Traffic Intensity Video Classification

Introduction

I found out from a NYTimes article that my hometown, New york City, recently passed a law giving citizens 50% of any revenue from idling fees in exchange for a three minute video of a commercial vehicle idling in the street, with the company's label easily visible. This is often above $80 per video, and NYC has thousands of idling trucks all over the city. Then the story continued, saying that the city was leaving millions of dollars of fines on the table because they didn't have the work force to process all the fines. When I read that, my first thought was that this is a reasonably solvable video-classification problem, and it would be easy to verify that a video is of an idling truck and then pick out the company label with a text processor. So, I am interested in developing a video classifcation algorithm to hunt idling commercial trucks in NYC. The final goal of this research is to develope an easy-to-use app so that users can take a video adn automatically send a fine to the city government, and get paid as fast as possible. To start this endeavor, I am building a video classifcation algorithm with the Highway Traffic Videos Dataset at https://www.kaggle.com/datasets/aryashah2k/highway-traffic-videos-dataset. This will begin the first phase of the project, learning how to classify videos.

Description of Data

This dataset is a video classification problem where the target is the intensity of traffic on the I-5 highway near Seattle. Each video was taken by cameras viewing the highway from above, from 08/05/2004 to 08/06/2004, and they are all approximately 52 frames long. I chose this data because:

- It is a video dataset, which I can use to teach me how to build a video classification pipeline.
- It has a high usability score on Kaggle. This means I shouldn't have to mess with the data too much, and can focus my time on learning how to accurately classify videos.
- It is a 3-class classifcation problem (light, medium, or heavy traffic) involving cars/automobiles. This is similar to the data I would eventually use for the idling truck-catcher, which would be a 2-class problem with commerical vehicles.

As this dataset is 63MB of video data (254 videos), I paid the $9.99 for Colab Pro. Without this, none of the below methods would've run in any useful amount of time.

Description of All the Methods Applied

Preprocessing Methods

This data comes quite clean. There are the 254 videos in *.AVI* format, and there is tab-separated *info.txt* file containing video names, class labels, and metadata such as the date and weather conditions. However, there is a serious class imbalance problem in this dataset. Of the 254 videos 165, or 65%, are light traffic. The other 35% is split evenly between medium and heavy traffic. This puts a lot of emphasis on the my kfold validation later int he code, and makes it important to use stratification in all my train-test splits.

First I used a stratified train-test-split method to split the data into 80% train, and 20% test. Making sure to stratify the data was important because it ensured that the train and test sets contained the same proportions of classes.

The only preprocessing that I did was use a feature extractor to help the models train more effectively. In this case, I used the open-source InceptionV3 feature extractor, which is a convolutional neural network takes each frame of each video and "masks" it with what it believes to be important features. The model outputs the masks for each frame, each mask containing 2048 features for the frame.

Machine Learning Methods

After using the InceptionV3 CNN feature-extractor to mask each frame of all the videos, I move onto the models. In this paper I made use of a recurrent nerual network (RNN) architecture,

because this data videos which are essentially sequences of images.

Recurrent neural networks are a cutting edge variety of neural network that, for lack of better words, is able to retain a “memory” of the data it recently processed. In this case I used GRU layers, which take into account their most recent few outputs in determining what their current uotput will be. This is iportant for my data because within one video, the model should not be flipping between light, medium, and heavy traffic. Ideally, the model will quickly realize that the video is a specific class, and hold that prediction for all the other frames.

I also spent quite a lot of time on hyperparameter tuning my models. I used the keras Hyperband tuner. Below is my model architecture including hypertuning.

```
class_vocab = label_processor.get_vocabulary()

frame_features_input = keras.Input((MAX_SEQ_LENGTH, NUM_FEATURES))

mask_input = keras.Input((MAX_SEQ_LENGTH,), dtype="bool")

x = keras.layers.GRU(units=hp.Int(name='GRU_1', min_value=12, max_value=24,

step=2),return_sequences=True)(frame_features_input,mask=mask_input)

x = keras.layers.GRU(units=hp.Int(name='GRU_2', min_value=2,

max_value=12,step=2))(x)

#Now dense/dropouts

x = keras.layers.Dropout(rate=hp.Float(name='dropout_rate',

min_value=0.2,max_value=0.4, step=0.1))(x)

x = keras.layers.Dense(units=hp.Int(name='dense_1',

min_value=4, max_value=12, step=2),activation="relu")(x)

output = keras.layers.Dense(len(class_vocab), activation="softmax")(x)

rnn_model = keras.Model([frame_features_input, mask_input], output)

rnn_model.compile(loss="sparse_categorical_crossentropy",
```

```
optimizer=keras.optimizers.Adam(), metrics=["accuracy"])
```

This model is comprised of an input layer, two GRU layers, a dropout layer, dense layer, and output dense layer. Every where there is an `hp` means I am calling the Hyperband tuner to tune that parameter in the min and max range given. For example, `Dropout(rate=hp.Float(name='dropout_rate', min_value=0.2,max_value=0.4, step=0.1))` will tune the rate parameter in the dropout layer on floats between 0.2 and 0.4, in steps of 0.1 .

I ended up making six hypertuned models, and found them somewhat inconsistent. In the end, my fourth hypertuned model performed the best in this configuration:

Layer Tuned Parameter Value GRU1 14 neurons GRU2 12 neurons Dropout rate = 0.4 Dense 8 neurons

Training, Evaluation, and Comparison

In order to train, test, and validate my models, I saved them adn wrote a few helper functions. Here is an example of loading my best models and testing them.

```
 #Rebuild tuned_model_4 its weights and the optimizer

model_4=tf.keras.models.load_model("/content/drive/MyDrive/DATA410_Final_Project/Tu

model_0=tf.keras.models.load_model("/content/drive/MyDrive/DATA410_Final_Project/Tu

#Evaluate both models

_, acc_0 = model_0.evaluate(test_data, test_labels)

print(f"Test accuracy: {round(accuracy * 100, 2)}%")

_, acc_4 = model_4.evaluate(test_data, test_labels)

print(f"Test accuracy: {round(accuracy * 100, 2)}%")

#Collect prediction probabilities and make heatmaps of confusion matrices

probs_0 = model_0.predict(test_data, batch_size=BATCH_SIZE)
```

```
probs_4= model_4.predict(test_data, batch_size=BATCH_SIZE)

Plot_Heatmaps(probs_0,probs_4)
```

Confusion Matrix of a tuned model vs an untuned model

In this case, model_4 performed *very* well. However, it's performance did worry me about the potential that it could be over fitting to the imbalanced data, and only paying attention to the "light" class. So next I did some Kfold validation!

Stratified KFold Validation

Here the KFold validation loop I ran on my two best models.

```
 accs_0 = []

accs_4 = []

path_0 = "/content/drive/MyDrive/DATA410_Final_Project/Tuned_Models/Tuned_model_0.h

path_4 = "/content/drive/MyDrive/DATA410_Final_Project/Tuned_Models/Tuned_model_4.h

#this is the random state cross-validation loop to make sure our results are real

for i in range(12345,12356):

print('Random State: ' + str(i))

kf = StratifiedKFold(n_splits=5,shuffle=True,random_state=i)

for idxtrain, idxtest in kf.split(train_data[0],train_labels.flatten()):

#Split the train and test data

xtrain = (kfold_data[0][idxtrain],kfold_data[1][idxtrain])
```

```
ytrain = kfold_labels[idxtrain]

ytest = kfold_labels[idxtest]

xtest = (kfold_data[0][idxtest],kfold_data[1][idxtest])

#Model 0

model_0=tf.keras.models.load_model(path_0)

model_0.fit(xtrain, ytrain,

validation_split=0.2, epochs=25, verbose=0)

#test the model and get the test accuracy

_, acc_0 = model_0.evaluate(xtest, ytest, verbose=0)

#Model 4

model_4=tf.keras.models.load_model(path_4)

model_4.fit(xtrain, ytrain,

validation_split=0.2, epochs=25, verbose=0)

#test the model and get the test accuracy

_, acc_4 = model_4.evaluate(xtest, ytest, verbose=0)

#Append each accuracy

accs_0.append(acc_0)

accs_4.append(acc_4)
```

The above loop stratifies the full dataset into five train and test folds, making sure each fold as teh same proportion of each class. It then trains each model on the train fold, tests it on the test fold, and appends the test accuracies to pre-initialized lists. Then we repeat this process 10 times for 10 different random states. Ideally, this validation would happen with lots and lots of data,

and would be run for as many random states as it takes to convince the user. Below I show a histogram of the accuracies from of these validation loops.

55 KFold Accuracies for model_0 and model_4

After seeing the histogram of accuracies I am more confident that model_4 really is a fantastic model! It reliably reaches extremely high accuracies.

Discussion and Inferences

As this paper was supposed to be a test case for a future audio-video classifier, I am very happy with these results. I never thought that I would be able to converge on such great video-classification algorithm so quickly. Next steps for this project are to start introducing audio classifcation in parallel, so that a future version would be able to identify a truck that is actively idling, not just wether a truck is in the frame or not. Of course, this model could go much further with more data, and I am excited to collect more data for the idling truck identifier!

References

- Video Classification with a CNN-RNN Architecture by Sayak Paul https://colab.research.google.com/github/keras-team/keras-io/blob/master/examples/vision/ipynb/video_classification.ipynb#scrollTo=7-4EPFk5-n_J
- highway_view_some_frames by LESHABIRUKOV, https://www.kaggle.com/code/leshabirukov/highway-view-some-frames