Real Vs. Fake Face Detection

Anas Puthawala, Jayaram Chandar

CS 577 | Gady Agam

Fall 2022

Note: Jayaram and Anas worked on the paper and presentation, Anas focused on implementing the Naive CNN, K-fold cross validation, and functions to get data and Jayaram focused on the Siamese network, including setup,building it out and evaluating it.

Problem Statement

In the current day and age, advanced AI technology has its fingertips in almost every facet of life – from facial recognition software to unlock our iPhones, to risk analysis & fraud prevention for equity providers. Not every application of AI in our everyday life is beneficial though. Deepfakes are a type of video that shows in extreme realism and detail a politician, celebrity, or any person in general speaking about or performing an act that they have not actually performed or said. These are called deepfakes because they apply complex deep learning algorithms in these videos to generate these fake videos of people. These are extremely dangerous and a threat to democracy because you can practically circulate false information that is extremely convincing and has the power to fool even the most powerful facial recognition softwares [1]. According to a statistic, it was also reported that the number of deepfakes circulating the web doubles every six months [2]. With this in mind, there needs to be the flip-side of the coin, some form of technology that is centered around detecting deepfakes and subsequently curbing the explosive spread of it in society. Beyond deepfakes, there exists other types of AI that aim to bypass / cheat facial recognition softwares by creating fake faces that look very similar to real faces that the facial recognition was trained on. The problem is such that there is an inadequate amount of technology that can counter the explosive spread of deepfakes, and fake facial curations, which can extend to be very damaging if used in incorrect manners.

Proposed Solution

For this task, we've decided to distill the main problem down to detecting whether images of faces are real or fake. At the heart of the problems we've mentioned in the previous section lies the fact that it's mostly faces that are generated using advanced AI techniques. It's those faces that are overlaid on top of existing faces that may or may not look similar to the person they are trying to imitate. If we can have a system that is able to recognize faces in a streaming video and save those faces, then we can apply our proposed system of detecting whether a face is real or fake. This will be a binary classification task, where our input data will be a bunch of images that are real and those that are fake and generated by AI.

Our proposed system relies on the structure of siamese neural networks. Siamese networks have also been shown[5] to be good at learning sophisticated feature encodings with low amounts of data. We believed we could apply the idea here as our dataset is only about 2000 images in total. Our premise was that facial features in a fake image are not symmetrical and are very wacky in some cases. This is especially seen in left-eye to right-eye dissimilarity and this idea has been tried before in this paper[4]. What we specifically wanted to try was a mix of both these papers. We borrowed the twin CNN architecture from [5] as it was shown to be very good with learning image features from one example of a class. We coupled this with the eye separating mechanism from [4].

In addition, to ensure validity of our results and have a comparison, we'll also be evaluating the performance of a naive CNN architecture for our task and computing metrics such as accuracy, balanced accuracy, and visualizing the performance of the model via a confusion matrix to get a better understanding of the model's performance when considering true positives, false positives, true negatives and false negatives. Lastly, we'll also evaluate the performance of a random classifier in a similar approach to have a sort of baseline that we'd want to beat for the results in our siamese network approach.

Data Description

The data was obtained through Kaggle's Real and Fake Face data & challenge as uploaded by CIPLAB @ Yonsei University [3]. The data consists of 960 color images of fake faces each of size 600x600 and 1081 color images of real faces of size 600x600. Additionally, the dataset consists of varying levels (in terms of difficulty) of fake faces. So for example, the easy ones are much easier to classify as fake, whereas the hard ones are harder to classify as fake:

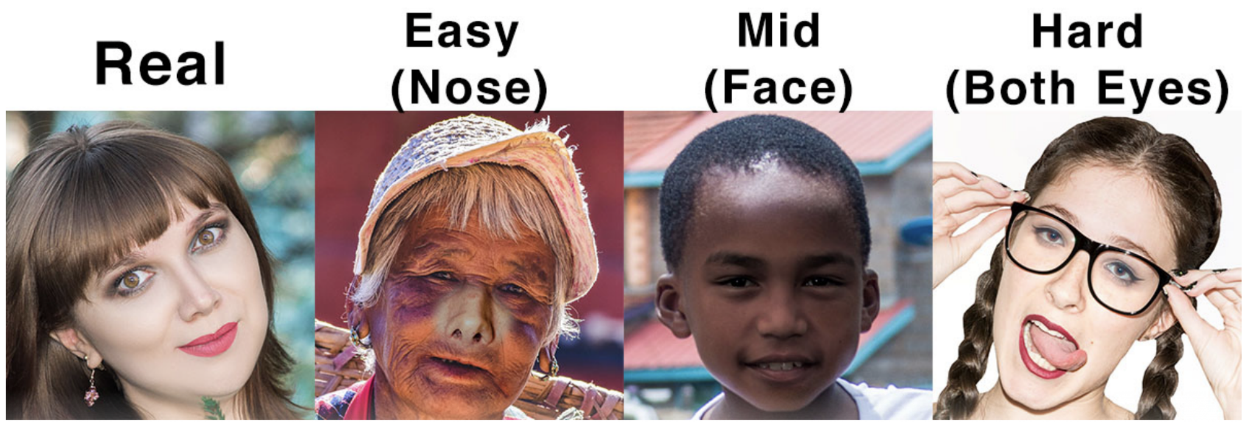


Fig. 1 - Data consisting of faces

Usually, the fake faces have something tampered with such as the nose looks off or the face is elongated in some manner. The data was shuffled to ensure our model randomly is exposed to all levels of difficulty when learning to classify fake faces.

Implementation Details

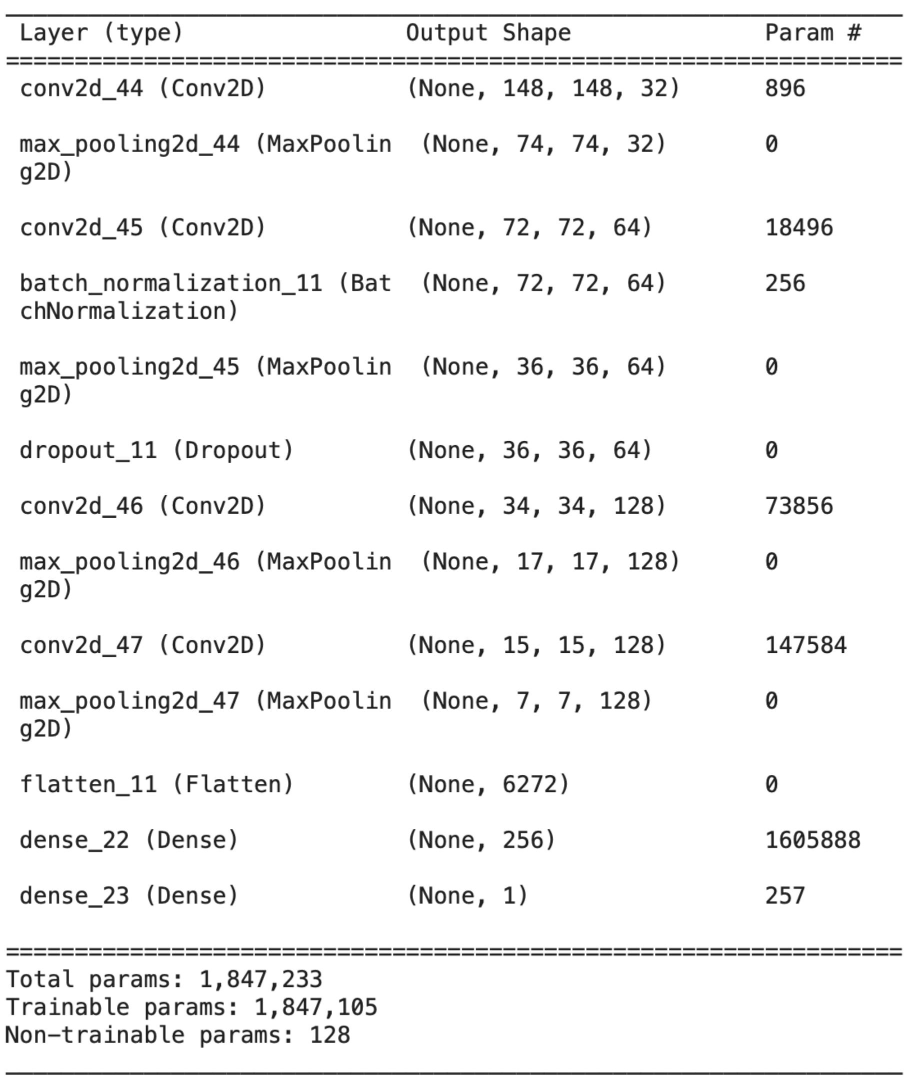
Acquiring a baseline

Random Classifier

A random classifier was utilized to acquire a baseline. To simulate the randomness, there were 100 trials run in which a numpy array was populated with guesses (0 and 1, where 0 is ‘Fake’ and 1 is ‘Real’). This would be the ‘y_preds’ and since we had the ‘y_true’ we could just go ahead and derive the balanced accuracy, accuracy, and visualize a confusion matrix.

Naive CNN

A CNN model was put together manually with several convolution layers, pooling layers, batch normalization, and flattened and passed through a dense network with a sigmoid activation in order to attain the binary classification mode. The Naive CNN model used had the following architecture:



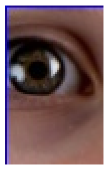
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_44 (Conv2D)          (None, 148, 148, 32)      896

 max_pooling2d_44 (MaxPoolin  (None, 74, 74, 32)       0
 g2D)

 conv2d_45 (Conv2D)          (None, 72, 72, 64)        18496

 batch_normalization_11 (Bat  (None, 72, 72, 64)       256
 chNormalization)

 max_pooling2d_45 (MaxPoolin  (None, 36, 36, 64)       0
 g2D)

 dropout_11 (Dropout)        (None, 36, 36, 64)        0

 conv2d_46 (Conv2D)          (None, 34, 34, 128)       73856

 max_pooling2d_46 (MaxPoolin  (None, 17, 17, 128)      0
 g2D)

 conv2d_47 (Conv2D)          (None, 15, 15, 128)       147584

 max_pooling2d_47 (MaxPoolin  (None, 7, 7, 128)        0
 g2D)

 flatten_11 (Flatten)        (None, 6272)              0

 dense_22 (Dense)            (None, 256)               1605888

 dense_23 (Dense)            (None, 1)                 257

=================================================================
Total params: 1,847,233
Trainable params: 1,847,105
Non-trainable params: 128
_________________________________________________________________
```

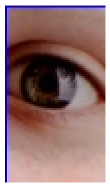
Fig. 2 - Naive CNN architecture

Additional care was taken to tune the hyperparameters such as epochs, LR, batch size etc. in order to attain the optimal performance for this model. After tuning, the model predicts on the testing data and we obtain our ‘y_preds_naivecnn’. We use this along with our known labels for our testing data to derive the accuracy, balanced accuracy, and visualize a confusion matrix to understand the model’s performance when it comes to true positives, true negatives, false positives, and false negatives. Due to the lack of data size, we also implemented k-fold cross validation to ensure an optimal reading of the accuracy and performance.

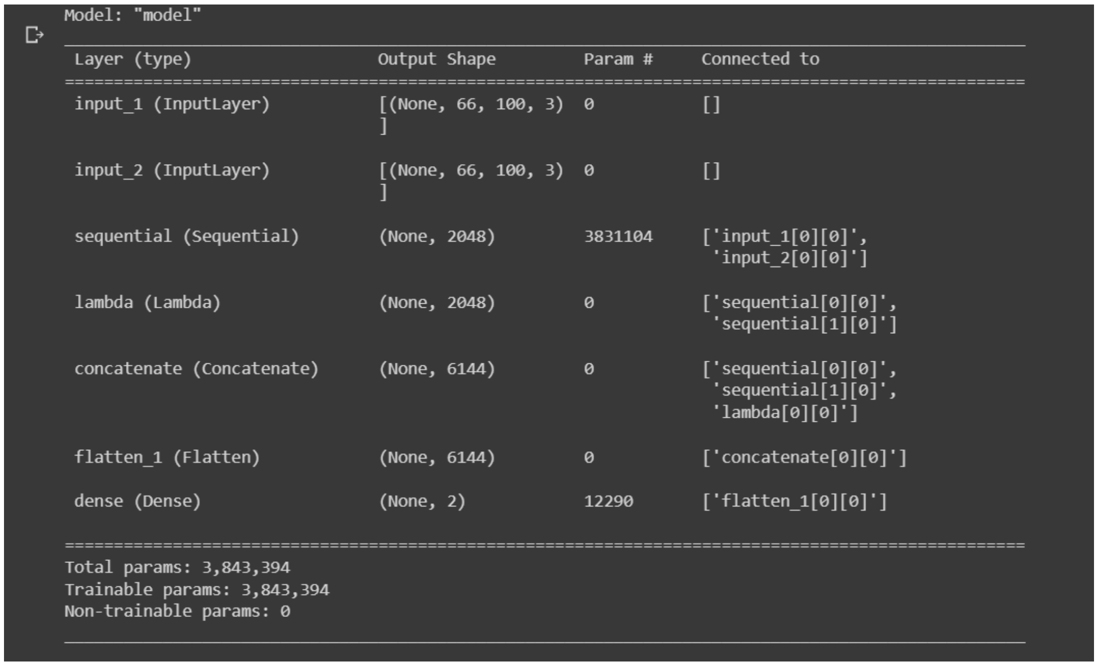
Siamese network

The siamese network was created based on two reference papers[4][5] as shown above. We needed to extract our regions of interest from the pictures. These regions of interest were the left eye and the right eye respectively. We did this preprocessing using the dlib module for both face detection and left eye and right eye detection. We also used a specific model to detect eyes as the default detector was having a significant number of false positives. We also tried the Haar Based classifier from opencv but that also suffered from the same problem. So we found this model[6] from an individual contributor. We used the default detector for the face detection and with these settings. For each image in our dataset, we created a left_eye.jpg and a right_eye.jpg which were just the bounding boxes from the image showing the eye. This model suffered from false negatives and it was not able to locate the eyes in 92 of the images. This was also partly due to the default face detector classifying random facial features as a face and the eye detector trying to detect eyes within it. This was still better than the default eye detection so we took the 92 image loss and let it slide. The images were resized to 66x100x3 due to the bounding boxes being rectangles naturally.


left eye


right eye

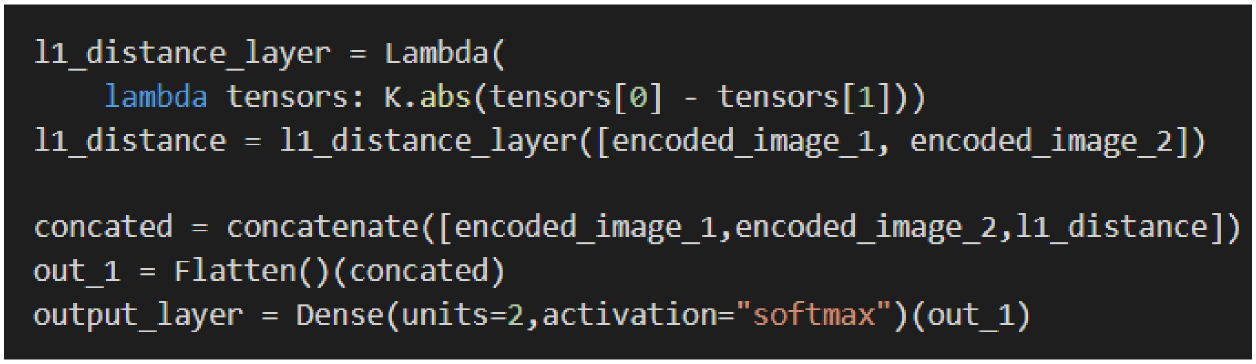
The next part was to build a data generator for the siamese networks as they needed a tuple of input tensors of the left eye and the right eye respectively. We created a simple custom data generator inherited from the keras sequence module.
The model architecture was as stated above the following.



```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 66, 100, 3)  0           []
                                ]

 input_2 (InputLayer)           [(None, 66, 100, 3)  0           []
                                ]

 sequential (Sequential)        (None, 2048)         3831104     ['input_1[0][0]',
                                                                  'input_2[0][0]']

 lambda (Lambda)                (None, 2048)         0           ['sequential[0][0]',
                                                                  'sequential[1][0]']

 concatenate (Concatenate)      (None, 6144)         0           ['sequential[0][0]',
                                                                  'sequential[1][0]',
                                                                  'lambda[0][0]']

 flatten_1 (Flatten)            (None, 6144)         0           ['concatenate[0][0]']

 dense (Dense)                  (None, 2)            12290       ['flatten_1[0][0]']

==================================================================================================
Total params: 3,843,394
Trainable params: 3,843,394
Non-trainable params: 0
__________________________________________________________________________________________________
```

Fig. 3 - Final dense layer of the siamese network

To elaborate more, the siamese network has two CNNs from [4] and the encoding similarity is measured by L1 distance and this distance was concatenated along with the encodings into a dense layer as shown by this line of code.



```
l1_distance_layer = Lambda(
    lambda tensors: K.abs(tensors[0] - tensors[1]))
l1_distance = l1_distance_layer([encoded_image_1, encoded_image_2])

concated = concatenate([encoded_image_1,encoded_image_2,l1_distance])
out_1 = Flatten()(concated)
output_layer = Dense(units=2,activation="softmax")(out_1)
```

Fig. 4 - Final dense layer of the siamese network

The final output layer just had the 2 units and softmax to do the final binary classification (same as one unit with sigmoid). The loss was binary cross entropy with the adam optimizer. We tuned the hyperparameters and removed the dropout layer and the kernel regularizers from the papers (the papers used much bigger datasets so overfitting was an issue for them). The adam optimizer was run with a learning rate of 1e-4. It was run with batch size of 16 for 10 epochs.

Results & Discussion

Baseline results

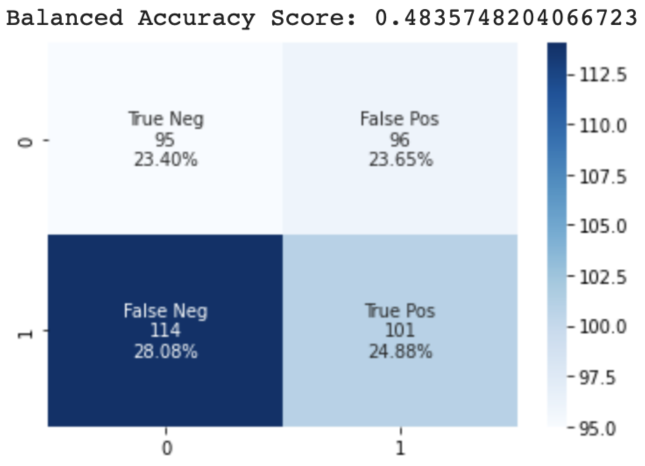


Fig. 5 - Random Classifier

As expected, the random classifier should be giving balanced accuracy of around .50.

Naive CNN network results

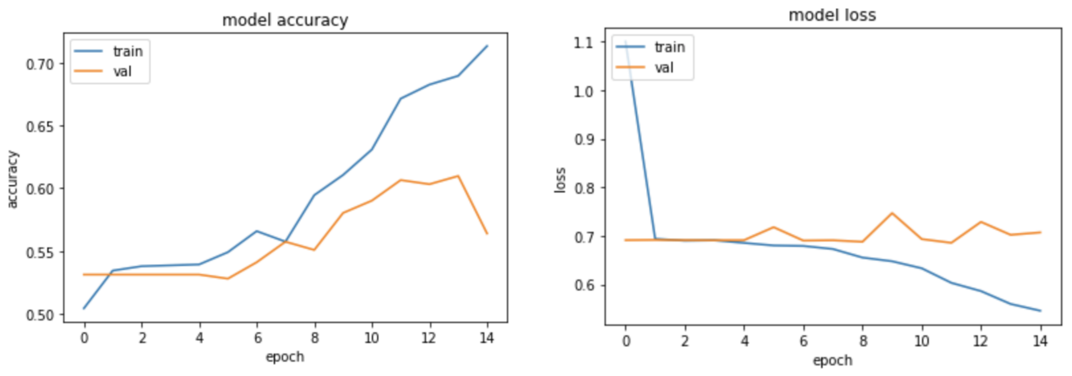


Fig. 6 - Naive CNN initial loss curves

When evaluating the performance of this model on the test set we received an accuracy of .57 and a loss of .74. We can see that the model is overfitting. Re-training with hyperparameter optimization (and only for around 10 epochs) results in the following:

Balanced Accuracy Score: 0.5100815779861196

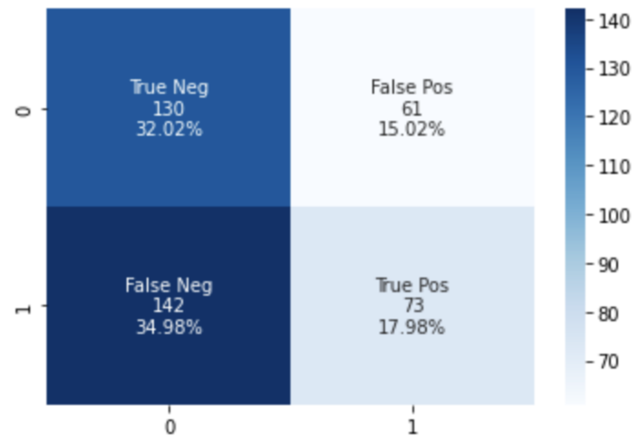


Fig. 6 - Naive CNN initial loss curves

The figure above shows the balanced accuracy of the Naive CNN model. The actual accuracy after training on the training + validation set and evaluating solely on the test set with batch size of 4, 10 epochs, and a LR of .00085 (with Adam optimizer) was ~55.9% – just above baseline (random guessing). The balanced accuracy is slightly more representative of the actual performance as it considers the true positives, negatives, false positives and negatives (specificity and sensitivity of the model). Note the high amount of negatives (true negatives and false negatives). The model, a majority of the time (~67% of the time), is predicting 0s – i.e. that the faces are fake. There's only a small percentage of positives being predicted that the faces are real. The K-fold mean accuracy over k=4 folds is .542 and the mean loss is .689

Siamese network results

Balanced Accuracy Score: 0.5565476190476191

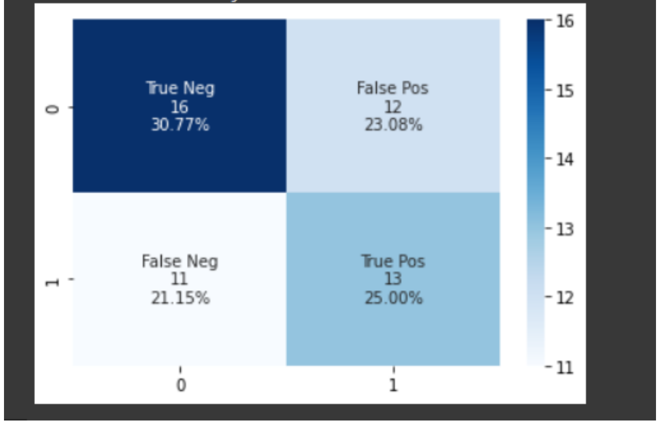
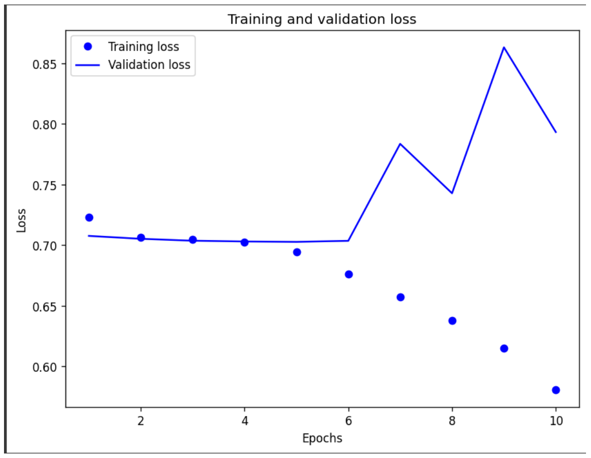


Fig. 7 - Siamese Network Results



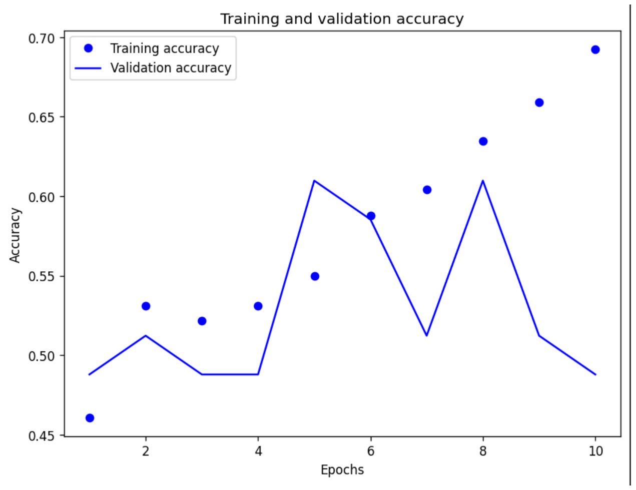


Fig. 8 - Siamese Network Loss Curve Results

Note that we don't have as much usable data as from the other model because the preprocessing pipeline did not work on 92 images. The Siamese network testing and balanced accuracy stayed pretty consistent at approximately 55%. We ran multiple rounds of testing for both CNN and the siamese network. From the confusion matrix, we can also see that the siamese network was better at picking up the fake images than the real images.   In most rounds, the siamese network did slightly better than the CNN on the balanced accuracy and the testing accuracy. A significant factor playing here is the lack of data to prove statistical significance. A future improvement we can try is to include more facial features as part of the input vectors and/or maybe using a pretrained model within the twin CNNs.

Human testing results

We also took 100 hard & medium images from the test set and tried to classify if they were fake or real and it was difficult to surpass ~60% accuracy. This demonstrates the caliber of this task.

Conclusion

Model performance improved (variably) after using siamese neural network for the task. The performance is still not as good as expected (very close to baseline guessing results in both cases). Future directions could be exploring various types of contrastive loss functions, aiming to figure out a methodology to be able to detect faces in real time video streaming and then applying the siamese network to classify whether the face is fake or real. It could be useful to use a pre-trained model for this task. Additionally, we primarily used the eyes as our feature in the faces to hone in on for the siamese network, adding the nose and other facial features as part of the feature vector could also provide valuable insight and results. Lastly, Data augmentation, different distance function for encoding distance, more fake data generation with GAN's and more real data in general can help with the performance.

References | Bibliography

[1] *Zurich.com*. [Online]. Available: https://www.zurich.com/en/media/magazine/2022/three-reasons-to-take-note-of-the-sinister-rise-of-deepfakes. [Accessed: 12-Nov-2022].

[2] “Report: Number of deepfakes double every six months | cybernews.” [Online]. Available: https://cybernews.com/privacy/report-number-of-expert-crafted-video-deepfakes-double-every-six-months/. [Accessed: 12-Nov-2022].

[3] C. I. P. L. A. B. @ Y. University, “Real and fake face detection,” *Kaggle*, 14-Jan-2019. [Online]. Available: https://www.kaggle.com/datasets/ciplab/real-and-fake-face-detection. [Accessed: 12-Nov-2022].

[4] J. Wang , B. Tondi, and M. Barni, “An eyes-based Siamese neural network for the detection of gan-generated face images,” *Frontiers*, 01-Jan-1AD. [Online]. Available: https://www.frontiersin.org/articles/10.3389/frsip.2022.918725/full. [Accessed: 15-Nov-2022].

[5] “Siamese neural networks for one-shot image recognition.” [Online]. Available: https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf. [Accessed: 16-Nov-2022].

[6] Davisking, “Dlib-models/shape_predictor_5_face_landmarks.dat.bz2 at master · Davisking/dlib-models,” *GitHub*. [Online]. Available: https://github.com/davisking/dlib-models/blob/master/shape_predictor_5_face_landmarks.dat.bz2. [Accessed: 16-Nov-2022].