Deepfake Image Detection Using Convolutional Neural Networks

Max Hansson, Martin Holmqvist, Rashiq Al Tariq[1]

Abstract

Deepfakes are artificially edited or generated images, videos, or audio and are a type of synthetic media. The analysis performed in this report focuses on the increasingly important task of detecting deepfake images that have been manipulated or generated artificially. The analysis involves building and training convolutional neural networks (CNNs), implemented using PyTorch. The aim is to develop a model that can distinguish between real images and artificially generated ones with sufficient performance when predicting unseen image data. Two models will be built for this purpose, one self built CNN and one transfer learning based CNN. Data augmentation and tuning of model parameters is used to increase the performance of the neural networks. Performance metrics such as accuracy, sensitivity and specificity are used to evaluate the models. The results show that the transfer learning CNN model outperformed the self built CNN in terms of all performance metrics.

1. Introduction

Artificial intelligence has seen rapid developments in recent years. From advanced large language models to self-driving cars, these developments come with hopes of advancements that will lead to improvements for society and the world as a whole. Examples of such improvements would be increased work productivity, innovation, and overall quality of life. However, in the midst of these promises of a better future, there are deep concerns surrounding the potential dangers that might present themselves with these developments. One area that has raised such concerns is that of artificially generated content. This report will cover the topic of ”Deepfake” images. Deepfakes are artificially edited or generated images, videos, or audio and are a type of synthetic media.

The concern related to this type of content stems from the fact that these deep-fake images could be used for malicious purposes. An article (Westerlund, 2019) details several scenarios highlighting the potential risks associated with the misuse of deepfake technology. For example, one could use deepfake images for impersonation, blackmail, or spreading misinformation. This type of technological abuse could result in potentially severe consequences to affected parties in a variety of different sectors such as politics, business, and news media. Therefore, it is of interest to develop models with the capability of discerning between deep-fake images and real images. As we spend increasingly more time on social media, and with artificially generated content rapidly improving, the necessity of being able to identify artificially generated content is quickly growing.

In this report, we will develop, compare and present two models that can distinguish deepfake images from real images. The goal is to determine if a classification model for deepfaked images benefits from transfer learning, or if a model solely trained on the specific data set is to be preferred. Thus, we will investigate what model is optimal for this kind of classification problem.

2. Data and Methods

2.1. Data

The data used for the analysis consists of 190 305 images with 256x256 pixels and are in a jpeg format. The pictures depict humans in a variety of settings and are either real pictures or deepfakes with each picture having a label indicating whether they are real or fake. In Figure 5 there are 4 examples of photos that are deepfakes and Figure 6 displays 4 real pictures. At first glance, the deepfakes look rather real but when looking closer one can notice details that expose them to being fakes such as the pink glasses in Figure 5. The data set contains 95 213 real pictures and 95 092 deepfake pictures. 140002 images will be used for training, 39428 for validation, and 10905 for testing the final models. The data was sourced from the Kaggle data set repository where the stated split of the dataset into training, validation and test sets was predetermined.

2.2. Methods

For the purpose of classifying images as real or fake, two models will be developed and implemented. The models to be implemented are both convolutional neural networks (CNN). The first CNN will be constructed from scratch without an existing base where different layering architectures will be built and tested. The second model will make use of

transfer learning to utilize the learned knowledge of existing pre-trained models to improve our own. There are a number of different pre-trained models available for the application of transfer learning. The pre-trained model to be used is the DenseNet121 model.

The second model will utilize transfer learning to improve its classification performance. The pre-trained model DenseNet121 is a CNN model trained on the ImageNet data set, which is a large dataset containing over 1.28 million images across 1000 classes (Huang et al., 2017). DenseNet121 is a rather small model compared to other transfer models but still has 121 layers. This pre-trained model is chosen over other bigger models due to computational reasons but still offers good results. On top of the pre-trained model, one fully connected layer has been added with 512 units and one activation layer that uses sigmoid as activation function. A dropout rate of 0.35 and batch normalization have been added in between these layers with ReLU as activation. The total number of trainable parameters for this model is 7 480 193.

The self trained model which is seen in Figure 9 in the Appendix is made up of eight convolutional layers, with regularization techniques such as dropout, weight decay, batch normalization, and early stopping. The model consists of 4 853 569 parameters which is less compared to the transfer learning model making it less expensive to train. For both models, early stopping has been used. The patience of the early stopping was set to two and is based off the validation loss at each epoch. That is, if there is no improvement in validation loss for two consecutive epochs, the model will stop running and save the weights for the best-performing epoch. We used ADAM as our optimizer with a learning rate scheduler that started at 0.001 for the transfer learning model, which is the standard learning rate for the chosen optimizer. The learning rate scheduler was used in order to avoid overfitting, with a gamma parameter of 0.1, which reduced the learning rate by a factor of 10 every 5 epochs. We also added weight decay for both models in order to further combat overfitting. This penalizes large weights in the model, making it more generalizable when predicting unseen data, and not over reliant on specific weight values. The loss function used is binary cross entropy loss, which is appropriate when you have a binary classification task, such as in our case. Our models final layer applies a Sigmoid activation function, which outputs probabilities between 0 and 1 for each sample. The loss function is designed to measure the difference between the predicted probabilities and true labels.

To tune important hyperparameters in our own self built model we have used random search. Random search allows us to look through a large parameter space without having to try every possible combination of hyperparameters, as in grid search. The hyperparameters and their distributions are presented in Table 1. Due to computational limitations, we only ran the random search models on 10 % of the training data. This gives a reasonable amount of data to find good hyper-parameters but will not be so large that the time it takes to train is unreasonable. We ran five different models until they triggered early stopping. The parameters we found using random search were used as guidance when trying out different combinations of hyperparameters until we found the ones that performed the best on the validation data.

Table 1. Hyperparameter Distributions and Best Values from Random Search

Hyperparameter	Distribution/Values	Value Used
Max Learning Rate	U: $[10^{-5}, 10^{-3}]$	0.001
Weight Decay	U: $[0, 10^{-2}]$	0.0007
Dropout Rate	U: $[0.1, 0.5]$	0.35
Batch Size	Discrete: $\{32, 64, 128\}$	128

2.3. Evaluation

To train and evaluate the model, the data is split into a separate training, validation, and test data set. The training data set is used to train the model in order to learn patterns and relationships in the different types of images and hence update the weights accordingly. 74% of the data will be used for the training. To monitor the performance of the model during training, the trained model is tested on the validation set after each epoch by calculating the current loss and accuracy. 20 % of the dataset will be used for the validation set. The main evaluation tool for the model during training is the validation loss, which the early stopping is based upon and eventually triggered by. We also measure the accuracy, which is the main comparison tool we use to infer what model performs the best. The validation loss is an estimate of the generalization error, which represents how well the model can predict unseen data.

The test set is used to assess the final model's predictive performance on completely new unseen data. The remaining 6 % of the data will be used for testing of the models. The accuracy will be calculated and used as the main evaluation tool. Sensitivity will calculated to see the fraction of real images was correctly classified. Specificity is also calculated in order to see the fraction of the fake pictures was correctly classified. These metrics are important to assess since it might be more important to correctly classify fake images rather than real images or the other way around.

3. Results

The following sections present the results from the training and testing process for each of the two CNN models.

3.1. Self built CNN

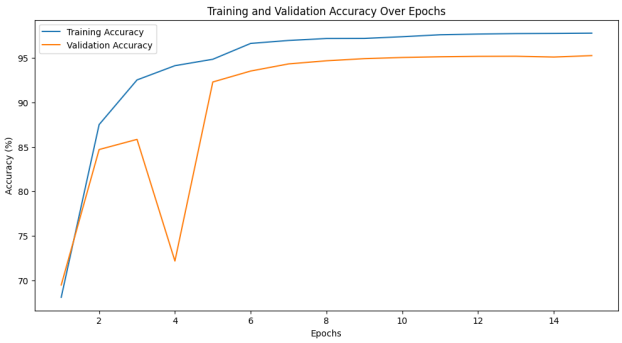

Figure 1. Training and validation accuracy over epochs for own model

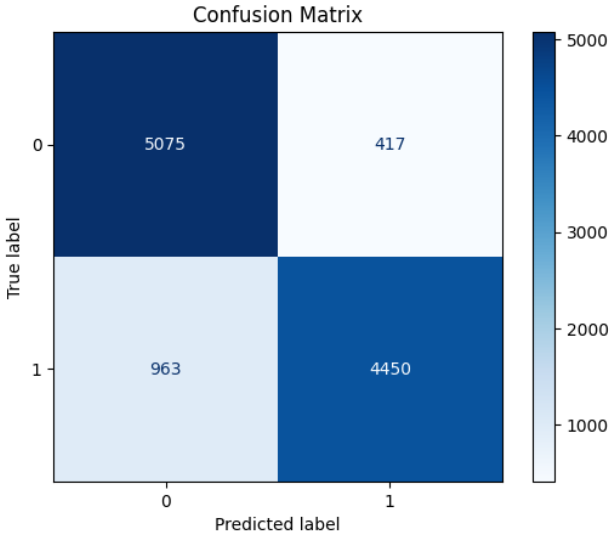

Figure 2. Confusion matrix of predictions in the test set for own model, 0 = fake, 1 = real

Figure 1 illustrates the change in training and validation accuracy during the training process for the self built CNN model. By the end of the process, the training accuracy is at 97.76% whilst the validation accuracy is at 95.24%. These results indicate that the model has learned the patterns in the data, as seen by the high training accuracy, whilst at the same time retaining generalizability in its classifications, as seen by the high validation accuracy. The small gap between the training and validation accuracy suggests that the model has not been significantly over-fit.

After testing, the first CNN model received a test accuracy of 87.36 % as calculated from figure 2 and seen in table 2, which is noticeably lower than that of the accuracy received in the training and validation process. This result suggests that the model struggles to generalize to unseen data. This could be caused by overfitting to the validation data when tuning the hyperparameters in the random search.

The sensitivity of the model was 0.8226, meaning 82.26 % of the actual real photos were identified by the model as real. The accuracy was, as previously mentioned 87.36 %. The model got a specificity of 0.9239, meaning it correctly classified 92.39 % of the fake images. The model got a rather high specificity, indicating that it is effective at correctly classifying fake images. The sensitivity is quite low compared to the other metrics, which means that the model may have issues correctly identifying the real images at an acceptable rate. Regarding the specific task of identifying deep-faked images, one could argue that having good specificity is preferred, as you would want to be able to correctly classify as many deep-faked images as possible. That way, you may sacrifice some of the sensitivity but will guarantee a model that does not let deepfaked images slip through unnoticed.

model	Accuracy	Sensitivity	Specificity
Own Model	0.8736	0.8226	0.9239
Transfer learning	0.9026	0.8732	0.9315

Table 2. Performance Metrics for the two CNN models

3.2. Transfer learning CNN

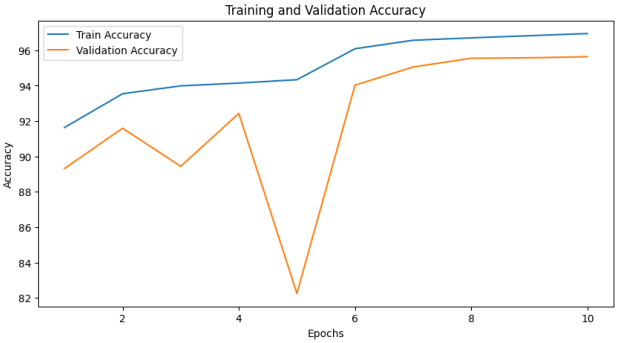

Figure 3. Training and validation accuracy over epochs for the transfer learning model

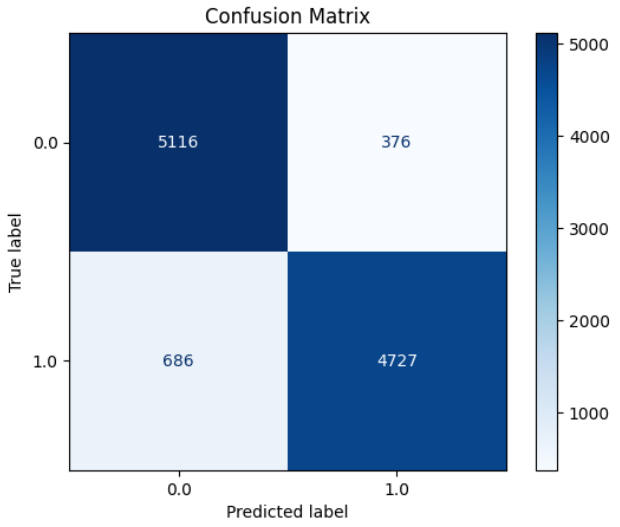

Figure 4. Confusion matrix of predictions in the test set for transfer model, 0 = fake, 1 = real

Figure 3 illustrates the change in training and validation accuracy during the training process for the transfer learning CNN model. By the end of the process, the training accuracy is at 96.95% whilst the validation accuracy is at 95.64%. . Similarly to the first model, these results suggest that the model has learned the patterns in the data, as seen by the high training accuracy, whilst at the same time retaining generalizability in its classifications. There is no sign of noteworthy overfitting in this case either, as the gap between the training and validation accuracy is small.

Figure 4 and Table 2 contains the results of the test performance for the transfer learning CNN model. As presented in the table, the transfer learning CNN achieved an accuracy of 0.9026, a sensitivity of 0.8732 and a specificity of 0.9315. All of these values are higher than those obtained from the test results of the first model. From these results, it is apparent that the transfer learning model has outperformed the self-built CNN by all observed metrics.

The difference in training and validation accuracy between the models was very small, indicating similar performance. However, the discrepancy in test performance would suggest that the transfer learning model is more robust and generalizes better when predicting new data. This could stem from its extensive training on the image net data set, which may cause it to find better and more general features, making it less prone to overfitting to the training data set.

4. Conclusion

In terms of performance, the results show that the transfer learning model outperformed the self trained model. This is an expected result, and the difference between self-built models and transfer learning will likely grow larger as the transfer learning models get bigger and more powerful. The enormous data sets these models are pretrained on makes them difficult to compete with for generic image classification tasks, such as classifying deepfakes of humans. Creating and training own models still has its place when it comes to more unique data, or when you want more control of the model architecture. However, the difference in performance was not that large and further research should be done comparing models which have been pretrained to ones that have not.

In our case, we had some computational issues in terms of time to train the models and possibility to appropriately tune the hyperparameters. If we had more time, or a faster GPU, we would have liked to run the random search on a larger portion of the data and with more models. This would make for a more robust tuning process and guarantee that we used appropriate hyperparameters for the task at hand. Given the time and resources at hand, the transfer learning models are to be preferred. However, due to the time and resource constraints we cannot make any conclusive remarks about the general performance of self built CNN models relative to transfer learning models outside of this specific use case.

Lastly, we would like to touch upon a number of ethical considerations related to our data. Since our dataset consists partly of pictures depicting real humans, there are certain ethical implications to consider regarding the use of the data for this type of analysis. One particularly important aspect is that of the privacy rights of the real individuals in the images. The ideal scenario would be if one could ensure that proper consent was given by the individuals in the images to be included in the dataset for the purpose of training and testing our model. In our case, we believe that the privacy of the individuals in the images is protected since all of the pictures are anonymized and cannot be traced back to any specific source. Another ethical cause for concern is the potential for our model to be repurposed for malicious uses, for example to develop more advanced and harder to detect deepfake images. To mitigate this issue one could limit public access to certain parts of the model or dataset.

References

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Westerlund, M. The emergence of deepfake technology: A review. *Technology innovation management review*, 9 (11), 2019.

5. Appendix

Figure 5. Examples of fake pictures

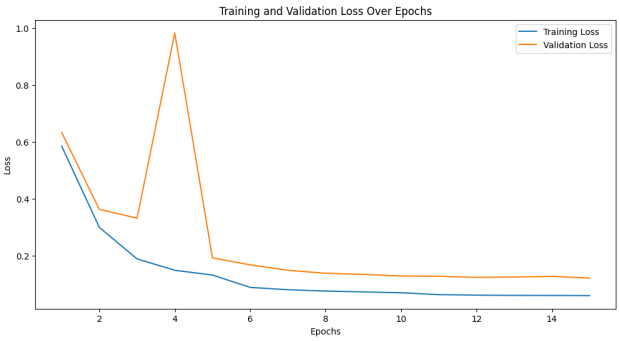

Figure 7. Training and validation loss over epochs for own model

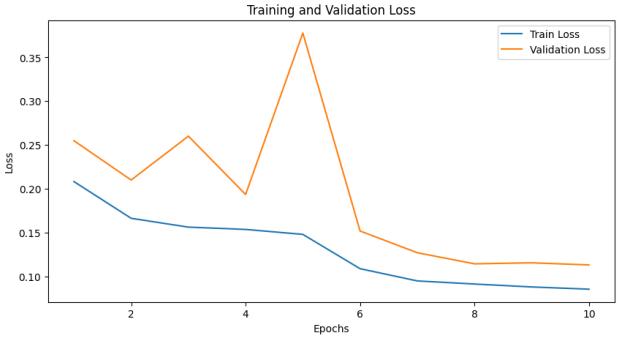

Figure 8. Training and validation loss over epochs for the transfer learning model

Figure 6. Examples of real pictures

Self built CNN

- **Convolutional layers:**

  Convolution layer 1 ($3 \rightarrow 64$, kernel= (3×3)) $\rightarrow$ BatchNorm $\rightarrow$ ReLU $\rightarrow$

  Convolution layer 2 ($64 \rightarrow 64$, kernel= (3×3)) $\rightarrow$ Batchnorm $\rightarrow$ ReLU $\rightarrow$ Maxpool (2×2, stride=2)

  Convolution layer 3 ($64 \rightarrow 128$, kernel= (3×3)) $\rightarrow$ Batchnorm $\rightarrow$ ReLU $\rightarrow$

  Convolution layer 4($128 \rightarrow 128$, kernel= (3×3)) $\rightarrow$ Batchnorm $\rightarrow$ ReLU $\rightarrow$ Maxpool (2×2, stride=2)

  Convolution layer 5 ($128 \rightarrow 256$, kernel= (3×3)) $\rightarrow$ BatchNorm $\rightarrow$ ReLU $\rightarrow$

  Convolution layer 6 ($256 \rightarrow 256$, kernel= (3×3)) $\rightarrow$ Batchnorm $\rightarrow$ ReLU $\rightarrow$ Maxpool (2×2, stride=2)

  Convolution layer 7 ($256 \rightarrow 512$, kernel= (3×3)) $\rightarrow$ BatchNorm $\rightarrow$ ReLU $\rightarrow$

  Convolution layer 8 ($512 \rightarrow 512$, kernel= (3×3)) $\rightarrow$ Batchnorm $\rightarrow$ ReLU $\rightarrow$

  Average pooling (1×1)

- **classification layers:**

  Linear ($512 \rightarrow 256$) $\rightarrow$ ReLU $\rightarrow$ Dropout

  Linear ($256 \rightarrow 128$) $\rightarrow$ ReLU $\rightarrow$ Dropout

  Linear ($128 \rightarrow 1$) $\rightarrow$ Sigmoid activation

Figure 9. Self built Convolutional Neural Network