## Summary

## Grading Criteria Points:

### Effort to curate dataset

1000 images of my face were used in the dataset. The images were captured by writing some code which captured video from my laptop webcam and took 1 frame every 3 seconds from the video. The frames were sent to a facial recognition library which returned the bounding box of my face. The program then cropped the image down to the bounding box and saved the result to a file.

I used this program 5 times, once per mood, which resulted in 200 256x256 images per mood, for a total of 1000 cropped and classified images to use as input data.

The training set of images was augmented during training by changing the brightness and rotation using the ImageDataGenerator.

### Effort to visualize input data

A subset of the training data images can be seen below.



### Effort to correctly split data into 3 sets

The 1000 image files were randomly separated into a folder structure suitable for use with ImageDataGenerators:

* train/ (600 files)
  + cry
  + neutral
  + sad
  + laugh
  + smile
* validate/ (200 files)
  + cry
  + neutral
  + sad
  + laugh
  + smile
* test/ (200 files)
  + cry
  + neutral
  + sad
  + laugh
  + smile

### Effort to design and test various neural network architectures

Apologies up front, but designing different architectures for this project was a huge learning curve for me, and the story must be told.

Upon the start of the project, I was having little success with any type of architecture. Initially, I had replicated the architecture from the MINST dataset classification that was taught in this course. The problem-sets are similar, so I assumed the MINST architecture would also work with mood-detection. However, I found that the MINST model would not train with my image data; training accuracy always remained around 20%, which is the baseline accuracy for my dataset. I struggled with this for weeks; adding/removing layers, changing filter counts, even re-creating the images. Nothing helped; my model was hopelessly stuck at 20%.

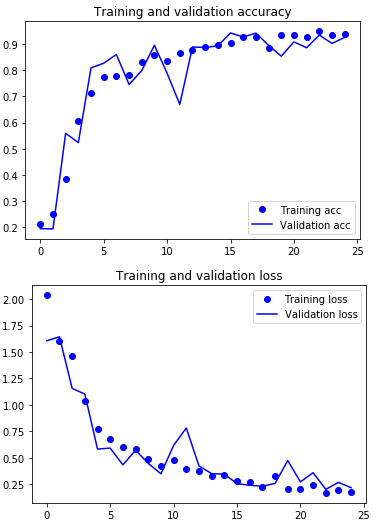
As is the case with most problems, as soon as you ask for help, you stumble upon a solution yourself. This was my case after sending my python notebook to the class professor, asking if he would take a look at what I was doing wrong. Still working on the problem a few days later, I just happened to change the Conv2d layers activations from ‘sigmoid’ to ‘reLU’. And the model trained. Not only did it train, it reached 100% accuracy. After informing the professor to “Nevermind”, I was finally able to start on the project.

Throughout the progression of the course, new topics were brought up in-class, and I tried a lot of them in my project. Most of the architectures I tested did converge with good training and validation accuracies, so it came down to picking the architecture that was the most efficient during training.

The following 6 architectures were tested and compared using 25 epochs each:

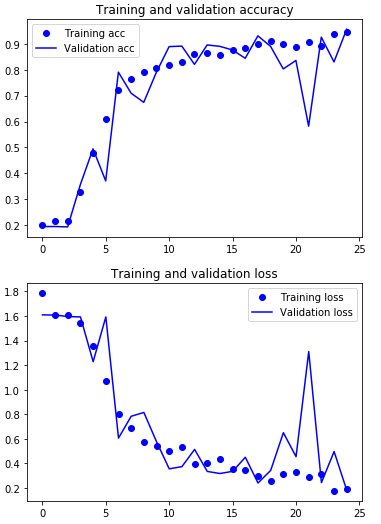
**Single Conv Layer**:

* model = models.Sequential()
* model.add( layers.Conv2D( 64, ( 3, 3 ), activation = 'relu', input\_shape=(256, 256, 1) ) )
* model.add( layers.Flatten() )
* model.add( layers.Dense( 5, activation = 'softmax' ) )
* model.summary()
* model.compile( optimizer = 'sgd', loss = 'categorical\_crossentropy', metrics = [ 'accuracy' ],)
* Total Params: 20,645,765
* Train Time: 375 seconds
* loss: 0.1759 - acc: 0.9385 - val\_loss: 0.2174 - val\_acc: 0.9245



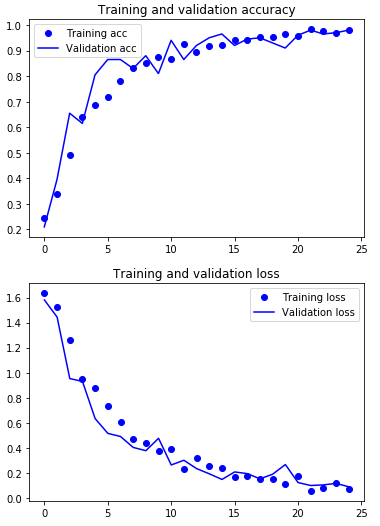
**2 Conv Layer:**

* model = models.Sequential()
* model.add( layers.Conv2D( 64, ( 3, 3 ), activation = 'relu', input\_shape=(256, 256, 1) ) )
* model.add( layers.Conv2D( 128, ( 3, 3 ), activation = 'relu' ) )
* model.add( layers.Flatten() )
* model.add( layers.Dense( 5, activation = 'softmax' ) )
* model.summary()
* model.compile( optimizer = 'sgd', loss = 'categorical\_crossentropy', metrics = [ 'accuracy' ],)
* Total Params: 40,717,061
* Train Time: 650 seconds
* loss: 0.1943 - acc: 0.9450 - val\_loss: 0.1817 - val\_acc: 0.9560



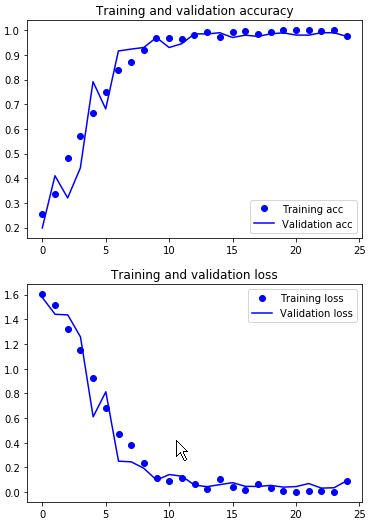
**2 Conv Layer with MaxPooling:**

* model = models.Sequential()
* model.add( layers.Conv2D( 64, ( 3, 3 ), activation = 'relu', input\_shape=(256, 256, 1) ) )
* model.add(layers.MaxPooling2D((2, 2)))
* model.add( layers.Conv2D( 128, ( 3, 3 ), activation = 'relu', ) )
* model.add( layers.Flatten() )
* model.add( layers.Dense( 5, activation = 'softmax' ) )
* model.summary()
* model.compile( optimizer = 'sgd', loss = 'categorical\_crossentropy', metrics = [ 'accuracy' ],)
* Total Params: 10,074,501
* Train Time: 375 Seconds
* loss: 0.0682 - acc: 0.9810 - val\_loss: 0.0880 - val\_acc: 0.9800



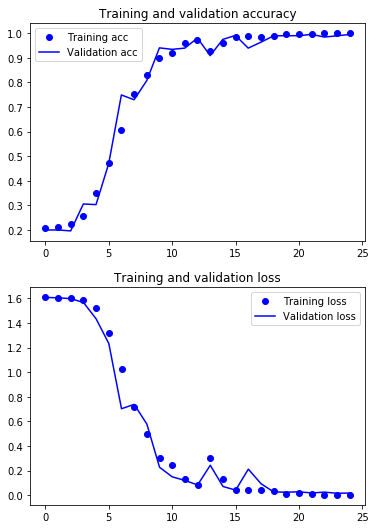
**4 Conv Layer with MaxPooling:**

* model = models.Sequential()
* model.add( layers.Conv2D( 8, ( 3, 3 ), activation = 'relu',input\_shape=(256, 256, 1) ) )
* model.add(layers.MaxPooling2D((2, 2)))
* model.add( layers.Conv2D( 16, ( 3, 3 ), activation = 'relu' ) )
* model.add(layers.MaxPooling2D((2, 2)))
* model.add( layers.Conv2D( 64, ( 3, 3 ), activation = 'relu' ) )
* model.add(layers.MaxPooling2D((2, 2)))
* model.add( layers.Conv2D( 128, ( 3, 3 ), activation = 'relu' ) )
* model.add( layers.Flatten() )
* model.add( layers.Dense( 5, activation = 'softmax' ) )
* model.summary()
* model.compile( optimizer = 'sgd', loss = 'categorical\_crossentropy', metrics = [ 'accuracy' ],)
* Total Params: 586,149
* Train Time: 325 seconds
* loss: 0.0918 - acc: 0.9785 - val\_loss: 0.0924 - val\_acc: 0.9750



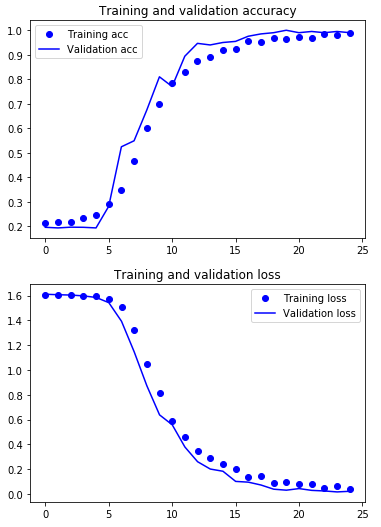
**5 Conv Layers with MaxPooling:**

* model = models.Sequential()
* model.add( layers.Conv2D( 4, ( 3, 3 ), activation = 'relu', input\_shape=(256, 256, 1) ) )
* model.add(layers.MaxPooling2D((2, 2)))
* model.add( layers.Conv2D( 8, ( 3, 3 ), activation = 'relu' ) )
* model.add(layers.MaxPooling2D((2, 2)))
* model.add( layers.Conv2D( 16, ( 3, 3 ), activation = 'relu' ) )
* model.add(layers.MaxPooling2D((2, 2)))
* model.add( layers.Conv2D( 64, ( 3, 3 ), activation = 'relu' ) )
* model.add(layers.MaxPooling2D((2, 2)))
* model.add( layers.Conv2D( 128, ( 3, 3 ), activation = 'relu' ) )
* model.add( layers.Flatten() )
* model.add( layers.Dense( 5, activation = 'softmax' ) )
* model.summary()
* model.compile( optimizer = 'sgd', loss = 'categorical\_crossentropy', metrics = [ 'accuracy' ],)
* Total Params: 176,805
* Train Time: 300 Secs
* loss: 0.0037 - acc: 0.9995 - val\_loss: 0.0175 - val\_acc: 0.9950



**5 Conv Layers with MaxPooling and Dropout**

* model = models.Sequential()
* model.add( layers.Conv2D( 4, ( 3, 3 ), activation = 'relu', input\_shape=(256, 256, 1) ) )
* model.add(layers.MaxPooling2D((2, 2)))
* model.add( layers.Conv2D( 8, ( 3, 3 ), activation = 'relu' ) )
* model.add(layers.MaxPooling2D((2, 2)))
* model.add( layers.Conv2D( 16, ( 3, 3 ), activation = 'relu' ) )
* model.add(layers.MaxPooling2D((2, 2)))
* model.add( layers.Conv2D( 64, ( 3, 3 ), activation = 'relu' ) )
* model.add(layers.MaxPooling2D((2, 2)))
* model.add(layers.Dropout(rate=.5, noise\_shape=None, seed=None));
* model.add( layers.Conv2D( 128, ( 3, 3 ), activation = 'relu' ) )
* model.add( layers.Flatten() )
* model.add( layers.Dense( 5, activation = 'softmax' ) )
* model.summary()
* model.compile( optimizer = 'sgd', loss = 'categorical\_crossentropy', metrics = [ 'accuracy' ],)
* Total Params: 176,805
* Train Time: 325 Secs
* loss: 0.0433 - acc: 0.9870 - val\_loss: 0.0211 - val\_acc: 0.9900



**Comparison metrics of the 6 architectures:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Total Params** | **Train Time**  **25 epochs**  **(seconds)** | **Training Loss** | **Training Acc** | **Validation Loss** | **Validation Acc** |
| Single Conv Layer | 20,645,765 | 375 | 0.1759 | 0.9385 | 0.2174 | 0.9245 |
| 2 Conv Layers | 40,717,061 | 650 | 0.1943 | 0.9450 | 0.1817 | 0.9560 |
| 2 Conv Layers with MaxPooling | 10,074,501 | 375 | 0.0682 | 0.9810 | 0.0880 | 0.9800 |
| 4 Conv Layers with MaxPooling | 586,149 | 325 | 0.0918 | 0.9785 | 0.0924 | 0.9750 |
| 5 Conv Layers with MaxPooling | 176,805 | 300 | 0.0037 | 0.9995 | 0.0175 | 0.9950 |
| 5 Conv Layers with MaxPooling and Dropout | 176,805 | 325 | 0.0433 | 0.9870 | 0.0211 | 0.9900 |

### Effort to evaluate your results

The model was evaluated with the remaining 200 test-images, with an accuracy of 96%.

testing\_acc: 0.964999994635582

### Effort to benchmark your method/results

### Documentation efforts

### Effort to document the training time

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Total Params** | **Train Time**  **25 epochs**  **(seconds)** | **Minutes** |
| Single Conv Layer | 20,645,765 | 375 | 6.25 |
| 2 Conv Layers | 40,717,061 | 650 | 10.8 |
| 2 Conv Layers with MaxPooling | 10,074,501 | 375 | 6.25 |
| 4 Conv Layers with MaxPooling | 586,149 | 325 | 5.4 |
| 5 Conv Layers with MaxPooling | 176,805 | 300 | 5 |
| 5 Conv Layers with MaxPooling and Dropout | 176,805 | 325 | 5.4 |

### Effort to study learning curves

This table compares the training loss/acc and validation loss/acc for the 6 architectures.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Single Conv Layer | 2 Conv Layers | 2 Conv Layers with MaxPooling | 4 Conv Layers with MaxPooling | 5 Conv Layers with MaxPooling | 5 Conv Layers with MaxPooling and Dropout |
|  |  |  |  |  |  |

Note in the last architecture (with dropout), the validation accuracy is higher than the training accuracy for most of the training, due to the dropout layer. This architecture produced a smooth training curve, slow to start, and slow to finish.

### Effort to prepare a “reproducible” Python Notebook (.ipynb) file.