Jeff Kilgore CS5390 Deep Learning Project

Summary

For my Deep Learning project, I chose the 'Mood Detection' problem in which my model should be able to predict 5 moods (cry, laugh, neutral, sad, smile) from facial expressions.

Grading Criteria Points:

1. 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. In total, the program 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.

2. Effort to visualize input data

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

3. 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

4. Effort to design and test various neural network architectures

At the start of the project, I just chose the same architecture as we learned for the MINST data-set. I quickly found out that the 'sigmoid' activations used in the MINST model was not working correctly (my model would not learn above the 20% baseline prediction). Soon afterwards, I switched the activations to 'reLu' and achieved much better results. So, 'reLu' was the activation used throughout the rest of the project.

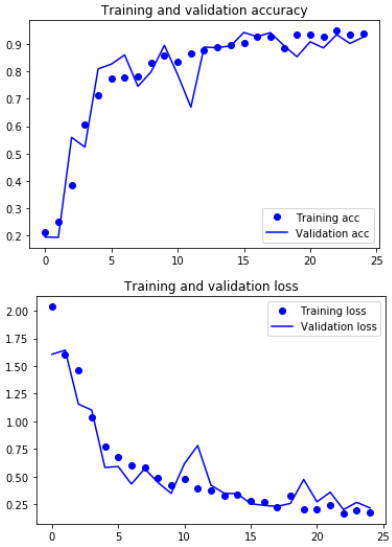
Throughout the progression of the course, new topics were brought up in-class, and I tried a lot of them in my project. Most of the architectures I tested did converge with good training and validation accuracies, so it came down to picking the architecture that was the most efficient during training.

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'],)

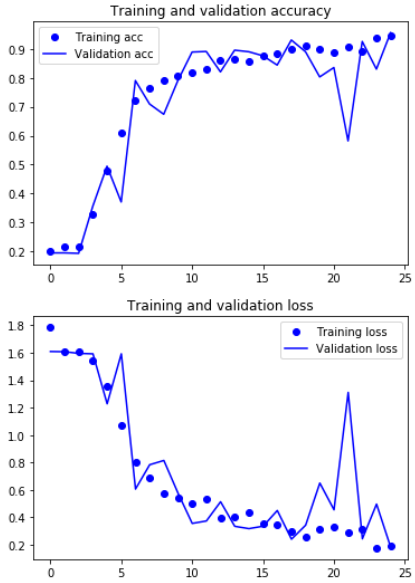
- Total Params: 20,645,765
- Train Time: 375 seconds
- loss: 0.1759 - acc: 0.9385 - val_loss: 0.2174 - val_acc: 0.9245



Notes: This model had far too many parameters, and seemed to quickly overfit.

2 Conv Layer:

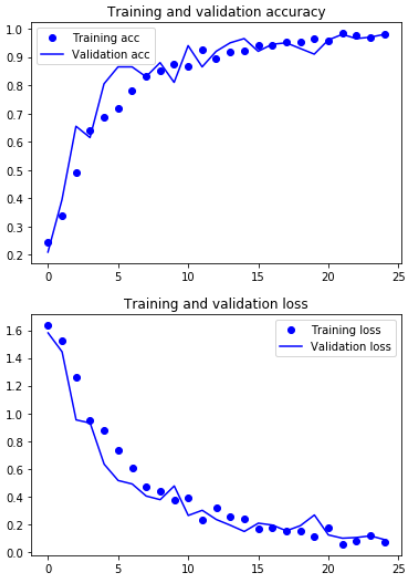
- model = models.Sequential()
- model.add(layers.Conv2D(64, (3, 3), activation = 'relu', input_shape=(256, 256, 1)))
- model.add(layers.Conv2D(128, (3, 3), activation = 'relu'))
- model.add(layers.Flatten())
- model.add(layers.Dense(5, activation = 'softmax'))
- model.summary()
- model.compile(optimizer = '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



Notes: Again, this model had far too many parameters, and seemed to quickly overfit. Time to add some reductions.

2 Conv Layer with MaxPooling:

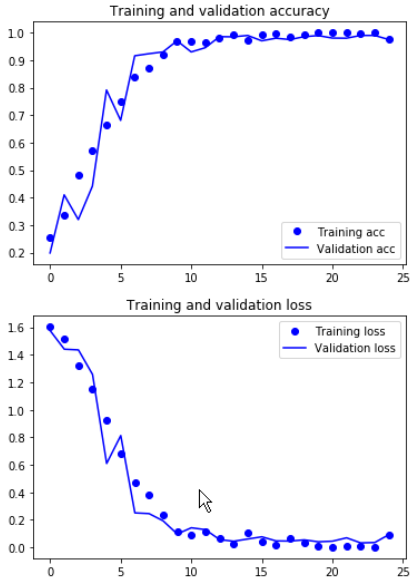
- model = models.Sequential()
- model.add(layers.Conv2D(64, (3, 3), activation = 'relu', input_shape=(256, 256, 1)))
- model.add(layers.MaxPooling2D((2, 2)))
- model.add(layers.Conv2D(128, (3, 3), activation = 'relu',))
- model.add(layers.Flatten())
- model.add(layers.Dense(5, activation = 'softmax'))
- 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



Notes: The MaxPooling did reduce the number of parameters down significantly, but there were still too many, and the model was still learning too quickly.

4 Conv Layer with MaxPooling:

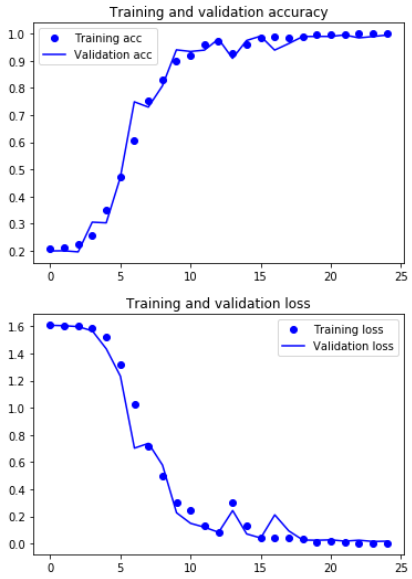
- model = models.Sequential()
- model.add(layers.Conv2D(8, (3, 3), activation = 'relu',input_shape=(256, 256, 1)))
- model.add(layers.MaxPooling2D((2, 2)))
- model.add(layers.Conv2D(16, (3, 3), activation = 'relu'))
- model.add(layers.MaxPooling2D((2, 2)))
- model.add(layers.Conv2D(64, (3, 3), activation = '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



Notes: This architecture attempts to reduce the dimensions of the network, and make it deeper. The number of parameters was also reduced another significant amount, but the learning curve is still too steep.

5 Conv Layers with MaxPooling:

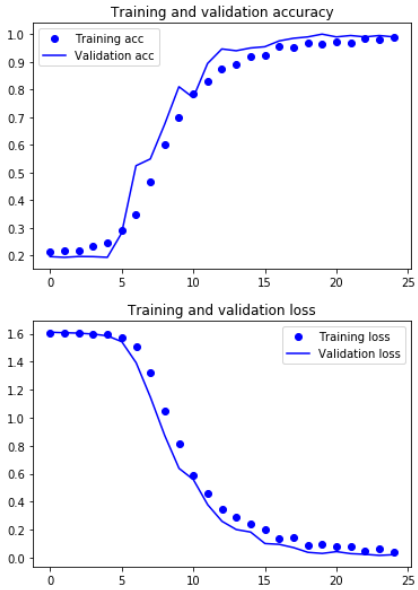
- model = models.Sequential()
- model.add(layers.Conv2D(4, (3, 3), activation = 'relu', input_shape=(256, 256, 1)))
- model.add(layers.MaxPooling2D((2, 2)))
- model.add(layers.Conv2D(8, (3, 3), activation = 'relu'))
- model.add(layers.MaxPooling2D((2, 2)))
- model.add(layers.Conv2D(16, (3, 3), activation = '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



Notes: Adding another set of Conv2d and MaxPooling layers reduced the parameters again. The learning curve is reducing, but it can be better.

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



Notes: Adding a dropout layer to the previous architecture smoothed out the learning curve to a rate that I feel is acceptable. Note that the dropout also decreased the training accuracy to below the validation accuracy for most of the training. This indicates that it was more difficult for the model to learn. This architecture was used for the final testing.

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

5. Effort to evaluate your results

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

```
testing_acc: 0.964999994635582
```

6. Effort to benchmark your method/results

While my model had high accuracy on the test-set, it didn't do so well in real world conditions.

The other 'Mood Detection' models that I found on the web to benchmark against used the FER dataset from Kaggle. This dataset has 7 classes, while my project model only has 5; so, doing a direct comparison wasn't possible. However, I was able to find a project on GitHub which had a model trained on FER, and a python program to open a laptop webcam and predict the moods in real-time. The project is located here (https://github.com/abhijeet3922/FaceEmotion_ID).

I was able to modify the code to display my own model predictions next to the benchmark predictions in real-time. I also made a video of the comparison, and uploaded it to YouTube.
https://youtu.be/LfzMVY15cLk

A couple of things that I noticed:

- My model seemed to be very dependent on the angle of my face in regards to the camera. Just rotating my face left and right produced different moods.
- Lighting also seemed to play a role in my model predictions. Different lighting produced different predictions.

7. Documentation efforts

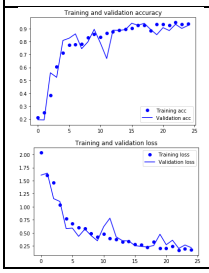
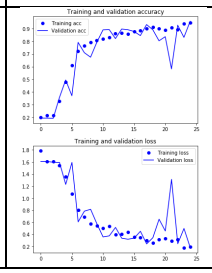
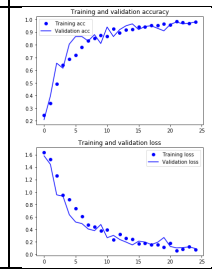
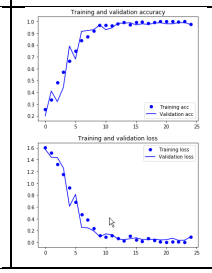
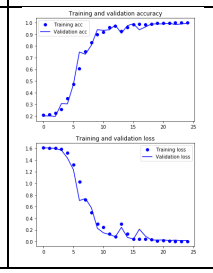
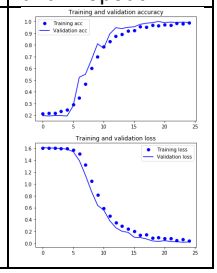
This report shows the efforts spent on documentation. Also, the python notebook is documented well.

8. Effort to document the training time

Model	**Total Params**	**Train Time 25 epochs (seconds)**	**Train Time (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

9. 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.

10. Effort to prepare a "reproducible" Python Notebook (.ipynb) file.

All files for this project, including a reproducible notebook are located in GitHub:

https://github.com/jnkx9c/DL_Project

A direct link to the notebook is here:
https://github.com/jnkx9c/DL_Project/blob/master/MoodDetection_Project.ipynb