Project Intro

My goal is to predict traffic levels from a car's perspective. I do this using a CNN architecture on a set of images that predict if there is low or medium traffic. The GUI for this project displays images from a car perspective of what is in front of it. The trained model then predicts the traffic level given the image.

GUI Use

My GUI shows the traffic prediction for each image. The image is shown to the left. Pressing the "Predict" button displays the results (Results are hard coded for now). You can go the next and previous image by hitting the "next" and "prev" button. As of right now, the only thing you need in order to run the GUI is the GUI.py file and the data folder with the images. The GUI is designed to run inside of a virtual environment. I created and tested using anaconda. Open a terminal or command prompt and type in line by line the following to create the virtual environment and install the GUIs dependencies.

1. conda create -n vm python=3.6.6
2. activate vm
3. pip install tensorflow
4. pip install keras
5. pip install Pillow
6. pip install matplotlib

To run the GUI, simply run the command "Python GUI.py". The GUI should then appear

Video: https://youtu.be/bU_lE6iKNzQ

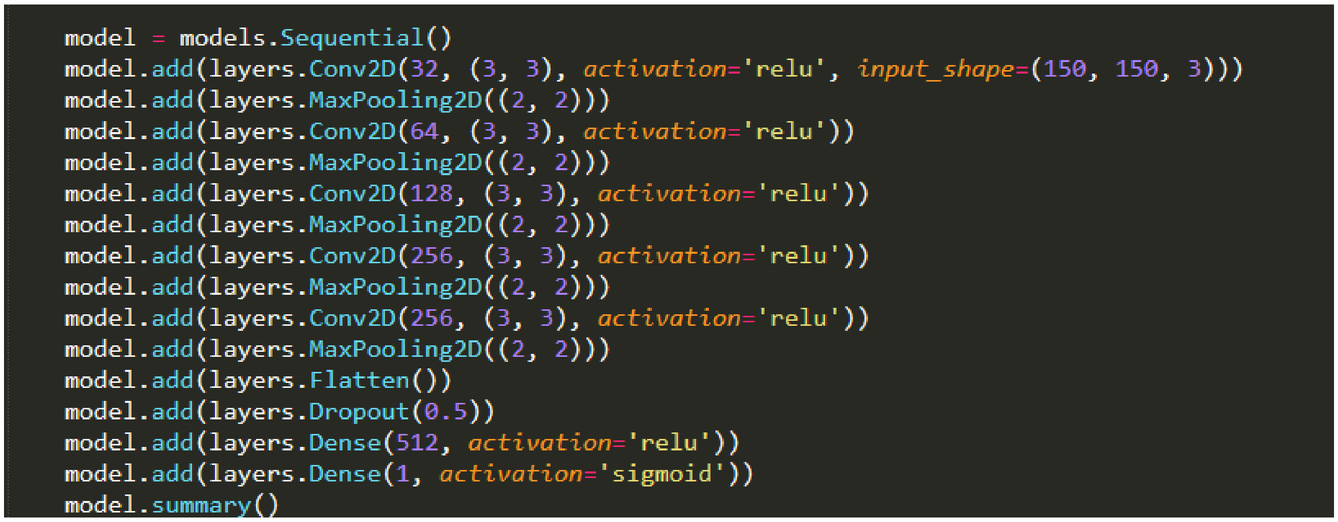
Dataset

The dataset I am using is from the Berkley Deep Drive. It consists of over 100,000 images of car windshield views. The link to the dataset is given below

Link: https://bdd-data.berkeley.edu/

Network Design

My design makes use of the CNN architecture. The network is given below.

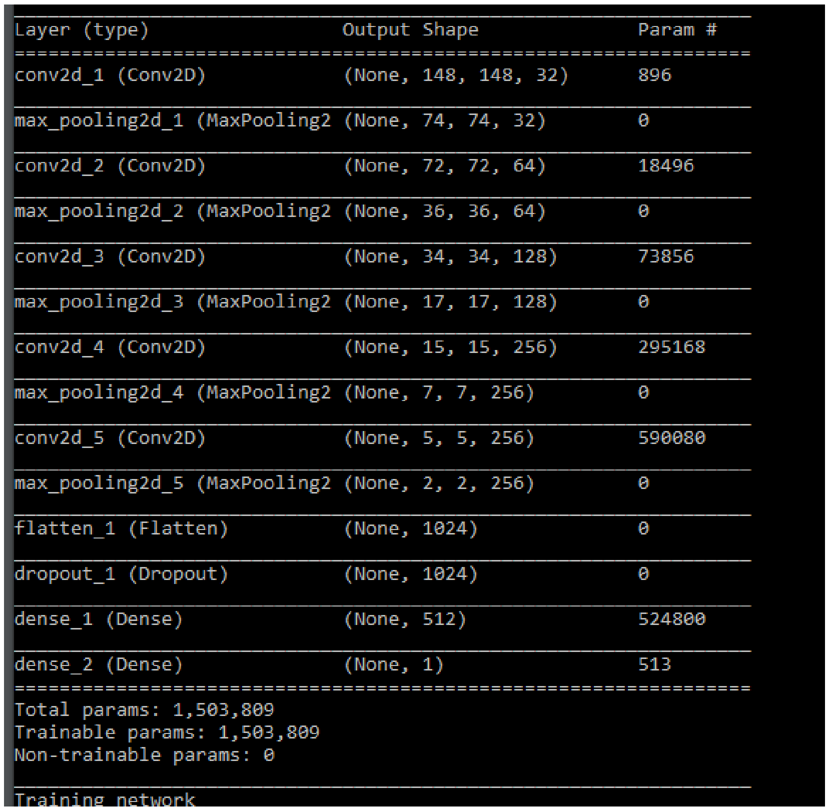


```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(256, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(256, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.summary()
```

My network consists of 5 sets of Conv2D and MaxPooling2D layers. A flatten layer is then used to flatten the output of these layers and then fed into a Dense layer. Dropout is applied to the end to fight overfitting.

Tensor Shapes

Below is a summary of the tensor shapes of each layer



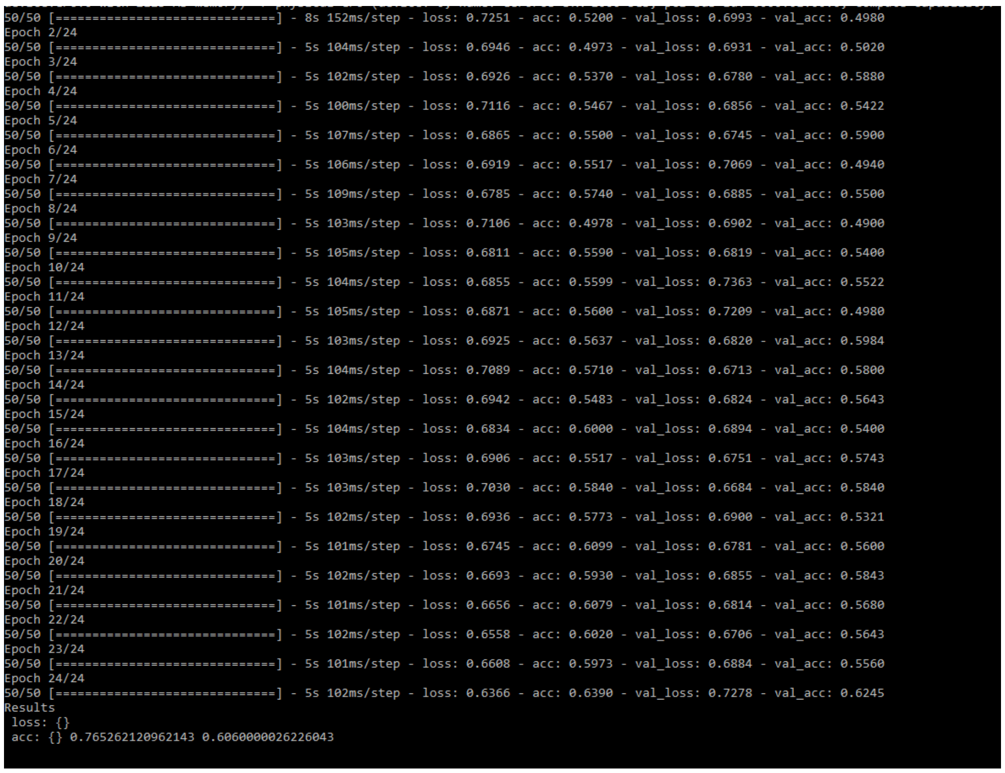
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 256)       295168
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 256)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 5, 5, 256)         590080
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 2, 2, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 1,503,809
Trainable params: 1,503,809
Non-trainable params: 0
_________________________________________________________________
Training network
```

The input to the network is a tensor of shape (20, 150, 150, 3). The output of the network is a tensor of shape (20,). The single output for each image is the probability that the traffic from the car’s perspective is low or medium.

Hyperparameters

Thus far, I have been focusing on perfecting my network model. The only hyperparameters I have really tuned have been epoch number. I have found with the current network; the optimal epoch number is. I plan to start tuning more parameters such as dropout rate once I have a set model.

Training and Test Performance



```
50/50 [==============================] - 8s 152ms/step - loss: 0.7251 - acc: 0.5200 - val_loss: 0.6993 - val_acc: 0.4980
Epoch 2/24
50/50 [==============================] - 5s 104ms/step - loss: 0.6946 - acc: 0.4973 - val_loss: 0.6931 - val_acc: 0.5020
Epoch 3/24
50/50 [==============================] - 5s 102ms/step - loss: 0.6926 - acc: 0.5370 - val_loss: 0.6780 - val_acc: 0.5880
Epoch 4/24
50/50 [==============================] - 5s 100ms/step - loss: 0.7116 - acc: 0.5467 - val_loss: 0.6856 - val_acc: 0.5422
Epoch 5/24
50/50 [==============================] - 5s 107ms/step - loss: 0.6865 - acc: 0.5500 - val_loss: 0.6745 - val_acc: 0.5900
Epoch 6/24
50/50 [==============================] - 5s 106ms/step - loss: 0.6919 - acc: 0.5517 - val_loss: 0.7069 - val_acc: 0.4940
Epoch 7/24
50/50 [==============================] - 5s 109ms/step - loss: 0.6785 - acc: 0.5740 - val_loss: 0.6885 - val_acc: 0.5500
Epoch 8/24
50/50 [==============================] - 5s 103ms/step - loss: 0.7106 - acc: 0.4978 - val_loss: 0.6902 - val_acc: 0.4900
Epoch 9/24
50/50 [==============================] - 5s 105ms/step - loss: 0.6811 - acc: 0.5590 - val_loss: 0.6819 - val_acc: 0.5400
Epoch 10/24
50/50 [==============================] - 5s 104ms/step - loss: 0.6855 - acc: 0.5599 - val_loss: 0.7363 - val_acc: 0.5522
Epoch 11/24
50/50 [==============================] - 5s 105ms/step - loss: 0.6871 - acc: 0.5600 - val_loss: 0.7209 - val_acc: 0.4980
Epoch 12/24
50/50 [==============================] - 5s 103ms/step - loss: 0.6925 - acc: 0.5637 - val_loss: 0.6820 - val_acc: 0.5984
Epoch 13/24
50/50 [==============================] - 5s 104ms/step - loss: 0.7089 - acc: 0.5710 - val_loss: 0.6713 - val_acc: 0.5800
Epoch 14/24
50/50 [==============================] - 5s 102ms/step - loss: 0.6942 - acc: 0.5483 - val_loss: 0.6824 - val_acc: 0.5643
Epoch 15/24
50/50 [==============================] - 5s 104ms/step - loss: 0.6834 - acc: 0.6000 - val_loss: 0.6894 - val_acc: 0.5400
Epoch 16/24
50/50 [==============================] - 5s 103ms/step - loss: 0.6906 - acc: 0.5517 - val_loss: 0.6751 - val_acc: 0.5743
Epoch 17/24
50/50 [==============================] - 5s 103ms/step - loss: 0.7030 - acc: 0.5840 - val_loss: 0.6684 - val_acc: 0.5840
Epoch 18/24
50/50 [==============================] - 5s 102ms/step - loss: 0.6936 - acc: 0.5773 - val_loss: 0.6900 - val_acc: 0.5321
Epoch 19/24
50/50 [==============================] - 5s 101ms/step - loss: 0.6745 - acc: 0.6099 - val_loss: 0.6781 - val_acc: 0.5600
Epoch 20/24
50/50 [==============================] - 5s 102ms/step - loss: 0.6693 - acc: 0.5930 - val_loss: 0.6855 - val_acc: 0.5843
Epoch 21/24
50/50 [==============================] - 5s 101ms/step - loss: 0.6656 - acc: 0.6079 - val_loss: 0.6814 - val_acc: 0.5680
Epoch 22/24
50/50 [==============================] - 5s 102ms/step - loss: 0.6558 - acc: 0.6020 - val_loss: 0.6706 - val_acc: 0.5643
Epoch 23/24
50/50 [==============================] - 5s 101ms/step - loss: 0.6608 - acc: 0.5973 - val_loss: 0.6884 - val_acc: 0.5560
Epoch 24/24
50/50 [==============================] - 5s 102ms/step - loss: 0.6366 - acc: 0.6390 - val_loss: 0.7278 - val_acc: 0.6245
Results
 loss: {}
 acc: {} 0.765262120962143 0.6060000026226043
```