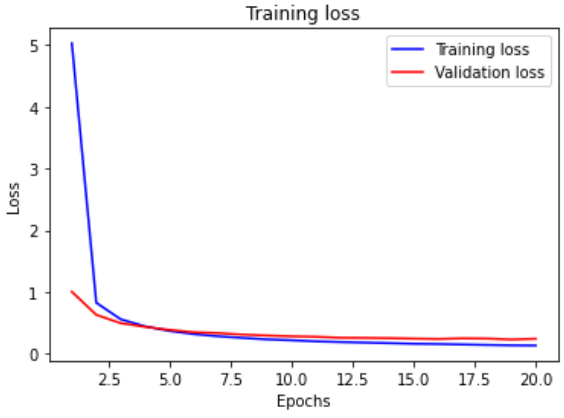
```
<ipython-input-608-9205892d4f95> in <module>
     23 print(f'Training for fold {fold_no} ...')
     24
---> 25 history = model.fit(
     26     train_data_generator,
     27     batch_size =20,

~\anaconda3\lib\site-packages\keras\utils\traceback_utils.py in error_handler(*args, **kwargs)
     62     filtered_tb = None
     63     try:
---> 64       return fn(*args, **kwargs)
     65     except Exception as e:  # pylint: disable=broad-except
     66       filtered_tb = _process_traceback_frames(e.__traceback__)
```

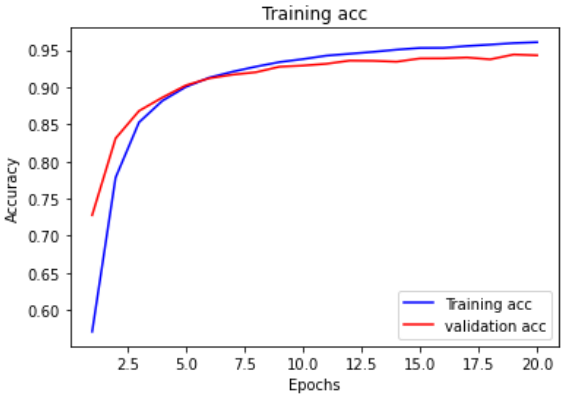
Figure 6.15 k-fold cross validation failure section ***4.2.6***

MLP experiments (section 4.1)

MLP - unit size experiments

32-32





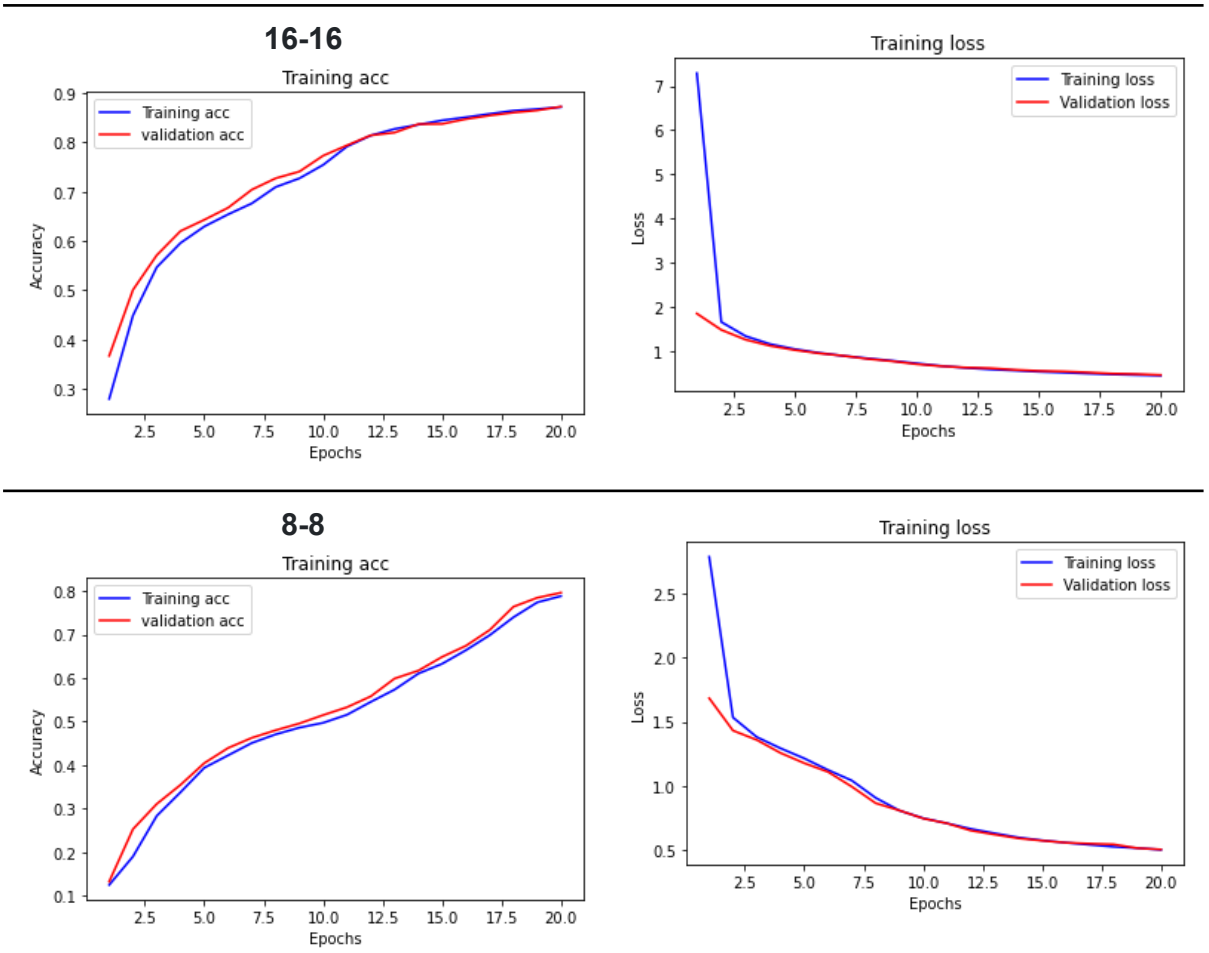
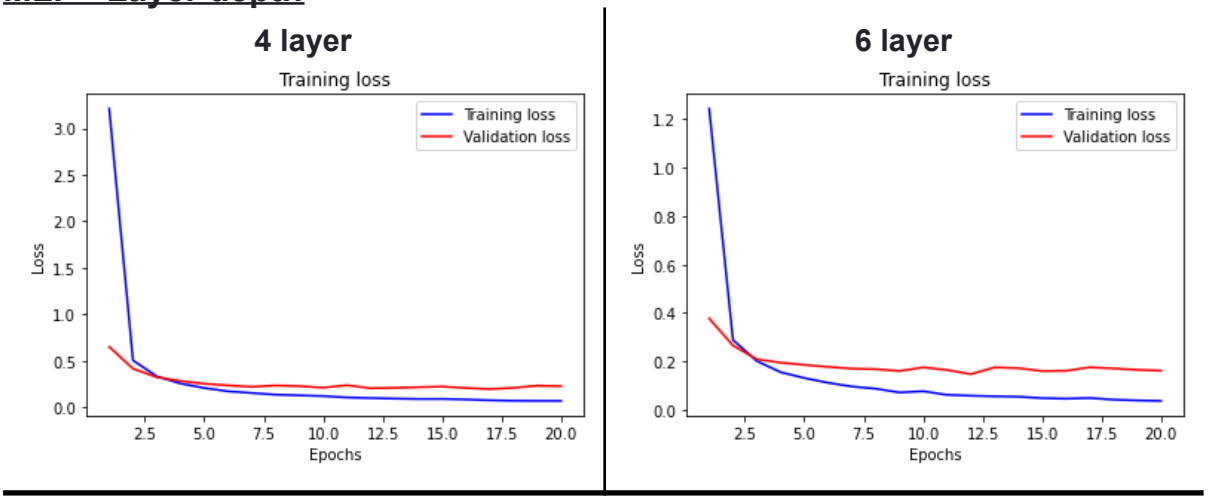


Figure 6.16 unit experiments section ***4.1.1***

MLP - Layer depth



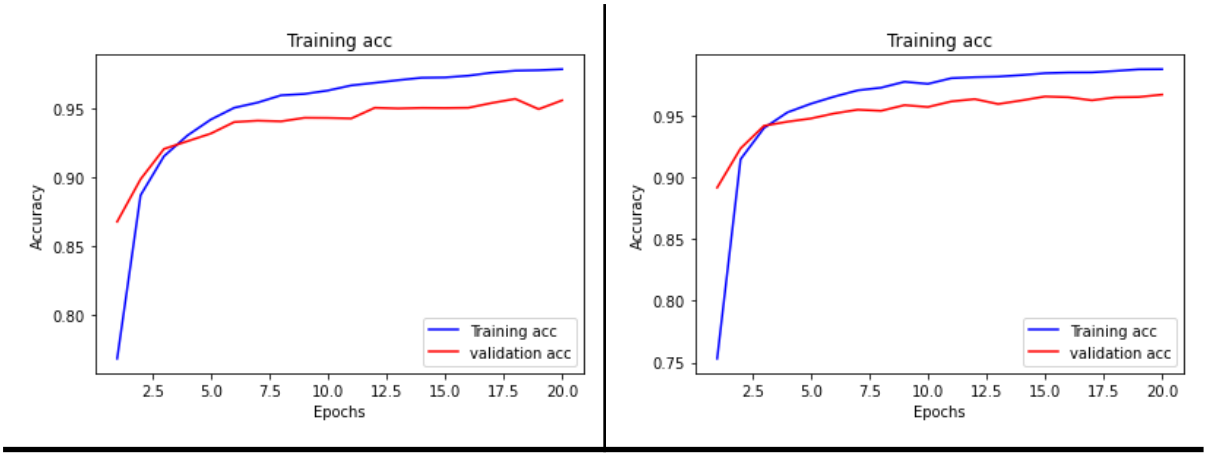


Figure 6.17 layer depth experiments section ***4.1.1***

MLP - dropout experiments

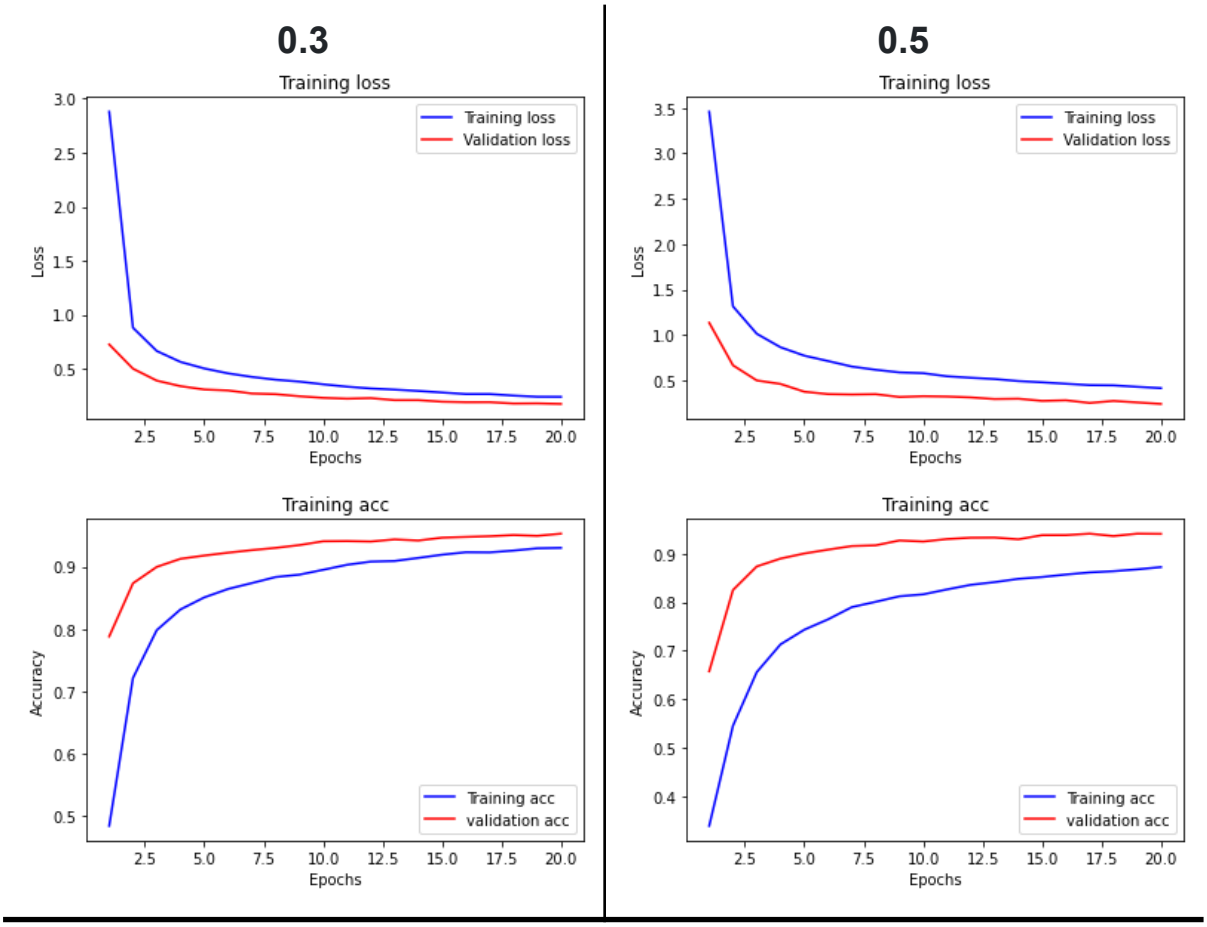
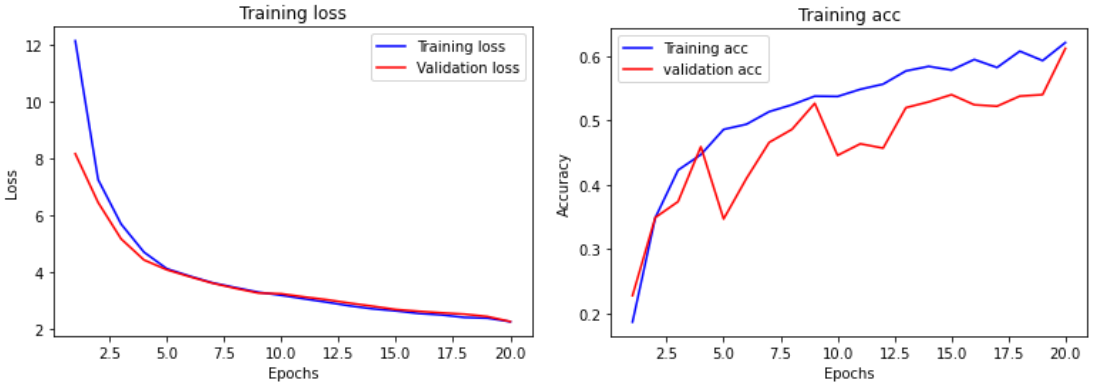


Figure 6.18 dropout experiments section ***4.1.1***

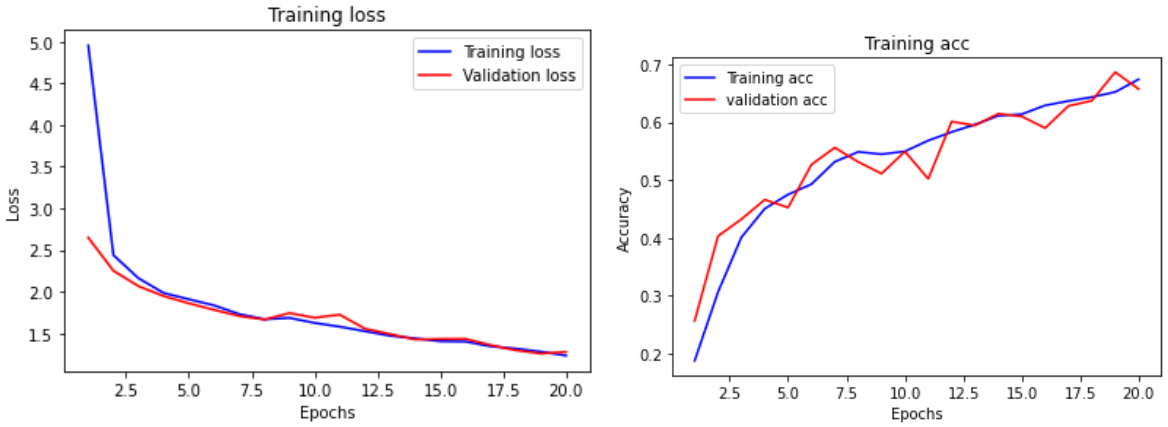
MLP - weight regullarisation experiments

L1 exp

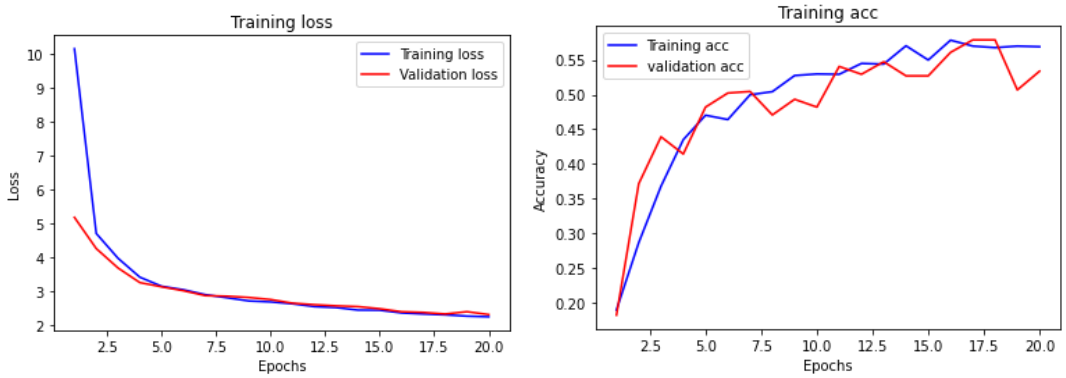
Exp 23



Exp 24

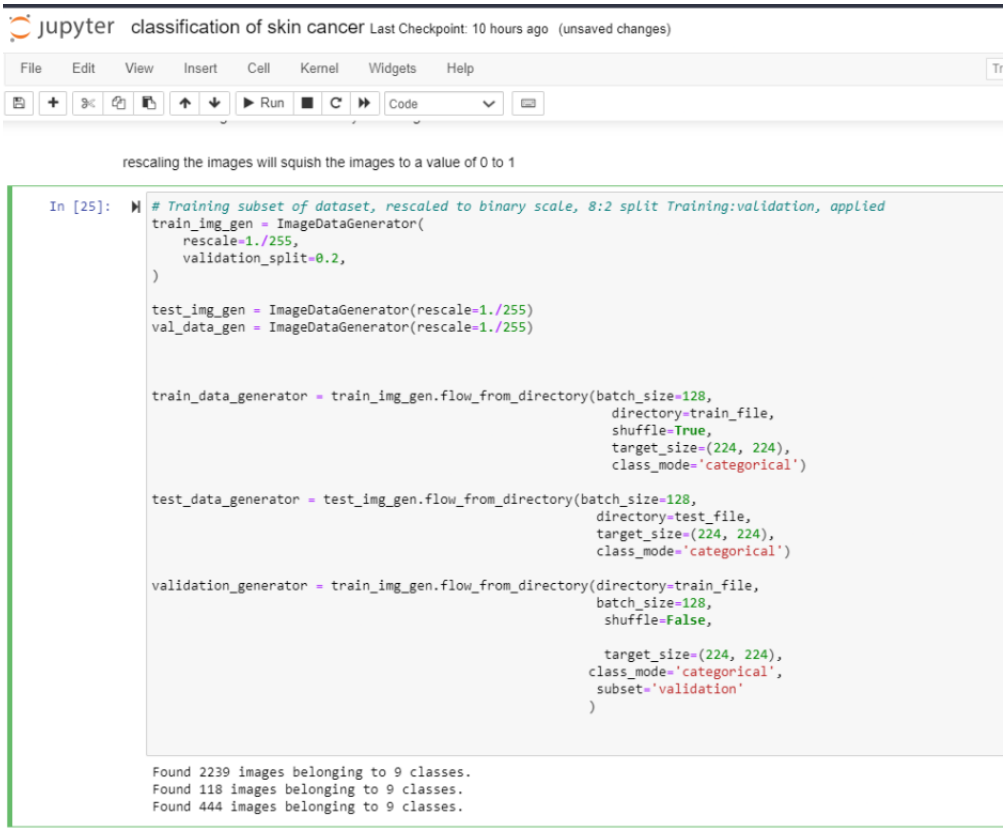


Exp 25



CNN code

ImageDataGenerator



Model while being trained

er classification of skin cancer Last Checkpoint: 10 hours ago (unsaved changes)

Logout

View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

```
        train_data_generator,
        batch_size =128,
        epochs=20,
        validation_data=validation_generator,
    )
```

```
Epoch 1/20
18/18 [==============================] - 125s 7s/step - loss: 5.1332 - accuracy: 0.2108 - val_loss: 1.7959 - val_accuracy:
0.3536
Epoch 2/20
18/18 [==============================] - 132s 7s/step - loss: 1.7193 - accuracy: 0.3716 - val_loss: 1.6023 - val_accuracy:
0.4887
Epoch 3/20
18/18 [==============================] - 140s 8s/step - loss: 1.5066 - accuracy: 0.4819 - val_loss: 1.4106 - val_accuracy:
0.5360
Epoch 4/20
18/18 [==============================] - 135s 7s/step - loss: 1.3843 - accuracy: 0.5252 - val_loss: 1.2735 - val_accuracy:
0.5383
Epoch 5/20
18/18 [==============================] - 136s 7s/step - loss: 1.2308 - accuracy: 0.5802 - val_loss: 1.0923 - val_accuracy:
0.6261
Epoch 6/20
18/18 [==============================] - 134s 7s/step - loss: 1.1072 - accuracy: 0.6195 - val_loss: 1.0055 - val_accuracy:
0.6779
Epoch 7/20
18/18 [==============================] - 136s 8s/step - loss: 1.0347 - accuracy: 0.6284 - val_loss: 0.9799 - val_accuracy:
0.6734
Epoch 8/20
18/18 [==============================] - 134s 7s/step - loss: 0.9652 - accuracy: 0.6789 - val_loss: 0.8834 - val_accuracy:
0.6757
Epoch 9/20
18/18 [==============================] - 154s 9s/step - loss: 0.8371 - accuracy: 0.6994 - val_loss: 0.7185 - val_accuracy:
0.7838
Epoch 10/20
18/18 [==============================] - 146s 8s/step - loss: 0.7503 - accuracy: 0.7418 - val_loss: 0.6415 - val_accuracy:
0.8108
Epoch 11/20
 2/18 [==>...........................] - ETA: 2:11 - loss: 0.6343 - accuracy: 0.7812
```

]:

```
history_dict = history.history
history_dict.keys()
history_dict['loss']
history_dict['val_loss']
```

Testing model on unseen test data

Testing the Model on Unseen Test data

In [99]:

```
from keras.models import load_model
model_x = load_model('exp_32.h5')

history = model_x.fit(
        test_data_generator,
        epochs=20,
    )
```

```
1/1 [==============================] - 13s 13s/step - loss: 1.2620 - accuracy: 0.5085
Epoch 12/20
1/1 [==============================] - 13s 13s/step - loss: 1.1298 - accuracy: 0.6102
Epoch 13/20
1/1 [==============================] - 13s 13s/step - loss: 1.0692 - accuracy: 0.6610
Epoch 14/20
1/1 [==============================] - 13s 13s/step - loss: 1.0157 - accuracy: 0.6695
Epoch 15/20
1/1 [==============================] - 13s 13s/step - loss: 0.9223 - accuracy: 0.6949
Epoch 16/20
1/1 [==============================] - 14s 14s/step - loss: 0.8658 - accuracy: 0.7203
Epoch 17/20
1/1 [==============================] - 14s 14s/step - loss: 0.8093 - accuracy: 0.7119
Epoch 18/20
1/1 [==============================] - 14s 14s/step - loss: 0.7373 - accuracy: 0.7542
Epoch 19/20
1/1 [==============================] - 14s 14s/step - loss: 0.7031 - accuracy: 0.7458
Epoch 20/20
1/1 [==============================] - 14s 14s/step - loss: 0.6737 - accuracy: 0.7373
```

Model after completing all experiments

- ***Model structure***

optimal Model

[75]:

```
model = Sequential([
# use drop out on first layer of relu

    layers.Conv2D(32, 3, activation='relu' ,input_shape=(224,224,3)),
    layers.Dropout(0.3),
    layers.MaxPooling2D(pool_size=(2,2)),
    layers.Conv2D(64, 3, activation='relu'),
    layers.MaxPooling2D(pool_size=(2,2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(9, activation='softmax')
])
```

- ***Model summary of optimal model***

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_14 (Conv2D)           (None, 222, 222, 32)      896

dropout_5 (Dropout)          (None, 222, 222, 32)      0

max_pooling2d_14 (MaxPoolin  (None, 111, 111, 32)      0
g2D)

conv2d_15 (Conv2D)           (None, 109, 109, 64)      18496

max_pooling2d_15 (MaxPoolin  (None, 54, 54, 64)        0
g2D)

flatten_7 (Flatten)          (None, 186624)            0

dense_14 (Dense)             (None, 128)               23888000

dense_15 (Dense)             (None, 9)                 1161

=================================================================
Total params: 23,908,553
Trainable params: 23,908,553
Non-trainable params: 0
_________________________________________________________________
```

- ***Training optimal model***

```
In [77]: model.compile(optimizer="Adam", loss='categorical_crossentropy', metrics=['accuracy'])
```

```
In [78]: history = model.fit(
             train_data_generator,
             batch_size =128,
             epochs=20,
             validation_data=validation_generator,
         )
         model.save('exp_32.h5')
```

```
y: 0.8378
Epoch 15/20
18/18 [==============================] - 144s 8s/step - loss: 0.5293 - accuracy: 0.8200 - val_loss: 0.5623 - val_accurac
y: 0.8153
Epoch 16/20
18/18 [==============================] - 141s 8s/step - loss: 0.4986 - accuracy: 0.8245 - val_loss: 0.5340 - val_accurac
y: 0.8536
Epoch 17/20
18/18 [==============================] - 144s 8s/step - loss: 0.4647 - accuracy: 0.8280 - val_loss: 0.4250 - val_accurac
y: 0.8896
Epoch 18/20
18/18 [==============================] - 142s 8s/step - loss: 0.4007 - accuracy: 0.8557 - val_loss: 0.4232 - val_accurac
y: 0.8694
Epoch 19/20
18/18 [==============================] - 143s 8s/step - loss: 0.4171 - accuracy: 0.8455 - val_loss: 0.3789 - val_accurac
y: 0.9234
Epoch 20/20
18/18 [==============================] - 143s 8s/step - loss: 0.3555 - accuracy: 0.8665 - val_loss: 0.3436 - val_accurac
y: 0.9009
```

MLP code

Loading MNIST dataset

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Data preprocessing Dataset

```
# Change from matrix to array of dimension 28x28 to array of dimension 784
dimension_new_data = np.prod(train_images.shape[1:])
train_data = train_images.reshape(train_images.shape[0], dimension_new_data)
test_data = test_images.reshape(test_images.shape[0], dimension_new_data)
```

```
# Change to float datatype
train_data = train_data.astype('float32')
test_data = test_data.astype('float32')
```

```
# Change the labels from integer to categorical data
train_labels_cat = to_categorical(train_labels)
test_labels_cat = to_categorical(test_labels)
```

```
train_labels_cat
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 1., 0.]], dtype=float32)
```

Data visualisation of MNIST dataset

Training model

Creating model

In [361]:
```
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(dimension_new_data,)))
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(64, activation='relu'))

model.add(Dense(digits, activation='softmax'))
```

In [249]:
```
model.summary()
```

```
Model: "sequential_32"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_108 (Dense)           (None, 64)                50240

 dropout_10 (Dropout)        (None, 64)                0

 dense_109 (Dense)           (None, 64)                4160

 dense_110 (Dense)           (None, 10)                650

=================================================================
Total params: 55,050
Trainable params: 55,050
Non-trainable params: 0
_________________________________________________________________
```

In [362]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

In [363]:
```
history = model.fit(train_data,
                    train_labels_cat,
                    batch_size=256,
                    epochs=20,
                    validation_data=(test_data, test_labels_cat)
                   )
```

```
cy: 0.9418
Epoch 15/20
235/235 [==============================] - 1s 3ms/step - loss: 0.2811 - accuracy: 0.9191 - val_loss: 0.1960 - val_accura
cy: 0.9464
Epoch 16/20
235/235 [==============================] - 1s 3ms/step - loss: 0.2668 - accuracy: 0.9230 - val_loss: 0.1913 - val_accura
cy: 0.9476
Epoch 17/20
235/235 [==============================] - 1s 3ms/step - loss: 0.2668 - accuracy: 0.9228 - val_loss: 0.1917 - val_accura
cy: 0.9487
Epoch 18/20
235/235 [==============================] - 1s 3ms/step - loss: 0.2527 - accuracy: 0.9257 - val_loss: 0.1793 - val_accura
cy: 0.9506
Epoch 19/20
235/235 [==============================] - 1s 4ms/step - loss: 0.2417 - accuracy: 0.9293 - val_loss: 0.1805 - val_accura
cy: 0.9493
Epoch 20/20
235/235 [==============================] - 1s 4ms/step - loss: 0.2411 - accuracy: 0.9300 - val_loss: 0.1756 - val_accura
cy: 0.9528
```

Evaluating model

```
[test_loss, test_acc] = model.evaluate(test_data, test_labels_cat)
print("Loss = {}, accuracy = {}".format(test_loss, test_acc))
```

```
313/313 [==============================] - 0s 891us/step - loss: 0.1269 - accuracy: 0.9653
Loss = 0.12693554162979126, accuracy = 0.9653000235557556
```

Project link = https://github.com/7unayd/ALIAHMED-JUNEAD-FINAL-PROJECT
https://drive.google.com/drive/folders/1DixMiNmfG5SCskdpbEsx5PAhLkr_UDXt?usp=sharing

7. Bibliography

[1]
P. Rajpurkar *et al.*, "CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning," *arXiv.org*, Nov. 14, 2017. [Online]. Available: https://arxiv.org/abs/1711.05225. [Accessed: Apr. 04, 2022]

[2]
S. S. Yadav and S. M. Jadhav, "Deep convolutional neural network based medical image classification for disease diagnosis," *Journal of Big Data*, vol. 6, no. 1, Dec. 2019, doi: 10.1186/s40537-019-0276-2.

[3]
"Skin Cancer (Non-Melanoma) - Introduction," *Cancer.net*, Jun. 25, 2012. [Online]. Available: https://www.cancer.net/cancer-types/skin-cancer-non-melanoma/introduction#:~:text=A%20tumor%20can%20be%20cancerous. [Accessed: Apr. 05, 2022]

[4]
D. Sarnoff, "Skin Cancer Information - The Skin Cancer Foundation," *The Skin Cancer Foundation*, 2018. [Online]. Available: https://www.skincancer.org/skin-cancer-information/. [Accessed: Apr. 05, 2022]

[5]
"Cancer," *stanfordhealthcare.org*. [Online]. Available: https://stanfordhealthcare.org/medical-conditions/cancer/cancer.html#:~:text=What%20is%20the%20difference%20between. [Accessed: Apr. 05, 2022]

[6]
The American Cancer Society medical and editorial content team, "Survival Rates for Melanoma Skin Cancer," *Cancer.org*, Mar. 01, 2015. [Online]. Available: https://www.cancer.org/cancer/melanoma-skin-cancer/detection-diagnosis-staging/survival-rates-for-melanoma-skin-cancer-by-stage.html

[7]
L. E. Davis, S. C. Shalin, and A. J. Tackett, "Current state of melanoma diagnosis and treatment," *Cancer Biology & Therapy*, vol. 20, no. 11, pp. 1366–1379, Aug. 2019, doi: 10.1080/15384047.2019.1640032. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6804807/. [Accessed: Apr. 05, 2022]

[8]
"TNM staging | Melanoma skin cancer | Cancer Research UK," *Cancerresearchuk.org*, May 21, 2019. [Online]. Available: https://www.cancerresearchuk.org/about-cancer/melanoma/stages-types/tnm-staging. [Accessed: Apr. 05, 2022]

[9]
W. Onsoi, J. Chaiyarit, and L. Techasatian, "Common misdiagnoses and prevalence of dermatological disorders at a pediatric tertiary care center," *Journal of International Medical*

Research, vol. 48, no. 2, p. 030006051987349, Sep. 2019, doi: 10.1177/0300060519873490. [Online]. Available: https://journals.sagepub.com/doi/full/10.1177/0300060519873490. [Accessed: Apr. 06, 2022]

[10]
H. Singh, A. N. D. Meyer, and E. J. Thomas, "The frequency of diagnostic errors in outpatient care: estimations from three large observational studies involving US adult populations," *BMJ Quality & Safety*, vol. 23, no. 9, pp. 727–731, Apr. 2014, doi: 10.1136/bmjqs-2013-002627.

[11]
H. Singh *et al.*, "Errors of Diagnosis in Pediatric Practice: A Multisite Survey," *Pediatrics*, vol. 126, no. 1, pp. 70–79, Jul. 2010, doi: 10.1542/peds.2009-3218. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2921702/. [Accessed: Apr. 06, 2022]

[12]
S. Liu, S. Liu, W. Cai, S. Pujol, R. Kikinis, and D. Feng, "Early diagnosis of Alzheimer's disease with deep learning," *researchers.mq.edu.au*, Jan. 01, 2014. [Online]. Available: https://researchers.mq.edu.au/en/publications/early-diagnosis-of-alzheimers-disease-with-deep-learning. [Accessed: Apr. 06, 2022]

[13]
C. R. Goodall, "Data Mining of Massive Datasets in Healthcare," *Journal of Computational and Graphical Statistics*, vol. 8, no. 3, p. 620, Sep. 1999, doi: 10.2307/1390880.

[14]
P. Kostkova *et al.*, "Who Owns the Data? Open Data for Healthcare," *Frontiers in Public Health*, vol. 4, Feb. 2016, doi: 10.3389/fpubh.2016.00007.

[15]
L. Aviñó, M. Ruffini, and R. Gavaldà, "Generating Synthetic but Plausible Healthcare Record Datasets," *arXiv:1807.01514 [cs, stat]*, Jul. 2018 [Online]. Available: https://arxiv.org/abs/1807.01514. [Accessed: Apr. 06, 2022]

[16]
J. McCarthy, "What is AI?," *Stanford.edu*, 2012. [Online]. Available: http://jmc.stanford.edu/articles/whatisai.html. [Accessed: Apr. 06, 2022]

[17]
H. Hsu, "AI and Play, Part 1: How Games Have Driven Two Schools of AI Research," *CHM*, Jul. 23, 2020. [Online]. Available: https://computerhistory.org/blog/ai-and-play-part-1-how-games-have-driven-two-schools-of-ai-research/. [Accessed: Apr. 06, 2022]

[18]
IBM Cloud Education, "What is Machine Learning?," *www.ibm.com*, Jul. 15, 2020. [Online]. Available: https://www.ibm.com/uk-en/cloud/learn/machine-learning. [Accessed: Apr. 06, 2022]

[19]
F. Chollet, *Deep Learning with Python*. Shelter Island (New York, Estados Unidos): Manning, Cop, 2018.

[20]
https://www.facebook.com/jason.brownlee.39, "A Gentle Introduction to Statistical Power and Power Analysis in Python," *Machine Learning Mastery*, Jul. 12, 2018. [Online]. Available: https://machinelearningmastery.com/statistical-power-and-power-analysis-in-python/. [Accessed: Apr. 07, 2022]

[21]
F. Chollet, *Deep Learning with Python*. Shelter Island (New York, Estados Unidos): Manning, Cop, 2018, p. 112.

[22]
Yufeng G, "The 7 Steps of Machine Learning," *Medium*, Aug. 31, 2017. [Online]. Available: https://towardsdatascience.com/the-7-steps-of-machine-learning-2877d7e5548e. [Accessed: Apr. 07, 2022]

[23]
Juan Cruz Martinez, "7 Steps of Machine Learning," *Livecodestream.dev*, Jun. 02, 2020. [Online]. Available: https://livecodestream.dev/post/7-steps-of-machine-learning/. [Accessed: Apr. 07, 2022]

[24]
"Convolutional Neural Networks - an overview | ScienceDirect Topics," *www.sciencedirect.com*, 2020. [Online]. Available: https://www.sciencedirect.com/topics/engineering/convolutional-neural-networks#:~:text=A%20CNN%20consists%20of%20neurons. [Accessed: Apr. 07, 2022]

[25]
R. M. Tharsanee, R. S. Soundariya, A. S. Kumar, M. Karthiga, and S. Sountharrajan, "7 - Deep convolutional neural network–based image classification for COVID-19 diagnosis," *ScienceDirect*, Jan. 01, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128245361000125. [Accessed: Apr. 07, 2022]

[26]
"Convolutional Neural Network (CNN)," *NVIDIA Developer*, Apr. 23, 2018. [Online]. Available: https://developer.nvidia.com/discover/convolutional-neural-network. [Accessed: Apr. 07, 2022]

[27]
R. Khandelwal, "Convolutional Neural Network: Feature Map and Filter Visualization," *Medium*, May 18, 2020. [Online]. Available: https://towardsdatascience.com/convolutional-neural-network-feature-map-and-filter-visualization-f75012a5a49c. [Accessed: Apr. 07, 2022]

[28]

S. Du, “Understanding Deep Self-attention Mechanism in Convolution Neural Networks,” *AI Salon*, May 29, 2020. [Online]. Available: https://medium.com/ai-salon/understanding-deep-self-attention-mechanism-in-convolution-neural-networks-e8f9c01cb251. [Accessed: Apr. 07, 2022]

[29]
D. Johnson, “Back Propagation Neural Network: Explained With Simple Example,” *www.guru99.com*, Mar. 08, 2022. [Online]. Available: https://www.guru99.com/backpropogation-neural-network.html. [Accessed: Apr. 07, 2022]

[30]
Piotr Skalski, “Gentle Dive into Math Behind Convolutional Neural Networks,” *Medium*, Apr. 12, 2019. [Online]. Available: https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9. [Accessed: Apr. 07, 2022]

[31]
N. Davis, “AI skin cancer diagnoses risk being less accurate for dark skin – study,” *the Guardian*, Nov. 09, 2021. [Online]. Available: https://www.theguardian.com/society/2021/nov/09/ai-skin-cancer-diagnoses-risk-being-less-accurate-for-dark-skin-study. [Accessed: Apr. 08, 2022]

[32]
D. Wen *et al.*, “Characteristics of publicly available skin cancer image datasets: a systematic review,” *The Lancet Digital Health*, vol. 0, no. 0, Nov. 2021, doi: 10.1016/S2589-7500(21)00252-1. [Online]. Available: https://www.thelancet.com/journals/landig/article/PIIS2589-7500(21)00252-1/fulltext

[33]
Simplilearn, “The Complete Guide to Machine Learning Steps,” *Simplilearn.com*, Nov. 12, 2021. [Online]. Available: https://www.simplilearn.com/tutorials/machine-learning-tutorial/machine-learning-steps. [Accessed: Apr. 08, 2022]

[34]
S. M. Dawes, S. Tsai, H. Gittleman, J. S. Barnholtz-Sloan, and J. S. Bordeaux, “Racial disparities in melanoma survival,” *Journal of the American Academy of Dermatology*, vol. 75, no. 5, pp. 983–991, Nov. 2016, doi: 10.1016/j.jaad.2016.06.006. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/27476974/. [Accessed: Apr. 09, 2022]

[35]
V. M. Harvey, H. Patel, S. Sandhu, S. F. Wallington, and G. Hinds, “Social Determinants of Racial and Ethnic Disparities in Cutaneous Melanoma Outcomes,” *Cancer Control*, vol. 21, no. 4, pp. 343–349, Oct. 2014, doi: 10.1177/107327481402100411. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4505912/. [Accessed: Apr. 09, 2022]

[36]
S. McDowell, “Study: Lack of Education About Melanoma May Contribute to Black-White Survival Disparities,” *www.cancer.org*, Jul. 23, 2019. [Online]. Available: https://www.cancer.org/latest-news/study-lack-of-education-about-melanoma-may-contribute-to-black-white-survival-disparities.html. [Accessed: Apr. 09, 2022]

[37]
K. H. Hall and R. P. Rapini, “Acral Lentiginous Melanoma,” *PubMed*, Jul. 28, 2021. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK559113/#:~:text=Acral%20lentiginous%20melanoma%20(ALM)%2C. [Accessed: Apr. 09, 2022]

[38]
Y. Wang, Y. Zhao, and S. Ma, “Racial differences in six major subtypes of melanoma: descriptive epidemiology,” *BMC Cancer*, vol. 16, no. 1, Aug. 2016, doi: 10.1186/s12885-016-2747-6.

[39]
J. C. Lester, “Why skin disease is often misdiagnosed in darker skin tones,” *www.ted.com*, Aug. 2021. [Online]. Available: https://www.ted.com/talks/jenna_c_lester_why_skin_disease_is_often_misdiagnosed_in_darker_skin_tones?language=en. [Accessed: Apr. 09, 2022]

[40]
A. Adelekun, G. Onyekaba, and J. B. Lipoff, “Skin color in dermatology textbooks: An updated evaluation and analysis,” *Journal of the American Academy of Dermatology*, Apr. 2020, doi: 10.1016/j.jaad.2020.04.084.

[41]
U. L. McFarling, “Lack of darker skin in textbooks, journals harms patients of color,” *STAT*, Jul. 21, 2020. [Online]. Available: https://www.statnews.com/2020/07/21/dermatology-faces-reckoning-lack-of-darker-skin-in-textbooks-journals-harms-patients-of-color/. [Accessed: Apr. 09, 2022]

[42]
J. Cruel, “Dermatology Has A Diversity Problem — & It Affects Black Women,” *www.refinery29.com*, Aug. 15, 2019. [Online]. Available: https://www.refinery29.com/en-gb/2019/08/240668/black-dermatologist-diversity-problem. [Accessed: Apr. 09, 2022]

[43]
“An Ongoing Commitment to Equity in Medicine,” *VisualDx*. [Online]. Available: https://www.visualdx.com/diversity/#:~:text=practicing%20exemplary%20medicine.-

[44]
A. S. Adamson and A. Smith, “Machine Learning and Health Care Disparities in Dermatology,” *JAMA Dermatology*, vol. 154, no. 11, p. 1247, Nov. 2018, doi: 10.1001/jamadermatol.2018.2348. [Online]. Available: https://jamanetwork.com/journals/jamadermatology/article-abstract/2688587. [Accessed: Apr. 12, 2022]

[45]

S. Levin, “A beauty contest was judged by AI and the robots didn’t like dark skin,” *The Guardian*, Sep. 08, 2016 [Online]. Available: https://www.theguardian.com/technology/2016/sep/08/artificial-intelligence-beauty-contest-doesnt-like-black-people. [Accessed: Apr. 12, 2022]

[46]
T. Simonite, “When It Comes to Gorillas, Google Photos Remains Blind,” *Wired*, Jan. 11, 2018. [Online]. Available: https://www.wired.com/story/when-it-comes-to-gorillas-google-photos-remains-blind/. [Accessed: Apr. 12, 2022]

[47]
A. Lashbrook, “AI-Driven Dermatology Could Leave Dark-Skinned Patients Behind,” *The Atlantic*, Aug. 16, 2018. [Online]. Available: https://www.theatlantic.com/health/archive/2018/08/machine-learning-dermatology-skin-color/567619/. [Accessed: Apr. 12, 2022]

[48]
G. P. Cannata, “Backpropagation in Fully Convolutional Networks (FCNs),” *Medium*, Feb. 03, 2021. [Online]. Available: https://towardsdatascience.com/backpropagation-in-fully-convolutional-networks-fcns-1a13b75fb56a. [Accessed: Apr. 12, 2022]

[49]
R. Kwiatkowski, “Gradient Descent Algorithm — a deep dive,” *Medium*, May 24, 2021. [Online]. Available: https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21. [Accessed: Apr. 12, 2022]

[50]
X. Zhou, “Understanding the Convolutional Neural Networks with Gradient Descent and Backpropagation,” *Journal of Physics: Conference Series*, vol. 1004, p. 012028, Apr. 2018, doi: 10.1088/1742-6596/1004/1/012028.

[51]
N. Donges, “Gradient Descent: An Introduction to One of Machine Learning’s Most Popular Algorithms,” *Built In*, Jul. 23, 2021. [Online]. Available: https://builtin.com/data-science/gradient-descent. [Accessed: Apr. 12, 2022]

[52]
R. Gomez, “Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names,” *Github.io*, May 23, 2018. [Online]. Available: https://gombru.github.io/2018/05/23/cross_entropy_loss/. [Accessed: Apr. 12, 2022]

[53]
Peltarion, “Categorical crossentropy loss function | Peltarion Platform,” *Peltarion.com*. [Online]. Available: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy. [Accessed: Apr. 12, 2022]

[54]
Eijaz Allibhai, “Holdout vs. Cross-validation in Machine Learning,” *Medium*, Oct. 03, 2018. [Online]. Available: https://medium.com/@eijaz/holdout-vs-cross-validation-in-machine-learning-7637112d3f8f. [Accessed: Apr. 12, 2022]

[55]
Krishni, “K-Fold Cross Validation,” *Medium*, Dec. 21, 2018. [Online]. Available: https://medium.datadriveninvestor.com/k-fold-cross-validation-6b8518070833. [Accessed: Apr. 12, 2022]

[56]
V. Jain, “Everything you need to know about ‘Activation Functions’ in Deep learning models,” *Medium*, Dec. 30, 2019. [Online]. Available: https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253. [Accessed: Apr. 12, 2022]

[57]
S. SHARMA, “Activation Functions in Neural Networks,” *Medium*, Sep. 06, 2017. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. [Accessed: Apr. 12, 2022]

[58]
Y. Adan, “What is the ReLU function formula? When would we use this? Why?,” *Quora*, 2019. [Online]. Available: https://www.quora.com/What-is-the-ReLU-function-formula-When-would-we-use-this-Why. [Accessed: Apr. 12, 2022]

[59]
H. Mujtaba, “An Introduction to Rectified Linear Unit (ReLU) | What is RelU?,” *GreatLearning*, Aug. 29, 2020. [Online]. Available: https://www.mygreatlearning.com/blog/relu-activation-function/. [Accessed: Apr. 12, 2022]

[60]
P. Ratan, “Convolutional Neural Network Made Easy for Data Scientists,” *Analytics Vidhya*, Oct. 28, 2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/. [Accessed: Apr. 12, 2022]

[61]
“2.1.3.1 - Range of Probabilities | STAT 200,” *PennState: Statistics Online Courses*. [Online]. Available: https://online.stat.psu.edu/stat200/lesson/2/2.1/2.1.3/2.1.3.1#:~:text=The%20probability%20of%20an%20impossible. [Accessed: Apr. 13, 2022]

[62]

P. Baheti, “12 Types of Neural Networks Activation Functions: How to Choose?,” *www.v7labs.com*, Mar. 08, 2022. [Online]. Available: https://www.v7labs.com/blog/neural-networks-activation-functions. [Accessed: Apr. 13, 2022]

[63]
K. Goyal, “6 Types of Activation Function in Neural Networks You Need to Know,” *upGrad blog*, Feb. 13, 2020. [Online]. Available: https://www.upgrad.com/blog/types-of-activation-function-in-neural-networks/#:~:text=The%20tanh%20function%20is%20much. [Accessed: Apr. 13, 2022]

[64]
J. Velasco, “A Smartphone-Based Skin Disease Classification Using MobileNet CNN,” *International Journal of Advanced Trends in Computer Science and Engineering*, pp. 2632–2637, Oct. 2019, doi: 10.30534/ijatcse/2019/116852019.

[65]
P. Hofl and Ph.D, “Artificial Intelligence Better than Dermatologists in Diagnosing Skin Cancer,” *Onco’Zine*, May 29, 2018. [Online]. Available: https://www.oncozine.com/artificial-intelligence-better-dermatologists-diagnosing-skin-cancer/#:~:text=Researchers%20have%20shown%20for%20the. [Accessed: Apr. 13, 2022]

[66]
H. A. Haenssle *et al.*, “Man against machine: diagnostic performance of a deep learning convolutional neural network for dermoscopic melanoma recognition in comparison to 58 dermatologists,” *Annals of Oncology*, vol. 29, no. 8, pp. 1836–1842, May 2018, doi: 10.1093/annonc/mdy166. [Online]. Available: https://academic.oup.com/annonc/article/29/8/1836/5004443. [Accessed: Apr. 13, 2022]

[67]
Y. N. Fu’adah, N. C. Pratiwi, M. A. Pramudito, and N. Ibrahim, “Convolutional Neural Network (CNN) for Automatic Skin Cancer Classification System,” *IOP Conference Series: Materials Science and Engineering*, vol. 982, p. 012005, Dec. 2020, doi: 10.1088/1757-899x/982/1/012005.

[68]
L. Alzubaidi *et al.*, “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,” *Journal of Big Data*, vol. 8, no. 1, Mar. 2021, doi: 10.1186/s40537-021-00444-8. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8010506/. [Accessed: Apr. 01, 2021]

[69]
M. Tripathi, “Underfitting and Overfitting in Machine Learning,” *datascience.foundation*, Jun. 13, 2020. [Online]. Available: https://datascience.foundation/sciencewhitepaper/underfitting-and-overfitting-in-machine-learning. [Accessed: Apr. 14, 2022]

[70]

B. Zhang *et al.*, “Opportunities and Challenges: Classification of Skin Disease Based on Deep Learning,” *Chinese Journal of Mechanical Engineering*, vol. 34, no. 1, Nov. 2021, doi: 10.1186/s10033-021-00629-5.

[71]
A. Katanskiy, “Skin Cancer ISIC,” *www.kaggle.com*. [Online]. Available: https://www.kaggle.com/datasets/nodoubttome/skin-cancer9-classesisic. [Accessed: Apr. 14, 2022]

[72]
M. Negm, “MLP vs CNN vs RNN Deep Learning, Machine Learning Model,” *www.linkedin.com*, Mar. 07, 2019. [Online]. Available: https://www.linkedin.com/pulse/mlp-vs-cnn-rnn-deep-learning-machine-model-momen-negm/. [Accessed: Apr. 15, 2022]

[73]
T. Gupta, “Deep Learning: Feedforward Neural Network,” *Medium*, Dec. 16, 2018. [Online]. Available: https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7. [Accessed: Apr. 15, 2022]

[74]
V. Gupta, “Understanding Feedforward Neural Networks | LearnOpenCV,” *LearnOpenCV*, Oct. 09, 2017. [Online]. Available: https://learnopencv.com/understanding-feedforward-neural-networks/. [Accessed: Apr. 15, 2022]

[75]
Dive Into Deep Learning, “4.7. Forward Propagation, Backward Propagation, and Computational Graphs — Dive into Deep Learning 0.17.5 documentation,” *d2l.ai*. [Online]. Available: https://d2l.ai/chapter_multilayer-perceptrons/backprop.html#:~:text=Forward%20propagation%20(or%20forward%20pass. [Accessed: Apr. 15, 2022]

[76]
A. Vergottis, “A simple universal workflow for machine learning models,” *InfiniCog*, Sep. 18, 2019. [Online]. Available: https://medium.com/infinicog/a-simple-universal-workflow-for-machine-learning-models-9f9bdcfdf714. [Accessed: Apr. 15, 2022]

[77]
GeeksforGeeks, “CNN | Introduction to Pooling Layer,” *GeeksforGeeks*, Aug. 05, 2019. [Online]. Available: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/. [Accessed: Apr. 19, 2022]

[78]
A. Gupta, “A Comprehensive Guide on Deep Learning Optimizers,” *Analytics Vidhya*, Oct. 07, 2021. [Online]. Available:

https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/#:~:text=An%20optimizer%20is%20a%20function. [Accessed: Apr. 19, 2022]

[79]
J. Jeong, “The Most Intuitive and Easiest Guide for CNN,” *Medium*, Jul. 17, 2019. [Online]. Available: https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480#:~:text=Flattening%20is%20converting%20the%20data. [Accessed: Apr. 19, 2022]

[80]
J. Brownlee, “Difference Between a Batch and an Epoch in a Neural Network,” *Machine Learning Mastery*, Jul. 19, 2018. [Online]. Available: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/. [Accessed: Apr. 19, 2022]

[81]
V. Gupta and A. Murzova, “Image Classification using MLP in Keras | LearnOpenCV,” *LearnOpenCV*, Oct. 23, 2017. [Online]. Available: https://learnopencv.com/image-classification-using-feedforward-neural-network-in-keras/. [Accessed: Apr. 20, 2022]

[82]
R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” *Insights into Imaging*, vol. 9, no. 4, pp. 611–629, Jun. 2018, doi: 10.1007/s13244-018-0639-9.

[83]
I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. Cambridge, Massachusetts: The Mit Press, 2016.

[84]
S. Goled, “How Do Activation Functions Introduce Non-Linearity In Neural Networks?,” *Analytics India Magazine*, Nov. 30, 2021. [Online]. Available: https://analyticsindiamag.com/how-do-activation-functions-introduce-non-linearity-in-neural-networks/#:~:text=The%20significance%20of%20the%20activation. [Accessed: Apr. 22, 2022]

[85]
E. Amor, “Understanding Non-Linear Activation Functions in Neural Networks,” *ML Cheat Sheet*, May 29, 2020. [Online]. Available: https://medium.com/ml-cheat-sheet/understanding-non-linear-activation-functions-in-neural-networks-152f5e101eeb#:~:text=What%20does%20non%2Dlinearity%20mean. [Accessed: Apr. 22, 2022]

[86]
J. Brownlee, “A Gentle Introduction to the Rectified Linear Unit (ReLU),” *Machine Learning Mastery*, Jan. 08, 2019. [Online]. Available: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/#:~:text=vanishing%20gradient%20problem.-. [Accessed: Apr. 22, 2022]

[87]
N. Davis, “Does ReLU work with negative values? – QuickAdviser,” *quick-adviser.com*, Mar. 18, 2019. [Online]. Available: https://quick-adviser.com/does-relu-work-with-negative-values/#Does_ReLU_work_with_negative_values. [Accessed: Apr. 22, 2022]

[88]
J. Brownlee, “Softmax Activation Function with Python,” *Machine Learning Mastery*, Oct. 18, 2020. [Online]. Available: https://machinelearningmastery.com/softmax-activation-function-with-python/. [Accessed: Apr. 22, 2022]

[89]
S. Zou, W. Chen, and H. Chen, “Image Classification Model Based on Deep Learning in Internet of Things,” *Wireless Communications and Mobile Computing*, vol. 2020, p. e6677907, Dec. 2020, doi: 10.1155/2020/6677907. [Online]. Available: https://www.hindawi.com/journals/wcmc/2020/6677907/. [Accessed: Apr. 22, 2022]

[90]
K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, “What is the best multi-stage architecture for object recognition?,” *nyuscholars.nyu.edu*, 2009. [Online]. Available: https://nyuscholars.nyu.edu/en/publications/what-is-the-best-multi-stage-architecture-for-object-recognition. [Accessed: Feb. 22, 2022]

[91]
P. Ganesh, “Types of Convolution Kernels : Simplified,” *Medium*, Oct. 18, 2019. [Online]. Available: https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37. [Accessed: Apr. 23, 2022]

[92]
K. C. Santosh, N. Das, and S. Ghosh, *Deep learning models for medical imaging*. London San Diego Cambridge Oxford Academic Press, Elsevier, 2022.

[93]
A. Thakur, “Intuitive understanding of 1D, 2D, and 3D convolutions in convolutional neural networks.,” *W&B*, Aug. 10, 2020. [Online]. Available:

https://wandb.ai/ayush-thakur/dl-question-bank/reports/Intuitive-understanding-of-1D-2D-and-3D-convolutions-in-convolutional-neural-networks---VmlldzoxOTk2MDA. [Accessed: Apr. 23, 2022]

[94]
M. Basavarajaiah, "Which pooling method is better? Maxpooling vs minpooling vs average pooling," *Medium*, Aug. 22, 2019. [Online]. Available: https://medium.com/@bdhuma/which-pooling-method-is-better-maxpooling-vs-minpooling-vs-average-pooling-95fb03f45a9#:~:text=Average%20pooling%20method%20smooths%20out. [Accessed: Apr. 23, 2022]

[95]
Arc, "Convolutional Neural Network," *Medium*, Dec. 25, 2018. [Online]. Available: https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05. [Accessed: Apr. 23, 2022]

[96]
K. E. Koech, "Cross-Entropy Loss Function," *Medium*, Nov. 25, 2021. [Online]. Available: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e#:~:text=Categorical%20cross%2Dentropy%20is%20used. [Accessed: Apr. 24, 2022]

[97]
S. Han *et al.*, "Optimizing Filter Size in Convolutional Neural Networks for Facial Action Unit Recognition," *openaccess.thecvf.com*, 2018. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2018/html/Han_Optimizing_Filter_Size_CVPR_2018_paper.html. [Accessed: Apr. 25, 2022]

[98]
Jason Brownlee, "How Do Convolutional Layers Work in Deep Learning Neural Networks?," *Machine Learning Mastery*, Apr. 16, 2019. [Online]. Available: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/. [Accessed: Apr. 26, 2022]

[99]
N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014 [Online]. Available: https://jmlr.org/papers/v15/srivastava14a.html. [Accessed: Apr. 27, 2022]
[100]

https://www.facebook.com/jason.brownlee.39, "A Gentle Introduction to Dropout for Regularizing Deep Neural Networks," *Machine Learning Mastery*, Dec. 02, 2018. [Online]. Available:

https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/. [Accessed: Apr. 27, 2022]

[101]
J. Brownlee, “Code Adam Optimization Algorithm From Scratch,” *Machine Learning Mastery*, Jan. 12, 2021. [Online]. Available: https://machinelearningmastery.com/adam-optimization-from-scratch/. [Accessed: Apr. 27, 2022]

[102]
J. Brownlee, “Understand the Impact of Learning Rate on Neural Network Performance,” *Machine Learning Mastery*, Jan. 24, 2019. [Online]. Available: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/#:~:text=The%20learning%20rate%20controls%20how. [Accessed: Apr. 27, 2022]

[103]
deeplizard, “Mapping Keras labels to image classes,” *deeplizard.com*, Dec. 21, 2017. [Online]. Available: https://deeplizard.com/learn/video/pZoy_j3YsQg. [Accessed: Apr. 28, 2022]

[104]
M. J. Anzanello and F. S. Fogliatto, “Learning curve models and applications: Literature review and research directions,” *International Journal of Industrial Ergonomics*, vol. 41, no. 5, pp. 573–583, Sep. 2011, doi: 10.1016/j.ergon.2011.05.001. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S016981411100062X. [Accessed: Apr. 29, 2022]

[105]
S. H. S. Basha, S. R. Dubey, V. Pulabaigari, and S. Mukherjee, “Impact of fully connected layers on performance of convolutional neural networks for image classification,” *Neurocomputing*, Oct. 2019, doi: 10.1016/j.neucom.2019.10.008.

[106]
R. Han, “What is weight regularization in neural networks -,” *DeZyre*, Aug. 02, 2021. [Online]. Available: https://www.projectpro.io/recipes/what-is-weight-regularization-neural-networks. [Accessed: Apr. 30, 2022]

[107]
J. McCaffey, “Test Run - L1 and L2 Regularization for Machine Learning,” *docs.microsoft.com*, Jun. 17, 2015. [Online]. Available: https://docs.microsoft.com/en-us/archive/msdn-magazine/2015/february/test-run-l1-and-l2-re

gularization-for-machine-learning#:~:text=L1%20regularization%20uses%20the%20sum. [Accessed: Apr. 30, 2022]

[108]
Li Deng, “The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web],” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, Nov. 2012, doi: 10.1109/msp.2012.2211477.

[109]
Z. LT, “Essential Things You Need to Know About F1-Score,” *Medium*, Feb. 25, 2022. [Online]. Available: https://towardsdatascience.com/essential-things-you-need-to-know-about-f1-score-dbd973bf1a3#:~:text=F1%2Dscore%20is%20one%20of. [Accessed: May 01, 2022]

[110]
A. Halevy, P. Norvig, and F. Pereira, “The Unreasonable Effectiveness of Data,” *IEEE Intelligent Systems*, vol. 24, no. 2, pp. 8–12, Mar. 2009, doi: 10.1109/mis.2009.36.

[111]
J. Hanlon, “Why is so much memory needed for deep neural networks?,” *Graphcore.ai*, Jan. 31, 2017. [Online]. Available: https://www.graphcore.ai/posts/why-is-so-much-memory-needed-for-deep-neural-networks. [Accessed: May 02, 2022]

[112]
S. Saxena, “Softmax | What is Softmax Activation Function | Introduction to Softmax,” *Analytics Vidhya*, Apr. 05, 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/04/introduction-to-softmax-for-neural-network/. [Accessed: May 02, 2022]

[113]
P.-L. Biaggi, “AI and data science tool up to battle COVID-19,” *Orange Business Services*, Apr. 07, 2020. [Online]. Available: https://www.orange-business.com/en/blogs/ai-and-data-science-tool-battle-covid-19. [Accessed: May 02, 2022]

[114]
Q. Ji, “Research on Recognition Effect of DSCN Network Structure in Hand-Drawn Sketch,” *Computational Intelligence and Neuroscience*, vol. 2021, p. e4056454, Nov. 2021, doi: 10.1155/2021/4056454. [Online]. Available: https://www.hindawi.com/journals/cin/2021/4056454/. [Accessed: May 02, 2022]

[115]

S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, “Understanding Data Augmentation for Classification: When to Warp?,” *IEEE Xplore*, Nov. 01, 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7797091. [Accessed: May 04, 2022]

[116]
M. Yawar, “What is AdaGrad?,” *www.codingninjas.com*, Mar. 01, 2022. [Online]. Available: https://www.codingninjas.com/codestudio/library/what-is-adagrad. [Accessed: May 04, 2022]

[117]
M. J. Kochnderfer and T. A. Wheeler, *Algorithms for optimization*. Cambridge, Massachusetts Etc.: The Mit Press, 2019.

[118]
GeeksforGeeks, “Intuition of Adam Optimizer,” *GeeksforGeeks*, Oct. 22, 2020. [Online]. Available: https://www.geeksforgeeks.org/intuition-of-adam-optimizer/#:~:text=Adam%20optimizer%20involves%20a%20combination. [Accessed: May 04, 2022]

[119]
H. Liu, “Chapter 7 - Rail transit channel robot systems,” *ScienceDirect*, Jan. 01, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128229682000073. [Accessed: May 05, 2022]

[120]
G. Strang, E. Herman, and Openstax College, *Calculus*. Houston, Texas: Openstax, Rice University, 2016.

[121]
J. Brownlee, “A Gentle Introduction to Learning Curves for Diagnosing Machine Learning Model Performance,” *Machine Learning Mastery*, Apr. 03, 2019. [Online]. Available: https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

[122]
“Classical ML Equations in LaTeX,” *blmoistawinde.github.io*. [Online]. Available: https://blmoistawinde.github.io/ml_equations_latex/#softmax. [Accessed: May 05, 2022]

[123]
S. Pandey, “How to choose the size of the convolution filter or Kernel size for CNN?,” *Analytics Vidhya*, Jul. 01, 2020. [Online]. Available:

https://medium.com/analytics-vidhya/how-to-choose-the-size-of-the-convolution-filter-or-kernel-size-for-cnn-86a55a1e2d15#:~:text=Smaller%20kernel%20sizes%20consists%20of. [Accessed: May 05, 2022]

[124]
C. Hansen, “Neural Networks: Feedforward and Backpropagation Explained,” *Machine Learning From Scratch*, Aug. 05, 2019. [Online]. Available: https://mlfromscratch.com/neural-networks-explained/#from-input-layer-to-hidden-layer. [Accessed: May 06, 2022]

[125]
“How Good Is Your Machine Learning Algorithm?,” *MyDataModels*, Oct. 21, 2020. [Online]. Available: https://www.mydatamodels.com/learn/how-good-is-your-machine-learning-algorithm/. [Accessed: May 06, 2022]