Experimental Results

Fasih Abdullah i200432 CS-B

October 2023

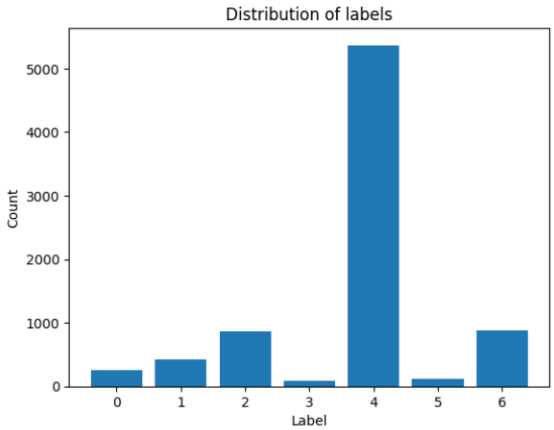
1 Introduction

This document shall cover the experimental results for each of the three tasks that have been implemented for Deep Learning Assignment 3. It shall be divided into three sections, one per task, and shall summarize the steps required in each task. The summary shall include details about the dataset, pre-processing steps, model architecture, hyper-parameter tuning, training performance, testing performance, and finally conclusions on the final results,

2 Task 1: Skin Disease Classification using CNN

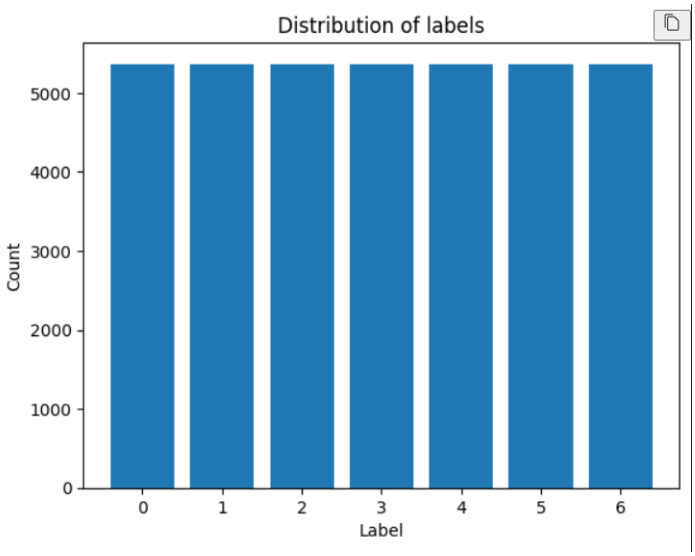
In this task, we are provided with a dataset from Kaggle which contains 8012 RGB images sized 28x28. We were required to load the images, perform classification, and predict among the six labels from the dataset.

Upon loading all of the images and plotting the class distribution it is apparent that the classes are imbalanced as shown in the following graph:



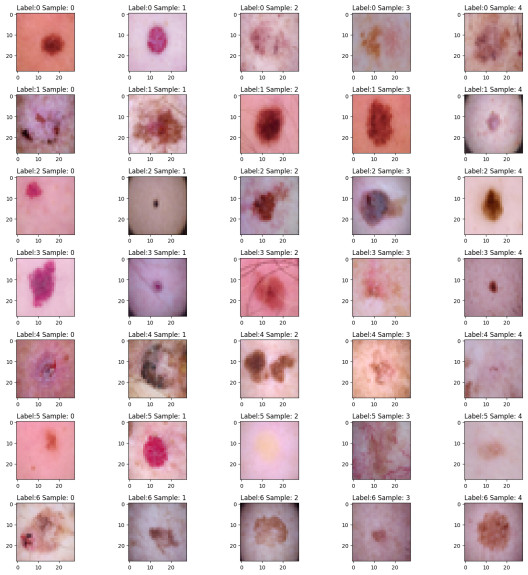
Images for label 4 dominate the dataset becoming the majority class and label 3 has the least images in the dataset becoming the minority class. Thus

the solution was to perform class balancing. In this case, the classes were balanced using the Random Over Sampler provided in the imblearn library. This resulted in balancing the images by resampling to 5367 images per class. This resulted in the following balanced class distribution:



Now that the classes have been balanced, the data is reshaped to (-1,28,28,3) i.e (count,width,height,channel) as that is required in the input layer of the CNN.

The next step was to print random samples per class so 5 random images were displayed per class in the following graph:



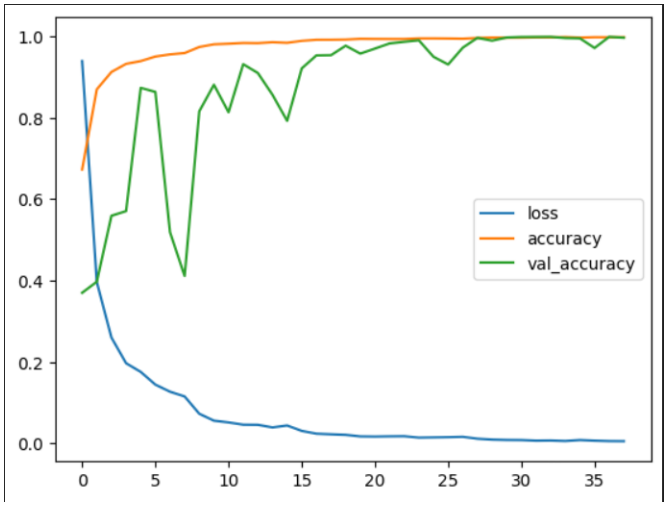
Now the CNN architecture is set as follows:

- 32 Conv2D 2x2 Kernels
- 64 Conv2D 3x3 Kernels
- Batch Normalization
- MaxPool2D 2x2
- 32 Conv2D 1x1 Kernels
- Batch Normalization
- MaxPool2D 2x2
- Flatten
- 256 Neuron Dense Layer with ReLu
- Dropout Layer
- Batch Normalization
- 32 Neuron Dense Layer with ReLu
- Dropout Layer
- Final 7 Neuron Dense Layer with Softmax

The optimizer used is the Adam Optimizer with sparse_categorical_crossentropy set as the loss function. The following callbacks have been set:

- Early Stopping
- Reduce Learning Rate on Plateau
- A custom callback that stops when val_accuracy is 1

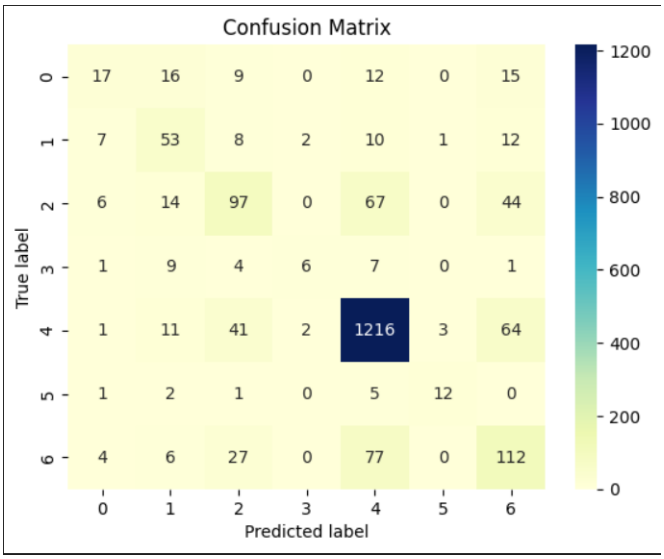
Now that the model architecture is decided, hyper-parameter tuning is performed to decide the following hyper-parameters: 1) Dropout Rate and 2) Batch Size. Once this was complete we obtained the model with the best hyper-parameters and the accuracy and loss were as follows:



As for performance on test data, the accuracy achieved was 75.53% and loss was 1.646. A classification report was printed which shows the f1-score, recall, precision, and accuracy per label as follows:

	precision	recall	f1-score	support
0	0.46	0.25	0.32	69
1	0.48	0.57	0.52	93
2	0.52	0.43	0.47	228
3	0.60	0.21	0.32	28
4	0.87	0.91	0.89	1338
5	0.75	0.57	0.65	21
6	0.45	0.50	0.47	226
accuracy			0.76	2003
macro avg	0.59	0.49	0.52	2003
weighted avg	0.75	0.76	0.75	2003

The resulting confusion matrix was:



The overall summarized test performance was as follows:

```
MSE:  1.963554667998003
F1:  0.5192906769116286
Precision:  0.589939473896062
Recall:  0.4902595030273305
Accuracy:  75.53669495756365
Classification Report:
              precision    recall  f1-score   support

           0       0.46      0.25      0.32        69
           1       0.48      0.57      0.52        93
           2       0.52      0.43      0.47       228
           3       0.60      0.21      0.32        28
           4       0.87      0.91      0.89      1338
           5       0.75      0.57      0.65        21
           6       0.45      0.50      0.47       226

    accuracy                           0.76      2003
   macro avg       0.59      0.49      0.52      2003
weighted avg       0.75      0.76      0.75      2003
```

In conclusion, the data being used has a lot of noise (unwanted hairs, and improper exposure etc.). So the best action would be to implement filters to remove these hindrances and to increase the dataset size and variety.

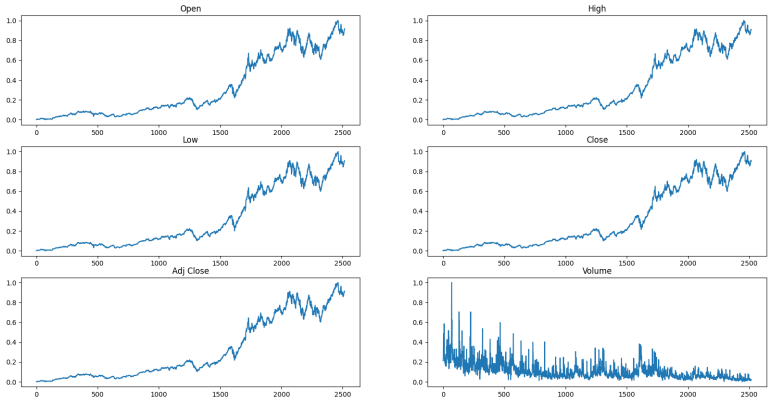
3 Task 2: Stock Price Prediction using RNN

In this task, we were required to download stock prices from the yfinance API and apply an RNN model to predict stock prices. To do this, first the dataset

for Apple stock prices for 10 years from 16-10-2013 to 16-10-2023 were loaded. The resultant shape of the data was 2516 rows each with 7 columns namely:

- Open
- High
- Low
- Close
- Adj Close
- Volume

Now that the dataset has been loaded, some pre-processing was done. First of all, missing values were filled using forward fill provided by the sklearn pre-processing module. Then the MinMaxScaler was applied to the data to normalize the column value to between 0 and 1. The distribution of the pre-processed data was as follows:



In this case, it was decided to predict the prices on the next immediate day based on prices and volumes from the past 7 days. So the data was converted from time series to sequence-by-sequence data. After the pre-processing was complete, a RNN model was defined with the following architecture:

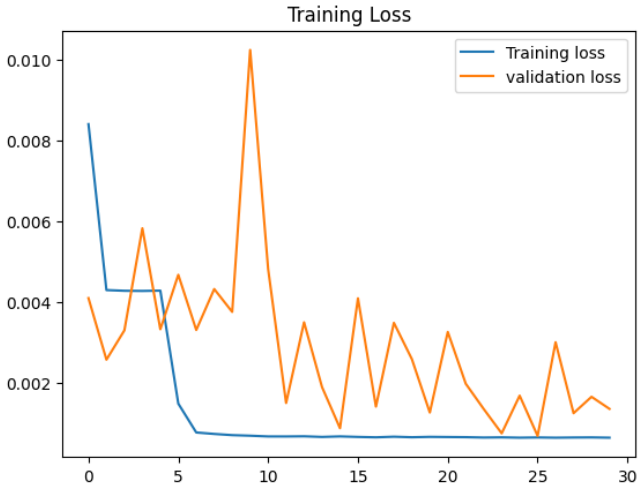
- LSTM layer
- A Dense layer with 16 neurons and ReLu
- A Dense layer with neurons equal to the number of columns with ReLu.

This RNN uses the Adam Optimizer and uses the mean_squared_error loss function. Once the model architecture was decided, hyper-parameter tuning was performed for the following hyper-parameters:

- Units i.e neurons to use in the LTSM layer
- Epochs for training

- Batch size for training

The performance on training data was as follows:



As for performance on test data, the Mean Squared Error was: 0.00353, the Root Mean Squared Error was: 0.0594 and the Mean Absolute Error was: 0.0522. In conclusion, we need a balance between underfitting and overfitting. This balance depends on the scenario and volatility of the data. Thus in this situation, the mean square error achieved is good enough to estimate the stock prices.

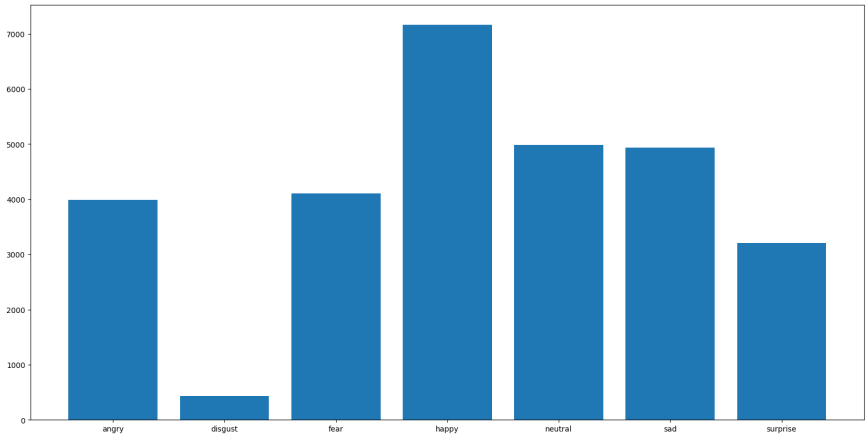
4 Task 3: Emotion Detection using CNN

In this task, it was required for us to use another facial recognition dataset from Kaggle and apply a CNN on it to detect and classify emotions. The datset contained two folders 1) Train and 2) validation. Thus following this scheme, the training was done on the images from the training folder, and the images from the validation folder were used for validation at the time of model fitting. The test data was obtained from the train images using the train test split which divided the data into 80% test data and 20% test data. There were a total of 28821 training images and a total of 7066 validation images.

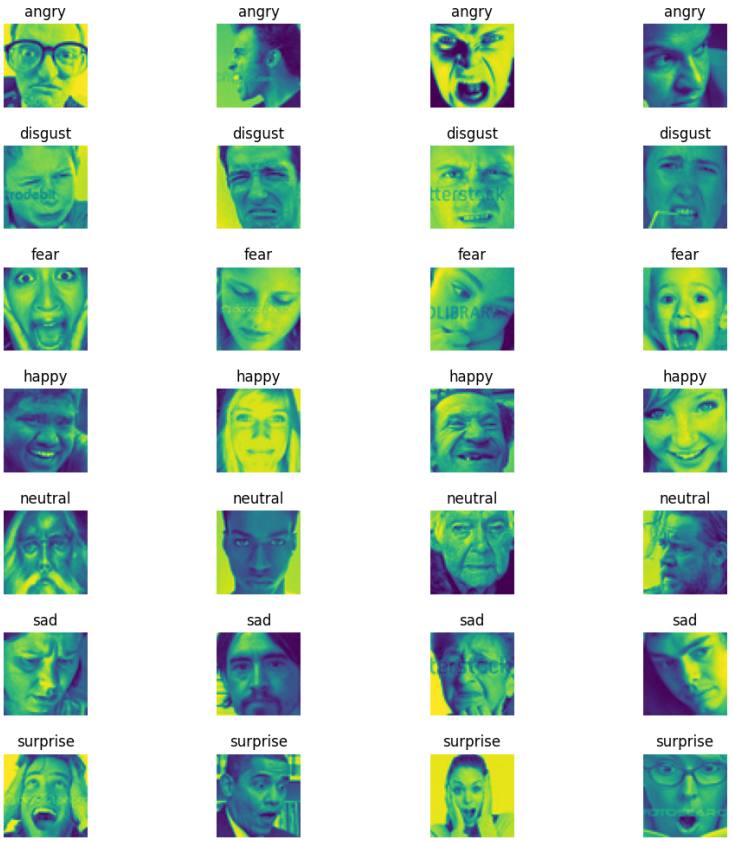
In pre-processing, the first step was extracting the labels for the images and dividing the data into labeled sections. The initial distribution upon loading and counting the labels was:

```
[(20289, 24281, 'angry', 3993),
 (24282, 24717, 'disgust', 436),
 (24718, 28820, 'fear', 4103),
 (4938, 12101, 'happy', 7164),
 (12102, 17083, 'neutral', 4982),
 (0, 4937, 'sad', 4938),
 (17084, 20288, 'surprise', 3205)]
```

This was plotted on a bar chart as:



After displaying the distribution of the data, 4 random images from each label were extracted and shown in the following graph:



Please note that the color is only due to the method used for loading the images and they are actually loaded as grey-scale images. Now some pre-processing was done to balance the classes. This was done using resampling from sklearn and the new images were altered slightly using ImageDataGenerator from Keras. The distribution of the classes was as follows:

```
(20289, 24281, 'angry', 3993)
Number of majority class images:  7164
Number of minority class images:  3993
(24282, 24717, 'disgust', 436)
Number of majority class images:  7164
Number of minority class images:  436
(24718, 28820, 'fear', 4103)
Number of majority class images:  7164
Number of minority class images:  4103
(12102, 17083, 'neutral', 4982)
Number of majority class images:  7164
Number of minority class images:  4982
(0, 4937, 'sad', 4938)
Number of majority class images:  7164
Number of minority class images:  4938
(17084, 20288, 'surprise', 3205)
Number of majority class images:  7164
Number of minority class images:  3205
```

After the class balancing and Data Augmentation, the number of training images had increased to 50148 images. Now in the next pre-processing step, the sklearn LabelEncoder was applied which encoded the textual labels to numerical ones interpretable by the CNN. After the pre-processing was complete, the images were reshaped to meet the input layer requirements and a CNN with the following architecture was set:

- 64 3x3 Conv2D Kernels
- Batch Normalization
- ReLu Activation
- 2x2 MaxPool2D
- Dropout Layer
- 64 5x5 Conv2D Kenels
- Batch Normalization
- ReLu Activation
- 2x2 MaxPool2d
- Dropout Layer
- Flatten Layer
- Dense Layer with 32 Neurons
- Batch Normalization
- Dense Layer with Neurons same as labels with Softmax

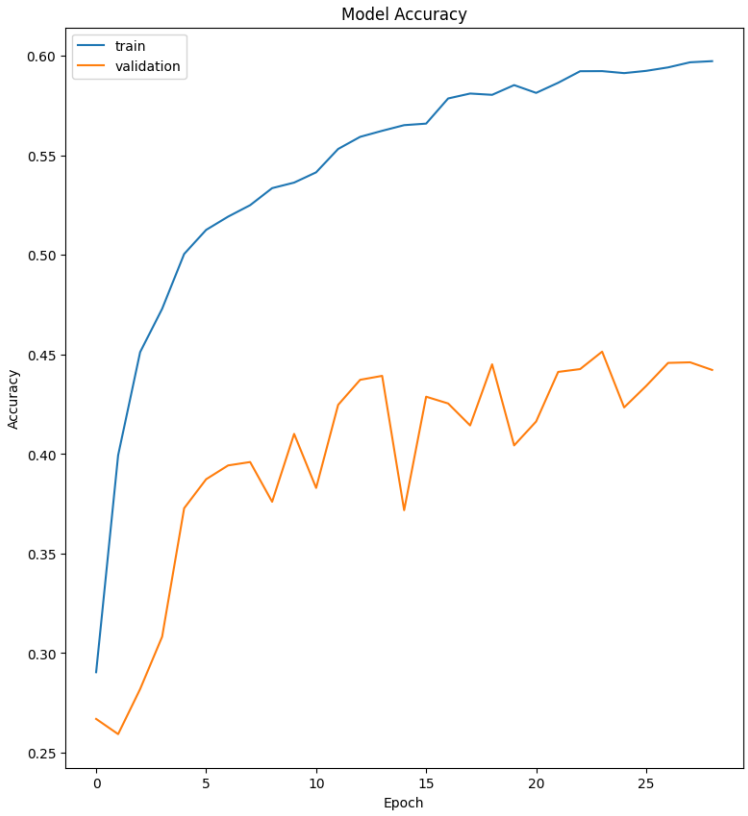
The optimizer for the model was Adam and the loss function used was sparse_categorical_crossentropy. The callback set was:

- Early Stopping
- Reduce Learning Rate on Plateau

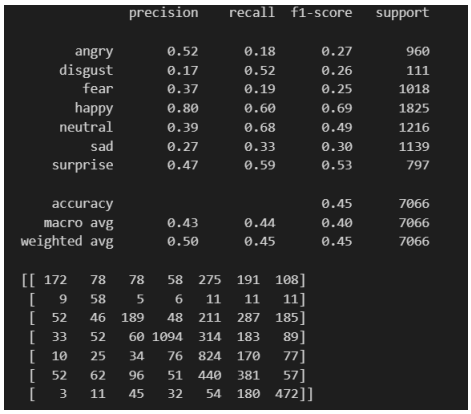
Once the CNN was set, hyper-parameter tuning was performed to select the following hyper-parameters after the dataset was shuffled to induce randomness:

- Dropout Rate
- Learning Rate
- Batch Size

The training and validation accuracies were as follows:



The test accuracy was: 54% and the training loss was: 1.1920. The classification report and confusion matrix were as follows:



	precision	recall	f1-score	support
angry	0.52	0.18	0.27	960
disgust	0.17	0.52	0.26	111
fear	0.37	0.19	0.25	1018
happy	0.80	0.60	0.69	1825
neutral	0.39	0.68	0.49	1216
sad	0.27	0.33	0.30	1139
surprise	0.47	0.59	0.53	797
accuracy			0.45	7066
macro avg	0.43	0.44	0.40	7066
weighted avg	0.50	0.45	0.45	7066

```
[[ 172   78   78   58  275  191  108]
 [   9   58    5    6   11   11   11]
 [  52   46  189   48  211  287  185]
 [  33   52   60 1094  314  183   89]
 [  10   25   34   76  824  170   77]
 [  52   62   96   51  440  381   57]
 [   3   11   45   32   54  180  472]]
```

In conclusion, the model overfits on the data very easily thus early stopping keeps a balance between underfitting and overfittting the data. The dataset used is not a very good one as it has low quality images and has too much noise (watermarks, underexposed images, and improper angles etc.). The best way forward to improve model performance is to switch to a better dataset.