May 3, 2022

Facial Recognition

Raeber, Chris
CS5390
University of Missouri Saint Louis

Abstract

As we can see from our day to day lives, biometric security and facial recognition is becoming increasingly popular. Whether it be to protect the information stored on your phone or computer or ensuring the protection of corporate and government secrets. Along with security, companies are beginning to utilize the value behind targeted advertisements for each of their consumers. This process is made easier through targeted product pitches based on user's social media pictures and accounts[1]. There are numerous approaches to biometric security, but this paper will focus on facial recognition software. The human face is the most important feature in the human body when it comes to manually identifying one another out of a group of individuals. It is because of this universality and uniqueness of human faces that facial recognition software has leapt to the front of biometric security innovations[2]. There are several fundamental processes that are needed for facial recognition: encoding of visual images into neural patterns or tensors, the detection of simple facial features like eyes, nose, mouth, cheekbones, the reduction of the neural patterns in dimensionality, and a correlation of the input image to the training data set[3]. This paper will attempt to show the use of a Deep Learning convolutional neural network to implement facial recognition software. As input, the model will be fed images of faces from training, validation, and test subsets. The model will then increase its ability to classify the training and validation subsets as either an image of me or of someone else. Then the model's accuracy will be checked on the test subset of never-before-seen images.

Contents

1 Links

Google Colab:
https://drive.google.com/drive/folders/1MAsh3Mas0DqoNhqqpEoY68FFj1yRJyPN?usp=sharing

Overleaf:
https://www.overleaf.com/read/xtsrzhpykwcc

2 Data

The data set for training is made up of 2052 images split with 725 images of me and 1327 images of other faces found on line or taken of my brother. The distribution can be seen in Figure 1

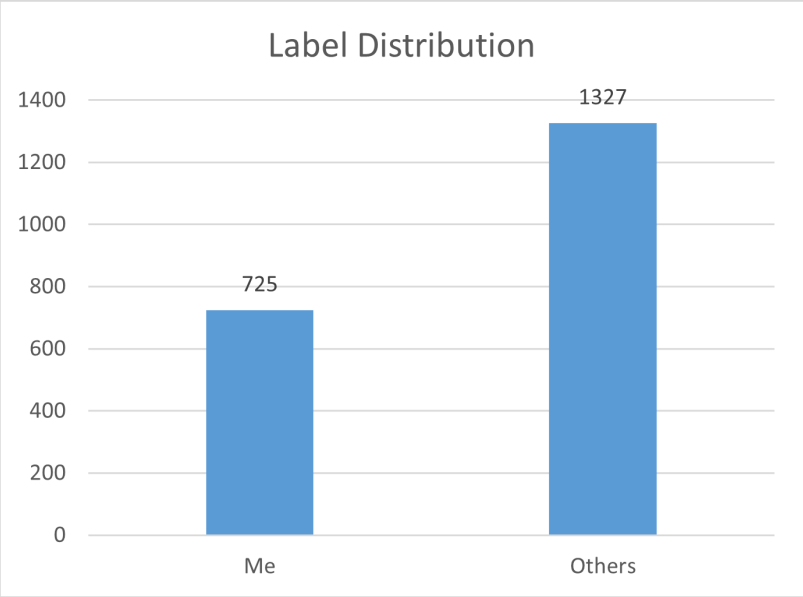

Figure 1:
Distribution of Output Values

2.1 Normalization

After the images had been uploaded to the workspace, they were resized to arrays of 256 x 256 representing RGB pixel intensities. These intensities were then rescaled so that they all fell within the range of 0 and 1. To do this, all intensities were taken and divided by the maximum allowed intensity of 255.

3 Model Overfitting

To determine the model for overfitting, a variety of convolutional network architectures were trained and evaluated. The goal of this training was to find a very accurate model that contained a low number of parameters. The initial model was created with a very large number of parameters. Subsequent models were then scaled down to include less parameters and then trained and evaluated. This process was repeated until a model was found with a loss greater than 2% and an accuracy lower than 98% indicating that the model was too small to overfit the data. Below are the resulting models, their evaluations, and the progress of their training.

3.1 Overfit Model Performance

Model	Parameters	Accuracy	Loss	Precision	F1
Model 0	64,545	92.323%	21.446%	100%	0.9
Model 1	17429	99.951%	0.192%	100%	0.9
Model 2	17,901	100.00%	0.008%	100%	1.0
Model 3	9,385	100.00%	0.075%	100%	1.0
Model 4	7,335	99.658%	1.363%	99%	0.9
Model 5	3,199	97.804%	2.549%	99%	0.9

Table 1: Performance of overfit models

From Table 1 it can be seen that models 2 and 3 had the best performance on the data so experiments in the next phase of the project will build upon these model architectures.

3.2 Labels as Additional Input

For this portion of the project, a relatively small model was created for testing. Next the output labels were added as additional input for the training data. This was then fed to the model for training. As expected, even a very small model was able to achieve high accuracy and low loss after training for just 8 epochs. Please visit the Google Colab link for additional information on the architecture and training of this model.

Model	Parameters	Accuracy	Loss	Precision	F1
Model 0	15,673	99.785%	0.708%	99.00%	0.9

Table 2: Performance of Model with Labels Included as Input

4 Model Performance On Split Data Set

4.1 Data Splitting

The overfit performance of Model 2 and 3 in Table 1 were shown to perform the best and had smaller architectures, implying that they are less likely to overfit to the data set. These model architectures were than utilized and slightly altered for testing on a split data set. The data set was split as:

70% Training – 20% Validation – 10% Testing

Additionally the baseline accuracies for the split data are:

Training 70.59% – Validation 70.00% – Testing 70.45%

The below Table 3 shows the model's performances when trained with the training and validation data then evaluated on the testing data. Overfit models 2 and 3 were trained as is and then slightly altered in one of two ways. The first was to utilize a layer of Dropout and Batch Normalization, the second was to add an additional Dense layer.

4.2 Model Evaluations

Model	Parameters	Dropout	Additional Dense	Accuracy	Loss	Precision	F1
Model 0	17,901	No	No	98.295%	11.118%	98.4%	.9879
Model 1	18,461	Yes	No	92.614%	31.257%	97.435%	.960
Model 2	118,717	No	Yes	97.727%	14.281%	96.875%	0.9841
Model 3	9,385	No	No	98.295%	4.751%	99.186%	0.9878
Model 4	9,945	Yes	No	96.023%	8.625%	96.062%	0.9721
Model 5	31,353	No	Yes	99.432%	1.004%	99.200%	0.9959

Table 3: Performance of Models on Test Data

From the above Table 3 we can see that model3 achieved the highest levels of generalization by performing best on the test data. This is seen though the models high accuracy and low loss. This model was then utilized and built upon in the later sections of this report.

5 Effects of Data Augmentation

5.1 Augmentation Techniques

The below Table 5 shows the model performance when trained with the training and validation data then evaluated on the testing data, while using various forms of data augmentation. Data Splitting model 3 from Section 4.2 was used for this testing as it had the highest accuracy with a low number of parameters. Next, various data augmentation layers were added to the model and were trained. The augmentation methods used were: random horizontal flip, random rotation of up to 180 degrees, random zoom up to 20%, and a random shift up, down, left, or right of up to 10%. An example of the Model 5 data augmentor is shown in Figure2

5.2 Augmentation Effects

Model	Flip	Rotation	Zoom	Translation	Accuracy	Loss	Precision	F1
Model 1	Yes	No	No	No	96.590%	15.034%	98.360%	0.956
Model 2	No	Yes	No	No	94.886%	11.408%	93.893%	0.9647
Model 3	No	No	Yes	No	99.432%	1.298%	100.00%	0.9959
Model 4	No	No	No	Yes	97.727%	10.161%	98.387%	0.9838
Model 5	Yes	Yes	Yes	Yes	93.750%	14.695%	93.798%	0.965

Table 4: Performance of Models on Test Data With Data Regularization

Figure 2: Visualization of Data Augmentation

From the above Table 5 we can see that model 3 which zooms the image has the lowest loss. This shows that the model is able to recognize images that are zoomed in just as well as the images that are not altered. We can also see that the least accurate model is the one that utilizes all four of the augmentation methods. However, the accuracy of this model is high enough to use going forward because it is mostly likely to be the least overfit. In comparison to the other training of previous models in this paper, the learning rates for the models that use data augmentation took longer to converge on a minimum val_loss most likely because they are trained on a wider variety of images.

6 Effects of Regularization Techniques

6.1 Regularization Techniques

The below Table 5 shows the model performance when trained with the training and validation data then evaluated on the testing data, while using various forms of data regularization. The architecture of model 3 from section 4.2 was used to build the various models in this stage. Also, the regularization technique from model 5 in section 5.2 was applied to the training data. Then, various forms of regularization were applied to this model architecture. The tested techniques were Batch Normalization, Dropout, L1 and L2 regularization. Also, the combinations of Batch Normalization and dropout as well as L1 and L2 were tested.

6.2 Model Performance with Regularization

Model	BatchNorm	Dropout	L1	L2	Accuracy	Loss	Precision	F1
Model 1	Yes	No	No	No	98.295%	7.446%	98.4%	0.9879
Model 2	No	Yes	No	No	98.886%	17.345%	95.275%	0.9641
Model 3	Yes	Yes	No	No	98.864%	2.005%	99.193%	0.9919
Model 4	No	No	Yes	No	94.886%	17.174%	94.573%	0.9644
Model 5	No	No	No	Yes	97.727%	9.114%	98.387%	0.9838
Model 6	No	No	Yes	Yes	97.727%	15381%	97.619%	0.9840

Table 5: Performance of Models on Test Data Utilizing Regularization

From the above Table 5, we can see that Model 3 which utilized both batch normalization and dropout has by far the lowest loss. Also, because this model has a very high accuracy in comparison to the others, it will be utilized and expanded upon going forward in this project. During the training process of these models it was found that often the validation loss and accuracy significantly differed from the training loss and accuracy. This showed the models tendency to overfit to the training data.

7 Pre-trained Models

The below Table 6 shows the model perfomance when trained with the training and validation data then evaluated on the testing data. Also, because early stopping was utilized in training, the number of training epochs and time per epochs are shown as well as the total training time per model.

7.1 Pre-trained Model Performance

Model	Epochs	Time per Epoch	Total Time	Accuracy	Loss	Precision	F1
ResNet	35	23s	13:25	91.477%	40.023%	93.600%	0.9397
DenseNet	96	26s	41:36	70.455%	9.169%	70.454%	0.8266
NASNet	65	24s	26:00	87.500%	24.750%	91.129%	0.9112
VGG16	84	26s	36:24	98.864%	6.709%	99.193%	0.9919
ResNet50	114	25s	47:30	96.591%	6.339%	95.384%	0.9763

Table 6: Performance of Pre-trained Models on Test Data

From the above Table 6 and the previous tables found in this paper, we can see that the model architectures of ResNet, DenseNet, and NASNet performed terribly on this data set. They have on average a significantly lower accuracy and substantially higher loss than all other models evaluated in this paper. The only bright spots of this testing round was the leveraging of the pretrained models VGG16 and ResNet50. These models were able to achieve relatively good results in comparison to models mentioned in previous sections of the paper. However, the training curves in Figure 3 are troubling and are discussed further in section 7.2.

7.2 Pre-trained Model Training

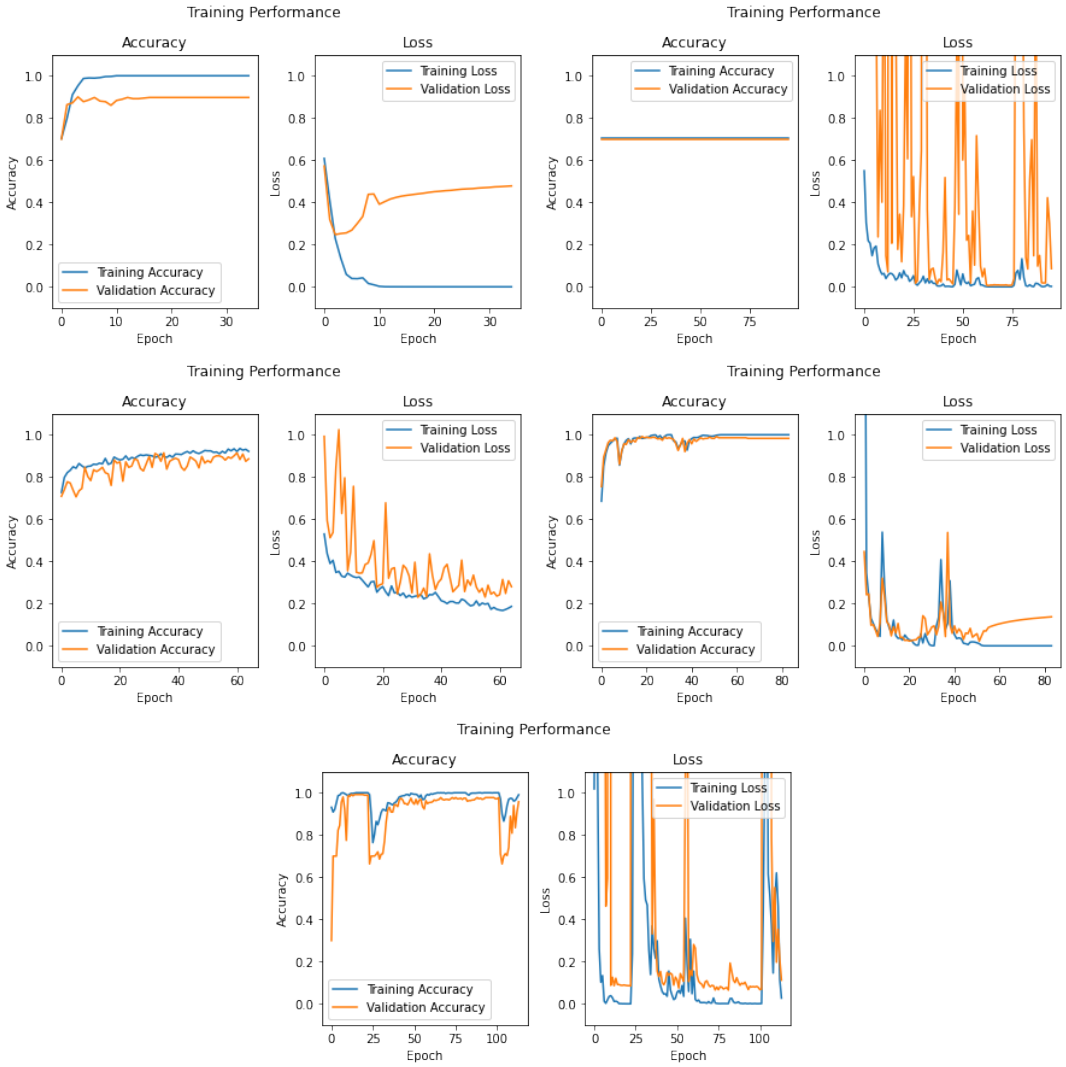

Figure 3:
From Left to Right: ResNet, DenseNet, NASNet, VGG16, ResNet50
Accuracy and Loss during training

In Figure 3 the various model accuracies and losses are tracked for the training and validation data throughout the training process. For training, early stopping and model callbacks were used to monitor validation loss and stop training if the validation loss of the model did not improve over the course of 32 epochs leading to different curves being plotted over different numbers of epochs for their x values. In comparison to the other training curves which can be found in the code submissions, this set of curves is highly irregular. First, the DenseNet curve shows an inability to improve beyond the baseline accuracy and the loss on the validation set is all over the place. This suggests that the model is failing to learn. Also in the ResNet50 curve we see an increase in validation loss at training continues suggesting overfitting. From Table 6 and Figure 3 we can see that the best performing model is VGG16.

8 Conclusion

The object of this project was to create a facial recognition Deep Learning convolutional network was met through the various phases discussed throughout the course of this paper. First, an initial model was created to overfit the entire dataset. Then the number of parameters in this model was reduced until the point where its accuracy suffered. Next, the model was trained on training and validation data and the accuracy was determined by its ability to predict the test data. This ensured that the model would not overfit. To counteract overfitting even further, data augmentation and regularization techniques were implemented. These made the model slightly less accurate but increased the generalization capacity of the network. Finally, pretrained models were used to help highlight the efficacy of the created network as solving this particular facial recognition problem. For future research, additional data should be added. This could be in the capacity of changing the problem to recognize multiple faces instead of just one. This would change the problem into a multiclass classification problem and would require the necessary code adjustments. Also, I was surprised by the accuracy of the facial recognition model throughout the course of this project. Even in the dataset I attempted to deceive the network with more challenging images. These came in the form of adding images of me wearing different hats, glasses and changing the angle of the image. Also, my brother, who shares a striking resemblance to me was able to provide similar images of himself to further trick the network. However, the accuracy remained very high even when applying these tricks. This highlights just how different human faces are from one another further justifying the use of facial recognition as a biometric security option.

References

[1] Steve Lohr. Facial recognition is accurate, if you're a white guy. In *Ethics of Data and Analytics*, pages 143–147. Auerbach Publications, 2018.

[2] Paramjit Kaur, Kewal Krishan, Suresh K Sharma, and Tanuj Kanchan. Facial-recognition algorithms: A literature review. *Medicine, Science and the Law*, 60(2):131–139, 2020.

[3] Robert J Baron. Mechanisms of human facial recognition. *International Journal of Man-Machine Studies*, 15(2):137–178, 1981.