23rd December, 2017

Artificial Intelligence

Hackathon Report

Nishant Sondhi	01FB15ECS298
Shahid Ikram	01FB15ECS274
Sanketh Rangreji	01FB15ECS267
Sumanth S Rao	01FB15ECS314

PROBLEM STATEMENT

A Computer Vision tasks using PASCAL VOC 2010 dataset.

Our goal is to do the following:

1) Object Classification
2) Object Localisation
3) Transfer Learning

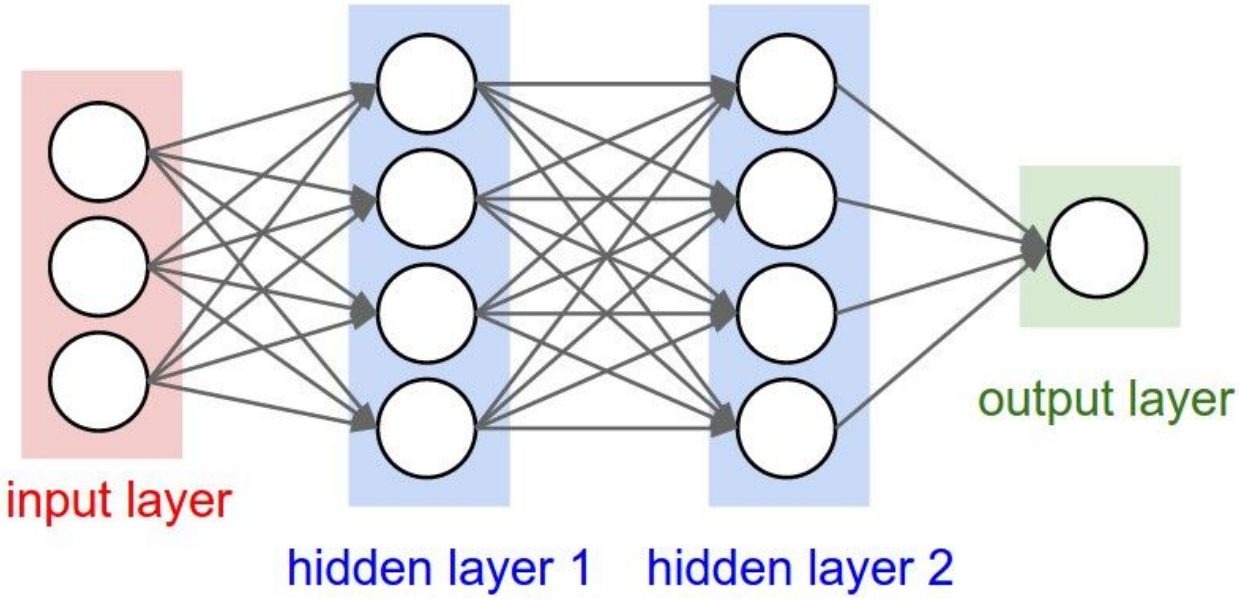
Introduction

Current approaches to object recognition make essential use of machine learning methods. To improve their performance, we can collect larger datasets, learn more powerful models, and use better techniques for preventing overfitting.

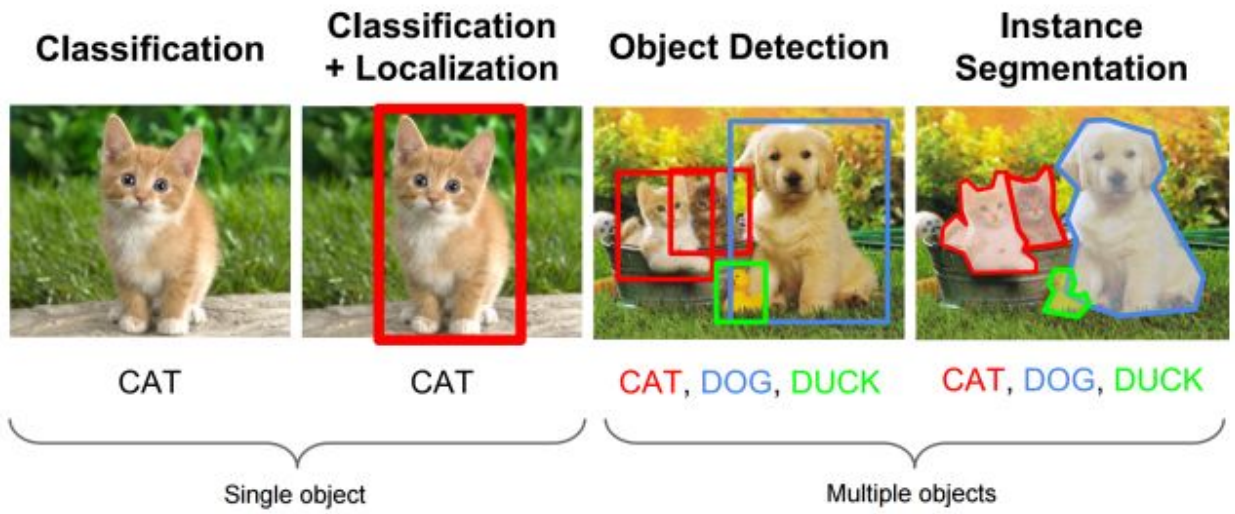
To learn about thousands of objects from millions of images, we need a model with a large learning capacity. Convolutional neural networks (CNNs) constitute one such class of models. Their capacity can be controlled by varying their depth and breadth, and they also make strong and mostly correct assumptions about the nature of images (namely, stationarity of statistics and locality of pixel dependencies). Thus, compared to standard feedforward neural networks with similarly-sized layers, CNNs have much fewer connections and parameters and so they are easier to train, while their theoretically-best performance is likely to be only slightly worse.

Convolution Neural Networks are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function

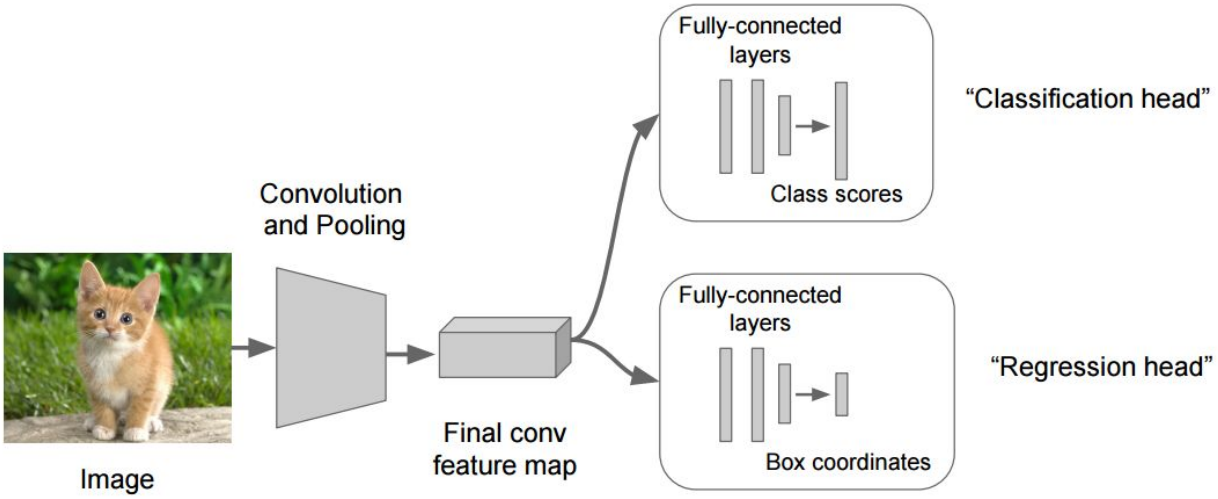
(e.g. SVM/Softmax) on the last (fully-connected) layer and all the tips/tricks we developed for learning regular Neural Networks still apply



Regression is about returning a number instead of a class, in our case we're going to return 4 numbers (x0,y0,width,height) that are related to a bounding box. You train this system with an image an a ground truth bounding box, and use L2 distance to calculate the loss between the predicted bounding box and the ground truth.



Normally what you do is attach another fully connected layer on the last convolution layer



Why do we pick Keras?

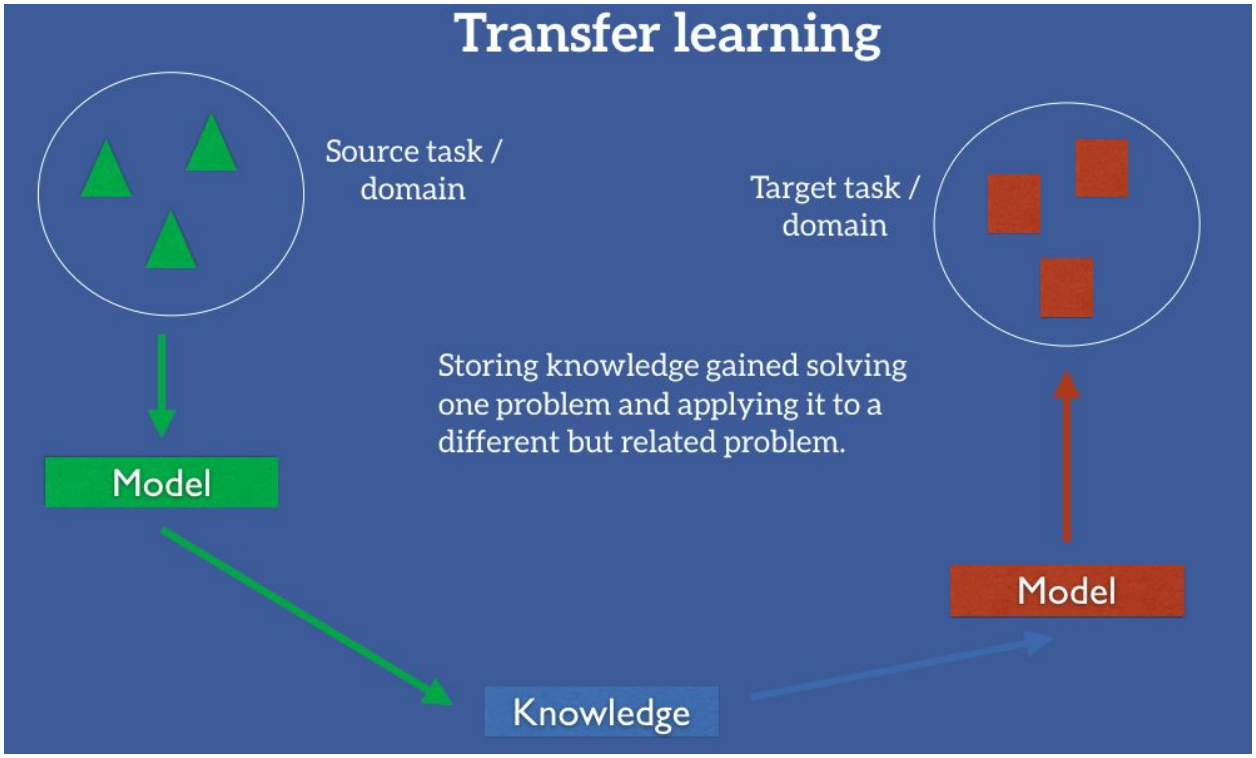
Keras is a simple to use neural network library built on top of Theano or TensorFlow that allows developers to prototype ideas very quickly. Unless there is some cutting-edge research that involves customizing a completely novel neural architecture with different activation mechanism, Keras provides all the building blocks that we need to build reasonably sophisticated neural networks.

Note on Hardware

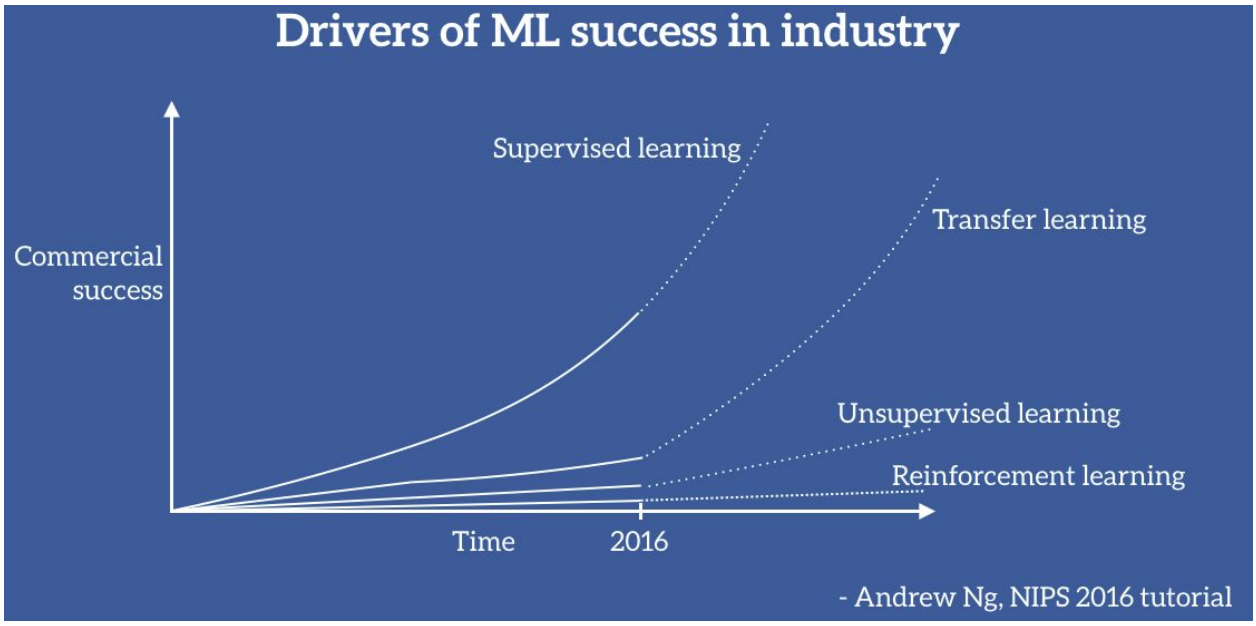
The Hardware configuration greatly affects the feasability of the models that we build.Tensorflow running on the CPU is paralyzed with the Resource Availability.Hence using Tensorflow with GPU is preferred to do the heavy computation involved in training for this project. The speed difference is very substantial. We are talking about a matter of hours with a GPU versus a matter of days with a CPU. We did face some difficulties running on the available hardware and hence had to reduce image dimensions and use grey scale.

Keras Applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning.

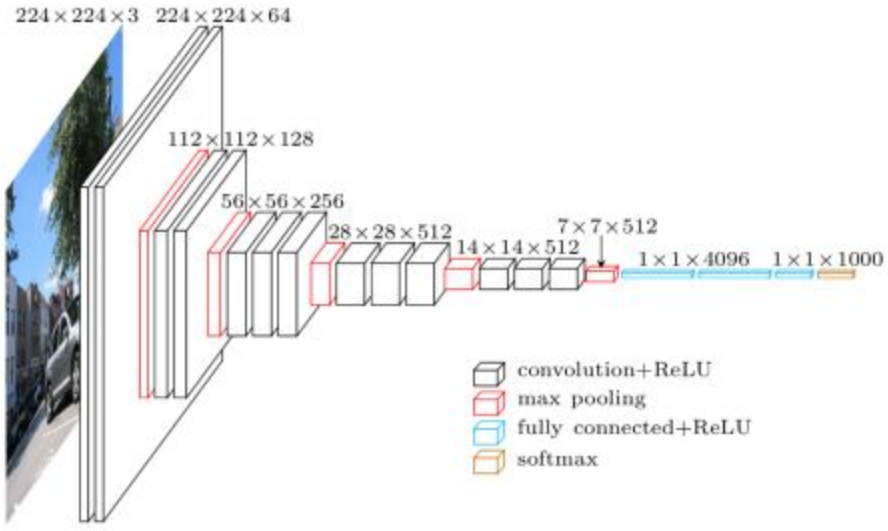
Transfer Learning allows us to deal with the scenarios by leveraging the already existing labeled data of some related task or domain.We store the source task in the source domain and apply it to our problem.The initial layers of the Pretrained Model remains the same and we remove the ending dense layers and replace them with ours to fit the output labels.



In practice, we seek to transfer as much knowledge as we can from the source setting to our target task or domain. This knowledge can take on various forms depending on the data: it can pertain to how objects are composed to allow us to more easily identify novel objects; it can be with regard to the general words people use to express their opinions, etc.



Fine-tune VGG16. VGG16 is a 16-layer Covnet used by the [Visual Geometry Group] (VGG) at Oxford University in the 2014 ILSVRC (ImageNet) competition. The model achieves a 7.5% top 5 error rate on the validation set



Weights are downloaded automatically when instantiating a model. They are stored at ~/.keras/models/.

The Architecture

Input dimensions - 100 * 100

Loss function - binary cross entropy

Output layer Activation - sigmoid

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_1 (Conv2D)              (None, 126, 126, 32)      320
_________________________________________________________________
pool_1 (MaxPooling2D)        (None, 63, 63, 32)        0
_________________________________________________________________
conv_2 (Conv2D)              (None, 60, 60, 64)        32832
_________________________________________________________________
pool_2 (MaxPooling2D)        (None, 15, 15, 64)        0
_________________________________________________________________
dropout_3 (Dropout)          (None, 15, 15, 64)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 14400)             0
_________________________________________________________________
dense_1 (Dense)              (None, 200)               2880200
_________________________________________________________________
dense_2 (Dense)              (None, 100)               20100
_________________________________________________________________
modeloutput (Dense)          (None, 15)                1515
=================================================================
Total params: 2,934,967
Trainable params: 2,934,967
Non-trainable params: 0
_________________________________________________________________
```

Accuracy: 93.51%
Precision: 40.32%
Recall: 33.05%
F-score: 26.78%

Localisation

Input dimensions - 128 *128

Loss function - Mean squared error

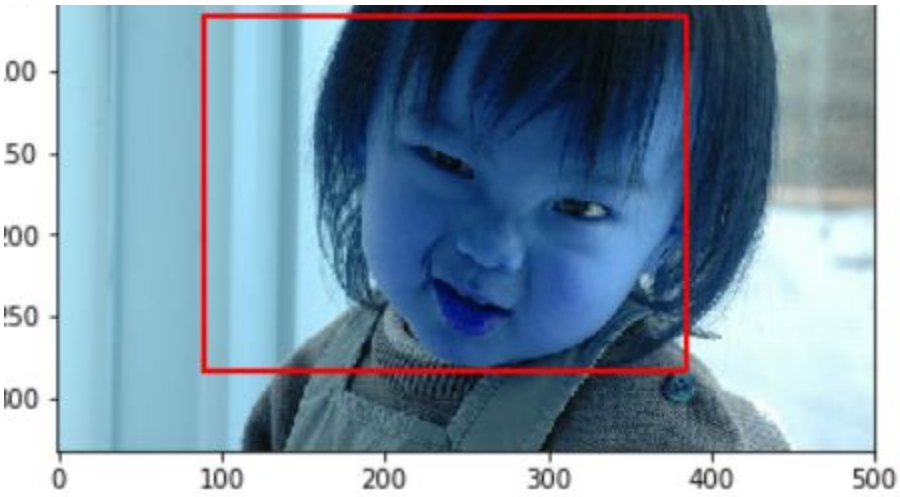
No activation in Output layer(just 4 neurons)

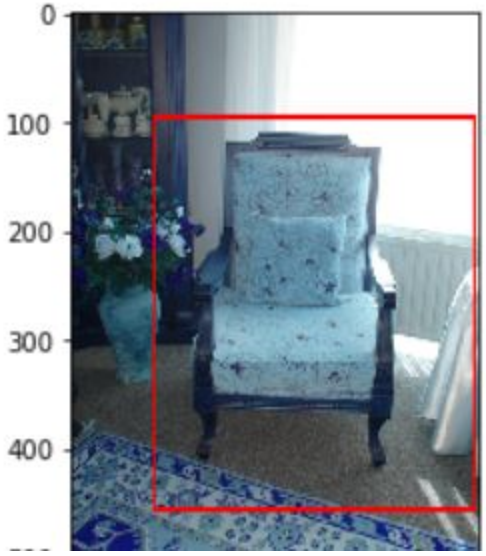
```
In [46]: model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_1 (Conv2D)              (None, 126, 126, 32)      320
_________________________________________________________________
pool_1 (MaxPooling2D)        (None, 63, 63, 32)        0
_________________________________________________________________
conv_2 (Conv2D)              (None, 60, 60, 64)        32832
_________________________________________________________________
pool_2 (MaxPooling2D)        (None, 15, 15, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 15, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 14400)             0
_________________________________________________________________
dense_1 (Dense)              (None, 150)               2160150
_________________________________________________________________
dense_2 (Dense)              (None, 80)                12080
_________________________________________________________________
modeloutput (Dense)          (None, 4)                 324
=================================================================
Total params: 2,205,706
Trainable params: 2,205,706
Non-trainable params: 0
_________________________________________________________________
```

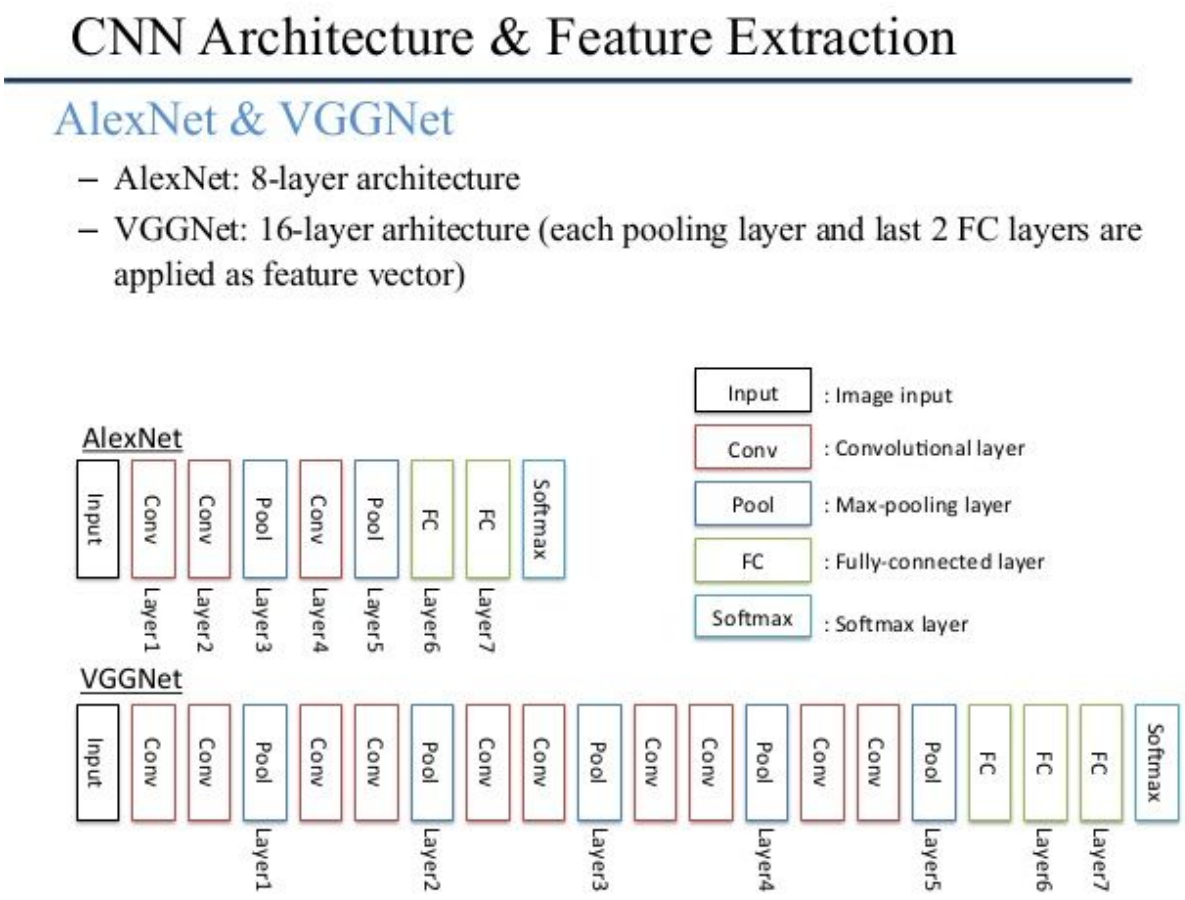
Accuracy : 88%

SAMPLE BOUNDING BOX PREDICTIONS





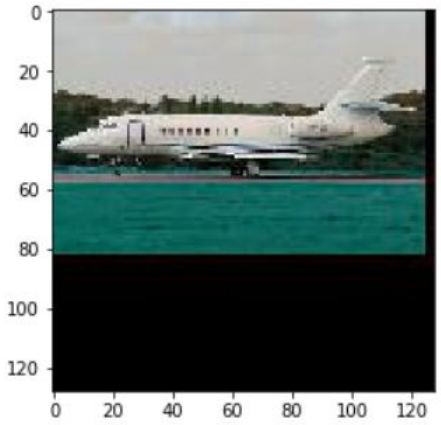
Transfer Learning



We used the precompiled VGG19 model as a part of of the Keras package, we used pretrained weights of ImageNet(as it dealt with similar classes).

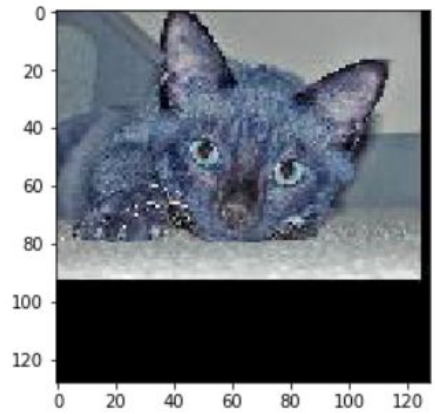
We then froze the convolution layers(we used only that part of the precompiled model) and added two Fully Connected layers of 150 and 80 neurons, followed by a Softmax layer for 15 classes.

Accuracy : 60%



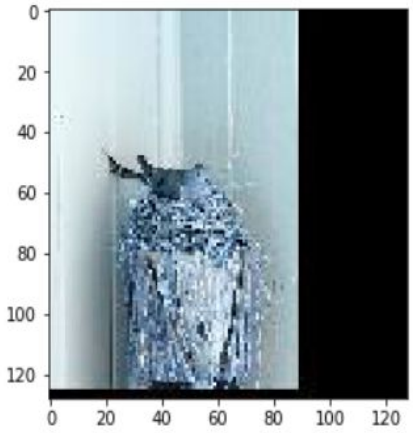

```
ıapping[np.argmax(final_model.predict(np.array(X_test.iloc[543]).reshape((1, 128, 128, 3))))]
```

```
aeroplane'
```




```
mapping[np.argmax(final_model.predict(np.array(X_test.loc[537]).reshape((1, 128, 128, 3))))]
```

```
'cat'
```




```
mapping[np.argmax(final_model.predict(np.array(X_test.loc[1581]).reshape((1, 128, 128, 3))))]
```

```
'bird'
```