Abstract

Image classification has been growing in popularity and importance, yet the models behind them, convolutional neural networks (CNN), are extremely complex. The dog and cats dataset from Kaggle is used to explore the architecture of a CNN, focusing on the number of layers and size of the filters in each layer. A dense network is also built to serve as a baseline as well to further illustrate how well a CNN performs on image classification tasks. Topics such as overfitting and data augmentation are addressed and commentary is provided on what CNNs are detecting. The challenges and insights gained from dogs and cats are generalized to a future hypothetical facial recognition project.

Introduction

Image classification has vast applications in the real world, ranging from digit identification and facial recognition to natural language processing. By testing different convolutional neural network (CNN) architectures, a greater understanding of their inner workings can be achieved. This assignment serves as a foundational effort to CNNs and explores key concepts and challenges when building this kind of model. The cats and dogs data set from Kaggle is used for this study. There are primary goals in this assignment:

1. Compare and contrast model performance of a dense fully-connected network (DNN) and a CNN using two hidden layers.
2. Test the performance of different CNN architectures, focusing on number of convolutional/max pooling layers and the number of nodes, or filters, in each layer. Specifically, a flat versus pyramid structure is tested.
3. Visualize what a CNN is actually doing by displaying the feature maps from each layer.

Literature Review

The Deep Learning with Python textbook by Chollet served as the primary source of information for this assignment. This textbook provided an overview of CNN models as well as code for basic CNN models, the use of data augmentation, and support on visualizing feature maps. One key point of CNN models is the convolutional layer: "Dense layers learn global patterns in their input feature space (i.e., all pixels), whereas convolutional layers learn local patterns: in the case of images, patterns found in small 2D windows of the inputs" (Chollet, p. 122). Because dense and convolutional layers learn differently, testing the performance of a model comprised of only dense layers versus a model with convolutional layers is a focus. Moreover, as it relates to feature maps, Chollet states: "As you go higher, the activations become increasingly abstract and less visually interpretable….[higher presentations carry] increasingly more information related to the class of an image" (p. 166). This statement will be explored later in the assignment but is a key concept and further supports Chollet's statement about CNNs acting as an "information distillation pipeline" (p. 166).

The Geron textbook provided general background on CNN architectures, specifically on the differences between the convolutional and pooling layers, which aided in overall understanding. Moreover, as it will be made clear in subsequent sections, one challenge with this assignment was memory. As Geron states, "Another problem with CNNs is that the convolutional layers require a huge amount of RAM, especially during training" (p. 370). Therefore, for future cases, building in ample time for training and leverage a server for coding is encouraged.

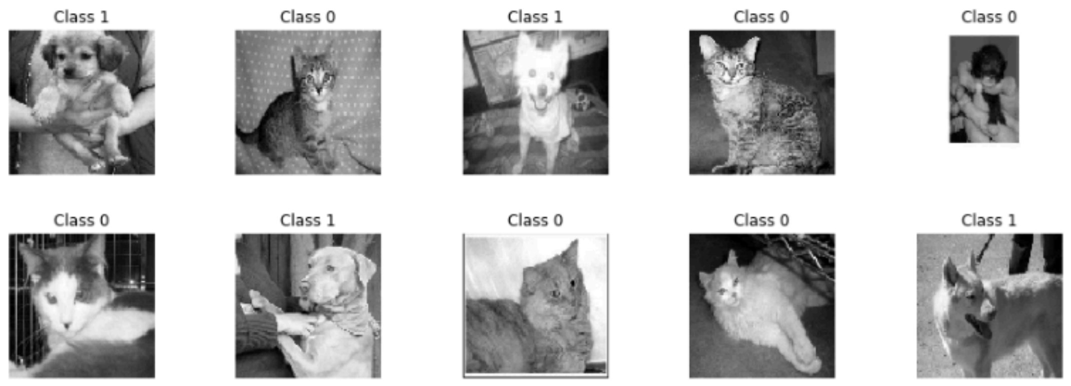
Geron also helped to provide perspective on overfitting techniques, as this is one pitfall of any deep neural network. Dropout is the primary overfitting technique used in this assignment, which happens to be one of the most popular regularization techniques (Geron, p. 311).

Lastly, Bengio provided guidance on the testing framework through his comments on the size of the hidden layers. He states, "we found that using the same size for all layers worked generally better or the same as using a decreasing size (pyramid-like) or increasing size (upside down pyramid), but of course this may be data dependent" (p. 11). As such, for this dogs and cats data set, comparing and contrasting the performance of a flat and pyramid-like structure became a topic of interest.

Methods

Data Preparation

There were several different data preparation steps that were performed in order to prepare the data for modeling. First, the dogs and cats training data set was downloaded from Kaggle and then imported. As part of the import process, the images were resized to a more manageable size – 96 x 96 pixels. Furthermore, the pixel values were standardized to be between [0,1] by dividing by 255. The labels for training were also created using the file names – dogs are encoded as a 1 and cats as a 0. One important decision was to read in the images as black and white. This decision helped to significantly speed up training time. If more time was allowed or different resources were given, then using color images would have been the obvious choice. Below is a snapshot of the first ten images.



Once the images were imported and processed, 5000 random images were chosen, forming the entire data set. From this subset, the data was broken into training, validation, and testing sets comprised of 3000, 1000, and 1000 images respectively. During the splitting process, work was done to ensure the dog and cat images were balanced so as not to create a biased model in the end.

Because DNNs and CNNs require a slightly different format for data, resizing/reshaping was performed to ensure that the data would work for both types of models.

DNN and CNN Approach

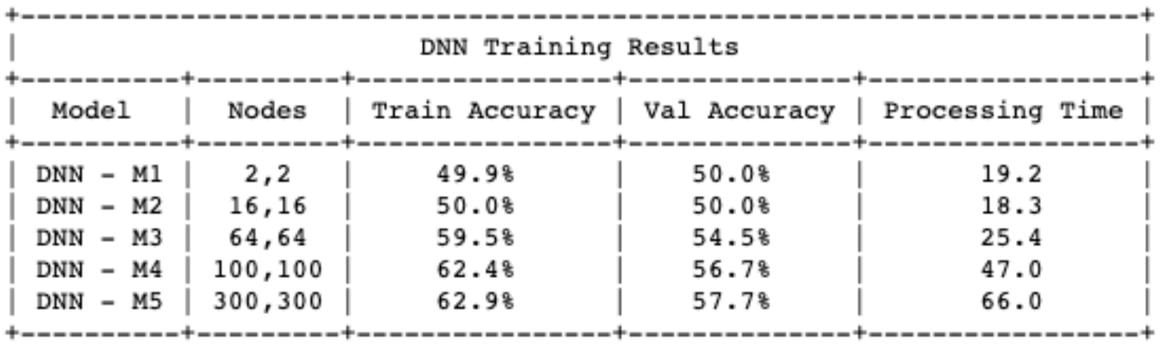
Both DNNs and CNNs utilized the same hyperparameter settings:

- Adam as the optimizer with a learning rate of .001
- Binary crossentropy as the loss
- RELU for the activation
- Sigmoid for the output, since this problem is a binary classification problem
- 50 for the number of epochs, to ensure that overfitting was observed
- 512 for the batch size, as this proved to be the batch size with the fastest processing times
- Accuracy as the primary performance metric, although loss and accuracy curves were generated for all models

One important item to mention is that memory issues were encountered throughout this assignment. Therefore, some training time was cut short.

DNNs

Only two hidden layers were considered in this assignment for the DNN design. The table below summarizes the training set results on all five DNN models.

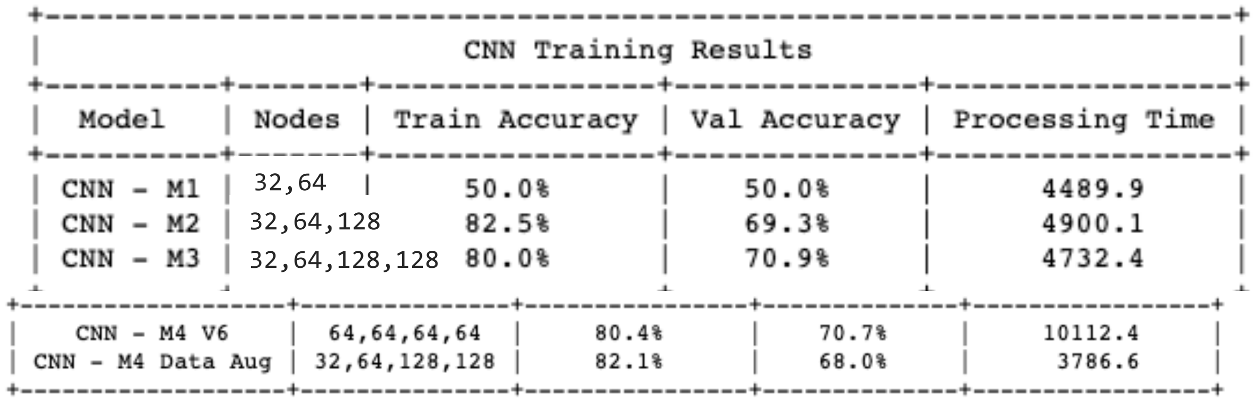


```
+-----------------------------------------------------------------------+
|                          DNN Training Results                         |
+----------+---------+----------------+--------------+-----------------+
|  Model   |  Nodes  | Train Accuracy | Val Accuracy | Processing Time |
+----------+---------+----------------+--------------+-----------------+
| DNN - M1 |   2,2   |     49.9%      |    50.0%     |      19.2       |
| DNN - M2 |  16,16  |     50.0%      |    50.0%     |      18.3       |
| DNN - M3 |  64,64  |     59.5%      |    54.5%     |      25.4       |
| DNN - M4 | 100,100 |     62.4%      |    56.7%     |      47.0       |
| DNN - M5 | 300,300 |     62.9%      |    57.7%     |      66.0       |
+----------+---------+----------------+--------------+-----------------+
```

While the processing time was very short for all of these models, as most finished in under a minute, the performance is sub-par. Model five, with 300 nodes in each layer, achieved a 57.7% accuracy on the validation and will be ran against the test set just to serve as a baseline for the CNN models.

CNNs

Since one of the primary goals of this assignment is to test different CNN architectures and to assess the performance of a flat vs. pyramid-structure design, several different CNN models were generated. In all models, a single dropout layer (with size = .20) was used as the main technique to protect against overfitting. Below is a table that summarizes the training set results from select CNN models, where nodes represents the size of the convolutional layer filter. Because memory issues were experienced, this table had to be stitched together through different kernel sessions.

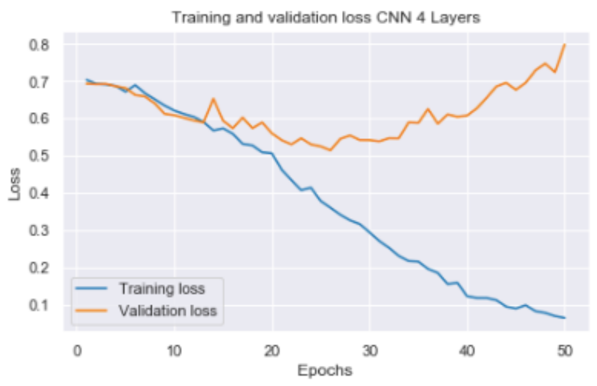


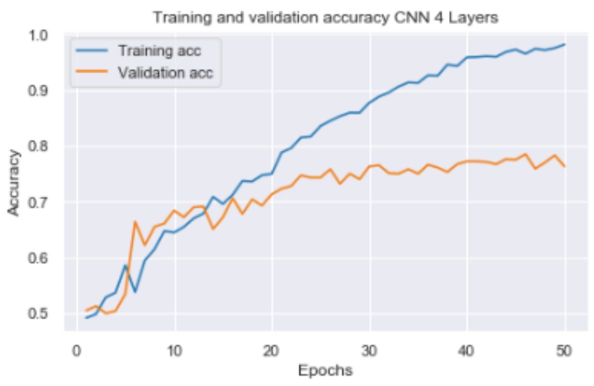
```
+--------------------------------------------------------------------------+
|                          CNN Training Results                            |
+----------+-------+----------------+--------------+-----------------+
|  Model   | Nodes | Train Accuracy | Val Accuracy | Processing Time |
+----------+-------+----------------+--------------+-----------------+
| CNN - M1 | 32,64   |     50.0%      |    50.0%     |     4489.9      |
| CNN - M2 | 32,64,128      82.5%      |    69.3%     |     4900.1      |
| CNN - M3 | 32,64,128,128  80.0%      |    70.9%     |     4732.4      |
+-----------------+---------------+---------------+-------------+----------------+
|    CNN - M4 V6    |  64,64,64,64  |     80.4%     |    70.7%    |    10112.4     |
| CNN - M4 Data Aug | 32,64,128,128 |     82.1%     |    68.0%    |     3786.6     |
+-----------------+---------------+---------------+-------------+----------------+
```

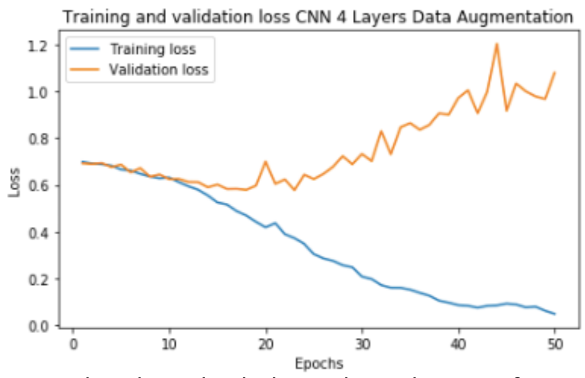
Compared to the DNN models, it is clear that CNN performance is better, since validation accuracy is slightly above 70%. Still, the two-layer CNN model had terrible performance, suggesting that a more complex model is needed to fully distill patterns from the images. Despite its increased performance, processing time is much more significant for CNN models and overfitting still seems to be a problem, as there is wider separation between training and validation accuracy. Moreover, the results for a pyramid structure (M3) compared to a flat model (M4 V6) are very comparable, but the flat model took 2.5 times longer to run than its pyramid counterpart. Therefore, a flat structure does not seem to be worth it in the end for this data set.

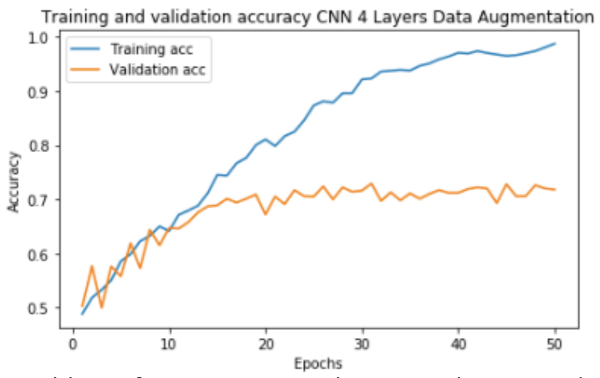
Even though CNN model performance is better compared to the DNN models, the performance in general is not that great. Therefore, data augmentation was applied to model three to see if injecting greater variability into the training images would improve validation set results. Data augmentation works by randomly transforming images based on user-defined parameters, impacting characters such as rotation angle and zoom for instance. It seems like data augmentation helped to boost training set performance but had an adverse effect on the validation set. Still, this model took less time to run. Below are the loss and accuracy curves for model three followed by the data augmentation model.

From the loss and accuracy plots below, both models have very similar performance, which is surprising. It was hypothesized that data augmentation would help boost performance and protect against overfitting. It seems that more severe data augmentation parameters are needed for it to have the desired effect.





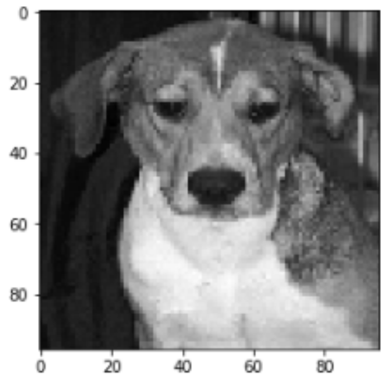




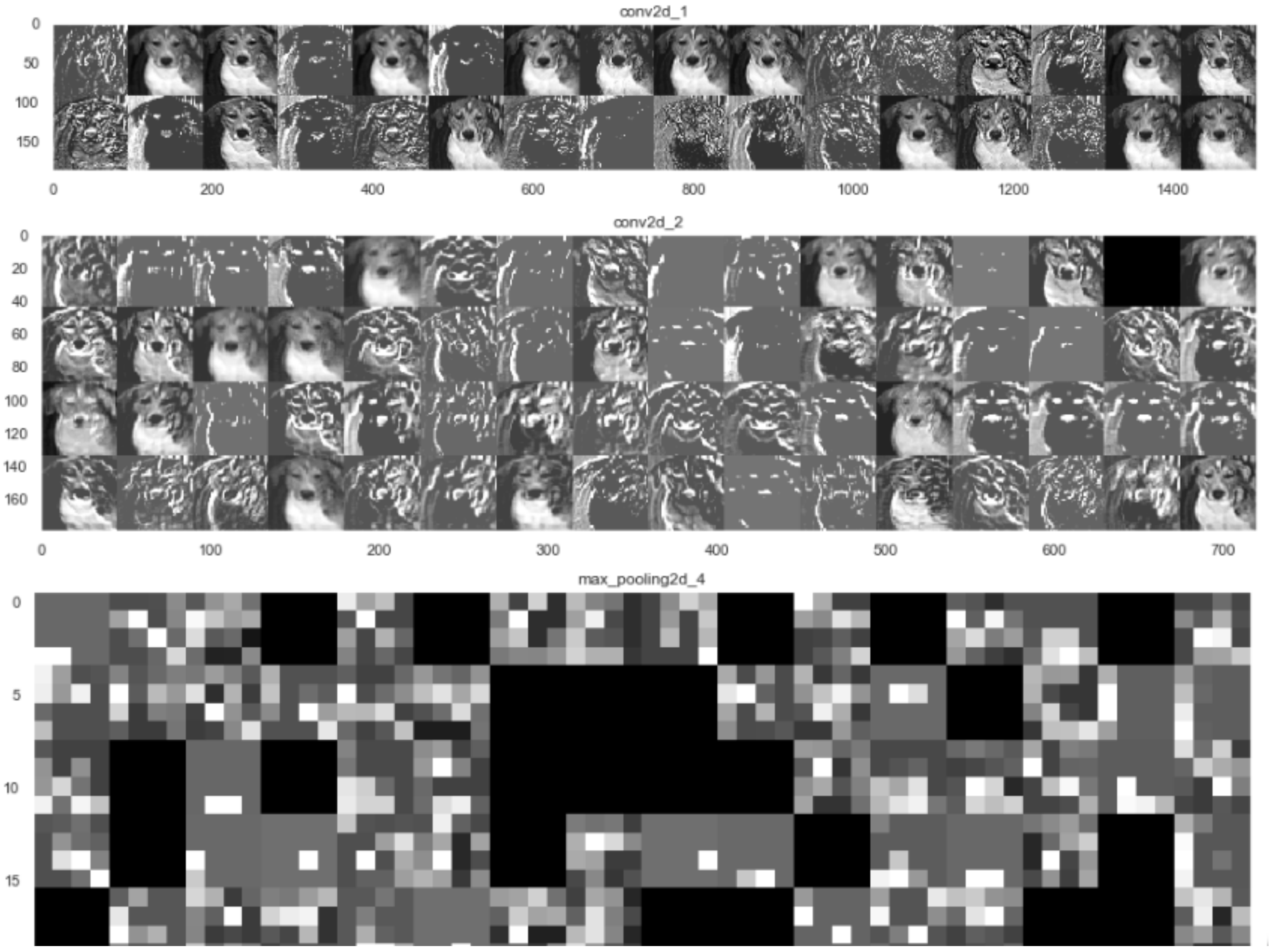
The plots also help to show that overfitting is a problem after approximately 20 epochs. In an ideal world, the models would be refit using fewer epochs, but time became a constraint and so no refitting was able to be performed. Additionally, if more time allowed, adding in L1/L2 regularization hyperparameters might have helped with overfitting as well.

Visualizing CNN Output

As a final step in the model building process, the output, or activation, of each layer in a CNN model is explored in greater detail. This exercise will utilize the model with four layers of a pyramid structure (32,64,128,128), since that model had the best training and validation set performance. Below is the sample image that will be used to display the filters from this model.



Because the heart of CNN models are the convolutional layer, which decompose an image into smaller feature maps, it is a natural progression to try and interpret what the resulting feature maps are actually doing through each layer in the model. Based on the architecture of this model, there are eight layers that could be analyzed. However, only select output from three layers is used to show the progression of what the model is doing. Below, the first two convolutional layers and the last max pooling layer are shown.

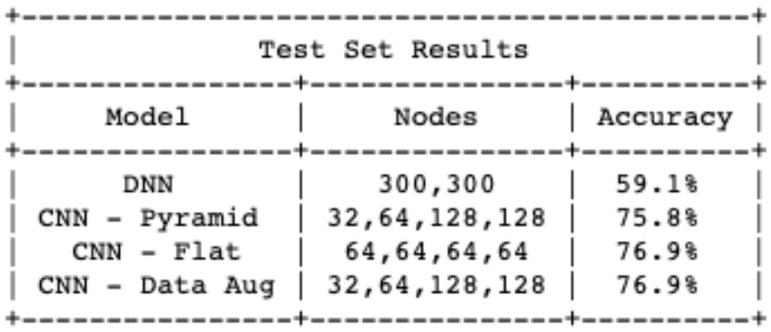


From the first two convolutional layers, it is clear that these layers are aiding in edge detection. There are several pictures of the dog that show the strength of its outline. Some of the pictures are dark except for the eyes and nose, which provide further support of CNNs detecting low-level features.

From the output of the last layer, it is clear that these results are way more abstract. It is difficult to draw definitive conclusions from these results, but it seems like different filters are focusing on different aspects of the picture. Some of the pixels are populated in the top half of the picture, or just the eyes, or the bottom half. Furthermore, the activations are more sparse in the topmost layer, which is a typical pattern expected in these models (Chollet, p. 166).

Results

The best DNN and CNN models are evaluated on the test data to aid in final model selection. It is important to mention that these models have not been refit using a smaller number of epochs, which means that test set performance is not as good as it could be due to the fact that the models were overfit on the training data. The table below summarizes these results using the exact models from training.



```
+-------------------------------------------+
|              Test Set Results             |
+----------------+---------------+----------+
|     Model      |     Nodes     | Accuracy |
+----------------+---------------+----------+
|      DNN       |    300,300    |  59.1%   |
| CNN - Pyramid  | 32,64,128,128 |  75.8%   |
|   CNN - Flat   |  64,64,64,64  |  76.9%   |
| CNN - Data Aug | 32,64,128,128 |  76.9%   |
+----------------+---------------+----------+
```

From this process, it is clear that CNN models outperform DNN models on this data set. While CNN models generally perform better on image classification tasks, it is still helpful to consider a DNN model, as it is much faster to train. In the end, while the flat and data augmentation CNN models had the exact same test set performance, the final model is the pyramid model. This model has strong test set performance and has less issues with overfitting compared to other models.

In addition, Bengio's points about a flat structure were confirmed through this assignment, since the flat-structured model and pyramid-structured model had equivalent performance. That being said, the flat model took 2.5 times longer to train. More tests need to be conducted to definitively determine if this pattern was seen for various model architectures, but based on processing speed alone, the flat structure was not selected as a final model.

Furthermore, this process helped to reveal that data augmentation could be a helpful step to consider when building a CNN model using a small data set, if implemented correctly. This step helps to ensure generalizability, which is vitally important when new images need to be classified, as there is greater risk that the new data is different from training data. Testing different data augmentation configurations is needed to realize the most gains with this approach. Due to time constraints, additional configurations could not be tested as part of this assignment.

Lastly, the visualization exercise helped to show how the lower-level CNN layers detect edges and preserve much of the input data. In contrast, the upper layers become much more abstract and help extract meaningful features that aid in classification. Together, a CNN works by detecting low-level features and combining them together to aid in classification.

If more time had been allocated, then some additional actions would have been taken, such as refitting the best models using a smaller epoch size, testing more flat structures, testing models with more variability in layer size, using a pre-trained network, and assessing/visualizing misclassifications. Hyperparameter tuning was not a focus of this assignment but should be a focus in future projects.

Conclusions

If a model needed to be built for a facial recognition task, then a deep learning model, with multiple hidden layers, should be used. This exercise is extremely complex, and this kind of data lends itself well to feature-extraction, making CNN models the perfect fit. Depending on the size of the training data available, it might be worthwhile to consider leveraging a pre-trained CNN model built using ImageNet data, since these models were built on much larger data sets and have a deep architecture. The convolutional base or lower levels could be frozen for training with new top layers defined in conjunction with business goals.

One challenge with developing a facial recognition model is that accuracy is of the utmost importance. Therefore, ensuring that there is a proper testing strategy/framework is key. Images of each user need to be captured ahead of time. Upon using an application with facial recognition, it is clear that a user is going to be using that application in a different lighting and at a different angle each time. As such, employing data augmentation to rotate, zoom, invert, and shift each image is a requirement to achieve such accuracy.

If certain people are not being identified correctly, then a comprehensive analysis of the misclassifications should be performed. Employing visualization techniques, such as understanding filters behind the model and visualizing heatmaps of class activations, would aid greatly in this endeavor. This exercise may help to reveal patterns or key areas in an image, which may lead to enhancements within the model, such as hyperparameter tuning or even stacking specialized models on top of a base CNN model.

All in all, building a facial recognition model is a tall order. This assignment helped to show that for future projects ample time should be given for testing, server resources should be provided, and a model strategy should be created to ensure that time is well spent during training.

References

Bengio, Y. (2012). Practical Recommendations for Gradient-Based Training of Deep Architectures. In G. Montavon, G. B. Orr, and K-R. Muller, eds., *Neural Network: Tricks of the Trade* (second edition). New York: Springer.

Chollet, F. (2018). Deep Learning with Python. Shelter Island: Manning.

Dogs vs. Cats. (2013). Retrieved April 22, 2019, from https://www.kaggle.com/c/dogs-vs-cats

Geron, A. (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. Sebastopol, Calif.: O'Reilly.