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).

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 posses 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 nineth 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.

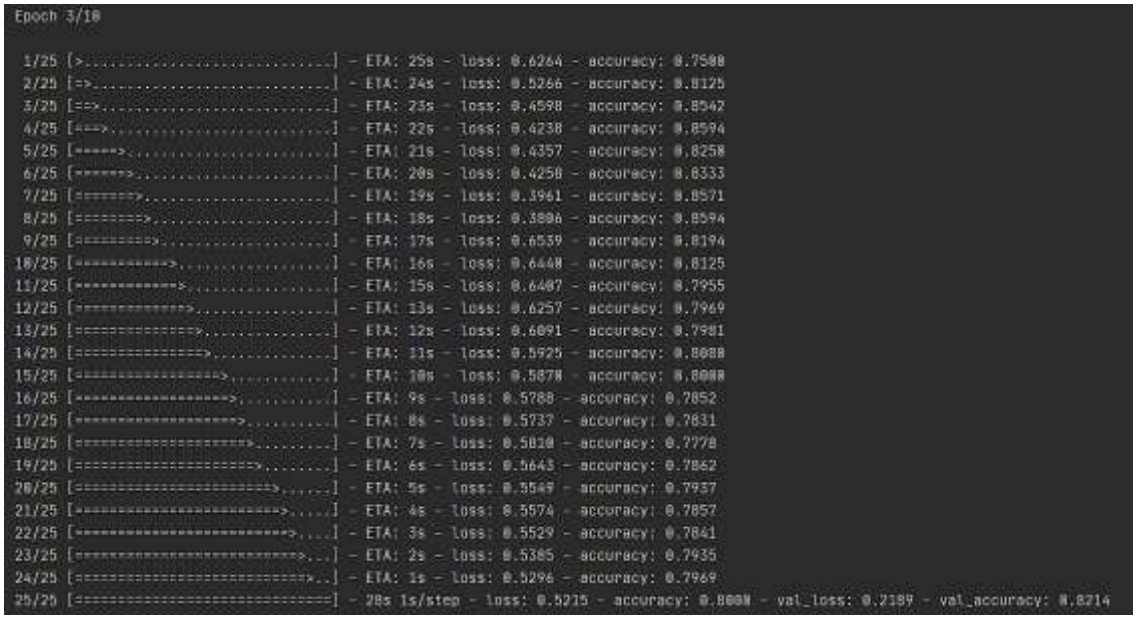

Fig. 2: 3rd Epoch Results

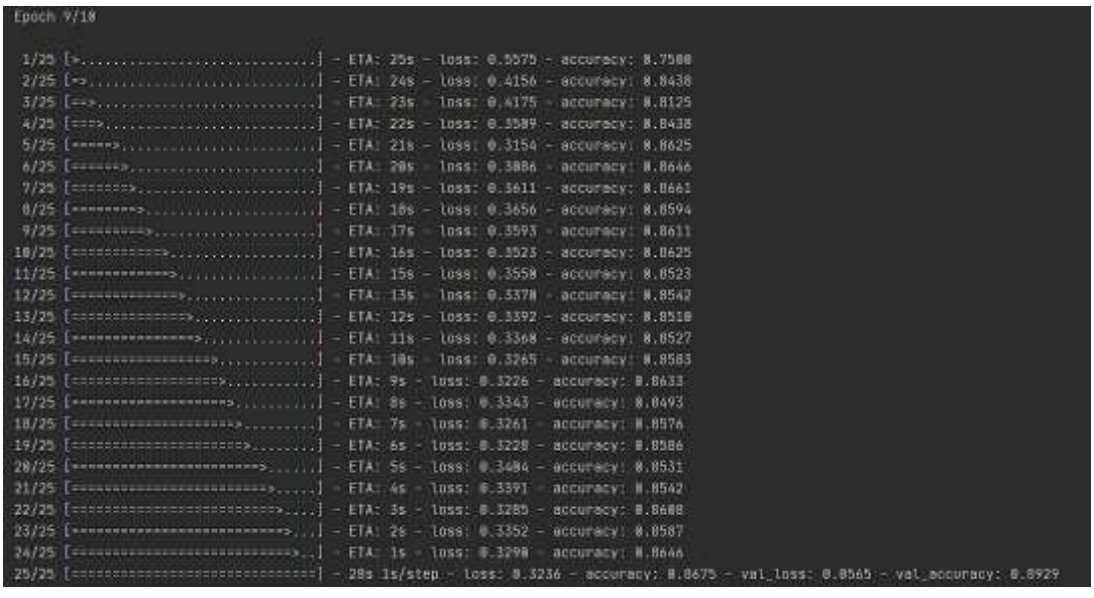

Fig. 3: 9th Epoch Results

Fig. 4: 10th 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 to 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	**7 minutes**	**5 minutes**
Accuracy	**82.15 %**	**84.75 %**
Overall Performance	**4/5**	**3/5**