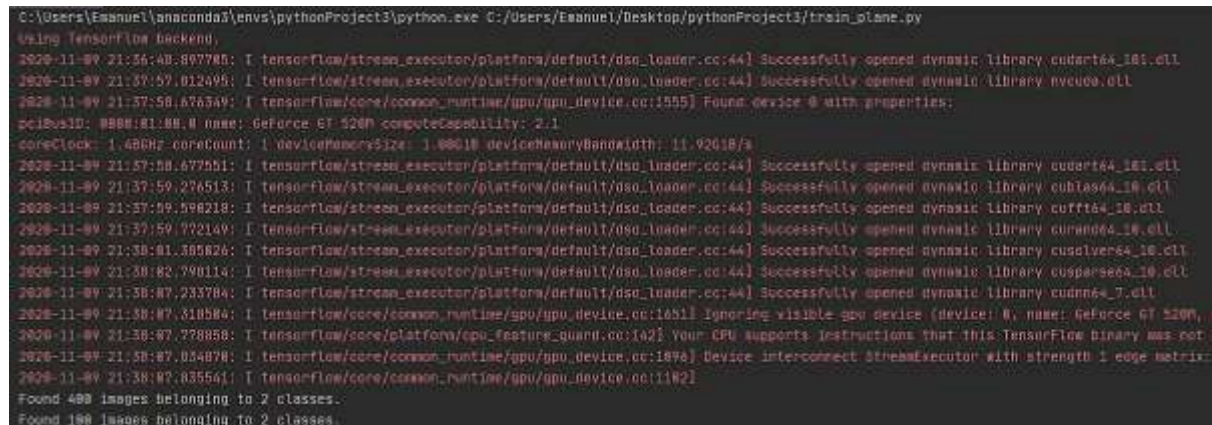


## Image Recognition with Machine Learning on Python

### Comments Regarding the Solution – Experimental Results

My main scope was to train a model from scratch in order to be able then to classify the data containing cars and planes. I achieved this, by using Convolutional Neural Networks (CNN).



```
C:\Users\Emanuel\anaconda2\envs\pythonProject3\python.exe C:/Users/Emanuel/Desktop/pythonProject3/train_plane.py
Using TensorFlow backend.
2020-11-09 21:56:48.897785: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cudart64_101.dll
2020-11-09 21:57:59.012495: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cublas44_10.dll
2020-11-09 21:57:59.676349: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1555] Found device 0 with properties:
pciBusID: 0000:81:00.0 name: GeForce GTX 520M computeCapability: 2.1
coreClock: 1.48GHz coreCount: 1 deviceMemorySize: 1.00GiB deviceMemoryBandwidth: 11.92GiB/s
2020-11-09 21:57:59.677551: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cudart64_101.dll
2020-11-09 21:57:59.276513: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cublas44_10.dll
2020-11-09 21:57:59.598218: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cufft64_10.dll
2020-11-09 21:57:59.772149: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library curand64_10.dll
2020-11-09 21:58:01.305826: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cusolver64_10.dll
2020-11-09 21:58:02.790114: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cusparse64_10.dll
2020-11-09 21:58:07.233784: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cudnn64_7.dll
2020-11-09 21:58:07.310584: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1631] Ignoring visible gpu device (device: 0, name: GeForce GTX 520M,
2020-11-09 21:58:07.728858: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not
2020-11-09 21:58:07.834078: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1894] Device interconnect StreamExecutor with strength 1 edge matrix:
2020-11-09 21:58:07.835541: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1182]
Found 400 images belonging to 2 classes.
Found 100 images belonging to 2 classes.
```

**Fig. 1:** Using TensorFlow Backend

As it can be seen in the **Fig. 1**, I've also used TensorFlow backend and some Keras libraries to perform the train of the data considered: a total of 400 images, 200 of them containing cars and 200 with planes. Then I validate the model on a set of 100 images, 50 of cars and 50 of planes.

I considered it important to have some validation data for my model, because the neural network possesses enough power to distinguish patterns even in random noise, so by seeing that it is able to get the correct answer on the *new* data (I say *new*, as some data that it was not provided with during the training stage). Also, in this way we can make sure that the patterns learned by it are not overly specific to the training data, but instead they prove truly useful to us.

We can see in the **Fig. 2**, that after the first two epochs, the *acc* and *val\_acc* numbers are similar. This is a good sign. It means that the rules that our network learns on the training data generalize well to the unseen validation (test) data. After two epochs, our network can predict whether a review is positive or negative correctly 82.15 percent of the time! After the ninth epoch, results are even better, see **Fig. 3**.

After the tenth epoch, the network is starting to memorize the training examples, with rules that are too specific. We can see in **Fig. 4** that it gets 86.25% accuracy on the training data, but only 79.76% on the held-out validation data. This means that our

network has *overfitted*, and we'd want to retrain it (by running the *compile* and *fit* steps again) for a lower number of epochs.

```
Epoch 3/10
1/25 [>.....] - ETA: 25s - loss: 0.6264 - accuracy: 0.7500
2/25 [=>.....] - ETA: 24s - loss: 0.5266 - accuracy: 0.8125
3/25 [==>....] - ETA: 23s - loss: 0.4598 - accuracy: 0.8542
4/25 [===>...] - ETA: 22s - loss: 0.4238 - accuracy: 0.8594
5/25 [====>..] - ETA: 21s - loss: 0.4357 - accuracy: 0.8258
6/25 [=====] - ETA: 20s - loss: 0.4258 - accuracy: 0.8333
7/25 [=====>.] - ETA: 19s - loss: 0.3961 - accuracy: 0.8571
8/25 [=====] - ETA: 18s - loss: 0.3886 - accuracy: 0.8594
9/25 [=====>.] - ETA: 17s - loss: 0.6539 - accuracy: 0.8194
10/25 [=====] - ETA: 16s - loss: 0.6448 - accuracy: 0.8125
11/25 [=====>.] - ETA: 15s - loss: 0.6487 - accuracy: 0.7955
12/25 [=====] - ETA: 13s - loss: 0.6257 - accuracy: 0.7949
13/25 [=====>.] - ETA: 12s - loss: 0.6091 - accuracy: 0.7981
14/25 [=====] - ETA: 11s - loss: 0.5925 - accuracy: 0.8088
15/25 [=====>.] - ETA: 10s - loss: 0.5878 - accuracy: 0.8088
16/25 [=====] - ETA: 9s - loss: 0.5788 - accuracy: 0.7852
17/25 [=====>.] - ETA: 8s - loss: 0.5737 - accuracy: 0.7831
18/25 [=====] - ETA: 7s - loss: 0.5810 - accuracy: 0.7778
19/25 [=====>.] - ETA: 6s - loss: 0.5643 - accuracy: 0.7862
20/25 [=====] - ETA: 5s - loss: 0.5549 - accuracy: 0.7937
21/25 [=====>.] - ETA: 4s - loss: 0.5574 - accuracy: 0.7857
22/25 [=====] - ETA: 3s - loss: 0.5529 - accuracy: 0.7841
23/25 [=====>.] - ETA: 2s - loss: 0.5385 - accuracy: 0.7935
24/25 [=====] - ETA: 1s - loss: 0.5296 - accuracy: 0.7969
25/25 [=====] - 28s 1s/step - loss: 0.5215 - accuracy: 0.8000 - val_loss: 0.2189 - val_accuracy: 0.8214
```

Fig. 2: 3<sup>rd</sup> Epoch Results

```
Epoch 9/10
1/25 [>.....] - ETA: 25s - loss: 0.5575 - accuracy: 0.7500
2/25 [=>.....] - ETA: 24s - loss: 0.4156 - accuracy: 0.8438
3/25 [==>....] - ETA: 23s - loss: 0.4175 - accuracy: 0.8125
4/25 [===>...] - ETA: 22s - loss: 0.3589 - accuracy: 0.8438
5/25 [====>..] - ETA: 21s - loss: 0.3154 - accuracy: 0.8625
6/25 [=====] - ETA: 20s - loss: 0.3886 - accuracy: 0.8646
7/25 [=====>.] - ETA: 19s - loss: 0.3611 - accuracy: 0.8661
8/25 [=====] - ETA: 18s - loss: 0.3656 - accuracy: 0.8594
9/25 [=====>.] - ETA: 17s - loss: 0.3593 - accuracy: 0.8611
10/25 [=====] - ETA: 16s - loss: 0.3523 - accuracy: 0.8625
11/25 [=====>.] - ETA: 15s - loss: 0.3558 - accuracy: 0.8523
12/25 [=====] - ETA: 13s - loss: 0.3378 - accuracy: 0.8542
13/25 [=====>.] - ETA: 12s - loss: 0.3392 - accuracy: 0.8518
14/25 [=====] - ETA: 11s - loss: 0.3368 - accuracy: 0.8527
15/25 [=====>.] - ETA: 10s - loss: 0.3265 - accuracy: 0.8583
16/25 [=====] - ETA: 9s - loss: 0.3226 - accuracy: 0.8633
17/25 [=====>.] - ETA: 8s - loss: 0.3343 - accuracy: 0.8493
18/25 [=====] - ETA: 7s - loss: 0.3261 - accuracy: 0.8576
19/25 [=====>.] - ETA: 6s - loss: 0.3228 - accuracy: 0.8586
20/25 [=====] - ETA: 5s - loss: 0.3484 - accuracy: 0.8531
21/25 [=====>.] - ETA: 4s - loss: 0.3391 - accuracy: 0.8542
22/25 [=====] - ETA: 3s - loss: 0.3285 - accuracy: 0.8688
23/25 [=====>.] - ETA: 2s - loss: 0.3352 - accuracy: 0.8587
24/25 [=====] - ETA: 1s - loss: 0.3298 - accuracy: 0.8646
25/25 [=====] - 28s 1s/step - loss: 0.3236 - accuracy: 0.8675 - val_loss: 0.8565 - val_accuracy: 0.8929
```

Fig. 3: 9<sup>th</sup> Epoch Results

```
25/25 [=====] - 28s 1s/step - loss: 0.2912 - accuracy: 0.8625 - val_loss: 0.6293 - val_accuracy: 0.7976
```

Fig. 4: 10<sup>th</sup> Epoch Results

Comparing our results to a **Support Vector Machine** classifier

CNN is a good choice when talking about image detection. You could definitely use CNN for sequence data, but they shine in going through huge amount of image and finding non-linear correlations. SVM are margin classifier and support different kernels to perform these classifications.

On our dataset, the SVM is significantly faster. We need about three minutes to vectorize the reviews, transforming each of our reviews into a vector containing a greater number of features. It takes some minutes to train the classifier on 400 reviews and validate on the other 100. For the given task, the SVM actually performs slightly better than the firstly used convolutional neural network, in terms of accuracy, and it does so in significantly less time.

Method	CNN	SVM
Affected time	<b>7 minutes</b>	<b>5 minutes</b>
Accuracy	<b>82.15 %</b>	<b>84.75 %</b>
Overall Performance	<b>4/5</b>	<b>3/5</b>

