University of Burgundy

MSCV

Report

Classification and Object Detection

Majeed Hussain
AkshayKumar Dudhagara

30.05.2019

Supervised by
Desire Sidibe

Contents

1 Introduction

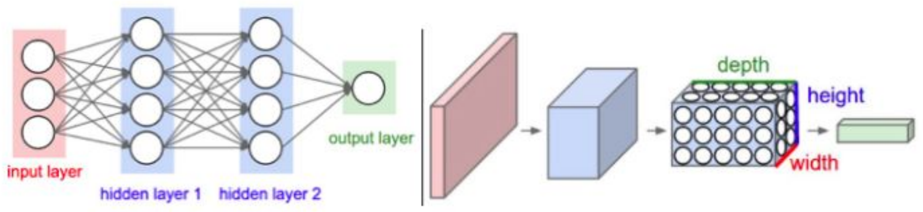
The Objective of this Project is to Classify and Detect different types of Objects using Neural Networks and deploy using Flask web based service. There are different types of Neural networks depending upon the problem. Here we use Convolutional Neural network which are widely popular for Image Classification and Recognition.

Convolutional Neural network are inspired by brain and the research on this topic was started in 1950's on mammals and continued. Later in 1998 a breakthrough for CNN was happened when researcher's namely Bengio, Le Cun, Bottou and Haffner released a paper on CNN's called **"LeNet-5"** it was able to classify the handwritten numbers.

1.0.1 Architecture

Convolutional Neural Networks have a different architecture than regular Neural Networks. Regular Neural Networks transform an input by putting it through a series of hidden layers. Every layer is made up of a set of neurons, where each layer is fully connected to all neurons in the layer before. Finally, there is a last fully-connected layer the output layer that represent the predictions.

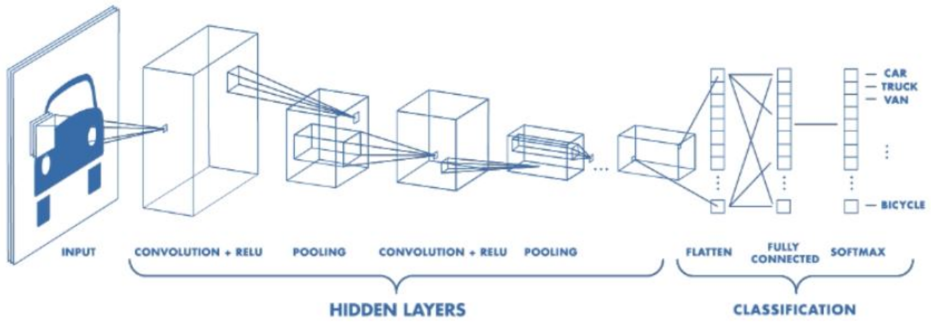
Convolutional Neural Networks are a bit different. First of all, the layers are organised in 3 dimensions: width, height and depth. Further, the neurons in one layer do not connect to all the neurons in the next layer but only to a small region of it. Lastly, the final output will be reduced to a single vector of probability scores, organized along the depth dimension.



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Normal NN vs CNN.—Source: http://cs231n.github.io/convolutional-networks/

This CNN's will perform series of convolutions and pooling operations during which features are extracted and the last fully connected layers will serve as classifiers, it gives the predictions of the class the object belongs to in terms of probabilities.



Architecture of a CNN.—Source: https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html

1.0.2 Tasks

This project is divided into two tasks

1. **Deploying Classification API**
2. **Deploying YOLO Object Recognition API**

For the first Classification task we started from basic simple network which gave low accuracy's to bigger networks which improved accuracy by adding more layers.This gave an insight how CNN's perform on different architectures. All the procedure we tried are explained with results in next session.

1.0.3 HOW TO RUN THE CODE

Follow the below instructions to run the code

Install Dependencies

To install dependencies go to working directory through terminal there you will find a **requirements.txt** file, using this file we install all the dependencies by following command.

```
pip install -r requirements.txt
```

Download VGG-16 model

I have uploaded the Fine-tuned VGG-16 model in Google drive, we need to download this and place inside **Classification-API folder.**

Follow the link below and download the Fine-tuned VGG-16 model
`https://drive.google.com/open?id=14sA6iycgN1qZgB-lqe8Z2z0zY39QW6vl`

Download YOLOv2 weights

In order to run YOLO-Object-Recognition-API we need weights download it and place in **bin** folder i.e, here **YOLO-Object-Recognition-API/darkflow/bin/**, click the below link it will download the weights.

`https://pjreddie.com/media/files/yolov2-voc.weights`

Notebooks

I have included the notebooks where we started from training small network followed by Data augmentation followed by fine tuning the VGG-16 and saving that model.

Run Classfication-API

In order to run the Classification API go to **Classification-API** folder through terminal and run the following commands.

```
export FLASK_APP=predict_app.py
flask run --host=0.0.0.0
```

Once the above commands are executed head towards the following link to see the Classification API

`http://0.0.0.0:5000/static/predict.html`

Run YOLO-Object-Recognition-API

In order to run the YOLO-Object-Recognition-API go to **YOLO-Object-Recognition-API/darkflow/** folder through terminal and run the following commands.

```
export FLASK_APP=yolo.py
flask run --host=0.0.0.0
```

Once the above commands are executed head towards the following link to see the YOLO Object Detection API

`http://0.0.0.0:5000/`

Youtube link Demonstration

Following is the Youtube link for demonstration of how to run the code and results.

`https://youtu.be/ztFiGbJPJwM`

2 Procedure

2.1 Classification

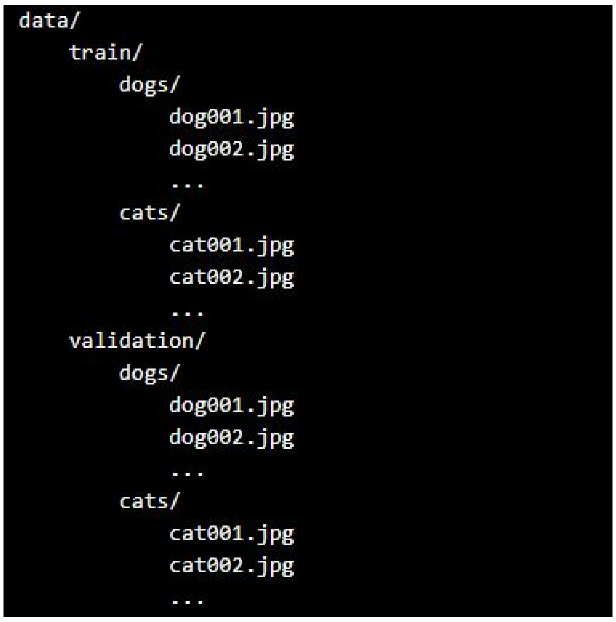
The first part of the Project is to build a model which does classification task and later on save that model and use in deploying stage. So we started with Simple Convolutional network which gave around just 50 percent accuracy later we applied Data augmentation techniques which increased the accuracy to 75 percent later on we used pretrained network namely VGG-16 an award winning network in Imagenet challenge.

All the above mentioned methods and there respective outputs are stated in following section.

2.1.1 Dataset

For this task we use Dogs and Cats Dataset from Kaggle. The Dataset contains 25,000 images. We used only 4000 images each class of 2000 images to train our network and 2000 images for validation.

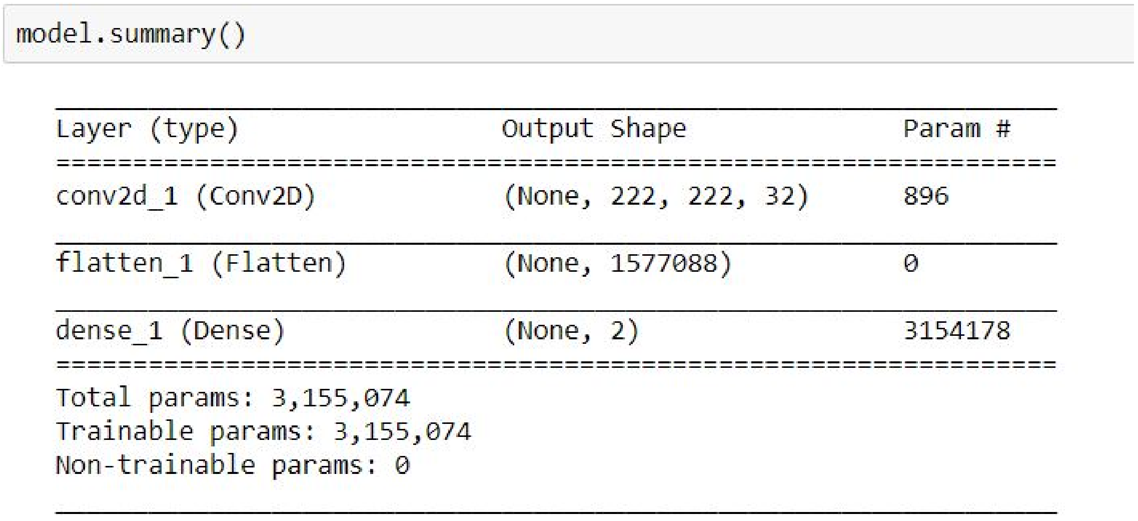
The dataset directory tree looks as below:



```
data/
    train/
        dogs/
            dog001.jpg
            dog002.jpg
            ...
        cats/
            cat001.jpg
            cat002.jpg
            ...
    validation/
        dogs/
            dog001.jpg
            dog002.jpg
            ...
        cats/
            cat001.jpg
            cat002.jpg
            ...
```

2.1.2 Build Simple Neural Network

We start with simple neural network with just one convolutional layer followed by flattening the layer followed by final classification layer of two classes.The architecture of the network as follows:



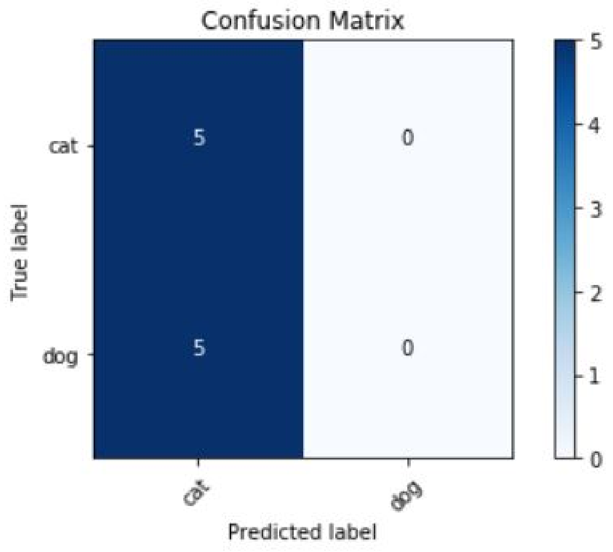
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 222, 222, 32)      896
_________________________________________________________________
flatten_1 (Flatten)          (None, 1577088)           0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 3154178
=================================================================
Total params: 3,155,074
Trainable params: 3,155,074
Non-trainable params: 0
_________________________________________________________________
```

Using 4000 samples of training data and 2000 samples of validation data we acquired validation accuracy of **50 percent.**

Using 10 test samples we plot the confusion matrix as follows:

```
Confusion matrix, without normalization
[[5 0]
 [5 0]]
```



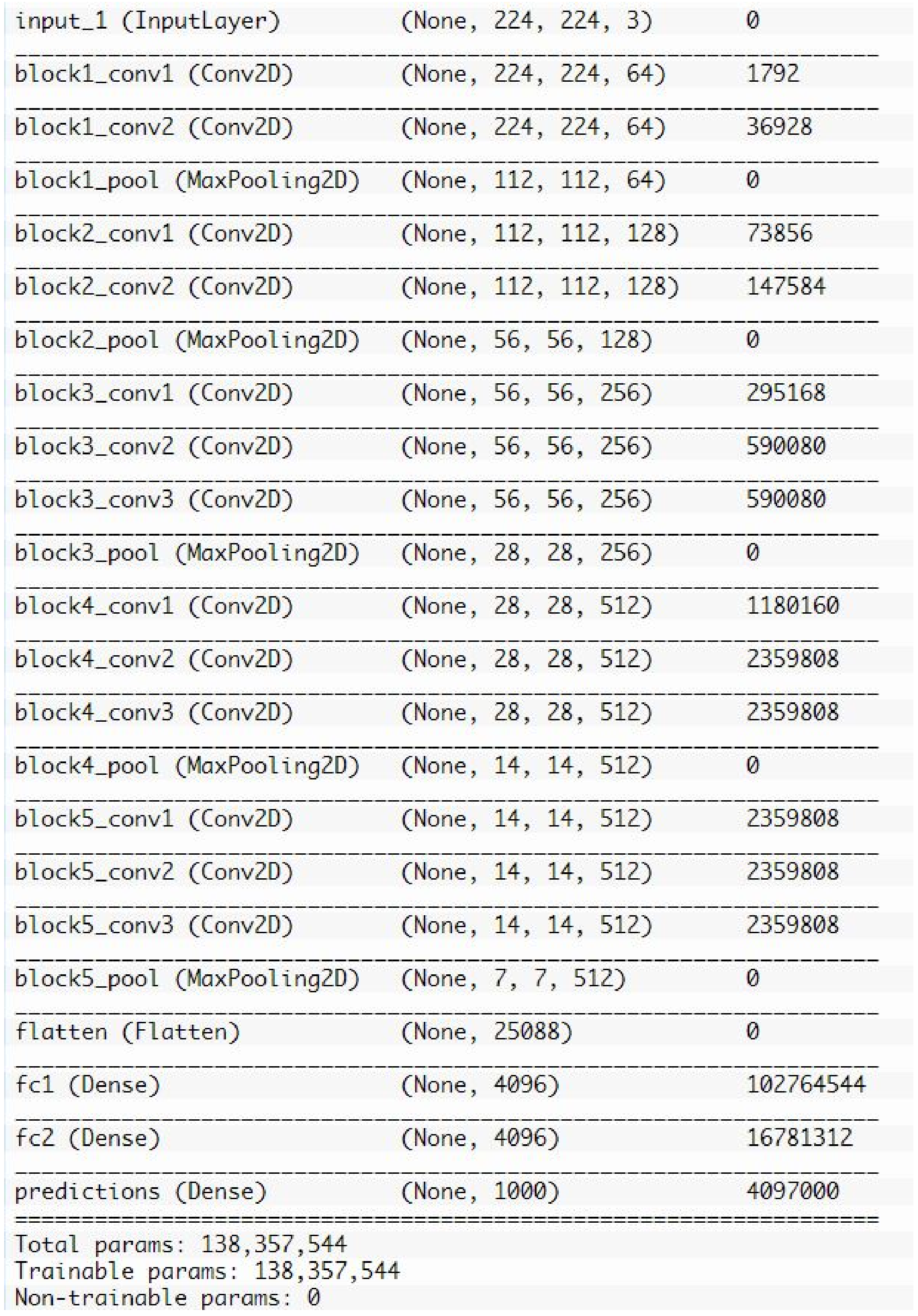
2.1.3 Using Data Augmentation

Further to increase the task we performed Data Augmentation technique.It is done when we have less samples for training.In order to make the most of our few training samples, we will augment them by number of random transformations, so that our model would never see twice the exact same picture. This helps prevent overfitting and helps the model generalize better.The random transformations are random rotations of image , width and height shift ,re-scaling, shear, zoom,flip etc. After generating some samples save them to a temporary directory, we use it for training the network. The accuracy after data augmentation increased by 25 percent i.e, validation accuracy obtained was **75 percent.**

2.1.4 Using Transfer learning

Another way to increase accuracy is by using pre-trained network. Here we used VGG-16 challenge winning network of Imagenet competition.This pre-trained network is already trained on a large dataset. The ImageNet dataset contains several *cat* classes and many *dog* classes among its total of 1000 classes, this model will already have learned features that are relevant to our classification problem. Such a network would have already learned features that are useful for most computer vision problems, and leveraging such features would allow us to reach a better accuracy than any method that would only rely on the available data.

Architecture of VGG-16 looks as below:

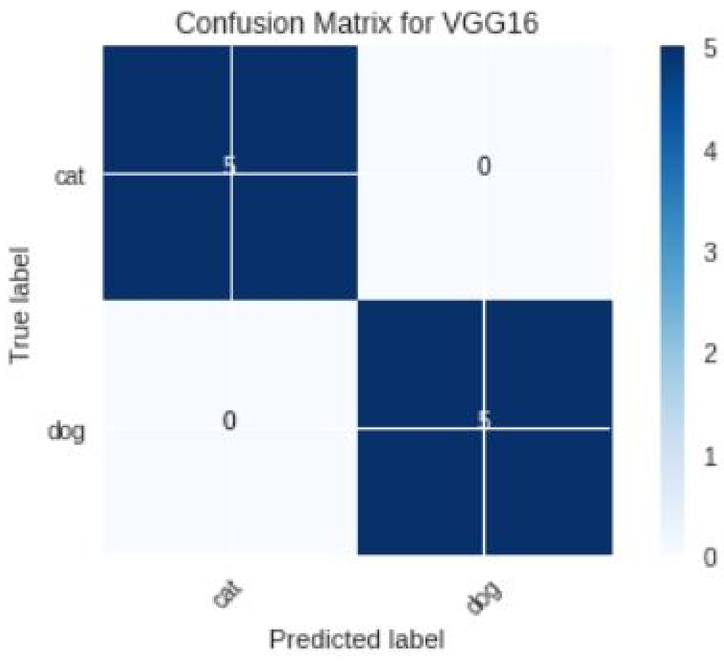


```
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
```

Fine Tuning VGG-16

As we can see the network has a final dense layer of 1000 classes since it has been trained on Imagenet dataset which contains 1000 classes. But for our problem we need final dense to be 2 since we are doing classification on two classes i.e, cats and dogs.So we remove last layer of 1000 neurons and replace with 2 neurons. Once our model is ready we freeze the intermediate layers since we don't want our intermediate layer weights to be changed and we now train the model on last layer and feed our test data to this network. Since it has already been trained with different objects in which cats and dogs are present the accuracy will be more.
The Validation accuracy obtained for the VGG-16 is **95 percent**.The confusion matrix for the test set is as follows:

```
Confusion matrix, without normalization
[[5 0]
 [0 5]]
```



Comparison table for all the different architecture networks is as follows:

	Training Accuracy	Validation Accuracy
Simple NN	49 %	50 %
Simple NN using Data Augmentation	85 %	75 %
Pre-trained Neural Network	98 %	95 %

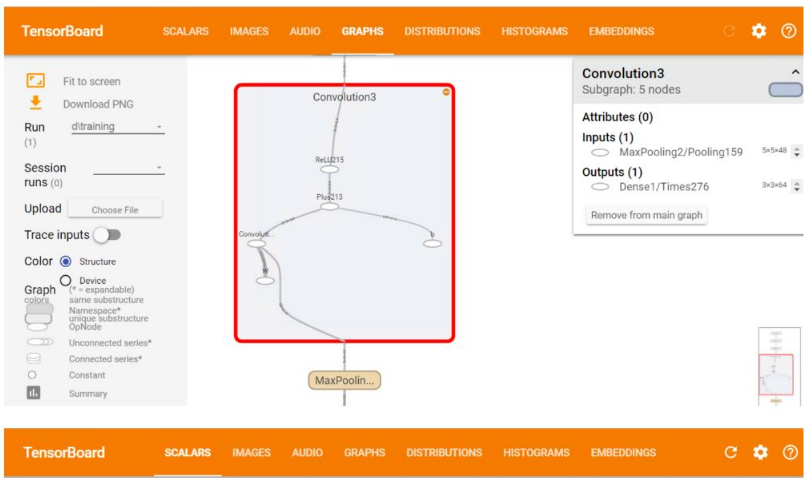
As now we have got Satisfactory accuracy we will now **save** this model in **h5** format and use this trained model in deploying using Flask web service.

2.1.5 Visualization

Visualization is key task to understand what does the CNN layers are looking for in each layer,for this visualization we used Tensorboard a visualization toolkit,TensorBoard is a visualization software that comes with any standard TensorFlow installation. Since we are using keras here we can directly use tensoboard there we used **keras callbacks** to record our graphs and then we use those graphs to visualize our accuracy plots and also the Intermediate layers. To see the graphs in tensorboard go to the **notebooks directory** through terminal and then run the following command;

```
cd Classification_and_Object-detection/notebooks
tensorboard --logdir path/to/Graph
```

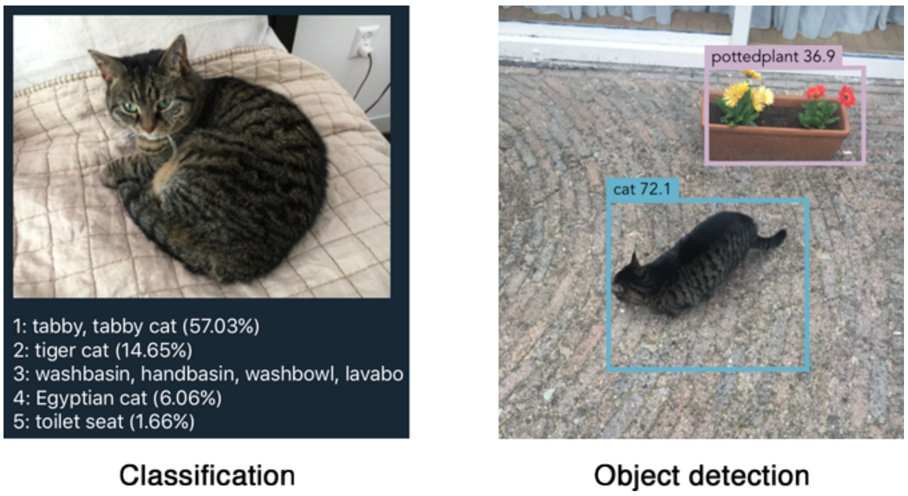
The output will looks as follows:



2.2 YOLO Object detection

In computer vision, Object detection is one of the classical problems: Recognize what the objects are inside a given image and also where they are in the image.

Detection is a more complex problem than classification, which can also recognize objects but doesn't tell exactly where the object is located in the image â and it wonât work for images that contain more than one object.



YOLO is a clever neural network for doing object detection in real-time.

2.2.1 How YOLO Works

YOLO actually looks at the image just once but in a clever way hence it s called You Only Look Once. According to the paper of **YOLOv2**, it became more accurate and faster than the previous version (YOLO). This is because YOLOv2 uses some techniques that YOLO didn'tât use, such as**Batch-Normalization** and **Anchor-Boxes**.

Batch-Normalization or BN is used to normalize the outputs of hidden layers. This makes learning much faster. Anchor-Boxes is assumption on the shapes of the bounding boxes. Since the shapes of objects weâre trying to detect do not vary so much, we donât have to find boxes that do not look like any of objects we want to detect. Letâs say we want to detect humans, then the shapes of anchor boxes are usually vertical rectangles and itâs less likely that they are squares or horizontal rectangle. So we donât have to search such boxes. This makes prediction much faster.

In our case of YOLOv2 (YOLO), the output is a 3-dimensional array (or Tensor in TensorFlow). Particularly in YOLOv2, the shape of output is 13x13xD, where D varies depending on how many classes of object we want to detect (D=5 for single class). The first 2 dimensional array (13x13) is calledÂ grid cells. So there are 169 grid cells in total.

One grid cell is ârresponsibleâ for detecting 5 bounding boxes, that is we can detect up to 5 boxes on a grid cell. A bounding box describes the rectangle that encloses an object. This means that the network can detect up to 169 x 5 = 845 boxes at once. This number of bounding boxes a grid cell can detect is actually the number of Anchor-Boxes we prepare, and we can change this number to whatever we want.

YOLO also outputs a confidence score that tells us how certain it is that the predicted bounding box actually encloses some object. This score doesnât say anything about what kind of object is in the box, just if the shape of the box is any good.

There are 13x13 = 169 grid cells in total, and each grid cell can detect up to B bounding boxes. One bounding box has 5 + C properties, therefore a grid cell has D = Bx(5+C) values (this is depth).

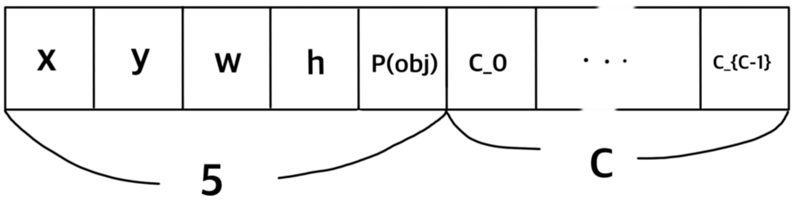
Each grid cell has depth of D. The value of D depends on the number of classes we want to detect. When we have C classes of object, D is

$D = B(5 + C)$

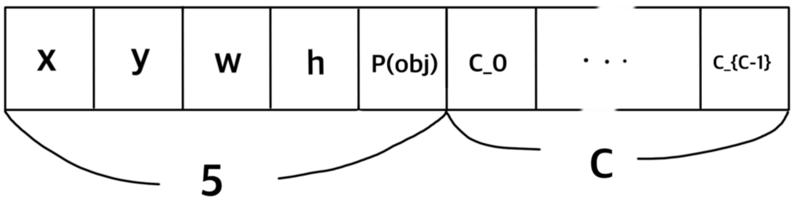
B is the number of bounding boxes a single grid cell is responsible for detecting, and 5 represents the prediction of the bounding box position and the probability that an object exists in this box:

1. x: x coordinate of the box center
2. y: y coordinate of the box center

3. w: box width
4. h: box height
5. P(obj): probability that an object exists in this box



Each grid cell is able to predict B bounding boxes. Since each bounding box prediction is composed of 5 + C values, the total length of predicted values on one grid cell is B*(5+C). I will consider the case when B = 5 and C = 1, so one grid cell has length D =30.Note that x, y, w and h are not in âpixelsâ since images on which we apply object detection do not have the same size. For example, one image may have size 1080x1920x3, while another may have 2160x4096x3 (where 3 is for RGB). Therefore, before we feed images to the network, we reshape them into 416x416x3 images such that they have the same size.



This image represents one of the B bounding boxes a grid cell predicts. The first 5 values are fixed while C varies depending on the number of classes. Pobj x $C_i$becomes the probability that an object of the i-th class exists in this bounding box. C represents conditional probabilities that, given an object exits in the box, the object belongs to a specific class:

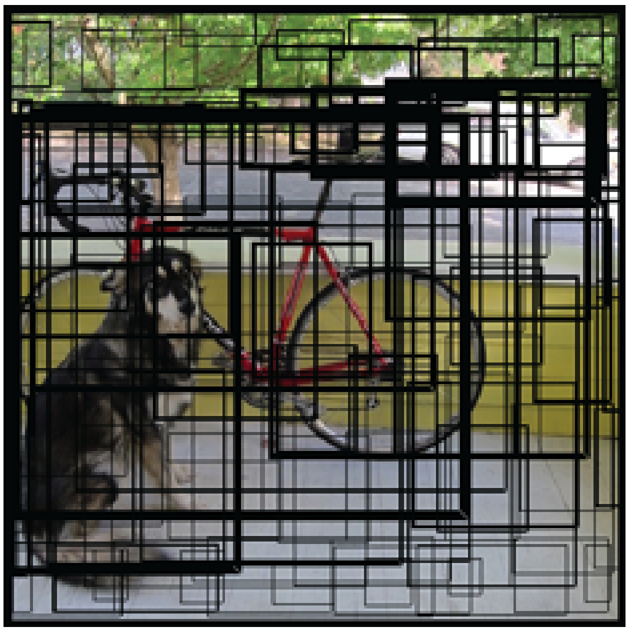
1. C_i = P(the obj belongs i-th class — an obj exists in this box)

where 0 â i â C-1. So the probability that an object of the i-th class is given by:

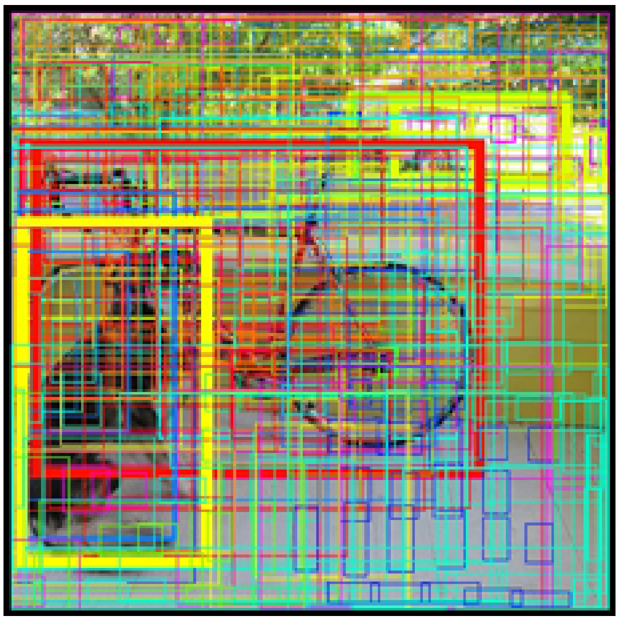
$$C_i P(obj) \leq 1.0$$

If this value is greater than a threshold, we think that the network predicted that an object of the i-th class exists in this bounding box.

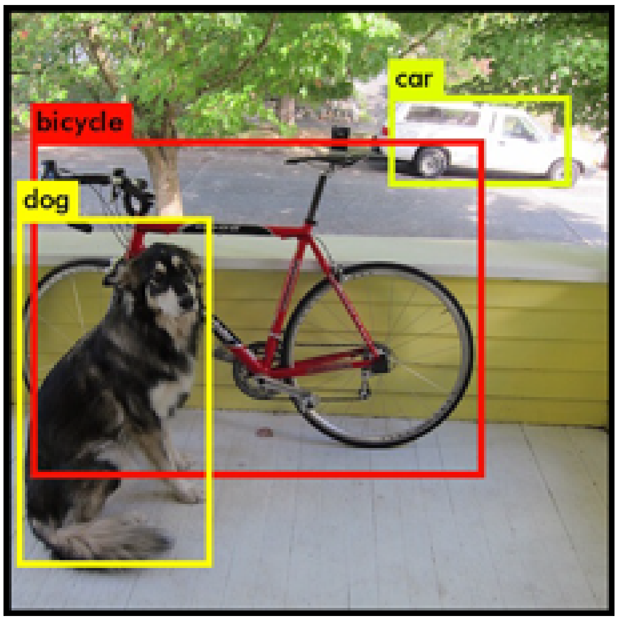
The predicted bounding boxes may look something like the following (the higher the confidence score, the fatter the box is drawn):



The confidence score for the bounding box and the class prediction are combined into one final score that tells us the probability that this bounding box contains a specific type of object. For example, the big fat yellow box on the left is 85 percent sure it contains the object **dog:**



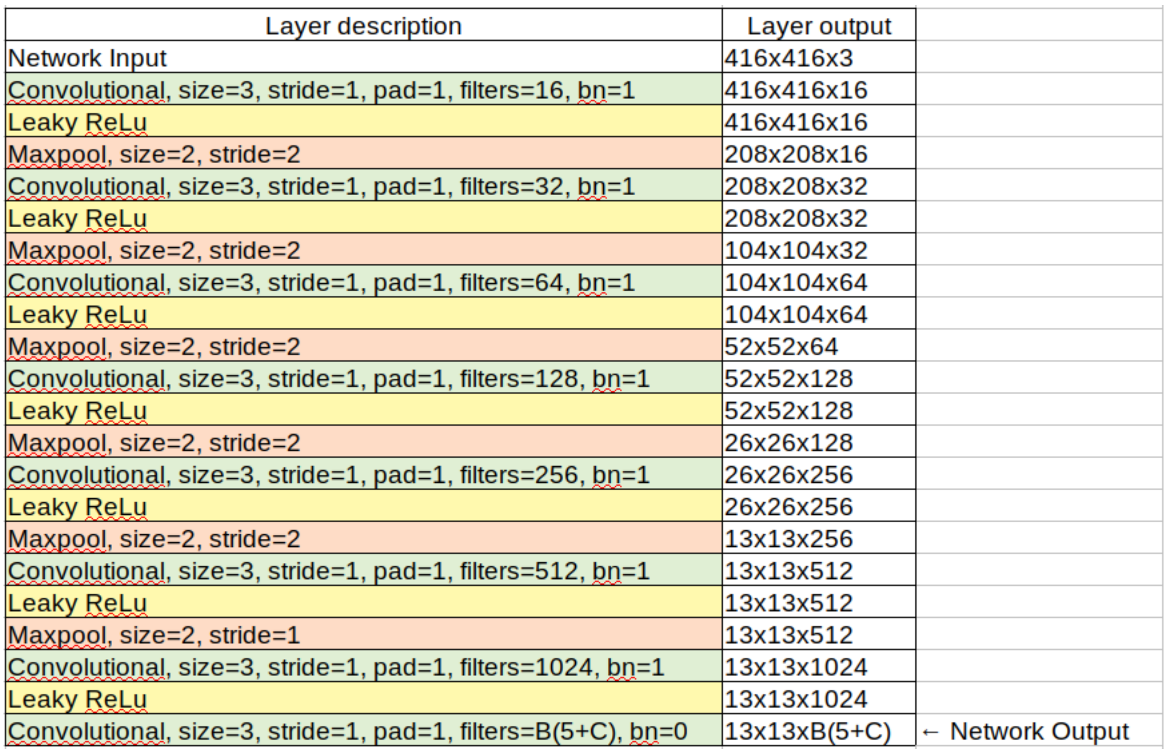
The final prediction is then:



From the 845 total bounding boxes we only kept these three because they gave the best results. But note that even though there were 845 separate predictions, they were all made at the same time â the neural network just ran once. And thatâs why YOLO is so powerful and fast.

2.2.2 Network Architecture

The input to the network is 416x416x3 image in YOLOv2. There is no fully connected layer in it.



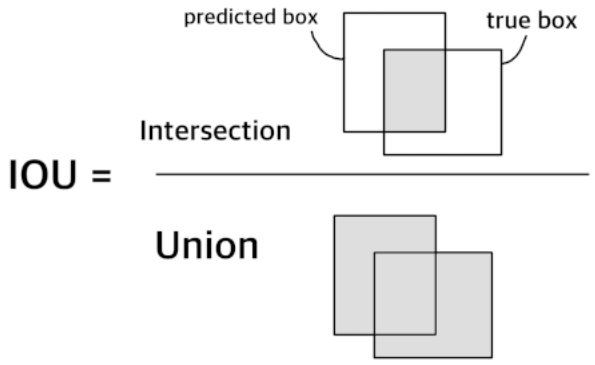
Layer description	Layer output	
Network Input	416x416x3	
Convolutional, size=3, stride=1, pad=1, filters=16, bn=1	416x416x16	
Leaky ReLu	416x416x16	
Maxpool, size=2, stride=2	208x208x16	
Convolutional, size=3, stride=1, pad=1, filters=32, bn=1	208x208x32	
Leaky ReLu	208x208x32	
Maxpool, size=2, stride=2	104x104x32	
Convolutional, size=3, stride=1, pad=1, filters=64, bn=1	104x104x64	
Leaky ReLu	104x104x64	
Maxpool, size=2, stride=2	52x52x64	
Convolutional, size=3, stride=1, pad=1, filters=128, bn=1	52x52x128	
Leaky ReLu	52x52x128	
Maxpool, size=2, stride=2	26x26x128	
Convolutional, size=3, stride=1, pad=1, filters=256, bn=1	26x26x256	
Leaky ReLu	26x26x256	
Maxpool, size=2, stride=2	13x13x256	
Convolutional, size=3, stride=1, pad=1, filters=512, bn=1	13x13x512	
Leaky ReLu	13x13x512	
Maxpool, size=2, stride=1	13x13x512	
Convolutional, size=3, stride=1, pad=1, filters=1024, bn=1	13x13x1024	
Leaky ReLu	13x13x1024	
Convolutional, size=3, stride=1, pad=1, filters=B(5+C), bn=0	13x13xB(5+C)	← Network Output

The structure of YOLOv2 used in darkflow. If bn=1, the convolutional layer uses Batch-Normalization, bn=0 otherwise. The shape of the final convolutional layer varies depending on B and C. This means that we cannot load any pretrained weights on this layer, while other layers can use pretrained weights. Also Note only the final layer does not use BN since itâs not necessary.

2.2.3 Anchor-Boxes for more accuracy

In YOLO, which is the previous version of YOLOv2, the prediction of the box shape was done randomly. That is, the network did not know what shapes of bounding box are most likely to detect an object of a certain class. For example, when the network tries to detect humans, it searches humans with square bounding box and vertical rectangle equally likely. However, in the real world, humans usually fit more in vertical rectangles than square boxes. So the network should search humans with vertical rectangle. This is the motivation of using Anchor-Boxes.

2.2.4 What is IOU?



This is short for Intersection Over Union. This is calculated by dividing the overlapped area of a predicted box and the truth box by the whole area made by the two boxes.
This value is used as a measure of how good the prediction is.

2.2.5 Loss Function composed of 3 losses

:

The prediction of YOLOv2 is composed of three parts:

1. the four coordinates for x, y, w and h

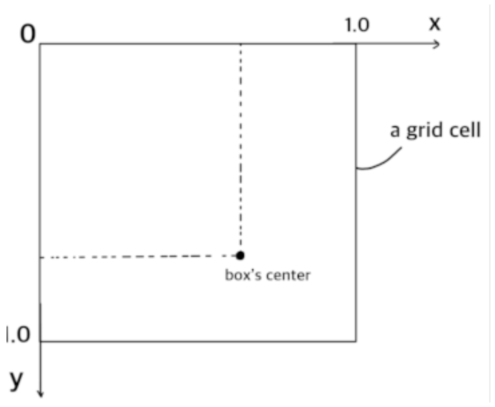
2. the probability P(obj) that an object exists in a bounding box

3. the conditional probability Cá$\mu = P(theobjbelongsi{-}thclass|anobjexistsinthisbox)$

therefore the loss is calculated for each prediction and then combined.

2.2.6 Loss for x, y, w and h

The coordinates of x and y lie between 0 and 1. This is because they are given by applying sigmoid on the x and y part of the output from network (at least in darkflow). This prediction is done relative to the grid cell. For example, if both x and y are 0.5, then this meas that the boxâs center falls on the center of the grid cell. The reason why the center coordinates are predicted this way is just we donât need to know the absolute coordinates if we know this grid cell position.



This image depicts one grid cell. The prediction of the center coordinates is done relative to the grid cell size. If the boxâs center lies at the bottom right of the cell, then both x and y are 1.0.

Then the loss is defined as a sum of squares of error:

$$\sum_{i=0}^{13^2-1} \sum_{j=0}^{B-1} \mathbf{1}_{i,j}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$

The sum is over all grid cells and all bounding boxes. $1_{i,j}^{obj}$ is 1.0 if the bounding box is âresponsibleâ for detecting this object, 0.0 otherwise. The term âresponsibleâ means that this bounding box has the highest IOU value among the B bounding boxes, and there is actually an object on this grid cell. The sum is done only on the bounding boxes that are responsible for detecting objects. This is because, for bounding boxes that do not actually contain any objects, we donât know the true center coordinates $\hat{x}_i$ and $\hat{y}_i$. This is also the case for loss of w and h.

The loss function for w and h looks almost the same as the one for x and h:

$$\sum_{i=0}^{13^2-1} \sum_{j=0}^{B-1} \mathbf{1}_{i,j}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

Sum-squared error often becomes large for larger bounding boxes, and becomes small for smaller boxes. To address this unfairness, we use square roots of weight and height instead of using them directory.

2.2.7 Loss for P(obj)

Each bounding box has the probability that the box contains an object. If the bounding box does not actually contain an object, the predicted probability should be decreased through training, and if the box actually contains an object, the predicted probability should be close to 1.0.

$$g_{obj} \sum_{i=0}^{13^2-1} \sum_{j=0}^{B-1} \mathbf{1}_{i,j}^{obj} (P(obj)_i - P(\hat{obj})_i)^2 + g_{noobj} \sum_{i=0}^{13^2-1} \sum_{j=0}^{B-1} (1 - \mathbf{1}_{i,j}^{obj})(P(obj)_i - P(\hat{obj})_i)^2$$

The first term is the loss for âresponsibleâ bounding boxes, and the second one is for the non-responsible ones. Note that g_{obj} should be greater than g_{noobj} according to the YOLO paper.

2.2.8 Loss for Conditional Probabilities C$_i$

We can define loss values only for responsible bounding boxes, since there are no truth labels for non-responsible bounding boxes.

$$g_{sprob} \sum_{i=0}^{13^2-1} \sum_{j=0}^{B-1} \sum_{classes} \mathbf{1}_{i,j}^{obj} (C_i - \hat{C}_i)^2$$

g_{sprob} is a constant.

2.3 Deploy using Flask

Flask is a Python web framework built with a small core and easy-to-extend philosophy.It is classified as a micro-framework because it does not require particular tools or libraries.It makes the process of designing a web application simpler.Flask supports extensions that can add application features as if they were implemented in Flask itself.

2.3.1 Tasks

Here we deployed our two models i.e,

1. Classification API
2. YOLO API

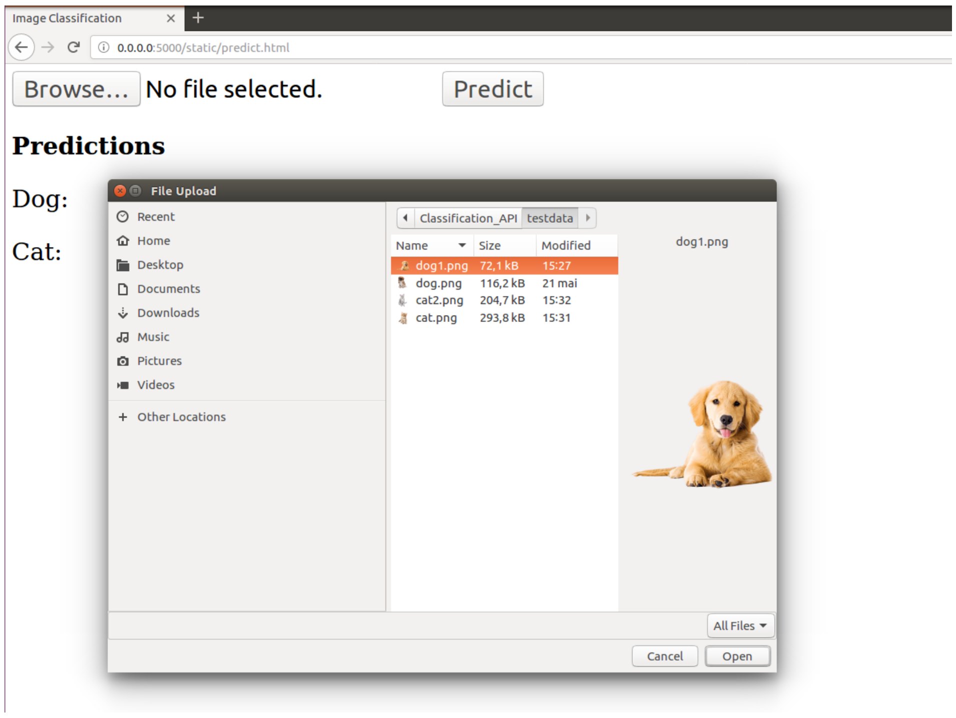
Both outputs are shown in result Section.

3 Results

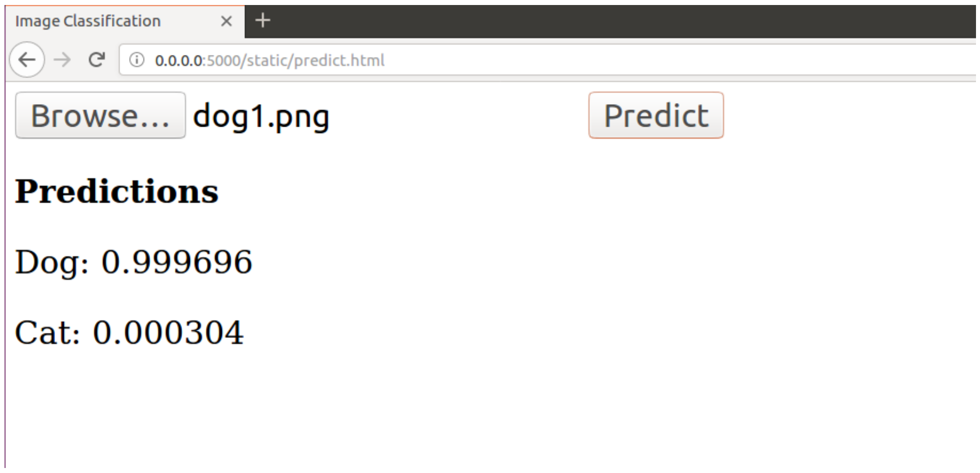
3.1 Classification API

Follow the **HOW TO RUN THE CODE** section provided in the Introduction section in the start to run the code in local machine.Following are the results of Classification API

1) First Press the **Browse** button and select the image from testdata folder and then press **Predict** button and wait for few seconds you will observe the Probability values of Classification task.



Name	Size	Modified
dog1.png	72,1 kB	15:27
dog.png	116,2 kB	21 mai
cat2.png	204,7 kB	15:32
cat.png	293,8 kB	15:31



Predictions

Dog: 0.999696

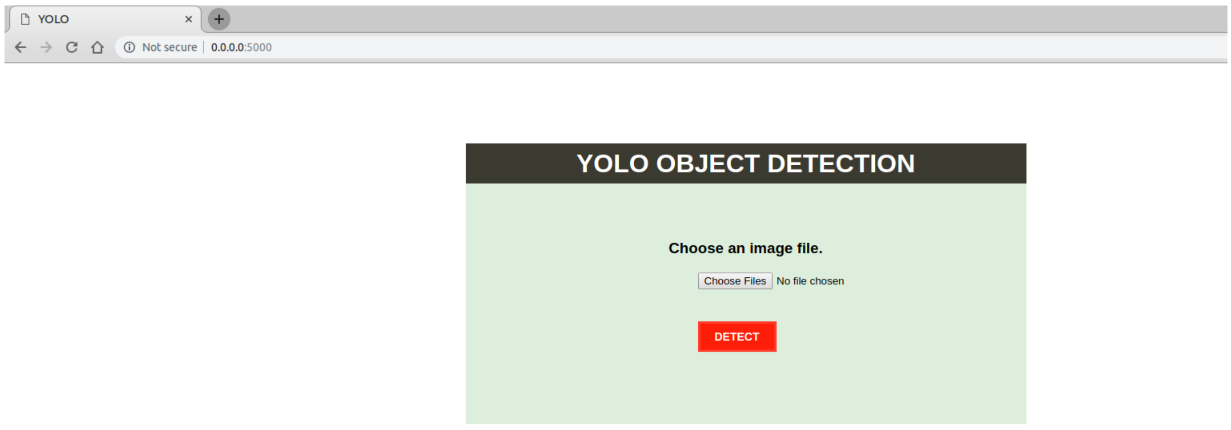
Cat: 0.000304

As we can see the Probability that given is Dog is around 99 percentage. Since it is Award winning Pre-trained network VGG-16 model it gives high accurate results.

3.2 YOLO Object detection

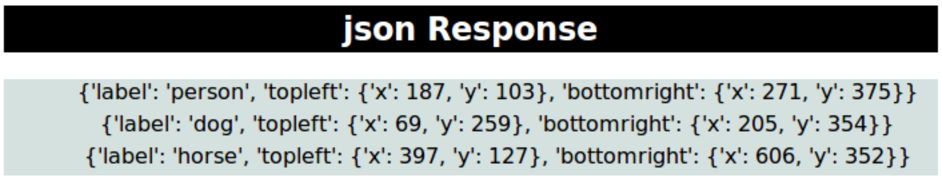
In Order to run the YOLO API also follow the instructions given in **HOW TO RUN THE CODE** session.
Following are the results of YOLO API.

1) Press **Choose files** and select any image from testdata and press **Detect** and wait for few seconds you will observe you result.



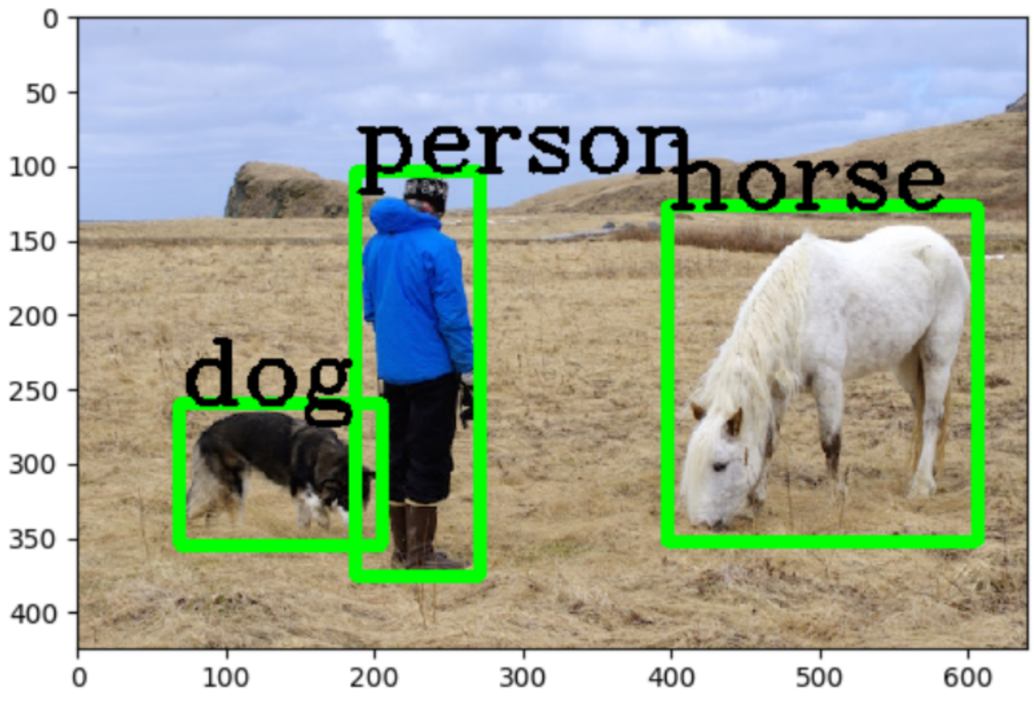
JSON response

Once we press the detect button after few seconds we get JSON response with image location coordinates,label and output imageas follows



json Response

```
{'label': 'person', 'topleft': {'x': 187, 'y': 103}, 'bottomright': {'x': 271, 'y': 375}}
{'label': 'dog', 'topleft': {'x': 69, 'y': 259}, 'bottomright': {'x': 205, 'y': 354}}
{'label': 'horse', 'topleft': {'x': 397, 'y': 127}, 'bottomright': {'x': 606, 'y': 352}}
```



Bibliography

[1] https://medium.com/@udemeudofia01/basic-overview-of-convolutional-neural-network-cnn-4fcc7dbb4f17

[2] https://link.springer.com/article/10.1007/s13244-018-0639-9

[3] https://medium.com/@y1017c121y/how-does-yolov2-work-daaaa967c5f7

[4] https://en.wikipedia.org/wiki/Flask(webframework)

[5] https://fr.slideshare.net/KirillEremenko/deep-learning-az-convolutional-neural-networks-cnn-module-2

[6] Other Medium Posts