UNIVERSITY OF BUCHAREST

FACULTY OF MATHEMATICS
AND COMPUTER SCIENCE

MASTER SECURITY AND APPLIED LOGIC

Deep learnining

REPORT KAGGLE AI GENERATED IMAGES CLASSIFICATION

Author
Dory Mathis

Professors
Radu Tudor Ionescu

Bucharest, January 2024

Contents

1. Introduction

The task of this Kaggle competition was to classify 5000 test AI generated images among 100 different classes. For the training dataset, 13 000 images were provided and 2000 more were used for validation. The training was made on an *Apple Macbook Air M1* laptop by using the *Keras* and *Tensorflow* library. Different models have been used, but the best results were obtained by a RESNET-type model. A more classical CNN (Convolutionnal Neuronal Network) and an LSTM (Long Short Term Memory) were also tested, even if an LSTM model is not usually used in image classification.

2. Pre-processing

The images used in the different models have all been pre-processed in the same way, apart from few slight differences in data augmentation. Firstly, the init function is used to log some information such has the data distribution, the shape of dataset, and few others. It also plots 4 random images from the training set so the user has an overview. The project uses the *flow_from_directory* function from Keras in order to create a dataframe of images but, it is required to firstly classify all the images with the same label into a single folder having the name of the related label. In order to do it, the *structure_data* function was implemented on the training and the validation dataset. It is then required to put all images in a separate folder and the *prepare_test* function was used to do it. When using the *flow_from_directory* function, the project uses the *ImageDataGenerator* on the training, the validation and the test images in order to normalize the images into a value between 0 and 1. For the training dataset, we also apply some data augmentation in order to generate transformed versions of images in the dataset. The augmentation includes several transformations:

1. Rotation: Images are randomly rotated within a chosen range.

2. Width and Height Shifts: Random shifts of the image along the width and height introduce spatial variance, simulating scenarios where objects of interest are not centrally aligned.

3. Shear Transformation: Shearing distorts the image in a fixed direction, simulating a change in viewing angle.

4. Zoom: Random zooming manipulates the scale of the images, allowing the model to recognize features at different scales.

5. Brightness Range Adjustment: Varying the brightness imitates different lighting conditions.

6. Horizontal Flipping: Mirroring images horizontally.

b92bec7a-3218-4032-b1d3-6ee349db5c26.png | Class: 9
c51d8873-7bf3-44fb-b2ad-93f0a0e1fabf.png | Class: 19
0ce38728-ae55-458b-aec1-ef920733b0de.png | Class: 76
7363e314-6fb8-4cca-8936-bb76d596ec68.png | Class: 67

Figure 2.1: Training images samples

7. Channel Shift Range: Adjusting the RGB channels of the images, simulating variations in color and lighting conditions. This helps the model in becoming more robust to color variations in the input data.

The preprocessing function return 3 arguments: the validation generator, the training one and the test generator. The training and validation generators are used to create and train the model, whereas the generator test is used for predictions.

3. RESNET model

The RESNET model (Residual Network) was chosen for its superior performance in image classification tasks. The RESNET architecture uses skip connections/shortcuts and allows passing through the network without passing through multiple layers. This architecture effectively addresses the vanishing gradient problem, enabling the training of much deeper networks. Different versions of RESNET exist, depending on the number of layers used, such as RESNET50, 101 and many others. In theory, the deeper the model, the more it is able to make a difference in the details of an image. After testing RESNET50, which turned out to be too deep for the dataset (it overfitted and kept an accuracy on the validation data of 1%), a lighter RESNET including 20 layers (RESNET20) was developed.

3.1 Model architecture

The model starts with a basic Convolutional layer + Batch normalisation + Relu activation, then residual layers are added. This consists of 3 layers of 3 blocks and each block contains a convolution + batch normalisation + Relu activation followed by a new convolution + batch normalisation. To create a skip layer, the initial gradient of the block is added to the output gradient of the block, followed by a final Relu activation. However, in the case of the first block of the second and third layers, it is necessary to add a new convolution to the input of the block, with a kernel (1,1) and a stride 2 followed by batch normalisation, in order to maintain a compatible shape between input and output at the time of concatenation. At the end of the model, a GlobalAveragePooling2D layer is used in order to reduce each feature map by to a single number which is the average of that feature map. Then, the flatten layer transform the feature map in a single vector in order to be used for the fully connected layer.

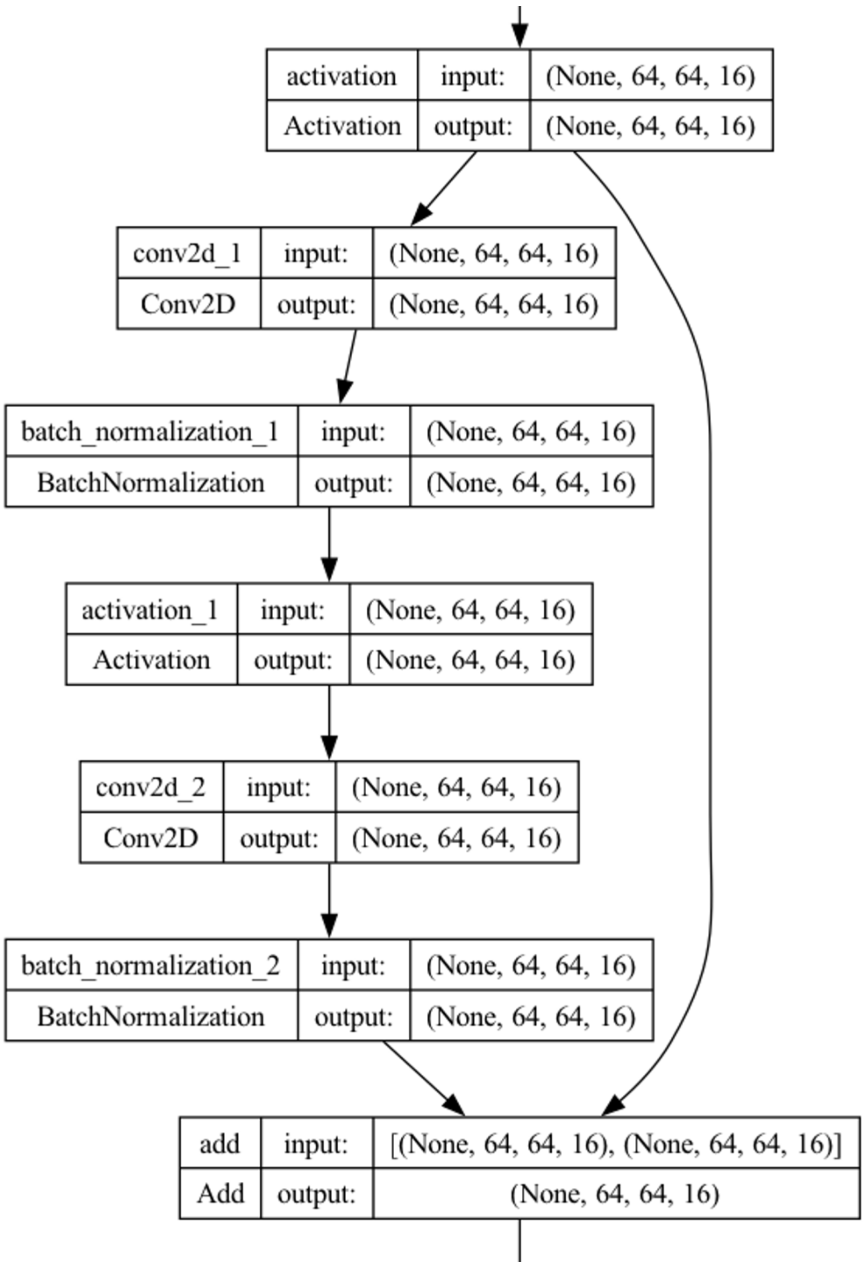


Figure 3.1: Skip layer of the RESNET20 model

3.2 Hyper parameters

For the RESNET model, the batch size used was set to 64. The kernel initializer used for the convolutional layer is the "HeNormal" one. The HeNormal initializer set the weights in a way that the variance of the outputs of a layer is equal to the variance of its inputs, maintaining a balance that helps prevent the gradients from vanishing or exploding. The first convolutional layer starts with a 16 filter size, 3x3 kernel size and padding set to "same". For the residual layers the filter size is the same for all blocks of an entire group but double after each group after starting at 16 ((16,16,16),(32,32,32)(64,64,64)). The kernel size is always 3x3 and the strides are 1x1 except for the first layer of the second and third group.

3.3 Training

For the training of the model, 3 callbacks are used: the EarlyStopping one (stop after 50 trials where the validation loss is the lowest), the ModelCheckpoint (where we save the model where the validation accuracy is maximum) and the ReduceLROnPlateau (reduce the learning rate when the validation loss stops improving). Thanks to the callbacks, there's no need to worry about the number of epochs, as the training will stop on its own once the best epoch has been reached, so a relatively high value can be chosen for the epochs (200).

3.4 Results

By plotting the training accuracy and validation accuracy, some observations can be made:

- The training accuracy (red line) increases sharply at the beginning, indicating that the model is quickly learning from the training data.
- The validation accuracy, (purple line), similarly increases sharply at the start, indicating that the model's initial learning is generalizing well to the unseen data.
- Between epoch 15 and 70 the validation accuracy is really unstable but tends to stabilize after around 70% of accuracy.
- The gap between the purple and red line may indicate that the model is over fitting a little bit.

Concerning the training and validation loss, the following observations can be made:

- In the initial epochs, there is a sharp decrease in both training loss (green line) and validation loss (blue line), suggesting rapid initial learning and improvement from the model.
- The training loss continues to decline steadily over epochs, eventually reaching a plateau, which indicates that the model is achieving a good fit on the training data.
- The validation loss is more unstable but overall shows a downward trend, although it does not level out as distinctly as the training loss. This points to the model's varied response to the complexities of the unseen validation data.
- A small gap is observed between the training and validation loss which indicates that the model is overfitting a little bit, this could be improved.

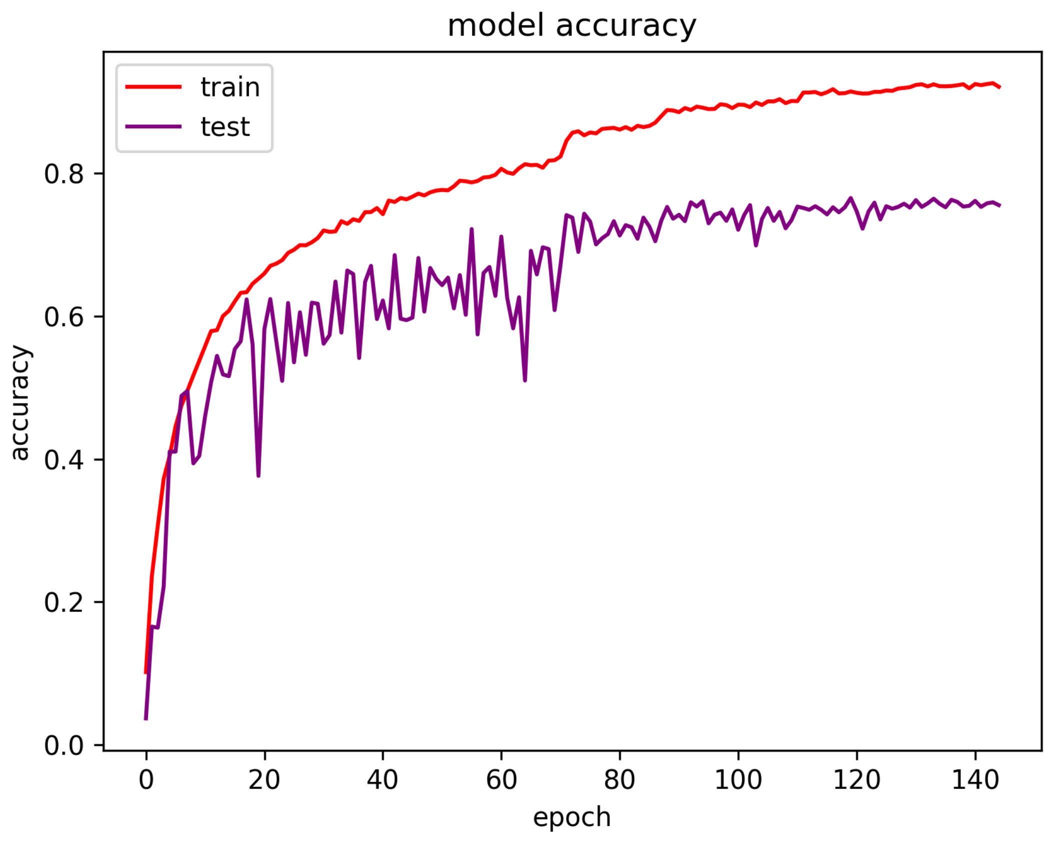


Figure 3.2: Training and validation accuracy of the RESNET20 model

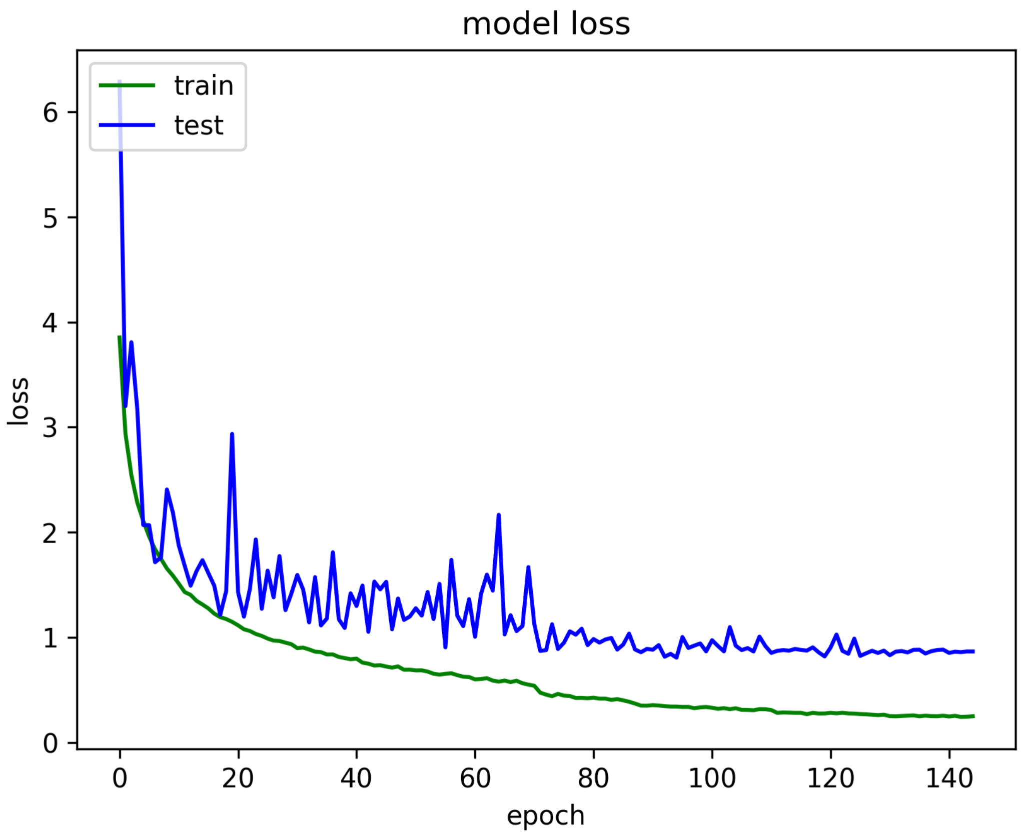
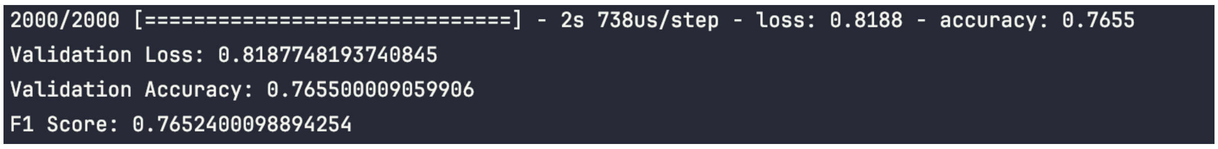


Figure 3.3: Training and validation loss of the RESNET20 model



```
2000/2000 [==============================] - 2s 738us/step - loss: 0.8188 - accuracy: 0.7655
Validation Loss: 0.8187748193740845
Validation Accuracy: 0.765500009059906
F1 Score: 0.7652400098894254
```

Figure 3.4: Scores of the RESNET20 model

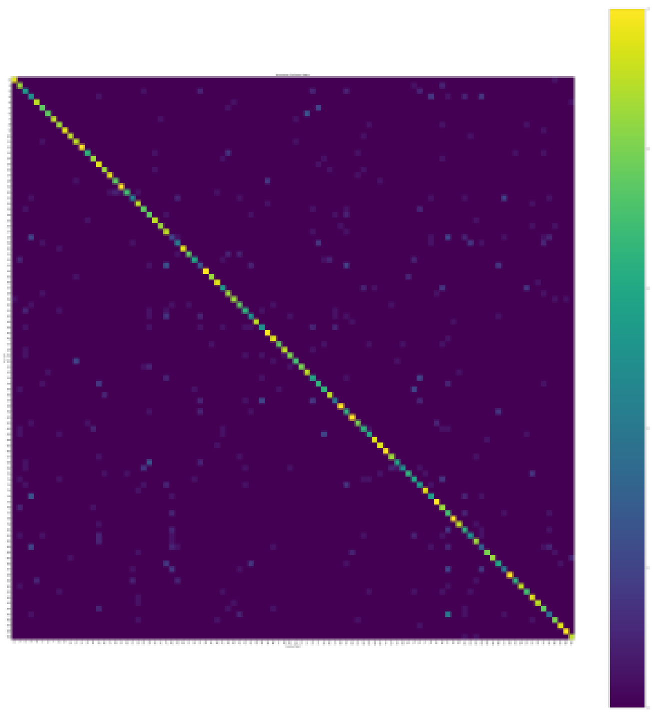


Figure 3.5: Normalized confusion matrix of the RESNET20 model

Concerning the scores of this model, the callbacks stopped training the model when it obtained the best combination of validation accuracy and validation loss. The validation accuracy was 76.55% for a validation loss of 0.8187% as shown in figure 3.4. The F1 score is also provided and had a value of 0,765 meaning the model has a good balance between precision (the number of true positive results divided by the number of all positive results, including those not identified correctly) and recall (the number of true positive results divided by the number of all samples that should have been identified as positive). Concerning the normalized confusion matrix on figure 3.5, it shows that for most classes, the percentage of correctly classified images is quite high (the lighter the colour, the higher the percentage) however, some off-diagonal elements appear lighter, indicating instances of misclassification. This points towards specific classes where the model may be confusing in exchanging one class for another.

4. Others models used

4.1 CNN

Another tested model is the 'basic' Convolutional Neural Network (CNN). For this model a batch of 128 was used and the optimizer *Adam* for the model compilation. Here again the following callbacks were used:

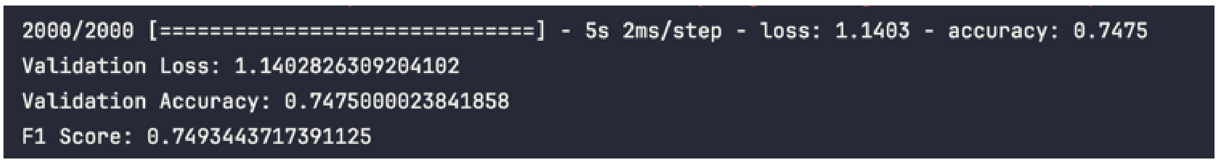
- EarlyStopping
- ModelCheckpoint
- ReduceLROnPlateau

4.1.1 Architecture

This model consists of 3 convolutional layers (Conv2D, MaxPooling2D, Dropout). Each convolutional layer utilizes 256, 512, and 512 filters respectively, with a kernel size of 3×3. The padding is set to 'same' to keep the spatial dimensions constant after convolution. After each convolutional layer, a MaxPooling2D layer with a pool size of 2×2 in order to reduce the spatial dimensions. The Dropout value is set to 0.2 in order to drop 20% of the neurons to prevent overfitting. Following the convolutional base, BatchNormalization is applied. Most of the time this layer is used after a convolutional layer but better results were observed by putting it at the end. The BatchNormalization is used to stabilize learning by normalizing the input layer by re-centering and re-scaling. Then, GlobalAveragePooling2D is utilized to reduce each feature map to a single number by averaging out the spatial dimensions, reducing the total number of parameters in the model and the result of it is passed to a first fully connected layer with 128 units. The dense layer uses the ReLU activation and is initialized with the VarianceScaling method to maintain a unit variance of gradients and weights. Additionally, it employs L2 regularization on the kernel and activity to further mitigate the risk of overfitting. The final layer uses the softmax activation in order to classify the output among the labels.

4.1.2 Results

As shown on figure 4.1, it is clear that this model does not perform as well as the previous one, its validation accuracy is slightly lower but the validation loss is higher. The gap between the training and validation lines is smaller than on the RESNET plots (figures 4.2 and 4.3). This means that the CNN model overfits less than the RESNET model, even though the final scores are lower. The CNN model seems also more stable than the RESNET one. The normalized confusion matrix (Figure 4.4) reveals that, while the model correctly classifies a high percentage of instances for most classes, there are noticeable instances of misclassifications, as indicated by the lighter shades off the diagonal.



```
2000/2000 [==============================] - 5s 2ms/step - loss: 1.1403 - accuracy: 0.7475
Validation Loss: 1.1402826309204102
Validation Accuracy: 0.7475000023841858
F1 Score: 0.7493443717391125
```

Figure 4.1: Scores for the CNN model

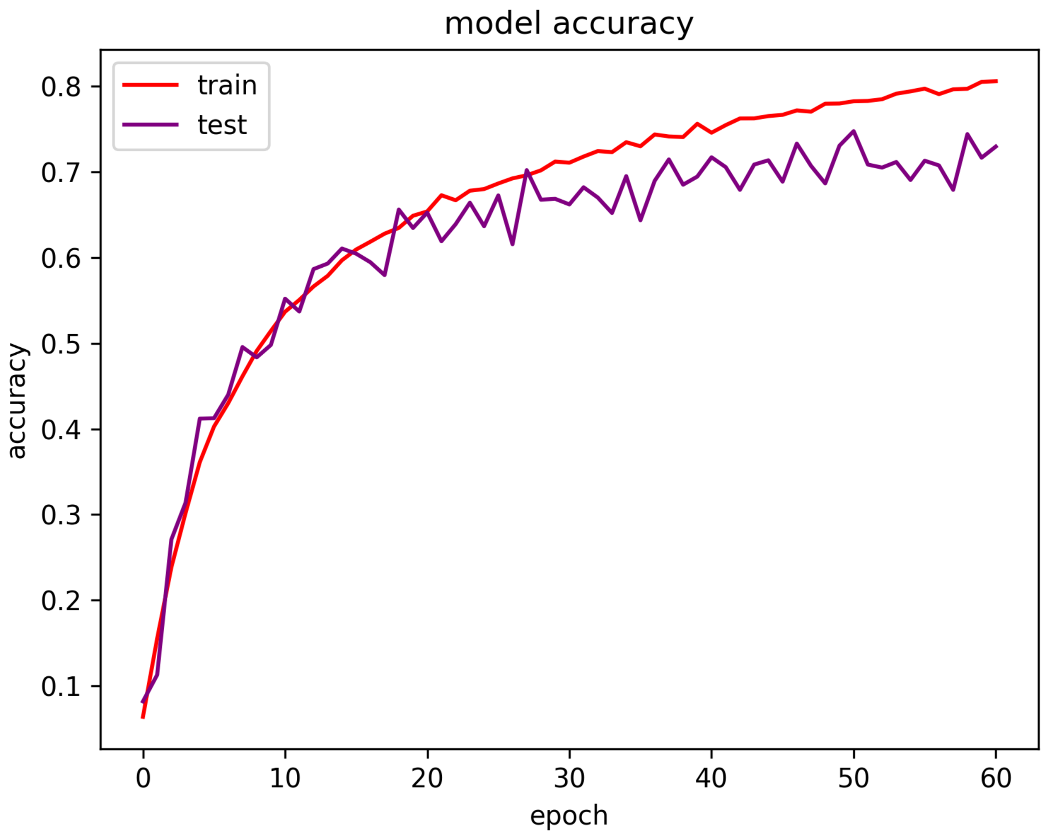


Figure 4.2: Accuracy of the CNN model

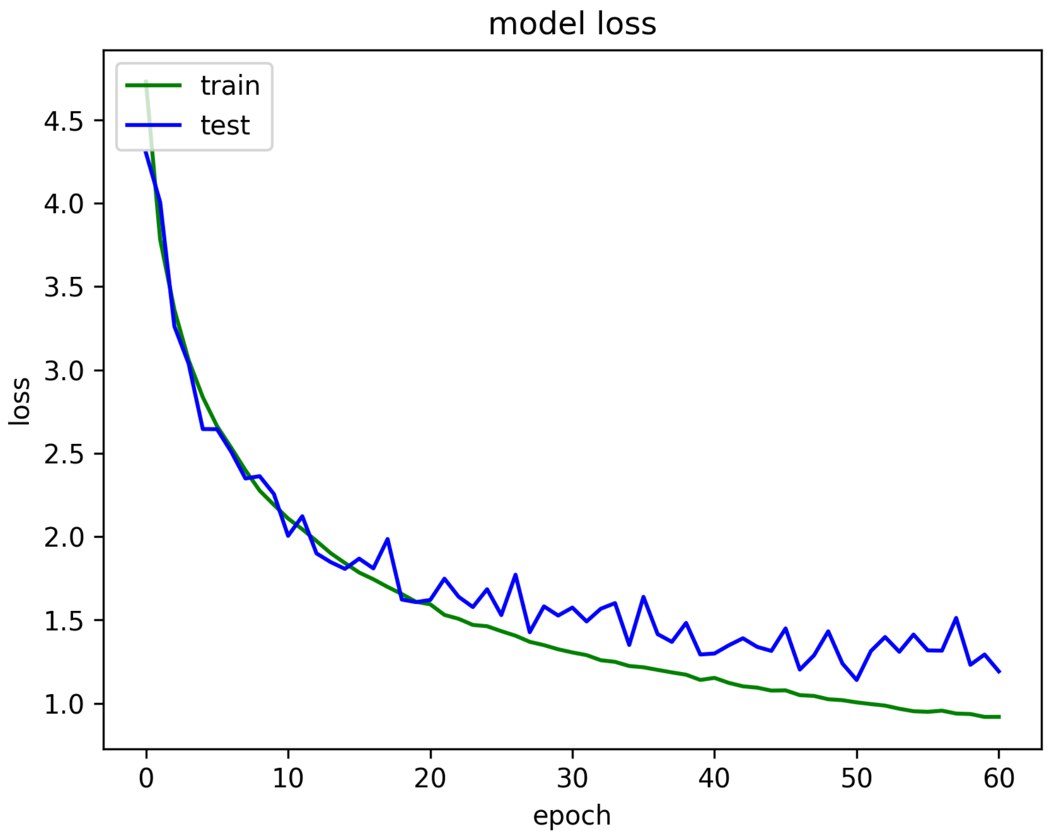


Figure 4.3: Loss of the CNN model

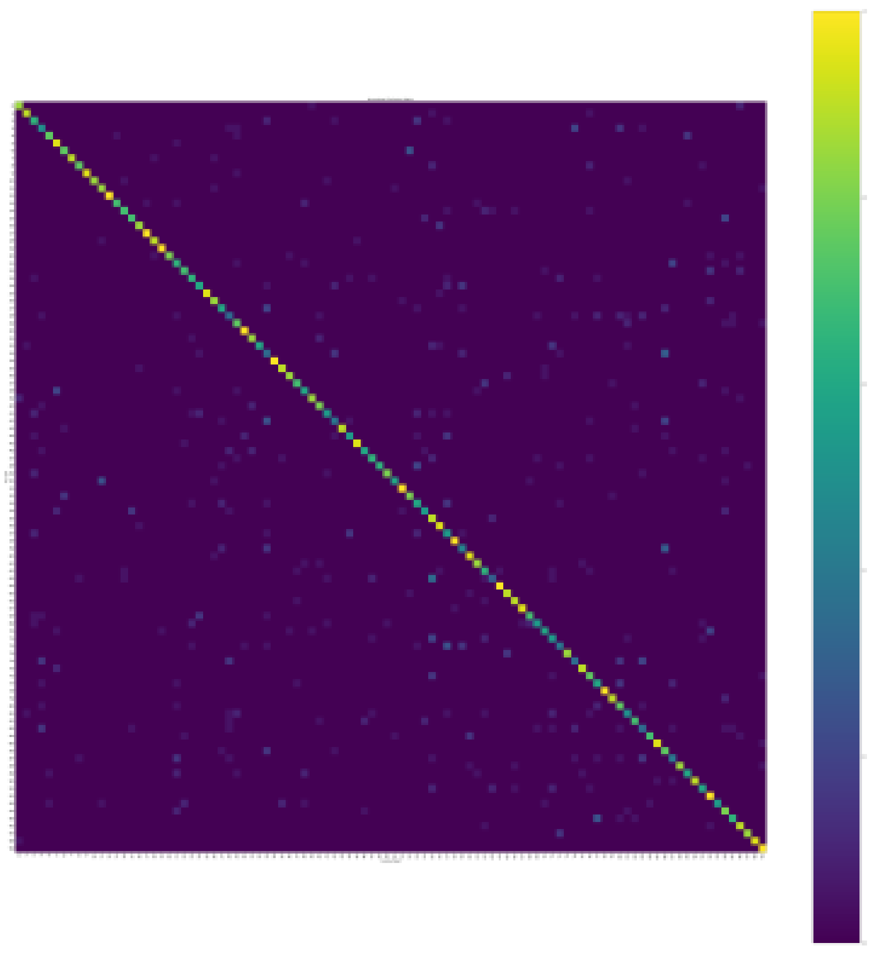


Figure 4.4: Normalized confusion matrix of the CNN model

4.1.3 Hyper parameters tuning

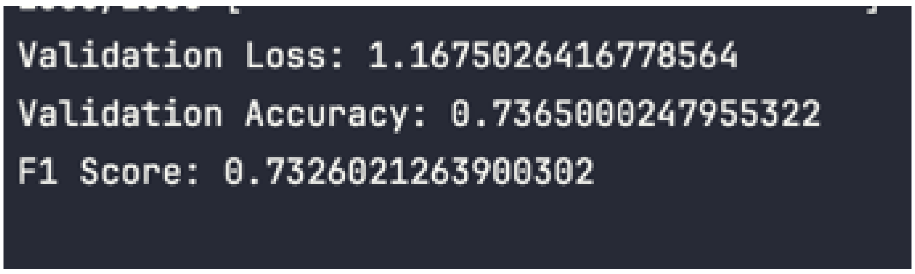
For the CNN model, the *Keras tuner* library was used in order to perform Hyper parameters tuning and search for a better model. Two Python files were used:

- **cnn_tuning.py** is used to search for the best hyper parameters tuning and train the model.
- **predict_tuning.py** is used to retrain the model with higher epoch by including the best hyper parameters found by **cnn_tuning.py**.

The design of the tuning model was based on the previous CNN model (3 layers of Conv2D + Maxpool2D + Dropout). The main difference here is that before calling the

Flatten layer, the model can choose between a GlobalMaxPooling2D layer or a Global-AveragePooling2D layer. It also add the possibility to use one extra Dense + Dropout layer before the final one. Of course, all hyper parameters had the choice between multiples values such as 32,64 or 128 for the filter size of the first Conv2D layer for example. The algorithm used is the *RandomSearch* with an objective of finding the best validation accuracy with a max trials of 15 and 20 epoch for each trial. When the search is done, the **predict_tuning.py** file is used to load the saved model and retrain it using higher epoch number. Unfortunately this method is highly time and resources consuming (more than 7 hours for the training), therefore, it was decided to run it only once, even if after some research it could have been better optimised. The results obtained by the hyper parameters tuning were lower than the ones obtained by the basic CNN model as shown in the figures 4.5, 4.6, 4.7. The best model obtained from this tuning has the following parameters and is from trial 12 (starting from 0):

```
{"values": {
      "filters_1": 128,
      "padding_1": "same",
      "kernel_1": 3,
      "strides_1": 2,
      "dropout_1": 0,
      "filters_2": 128,
      "kernel_2": 3,
      "padding_2": "same",
      "strides_2": 2,
      "dropout_2": 0.1,
      "filters_3": 512,
      "kernel_3": 3,
      "padding_3": "same",
      "strides_3": 1,
      "dropout_3": 0.1,
      "pooling": "global_avg_pooling",
      "Dense_sup": false,
      "dense_final": 128,
      "dropout_final": 0.3,
      "dense_units_sup": 512, -> is not relevant because the Dense_sup
   is False
      "dropout_sup": 0.30000000000000004 -> is not relevant
    }}
```



```
Validation Loss: 1.1675026416778564
Validation Accuracy: 0.7365000247955322
F1 Score: 0.7326021263900302
```

Figure 4.5: Scores of the tuned model

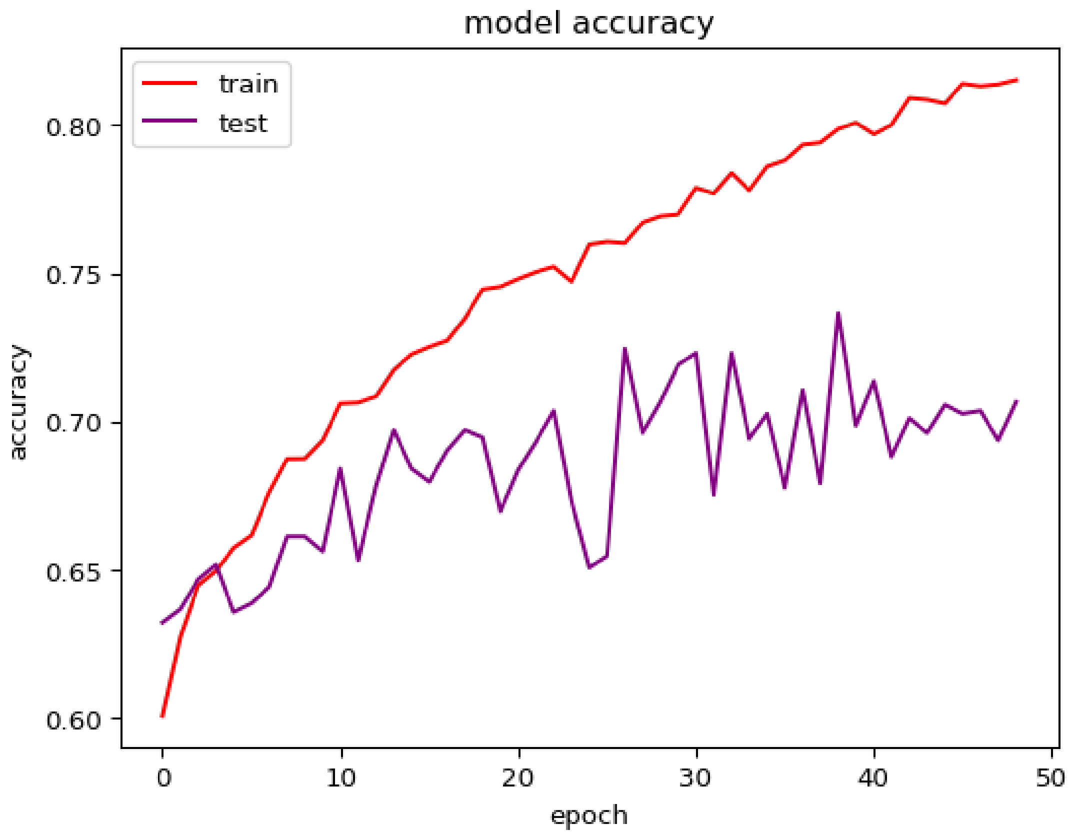


Figure 4.6: Accuracy of the tuned model

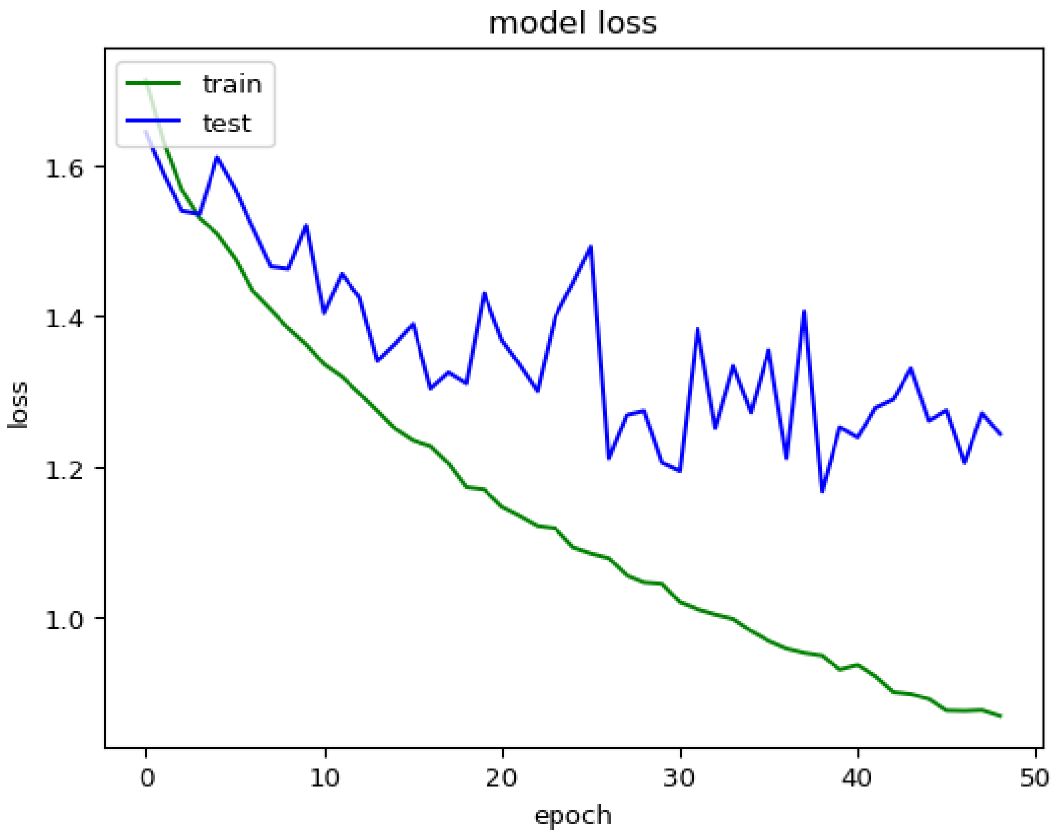


Figure 4.7: Loss of the tuned model

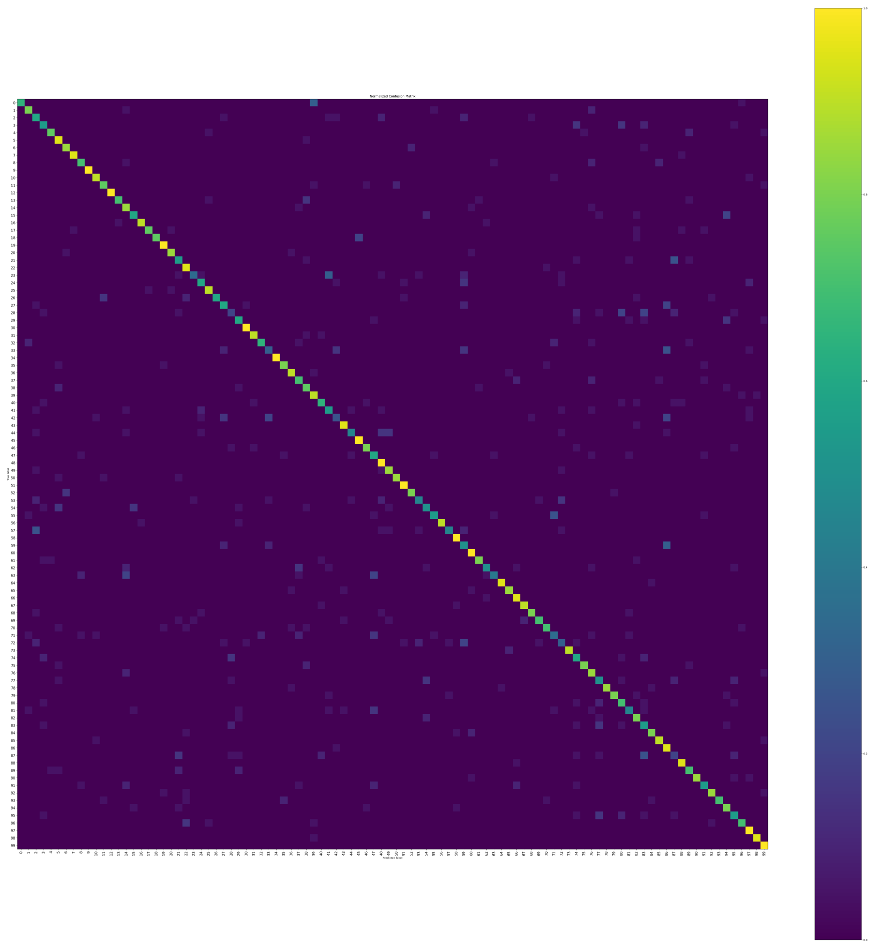


Figure 4.8: Normalized confusion matrix of the tuned model

4.2 LSTM

The last tested model is the LSTM (Long Short Term Memory). In general, the convolutional architecture is preferred for images classification but LSTM was implemented in order to see the difference. Here again the pre-processing step is the same than for the others models but the model architecture is sightly different. LSTM model works well with 1D features so it was necessary to reshape the input with a first reshaping layer. Instead of having a (64,64,3) input shape, it is configured to output a (64,64x3) shape. Then, 5 LSTM + Dropout layers are used starting with 64 units then 2x128 and 2x256 units, with a Dropout of 20%. Then it uses a first fully connected layer with 256 units and Relu activation, l2 regularizer and VarianceScaling for the kernel initializer with a new Dropout of 20% before the last fully connected layer using the softmax activation and the 100 units as labels. The training was made using *Adam* optimizer and the same callbacks as usual. The followings results are observed:

- As predicted, this model as the lowest accuracy and F1 score and the largest loss compared to the RESNET and CNN models as shown in figure 4.11.
- Quiet surprising the model is not overfitting but it is underfitting starting from epoch 40 as we can see on the accuracy figure 4.9 and the loss figure 4.10.
- The confusion matrix has high percentage of missclassification (only a small amount of classes have score in the yellow color) as figure 4.12 is showing.

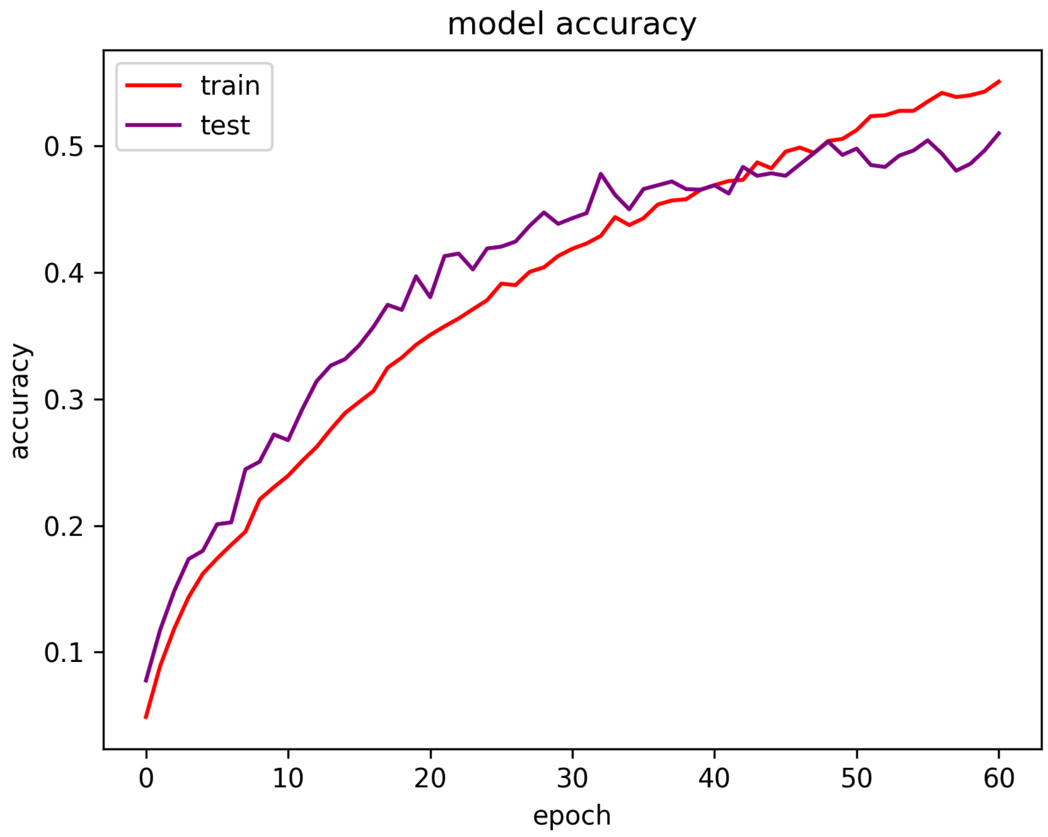


Figure 4.9: Accuracy of the LSTM model

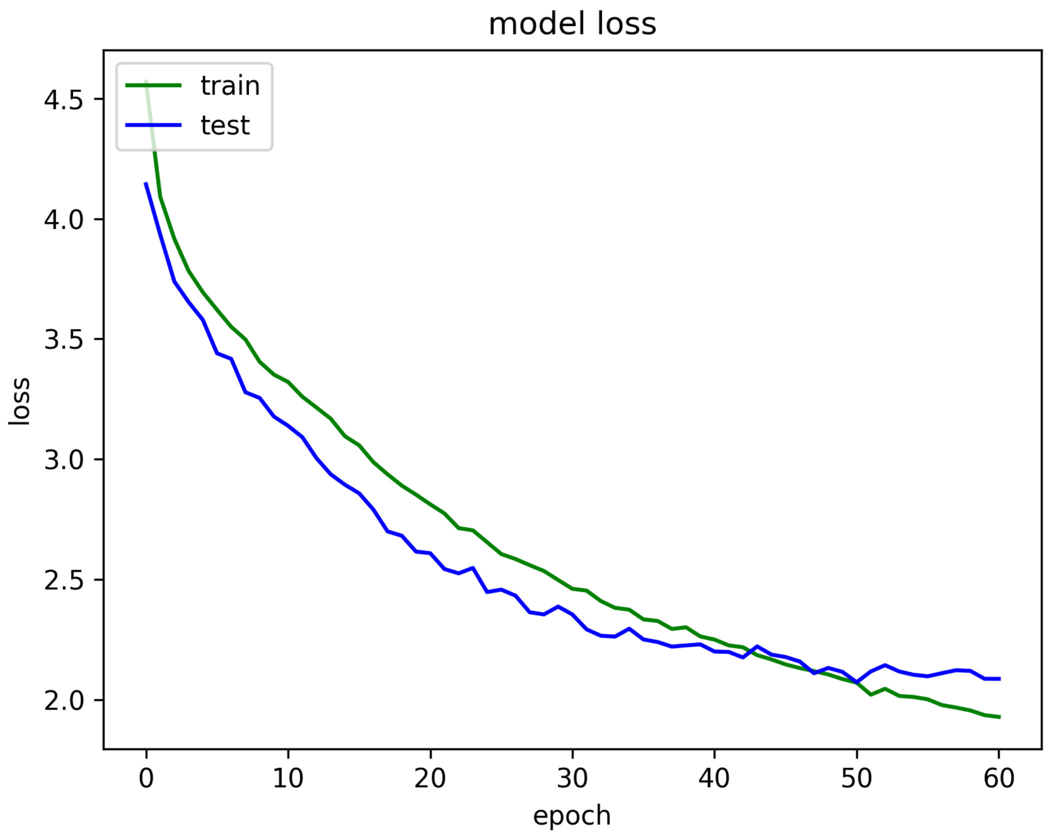
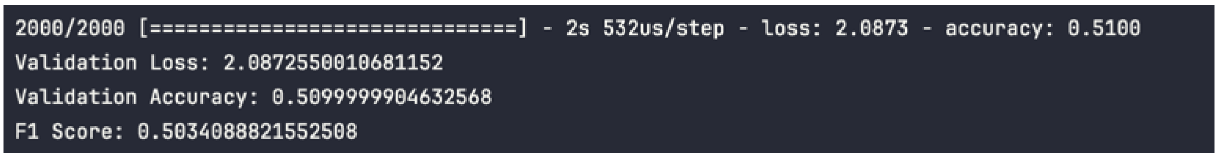


Figure 4.10: Loss of the LSTM model



```
2000/2000 [==============================] - 2s 532us/step - loss: 2.0873 - accuracy: 0.5100
Validation Loss: 2.0872550010681152
Validation Accuracy: 0.5099999904632568
F1 Score: 0.5034088821552508
```

Figure 4.11: Scores of the LSTM model

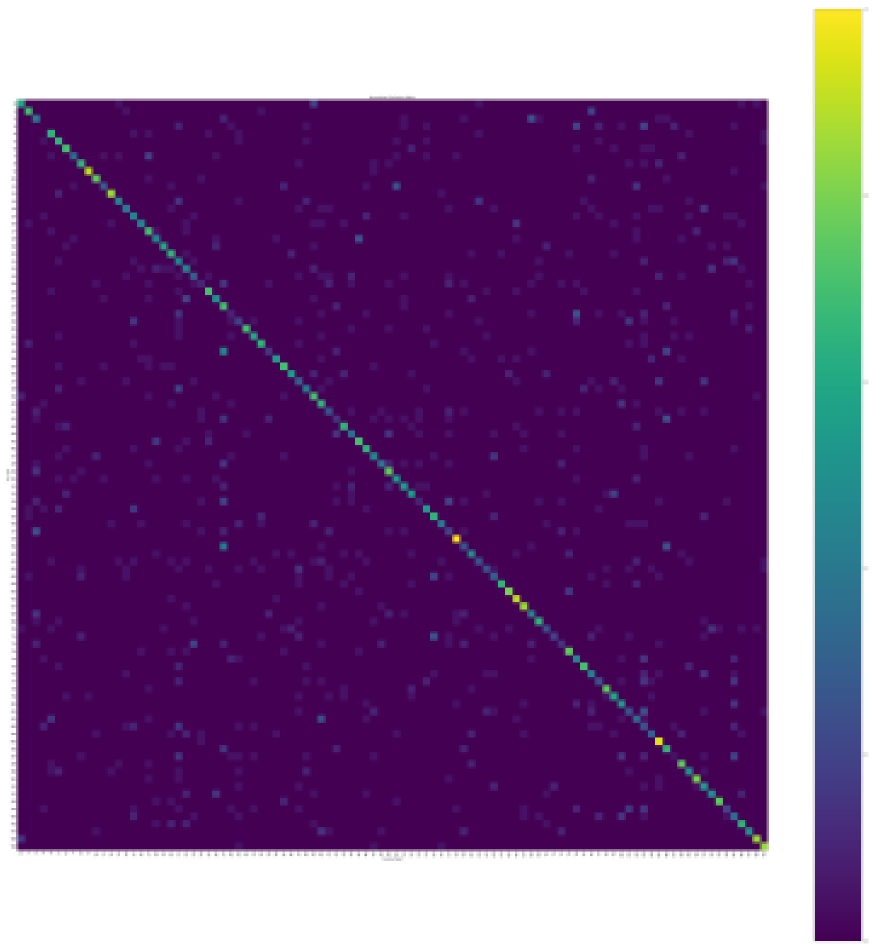


Figure 4.12: Normalized confusion matrix of the LSTM model

5. Conclusions

The exploration of different neural network architectures for the classification of AI-generated images presented valuable insights. The RESNET20 model demonstrated the best performance with a validation accuracy of 76.55% and a F1 score of 0.765, indicating a robust capability in feature detection and generalization. Moreover, this model got an F1 score of 0.75017 in the final Kaggle leaderboard, having the 25th best score in the ranking. Despite its efficiency, a slight overfitting was observed, suggesting some improvement can be made. The CNN model, while slightly underperforming compared to RESNET20, showed less overfitting and a more stable behavior during training. Unexpectedly, the LSTM model, usually not suited for image classification, got not such a bad score and showed some rather surprising results.

Bibliography

[1] *CNN - How to use 160,000 images without crashing.* URL: %5Curl%7Bhttps://kaggle.com/code/vbookshelf/cnn-how-to-use-160-000-images-without-crashing%7D (visited on 12/05/2023).

[2] *Convolutional Neural Networks for Image Classification — by KHWAB KALRA — Medium.* URL: %5Curl%7Bhttps://medium.com/@khwabkalra1/convolutional-neural-networks-for-image-classification-f0754f7b94aa%7D (visited on 12/22/2023).

[3] *Imbalanced Garbage Classification — ResNet50.* URL: https://kaggle.com/code/farzadnekouei/imbalanced-garbage-classification-resnet50 (visited on 01/05/2024).

[4] *Implementing ResNet in PyTorch.* GitHub. URL: %5Curl%7Bhttps://github.com/FrancescoSaverioZuppichini/ResNet%7D (visited on 01/04/2024).

[5] *Kannada MNIST:CNN Tutorial with App.(Top %2).* URL: %5Curl%7Bhttps://kaggle.com/code/benanakca/kannada-mnist-cnn-tutorial-with-app-top-2%7D (visited on 12/23/2023).

[6] *machine-learning-articles/how-to-build-a-resnet-from-scratch-with-tensorflow-2-and-keras.md at main · christianversloot/machine-learning-articles.* GitHub. URL: %5Curl%7Bhttps://github.com/christianversloot/machine-learning-articles/blob/main/how-to-build-a-resnet-from-scratch-with-tensorflow-2-and-keras.md%7D (visited on 01/05/2024).

[7] Adrian Rosebrock. *Convolutional Neural Networks (CNNs) and Layer Types.* PyImageSearch. May 14, 2021. URL: %5Curl%7Bhttps://pyimagesearch.com/2021/05/14/convolutional-neural-networks-cnns-and-layer-types/%7D (visited on 12/19/2023).

[8] Yashowardhan Shinde. *How to code your ResNet from scratch in Tensorflow?* Analytics Vidhya. Aug. 26, 2021. URL: https://www.analyticsvidhya.com/blog/2021/08/how-to-code-your-resnet-from-scratch-in-tensorflow/ (visited on 01/05/2024).

[9] Keras Team. *Keras documentation: KerasTuner*. URL: https://keras.io/keras_tuner/ (visited on 12/13/2023).

[10] *Youtube-apprendre-le-deeplearning-avec-tensorflow/#4 - CNN/A_deep_introduction_to_CNN (1).ipynb at master · anisayari/Youtube-apprendre-le-deeplearning-avec-tensorflow*. GitHub. URL: %5Curl%7Bhttps://github.com/anisayari/Youtube-apprendre-le-deeplearning-avec-tensorflow/blob/master/%234%20-%20CNN/A_deep_introduction_to_CNN%20(1).ipynb%7D (visited on 12/06/2023).