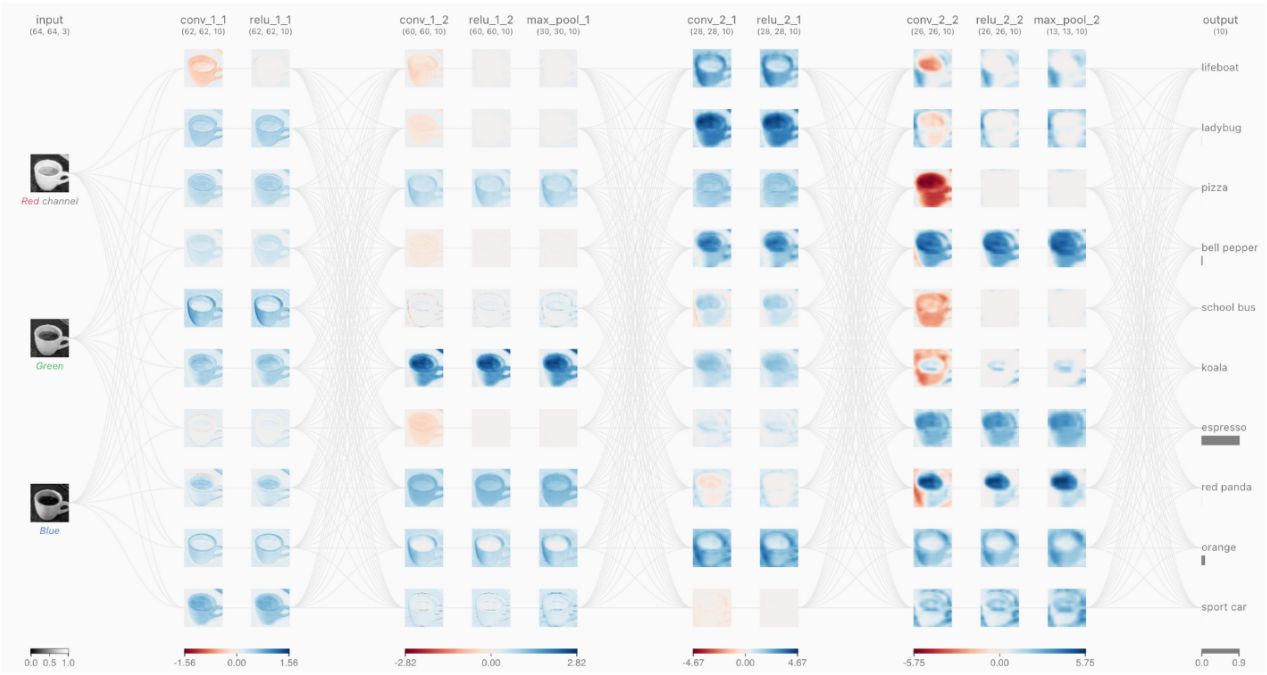
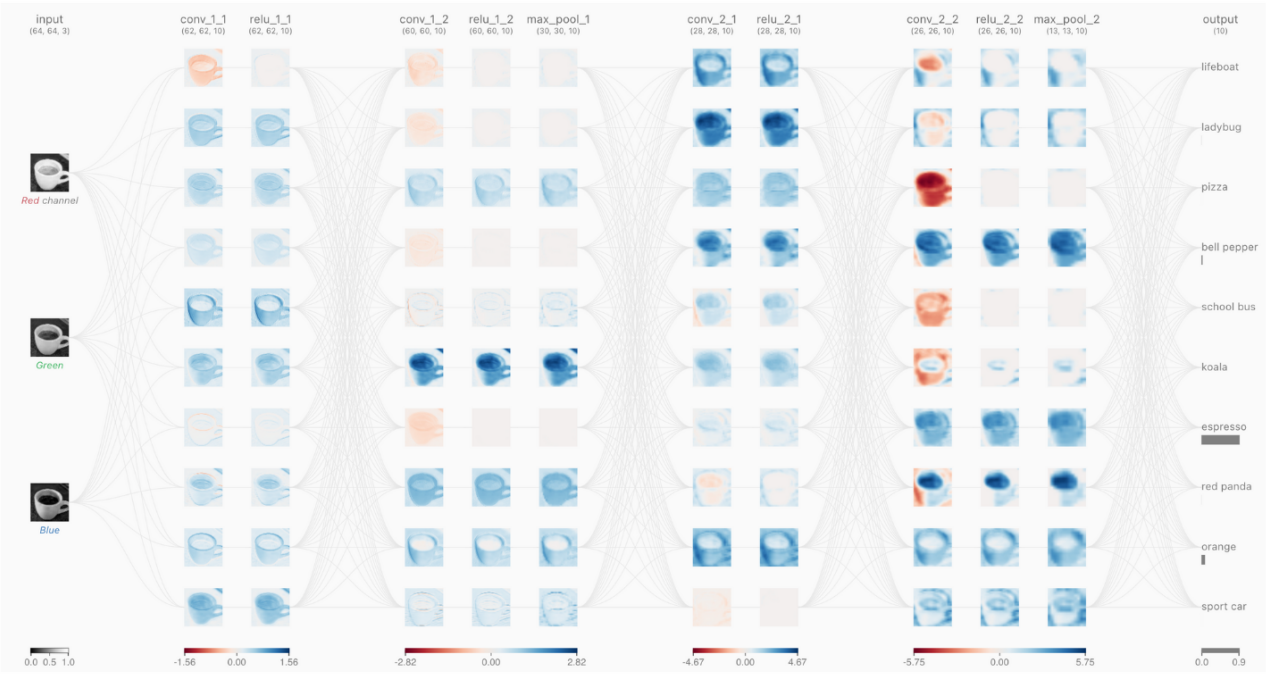
SAIDL Core ML

Core ML : Variations of Softmax

First, all the required libraries are imported. The model is built on PyTorch. Then the datasets are downloaded using the datasets.CIFAR100 model. I have used the TinyVGG as the CNN Architecture for Image Classification.

I have seen the architecture from CNN Explainer and got to know that it worked pretty impressively for image classification. So I modelled this architecture.

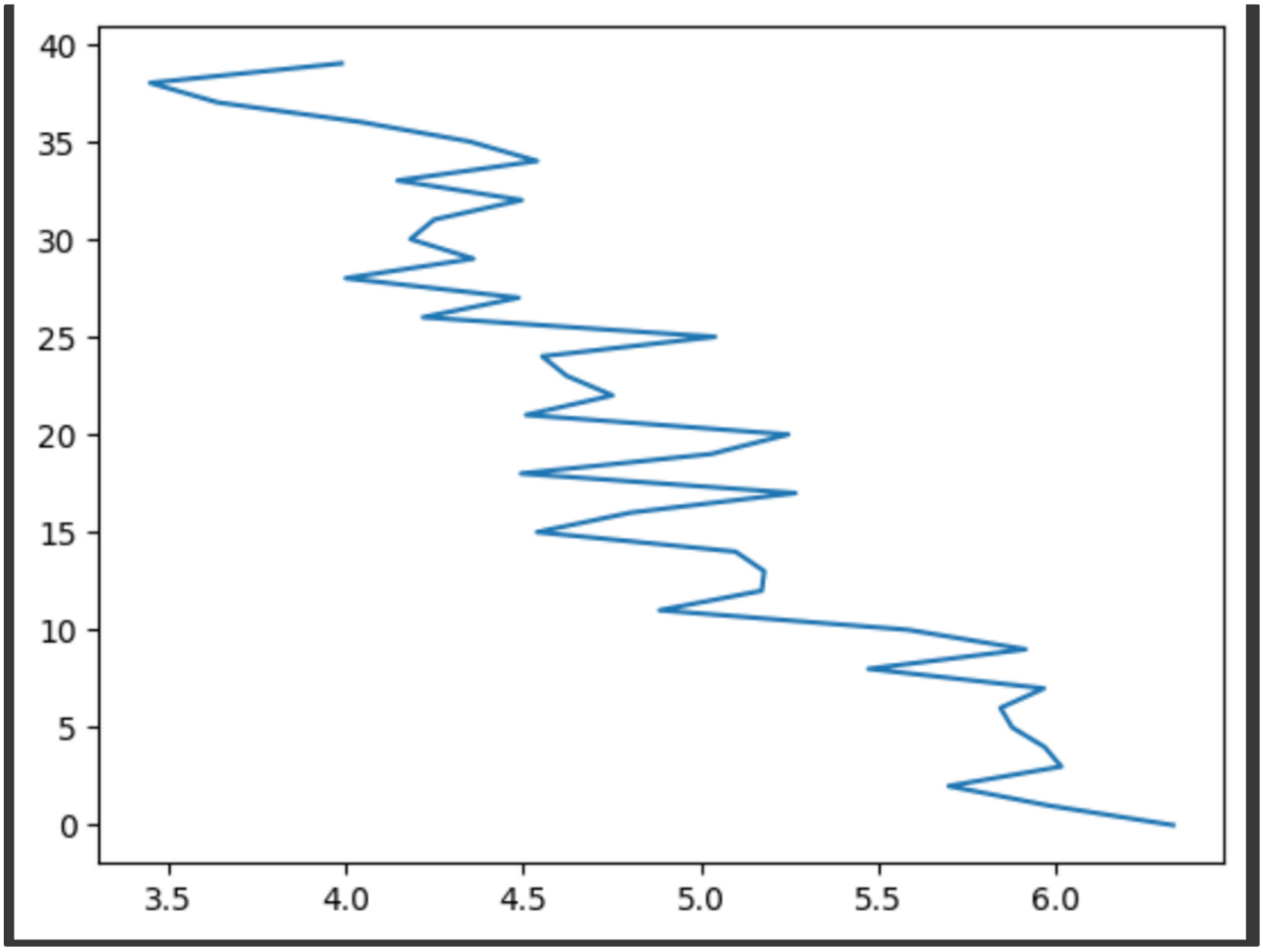




The first blocks two convolutions layers followed by ReLu activations to induce non-linearity in the output and then at last we have the max pool filter. The second block is a replica of the first block. The third block consists of a fully connected layer. I have used the CrossEntropy Loss for image

classification as it was the go to loss function and used the Stochastic Gradient Descent Algorithm as it is the best optimiser while doing batch gradient descent. Then there is the train step where it takes the X , y in batches with a batch size of 32 as it was one of the commonly used batch size. Then there is the test step which is executed after every epoch, so that the accuracy is calculated and we get to see the working of the model. It is actually executed after every epoch so that we can stop when there is not enough improvement in the accuracy of the model. But here I specifically decided the epochs to be 10 so that I can compare the accuracy and other evaluation metrics with other activation functions.The Softmax Function took on average 61.13s per epoch and produced accuracy of 23% on test dataset. The time complexity of the function is the order of n, as the size of the input the number of computations is also increased in the same proportion.

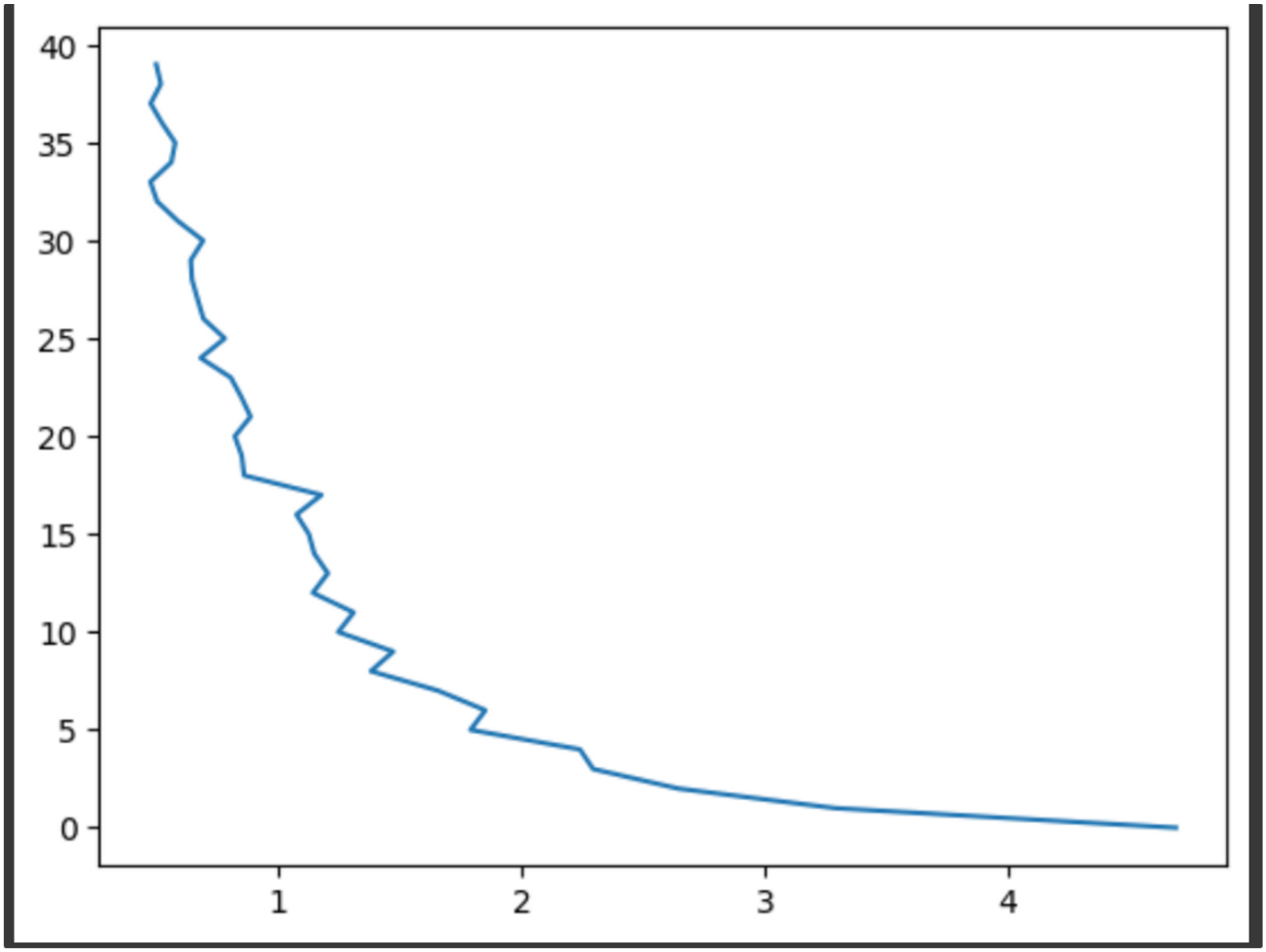
Then for Gumbel Softmax I followed the same architecture as of Softmax Function. Then I modelled the activation function by just adding a disturbance term to the logits and performing softmax function. The disturbance is in general taken from normal distribution which is done by using the torch.randn function .I used the negative log likelihood function as the loss function here, as I was directly getting the probabilities of each class using the Gumbel softmax activation while if I use the cross entropy loss it has an inbuilt log softmax function in it to compute the loss. The time complexity of this function is also order of n as it also performs an inbuilt softmax function. This function took at an average 58.39 sec per epoch and had a test accuracy of 16.7% on the test dataset. This how loss was falling after every 500 batches.



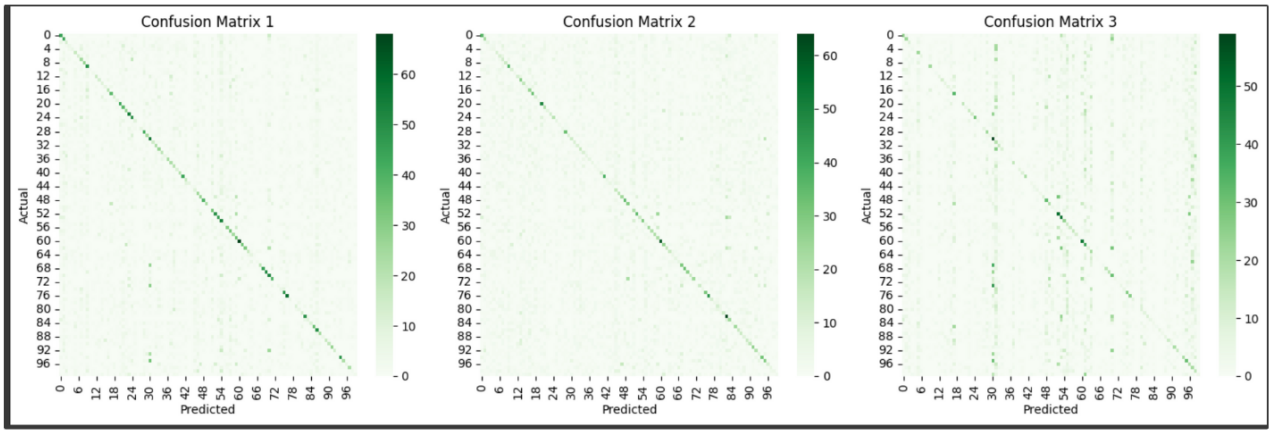
Then at last for Hierarchical Softmax. I read a lot of research papers and saw a lot of videos for

modelling this function. First I used the same CNN Architecture, which outputs 100 logits for each class. Then I made a binary tree which has 128 leaf nodes as it was closest perfect binary tree that can output 100 categories. Then I made a neural network layer for each node in the binary tree. These hidden layers each have 1 node and 100 parameters. So when X and y are passed as inputs, first the X is passed through the CNN Architecture to output 100 logits. After this the path to be followed from the parent node to the output is found out and then the probability of the class y is found out by multiplying all the probabilities of the branches in the path. It is as , if suppose the output is 19, then the leaf node corresponding to this number is found it by adding 127 to y. Then to find the parent node , it is found by subtracting 1 from leaf node number and dividing it by 2. We have to subtract 1 as the indexing is starting from 0. And the probability to this branch is found out by applying sigmoid on the dot product on the logits and the parameters of the hidden layer, which is basically connecting a layer with 100 nodes to a layer with 1 node. The depending on if the child node was odd number or even number sigmoid is performed on the product or the negative of the product, so that both the branches have the probabilities assigned as p and 1-p. Then multiplication is performed of all the probabilities of the branches transversed by the model. So here as we are passing the model only through the path required by the output, the time complexity out here is order of log n.

This is how the loss fell after every 500 batches.



Then I compared the all the models using the coefficient matrix, F1 score, performance, recall score and precision.




```
Metric       Model 1    Model 2    Model 3
---------  ---------  ---------  ---------
Accuracy     0.2314     0.1671     0.1183
Precision    0.237453   0.173658   0.147382
Recall       0.2314     0.1671     0.1183
F1 score     0.212231   0.163595   0.102985
/usr/local/lib/python3.9/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predict
  _warn_prf(average, modifier, msg_start, len(result))
```