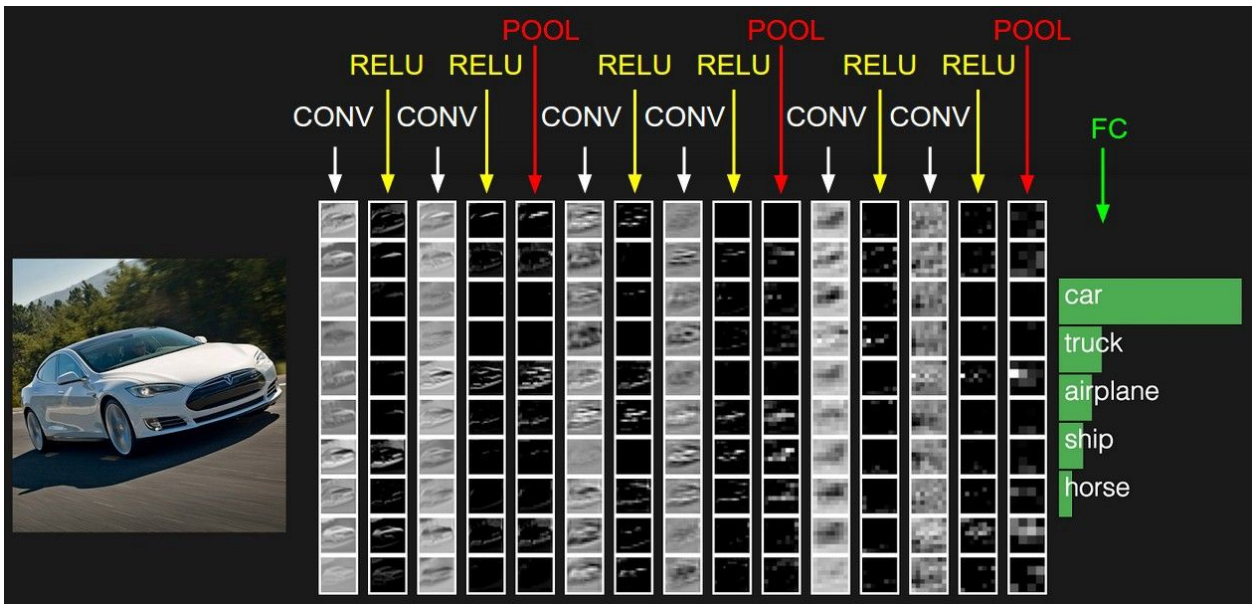
COL341

ASSIGNMENT - 2 B
REPORT



Introduction

For this part, I implemented two CNN models; (1) General model, (2) Inception Net architecture. Finally, I submitted the former model due to better results under the given constraints.

While creating the above mentioned architectures, I experimented with many features in Keras, like data augmentation, optimizers, activations, kernel initializer/regularizers, and many more. Eventually used the features that improved the model.

General Model

The overall model details are summarised below. The block diagram for the same can be seen on next page.

Model Architecture :

Conv2D layers : 8

Max Pooling layers : 5

Batch Normalisation : 5

Fully Connected layers : 3

Dropout layers : 5

Activation used : ReLu

Epochs : 15

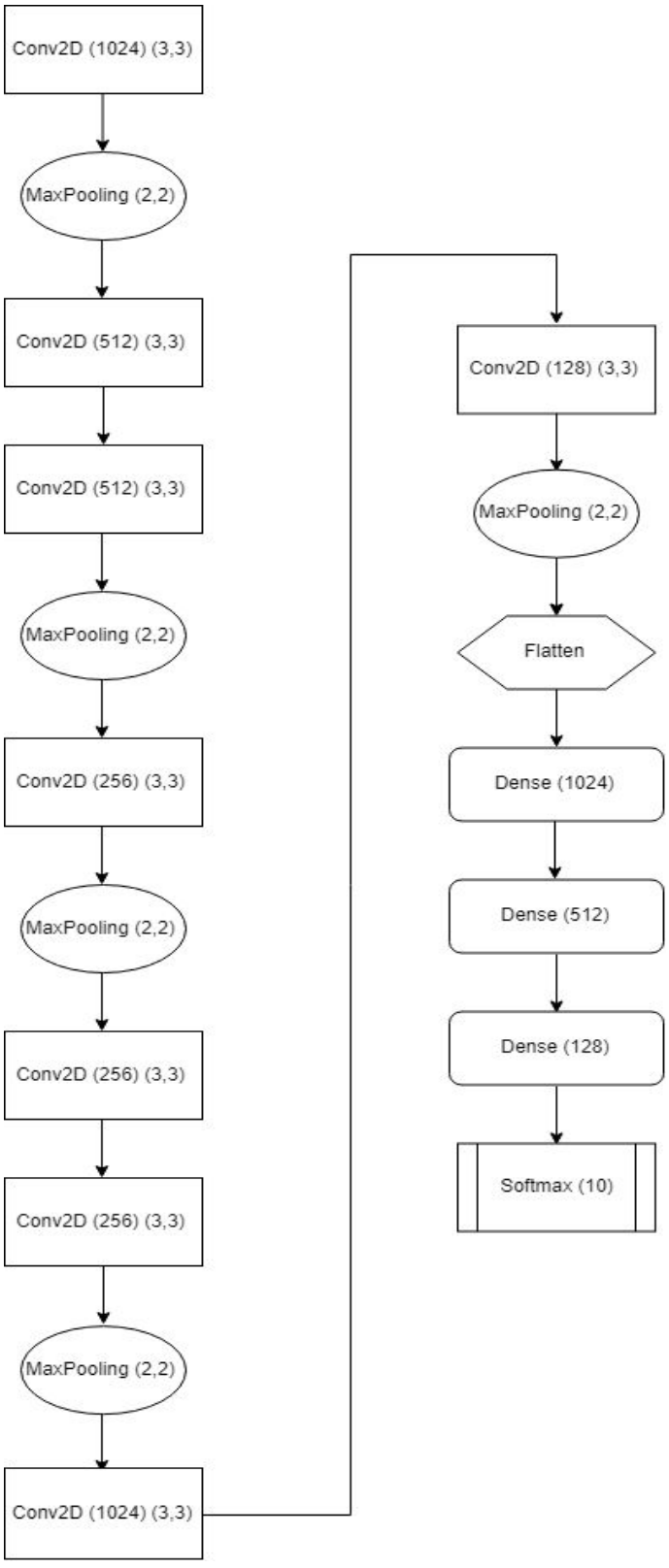
Training set : 40 000 Validation set : 10 000

Optimiser : Adam

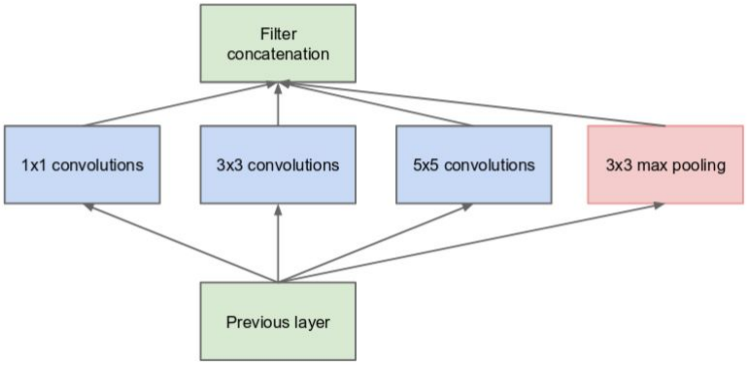
Accuracy achieved :

1. After 15 epochs : 82.5%
2. After 25 epochs : 89.4%

I tried to use data augmentation, contrary to general beliefs, it performed worser than the model without it.


Conv2D (1024) (3,3)
MaxPooling (2,2)
Conv2D (512) (3,3)
Conv2D (512) (3,3)
MaxPooling (2,2)
Conv2D (256) (3,3)
MaxPooling (2,2)
Conv2D (256) (3,3)
Conv2D (256) (3,3)
MaxPooling (2,2)
Conv2D (1024) (3,3)
Conv2D (128) (3,3)
MaxPooling (2,2)
Flatten
Dense (1024)
Dense (512)
Dense (128)
Softmax (10)

Inception Net Architecture



(a) Inception module, naïve version

In this architecture, I defined a function for inception. What it did was, it simultaneously convolved input layer with filters of sizes (1) 1*1, (2) 3*3 and (3) 5*5. Conventionally as in the research paper, we are supposed to add a max-pooling layer as well, but, I didn't add one. The convolved outputs were then concateneted and sent to the next layer as input. I added '3' such inception blocks all with number of filters as 128. After 3 inception blocks, I added 3 dense layers (after flattening) with 1024, 512, 128 neurons respectively.

OBSERVATIONS

The model didn't perform as per expectations. After 25 epochs, it had an accuracy of ~80%. But after introducing data augmentation, I could achieve an accuracy of 90.6% on the test set after '"40"' epochs (on kaggle's GPU). Due to bad performance in limited time constraint, I had to drop the idea of using this architecture.