<u>ASSIGNMENT-5</u>

Bhogi Suraj Krishna **180050024**

Q1.3)

<u>MNIST</u> : Topology - Two fully connected Layers with batch_size=**20**, epochs=**5**, learning_rate=**0.02**

784(input) **--->** Fullyconnected **--->** 20 **--->** Fullyconnected **--->** 10(output)
(**relu** activation) (**softmax** activation)
weights_dimensions=[784,20] weights_dims=[20,10]

Seed	3	4	5	6	7	10	60	200	335	1234
Accuracy	92.9	93.2	93	93.5	93.3	93.3	93	92.9	93.2	93.4

As can be observed, the average accuracy achieved is about **93%**

<u>CIFAR10</u> : Topology is convolution layer followed by and average pooling and then flattening out to send to a final fullyconnected layer with batch_size=**10**, epochs=**10**, learning_rate=**0.15**
[3,32,32] = Input or image size and 10=output size(number of classes/digits)

[3,32,32] **--->** Convolution **--->** [10,15,15] **--->** Averagepooling **--->**
(filter_size=[3,3], num_filters=10) (filter_size=[3,3], stride=3)
stride=2,**relu** activation

[10,5,5] **--->** flattenlayer **--->** 250 **--->** fullyconnected **--->** 10(output nodes)
(softmax activation)
weights_dims=[250,10]

Seed	1	2	3	4	5	6	54	200	335	1234
Accuracy	41.2	35.8	43	33.9	43.3	42.9	35	41.6	40.3	38.8

So the average accuracy obtained is around **39.6%**

Circle : Experimentally minimal Topology is Two fully connected Layers with softmax activation for both with batch_size=**30**, epochs=**50**, learning_rate=**0.35**

2(input size)	**--->**	Fullyconnected 3 (softmax activation) weights_dims=[2,3]	**--->**	Fullyconnected (softmax activation) weights_dims=[3,2]	**--->**	2(output nodes)

Seed	1	2	3	4	5	10	100	335	1234	12567
Accuracy	97.4	96.9	96.9	97.7	97.4	95.9	96.7	97.9	96.1	97.1

(Average accuracy around **97%**)

I have tried by keeping 2 nodes in hidden layer instead of 3, I got 79,83.7,78.7,82.6(80% is average) for seeds 1,2,3,4. Also I kept 3 hidden nodes but changed first activation to relu, then it is working in most cases but deviating largely for some cases(like 81.1 for seed 5 and 76.7 for seed 12), so I kept softmax and 3 hidden nodes for the robustness.

Explanation for topology: The reason for only needing three hidden nodes may be that the **semicircle fits well in a triangle**, i.e; three lines. Each hidden node represents a linear separator whose line of separations from 3 nodes form a triangle around circle and then finally points inside these lines or triangle are given as +ve and outside are -ve. If 2 hidden nodes, then it forms two parallel lines up and below the circle, so area inside the parallel lines is given as +ve but the triangle gives more accuracy than two parallel lines.

XOR : Experimental minimal Topology is Two fully connected layers with softmax activation for both with batch_size=**15**, epochs=**20**, learning_rate=**0.25**

2(input size)	**--->**	Fullyconnected 4 (softmax activation) weights_dims=[2,4]	**--->**	Fullyconnected (softmax activation) weights_dims=[4,2]	**--->**	2(output nodes)

Seed	1	2	3	4	5	10	34	100	335	1234
Accuracy	99.1	98.1	99.5	99.2	99.6	99.1	98.7	99.5	99.7	99.4

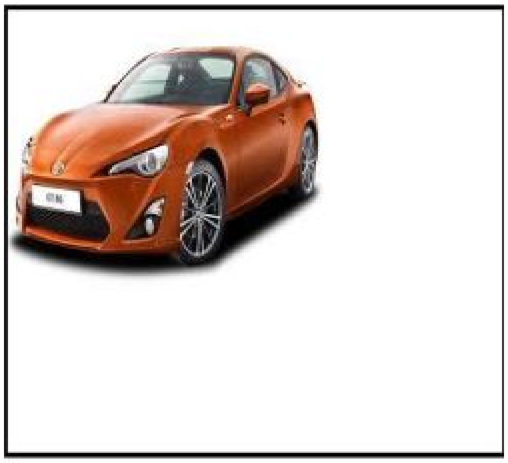
(Average accuracy around **99.2%**)

I have tried by keeping 3 nodes in hidden instead of 4, I got 82.2,64.7,75.4,85.7,84.6 for seeds 1,2,3,5,10.(78.5 is average). I have even tried relu instead of softmax in hidden layer and got 97.2,97.7,63,83,83.3,98.2 for seeds 1,2,3,4,5,10. So for robustness, I have maintained softmax with 4 hidden nodes.

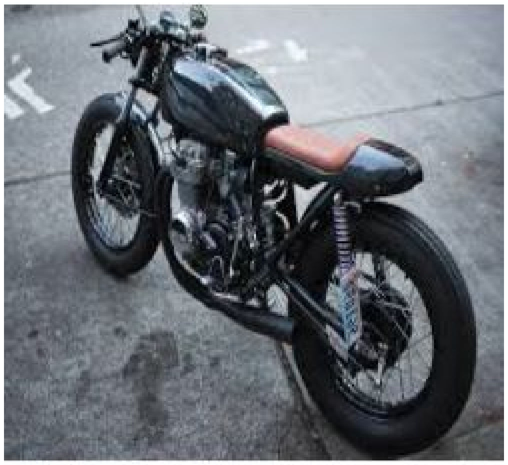
<u>Explanation for topology</u> : We kept 4 hidden nodes to depict each of 4 quadrants in the space.The 4 nodes can be thought of as x1+x2, x1-x2, .. .So, for example, if point in Q3, then -x2-x1 activates exponentially(since softmax), and remaining nodes are very less. And the output is such that first node is activated by nodes of Q1,3 and second by Q2,4. So for Q3 point first node of final output activates and gives +ve as output.

2. THEORY PROBLEM

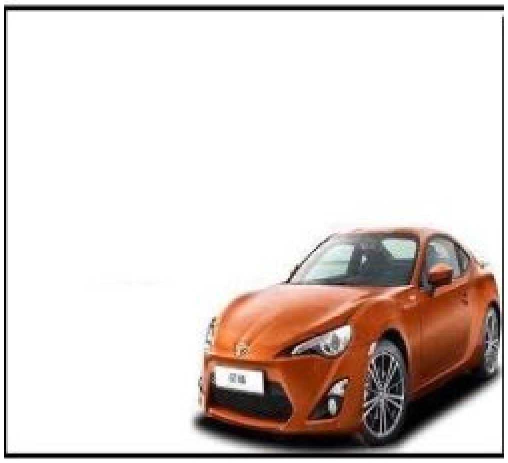
1. **Task 1** : Image classification



First detects the small edges, and then small shapes and then complex features in deeper layers like wheels, mirror, body of the car,..



Similar to car but will have more activations in the intermediate layers, i.e; different and more features detected like handle, engine,...So, if these are detected, it may conclude it more as a bike than a car



Similar to pic1 but the positions of the high activations in the middle layers will be different but max-pooling concentrates them into a smaller matrix and helps in detecting the same features as pic1

The components/operators extracted in the intermediate layers to distinguish between different vehicles : We know CNN is a layered network, we can see that initial layers detect some simple features like horizontal,vertical edges and then next layer would detect curves in the image form these edges and then simple shapes(like circular, …) and then it goes to more complex features(like handle, wheels, body,..) and then we flatten the obtained matrix so that all the major features/shapes obtained now get into a single array and then we may apply some more fully connected layers to filter out less relevant features and then at last we can apply a classification function like softmax to detect which object it is.
So, for example, if we think about car and bike, we see that car has more area compared to bike but bike has more visible features(wheels, handle,engine,...). So, if it finds more activated regions in the intermediate layers, so more features detected, and so can classify as bike.

CNN handles differences in object's locations with the help of pooling(max or avg) done immediately after a convolution. What it does is it makes the major features more visible for the further layers by decreasing the dimensions, i.e; removing minute/tiny activations obtained. So even in case of translation (even if translated version is not present in train set) due to max pooling, the features are easily detected by the further layers.

CNN also handles differences in size. What we can do is have some parallel convolutional layers each with a different filter size applied on the input image and after some layers in each convolution network, the features get detected so, we can merge them into a single 1-Dimensional layer and then give the output. For example, filter1 is smaller than filter2 in size and we have two car images with different car sizes(carimg1<carimg2), then features in carimg1 get detected in filter1 but may not be good enough by filter2 where as it is vice versa for carimg2 If we have a single filter size for input image, then we may not classify carimg1,2 correctly.
Note : We can also find filter1 from filter2 by down/upsampling it as both detect the same features, so they have similar values.
Also for **size invariance**, CNN has to be trained on samples having different sizes of same objects.

2. **Task2** :(sliding window / grids) Image detection

So, now we want to detect well separated objects/vehicles in an image. What we can do is have a window and slide it along the image or divide into grid and take each grid cell and apply pre-trained CNN to it to know whether an object is present in the window or not.**Since, well separated objects, so most probably, only one complete object can be present in a window**(a reasonable assumption). Thus, we can use the one **trained in task1** where only one object is classified. In this way we are classifying the object in each window and also

position of the object will be the position of the window.
Since, a single sized window may not fit the object, so we can have multiple fixed sized grids/ sliding window and send each to the network. We also **need to have a thresholding operation** after obtaining the probabilities through softmax, i.e; confidence that a grid cell will completely have the object (because for example, we may just have half of the car in the grid cell and so our softmax gives max probability for car as many features of the car are present but it will not be large enough)
Note : Since input for task1 CNN **needs to be size of original image**, so we can add a background(white) to our grid before sending it to CNN. This will not harm our model because our CNN in task1 doesn't consider any relation between the surroundings and the object.
Limitations : One obvious limitation is that it is very expensive in terms of computation, as now we may be checking for 1000s of grids for a single image. It is also time consuming. Also we are not giving the accurate boundaries of the object, we are just giving the position of grid or window, i.e; box in which the object is present.

3. **Task3** : (Overlapping objects detection)
   Since we will have overlapping objects, first we can take a dataset which is a mixture of single,complete object images and single,cropped object images and then train a CNN similar to task1 but keeping last layer as sigmoid so that each class probability will be deduced independently(since we may have **multiple objects in a single window**). Now, we can do something similar to task2, i.e; sliding window and detecting whether a vehicle is present or not. Remember that we have to decrease the threshold for probabilities obtained after sigmoid as here we have cropped objects, so there may not be high activations as compared to before, i.e; the classifier will be weaker, so threshold should be less.
   Limitations : It will have all the limitations of task2 since we are using sliding window but along with these, one major thing is that this is a **very weak classifier**, the threshold for an object to be detected needs to be very less as objects may not be complete and also since it is trained on partial objects so it may detect multiple windows of a single object in case of complete objects, thus making it more **error prone**.