HW7-8: Facial Recognition
November 10, 2019
RBE595ST: Deep Learning for Advanced Robot Perception

Team #2: Amjad Khan, Brian Wells, Craig Miller, Fanxing Meng, Kyle Cantrell, Revant Mahajan, Shreyas Chandra Sekhar, Tanuj Sane, Thomas Pitera, Troy Denning

Introduction

The challenge for the team was to create a deep learning algorithm that is able to correctly identify each student's face in a large database of photos. Each student created five 1-minute videos of their faces at various angles to generate the image data needed for training. Several models were then created and trained using the large data set generated. A GitHub repository was set up to share the code and slack channel and email chain used to compare results, troubleshoot, and exchange ideas. The best models were documented and their success evaluated.

Dataset Generation

Each team member took five ~60-second videos of their face using their phone in the vertical orientation and uploaded their videos to a computer. On the computer, the mp4 files were converted to jpg files at roughly 30 frames per second (FPS).

To convert our video files from mp4 to jpg files, a utility function (mp4_to_jpg.py) was created to leverage OpenCV's built-in functionality. Our utility function is an enhancement of Keith Weaver's code. Our enhancements including specifying the frame rate, abstracting functionality, and improving object naming.

After all images were made available as jpgs, we resized the images using two additional utility functions (resize_images.py and resize_images_crop.py). These functions leverage the PIL image processing library and were quickly able to process all images. resize_images_crop.py is an improvement of resize_images.py that allows the user to maintain aspect ratio when converting their images. Initially, we resized the images to 224x224, however, processing these images took a considerable amount of RAM and caused issues on many of our machines. As discussed later in this report, the imageset was resized to other dimensions to observe the impact on memory allocation and the performance of our neural networks.

Results and Discussion

Model #1

4 Convolutional Layers, 50% Dropout, Glorot normal initialization, 10 Epochs

This model consists of 2 convolutional layers with 64 filters and kernel size of 3x3 followed by 2 pooling layers interleaved in between. The next 2 layers increase the number of filters to 128 with a kernel size of 3x3. This method is based on the success of the VGG16 model, where

increasing the depth of the convolutional layers increases accuracy. The last three layers are fully connected and flattened with 512 nodes each with Dropout set to 50%. Additionally, these layers are initialized with the Glorot normal method based on the success of Xavier initialization in CNN's.

The input to this network was scaled down and tested at 32x32x3 for a load management measure. A summary of this model can be seen below.

Layer (type)	Output Shape	Param #
conv2d_70 (Conv2D)	(None, 30, 30, 64)	1792
max_pooling2d_46 (MaxPooling	(None, 15, 15, 64)	0
conv2d_71 (Conv2D)	(None, 13, 13, 64)	36928
max_pooling2d_47 (MaxPooling	(None, 6, 6, 64)	0
conv2d_72 (Conv2D)	(None, 4, 4, 128)	73856
conv2d_73 (Conv2D)	(None, 2, 2, 128)	147584
max_pooling2d_48 (MaxPooling	(None, 1, 1, 128)	0
dropout_23 (Dropout)	(None, 1, 1, 128)	0
flatten_9 (Flatten)	(None, 128)	0
dense_25 (Dense)	(None, 512)	66048
dropout_24 (Dropout)	(None, 512)	0
dense_26 (Dense)	(None, 512)	262656
dropout_25 (Dropout)	(None, 512)	0
dense_27 (Dense)	(None, 10)	5130

Total params: 593,994
Trainable params: 593,994
Non-trainable params: 0

Figure 1. Model Summary of Model #1

For hyperparameters of this model we used the following:

- Adam Optimizer
- Glorot (Xavier) Normal Initialization
- ReLU Activation in every layer other than the output layer (Softmax)
- Categorical Cross Entropy Loss
- BatchSize of 64.

The results with this network were phenomenal with an input size of 32x32x3 with a fast convergence rate and eventual accuracy of 99.98%. A plot of the results can be seen below.

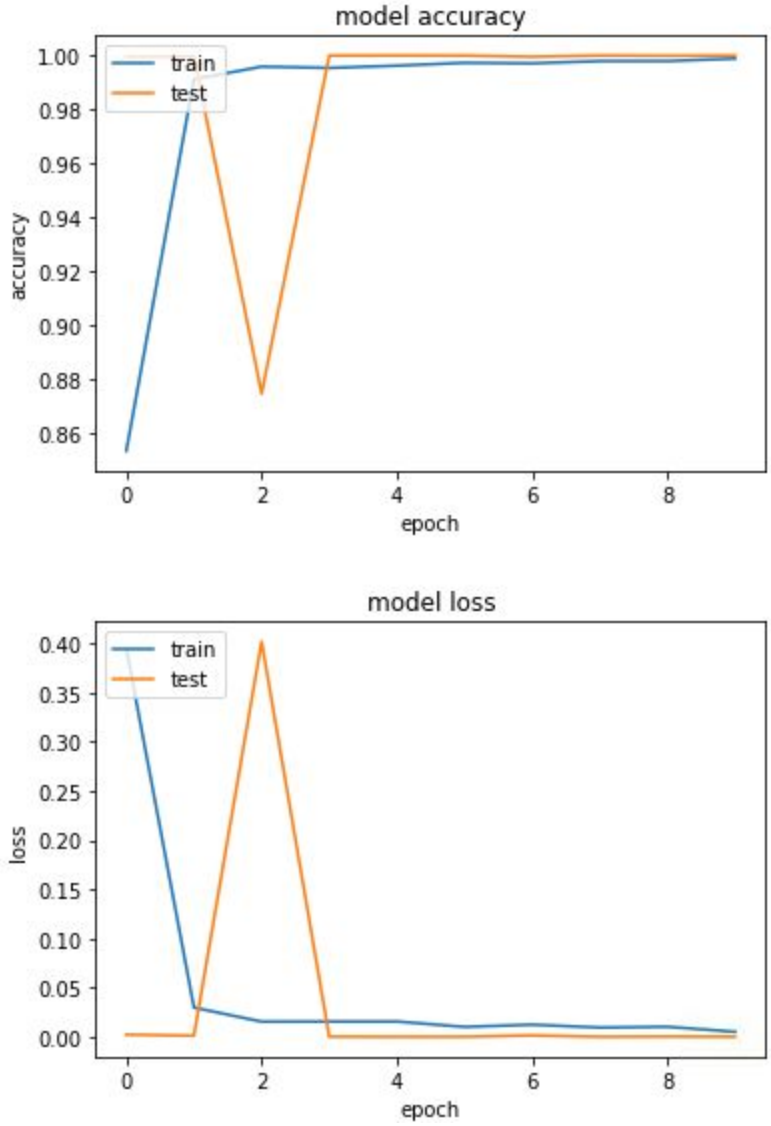

Figure 2. Performance Plot of Model #1

```
Epoch 1/10
 - 13s - loss: 0.3641 - accuracy: 0.8621 - val_loss: 0.0092 - val_accuracy: 0.9977
Epoch 2/10
 - 12s - loss: 0.0234 - accuracy: 0.9929 - val_loss: 7.2649e-04 - val_accuracy: 0.9998
Epoch 3/10
 - 12s - loss: 0.0147 - accuracy: 0.9959 - val_loss: 5.3131e-04 - val_accuracy: 0.9998
Epoch 4/10
 - 12s - loss: 0.0148 - accuracy: 0.9960 - val_loss: 3.2394e-04 - val_accuracy: 0.9999
Epoch 5/10
 - 12s - loss: 0.0109 - accuracy: 0.9970 - val_loss: 0.0019 - val_accuracy: 0.9996
Epoch 6/10
 - 12s - loss: 0.0064 - accuracy: 0.9983 - val_loss: 7.5671e-06 - val_accuracy: 1.0000
Epoch 7/10
 - 12s - loss: 0.0169 - accuracy: 0.9960 - val_loss: 4.4239e-04 - val_accuracy: 0.9998
Epoch 8/10
 - 12s - loss: 0.0033 - accuracy: 0.9991 - val_loss: 4.9712e-05 - val_accuracy: 1.0000
Epoch 9/10
 - 12s - loss: 0.0120 - accuracy: 0.9973 - val_loss: 2.5218e-04 - val_accuracy: 0.9999
Epoch 10/10
 - 12s - loss: 0.0102 - accuracy: 0.9980 - val_loss: 4.6355e-04 - val_accuracy: 0.9998
```

Figure 3. Model#1 Output showing 99.77-100% accuracy in validation.

This “VGG-Lite” model was able to perform exceedingly well with a 12 second per step timing resulting in a 120-second training and evaluation on a Tesla GPU (Google Cloud VM). At 99.98% accuracy on validation data (with several epochs at 100% accuracy) this model has proved to be successful. Contrastly the full VGG16 model suffered from drastic underfitting and took significantly longer to train, offering no accuracy or performance enhancements over this version.

Model #2

2 Convolutional Layers, 20% Dropout, He normal initialization, 10 Epochs

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 30)	2280
max_pooling2d_1 (MaxPooling2	(None, 14, 14, 30)	0
conv2d_2 (Conv2D)	(None, 12, 12, 15)	4065
max_pooling2d_2 (MaxPooling2	(None, 6, 6, 15)	0
dropout_1 (Dropout)	(None, 6, 6, 15)	0
flatten_1 (Flatten)	(None, 540)	0
dense_1 (Dense)	(None, 128)	69248
dropout_2 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 128)	16512
dropout_3 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 128)	16512
dropout_4 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 64)	8256
dense_5 (Dense)	(None, 10)	650

Figure 4. Model Summary of Model #2 (2 Convolutional Layers, 20% Dropout, He normal initialization, 10 Epochs)

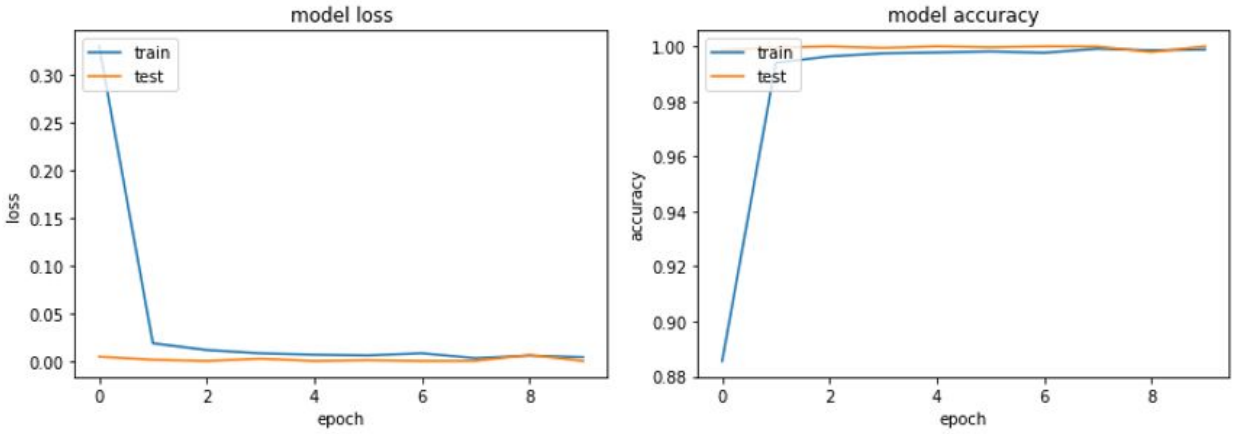

Figure 5. Model #2 Performance

Before training Model #2, the read_data function from the deep_utils folder was called to read all of the resized image data and save it as three separate pickle files. The pickled data was then loaded into the main CNN file and then split into training and test groups. To avoid memory allocation errors, this model used 32x32 pixel training and test images.

The original basis of this CNN comes from the large CNN model used to process the CIFAR dataset earlier in the course. Two convolutional layers were used at the beginning of this network. However, additional dense layers were added and he_normal initialization was utilized to improve model accuracy. This model also took advantage of the adam optimizer. ReLu activation functions were used for all layers with the exception of the output layer, which used softmax. A batch size of 128 was used. After the second epoch, the validation accuracy already surpasses 99%. While these results are terrific for our test split, this model may not perform well if presented with new, previously unseen images. After plotting the results, the model and weights are saved as yaml and h5 files for later use. This model took ~87 seconds to run on a single NVIDIA GeForce RTX 2060 GPU.

This model was also ported to an iMac Pro and run with a 16GB AMD GPU at the full 224 x 224 resolution. An accuracy of 99.99% was quickly obtained (~3 epochs) just as the lower resolution model had performed.

Model #3

Model #3 was an attempt at training with the higher resolution (224 x 224) images while skirting the memory issues encountered with the better performing model. The model summary for model #3 is shown below in figure 6. The accuracy and loss plots for the higher resolution model are shown below in figure 7. While the performance did not meet that of the best model, it

was capable of running the higher resolution images with less memory. In would be interesting to investigate and understand the trade off between the two approaches to enable a better memory constrained implementation.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 220, 220, 30)      2280
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 110, 110, 30)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 106, 106, 16)      12016
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 53, 53, 16)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 53, 53, 16)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 44944)             0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               5752960
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 84)                10836
_________________________________________________________________
dropout_3 (Dropout)          (None, 84)                0
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 680
=================================================================
Total params: 5,778,772
Trainable params: 5,778,772
Non-trainable params: 0
_________________________________________________________________
```

Figure 6. Model #3 Summary

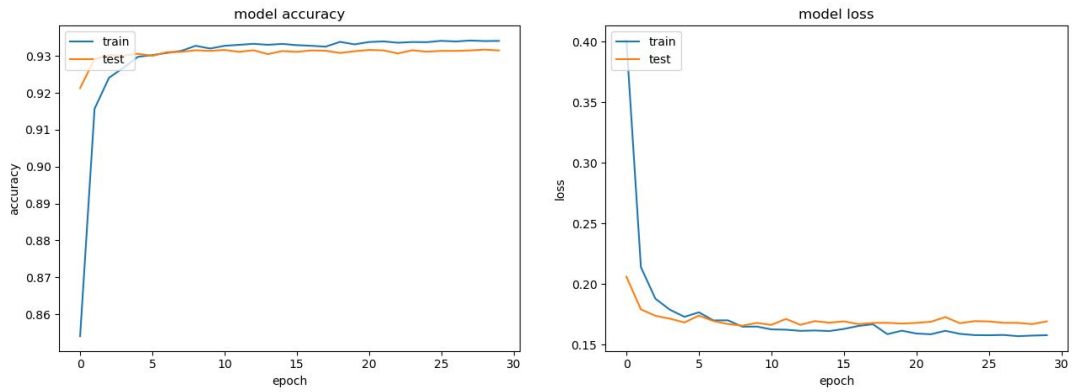

Figure 7. Model #3 Performance

Conclusion

The best performing model appears to be model #2. The additional convolutional and dense layers added to the base implementation along with the use of the 'he_normal' initializer were identified as the key contributors to the success of the best performing model. Training this model on 224x224 required a high-performance computer with a significant amount of RAM (16GB), however, it was not impossible. Smaller implementations of this model using 32x32 pixel images could be run in a matter of seconds on most machines. The high accuracy (>99.99%) of Model #2 on 224x224 images validates Team #2's methodology for facial recognition using a deep convolutional neural network.

Prediction Utility

As a method of further verification, a prediction utility was developed to allow users to select sections of user-specified input images to use as inputs to the model. This was used to gauge performance of the model on real-world images. The model is loaded from the filesystem in .yaml format, and weights are loaded from .h5. The model used in the utility is trained on the 224x224 dataset, and so the ROI selection allows 224x224 selections.

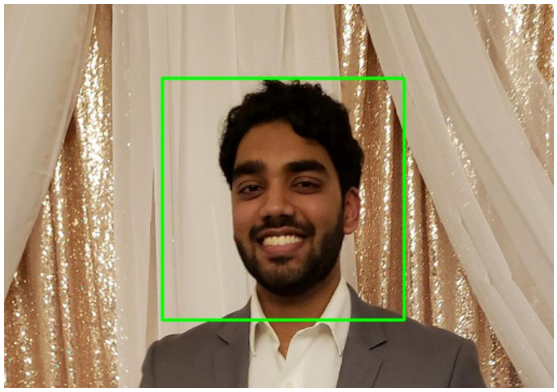

Figure 9: 224x224 ROI selection in prediction utility

On the ten images tested, the model was able to correctly identify two of them - further testing with a large set of real-world images is required to make a definitive conclusion about performance on real-world images (as opposed to those in the dataset).

Reference Links

1. Model #2 and Weights
2. Team #2 HW7-8 GitHub
3. Team #2 Image Data
4. Team #2 Pickled Data (32x32)