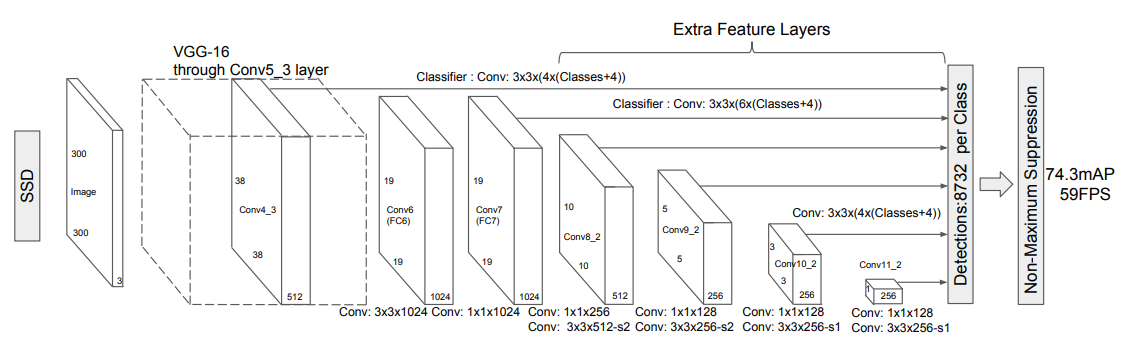
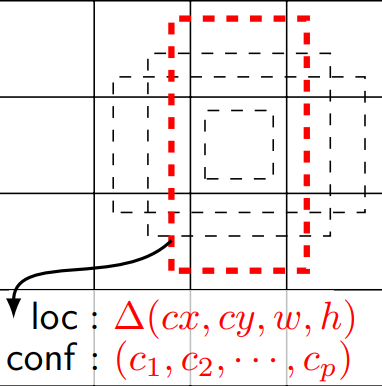
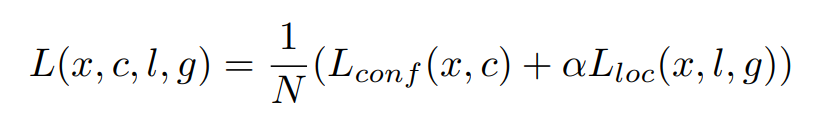
Single Shot Detector:

When we use SSD we only need one **single  
shot to detect** multiple objects within the image.  
  
Our model is SSD with Mobile-Net backbone, which means that it’s an improvement of the SSD architecture or in other words the backbone of the network is for feature extraction that are to be used for prediction in the later stages of the network. So as a result we will explain on the SSD network and then on the Mobile-Net network and will see if it is an improvement over the default.

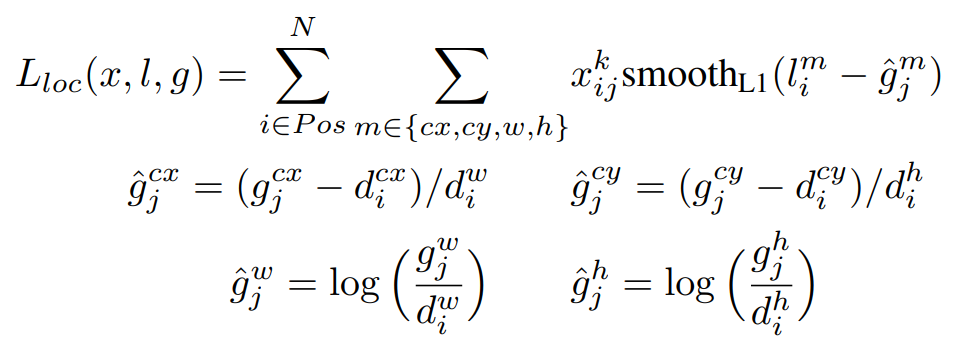


  
SSD architecture:   
the network is constructed by using the base layer (which the default is vgg-16 and ours will be Mobile-Net) to extract features and moving to a large array of smaller and smaller kernels of convoluted layers to detect objects.  
Each layer will construct 4 bounding boxes, so if the layer k is m x n there will be (classes + 4) kmn outputs.  
e.g. let’s take layer conv4\_3, k = 4, m, n = 38 and let’s assume there are 10 classes.  
Output: 38 x 38 x 4 x (10 + 4) = 80864  
the amount of bounding boxes = 38 x 38 x 4 = 5776

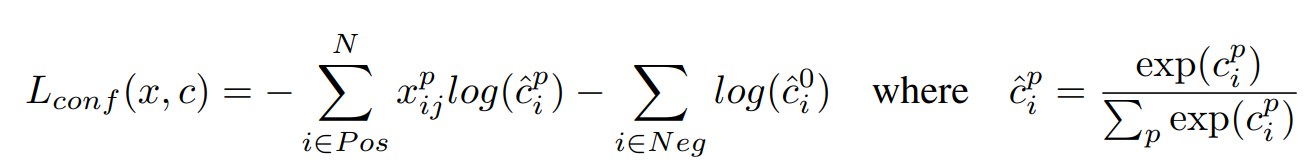
SSD Loss Function:



The localization loss is the mismatch between the ground truth box and the predicted boundary box. SSD only penalizes predictions from positive matches. We want the predictions from the positive matches to get closer to the ground truth. Negative matches can be ignored.



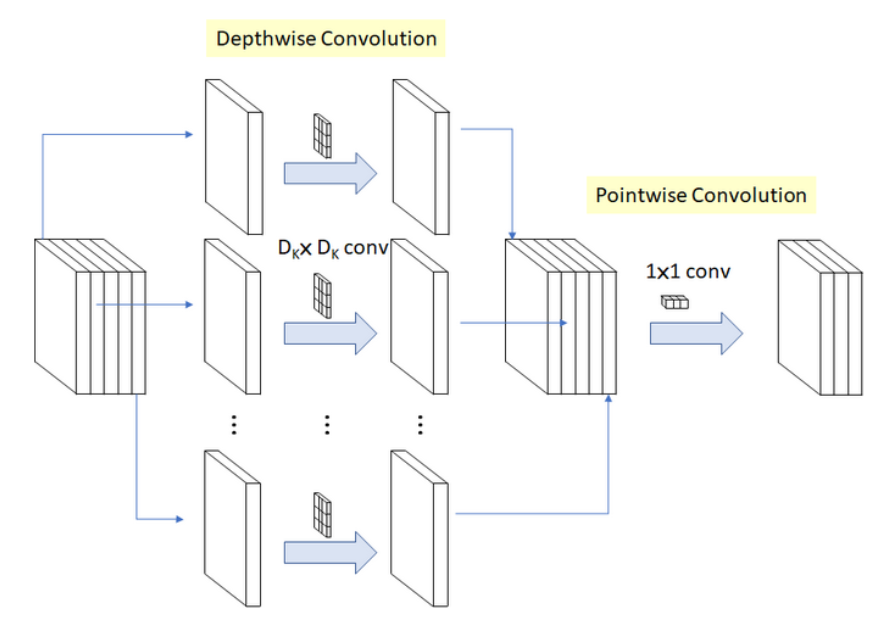
The confidence loss is the loss of making a class prediction. For every positive match prediction, we penalize the loss according to the confidence score of the corresponding class.

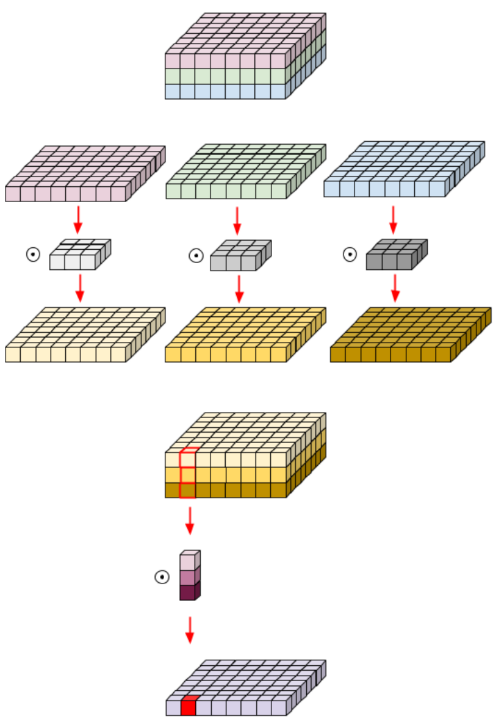


In this network the base layer can be change according to the need of more accurate results or less computationally heavy.

Mobile-Net:

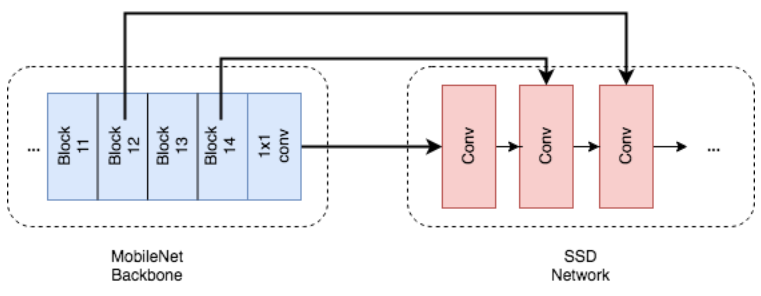
As the name suggests this neural network is meant to be used for mobile devices and by so it is less computationally heavy but accurate none the less.



Mobile-Net architecture:  
the mobile net network uses **Depth-wise separable convolution.  
Depthwise Separable Convolution** splits the computation into two steps: depthwise convolution applies a single convolutional filter per each input channel and pointwise convolution is used to create a linear combination of the output of the depthwise convolution. The comparison of standard convolution and depthwise separable convolution is shown to the right. As we can see we first divide the plain to his layers and conduct a conv layer on each point and then combine each resulting plain and perform a 1 x 1 conv layer on each resulting plain point.

Mobile-Net also uses global hyperparameters to effectively reduce the computational cost further.

* Width Multiplier: Thinner Models:   
  For each layer, the width multiplier α will be multiplied with the input and the output channels (N and M) in order to narrow a network.  
    
  Here α will vary from 0 to 1, with typical values of [1, 0.75, 0.5 and 0.25]
* Resolution Multiplier: Reduced Representation:  
  For a given layer, the resolution multiplier ρ will be multiplied with the input feature map. we can express the computational cost by applying **width multiplier** and **resolution multiplier** as:  
  

Mobile-Net SSD:   
the network will be as follows:  
first the Mobile-Net as the base layer of the network and then the resulting inputs will be given to the later layers in the SSD network.  
  


**Part 2: Implementation**

**Dataset**

We used the Dogs dataset from the open images dataset, containing more than 30,000 dogs images separated into train, validation, and test sets.

**Training Pipeline**

We followed the implementation of <https://github.com/qfgaohao/pytorch-ssd>. We used the MobilenetV1 SSD architecture.  
For the transfer learning, we downloaded the pre trained classification Mobilenet weights and used it to initialize the weights of the SSD base-net.  
The augmentation used in training were random mirroring of the image and the bounding boxes, random cropping of the image together with the filtering of cropped out bounding boxes, resizing the image and bounding boxes to smaller size and filling the border with zeros, and photometric distort of the image (contrast, saturation, hue).

We run 2 main experiments:

1. All the parameters are trained.
2. The base network parameters are frozen.

In both experiments, we used batch size 32, cosine learning rate scheduler, learning rate of 0.01, and a 0.001 learning rate for the base network.  
The training loss consisted of a classification loss and a regression loss as mentioned in part 1.   
To get the curves, we parsed the training logs (see files parse.py, parse\_val.py).

**The resulting training curves:**Experiment 1 (Training all parameters):

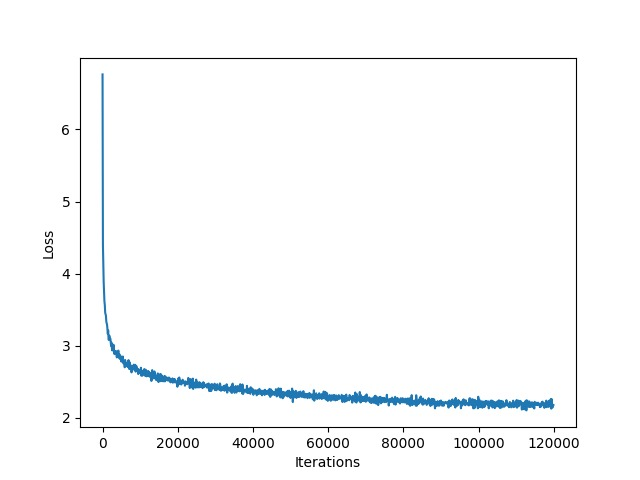
A picture containing shape

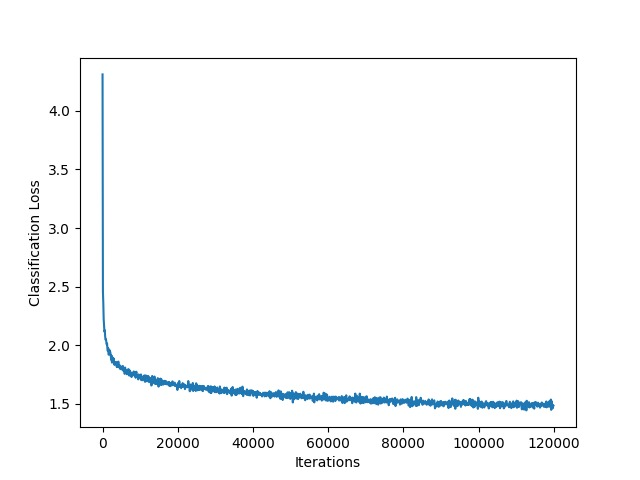
Description automatically generatedShape

Description automatically generated with medium confidence

Shape

Description automatically generated with medium confidence

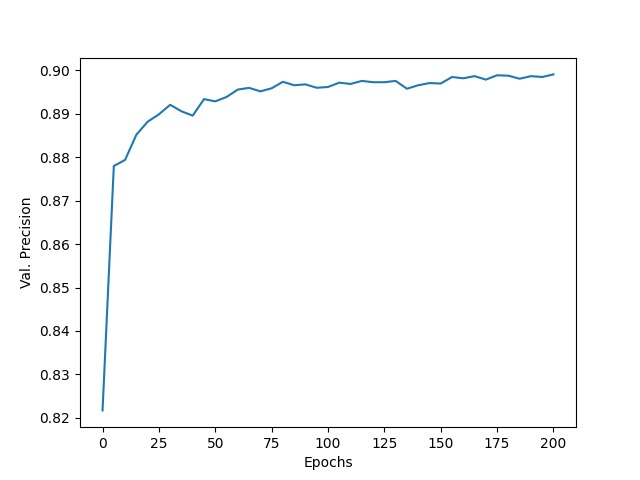
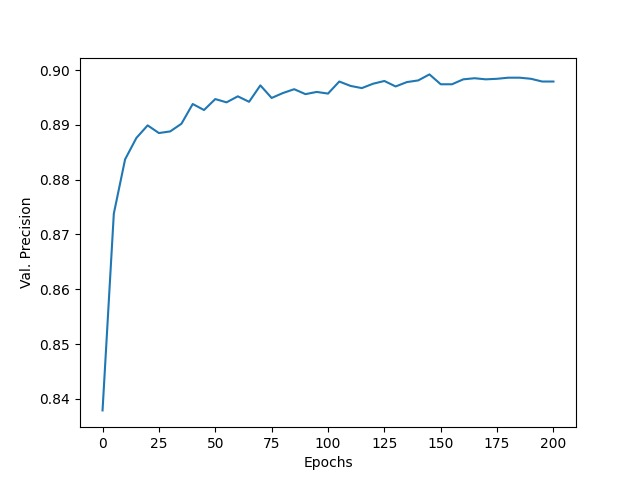
The training curves of experiment 2 (freezing the base net):



**Precision during training**

Finally, we compute the precision of the models during training.

The precision is defined as the ratio between the number of detected dogs and the total number of dogs in an image.

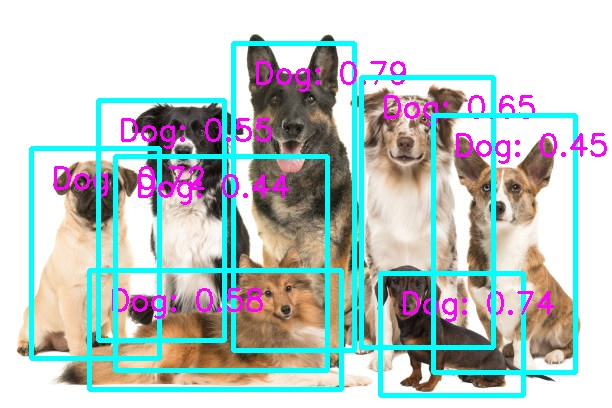
To do so we run the eval\_ssd.py script while changing the dataset type to “validation” for each checkpoint saved after each validation run.  
(Right picture is the second experiment and the left is the first experiment)

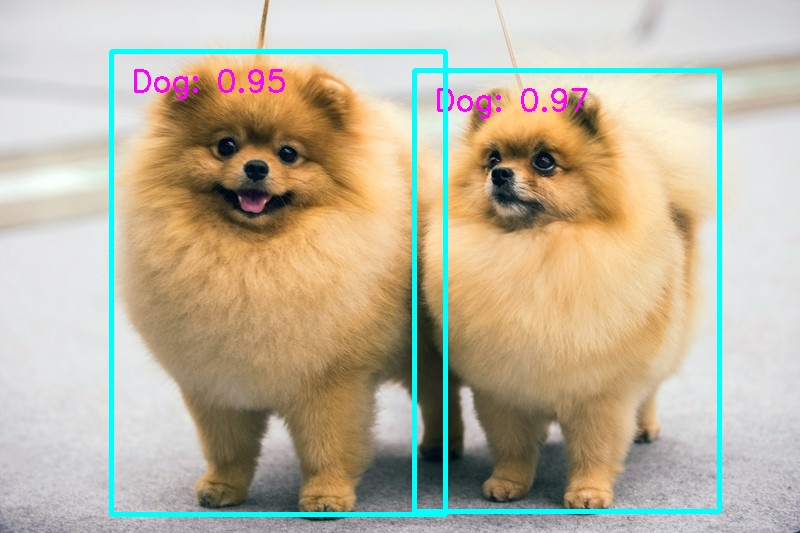
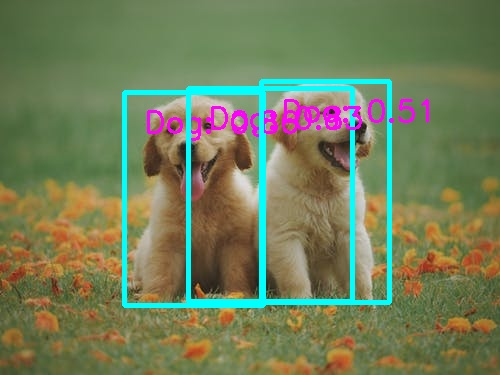
**Results**

We run evaluation on the test set with the final model.  
The precision obtained for experiment 1 was 0.89981 -> 89.981%.  
The precision obtained for experiment 2 was 0.89948 - > 89.948%.  
As can be seen by the training curves and precision, both models achieve  
comparable results, with experiment 1 obtaining slightly better precision.

Example results:

A picture containing ground, mammal, standing, sandy

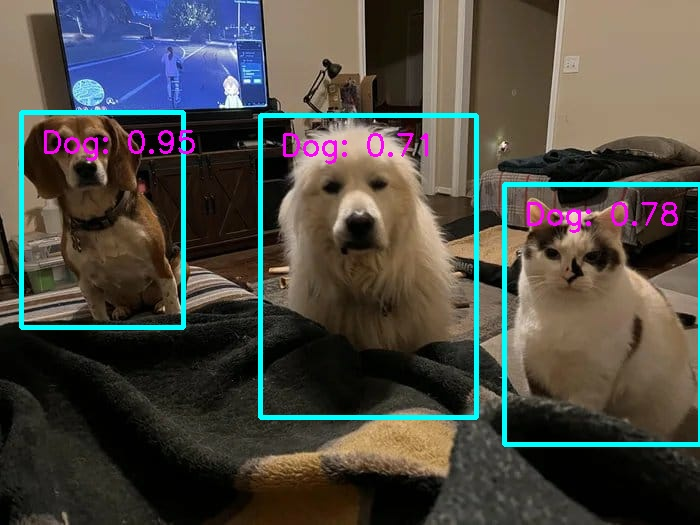
Description automatically generatedA picture containing text, dog, standing, mammal

Description automatically generated

**Fail cases:**

Since we train the models for only dogs, it is possible that there were only a few images of cats (that should not be counted as dogs), and the model did not learn to this extent the difference between dogs and cats. Here is an example:

A picture containing outdoor, ground, aquatic bird, sandy

Description automatically generated

Another case is when the dog in the picture is small, and in the following example (taken from one of the submitted videos) its colors are also similar to the background.

Another problem that we noticed was in pictures that contains many little dogs. Sometimes a single box contains multiple dogs. We suspect it might be related to the NMS performed after the detection, which unite bounding boxes with large overlap.

A group of dogs in a cage

Description automatically generated with medium confidence