Traffic Prediction Project

Note: Instructions on how to run the project is down below
Github repository: https://github.com/jacksondelametter/Traffic_Prediction.git

Topic

My project's function is to predict traffic levels from a vehicle'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 vehicle's perspective of what is in front of it. The trained model then predicts the traffic level given the image.

Intuition and Design

Most traffic notifying technologies today use GPS to determine traffic level. What is not used is a neural network to handle this function. An interesting idea would be for a vehicle to use computer vision in order to determine the traffic level. That vehicle could then send that information to notify other vehicles of the current traffic level.

My goal for this project was to use a vehicle's vision to determine if the traffic is low, medium, or high. I also aimed to use metrics such as vehicle count, position, and closeness to determine the traffic level. I was successful in implementing all of the metrics, but ran into a few issues along my research.

An important find that I realized was that it was difficult to categorize the traffic level into the three categories stated above. The reason for this was the vehicle's range of view (A vehicle can only see so much). Because of this, I found that getting rid of the high traffic category was sufficient.

Each vehicle only looks at other vehicles that are relatively close and in the same drivable area. This ensures that vehicles that are far away do not get incorporated in the current traffic level. It also ensures that vehicles parked or going in the opposite direction also do not get incorporated in the current traffic level. This labeling scheme though is not perfect. It cannot always determine if a car parked on the side of the street is actually parked or in traffic.

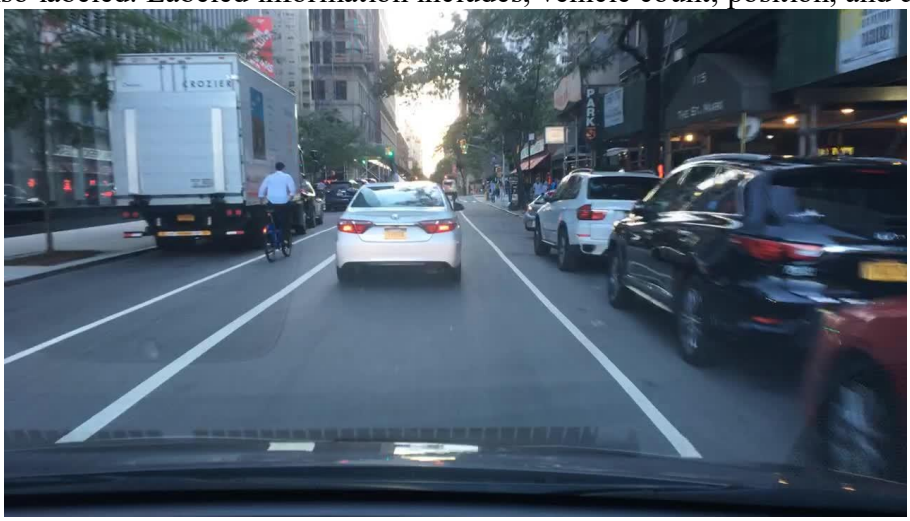
Although the labeling of images is very good, it is not perfect. A mode has been added into the GUI to label images yourself rather than relying on the automated labeling process of the program. The model is trained using these labels so it may predict a traffic level when the label is clearly wrong. In some cases, though, the trained model outperforms the labeling system. I urge you to try this mode to see its full potential.

Dataset

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

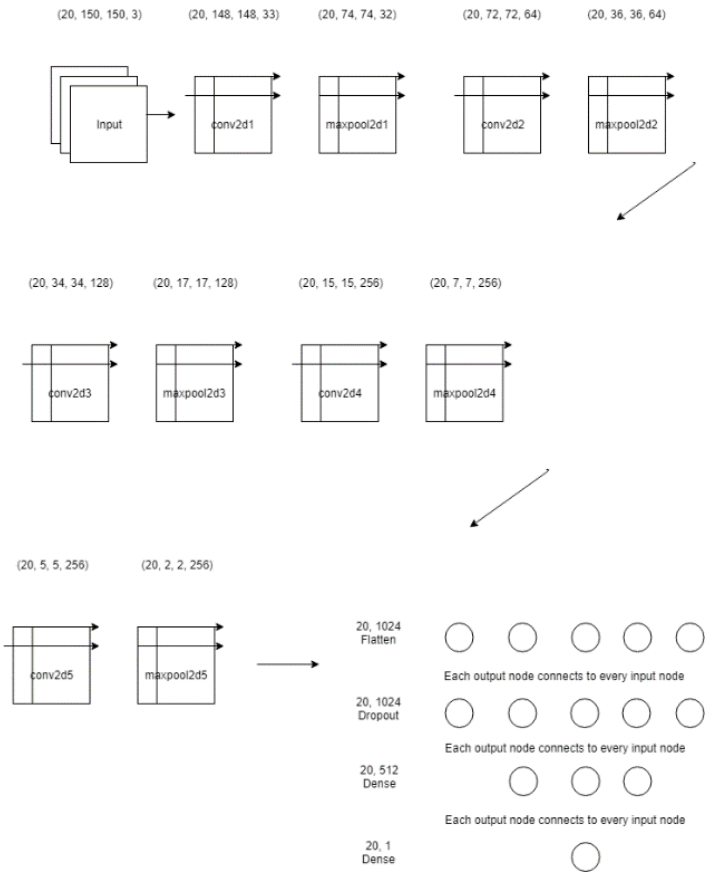
Link: https://bdd-data.berkeley.edu/

Each image is also labeled. Labeled information includes, vehicle count, position, and drivable areas.



Network Design

My design makes use of the CNN architecture. The network architecture is given below. On top of each layer represents the output tensor.



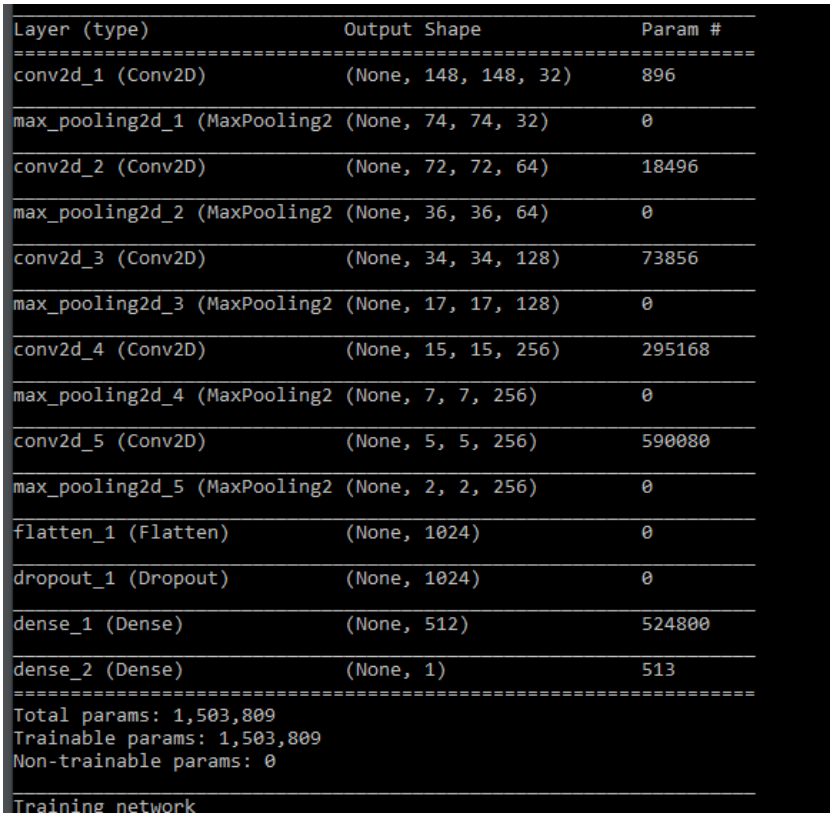
The code of the network is also given below.

```
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 five 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) where the first dimension is the batch size. The figure above gives the shape of the layers between the first and the last layers. The Conv2D layers reduces each input image size by roughly two pixels (from the padding). The MaxPooling2D layers reduce each input image size by roughly half. The number of extracted features increase as the model gets deeper. This number goes from 32 to 256. After the flattening of the batch of images. The output of the network is a tensor of shape (20, 1). The single output for each image is the probability that the traffic from the car's perspective is low or medium.

Hyperparameters

The hyperparameters chosen to tune included batch size, epoch number, and dropout rate. For batch size, I just happened to pick the optimum number (which is 20). I tried values ranging from 15-50 but most of these numbers resulted in worse test results.

For epoch number, I initially chose an epoch of 30. Although this is a relatively large number, it allowed me to see the overfitting clearly anytime I changed another hyperparameter or pre-processing attribute. After I was done changing other factors, I found that the optimum epoch number came to be 16.

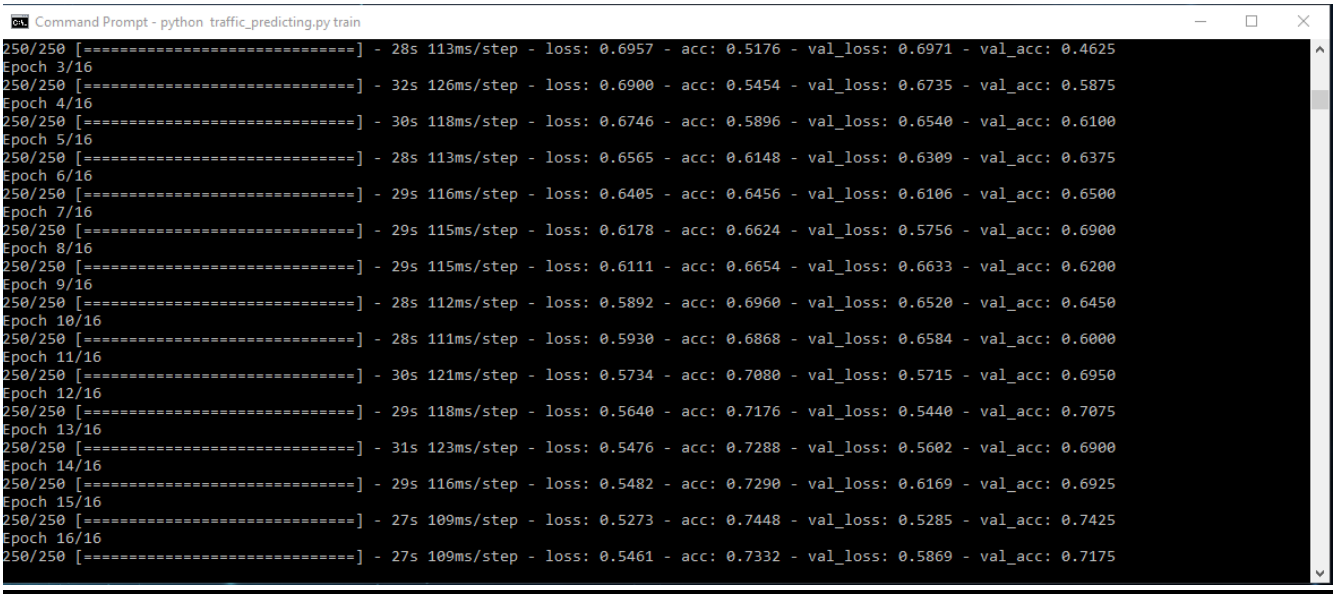
For dropout rate, I found the optimum number to be 0.5. The only other number I tried was 0.2. I found that dropping the rate resulted in more overfitting.

Pre-Processing

Although the model architecture and tuning of the hyperparameters were important to increase performance, I found that tuning the way images were pre-processed gave the biggest performance increases and gave more correctness to the labeling of images. Some tuning attributes included vehicle closeness and drivable area locations. I found that by tuning the vehicle closeness, I was able to achieve over a five percent increase. This is because the model was getting confused about vehicles being far in the distance. By determine parking areas on an image, I was able to modify drivable areas on an image. This also gave roughly a 5-10 percent increase. These modifications not only improved the model's accuracy, but gave more correctness to each image being processed. Adding more data also increased the performance. I did however find that after training on 10,000 images, the performance did not increase by much.

Training and Testing Performance

I started off with a baseline model with a test accuracy of around 50%. After tuning the model, hyperparameters, and pre-processing attributes, I was able to achieve an test accuracy of nearly 73%.



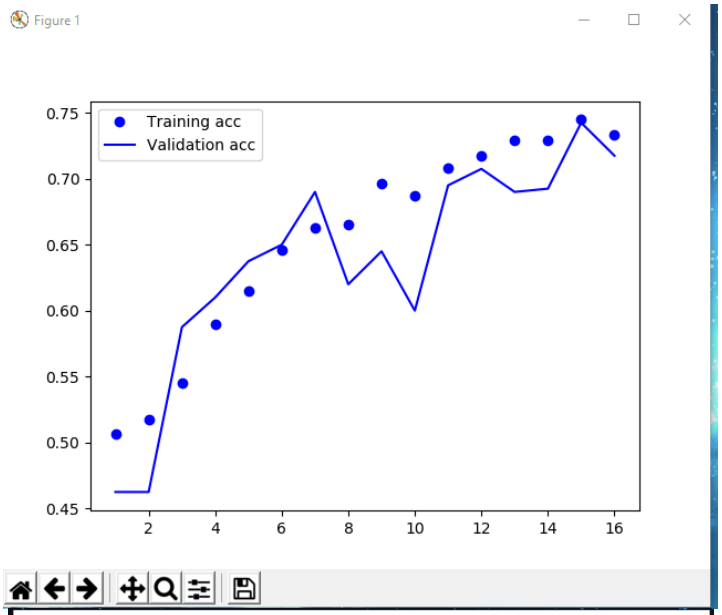


Figure 1
Training acc
Validation acc
0.75
0.70
0.65
0.60
0.55
0.50
0.45
2
4
6
8
10
12
14
16

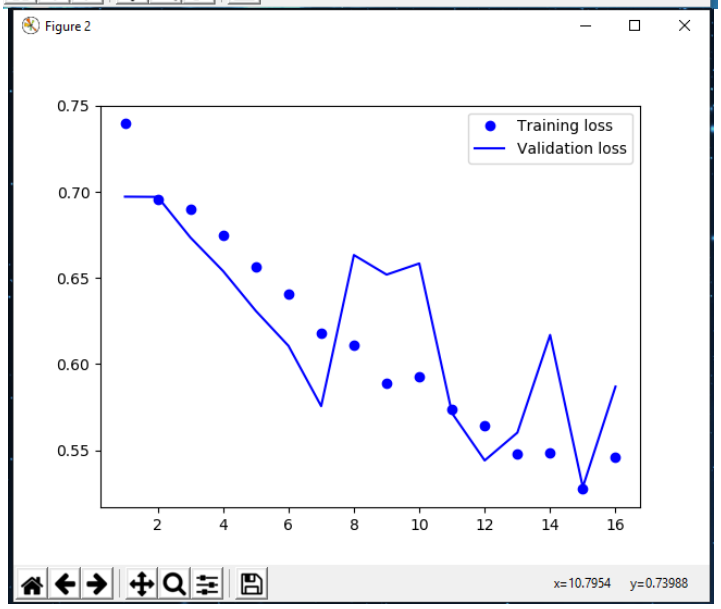
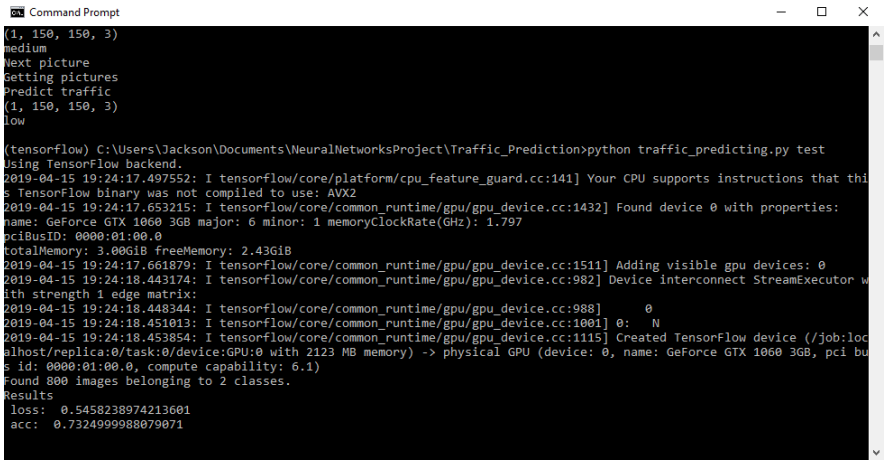


Figure 2
Training loss
Validation loss
0.75
0.70
0.65
0.60
0.55
2
4
6
8
10
12
14
16
x=10.7954
y=0.73988


Command Prompt
(1, 150, 150, 3)
medium
Next picture
Getting pictures
Predict traffic
(1, 150, 150, 3)
low
(tensorflow) C:\Users\Jackson\Documents\NeuralNetworksProject\Traffic_Prediction>python traffic_predicting.py test
Using TensorFlow backend.
2019-04-15 19:24:17.497552: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that thi
s TensorFlow binary was not compiled to use: AVX2
2019-04-15 19:24:17.653215: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1432] Found device 0 with properties:
name: GeForce GTX 1060 3GB major: 6 minor: 1 memoryClockRate(GHz): 1.797
pciBusID: 0000:01:00.0
totalMemory: 3.00GiB freeMemory: 2.43GiB
2019-04-15 19:24:17.661879: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1511] Adding visible gpu devices: 0
2019-04-15 19:24:18.443174: I tensorflow/core/common_runtime/gpu/gpu_device.cc:982] Device interconnect StreamExecutor w
ith strength 1 edge matrix:
2019-04-15 19:24:18.448344: I tensorflow/core/common_runtime/gpu/gpu_device.cc:988]   0
2019-04-15 19:24:18.451013: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1001] 0:   N
2019-04-15 19:24:18.453854: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115] Created TensorFlow device (/job:loc
alhost/replica:0/task:0/device:GPU:0 with 2123 MB memory) -> physical GPU (device: 0, name: GeForce GTX 1060 3GB, pci bu
s id: 0000:01:00.0, compute capability: 6.1)
Found 800 images belonging to 2 classes.
Results
 loss: 0.5458238974213601
 acc: 0.7324999988079071

Installation Instructions

I have tried to make the installation process as painless as possible (Unfortunately there is only so much I can do). If you have any issues with the installation process, email me at delamet@tamu.edu

Downloading the Dataset

Unfortunately, I am not able to include a portion of my dataset in the github repository. I have detailed instructions on how to download the dataset (I have talked to Dr. Jiang about this and said to leave instructions on how to download it).

1. Follow the link https://bdd-data.berkeley.edu/
2. Scroll to the bottom and click on the "Download Dataset" button
3. Login using the following credentials (Although this is my tamu email, the password is not the same as my netid password). Email: delamet@tamu.edu Password: Password1234. Alternatively, you can create an account yourself and login.
4. Click on the Downloads tab on the left
5. Under the BDD100K dropdown, click on the "Images" and "Labels" button to start downloading. (Note: these are both large downloads)

Environment Setup

I developed and tested this project using both windows and ubuntu while in an Anaconda virtual environment. Although it may work for other types of virtual environments, it is recommended to use Anaconda.

1. Create a new folder and change to that directory– mkdir traffic_pred, cd traffic_pred
2. Create a new folder – mkdir images (Note: you can put any vehicle images you like in this folder, do not change into this directory)
3. Unzip the images and labels download into the traffic_pred directory. Depending on the unzipping tool, you may get different unzipped paths, make sure the paths of both the images and labels start with the following directory structures, images: \bdd100k_images\bdd100k\images\...., labels: \bdd100k_labels_release\bdd100k\labels\...
4. Download the project from the github repository – git clone https://github.com/jacksondelametter/Traffic_Prediction.git (Note: this should put the project into a folder called Traffic_Prediction)
5. Install Anaconda, issue the following commands to create the virtual environment and install all dependencies
6. conda create -n vm python=3.6.6
7. conda activate vm
8. pip install tensorflow
9. pip install keras
10. pip install Pillow
11. pip install matplotlib

Pre-Processing Images

Before you can run the GUI on the pre-trained model or train a new model, you first must process the downloaded images.

1. Make sure you are in the directory traffic_pred
2. Change to project src code – cd Traffic_Prediction
3. Run the pre-processing – python traffic_predicting.py process
4. Let the program run, in the previous directory. Once completed you should see new directories train, test, val, and gui with images inside of them. (Note: If you get an error when the program is starting stating you don't have permission to create the directories, try running it again)
5. Note: If you encounter any other type of error, then you did not unzip the images and labels correctly

GUI Use

The directory Traffic_Prediction comes with a model and weights in order to run the GUI without training the model. My GUI shows the traffic prediction for each image. The image is shown to the left. Pressing the "Predict" button displays the prediction results from the trained model. You can go the next image by hitting the "next" button.

My GUI has two modes of operation, labeled_images mode displays the pre-processed images in the gui folder (Note: these are not the same images used for training, validating, or testing). Hitting the predict button predicts the traffic level, shows the labeled answer, and shows the accuracy of the predicted images, non_labeled_images displays non pre-processed images. Hitting the predict button only predicts the traffic level. Below show instructions on how to run the GUI

1. Make sure you are in the directory traffic_pred
2. Change to the directory Traffic_Prediction – cd Traffic_Prediction
3. For labeled_images mode – python GUI.py labeled_images (You should see the GUI appear)
4. For non_labeled_images mode – python GUI.py non_labeled_images images (Note: the second argument is the directory with the images you want to display relative to the previous directory. In the previous directory you should have created a directory called images and placed images in that directory)
5. Video for how to use the GUI is included in the github repository

Training and Testing

If you want to train and test a new model, follow the instructions below. (Note: you must have pre-processed the data first)

1. Make sure you are in the directory traffic_pred
2. Change to the directory Traffic_Prediction – cd Traffic_Prediction
3. To train the model – python traffic_predicting train (I trained using a GPU, your training time could be significantly longer)
4. To test the model – python traffic_prediction test

Annotated Code

```
# traffic_predicting.py

from keras import backend as K
import shutil
from tensorflow.python.client import device_lib
from keras.preprocessing.image import ImageDataGenerator
import os
import json
from keras import layers
from keras import models
import matplotlib.pyplot as plt
from random import sample
from keras import regularizers
from keras.models import model_from_json
import sys

# Sets up all the directory paths that will be created in procesing
os.chdir('..')
current_dir = os.getcwd()
images_path = os.path.join(current_dir, 'bdd100k_images')
images_path = os.path.join(images_path, 'bdd100k')
images_path = os.path.join(images_path, 'images')
images_path = os.path.join(images_path, '100k')
train_images_path = os.path.join(images_path, 'train')
test_images_path = os.path.join(images_path, 'test')
val_images_path = os.path.join(images_path, 'val')

labels_path = os.path.join(current_dir, 'bdd100k_labels_release')
labels_path = os.path.join(labels_path, 'bdd100k')
labels_path = os.path.join(labels_path, 'labels')
train_labels_path = os.path.join(labels_path, 'bdd100k_labels_images_train.json')
val_labels_path = os.path.join(labels_path, 'bdd100k_labels_images_val.json')

train_dir = os.path.join(current_dir, 'train')
test_dir = os.path.join(current_dir, 'test')
val_dir = os.path.join(current_dir, 'val')
gui_dir = os.path.join(current_dir, 'gui')

src_path = os.path.join(current_dir, 'Traffic_Prediction')

# Variables used to give the number of images the train, test, val and gui directories
train_dir_size = 50000
val_dir_size = 4500
test_dir_size = 4500
gui_dir_size = 1000
train_size = 5000
```

```
val_size = 400
test_size = 400
gui_size = 100

# Used to indicate the batch number for taining and testing
batch_no = 20;


def preprocess():
	'''
		preprocesses images into low and medium categories in directories train, test, val, and
gui
	'''

	# Removes and creates these directories
	print("Making train, test, and val directories")
	make_category_dirs(train_dir)
	make_category_dirs(test_dir)
	make_category_dirs(val_dir)
	make_category_dirs(gui_dir)

	# Gets the images names in the labels files for adding them into the directories stated above
	# Note the variables at the top of the file determine the sizes for each directory
	train_labels = get_labels_file(train_labels_path)
	temp_val_labels = get_labels_file(val_labels_path)
	val_labels = temp_val_labels[0:val_dir_size]
	test_labels_limit = val_dir_size + test_dir_size
	test_labels = temp_val_labels[val_dir_size:test_labels_limit]
	gui_labels_limit = test_labels_limit + gui_dir_size
	gui_labels = temp_val_labels[test_labels_limit:gui_labels_limit]

	# Categorizes pictures and adds then into the specified directory
	print("Categorizing train, test, val, and gui directories");
	categorize_images(train_labels, train_dir, train_size, train_images_path, train_dir_size)
	categorize_images(test_labels, test_dir, test_size, val_images_path, test_dir_size)
	categorize_images(val_labels, val_dir, val_size, val_images_path, val_dir_size)
	categorize_images(gui_labels, gui_dir, gui_size, val_images_path, gui_dir_size)

# Used to get a label file from the specified directory
def get_labels_file(labels_path):
	file_object = open(labels_path)
	if(file_object == None):
	   print("Could not find labels file")

	print('Found labels file')

	print('Loading json files')
	return json.load(file_object)
```

```
# Creates the given directory
def make_dir(dir):
    if(os.path.exists(dir)):
        shutil.rmtree(dir)
    os.mkdir(dir)
    print('Created director ', dir)

# Creates the given directory with the two categories in it
def make_category_dirs(dir):
    make_dir(dir)
    low_traffic_dir = os.path.join(dir, 'low')
    make_dir(low_traffic_dir)
    medium_traffic_dir = os.path.join(dir, 'medium')
    make_dir(medium_traffic_dir)

# Categorizes the given images from the labels and adds them into the given directory
def categorize_images(labels, save_dir, save_dir_size, images_dir, image_dir_size):
    low_dir = os.path.join(save_dir, 'low')
    medium_dir = os.path.join(save_dir, 'medium')
    low_traffic_max_thresh = 4
    low_traffic_num = 0
    medium_traffic_num = 0
    print('Randomizing labels')
    randomozed_labels = sample(labels, image_dir_size)
    print('Categorizing images')
    for image_value in randomozed_labels:
        image_name = image_value['name']
        image_labels = image_value['labels']
        #print("image is ", image_name)
        drivable_areas = getDrivableArea(image_labels)
        car_count = 0;
        for label in image_labels:
            if isVehicle(label, drivable_areas):
                    car_count = car_count + 1
        src = os.path.join(images_dir, image_name)
        if low_traffic_num > save_dir_size or medium_traffic_num > save_dir_size:
            break
        if car_count <= low_traffic_max_thresh and low_traffic_num < save_dir_size:
            shutil.copy(src, low_dir)
            low_traffic_num = low_traffic_num + 1
        elif car_count > low_traffic_max_thresh and medium_traffic_num < save_dir_size:
            shutil.copy(src, medium_dir)
            medium_traffic_num = medium_traffic_num + 1
    print('Categorized ', save_dir)
    print('low traffic no: ', low_traffic_num)
    print('medium traffic no: ', medium_traffic_num)

# Gets the driveable areas for an image
def getDrivableArea(image_labels):
```

```
        drivable_areas = []
        for label in image_labels:
                category = label['category']
                if category == 'drivable area':
                        attr = label['attributes']
                        area_type = attr['areaType']
                        poly2d = label['poly2d']
                        vertices = poly2d[0]
                        vertices = vertices['vertices']
                        drivable_areas.append(vertices)

        return drivable_areas

# Determines if the given vehicle is in a drivable area
def inDrivableArea(drivable_areas, box2d):
        x1_car = box2d['x1']
        x2_car = box2d['x2']
        in_left_bounds = False
        in_right_bounds = False
        # Determines if car is in left hand lane
        for area in drivable_areas:
                for vertex in area:
                        x_vertex = vertex[0]
                        if x1_car > x_vertex and x2_car > x_vertex:
                                in_left_bounds = True
                                break
        for area in drivable_areas:
                for vertex in area:
                        x_vertex = vertex[0]
                        if x1_car < x_vertex and x2_car < x_vertex:
                                in_right_bounds = True
                                break
        return in_left_bounds and in_right_bounds

# Determines if the given vehicle is close or not
def isClose(box2d):
        y_thresh = 60
        y1 = box2d['y1']
        y2 = box2d['y2']
        y_size = y2 - y1
        if(y_size <= y_thresh):
                return False
        return True

# Determines if the given label is a vehicle or not
def isVehicle(label, drivable_areas):
        category = label['category']
        close = False
        drivable = False
```

```
        if category == 'car' or category == 'truck' or category == 'bus':
                box = label['box2d']
                close = isClose(box)
                drivable = inDrivableArea(drivable_areas, box)
                return close and drivable
        return False

# Trains the network
def train_network():
        train_datagen = ImageDataGenerator(rescale=1./255)
        test_datagen = ImageDataGenerator(rescale=1./255)
        val_datagen = ImageDataGenerator(rescale=1./255)

        print('Creating image generators')
        train_generator = train_datagen.flow_from_directory(train_dir, target_size=(150, 150),
batch_size=batch_no, class_mode='binary', shuffle=True)
        val_generator = val_datagen.flow_from_directory(val_dir, target_size=(150, 150),
batch_size=batch_no, class_mode='binary', shuffle=True)
        test_generator = test_datagen.flow_from_directory(test_dir, target_size=(150, 150),
batch_size=batch_no, class_mode='binary', shuffle=True)
        test_generator = test_datagen.flow_from_directory(test_dir, target_size=(150, 150),
batch_size=batch_no, class_mode='binary', shuffle=True)

        print('Creating network model')
        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()

        model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['acc'])

        train_epoch_steps = train_size / batch_no
        val_epoch_steps = val_size / batch_no
        print("Training network")
        history = model.fit_generator(train_generator, steps_per_epoch=train_epoch_steps, epochs=16,
validation_data=val_generator, validation_steps=val_epoch_steps)
```

```
        # Saves the model and its trained weight
        os.chdir(src_path)
        model_json = model.to_json()
        with open("model.json", "w") as json_file:
                json_file.write(model_json)
        model.save_weights("model.h5")

        test_model()

        # Displays the training results
        acc = history.history['acc']
        val_acc = history.history['val_acc']
        loss = history.history['loss']
        val_loss = history.history['val_loss']

        epochs = range(1, len(acc) + 1)
        plt.plot(epochs, acc, 'bo', label='Training acc')
        plt.plot(epochs, val_acc, 'b', label='Validation acc')
        plt.legend()
        plt.figure()

        plt.plot(epochs, loss, 'bo', label='Training loss')
        plt.plot(epochs, val_loss, 'b', label='Validation loss')
        plt.legend()

        plt.show()

# Tests the model with the test data
def test_model():
        os.chdir(src_path)
        json_file = open('model.json')
        loaded_model_json = json_file.read()
        json_file.close()
        model = model_from_json(loaded_model_json)
        model.load_weights('model.h5')
        test_datagen = ImageDataGenerator(rescale=1./255)
        test_generator = test_datagen.flow_from_directory(test_dir, target_size=(150, 150),
batch_size=batch_no, class_mode='binary', shuffle=True)
        test_epoch_steps = test_size / batch_no
        model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['acc'])
        results = model.evaluate_generator(generator=test_generator, steps=test_epoch_steps)
        print('Results\n loss: ', results[0], '\n', 'acc: ', results[1], '\n')
        return results

# Program starts here
if len(sys.argv) == 0:
        print('Must have mode argument: preprocess or train')
command = sys.argv[1]
if command == 'process':
```

```
        preprocess()
elif command == 'train':
        train_network()
elif command == 'test':
        test_model()
else:
        print('Invalid argument')

# GUI.py

import os
from tkinter import *
import matplotlib.image as image
from PIL import ImageTk, Image
import matplotlib.pyplot as plt
import sys
from random import sample
from keras.models import model_from_json
import keras
import numpy as np
from keras.preprocessing import image

# Gets the model and weights
os.chdir('..')
current_dir = os.getcwd()
os.chdir('Traffic_Prediction')
json_file = open('model.json')
loaded_model_json = json_file.read()
json_file.close()
model = model_from_json(loaded_model_json)
model.load_weights('model.h5')
print('Loaded model with weights')

# Class used to make gui
class main:
    def __init__(self, master, mode):

        # Setups gui for both labeled and non labeled mode
        self.master = master
        self.mode = mode
        if mode == 'labeled_images':
            self.get_pictures()
        else:
            self.get_non_labeled_pics()
        self.picture_frame = Frame(self.master, padx=5, pady=5)
        self.picture_label = Label(self.picture_frame)
        self.picture_label.pack()
        self.picture_frame.pack(side=LEFT)
```

```
        info_frame = Frame(self.master, padx=5, pady=5)
        Label(info_frame,text="Traffic Predicting",fg="black",font=("",20,"bold")).pack(pady=10)
        self.prediction_label = Label(info_frame,text="Traffic Prediction:
None",fg="blue",font=("",20,"bold"))
        self.answer_label = Label(info_frame,text="Traffic Answer: None",fg="blue",font=("",20,"bold"))
        self.acc_label = Label(info_frame,text="Acc: None",fg="blue",font=("",20,"bold"))
        self.setup_acc()

        self.prediction_label.pack(pady=20)
        if mode == 'labeled_images':
            # If labeled mode, includes answer and accuracy in gui
            self.answer_label.pack(pady=20)
            self.acc_label.pack(pady=20)
        self.next_picture()

        arrow_frame = Frame(info_frame, pady=20)
        self.next_button = Button(arrow_frame,font=("",10),fg="white",bg="red", text="Next",
command=self.next_picture)
        self.next_button.pack(side=RIGHT)
        arrow_frame.pack(side=BOTTOM)

        Button(info_frame,font=("",15),fg="white",bg="red", text="Predict",
command=self.predict_traffic).pack(side=BOTTOM)
        info_frame.pack(side=RIGHT,fill=Y)

    # Setup for accuraccy metrics
    def setup_acc(self):
        self.correct_preds = 0;
        self.total_preds = 0;

    # Predict button was pressed, use model and weight to predict traffic
    def predict_traffic(self):
        print('Predict traffic')
        label = self.picture_dic[self.current_pic]
        if self.mode == 'labeled_images':
            pic_path = os.path.join(pic_dir, label)
            pic_path = os.path.join(pic_path, self.current_pic)
        else:
            pic_path = os.path.join(pic_dir, self.current_pic)
        pic = image.load_img(pic_path, target_size=(150, 150))
        pic_array = image.img_to_array(pic)
        pic_array = pic_array / 255
        img = np.expand_dims(pic_array, axis=0)
        print(img.shape)
        result = model.predict_classes(img)
        prediction = result[0]
        if prediction == 0:
            prediction = 'low'
        else:
```

```
            prediction = 'medium'
        print(prediction)
        if prediction == label:
            self.correct_preds = self.correct_preds + 1
        self.total_preds = self.total_preds + 1
        self.prediction_label['text'] = 'Traffic Prediction: {}'.format(prediction)
        self.answer_label['text'] = 'Traffic Answer: ' + label
        self.acc_label['text'] = 'Acc: ' + str((self.correct_preds / self.total_preds * 100)) + '%'

    # Sets the next picture in line to the picture on the gui
    def next_picture(self):
        print('Next picture')
        if self.picture_index < len(self.picture_list) - 1:
            self.picture_index += 1
            self.prediction_label['text'] = "Traffic Prediction: None"
            self.answer_label['text'] = "Traffic Answer: None"
        img = self.get_next_picture()
        self.picture_label.configure(image=img)
        self.picture_label.image = img

    # Helper method used by next_picture(), returns the next picture in line
    def get_next_picture(self):
        print('Getting pictures')
        self.current_pic = self.picture_list[self.picture_index]
        picture_label = self.picture_dic[self.current_pic]
        if self.mode == 'labeled_images':
            picture_path = os.path.join(pic_dir, picture_label)
            picture_path = os.path.join(picture_path, self.current_pic)
        else:
            picture_path = os.path.join(pic_dir, self.current_pic)
        img = ImageTk.PhotoImage(file=picture_path)
        return img

    # Gets all the pictures in the gui folder, called in labeled_mode
    def get_pictures(self):
        low_dir = os.path.join(pic_dir, 'low')
        medium_dir = os.path.join(pic_dir, 'medium')
        pics = {}
        low_pics = self.get_pictures_from_dir(low_dir, pics, 'low')
        medium_pics = self.get_pictures_from_dir(medium_dir, pics, 'medium')
        self.picture_dic = pics
        self.picture_list = sample(pics.keys(), len(pics.keys()))
        self.picture_index = -1

    # Adds all the pictures names in the given directory to the dictionary pics
    def get_pictures_from_dir(self, dir, pics, label):
        count = 0
        for pic_name in os.listdir(dir):
            if count != 100:
```

```
                pics[pic_name] = label
    # Gets all non labeled pics, called in non_labeled_mode
    def get_non_labeled_pics(self):
            pics = {}
            self.get_pictures_from_dir(pic_dir, pics, 'none')
            self.picture_dic = pics
            self.picture_list = sample(pics.keys(), len(pics.keys()))
            self.picture_index = -1

# Program starts here
if len(sys.argv) == 1:
    print('Enter mode: labeled_images or non_labeled_images')
    print("For labeled_images mode, use argument labeled_images, Demo will use processed images")
    print('For non_labeled images, use argument non_labeled_images followed by directory relative to
previous directory')
    sys.exit()
mode = sys.argv[1]

if mode == 'labeled_images':
    pic_dir = os.path.join(current_dir, 'gui')
elif mode == 'non_labeled_images':
    pic_dir = os.path.join(current_dir, sys.argv[2])
else:
    print('Enter a valid mode')

root = Tk()
main(root, mode)
root.title('Traffic detector')
root.resizable(0, 0)
root.mainloop()
```