Artificial Intelligence CSC 665-01 Spring 2018
Term Project Documentation
San Francisco State University

Racial Recognition

Team Members
Ralph Acosta
Jasmine Mann
Andrew Hutzel
Sannuj Singhal

Submitted to Ilmi Yoon
Department of Computer Science, SFSU
Version 1.0
Last Modified May 21, 2018

INDEX

1. Overview

After completing homeworks 5 and 6, and learning about convolutional neural networks, we were left inspired and intrigued. This motivation lead us to deciding that we wanted to attempt the challenge of building a software that uses a convolutional neural network to identify people of different ethnicities. From there we came up with the idea for our project, Racial Recognition. Racial Recognition aims to classify people between four different ethnicities: African American, Asian, Caucasian, and Hispanic.

Convolutional neural networks (CNN's) are a technique in deep learning most commonly used for image analysis and classification problems. This particular strength about them made them perfect to use for our project. Racial Recognition uses two main CNN models: the sequential model and the VGG16 model. Both these models are available in the Keras neural network library.

In this paper, we discuss the design and implementation of the Sequential and VGG16 models on our own datasets, and analyze the results.

- Link to GitHub repository: https://github.com/jkmann/TermProject665
- Link to video demo: https://www.youtube.com/watch?v=tNJC2zIWCiQ&feature=youtu.be
- Images can be found in the following datasets: Racesv2.0, DatasetOfRaces, and DatasetOfRacesGender.
- Corresponding code can be found in the following notebooks: CNN.ipynb, CNN4Ethnicities.ipynb, and CNNGender.ipynb.

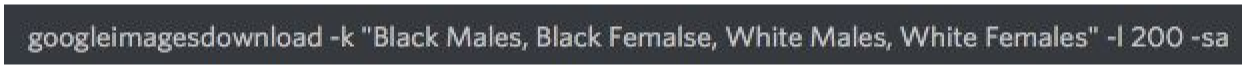
Dependencies:

- Google Cloud Compute Engine
- Anaconda
- Python 3.6
- Keras
- Tensorflow
- jupyter
- matplotlib
- numpy
- scikit-Learn
- google_images_download

2. Problem Statements

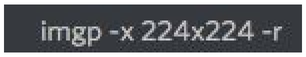
Data Gathering:

To gather and create the required data and datasets we will implement a python script that we found on GitHub for searching and downloading images to our local disk. We will clone the program through GitHub and then manually set it up and run it in our command line with commands such as:



```
googleimagesdownload -k "Black Males, Black Femalse, White Males, White Females" -l 200 -sa
```

Once we gather the images we need, we will resize them to our preferred size of 224x224, as they cannot be bigger than that to use with the VGG16 model. To do that, we will use the ImageResizer tool available on simpleimageresizer.com.



```
imgp -x 224x224 -r
```

- Tool used to scrape images with keywords off of google: https://github.com/hardikvasa/google-images-download
- Image Resizer: http://www.simpleimageresizer.com/

Setting Up Data:

In order to train and test our model, we will break up our gathered data (images) into three distinct directories within our datasets: train, valid, and test.

- Train: the set of data used to train the model during each epoch. We hope that in doing this, we will be able to deploy our model and have it detect on new data that it has never seen before and make predictions based off of what it has learned

- Valid: Separate from training set, used to validate our model during training, this process helps give us assistance by giving information that can be useful in adjusting our hyperparameters. Each epoch our model will be trained on the training set and simultaneously validated on the validation set. This set is mainly used to ensure our model is not overfitting the data in the training set

- Test: Used to test the model after the model has already been trained. This is also an entirely different set from the test and valid, so once the data is trained and validated, we move on to the test set. The test set will be the only set with no labels, however the other two sets must be labeled so we can see the metrics given during training, such as loss and accuracy from each epoch. This set should be similar to deploying the model.

Building and Training the Model:

Working with keras we attempted to build a CNN that can predict whether an image is of a Caucasian, Hispanic, Asian, or African American person. We built a simple CNN from scratch and trained that. This did not perform that well.

Our next option was trying a VGG16 model which we needed to import and build. The VGG16 model was a pre-trained model. We needed to fine tune the last classifier by training it on only the four ethnicity categories that we wanted to use for our program. We were then able to compare results between our model from scratch, and the one built from the VGG16 model.

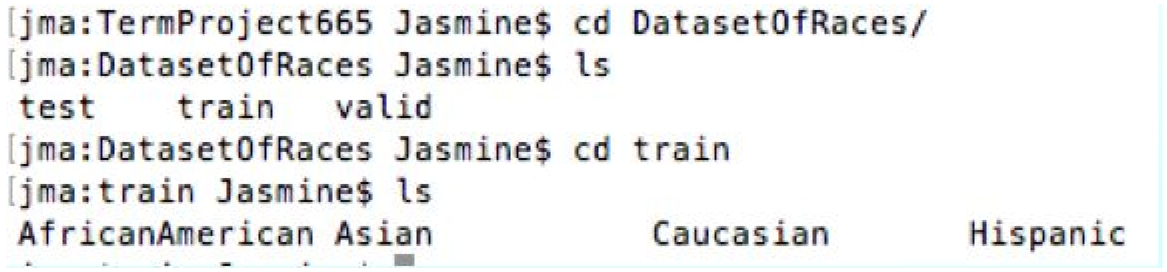
3. Solution Design

Data Setup:

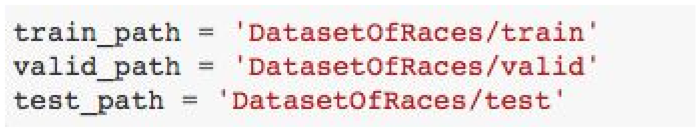
- In our project folder, there is a DatasetOfRaces directory with three directories inside, Test, Train, and Valid:
    1. Test: contains 40 images that we use to test our model, to predict ethnicity (these should not be the same images from the train or valid sets)
    2. Train: contains 400 images which are the images the model will be trained with
    3. Valid: contains 200 images used for the validation set that our model is validating over each epoch or iteration

- In terminal, I cd into DatasetOfRaces/ and there using ls I can navigate to either test, train or valid. The train and valid directories will both have the AfricanAmerican, Asian, Caucasian, and Hispanic directories each containing the 400 or 200 images. The test directory will be the one without these directories and the images dumped in.

- In code we need to create variables that contain the paths to the directories. So here, train_path, valid_path, and test_path are set equal to the relative path to the directories.



```
[jma:TermProject665 Jasmine$ cd DatasetOfRaces/
[jma:DatasetOfRaces Jasmine$ ls
test    train   valid
[jma:DatasetOfRaces Jasmine$ cd train
[jma:train Jasmine$ ls
AfricanAmerican Asian           Caucasian       Hispanic
```



```
train_path = 'DatasetOfRaces/train'
valid_path = 'DatasetOfRaces/valid'
test_path = 'DatasetOfRaces/test'
```

- Batches are set equal to imageDataGenerator(), a Keras object that generates batches of tensor image data.
    1. This format is crucial for the images to be able to be read by the keras model.
    2. Calling flow_from_directory generates batches of normalized data from the path.
    3. We pass in the path variables, the target size for our images which is 256 x 256. The classes are the categories we're classifying into.
    4. The batch_size is the amount of pictures we're training over each epoch. The batch sizes should be different depending on the number of pictures that are in each directory.

```
train_batches = ImageDataGenerator().flow_from_directory(train_path, target_size=(256, 256), class
es=['AfricanAmerican', 'Asian', 'Caucasian', 'Hispanic'], batch_size=50)
valid_batches = ImageDataGenerator().flow_from_directory(valid_path, target_size=(256, 256), class
es=['AfricanAmerican', 'Asian', 'Caucasian', 'Hispanic'], batch_size=25)
test_batches = ImageDataGenerator().flow_from_directory(test_path, target_size=(256, 256), classes
=['AfricanAmerican', 'Asian', 'Caucasian', 'Hispanic'], batch_size=40)
```

- Block of code from GitHub that allows us to plot images with their labels into our Jupyter notebook. This helps to plot view our results with the images to see what we are dealing with.

```
def plots(ims, figsize=(40,24), rows=10, interp=False, titles=None):
    if type(ims[0]) is np.ndarray:
        ims = np.array(ims).astype(np.uint8)
        if(ims.shape[-1] != 3):
            ims = ims.transpose((0,2,3,1))
    f = plt.figure(figsize=figsize)
    cols = len(ims)//rows if len(ims)%2 == 0 else len(ims)//rows+1
    for i in range(len(ims)):
        sp = f.add_subplot(rows, cols, i+1)
        sp.axis('Off')
        if titles is not None:
            sp.set_title(titles[i], fontsize=16)
        plt.imshow(ims[i], interpolation=None if interp else 'none')
```

- In this batch of images we plotted from the GitHub code, we can see the images of people with different ethnicities. The numbers above the images are the labels called an encoding.
    - African Americans are [1, 0, 0, 0]
    - Asians are [0, 1, 0, 0]
    - Caucasians are [0, 0, 1 0]
    - Hispanics [0, 0, 0, 1]

```
imgs, labels = next(train_batches)
```

```
plots(imgs, titles=labels)
```

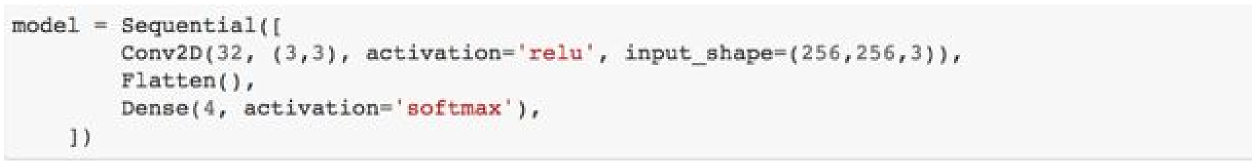
4. Solution Implementation

Sequential Model:

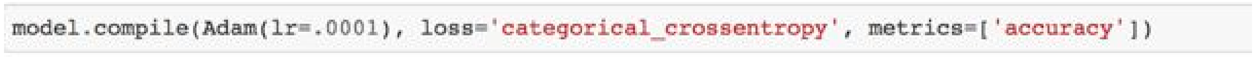
- Using a sequential model and the first layer as a Convolutional layer, we use Conv2D, a 2 dimensional used for dealing with 2D images
    - 32, is an arbitrary number which is the number of output layers in the convolution
    - (3,3) is a 2 integer tuple for kernel size, where 3,3 is specifying the width and height making our convolutional window 3 x 3
    - Our activation = “Relu”, which is the most used activation function in the world right now
    - Input_shape = (height, width, channel) is required in the first layer of any sequential model, we use 256 x 256, the same values as our image target size. The channel layer is set to 3, which signifies rgb color scale
    - Flatten, takes the output from the previous layer and flattens it into a one-dimensional tensor
    - It is then fed into the dense layer with 4 nodes, which is the output layer that is going to classify the images into one of the four categories.

- Model.compile() is used to compile our model, we do this using the Adam optimization function, with learning rate .0001, loss is equal to categorical_crossentropy, and single string of accuracy for metrics.
    - The Adam optimization algorithm is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing, this optimization function is used by many other researchers

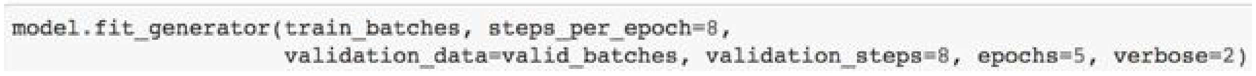
- model.fit_generator() fits the model on data generated batch by batch(what we designed earlier with the train_batches, valid_batches, test_batches variables)
    - We give the fit generator our train_batches
    - Steps_per_epoch is the size of the data set/ batch size, to figure out what it would take to iterate over the all the images in the directory one time
    - Validation data is pointed to our valid batches
    - Validation steps are the same as our steps per epoch
    - We’re running five epochs
    - Verbose is set to 2 to show how much output we want to print



```
model = Sequential([
        Conv2D(32, (3,3), activation='relu', input_shape=(256,256,3)),
        Flatten(),
        Dense(4, activation='softmax'),
    ])
```



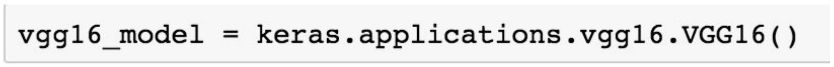
```
model.compile(Adam(lr=.0001), loss='categorical_crossentropy', metrics=['accuracy'])
```



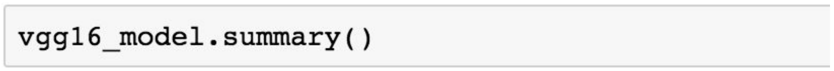
```
model.fit_generator(train_batches, steps_per_epoch=8,
                    validation_data=valid_batches, validation_steps=8, epochs=5, verbose=2)
```

VGG16 Model:

In order to be able to use the VGG16 model, we called in a VGG16 object using Keras.

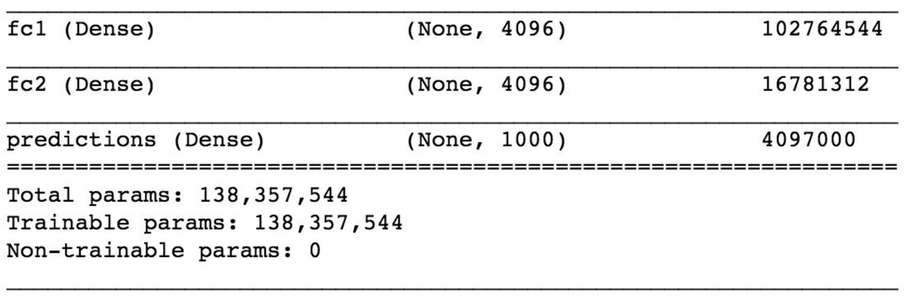


```
vgg16_model = keras.applications.vgg16.VGG16()
```



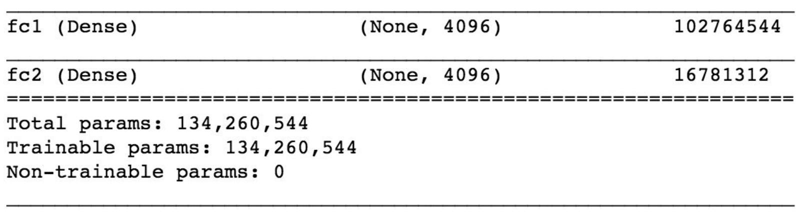
```
vgg16_model.summary()
```

This VGG16 object gave us pre-trained data and originally looked as such.

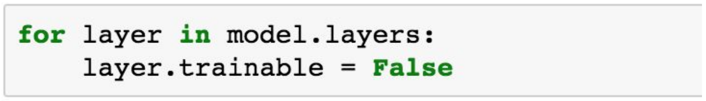


```
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
```

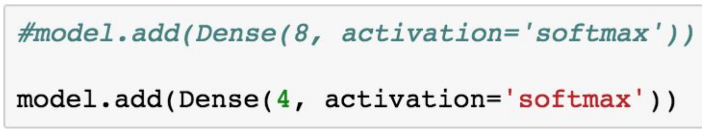
Since we planned to train our own data, we needed to remove the last layer in this pre-trained VGG16 model. We used the pop function, already provided in the VGG16 model, in order to remove the last trained layer. We then froze all layers so that way after adding our output layer to the model, we would only need to train the images within that output layer. Finally, we added our output layer using a “softmax” activation.



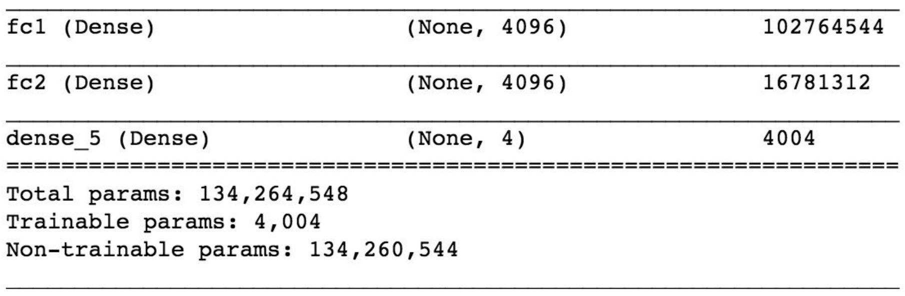
```
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
=================================================================
Total params: 134,260,544
Trainable params: 134,260,544
Non-trainable params: 0
_________________________________________________________________
```



```
for layer in model.layers:
    layer.trainable = False
```



```
#model.add(Dense(8, activation='softmax'))

model.add(Dense(4, activation='softmax'))
```



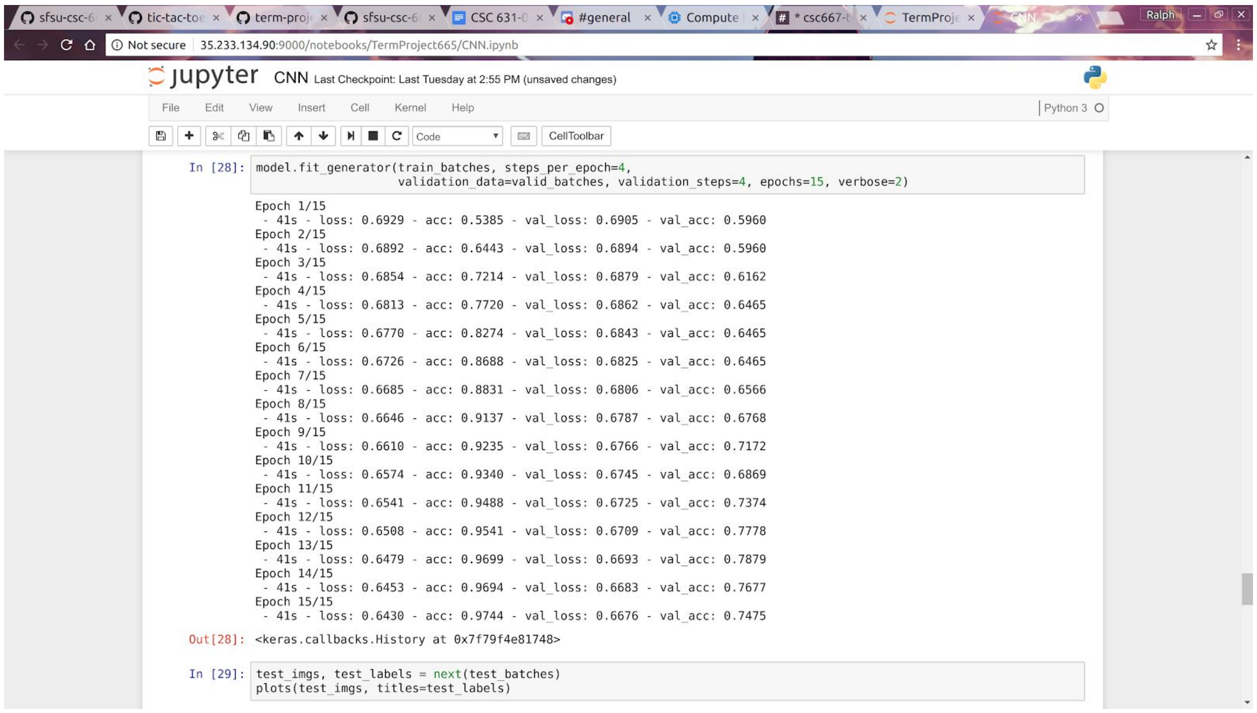
```
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544 
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312  
_________________________________________________________________
dense_5 (Dense)              (None, 4)                 4004      
=================================================================
Total params: 134,264,548
Trainable params: 4,004
Non-trainable params: 134,260,544
_________________________________________________________________
```

5. Solution Result

Out of the many iterations that we attempted on our program, we will be talking about the three.

We tried to use three main different kinds of datasets in our program: a 2 class dataset, a 4 class dataset, and an 8 class dataset.

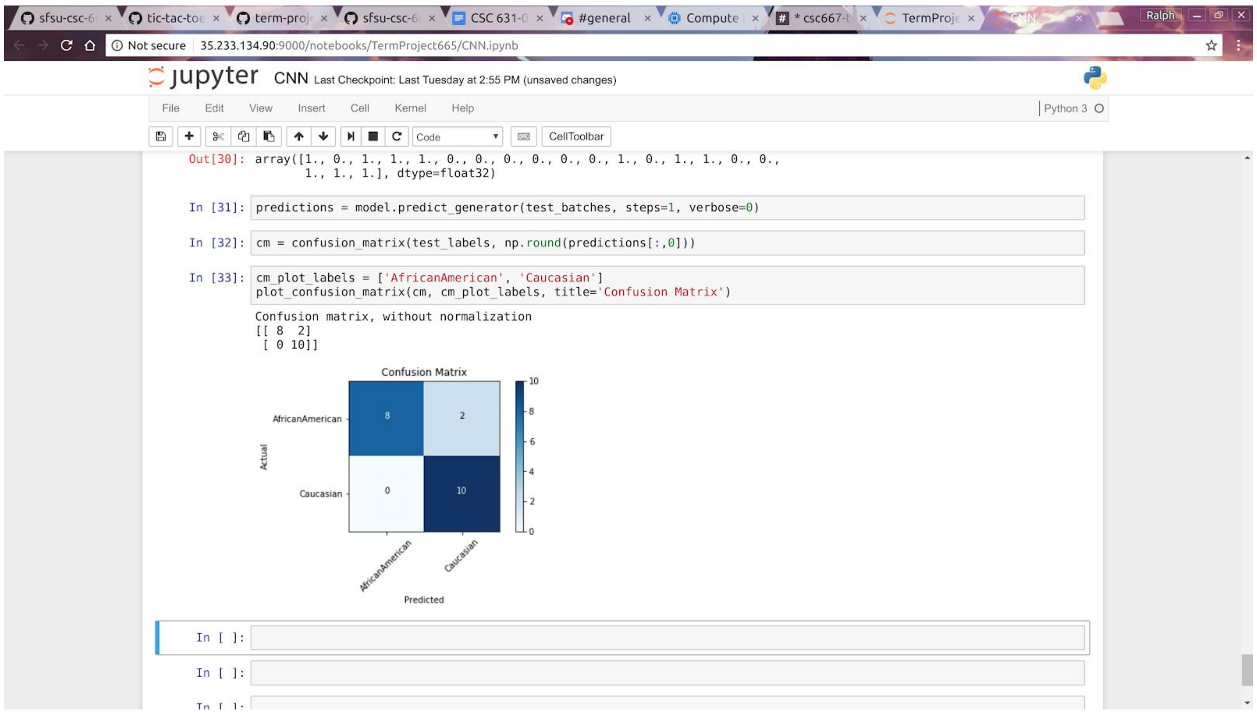
The 2 class dataset, called Racesv2.0 in our code, had two out of our four attempted ethnicities: Asian and Caucasian. This dataset was the only one to give us our desired accuracy rating of above 75% for the train and valid sets.


Not secure 35.233.134.90:9000/notebooks/TermProject665/CNN.ipynb
jupyter CNN Last Checkpoint: Last Tuesday at 2:55 PM (unsaved changes)
File Edit View Insert Cell Kernel Help
Python 3
Code
CellToolbar

```
In [28]: model.fit_generator(train_batches, steps_per_epoch=4,
                             validation_data=valid_batches, validation_steps=4, epochs=15, verbose=2)

         Epoch 1/15
          - 41s - loss: 0.6929 - acc: 0.5385 - val_loss: 0.6905 - val_acc: 0.5960
         Epoch 2/15
          - 41s - loss: 0.6892 - acc: 0.6443 - val_loss: 0.6894 - val_acc: 0.5960
         Epoch 3/15
          - 41s - loss: 0.6854 - acc: 0.7214 - val_loss: 0.6879 - val_acc: 0.6162
         Epoch 4/15
          - 41s - loss: 0.6813 - acc: 0.7720 - val_loss: 0.6862 - val_acc: 0.6465
         Epoch 5/15
          - 41s - loss: 0.6770 - acc: 0.8274 - val_loss: 0.6843 - val_acc: 0.6465
         Epoch 6/15
          - 41s - loss: 0.6726 - acc: 0.8688 - val_loss: 0.6825 - val_acc: 0.6465
         Epoch 7/15
          - 41s - loss: 0.6685 - acc: 0.8831 - val_loss: 0.6806 - val_acc: 0.6566
         Epoch 8/15
          - 41s - loss: 0.6646 - acc: 0.9137 - val_loss: 0.6787 - val_acc: 0.6768
         Epoch 9/15
          - 41s - loss: 0.6610 - acc: 0.9235 - val_loss: 0.6766 - val_acc: 0.7172
         Epoch 10/15
          - 41s - loss: 0.6574 - acc: 0.9340 - val_loss: 0.6745 - val_acc: 0.6869
         Epoch 11/15
          - 41s - loss: 0.6541 - acc: 0.9488 - val_loss: 0.6725 - val_acc: 0.7374
         Epoch 12/15
          - 41s - loss: 0.6508 - acc: 0.9541 - val_loss: 0.6709 - val_acc: 0.7778
         Epoch 13/15
          - 41s - loss: 0.6479 - acc: 0.9699 - val_loss: 0.6693 - val_acc: 0.7879
         Epoch 14/15
          - 41s - loss: 0.6453 - acc: 0.9694 - val_loss: 0.6683 - val_acc: 0.7677
         Epoch 15/15
          - 41s - loss: 0.6430 - acc: 0.9744 - val_loss: 0.6676 - val_acc: 0.7475

Out[28]: <keras.callbacks.History at 0x7f79f4e81748>

In [29]: test_imgs, test_labels = next(test_batches)
         plots(test_imgs, titles=test_labels)
```

As you can see, we reached an accuracy rating of 97% on our test set and almost 75% on our valid set.

We also had a tremendous amount of success on our predictions with this model.

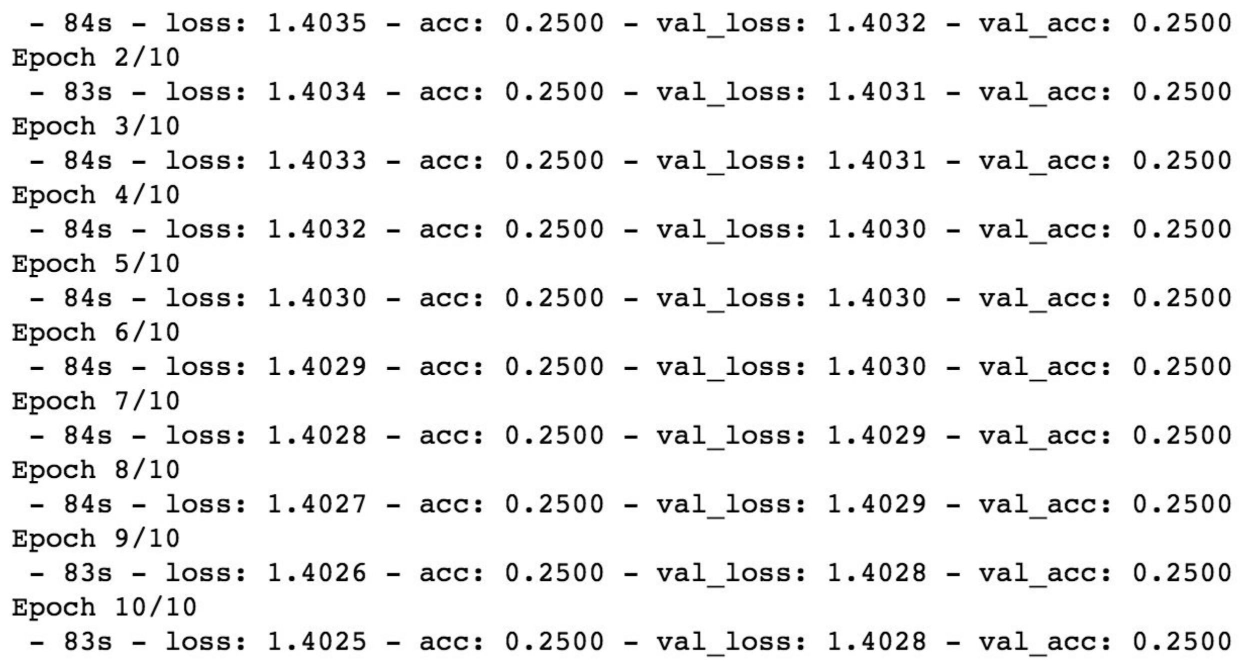


```
Out[30]: array([1., 0., 1., 1., 1., 0., 0., 0., 0., 0., 1., 0., 1., 1., 0., 0.,
                1., 1., 1.], dtype=float32)

In [31]: predictions = model.predict_generator(test_batches, steps=1, verbose=0)

In [32]: cm = confusion_matrix(test_labels, np.round(predictions[:,0]))

In [33]: cm_plot_labels = ['AfricanAmerican', 'Caucasian']
         plot_confusion_matrix(cm, cm_plot_labels, title='Confusion Matrix')

         Confusion matrix, without normalization
         [[ 8  2]
          [ 0 10]]
```

As the confusion matrix above shows, we had a 90% prediction success rate on this trained dataset.

This was very good in terms of results and we hoped to reach the same kind of success with our 4 class and 8 class datasets, but were unable to.

Our 4 class dataset, called DatasetOfRaces, was not able to reach the same level of success as our Racesv2.0 dataset and, in fact, performed much worse than expected. This dataset consisted of all four ethnicities that we were attempting to classify: African American, Asian, Caucasian, and Hispanic.



```
 - 84s - loss: 1.4035 - acc: 0.2500 - val_loss: 1.4032 - val_acc: 0.2500
Epoch 2/10
 - 83s - loss: 1.4034 - acc: 0.2500 - val_loss: 1.4031 - val_acc: 0.2500
Epoch 3/10
 - 84s - loss: 1.4033 - acc: 0.2500 - val_loss: 1.4031 - val_acc: 0.2500
Epoch 4/10
 - 84s - loss: 1.4032 - acc: 0.2500 - val_loss: 1.4030 - val_acc: 0.2500
Epoch 5/10
 - 84s - loss: 1.4030 - acc: 0.2500 - val_loss: 1.4030 - val_acc: 0.2500
Epoch 6/10
 - 84s - loss: 1.4029 - acc: 0.2500 - val_loss: 1.4030 - val_acc: 0.2500
Epoch 7/10
 - 84s - loss: 1.4028 - acc: 0.2500 - val_loss: 1.4029 - val_acc: 0.2500
Epoch 8/10
 - 84s - loss: 1.4027 - acc: 0.2500 - val_loss: 1.4029 - val_acc: 0.2500
Epoch 9/10
 - 83s - loss: 1.4026 - acc: 0.2500 - val_loss: 1.4028 - val_acc: 0.2500
Epoch 10/10
 - 83s - loss: 1.4025 - acc: 0.2500 - val_loss: 1.4028 - val_acc: 0.2500
```

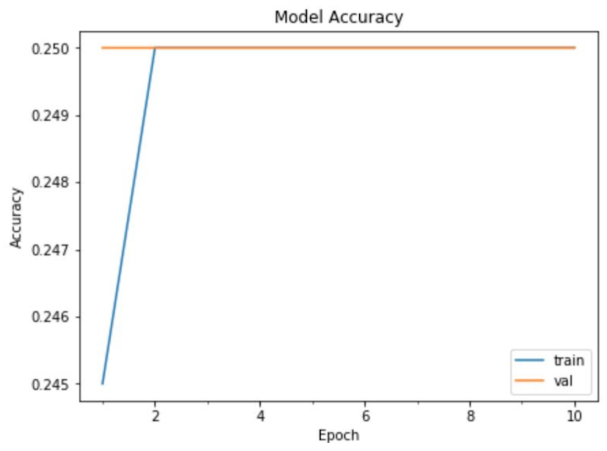
The result from this model was extremely shocking to us as we prepped the VGG16 model and it performed no better than our non-VGG16 model.

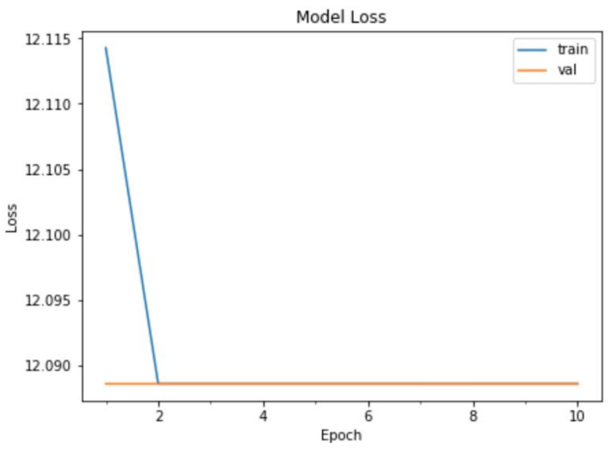
Another dataset we tried to use on our program was an 8 class dataset, which we called DatasetOfRacesGender. This dataset consisted of the four ethnicities we attempted to classify split into male and female genders.

```
 - 83s - loss: 2.0790 - acc: 0.1000 - val_loss: 2.0820 - val_acc: 0.1200
Epoch 2/10
 - 83s - loss: 2.0778 - acc: 0.1150 - val_loss: 2.0813 - val_acc: 0.1250
Epoch 3/10
 - 83s - loss: 2.0767 - acc: 0.1250 - val_loss: 2.0806 - val_acc: 0.1300
Epoch 4/10
 - 83s - loss: 2.0756 - acc: 0.1375 - val_loss: 2.0799 - val_acc: 0.1250
Epoch 5/10
 - 84s - loss: 2.0746 - acc: 0.1550 - val_loss: 2.0792 - val_acc: 0.1300
Epoch 6/10
 - 84s - loss: 2.0735 - acc: 0.1675 - val_loss: 2.0786 - val_acc: 0.1350
Epoch 7/10
 - 84s - loss: 2.0725 - acc: 0.1775 - val_loss: 2.0779 - val_acc: 0.1400
Epoch 8/10
 - 84s - loss: 2.0715 - acc: 0.2000 - val_loss: 2.0773 - val_acc: 0.1400
Epoch 9/10
 - 84s - loss: 2.0705 - acc: 0.2100 - val_loss: 2.0766 - val_acc: 0.1350
Epoch 10/10
 - 85s - loss: 2.0695 - acc: 0.2175 - val_loss: 2.0760 - val_acc: 0.1500
```

The results from this training were bad but improved much more than the training data from our 4 class dataset.

As the histogram below shows, our accuracy improved quite a bit while also decreasing the loss value.





6. Conclusion

In the end, after testing many different dataset implementations, our software, Racial Recognition, worked as well as we wanted it to for only one of our datasets, Racesv2.0. This dataset reached a prediction accuracy of 90% on the test set.

We created and attempted to use many different datasets on our program but were unable to achieve success with them. This was disappointing for us, as all of us in the group worked hard to create the datasets and the related code.

We ran into many issues that we were unable to solve in time and that prevented us from achieving the results that we wanted.

Overall, we learned a lot from this project and have a sure belief that we can do an even better job of implementing such a project in the future.

7. Peer Evaluations

Andrew Hutzel (915841776)

Evaluation Criteria	Group member: Jasmine	Group member: Ralph	Group member: Sannuj
Attends group meetings regularly and arrives on time.	4	5	4
Contributes meaningfully to group discussions.	5	5	5
Completes group assignments on time.	4	5	4
Prepares work in a quality manner.	4	4	5
Demonstrates a cooperative and supportive attitude.	5	5	4
Contributes significantly to the success of the project.	5	5	5

Ralph Acosta (916273896)

Evaluation Criteria	Group member: Jasmine	Group member: Andrew	Group member: Sannuj
Attends group meetings regularly and arrives on time.	4.8	4.8	4.8
Contributes meaningfully to group discussions.	4.5	4.7	4.9
Completes group assignments on time.	4.5	4.7	4.7
Prepares work in a quality manner.	4.5	4.7	4.7
Demonstrates a cooperative and supportive attitude.	4.7	4.7	4.7
Contributes significantly to the success of the project.	4.7	4.7	4.9

Jasmine Mann (913553685)

Evaluation Criteria	Group member: Andrew	Group member: Ralph	Group member: Sannuj
Attends group meetings regularly and arrives on time.	5	5	4
Contributes meaningfully to group discussions.	4.5	4.5	5
Completes group assignments on time.	5	5	5
Prepares work in a quality manner.	5	5	5
Demonstrates a cooperative and supportive attitude.	5	5	4.5
Contributes significantly to the success of the project.	5	5	5

Sannuj Singhal (916300065)

Evaluation Criteria	Group member: Andrew	Group member: Ralph	Group member: Jasmine
Attends group meetings regularly and arrives on time.	5	5	4.5
Contributes meaningfully to group discussions.	5	5	5
Completes group assignments on time.	5	5	4.5
Prepares work in a quality manner.	5	5	5
Demonstrates a cooperative and supportive attitude.	5	5	5
Contributes significantly to the success of the project.	5	5	5