Central Neural Network

Oussoubi Niakate – 20242905

Ameen Popoola – 20231717

[Colab link](https://colab.research.google.com/drive/1DM_VS9IXrqdU1uA1I8uuZO8TWF9OsnVQ#scrollTo=ZU1Niwvti7Pz)

Table of Contents

[1) The Data Set 1](#_ohevqoyt0r5i)

[a)Visualisation of some of the key attributes 2](#_fmihl87qvdme)

[b) Normalising the Data 2](#_g6bj8cnpkreb)

[2) Network Structure and Hyperparameters 3](#_wfjw5opstw8r)

[a) Network Architecture Diagram 3](#_vbysv8w17llf)

[b) Features Explained 4](#_8ccv8ai006v)

[3) The Loss Function 6](#_lyqbjq6xnp0o)

[4) The Optimiser 7](#_k1ju3hz2tqr7)

[5) Cross Validation 7](#_30eyjfpoigl4)

[6) Results 9](#_o4fi1vrmldx9)

[7) Result Evaluation 11](#_j9pg3m207fon)

[8) Impact of Varying Hyperparameters 13](#_iuow6pn8hugi)

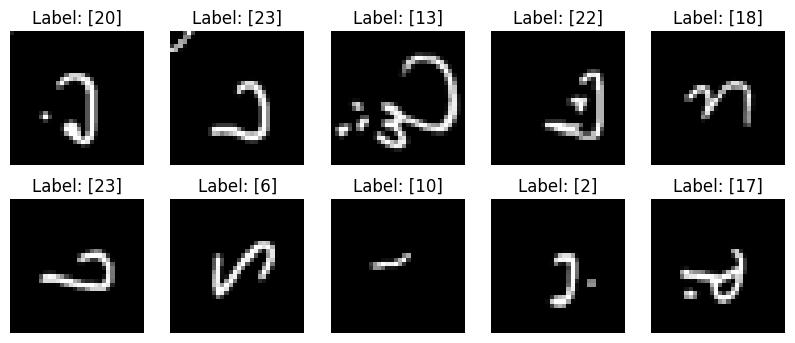
# 

## 1) The Data Set

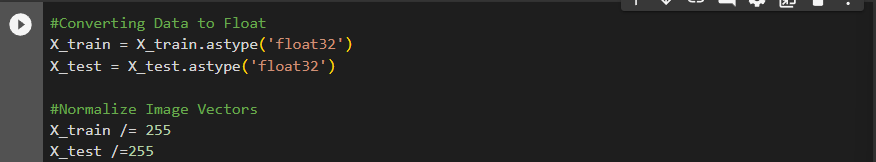
We chose to use a dataset containing Arabic Handwritten Characters that we found on [kaggle](https://www.kaggle.com/datasets/mloey1/ahcd1), which was previously used by A. El-Sawy, M. Loey, and H. EL-Bakry for three of their papers, and one of them being *“Deep Learning Autoencoder Approach for Handwritten Arabic Digits Recognition”* Which was published in 2017. In this dataset we were given 4 CSV files which contained training and test labels and images.This dataset contains images of 28 different Hand Written Arabic Letters.

# a)Visualisation of some of the key attributes

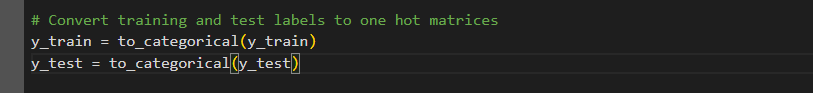
To view some of the images within the dataset, we ran a loop which displayed 10 random images from it in a 32x32 shape.



# b) Normalising the Data

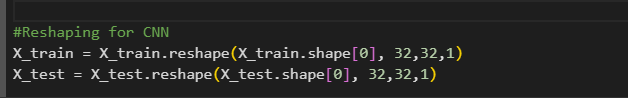
****

* For normalising the data we first changed the datatype of the X\_train,X\_test values to float32. This is to ensure that once we normalised the Image Vectors by dividing by 255, if there were floating values present within the vector, there would not be any issues.



* We then converted our y-train,y-test labels to one hot matrices using the *to\_categorical*

Function. This converts the categoricals labels into a binary matrix representation to be used when training the model

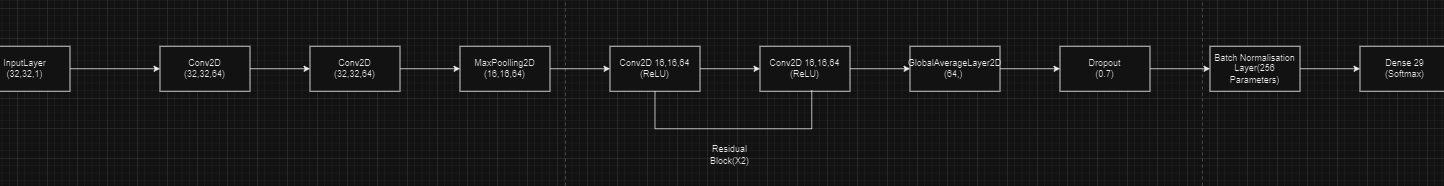


* We then reshaped our X\_train,X\_test labels to contain the number of samples, 32x32 pixel images as this is the minimum image sizes required for a ResNet model and a dimension for 1 for our channel as the images are greyscale images.

## 2) Network Structure and Hyperparameters

# a) Network Architecture Diagram

In our network architecture, we created a 8 layer ResNet model. Below you will see a diagram of the architecture. The X2 beside the Residual Block indicates the Block occurs twice.There is also an Add Function in this.

****

Below is the summary of the model.

Model: "resnet8"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # Connected to

==================================================================================================

input\_1 (InputLayer) [(None, 32, 32, 1)] 0 []

conv2d (Conv2D) (None, 32, 32, 64) 640 ['input\_1[0][0]']

conv2d\_1 (Conv2D) (None, 32, 32, 64) 36928 ['conv2d[0][0]']

max\_pooling2d (MaxPooling2 (None, 16, 16, 64) 0 ['conv2d\_1[0][0]']

D)

conv2d\_2 (Conv2D) (None, 16, 16, 64) 36928 ['max\_pooling2d[0][0]']

conv2d\_3 (Conv2D) (None, 16, 16, 64) 36928 ['conv2d\_2[0][0]']

add (Add) (None, 16, 16, 64) 0 ['conv2d\_3[0][0]',

'max\_pooling2d[0][0]']

activation (Activation) (None, 16, 16, 64) 0 ['add[0][0]']

conv2d\_4 (Conv2D) (None, 16, 16, 64) 36928 ['activation[0][0]']

conv2d\_5 (Conv2D) (None, 16, 16, 64) 36928 ['conv2d\_4[0][0]']

add\_1 (Add) (None, 16, 16, 64) 0 ['conv2d\_5[0][0]',

'activation[0][0]']

activation\_1 (Activation) (None, 16, 16, 64) 0 ['add\_1[0][0]']

global\_average\_pooling2d ( (None, 64) 0 ['activation\_1[0][0]']

GlobalAveragePooling2D)

dropout (Dropout) (None, 64) 0 ['global\_average\_pooling2d[0][

0]']

batch\_normalization (Batch (None, 64) 256 ['dropout[0][0]']

Normalization)

dense (Dense) (None, 29) 1885 ['batch\_normalization[0][0]']

==================================================================================================

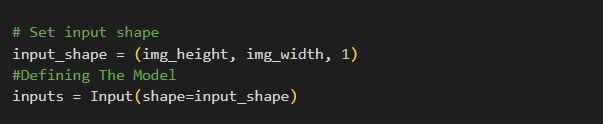
Total params: 187421 (732.11 KB)

Trainable params: 187293 (731.61 KB)

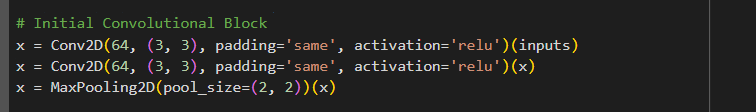
Non-trainable params: 128 (512.00 Byte)

# b) Features Explained

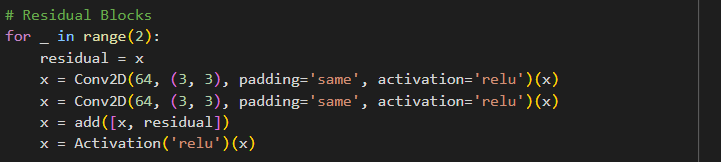
The Convoluted Neural Network(CNN) we chose to base our model on was ResNet. We used an 8 layer ResNet model.



* This consisted of an ***input layer*** which expects a shape of 32x32 pixel images and a single channel for the greyscale image.



* The model has ***one convolutional block*** with 2 two dimensional convolutions with 64 filters and kernels of size 3x3. This layer uses a ReLU activation function, a function commonly used in Neural Networks. It then has a Max Pooling layer with size 2x2 used for processing the image data.



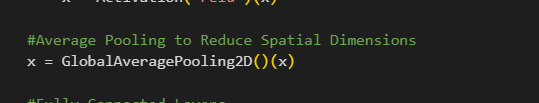
* In this model , we have two residual blocks.The **residual block** is the defining element of the ResNet architecture.

*Residual Learning allows for the training of very deep networks and allows for easy training of low-level and high-level features, along with allowing them to be substantially deeper, enabling improved performance for both visual and non-visual tasks.ResNet does this by introducing the concept of a residual block. The residual block uses the concept of shortcut connections and skip connections. Shortcut Connections are designed to combat the issue of vanishing gradients by providing a direct path for information to flow through the network without passing through the intermediate layers.*

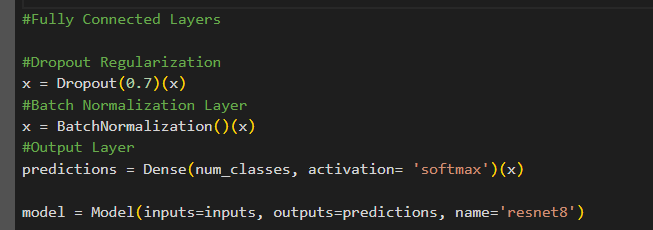
*Residual Learning has allowed ResNet to become one of the better CNN’s when it comes to Object Recognition, Image Classification and Segmentation. It continues to have a great impact on the design of deep learning architecture and continues to influence advancements in the field.*

In our 8 layer model based upon ResNet, we use a *loop* to create the two blocks. The residual value is then created when we use the *x value* as the residual to be added. We use two convolution layers similar to the previously initialised convolution block, with 64 filters and a kernel of size 3x3. These layers are then responsible for the residual mapping(*the difference between the input x and the output of the function H(x*).

We then implement the shortcut connection by using *x = add([x, residual]),* which adds the output of the convolutional layerto the residual value*(x).* We then applied a ReLu activation is applied to the sum of the convolutional output and the residual.



* ***GlobalAveragePooling2D*** layer reduces spatial dimensions to 1x1 by taking the average over spatial dimensions.



* In the fully connected layers, we established a dropout layer with a dropout rate 0.7.We then added the ***Batch Normalisation Layer***.
* We added a ***Dense Layer*** for the Output Layer which specifies the number of classes and uses the softmax activation function, which squashes the output values into a probability distribution.

## 3) The Loss Function

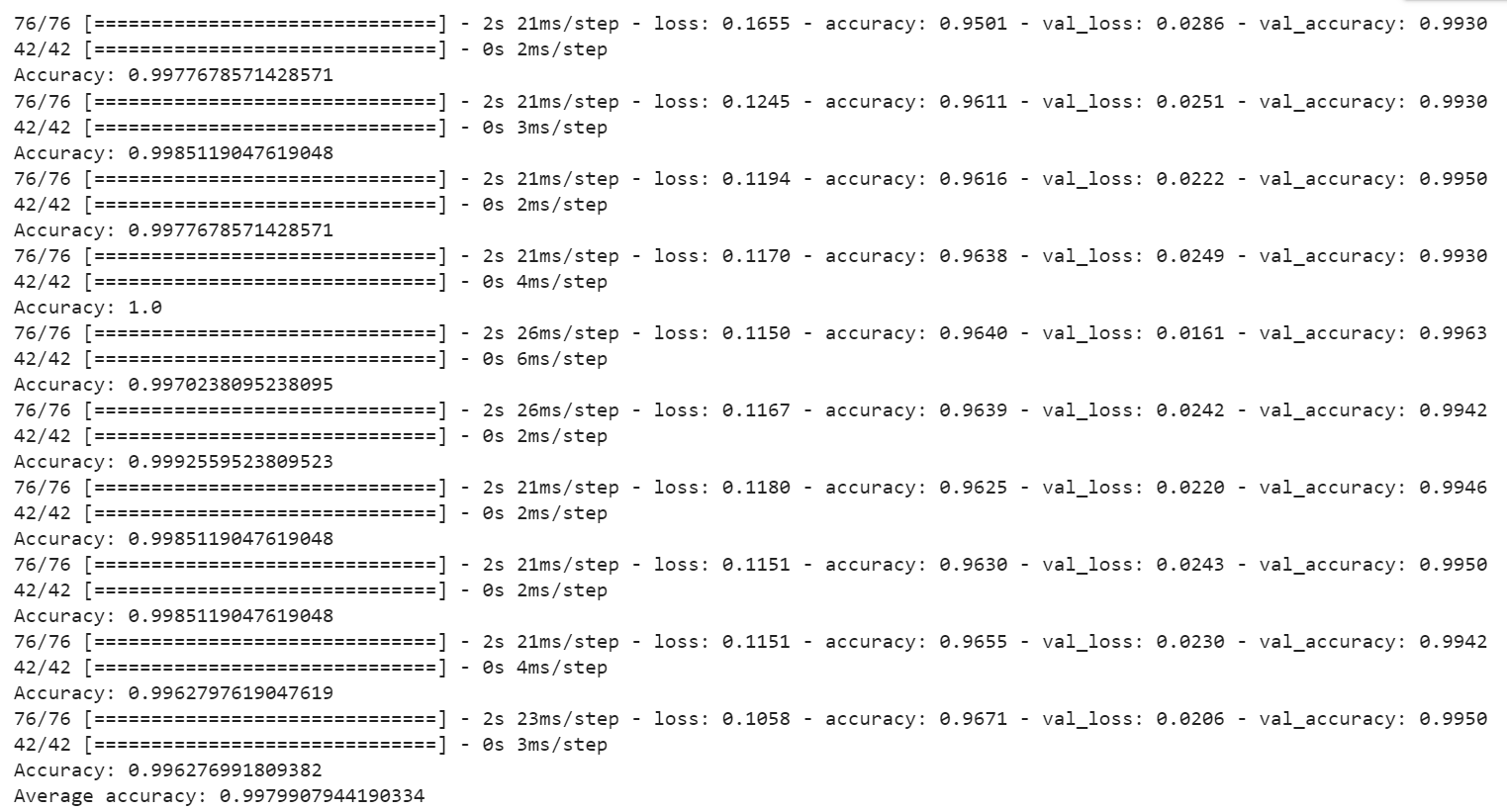
* We used ***binary crossentropy*** to calculate the loss between the true labels and the predicted probabilities.

## 4) The Optimiser

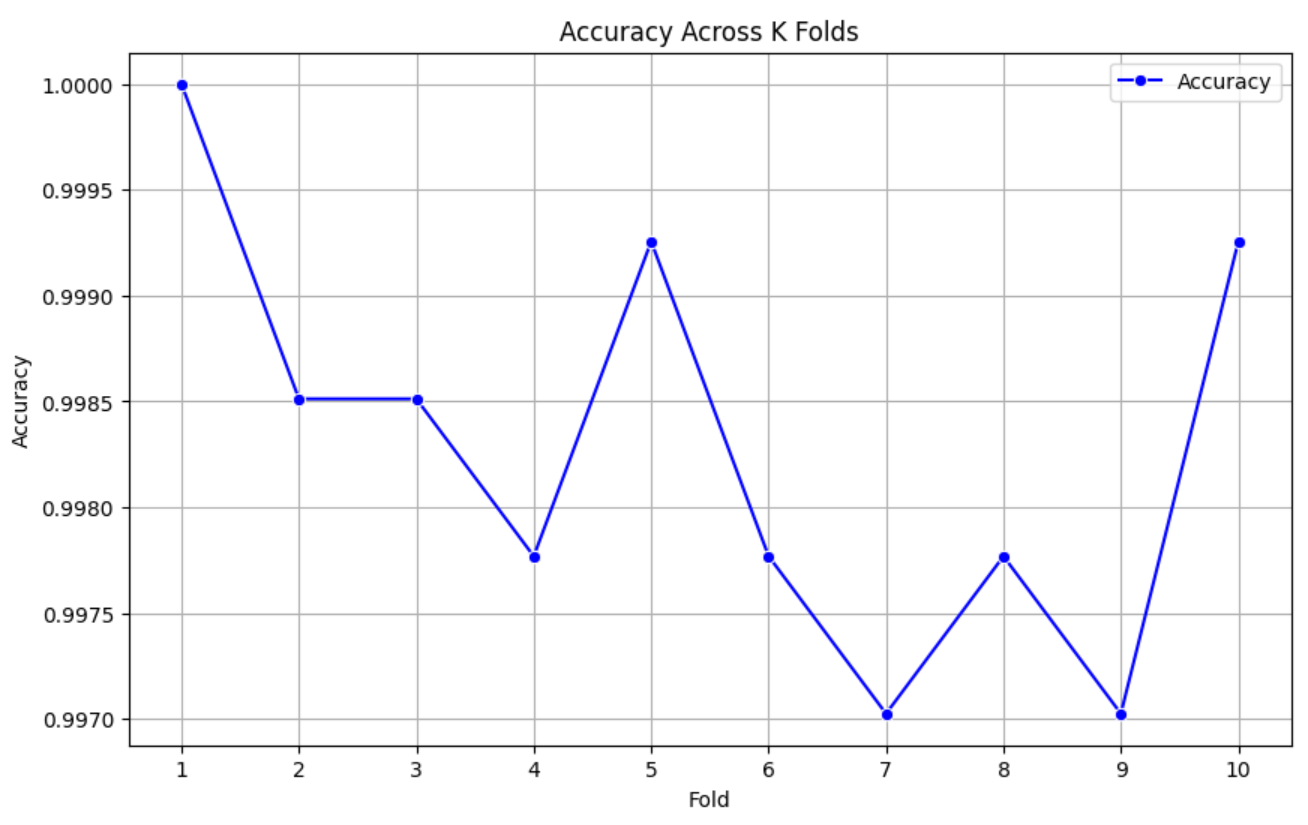
* When compiling the model we used the ***Adam Optimizer*** function.This is an extension of the stochastic gradient descent (SGD) algorithm.

## 5) Cross Validation

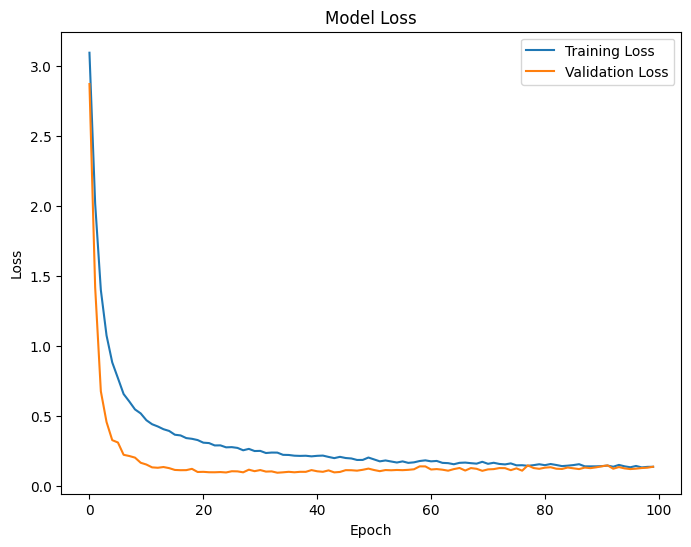
We decided to use ***K-Fold Cross Validation*** technique to evaluate the performance of our dataset.The dataset is divided into k subsets or folds. The model is trained and evaluated 10 times, using a different fold as the validation set each time.

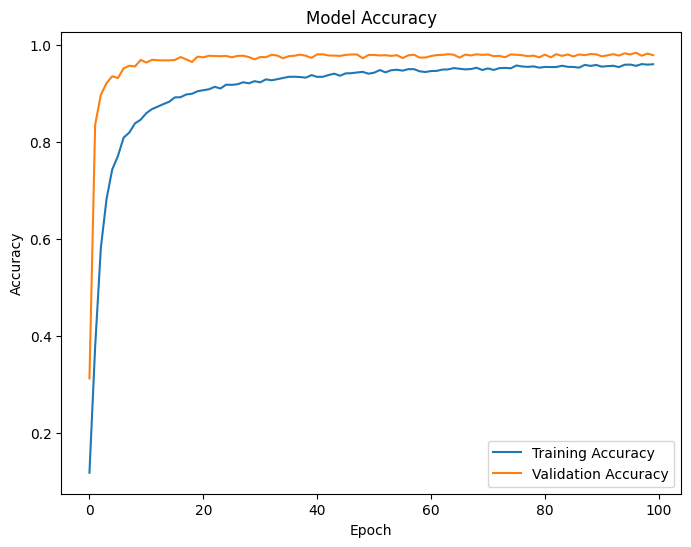


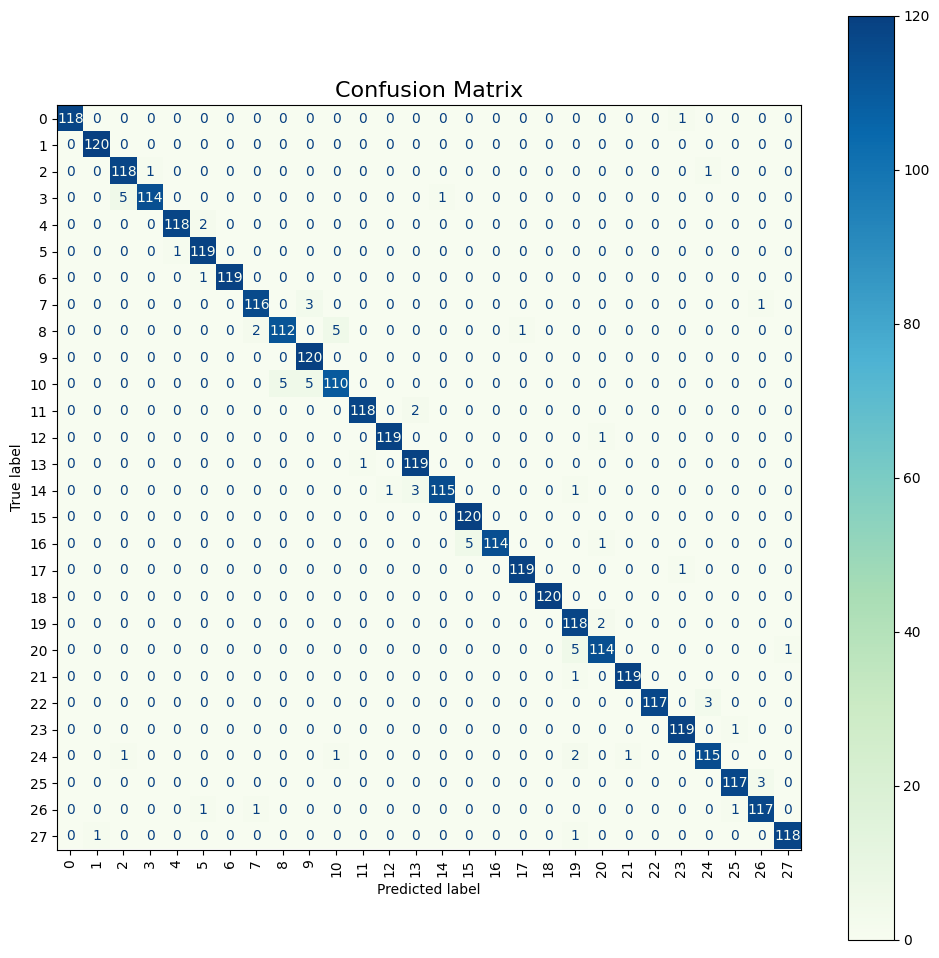
In this image we can see that both the training and validation sets have a very high accuracy of above 0.99, making it robust across all 10 folds. Here is a graph below representing the accuracy over K folds:



## 6) Results







## 7) Result Evaluation

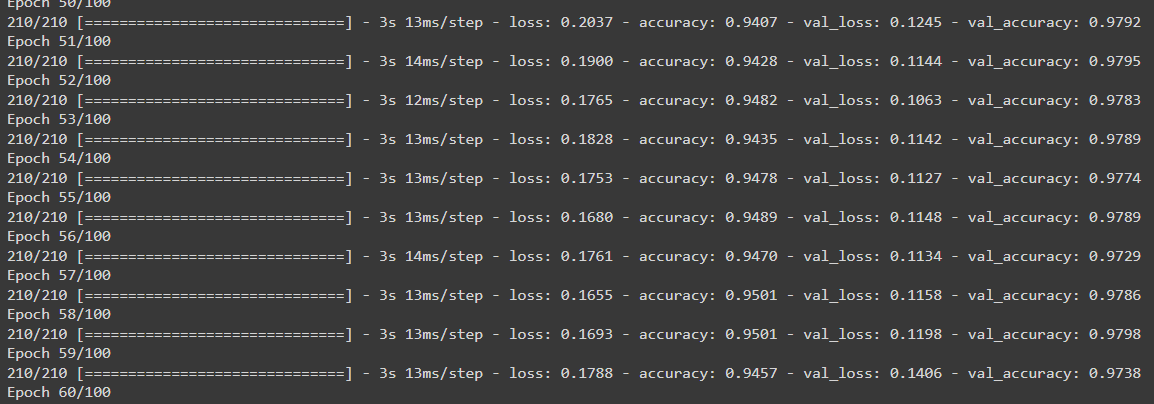
* HYPERPARAMETERS - EPOCHS = 100

BATCH\_SIZE = 64

LOSS FUNCTION = CATEGORICAL CROSSENTROPY

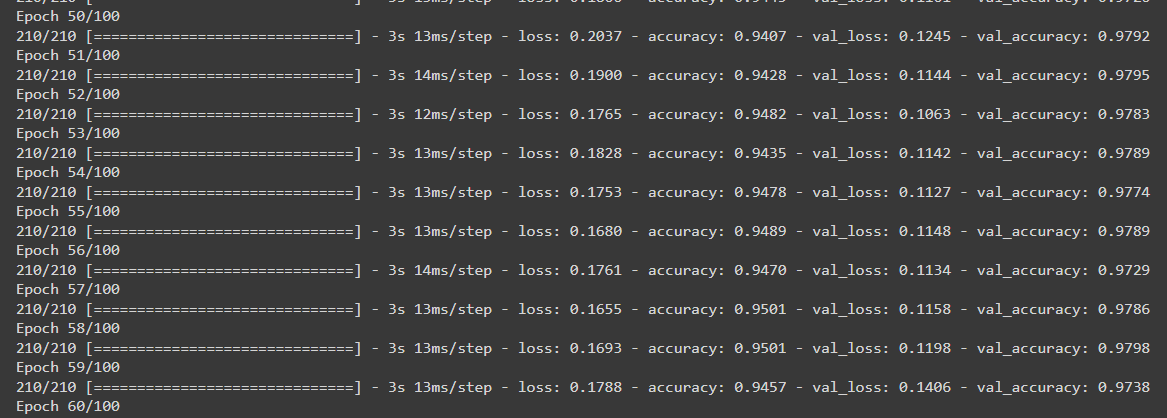
**Loss**

In the first graph we can see the model loss graph, the model loss at the beginning of the model is extremely high but as the model runs the loss reduces and reaches a consistent loss rate as we can see in the image below. After the 50th Epoch, the loss is consistently under .2. The decreasing training loss suggests that the model is optimizing weights to minimize the error between predicted labels and actual labels



**Accuracy**

In the plotted graph for the model accuracy, we can see that the accuracy of the model is initially low but as the model continues to run the accuracy reaches over .9 accuracy as shown in the image below, from the 50th epoch the accuracy is consistently over .9. This indicates that the model is learning from the training data and its improving its ability to correctly classify images.



**Validation Accuracy and Loss**

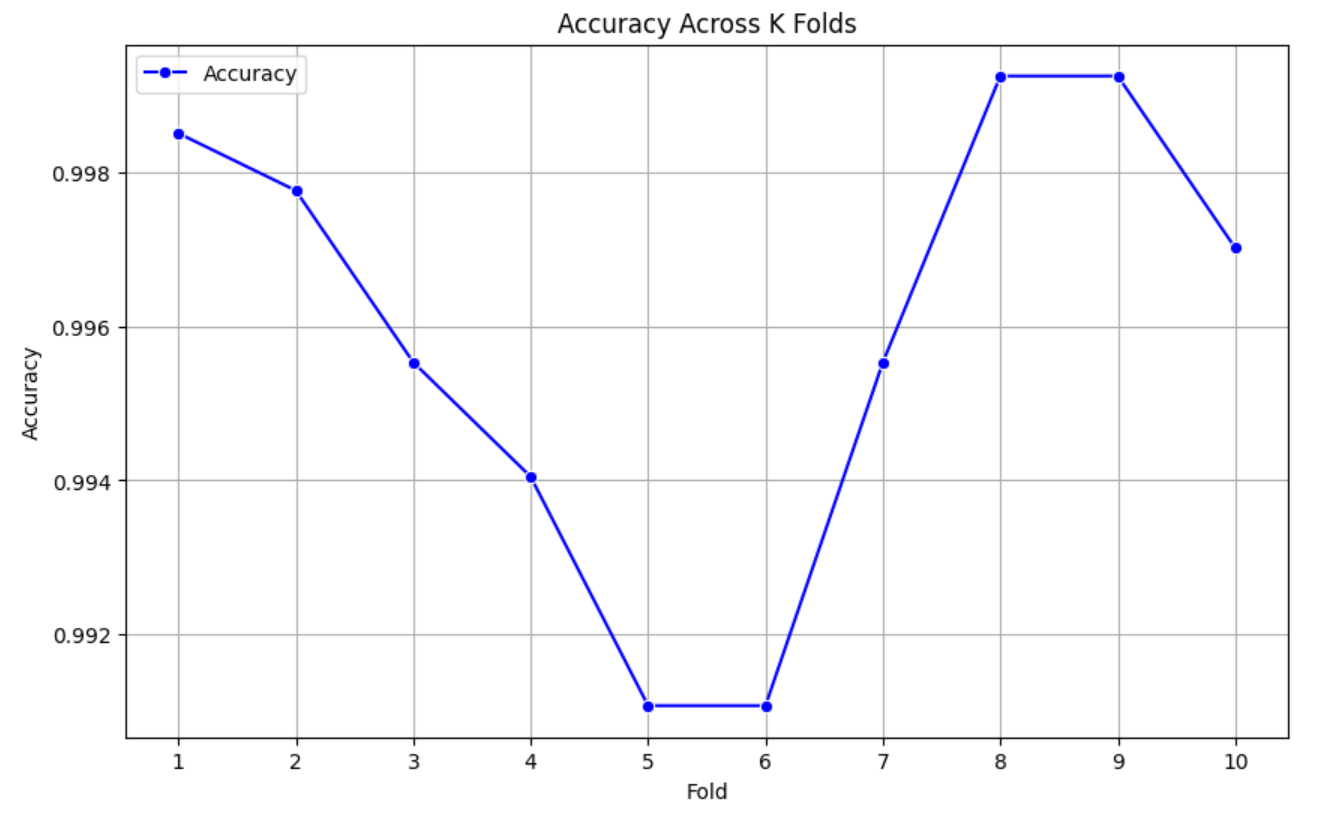
The increasing validation accuracy and decreasing validation loss show that the data generalizes well to unseen data. The similar values between training accuracy and validation accuracy show that the dataset is not subject to overfitting or underfitting



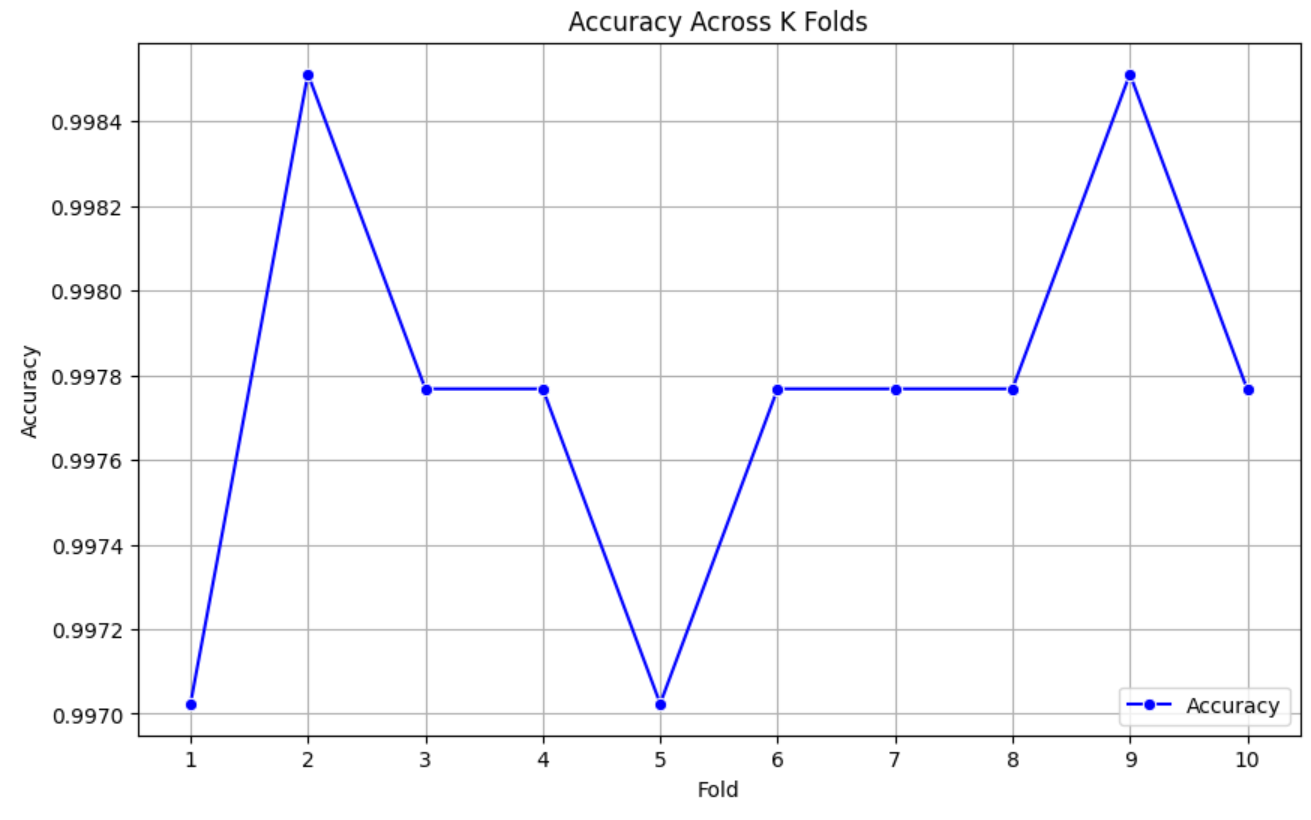
## 8) Impact of Varying Hyperparameters

***Cross Validation***

When we reduced the batch size from 128 to 64, we discovered that the average accuracy reduced from 0.9988 to 0.9959, and the training epochs consisted of 152 steps.



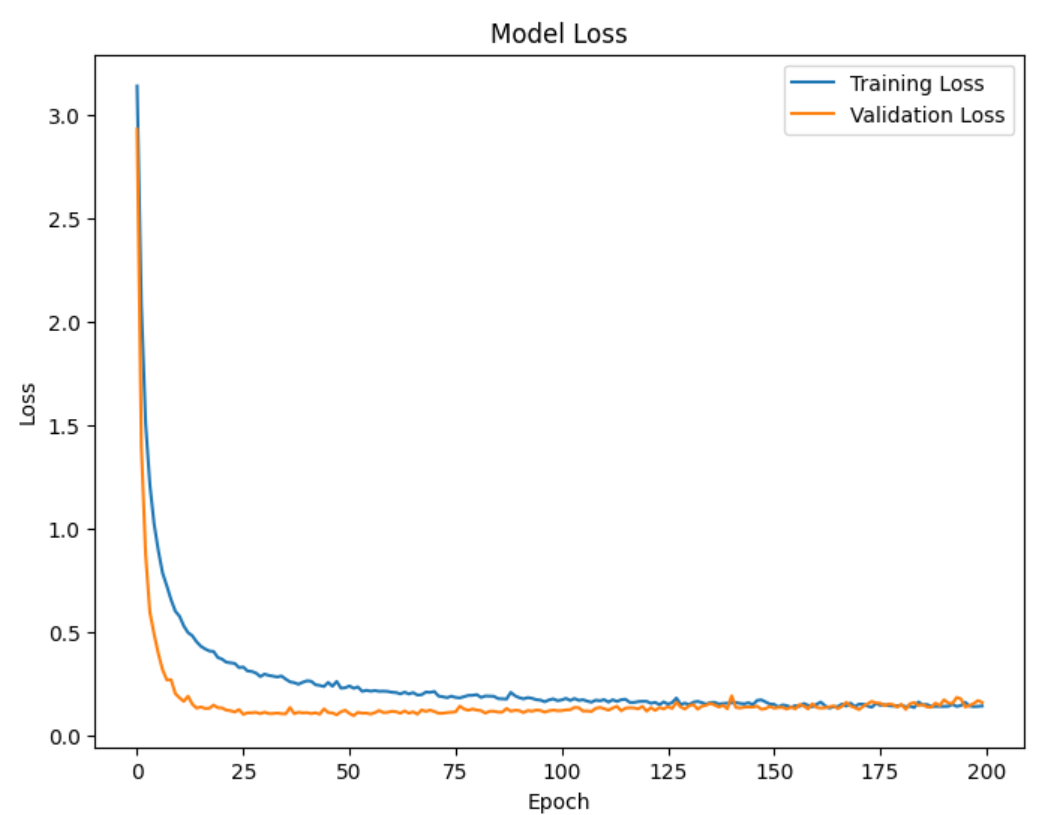
Similarly, when we doubled the batch size from 128 to 256, the accuracy reduced to 0.9977 and the training epochs consisted of 38 steps.

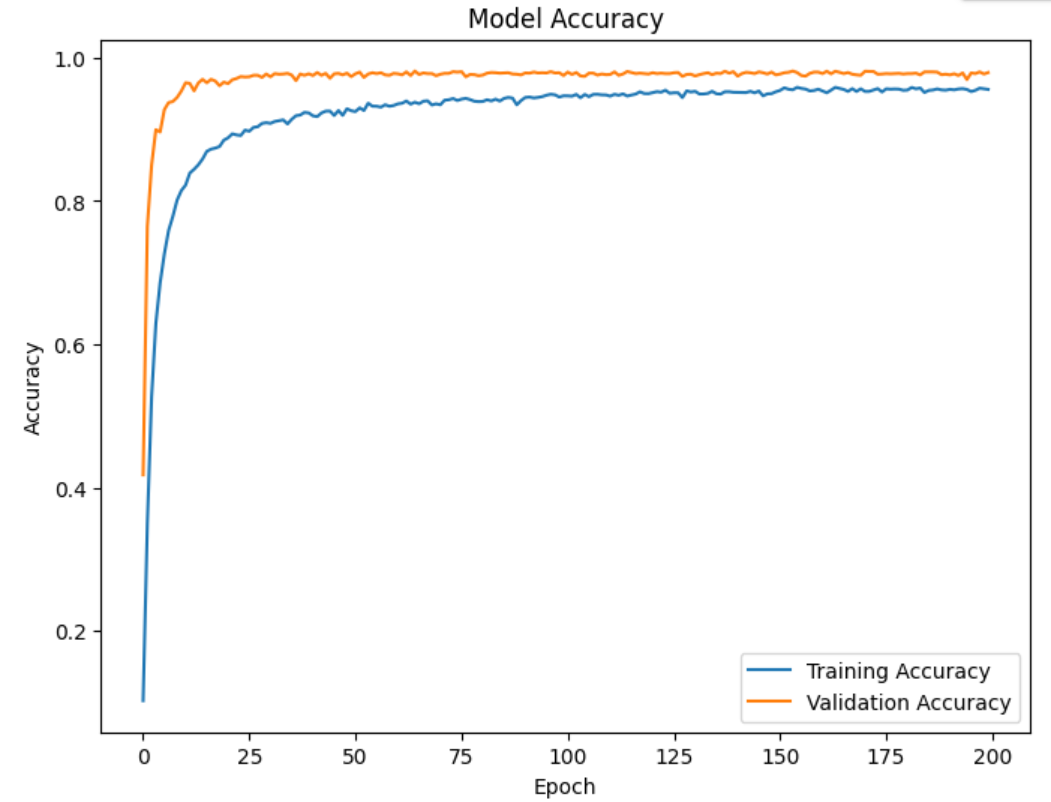


***Model***

***Changing the Amount of Epochs***

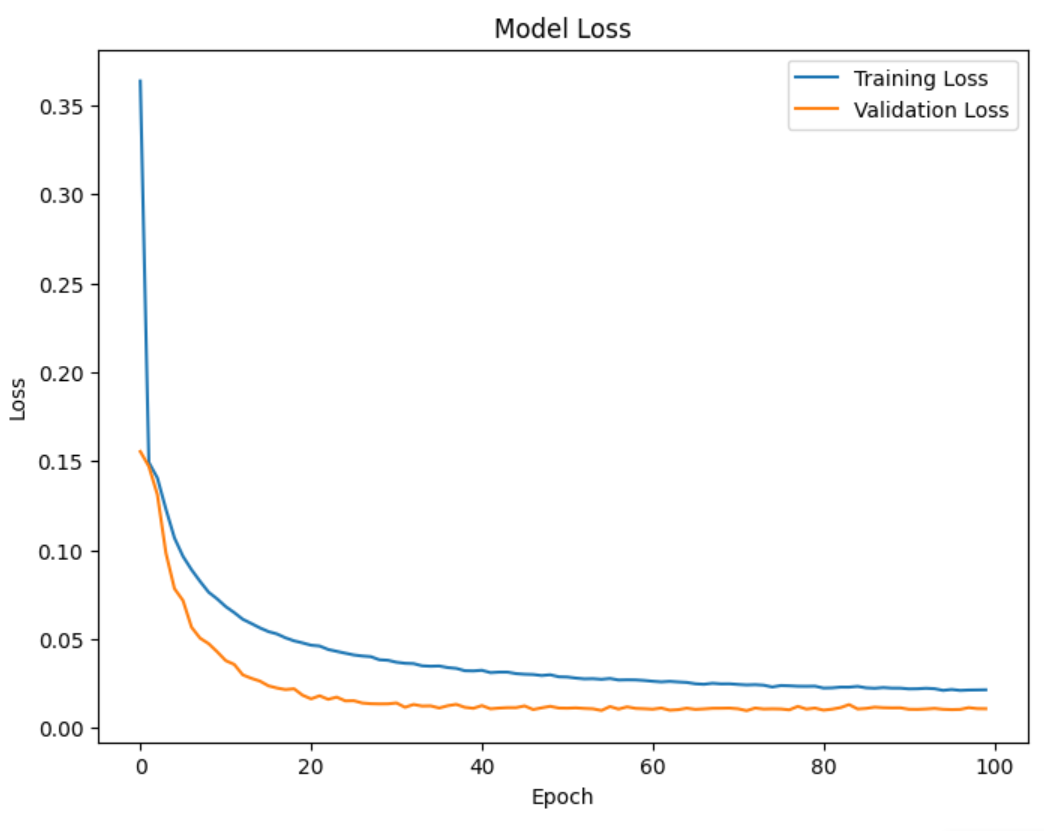
When doubling the amount of epochs from 100 to 200 there was no significant difference between the training & validation loss and accuracy values as shown in the graphs below.

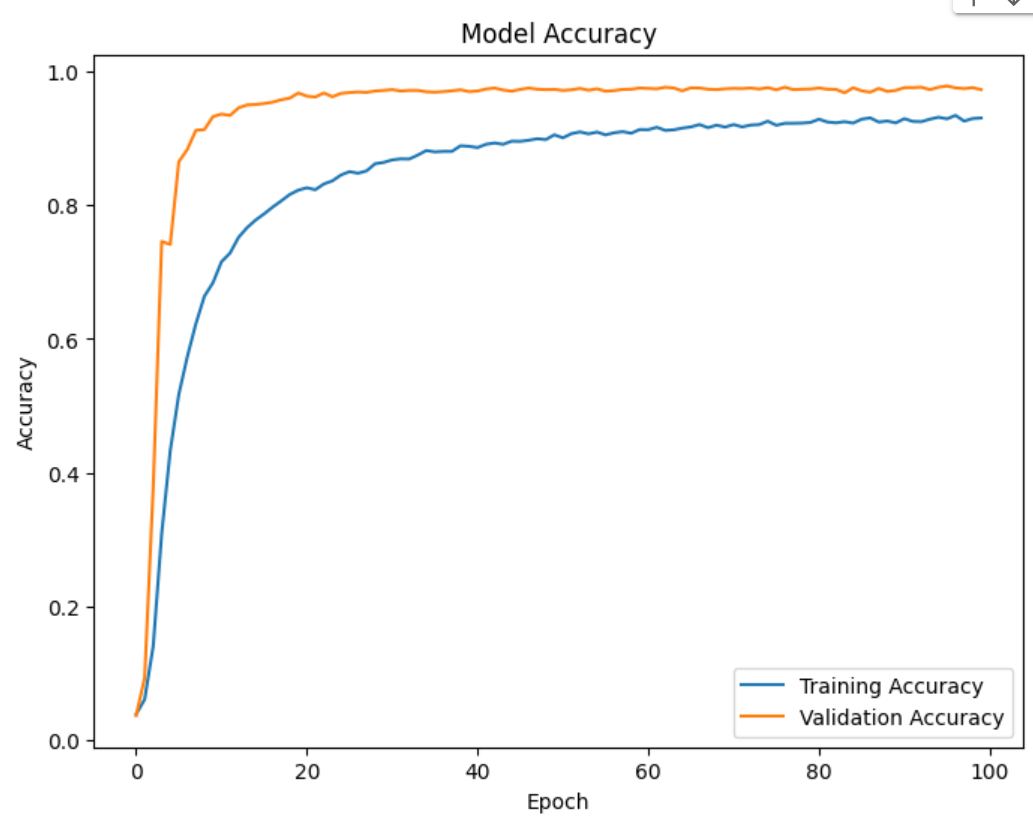




***Changing the loss function***

When changing the classification from “categorical\_crossentropy” to “binary\_crossentropy”, the difference was noticeable, especially in the training and validation loss values.





**References**

**1.Understand Deep Residual Networks** <https://medium.com/@waya.ai/deep-residual-learning-9610bb62c355#:~:text=The%20residual%20learning%20framework%20eases,number%20of%20parameters%20(weights).>

2.**Understanding and Coding a ResNet in Keras**

[**https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33**](https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33)