Dog Breed Classification using YOLO

Durga Yasasvi IMT2016060
Siddarth Reddy IMT2016037
Srujan Swaroop IMT2016033
Rohith Yogi IMT2016072

Abstract: The aim of this project is to detect and classify dog breeds. Unlike treating this as a fine grained image classification problem only, we bring YOLO to perform both detection and classification of the dog breed . We used YOLO as a simple, sequential and scalable approach for dog detection and breed classification. We applied transfer learning methods and worked on pretrained networks on our data-set. We then experimented on different pretrained networks like VGG16, AlexNet to obtain a good accuracy.

1. Introduction

This project aims to identify dog breeds from the images. This is a fine grained detection and classification problem and uses YOLO like model(where the pre-trained network used is VGG-16 networks instead of conventional Darknet) to localize bounding boxes and classify the detected dog. Convolutional neural networks (CNN) have been used to great effect in applications such as object classification. In many situations, we can imagine the features (both low-level and higher-level) that are learned by the CNNs in the process of training. However, when the objects the CNNs are trying to categorize share many similar features, such as the breeds of dogs(All breeds share similar body features and overall structure, hence differentiating the right breed is a difficult problem. Also, there is high intra breed variation and low inter breed variations.), it becomes hard to imagine the specific features that CNNs must learn in order to categorize these dogs correctly. It is therefore interesting to see how well CNNs with fully connected layers can perform on only dog breeds, compared to normal classification of different objects like cat/dog, car/bike etc.This is where the model like ours falls into place.

So for YOLO we tried two different networks for the classification to see which one performs better over the other and implemented each of them from scratch. For network we used basic DARKNET(same as the one used for YOLO-original) and pretrained networks like VGG-16 and AlexNet as building blocks for the YOLO network followed by FC for better detection. We will discuss more about the architecture in later sections. Then using the transfer learning the pre-trained model was bulit and using custom training we trained the model on our dataset.

2. Dataset

The Stanford Dogs dataset contains images of 120 breeds of dogs from around the world. This dataset has been built using images and annotation from ImageNet for the task of fine-grained image categorization. Contents of this dataset:

- Number of categories: 120
- Number of images: 20,580
- Annotations: Class labels, Bounding boxes
- Number of train Images: 12000
- Number of test images: 8580

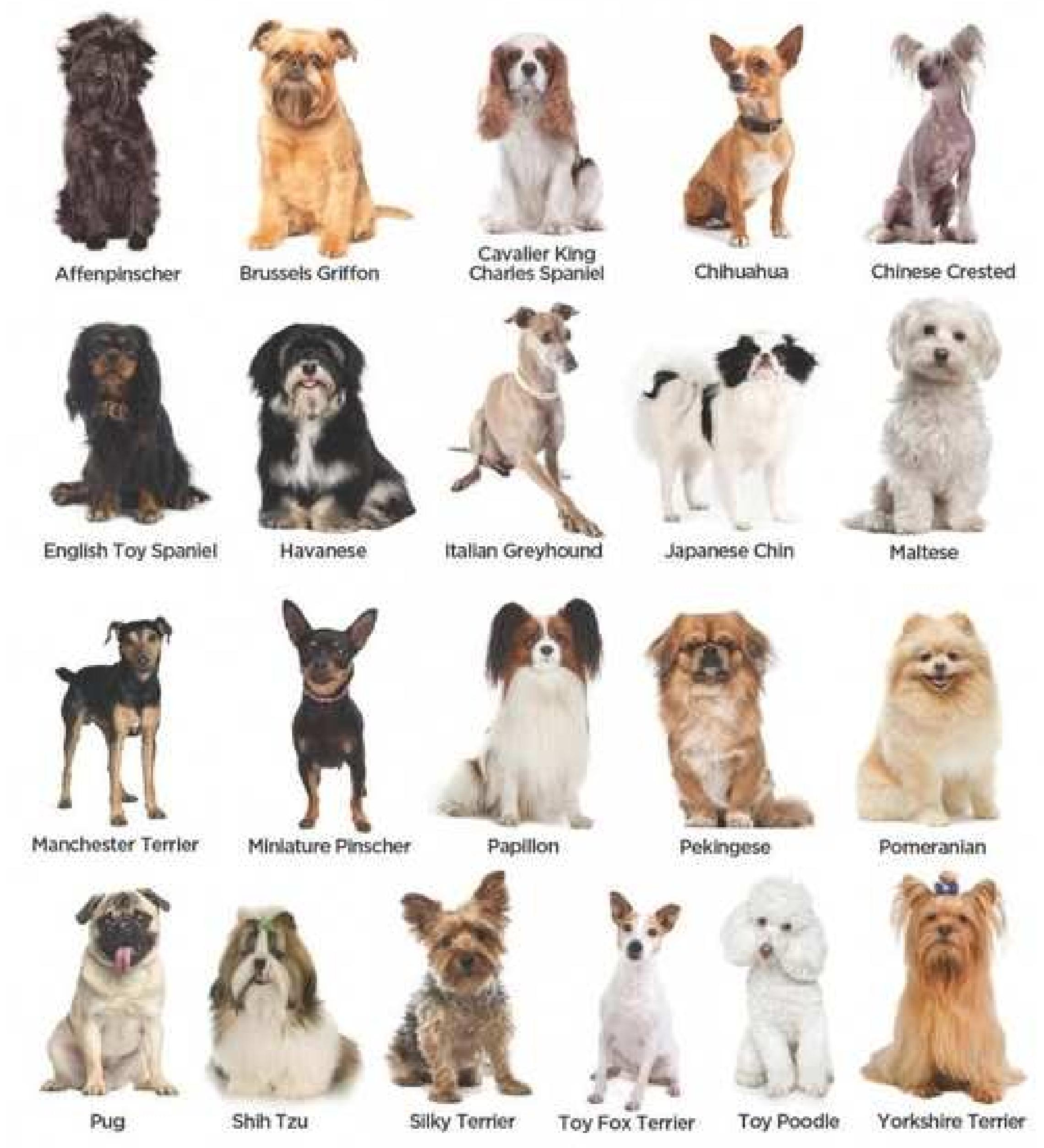


Fig. 1: Stanford Dog Breeds Dataset.

3. Approach

3.1. Motivation

Our main motivation for this project is to learn and how to use/train the Darknet, VGG-16 and AlexNet network. So, we tried these three different networks for this dog breed classification task. We wanted to evaluate the performance of these three architectures and see which performs better. The reason for selecting Darknet is that it was used in the YOLOv1-paper and VGG-16 is because comparatively a better framework to other networks like Alexnet. As VGG16 has higher accuracy on Imagenet data set.

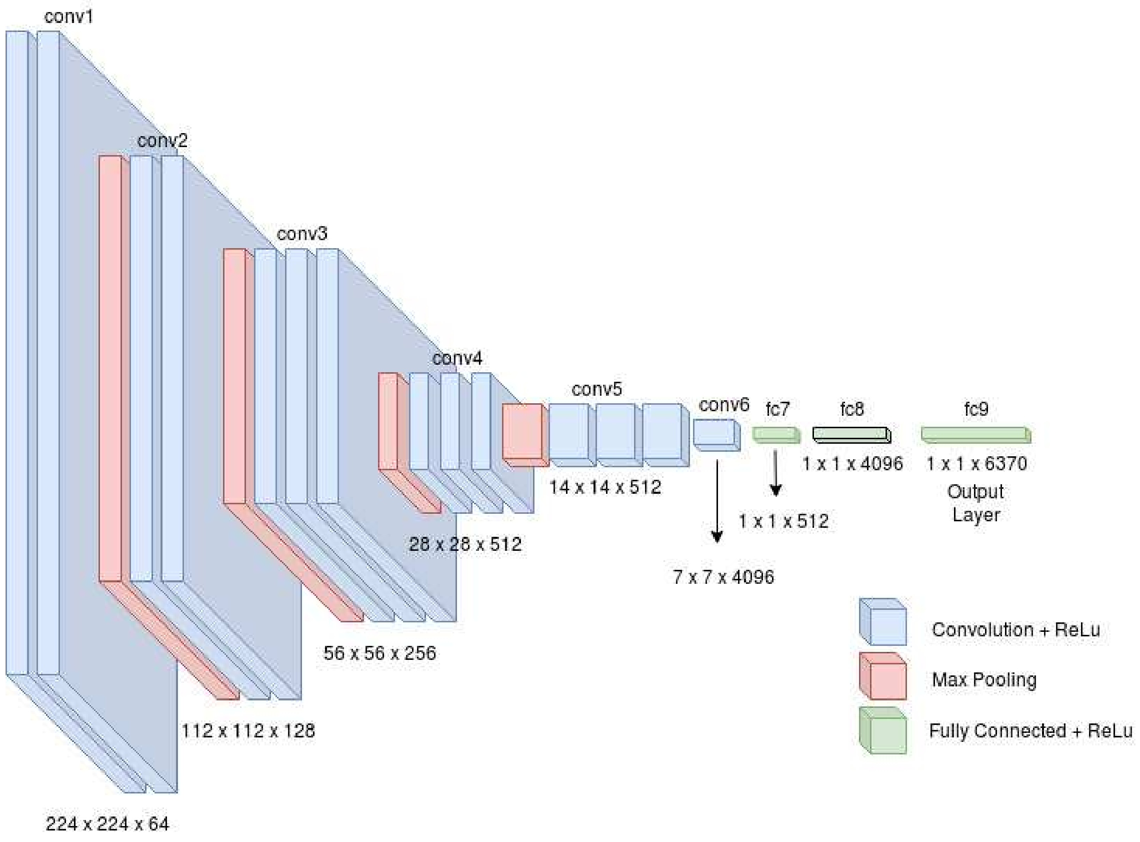


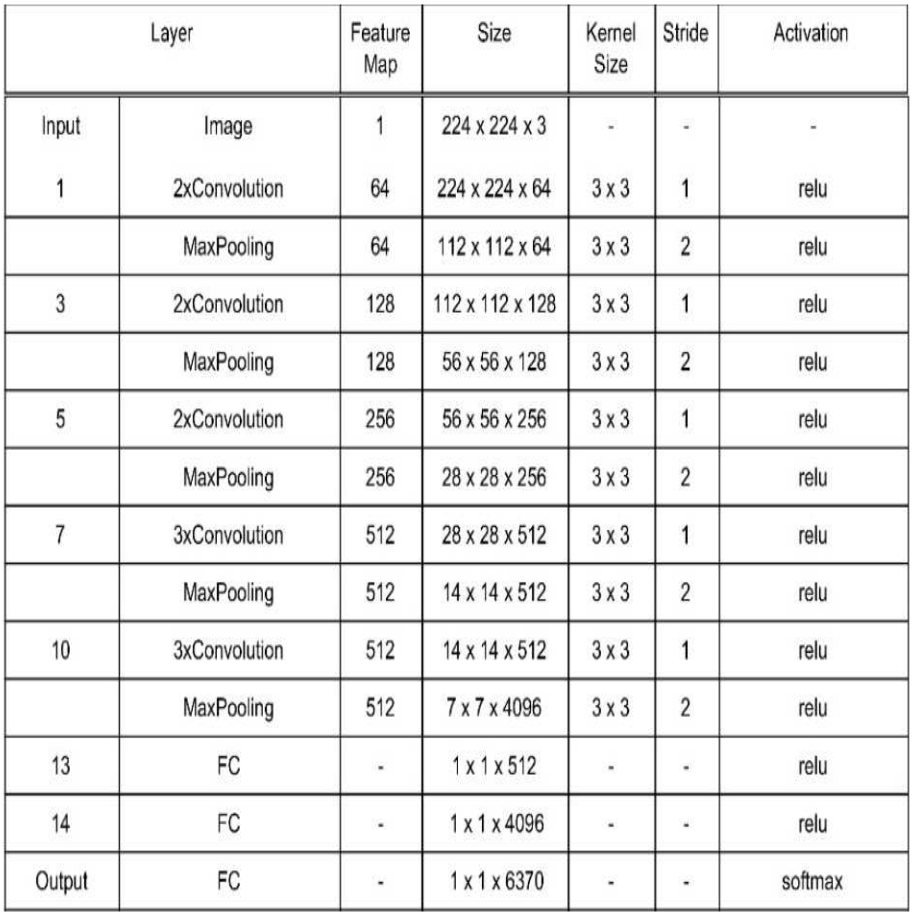
Fig. 2: Our Network Architecture

As we are using YOLO here, our model learns the 5 encoding for every dog in the image which are x, y, w, h and $confidence$. These 5 parameters represent the bounding box co-ordinates and the confidence scores for those boxes (which contains dogs). Here the confidence prediction represents the IOU between the predicted box and the ground truth box.

3.2. Network Architecture

We follow the same architecture as what the YOLO implemented as base for our design. YOLO's architecture has 24 layers followed by 2 fully connected layers. In YOLO, the first 20 layers are same as of Darknet framework. The pre-trained weights are obtained by training the network on ImageNet dataset.

Our network follows the same architecture except that the Darknet framework is replaced with other pretrained networks. As mentioned earlier, we used VGG16 and AlexNet framework as pretrained networks which were trained on ImageNet dataset. Hence the first 20 layers are replaced with VGG16. As it is dog breed specific classification task, the variability between classes is very low hence the requirement of number of convolutional layers after pretrained network is not abundant as features of the dogs are assumed to be learnt during pretraining of VGG-16 on ImageNet dataset. Therefore the intention of adding layers turned more towards object detection rather than pure classification. Hence number of convolutional layers added are less as compared to fully connected layers as bounding coordinates needed to be regressed
Therefore our architecture is described as follows:



Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	224 x 224 x 3	-	-	-
1	2xConvolution	64	224 x 224 x 64	3 x 3	1	relu
	MaxPooling	64	112 x 112 x 64	3 x 3	2	relu
3	2xConvolution	128	112 x 112 x 128	3 x 3	1	relu
	MaxPooling	128	56 x 56 x 128	3 x 3	2	relu
5	2xConvolution	256	56 x 56 x 256	3 x 3	1	relu
	MaxPooling	256	28 x 28 x 256	3 x 3	2	relu
7	3xConvolution	512	28 x 28 x 512	3 x 3	1	relu
	MaxPooling	512	14 x 14 x 512	3 x 3	2	relu
10	3xConvolution	512	14 x 14 x 512	3 x 3	1	relu
	MaxPooling	512	7 x 7 x 4096	3 x 3	2	relu
13	FC	-	1 x 1 x 512	-	-	relu
14	FC	-	1 x 1 x 4096	-	-	relu
Output	FC	-	1 x 1 x 6370	-	-	softmax

Fig. 3: Our Network Design with VGG 16 pretrained network

1) 12 convolutional layers from pre-trained VGG-16 / 8 convolutional layers from pre trained network AlexNet.
2) Followed by one convolutional layer intended to train during transfer learning.
3) Followed by 2 fully connected layers with random weights(for transfer learning using our data set).
4) And followed by final output layer which outputs a tensor which has following information.

- predictions for each grid cell for B bounding boxes
- for each bounding box $\{x, y, w, h\}$ is predicted along with the corresponding confidence score which tells about how confidence about how accurate the predicted bounding box is.
- Along with above predictions each grid cell also predicts C class conditional probabilities.

This information is encoded as $S * S * (B * 5 + C)$ tensor. As S is size of each grid,B is number of bounding boxes predicted for each grid cell ,C is the number of classes.
Our grid size(S)=7 and number of bounding boxes predicted for each grid(B)=2 and number of dog breed classes(C)=7.
Therefore our output tensor is in the form $7 * 7 * (2 * 5 + 120)$
$\Rightarrow 49 * (10 + 120) = 49 * 130(16370)$.
As our dataset has high intra class variation, we added a dropout layer with dropout probability 0.5 just before final output layer which helps as a regularizer.

4. Training

4.1. Transfer Learning

Deep learning requires the ability to learn features automatically from the data, which is generally only possible when lots of training data is available - especially for problems where the input samples are very high-dimensional, like images.

The transfer learning technique is using an existing CNN feature extraction and the associated trained network weights, and transferring the learned general knowledge from one problem to other one. Here we are using the pre-trained VGG-16 and AlexNet network (obtained weights from large data-set ImageNet) as our transfer learning model as it as already extracted some information(features) about the objects in Imagenet. For transfer-learning, we remove the three layers, and then treat the rest of the network as a fixed feature extractor for the our problem-specific data-set.

4.2. Fine-Tuning

As mentioned in previous section, our data is more similar to the ImageNet data and also the Dog breed data-set is arguably large enough. Hence we train on few extra layers which weren't part of the pretrained network and then we freeze these pretrained layers and fine tune the weights of by continuing the back-propagation with a lesser learning rate.

In CNNs the lower layers contain generic features where as the higher layers contain features specific to the class. So,in our model the lower layers contain may be features of dog and higher layer contains breed specific information. So prevent over-fitting and preserve generalization we freeze the lower layers of pre-trained network. To implement this property we enforce a learning parameter such that loss is not propagated to lower layers because of loss gradient.

$$learning - parameter = 0.0001$$

The pretrained networks VGG16 and AlexNet takes the input image of size 224×224 and 227×227. Our final layer predicts both the class probabilities and box coordinates. Later we use the same normalization functions and activation functions as described in the YOLO paper. Also we follow the same loss as given in the YOLO v1 paper.

4.3. Loss function

The loss function is designed to incorporate the following in order to perform robust object detection:-

- As many grid cells does not contain objects , such grid cells contribution to the loss function must be considerably less than that which contains object hence :

$$\lambda_{coord} = 5$$

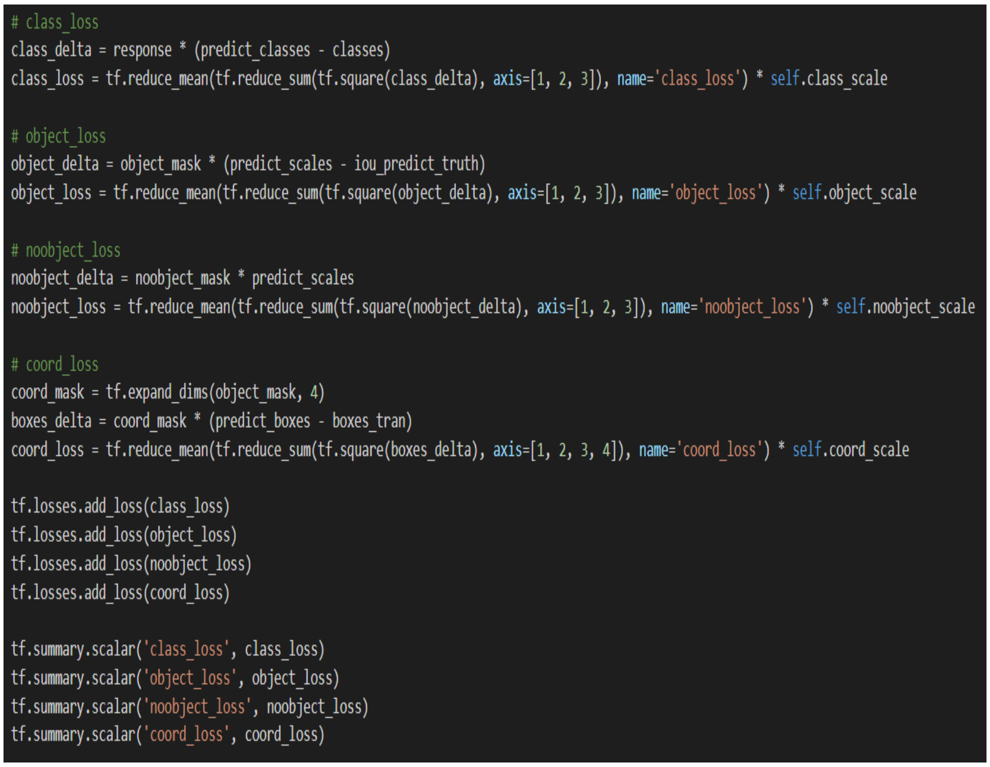
$$\lambda_{noobj} = 1$$

- The model predicts multiple bounding boxes per grid cell the weightage should be give to the bounding box which has maximum confidence score,.

$$1_{ij}^{obj} = 1$$

- The above parameters ensures that loss functions penalizes the grid cell if object is the present in that grid cell.
- Also it penalizes the most responsible bounding box which has highest IOU and weighted by the corresponding class which has highest class conditional probability class scale

$$C_{ij} = 2$$



```
# class_loss
class_delta = response * (predict_classes - classes)
class_loss = tf.reduce_mean(tf.reduce_sum(tf.square(class_delta), axis=[1, 2, 3]), name='class_loss') * self.class_scale

# object_loss
object_delta = object_mask * (predict_scales - iou_predict_truth)
object_loss = tf.reduce_mean(tf.reduce_sum(tf.square(object_delta), axis=[1, 2, 3]), name='object_loss') * self.object_scale

# noobject_loss
noobject_delta = noobject_mask * predict_scales
noobject_loss = tf.reduce_mean(tf.reduce_sum(tf.square(noobject_delta), axis=[1, 2, 3]), name='noobject_loss') * self.noobject_scale

# coord_loss
coord_mask = tf.expand_dims(object_mask, 4)
boxes_delta = coord_mask * (predict_boxes - boxes_tran)
coord_loss = tf.reduce_mean(tf.reduce_sum(tf.square(boxes_delta), axis=[1, 2, 3, 4]), name='coord_loss') * self.coord_scale

tf.losses.add_loss(class_loss)
tf.losses.add_loss(object_loss)
tf.losses.add_loss(noobject_loss)
tf.losses.add_loss(coord_loss)

tf.summary.scalar('class_loss', class_loss)
tf.summary.scalar('object_loss', object_loss)
tf.summary.scalar('noobject_loss', noobject_loss)
tf.summary.scalar('coord_loss', coord_loss)
```

Fig. 4: Our Code demonstration for loss function

Hence Model is optimized using the Loss Function:

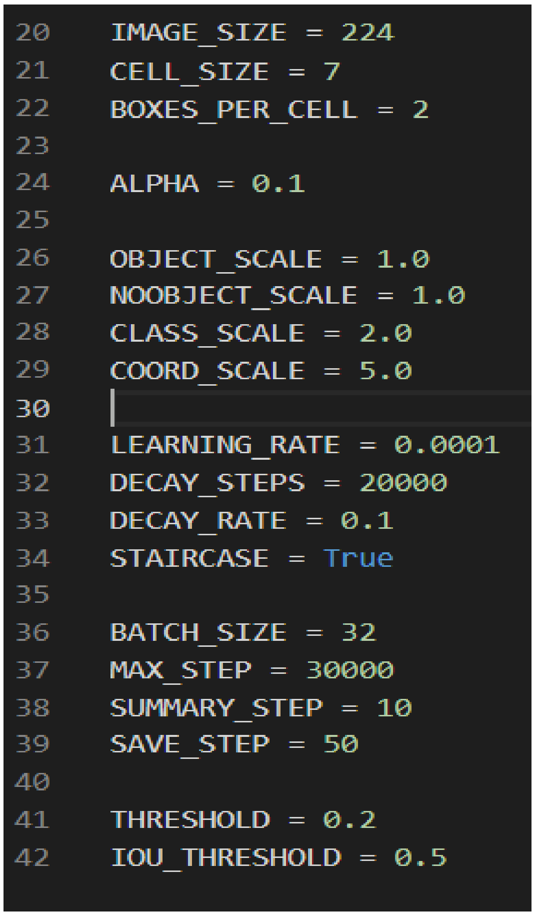
$$\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2]+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}[(\sqrt{w_i}-\sqrt{\hat{w}_i})^2+\sqrt{h_i}-\sqrt{\hat{h}_i})^2]$$

$$+\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}[(C_i-\hat{C}_i)^2+\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}1_{ij}^{obj}[(C_i-\hat{C}_i)^2+\sum_{i=0}^{S^2}\sum_{j=0}^{B}C_{ij}*1_{i}^{obj}[(p_i(c)-\hat{p}_i(c))^2$$

4.4. Training configuration specifications

The training is carried through the following mentioned configuration :

- Model trained for 110 epochs .
- Each epoch consisting of 300 iterations
- Total number of iterations for which the model trained is 33,350 iterations descent is carried through mini batches of size-32(Mini batch gradient descent)
- Each image sample is used in training in normalized form.
- Batch normalization is applied for all the batches and batches of image samples are selected in random fashion.



```
IMAGE_SIZE = 224
CELL_SIZE = 7
BOXES_PER_CELL = 2

ALPHA = 0.1

OBJECT_SCALE = 1.0
NOOBJECT_SCALE = 1.0
CLASS_SCALE = 2.0
COORD_SCALE = 5.0

LEARNING_RATE = 0.0001
DECAY_STEPS = 20000
DECAY_RATE = 0.1
STAIRCASE = True

BATCH_SIZE = 32
MAX_STEP = 30000
SUMMARY_STEP = 10
SAVE_STEP = 50

THRESHOLD = 0.2
IOU_THRESHOLD = 0.5
```

Fig. 5: Our Code demonstration for training configuration

5. Testing

5.1. Inference

For testing we pass the test image to the network and the network outputs 1370 tensor which consists of bounding box information,confidence score,class conditional probability distribution.So to extract the exact bounding box for the dog and it's corresponding dog breed class,it requires an extent of post processing like confidence thresholding and non maximal suppression .

5.2. Confidence thresholding

Since our model predicts multiple bounding boxes per grid cell, but out of those, most of them will have a very low confidence score associated, hence we only consider predictions above a certain threshold confidence score.So our adapted threshold value is :

$$confidence - threshold = 0.2$$

So all boundary boxes with confidence scores less than 20$\%$ are not considered as candidates for optimal bounding boxes. We run the original image through our model and this what the object detection algorithm returns after confidence thresholding.

5.2.1. Non-maximum Suppression

But still there are many candidate bounding boxes above confidence threshold and we should use non maximal suppression and obtain the optimal bounding box among all candidate bounding

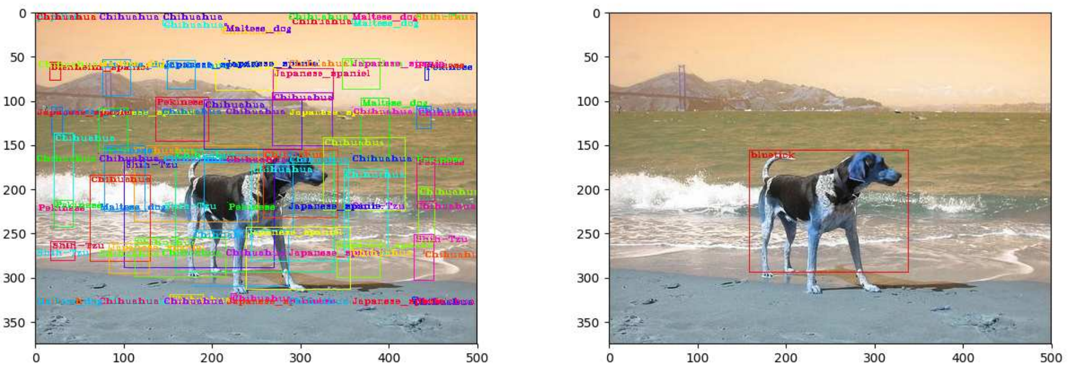
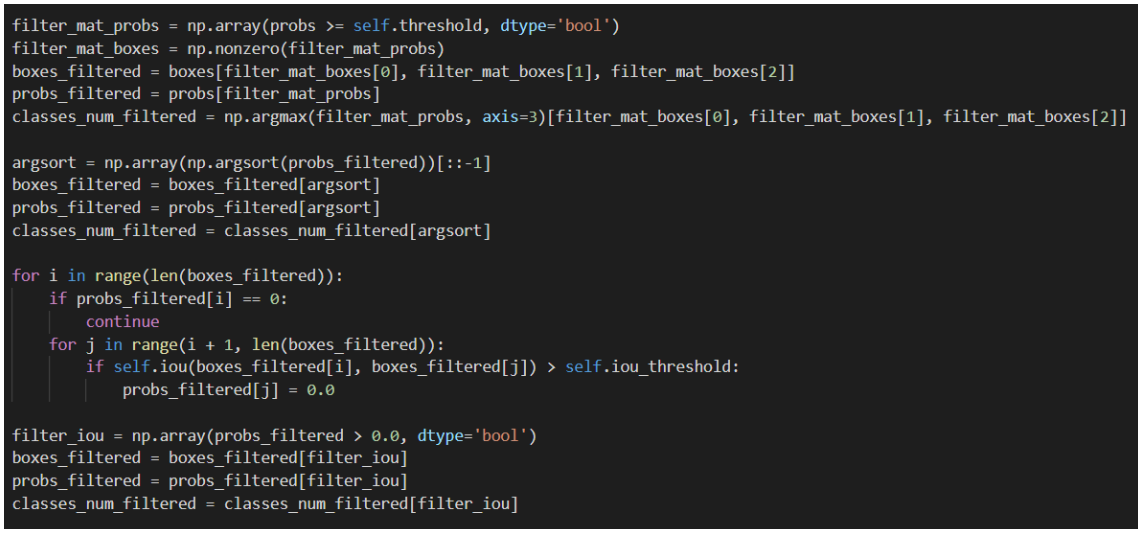


Fig. 6: Image before and after the Non Maximum Suppression



```
filter_mat_probs = np.array(probs >= self.threshold, dtype='bool')
filter_mat_boxes = np.nonzero(filter_mat_probs)
boxes_filtered = boxes[filter_mat_boxes[0], filter_mat_boxes[1], filter_mat_boxes[2]]
probs_filtered = probs[filter_mat_probs]
classes_num_filtered = np.argmax(filter_mat_probs, axis=3)[filter_mat_boxes[0], filter_mat_boxes[1], filter_mat_boxes[2]]

argsort = np.array(np.argsort(probs_filtered))[::-1]
boxes_filtered = boxes_filtered[argsort]
probs_filtered = probs_filtered[argsort]
classes_num_filtered = classes_num_filtered[argsort]

for i in range(len(boxes_filtered)):
    if probs_filtered[i] == 0:
        continue
    for j in range(i + 1, len(boxes_filtered)):
        if self.iou(boxes_filtered[i], boxes_filtered[j]) > self.iou_threshold:
            probs_filtered[j] = 0.0

filter_iou = np.array(probs_filtered > 0.0, dtype='bool')
boxes_filtered = boxes_filtered[filter_iou]
probs_filtered = probs_filtered[filter_iou]
classes_num_filtered = classes_num_filtered[filter_iou]
```

Fig. 7: Our Code demonstration for IOU thresholding and non maximum suppression

boxes for the dog present in the image. The main purpose of non maximum suppression is to pick one of the multiple candidates for object. So it eliminates some candidates that are different detections of the same object. It makes sure that the object is identified only once. The candidate bounding boxes are eliminated iteratively through following procedure:-

- Now we define a certain IOU-threshold, for every candidate bounding box if it's IoU with ground truth $>$ IoU-threshold then it is considered as a True Positive, else it is considered a False positive.
- Discard all false positive bounding boxes and proceed further processing with remaining true positive bounding boxes

$$IOU - threshold = 0.5$$

- For all remaining True Positive candidate bounding boxes for the object, select the box with the largest confidence score among all remaining candidate boxes.
- item Then discard with IoU overlap $\geq$ 0. 5 with bounding box with the highest confidence.

After Non maximum suppression, we output the optimal bounding box fit for the dog present in the image and the class corresponding to the highest class conditional probability. In the above section IoU refers to Intersection over Union,which is defined as atio between the intersection and the union of the predicted boxes and the ground truth boxes.

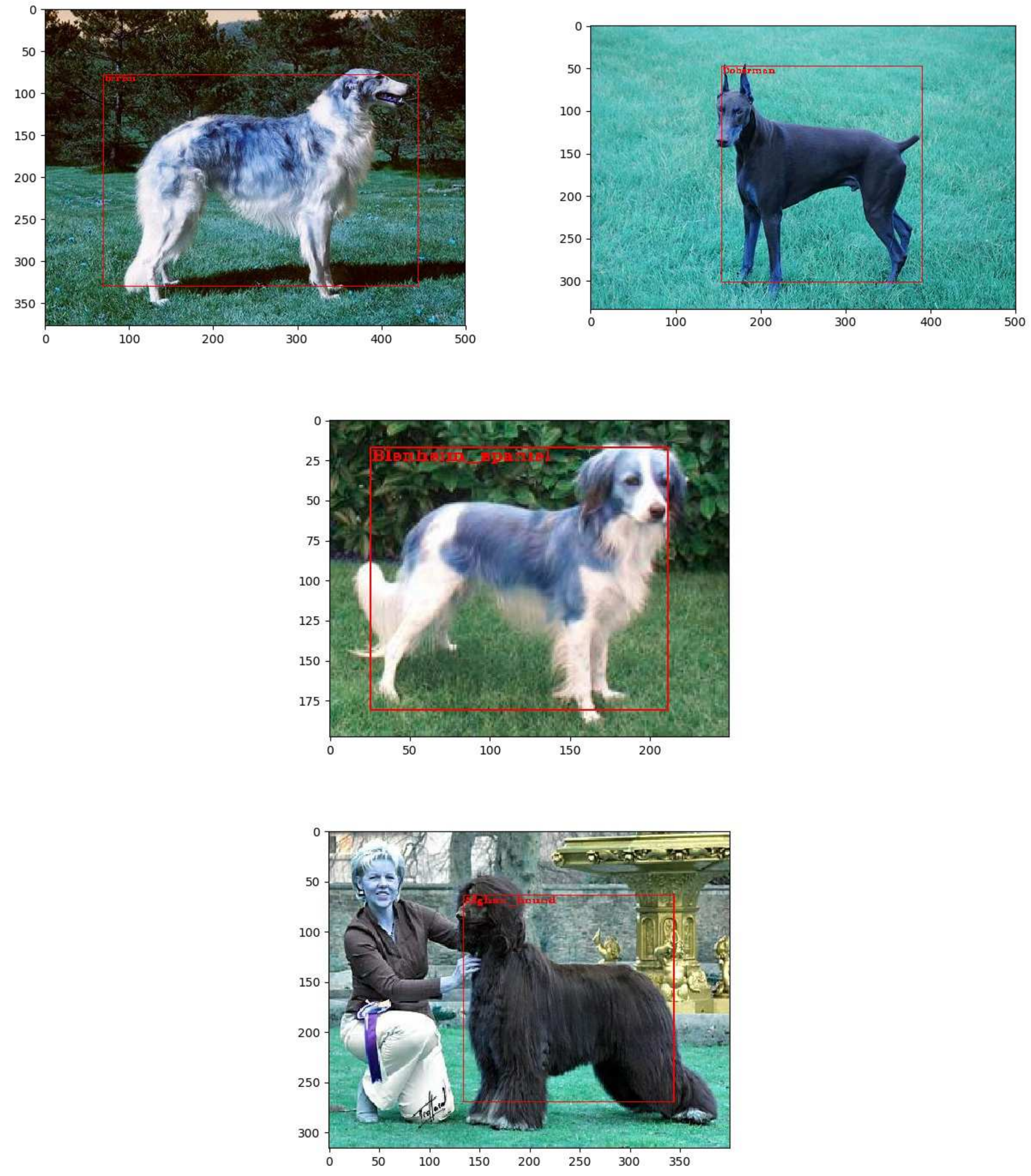


Fig. 8: Samples of dog breed images detected by our model (VGG-16 YOLO)

6. Results

Below are a few predictions (bounding box of the dog and its label) made by the model.

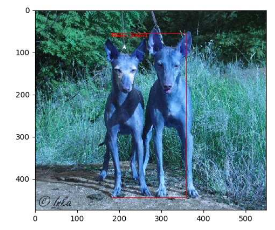


Fig. 9: Our

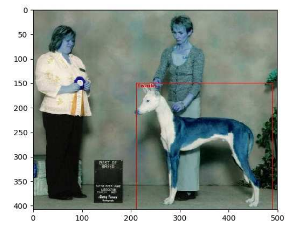


Fig. 10: Our

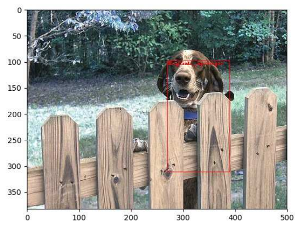


Fig. 11: Our

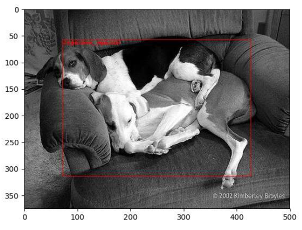


Fig. 12: Our

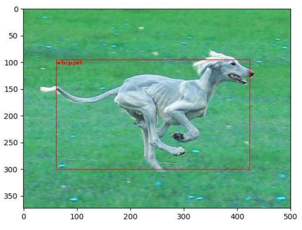


Fig. 13: Our

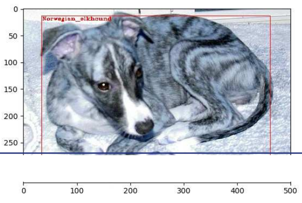
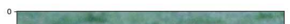


Fig. 14: Our



6.1. Comparisons

Other than VGG-16 pretrained network we have also explored few other pretrained nets with the similar architecture and recorded the observations.The networks explored by us are as follows:-

- VGG-16
- Darknet
- Alexnet

These are the some of the deductions from of our observations of models on our dataset:-

1) VGG-16 was significantly more accurate than remaining pretrained networks with 72.12% mAP
2) But in terms of computation darknet was effectively faster than VGG-16 but was not as accurate as VGG-16 with mAP of 65.89%
3) Compared to the rest Alexnet exhibited poor performance with 58.23% mAP,

Note: We used Darknet Reference architecture.
Here mAP is referred to mean average precision

7. Conclusion

The variation of breeds in dogs is extremely high and also the variation within breeds is also very significant with various structural and physical complex aspects like occlusion,posture,lighting. Hence in order to obtain robustness in object detection and classification among breeds the mode need to be very accurate.Hence we made a conditional trade-off of accuracy over speed,which essentially enforced us to select VGG-16 over other networks which is significantly accurate than other nets(including Darknet) with compromise speed as compared to Darknet.But relatively VGG 16 would stand as a best preferable choice for the given dog breed classification for for it's high end efficiency and reasonable speed,which is also projected by the above mentioned results in results section.

8. References

Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. **You Only Look Once**: Unified, Real-Time Object Detection, University of Washington, Allen Institute for AI, Facebook AI Research.

https://arxiv.org/pdf/1506.02640.pdf

https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006

https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c

https://github.com/pjreddie/darknet/wiki/YOLO:-Real-Time-Object-Detectiontraining-yolo

https://towardsdatascience.com/Understanding mAP