Darius Bailey | Nelson Foster

DATS 6203 - 11

Machine Learning II

*Final Project Report - Group 3*

***Facial Recognition***

***Introduction***

Facial recognition has been at the forefront of deep learning applications. Our project sought to leverage publicly available facial image datasets and frameworks to train a deep network to correctly classify facial expressions.

***Description of the Datasets***

***YaleFaces***

When initially seeking datasets to work with, we found that large datasets of facial images were more difficult to acquire than anticipated. As a point of departure, we settled upon the [YaleFace](http://vision.ucsd.edu/content/yale-face-database) dataset. We chose this dataset because it was the only one we could find that was easily accessible and encoded with facial expression classifications.

The YaleFace Dataset was compiled by the University of California – San Diego’s Computer Vision Program, and consists of 165 greyscale images with the following facial expressions, which were used as classes for the models:

Class 0: ‘glasses',

Class 1: ‘happy

Class 2: 'leftlight',

Class 3: 'normal',

Class 4: 'rightlight',

Class 5: 'sad',

Class 6: 'sleepy',

Class 7: 'surprised',

Class 8: 'wink',

Class 9: 'centerlight',

Class 10: ‘noglasses'.

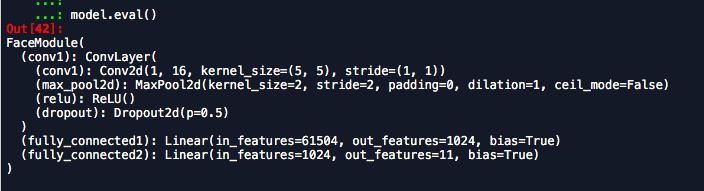
***ColorFERET***

Subsequently, we explored using the National Institute of Standards & Technology’s (NIST) Color Face RecognitionTechnology[Color FERET](https://www.nist.gov/itl/iad/image-group/color-feret-database) database, a color image facial recognition database used to evaluate and research biometric systems. After requesting access to the dataset from NIST, we discovered that it had over 11,000 images, however it was geared toward the orientation and position of faces. Also, the structure of the dataset (multiple nested folders, images encoded in a dated format, ground truth files difficult to associate with the images, etc.) made this dataset difficult to work with using the available frameworks we were familiar with. After using various tools such as Apple’s Automator to preprocess the data into a single, usable directory in a modern image format, we found that the various classes of images (Frontal, Left, Right, Quarter-Left, etc., were incompatible with the network we trained. Hence, our project focused on the YaleFaces dataset.

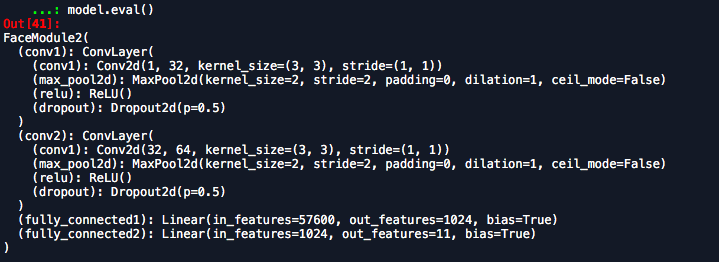
***Experimental Setup***

PyTorch was used as the primary deep learning framework for this exercise, and four convolutional neural network designs were tested. Each design experimented with different numbers of convolutional layers, and batch normalizations

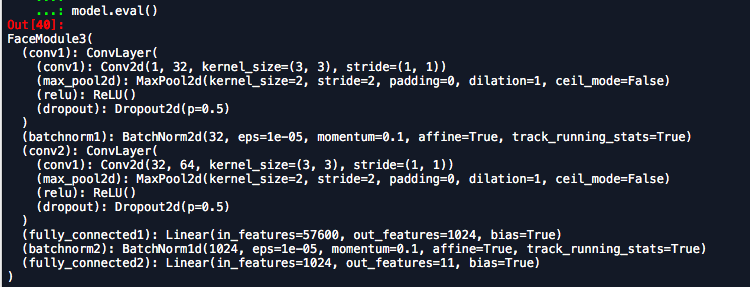
*FaceModule*



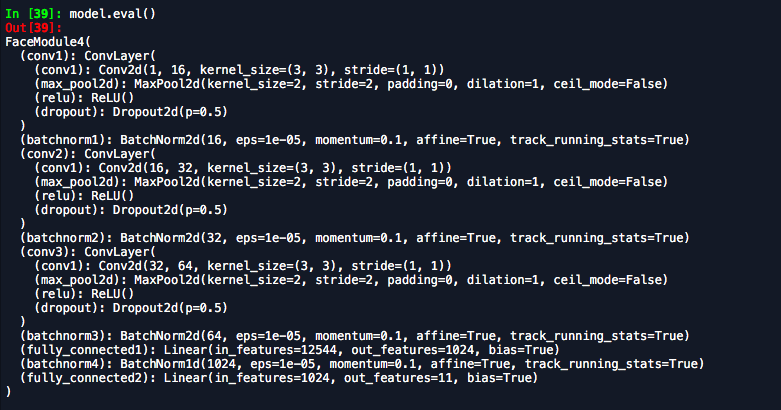
*FaceModule2*

**

*FaceModule3*

**

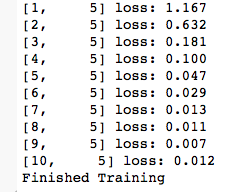
*FaceModule4*

**

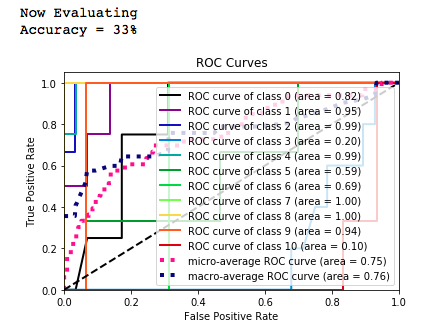
***Results***

*FaceModule*

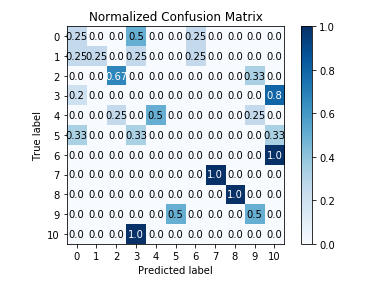
*FaceModule, the initial network consisted of one convolutional layer (with one dropout layer), and two fully connected layers. Overall performance was at 33% Accuracy.*

**

*Figure 1.1. Loss per Class, FaceModule*

**

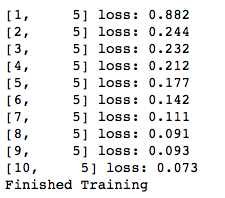
*Figure 1.2. Overall Accuracy and Accuracy per Class, FaceModule*

**

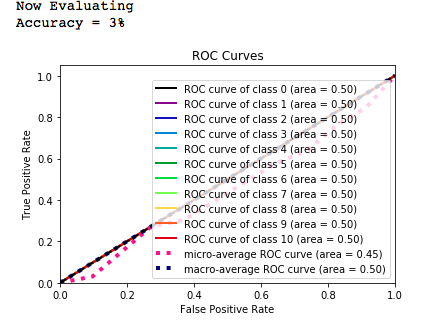
*Figure 1.3. Confusion Matrix, FaceModule*

*FaceModule2*

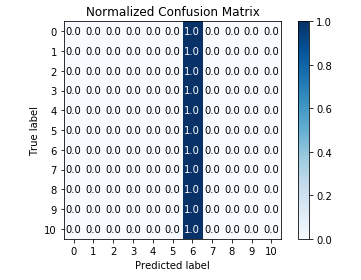
For FaceModule 2, an additional convolutional layer was added, and the out-channels were increased from 16 to 32. This actually decreased overall accuracy to 3%.



*Figure 2.1.. Loss per Class, FaceModule2*



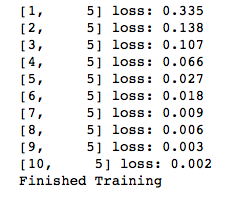
*Figure 1.2. Overall Accuracy and Accuracy per Class, FaceModule2*



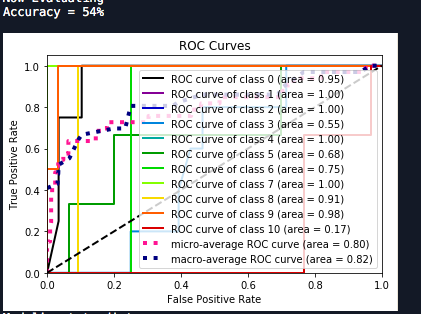
*Figure 2.3. Confusion Matrix, FaceModule2*

*FaceModule3*

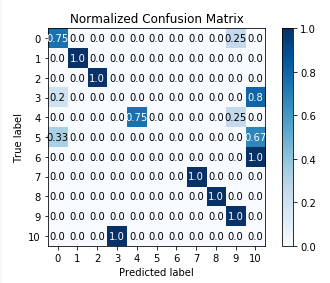
In FaceModule3, a Batch Normalization layer was added. This significantly improved performance, to 54%.



*Figure 3.1.. Loss per Class, FaceModule3*

**

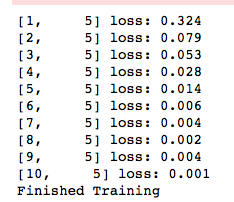
*Figure 3.2: Overall Accuracy and Accuracy per Class, FaceModule3*



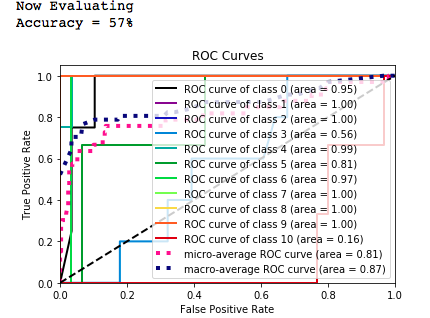
*Figure 3.3: Confusion Matrix, FaceModule3*

*FaceModule4*

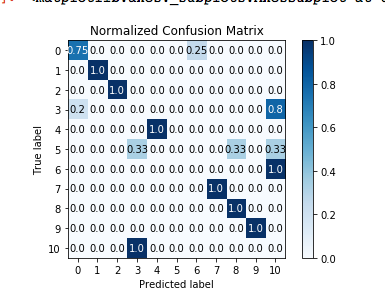
FaceModule4 leveraged three convolutional layers and four batch normalization layers, raising accuracy to 57%.



*Figure 4.1: Loss per Class, FaceModule4*

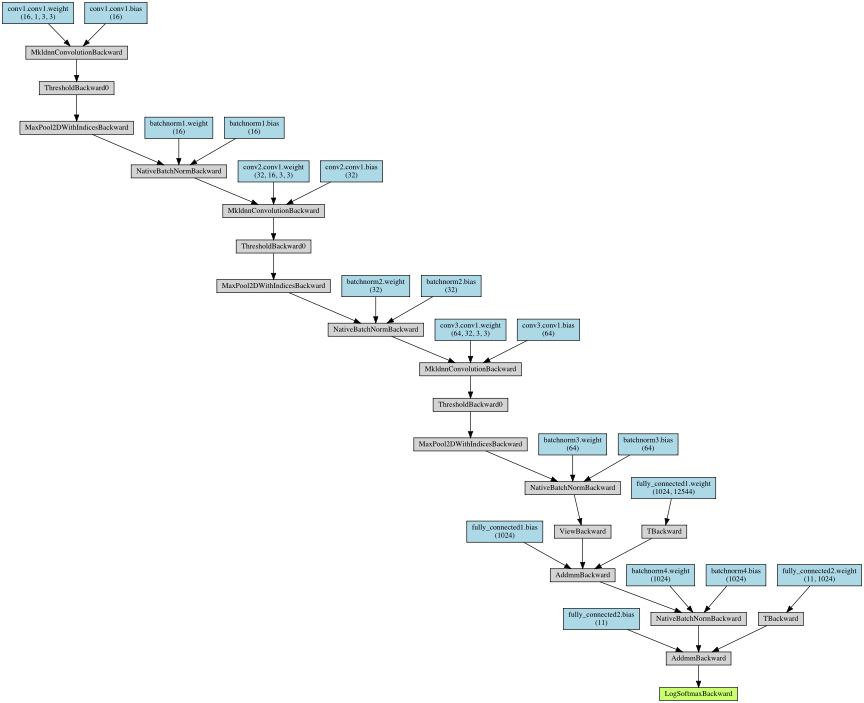


*Figure 4.2: Overall Accuracy and Accuracy per Class, FaceModule4*

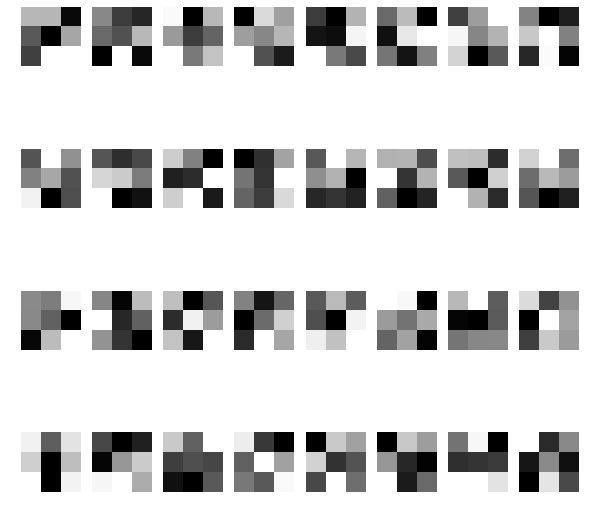
**

*Figure 4.3: Confusion Matrix, FaceModule4*

Overall, batch normalization layers seem to be critical in improving accuracy. A visualization of the network is as follows:



*Figure 5. 1: Network Architecture, FaceModule4*

**

*Figure 5. 2: Filters and Layers , FaceModule4*

*Summary and Conclusions:*

Though an overall accuracy of 57% may not seem like good performance, when looking at the various classes, there three specific classes that the networks struggle with (i.e. accuracy below ~80%): Class 3 (‘Normal’), Class 5 (‘Sad’), and Class 10 (‘no glasses’). Perhaps there are fine tuning methods that could be deployed so that the network can better identify the subtleties of these three categories.

References

[CV2 - classification w/o neural net](https://www.youtube.com/watch?v=tybtJRdeR3A)

PyTorch - Flower Image Classification - [Youtube](https://www.youtube.com/watch?v=zFA8Cm13Xmk)  [Github Repo](https://github.com/LeanManager/PyTorch_Image_Classifier)

[XML to JSON Module](https://pypi.org/project/xmljson/)

[XML Parser Module](https://docs.python-guide.org/scenarios/xml/)

[Loading images from multiple files (colorFERET)](https://discuss.pytorch.org/t/how-to-load-images-from-different-folders-in-the-same-batch/18942)

[PyTorch Visualizations](https://github.com/utkuozbulak/pytorch-cnn-visualizations)

[CNN Visualizations](https://github.com/sar-gupta/convisualize_nb/blob/master/cnn-visualize.ipynb)

[PyTorch Visualizations - Model](https://github.com/Airconaaron/blog_post_visualizing_pytorch_cnn/blob/master/Visualizing%20Learned%20Filters%20in%20PyTorch.ipynb)

*Textbook: Chapter 5, Deep Learning for Computer Vision.* Chollet, Francois (2018). *Deep Learning with Python. Manning Publications Co.*