Thomas Martin, Tran Huynh, Sahil Sutaria
CS 663
12-2-19

Project 3 LSTM Documentation: Part 4 – Live Testing

Link to YouTube Video: https://www.youtube.com/watch?v=nPw4iH9fL8A&feature=youtu.be

Link to GitHub Repository: https://github.com/tmartin293/CS663_Crosswalk_Detection

Installation Instructions:

- All Juypter Notebooks were tested using a Python 3.7.1 kernel and TensorFlow 2.0
- Please pip install or conda install all required packages prior to testing the Jupyter Notebook or the Python script: TensorFlow 2.0, cv2 (OpenCV), glob, and numpy

Screenshots:

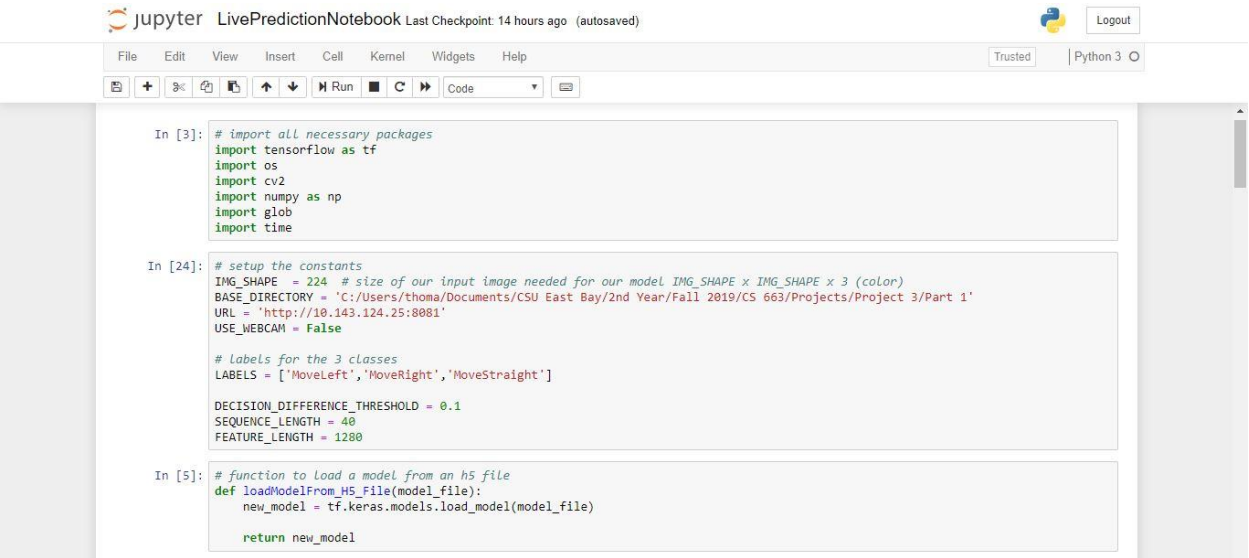

Figure 1: First 3 cells of the Jupyter Notebook

jupyter LivePredictionNotebook Last Checkpoint: 14 hours ago (autosaved)

```
In [6]: # function to create a feature extraction model (MobileNetV2 CNN)
        def CreateCNN(image_shape):
            mobilenet_v2 = tf.keras.applications.mobilenet_v2.MobileNetV2(input_shape=(image_shape,image_shape,3),
                                                                          include_top=False, weights='imagenet')

            cnn_output = mobilenet_v2.output
            pooling_output = tf.keras.layers.GlobalAveragePooling2D()(cnn_output)
            feature_extraction_model = tf.keras.Model(mobilenet_v2.input,pooling_output)

            return feature_extraction_model

In [7]: # function to re-size and pre-process an image from the camera and return a tensor
        def ProcessImage(image,image_size):
            img = tf.image.resize(image, (image_size,image_size))
            img = tf.keras.applications.mobilenet_v2.preprocess_input(img)
            img = np.expand_dims(img, axis=0)

            return img

In [8]: # function to extract all of the features from a list of properly sized images
        def ExtractFeatures(image_array,feature_extraction_model):
            all_features = []
            image_array = np.asarray(image_array)

            # loop through all the images in the image_array list and extract their features
            for i in range(image_array.shape[0]):
                features = feature_extraction_model(image_array[i])
                features = tf.reshape(features,(features.shape[0], -1))
                features = features.numpy()
                all_features.append(features)
```

Figure 2: Cells 4, 5, and 6 of the Jupyter Notebook

jupyter LivePredictionNotebook Last Checkpoint: 14 hours ago (autosaved)

```
In [8]: # function to extract all of the features from a list of properly sized images
        def ExtractFeatures(image_array,feature_extraction_model):
            all_features = []
            image_array = np.asarray(image_array)

            # loop through all the images in the image_array list and extract their features
            for i in range(image_array.shape[0]):
                features = feature_extraction_model(image_array[i])
                features = tf.reshape(features,(features.shape[0], -1))
                features = features.numpy()
                all_features.append(features)

            return all_features

In [9]: # function takes a list of feature vectors and returns a properly formatted numpy array for prediction
        def PrepareFeatures(features,sequence_length,feature_size):
            features = np.asarray(features)
            features = np.reshape(features,(1,sequence_length,feature_size))
            padded_sequence = np.zeros((1,sequence_length,feature_size))
            padded_sequence[0:feature_size] = np.array(features)

            return padded_sequence

In [10]: # function to use an LSTM model to make a prediction on live video data
         def predict(input, model):
             prediction = model.predict(input, batch_size=1, verbose=0)

             return prediction
```

Figure 3: Cells 6, 7, and 8 of the Jupyter Notebook

jupyter LivePredictionNotebook Last Checkpoint: 14 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

In [11]:
```
# function to make a classification decision and return a label
def makeDecision(predictions, class_labels):
    max_index = np.argmax(predictions)
    predictions2 = np.delete(predictions,[max_index],None)
    max_index2 = np.argmax(predictions2)

    if(max_index2 >= max_index):
        max_index2 = max_index2 + 1

    if(predictions[0][max_index] >= predictions[0][max_index2] and
      (predictions[0][max_index]-predictions[0][max_index2]) > DECISION_DIFFERENCE_THRESHOLD):
        label = class_labels[max_index]
    else:
        label = "Unknown"

    return label
```

In [12]:
```
# creates a folder to store prediction info in the specified parent directory
def CreatePredictionFolder(parent_dir,classification):
    timestr = time.strftime("%Y%m%d-%H%M%S")
    prediction_directory = classification + '_live_' + timestr
    prediction_directory = os.path.join(parent_dir,prediction_directory)
    os.mkdir(prediction_directory)

    return prediction_directory
```

Figure 4: Cells 9 and 10 of the Jupyter Notebook

jupyter LivePredictionNotebook Last Checkpoint: 14 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

In [13]:
```
# function takes a list of BGR frames and saves them to the prediction_folder
# the frames must be in BGR format as OpenCV will rearrange the channels during saving
# so that each saved frame will be in RGB format
def SavePredictionFrames(all_BGR_frames,prediction_folder):
    all_frames = np.asarray(all_BGR_frames)
    for i in range(all_frames.shape[0]):
        img_file = 'Image_' + str(i+1) + '.jpg'
        img_file = os.path.join(prediction_folder,img_file)
        frame = all_frames[i]
        cv2.imwrite(img_file,frame)
```

In [14]:
```
# creates a .txt file for all of the feature vectors used to make a prediction
def SaveVideoFeatures(prediction_dir,features):
    filename = os.path.join(prediction_dir,'FeatureVectors.txt')
    np.savetxt(filename,features[0],delimiter = ', ',newline = '\n\n\n',
               header = '\n\n', footer = '\n\n')
```

In [15]:
```
# creates a .txt file for all of the prediction results (label and prediction vector)
def SavePredictionResults(prediction_dir,prediction_results,classification):
    filename = os.path.join(prediction_dir,'PredictionResults.txt')
    np.savetxt(filename,prediction_results,delimiter = ', ',newline = '\n\n',
               header = '\n\nPrediction: ' + classification + '\n\n')
```

Figure 5: Cells 11, 12, and 13 of the Jupyter Notebook

jupyter LivePredictionNotebook Last Checkpoint: 14 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

In [16]:
```
# load the saved model and create the MobileNetV2 CNN
model_files = os.path.join(BASE_DIRECTORY, '*.h5')
model_paths = tf.io.gfile.glob(model_files)
model_file = model_paths[0]
model = loadModelFrom_H5_File(model_file)
mobilenet = CreateCNN(IMG_SHAPE)
```

```
WARNING:tensorflow:Sequential models without an `input_shape` passed to the first layer cannot reload their optimizer state. As
a result, your model isstarting with a freshly initialized optimizer.
```

In [17]:
```
# make a folder to store all of the live capture predictions if it doesn't already exist
results_directory = os.path.join(BASE_DIRECTORY,'LiveCaptureResults')
if not os.path.exists(results_directory):
    os.mkdir(results_directory)
```

In [28]:
```
# use either a webcam or a phone acting as an IP camera to capture video
if USE_WEBCAM:
    vc = cv2.VideoCapture(0)

else:
    vc = cv2.VideoCapture(URL)

current_frame = 0
max_images = SEQUENCE_LENGTH
all_images = []
all_BGR_frames = []

if vc.isOpened(): # try to get the first frame
    is_capturing, frame = vc.read()    #read frame
```

Figure 6: Cells 14, 15, and 16 of the Jupyter Notebook

jupyter LivePredictionNotebook Last Checkpoint: 14 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

In [28]:
```
# use either a webcam or a phone acting as an IP camera to capture video
if USE_WEBCAM:
    vc = cv2.VideoCapture(0)

else:
    vc = cv2.VideoCapture(URL)

current_frame = 0
max_images = SEQUENCE_LENGTH
all_images = []
all_BGR_frames = []

if vc.isOpened(): # try to get the first frame
    is_capturing, frame = vc.read()    #read frame

else:
    is_capturing = False

# continue executing as long as frames can be captured and an interrupt doesn't occur
while is_capturing:
    try:

        # capture a frame from the video capture stream and adjust its size and channel format
        is_capturing, frameorig = vc.read()
        BGR_frame = cv2.resize(frameorig,(IMG_SHAPE,IMG_SHAPE),interpolation = cv2.INTER_LINEAR)  # resize BGR frame
        RGB_frame = cv2.cvtColor(BGR_frame,cv2.COLOR_BGR2RGB)    # convert BGR to RGB

        all_BGR_frames.append(BGR_frame)
```

Figure 7: Cell 16 of the Jupyter Notebook

jupyter LivePredictionNotebook Last Checkpoint: 14 hours ago (autosaved)

```
# continue executing as long as frames can be captured and an interrupt doesn't occur
while is_capturing:
    try:

        # capture a frame from the video capture stream and adjust its size and channel format
        is_capturing, frameorig = vc.read()
        BGR_frame = cv2.resize(frameorig,(IMG_SHAPE,IMG_SHAPE),interpolation = cv2.INTER_LINEAR)  # resize BGR frame
        RGB_frame = cv2.cvtColor(BGR_frame,cv2.COLOR_BGR2RGB)    # convert BGR to RGB

        all_BGR_frames.append(BGR_frame)

        # pre-process image into a tensor and convert the underlying data from type uint8 to float32
        image = ProcessImage(RGB_frame,IMG_SHAPE)
        all_images.append(image)
        max_images -= 1

        # after getting all SEQUENCE_LENGTH frames
        if max_images == 0:
            all_features = ExtractFeatures(all_images,mobilenet)
            prediction_input = PrepareFeatures(all_features,SEQUENCE_LENGTH,FEATURE_LENGTH)
            prediction = predict(prediction_input, model)
            classification = makeDecision(prediction,LABELS)

            # make folder to store each separate prediction
            prediction_directory = CreatePredictionFolder(results_directory,classification)

            # create file to store the prediction results and the feature vectors
            SaveVideoFeatures(prediction_directory,prediction_input)
            SavePredictionResults(prediction_directory,prediction,classification)
```

Figure 8: Cell 16 of the Jupyter Notebook

jupyter LivePredictionNotebook Last Checkpoint: 14 hours ago (autosaved)

```
        # pre-process image into a tensor and convert the underlying data from type uint8 to float32
        image = ProcessImage(RGB_frame,IMG_SHAPE)
        all_images.append(image)
        max_images -= 1

        # after getting all SEQUENCE_LENGTH frames
        if max_images == 0:
            all_features = ExtractFeatures(all_images,mobilenet)
            prediction_input = PrepareFeatures(all_features,SEQUENCE_LENGTH,FEATURE_LENGTH)
            prediction = predict(prediction_input, model)
            classification = makeDecision(prediction,LABELS)

            # make folder to store each separate prediction
            prediction_directory = CreatePredictionFolder(results_directory,classification)

            # create file to store the prediction results and the feature vectors
            SaveVideoFeatures(prediction_directory,prediction_input)
            SavePredictionResults(prediction_directory,prediction,classification)

            # save all of the SEQUENCE_LENGTH frames used to make the prediction
            SavePredictionFrames(all_BGR_frames,prediction_directory)
            current_frame = 0
            all_images = []
            all_BGR_frames = []
            max_images = SEQUENCE_LENGTH

        current_frame += 1

    except KeyboardInterrupt:
        vc.release()
```

Figure 9: Cell 16 of the Jupyter Notebook

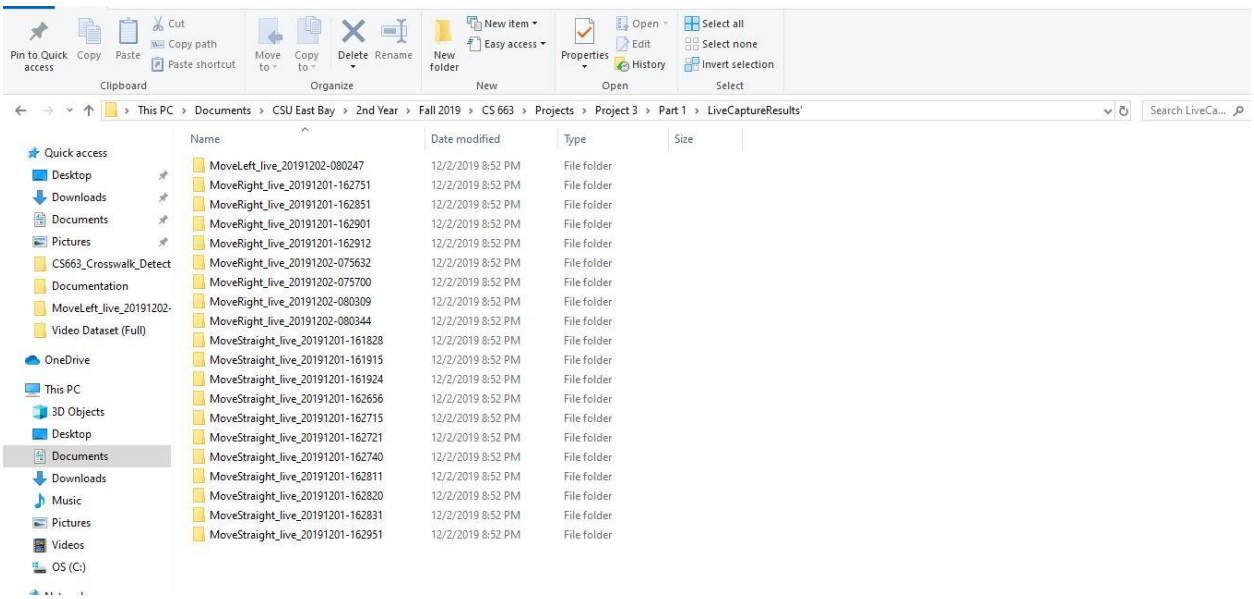

Figure 10: 20 Live prediction folders created with the Python script

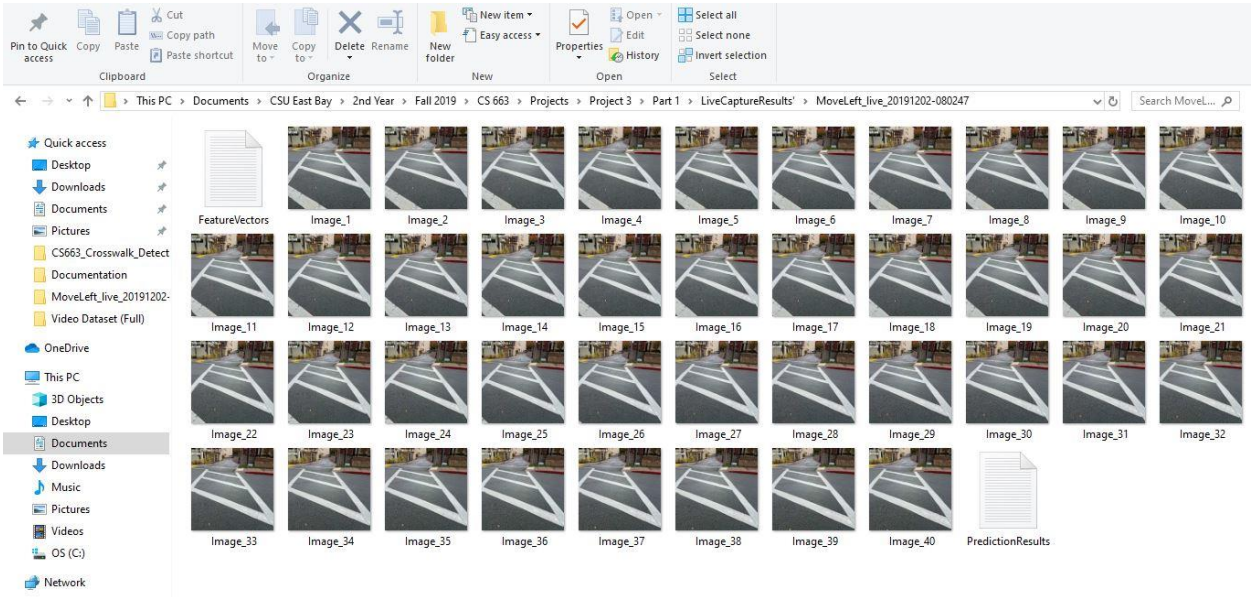

Figure 11: Contents of one of the live prediction folders (.txt file of feature vectors, .txt file of prediction results, and 40 .jpg image files)

LSTM Live Prediction Accuracy:

Of the 20 live predictions that were done using the TensorFlow LSTM model trained with 20 epochs 13 of the crosswalk navigation direction classifications were correct. These prediction results correspond to an accuracy score of 65%. There are a number of factors that could have led to our model only being able to achieve an accuracy score of 65% including: too few video samples per class, the LSTM model not being trained with a large enough number of epochs, issues with the layers of the LSTM model, and issues stemming from the IP Camera Android application.