Deep Learning Project

Seungho Jang

February 2020

1 Harry Potter Actors Recognition

1.1 Motivation

The motivation for this project came from my childhood. Many people grew up with the Harry Potter movies, from the time they were young, and this applied to me as well. These movies are very nostalgic and surprisingly recognized over the world.

1.2 Data Collection

Data were collected by getting 1 frames per 1 sec from the Movie. Also, the data was collected from two movies: The Philosopher's Stone and the Chamber of Secrets. First of all there are 6 main characters were chosen and used for data collection: Harry, Hermione, Ron, Malfoy, Hagrid, and Professor Dumbledore. This was a bit challenging because Harry is a main character and tends to be present more than other character. This is why I had to collect more data from the Chamber of Secrets. Oversampling of Harry's pictures was not preferable because it could result in imbalanced data. The original picture size was it was 1920 x 1080, frames were resized to 600 by 500 using VLS Players scene filter. Then resized further to 140 x 90 to make a deep learning model to learn quickly. The pictures of Malfoy, Professor Dumbledore, and Hagrid were upsampled from each movies, because they are not main character unlike Harry, Hermione, and Ron. The total picture of 1,150 pictures were collected. Each character had a total of about 190 to 200. This data can be collected more for training as well as through data augmentation. Splitting 190 images to each train(70 percent), Validation(15 percent), and Test(15 percent) was done into the Data folder.

The sample images of each character are shown in Figure 1. Also, Normalization was done in each image by dividing 255, which is standardization.

(a) Harry (b) Hagrid (c) Ron

(d) Malfoy (e) Hermione (f) Dumbledore

Figure 1: The characters in the Harry Potter Movie

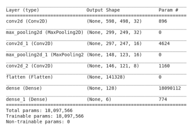
1.3 Data Pre-Processing

As mentioned, the original size has been resized to 600 by 500. Then, the size of each image was reduced to 140 by 90 and converted to gray images, concerning the speed of computation and learning capability.

2 Phase : Building a Over-fitting Model.

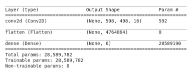
2.1 Using all the data, without splitting, obtain close to 100 percent accuracy.

As the size of images is concerned, the size of images was 600 by 500 as well as sigmoid activation and binary cross-entropy as loss function observing in figure 2.(a) and (b). The First model was built from the knowledge that I have; the architecture composed with three convolutional layer with Maxpooling, one dense layer with 128 neurons, and the other dense layer with 6 neurons. The second model uses one convolutional layer and two additional layers with the same format, though one is dense with 128 neurons and the other is dense with 6 neurons. As a result, both of the CNN networks were able to reach almost 100 percent by looking at the performance metrics. This result means overfitting, as shown in the figure 3 and 4. As a result, I believe that the number of CNN layers have to be two or three more layers so that it can over-fit the data quickly.



```
Layer (type)                 Output Shape              Param #
================================================================
conv2d (Conv2D)              (None, 598, 498, 32)      896

max_pooling2d (MaxPooling2D) (None, 299, 249, 32)      0

conv2d_1 (Conv2D)            (None, 297, 247, 16)      4624

max_pooling2d_1 (MaxPooling2 (None, 148, 123, 16)      0

conv2d_2 (Conv2D)            (None, 146, 121, 8)       1160

flatten (Flatten)            (None, 141328)            0

dense (Dense)                (None, 128)               18090112

dense_1 (Dense)              (None, 6)                 774
================================================================
Total params: 18,097,566
Trainable params: 18,097,566
Non-trainable params: 0
```

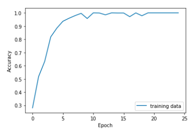
(a) First CNN Model



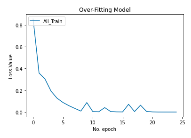
```
Layer (type)                 Output Shape              Param #
================================================================
conv2d (Conv2D)              (None, 598, 498, 16)      592

flatten (Flatten)            (None, 4764864)           0

dense (Dense)                (None, 6)                 28589190
================================================================
Total params: 28,589,782
Trainable params: 28,589,782
Non-trainable params: 0
```

(b) Second CNN Model

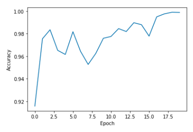
Figure 2: Comparison of two Architecture



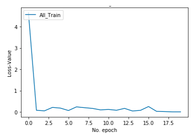
(a) Accuracy



(b) Loss

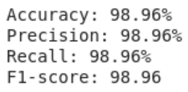


(c) Accuracy



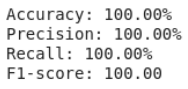
(d) Loss

Figure 3: Accuracy report First Model (a),(b) and Second Model (c).(d)



```
Accuracy: 98.96%
Precision: 98.96%
Recall: 98.96%
F1-score: 98.96
```

(a)



```
Accuracy: 100.00%
Precision: 100.00%
Recall: 100.00%
F1-score: 100.00
```

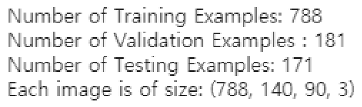
(b)

Figure 4: Precision, Recall, F1-Score for both Architecture

3 Phase : Split and Evaluate on test set

3.1 Split the data into training, development, and test set

The distribution of training, development, and test data as well as the size of images were changed to 140x90 for color images, as shown figure 5.



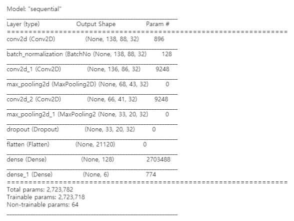
```
Number of Training Examples: 788
Number of Validation Examples : 181
Number of Testing Examples: 171
Each image is of size: (788, 140, 90, 3)
```

(a)

Figure 5: Distribution of Training, Development, and Test Set

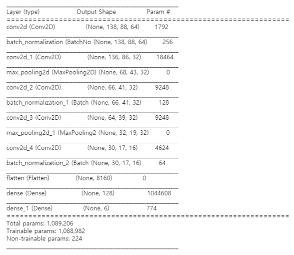
3.2 Study the performance when the number of filters and layers are increased/change

In this section, each architecture was built depending on the number of CNN Layers. As observed, 3 layers of Convolutional Neural Network was the best above small high number of layers (Second Model) and low number of layers (Third Model) as shown in the figure 6, and the result or performance metric is shown in figure 7.
As observed, the third model was the best compared to others for validation accuracy, but there were no significant changes between first model and third model. Each model's accuracy used, though the first model was the best in comparison to the other models: the accuracy for model one was 55 percent, as shown in figure 8. So, I chose the first model to see what mislabeled images there were and drew confusion matrix for true label versus predicted label, as shown in figure 9.



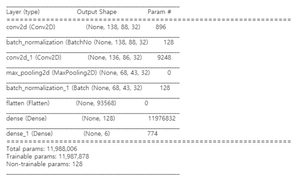
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 138, 88, 32)       896
_________________________________________________________________
batch_normalization (BatchNo (None, 138, 88, 32)       128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 136, 86, 32)       9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 68, 43, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 66, 41, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 33, 20, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 33, 20, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 21120)             0
_________________________________________________________________
dense (Dense)                (None, 128)               2703488
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 774
=================================================================
Total params: 2,723,782
Trainable params: 2,723,718
Non-trainable params: 64
_________________________________________________________________
```

(a) First Model



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 138, 88, 64)       1792
_________________________________________________________________
batch_normalization (BatchNo (None, 138, 88, 64)       256
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 136, 86, 32)       18464
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 68, 43, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 66, 41, 32)        9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 66, 41, 32)        128
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 39, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 19, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 30, 17, 16)        4624
_________________________________________________________________
batch_normalization_2 (Batch (None, 30, 17, 16)        64
_________________________________________________________________
flatten (Flatten)            (None, 8160)              0
_________________________________________________________________
dense (Dense)                (None, 128)               1044608
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 774
=================================================================
Total params: 1,089,206
Trainable params: 1,088,982
Non-trainable params: 224
_________________________________________________________________
```

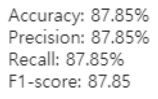
(b) Second Model



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 138, 88, 32)       896
_________________________________________________________________
batch_normalization (BatchNo (None, 138, 88, 32)       128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 136, 86, 32)       9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 68, 43, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 68, 43, 32)        128
_________________________________________________________________
flatten (Flatten)            (None, 93568)             0
_________________________________________________________________
dense (Dense)                (None, 128)               11976832
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 774
=================================================================
Total params: 11,988,006
Trainable params: 11,987,878
Non-trainable params: 128
_________________________________________________________________
```

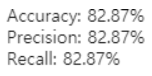
(c) Third Model

Figure 6: Different architecture depending on the number of CNN layers



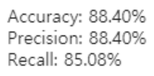
```
Accuracy: 87.85%
Precision: 87.85%
Recall: 87.85%
F1-score: 87.85
```

(a) First Model



```
Accuracy: 82.87%
Precision: 82.87%
Recall: 82.87%
```

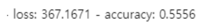
(b) Second Model



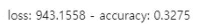
```
Accuracy: 88.40%
Precision: 88.40%
Recall: 85.08%
```

(c) Third Model

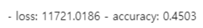
Figure 7: Performance Metrics depending on the number of CNN layers



```
· loss: 367.1671 - accuracy: 0.5556
```



```
loss: 943.1558 - accuracy: 0.3275
```



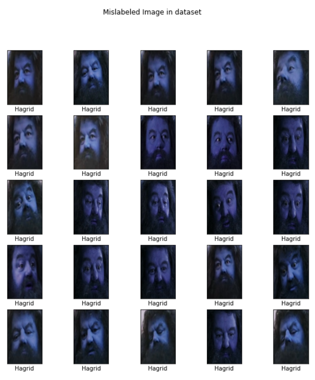
```
- loss: 11721.0186 - accuracy: 0.4503
```

(a) First Model (b) Second Model (c) Third Model

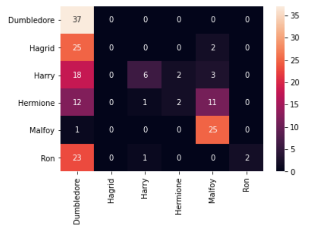
Figure 8: Performance Metrics on Test Data

As shown below, the analysis on misclassified images shows that the model is hard to learn the images of Hagrid. It's unknown why the model is not understanding Hagrid, though it may be because all the test images of Hagrid were composed with blue and black, which makes it difficult for the model to learn.

Figure 9 shows comparison between predicted label versus true label. It shows that it predicted really well on Dumbledore and Malfoy, and there are a small amount of matched features between Harry, Hermione, and Ron. Still, those features are saying that most of them are Dumbledore while Hagrid was not matched at all.



(a)

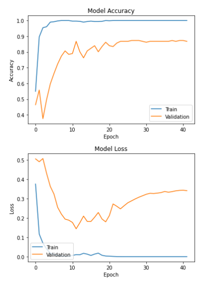


(b)

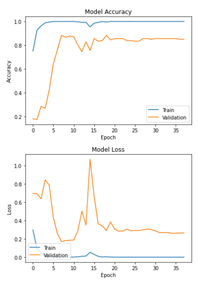
Figure 9: Miss-Classified Images and Confusion matrix

3.3 Plot the learning curves

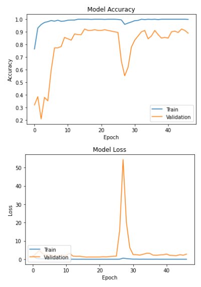
All learning curves were observed, the first model might be overfitting because of the threshold of early stopping. However, the three layers of CNNs were appropriate compared to other models because there are no high peaks where the loss increased instantly.



(a) First Model



(b) Second Model

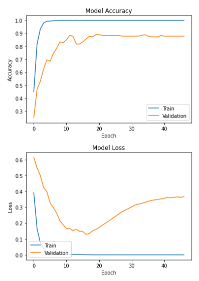


(c) Third Model

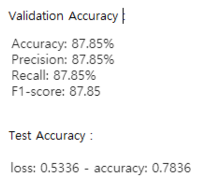
Figure 10: Learning Curve

4 Phase : Data Augmentation

Before getting into phase 4, changing the color to gray images was done in order to increase the performance metrics. Then, the result shows that the test accuracy has been increased overall on previous models, as shown in figure 11. However, the first model still has gotten the best accuracy among models. The learning curve, miss-classified images, and the confusion matrix are shown below.



(a) First Model



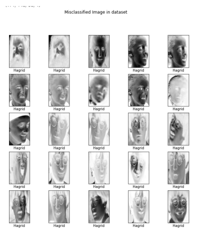
Validation Accuracy :

Accuracy: 87.85%
Precision: 87.85%
Recall: 87.85%
F1-score: 87.85

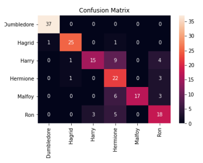
Test Accuracy :

loss: 0.5336 - accuracy: 0.7836

(b) Performance Metrics on Validation/Test



(c) Miss-classified Images



(d) Confusion Matrix

Figure 11: Result for changing color images to gray images

4.1 With the best model obtained from the previous step, apply various techniques of data augmentation.

Out of all models in the previous steps, the first model was chosen. The parameters for the data augmentation included percentages of rotation, shifting range, shear range, and zoom range have changed. All data augmentation was done by allowing horizontal flip and normalized. There were two categories of parameters that were considered for augmenting data.

First of all, finding a proper batch size and steps per epochs was necessary. In this experiment, batch size was one of the parameters, but not on how much percentages of rotation, shifting range, shear range, and zoom range were changed. Within 40 rotation and 20 percentage of changing other parameters were set, as well as the steps per epoch being 100. The batch sizes, between 20 and 50 were compared. All the results are in table 1 and 2.

The result shows that the test accuracy is about the same. However, the accuracy on validation was 4 percent different and batch sizes of 50 were used for testing how much percentages those parameters changed, as discussed. The independent variable was the batch sizes and set to 50.

Batch Size	**Validation Accuracy**	**Test Accuracy**
20	92.27	92.39
50	96.13	90.64

Table 1: Accuracy Report on different batch size

Batch Size	**Validation Accuracy**	**Test Accuracy**
20	92.81	92.39
50	96.13	91.22

Table 2: Accuracy Report on different batch size for augmentation

In table 3, the result shows different accuracy on validation and test data. It seems the best model can be picked in either 40 percent and 20 percent because they have about 92-95 on validation accuracy as well as 90-92 on test accuracy.

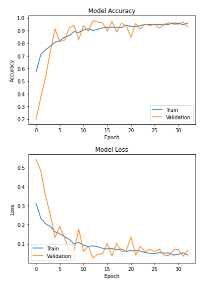
Percentage	**Validation Accuracy**	**Test Accuracy**
20	92.27	92.39
30	86.18	75.43
40	95.02	90.05
50	89.50	80.70

Table 3: Accuracy Report on different percentages of rotation, shifting range, shear range, and zoom range

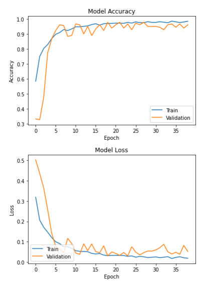
Percentage	**Validation Accuracy**	**Test Accuracy**
20	92.81	92.39
30	86.18	75.43
40	95.02	90.05
50	89.50	80.70

Table 4: Accuracy Report on different percentages of rotation, shifting range, shear range, and zoom range for augmentation

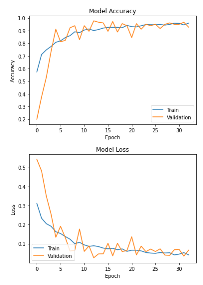
4.2 Plot your learning curves



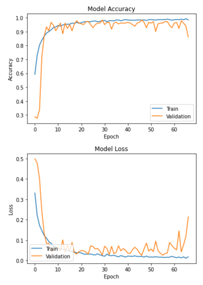
(a) Batch = 20



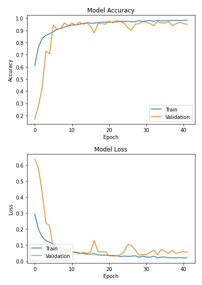
(b) Batch = 50



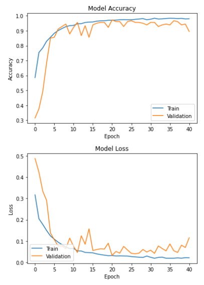
(c) 20 percent



(d) 30 percent



(e) 40 percent



(f) 50 percent

5 Phase 5. Effect of Regularization

5.1 With the best model obtained from the previous step, apply various techniques of regularization

As shown in the table above, the regularization was already done in model. So the observation from this phase are to change the batch normalization after first convolution layer to Dropout or L2 regularization layer, and change the dropout layer after max pooling layer to batch normalization or L2 regularization layer.

5.2 Changing Batch Normalization to L2 and Dropout layer

It seems that validation accuracy is constant as 92.81. Yet, each test accuracy was different among each other. It is possible that slight vanishing gradient problem can exist, such as L2 was the lowest accuracy on test set. But luckily that Batch Normalization layer was chosen in the beginning and had a best result among different regularization layers. The results are shown in the table below.

Regularization Layer	**Validation Accuracy**	**Test Accuracy**
L2	92.81	80.11
Dropout	92.81	84.21
Batch Normalization	92.81	92.40

Table 5: Accuracy Report on Different Regularization Layers

5.3 Changing Dropout layer to Batch Normalization L2 layers

Again, it seemed that original architecture was the best comparing different regularization layers. The L2 has a low validation accuracy among other layer. But Dropout regularization layer was the best among L2 and Batch normalization layer. The results are shown in the table below.

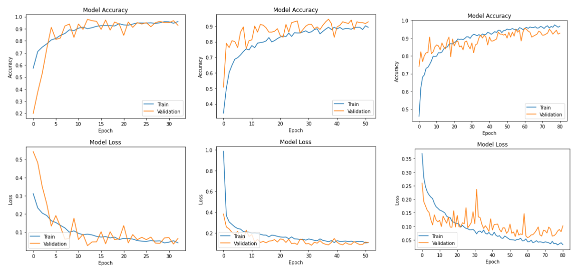
Regularization Layer	**Validation Accuracy**	**Test Accuracy**
L2	88.95	88.88
Dropout	92.81	92.40
Batch Normalization	92.81	88.88

Table 6: Accuracy Report on Different Regularization Layers

5.4 Plot Learning Curve

Learning curves are displayed in Figure 12 and 13.

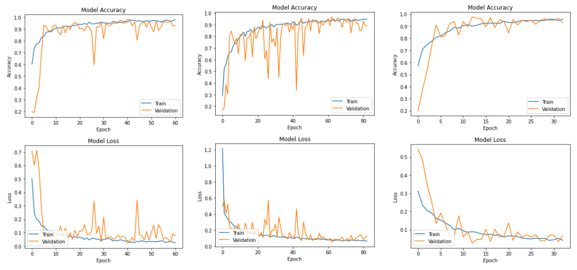
5.4.1 Changing Batch Normalization to L2 and Dropout layer



(a) Batch Normalization (b) L2 Regularization (c) Dropout Layer

Figure 12: Regularization Layers

5.4.2 Changing Dropout layer to Batch Normalization L2 layers



(a) Batch Normalization (b) L2 Regularization (c) Dropout Layer

Figure 13: Regularization Layers

6 Phase. Use pretrained models and recent architectures

For this phase, four architectures were used: VGG16, ResNet50, Inception and DenseNet. There were two categories for comparison such as the fine-tuning and dropout layers between two dense layers.

6.1 Result

Table 7, shows frozen convolutional layers. As shown below, VGG16 with regularizer and DenseNet provided a good result when compared to original. The original accuracy on validation was 92.81 and test accuracy was 92.40.

Architecture	**Validation Accuracy**	**Test Accuracy**
VGG16	93.37	88.30
VGG16 w/ Dropout	93.92	85.38
ResNet	92.81	92.40
ResNet w/ Dropout	68.50	67.83
Inception	66.02	54.38
Inception w/ Dropout	83.33	83.33
DenseNet	94.47	96.49
DenseNet w/ Dropout	94.47	93.56

Table 7: Accuracy Report on Different Architecture

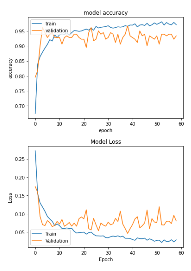
The table shows unfreezing one block was done in the architecture and the rest of the block is frozen. Within fine tuning, only VGG16 and DenseNet were used because it gives maximum accuracy and does not cause any overfitting. As shown below, the overall accuracy has been increased as compared to all-frozen model. However, the DenseNet with unfrozen layer causes an overfitting problem.

Architecture	Validation Accuracy	Test Accuracy
VGG16	93.92	95.90
DenseNet	94.47	93.56

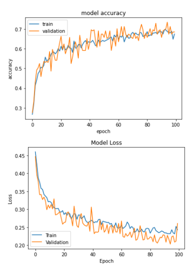
Table 8: Accuracy Report on Fine-tuned VGG16 and DenseNet

6.2 Learning Curve

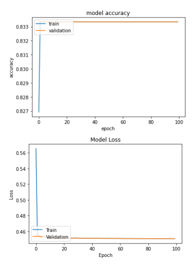
6.2.1 Plot pretrained Network without Regularization



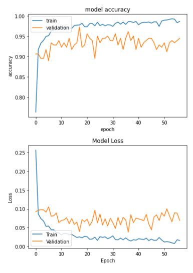
(a) VGG16



(b) ResNet



(c) Inception

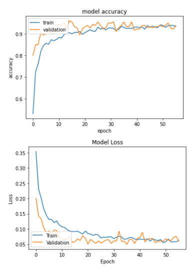


(d) DenseNet

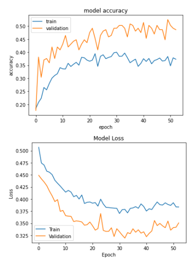
Figure 14: Four Architecture without regularization

As shown above, ResNet and VGG16 were overfitting while Inception Network has been converging so fast that it does not show smooth learning curve. The DenseNet was a relatively good model for given inputs because there is no overfitting problem.

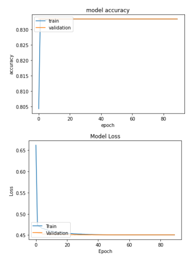
6.2.2 Plot pretrained Network with Regularization



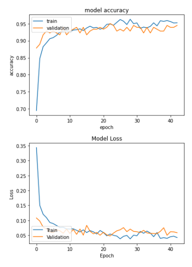
(a) VGG16



(b) ResNet



(c) Inception

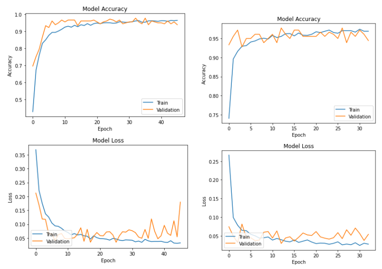


(d) DenseNet

Figure 15: Four Architecture with regularization

Within fine tuning only VGG16 and DenseNet were used because it gives maximum accuracy, and does not cause any overfitting problem.

6.3 Plot fine-tuned Network



(a) VGG16 (b) DenseNet

Figure 16: Fine-tuned Neural Network

7 Result

The best model was obtained by using the combination of data augmentation and fine-tuning on the most recent architectures. Either VGG16 and DenseNet is preferable architecture for this project, and further improvement can be made with different augmentation and regularization methods.