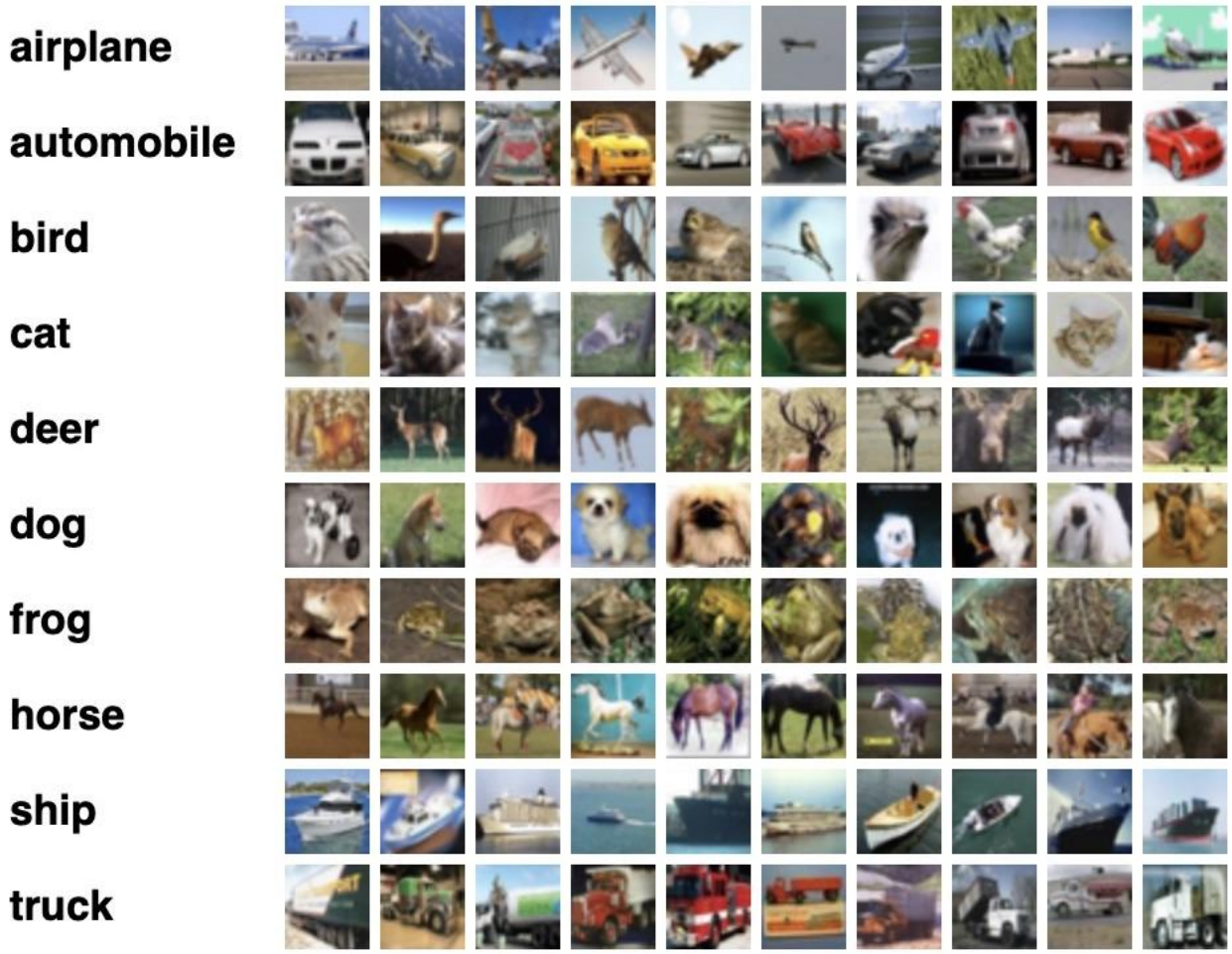
IMAGE CLASSIFICATION FOR CIFAR100 DATA-SET



Introduction

Core ML: Variations of Softmax

The softmax function is a commonly used activation function in neural networks, particularly for classification tasks. It takes in a vector of K real numbers and outputs a probability distribution of K possible outcomes. This function is useful because it allows us to map probabilistic distributions using neural networks. However, the softmax function can become computationally expensive when the number of classes, denoted as C, is large. This is because the denominator of the softmax function involves a summation over all C classes, taking O(C) time. This can be problematic as the number of classes increases, as it can significantly slow down the training and evaluation of the model. Overall, this experiment will provide insights into the trade-offs between different softmax implementations in neural networks, particularly in the context of classification tasks with many classes. By understanding the impact of softmax implementations on model performance and training time, we can make more informed decisions when designing and training neural networks for classification tasks. This is useful for many tasks and is used extensively in classification tasks. Though the function looks simple, it can get computationally expensive. This is because the summation of the denominator takes O(C) time, where C is the number of classes. This is a non-issue when the number of classes is small but can cause problems as the number increases. Report the results/analysis, observations and inferences in a technical report in addition to the code.

You are tasked with the following:

1. Develop a convolutional neural network (CNN) model on the CIFAR 100 dataset for image classification with the standard softmax.

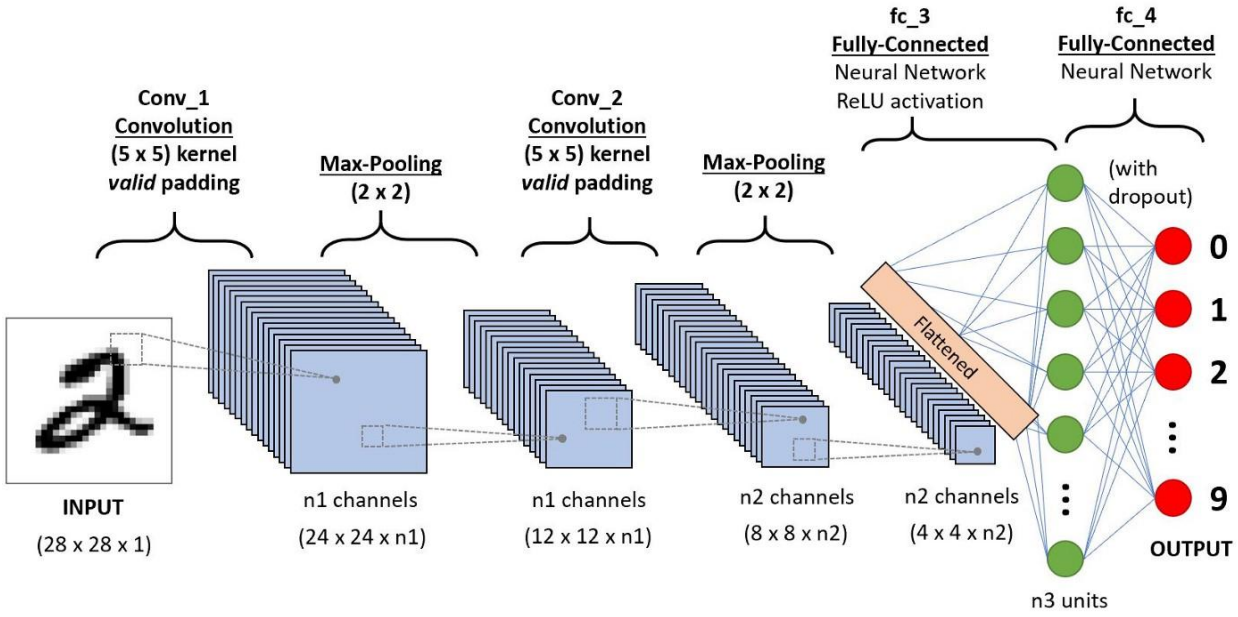
2. Make a second model with the same architecture but different softmax implementations that reduce the computational complexity of the softmax function (eg: hierarchical. Gumbel-Softmax which is an alternative to the standard softmax that uses a Gumbel distribution to simulate the process of sampling from a categorical distribution, Adaptive Softmax). Report the time complexity for each of the softmax functions you are using.

3. Evaluate the performance of the models using a range of evaluation metrics, including accuracy, precision, recall, F1 score, and confusion matrix.

4. Compare the performance and epoch time between the methods.

BONUS: Develop a transformer-based architecture that integrates these variations of softmax function for image classification using the CIFAR-100 dataset and evaluate the performance across the above-mentioned metrics and also compare the results with the above question using a CNN-based architecture.

CONVOLUTION NEURAL NETWORK ARCHITECTURE



a. Using Standard Softmax function

Parameters:
Learning rate : 0.01
Momentum: 0.9
Batch Size: 64
Epochs : 50

Input Data:
Train : X=(35000, 32, 32, 3), Y=(35000, 1)
Validation : X=(15000, 32, 32, 3), Y=(15000, 1)
Test : X=(10000, 32, 32, 3), Y=(10000, 1)

Results :
Epoch 1/50
547/547 [==============================] - 32s 56ms/step - loss: 4.5660 - accuracy: 0.0325 - val_loss: 4.2207 - val_accuracy: 0.0539
Epoch 2/50
547/547 [==============================] - 27s 49ms/step - loss: 4.3205 - accuracy: 0.0488 - val_loss: 4.0878 - val_accuracy: 0.0688
Epoch 3/50

547/547 [==============================] - 27s 49ms/step - loss: 4.2093 - accuracy: 0.0605 - val_loss: 3.9300 - val_accuracy: 0.1011
.
.

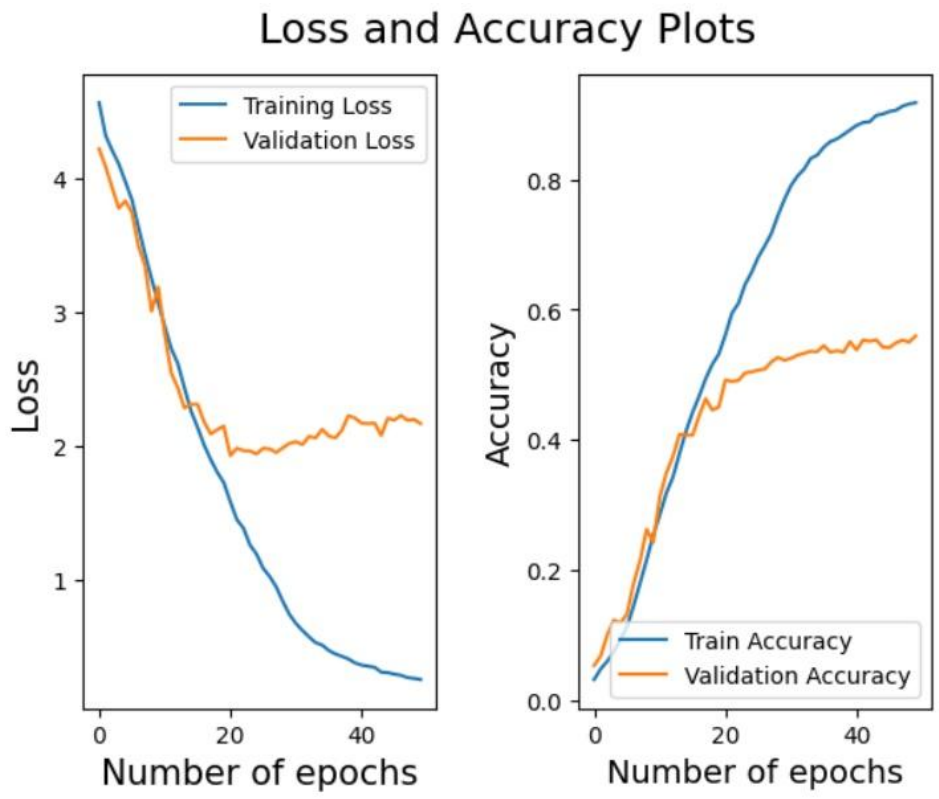
Epoch 49/50
547/547 [==============================] - 27s 49ms/step - loss: 0.2679 - accuracy: 0.9156 - val_loss: 2.2014 - val_accuracy: 0.5503
Epoch 50/50
547/547 [==============================] - 27s 49ms/step - loss: 0.2595 - accuracy: 0.9175 - val_loss: 2.1704 - val_accuracy: 0.5596

313/313 [==============================] - 2s 8ms/step - loss: 2.1162 - accuracy: 0.5709

Accuracy for test data is >**57.090%**
Accuracy score: 0.570900
Precision score: 0.568185
Recall score: 0.570900
F1 score : 0.564955
Cohens Kappa score 0.566566
Confusion Matrix :
[[77 0 1 ... 0 0 0]
 [0 69 0 ... 0 0 0]
 [3 1 36 ... 0 3 0]
 ...
 [0 1 0 ... 60 0 0]
 [0 1 7 ... 0 17 1]
 [0 1 0 ... 0 0 60]]

Execution:
The code completed execution in 24 minutes using 1 GPU resource on Google Colab

Plots:



b. Using Gumbel-Softmax function

Parameters:
Learning rate : 0.01
Momentum: 0.9
Batch Size: 64
Epochs : 50

Input Data:
Train : X=(35000, 32, 32, 3), Y=(35000, 1)
Validation : X=(15000, 32, 32, 3), Y=(15000, 1)
Test : X=(10000, 32, 32, 3), Y=(10000, 1)

Results :
Epoch 1/50
547/547 [==============================] - 41s 54ms/step - loss: 4.1163 - accuracy: 0.0760 - val_loss: 3.7127 - val_accuracy: 0.1409
Epoch 2/50
547/547 [==============================] - 27s 50ms/step - loss: 3.4879 - accuracy: 0.1681 - val_loss: 3.2912 - val_accuracy: 0.2028

Epoch 3/50
547/547 [==============================] - 27s 49ms/step - loss: 3.0863 - accuracy: 0.2396 - val_loss: 2.8866 - val_accuracy: 0.2731

.
.

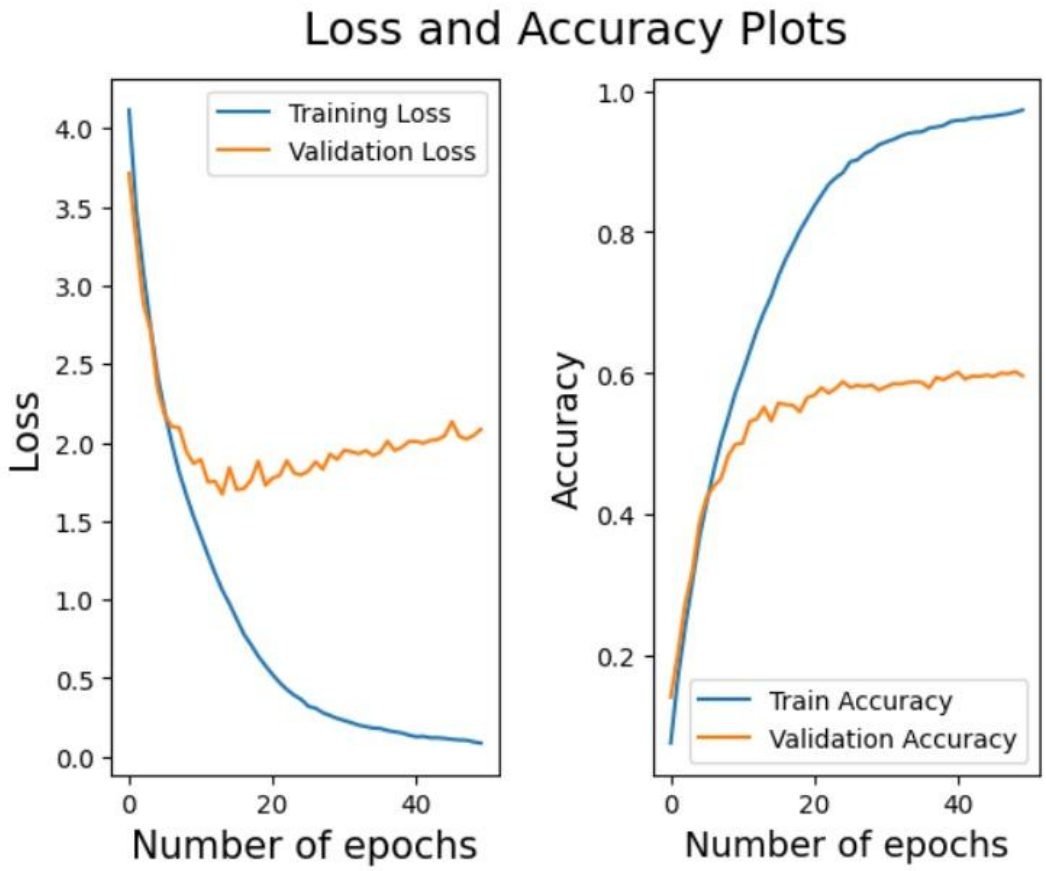
Epoch 49/50
547/547 [==============================] - 27s 49ms/step - loss: 0.0944 - accuracy: 0.9699 - val_loss: 2.0445 - val_accuracy: 0.6023
Epoch 50/50
547/547 [==============================] - 27s 49ms/step - loss: 0.0868 - accuracy: 0.9730 - val_loss: 2.0831 - val_accuracy: 0.5965

313/313 [==============================] - 3s 8ms/step - loss: 2.0722 - accuracy: 0.5950

Accuracy for test data is >**59.500%**
Accuracy score: 0.593100
Precision score: 0.597079
Recall score: 0.593100
F1 score : 0.587648
Cohens Kappa score 0.588990
Confusion Matrix :
[[75 1 1 ... 0 0 0]
 [0 71 0 ... 0 0 0]
 [3 2 39 ... 0 6 0]
 ...
 [0 0 0 ... 61 0 0]
 [0 0 5 ... 0 28 1]
 [0 1 0 ... 0 0 61]]

Execution:
The code completed execution in 24 minutes using 1 GPU resource on Google Colab

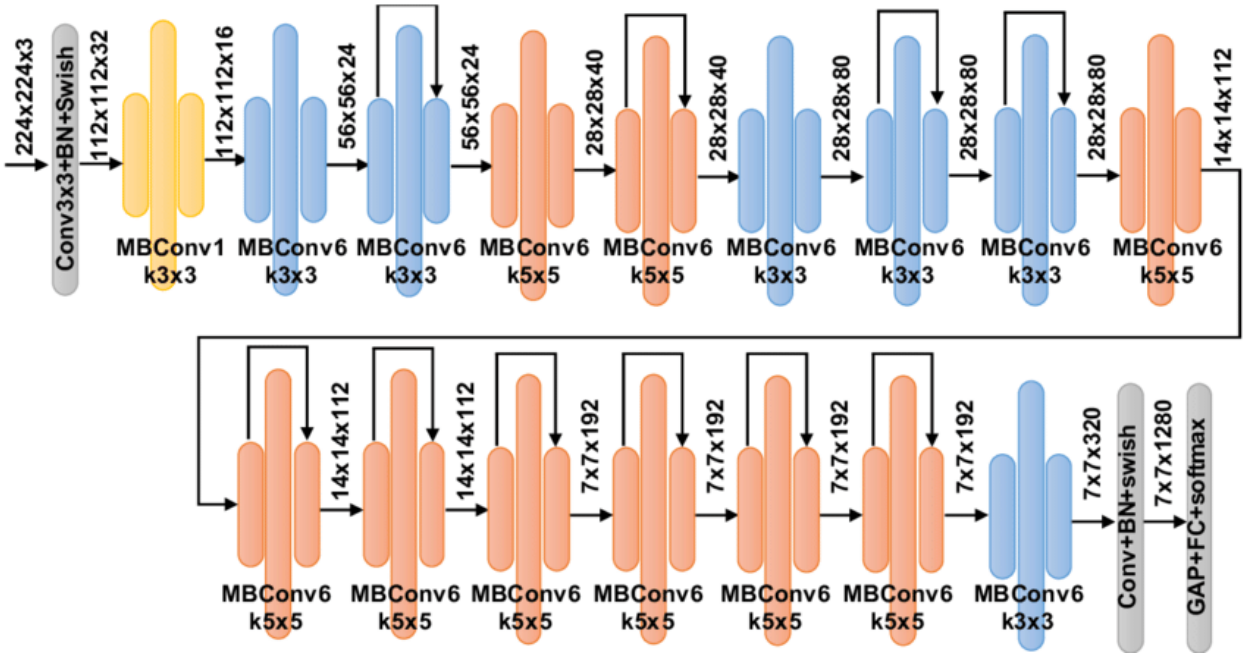
Plots:



EFFICIENTNET ARCHITECTURE

Parameters:
Learning rate : 0.01
Momentum: 0.9
Batch Size: 100
Epochs : 10

Input Data:
Train : X=(35000, 32, 32, 3), Y=(35000, 1)
Validation : X=(15000, 32, 32, 3), Y=(15000, 1)
Test : X=(10000, 32, 32, 3), Y=(10000, 1)



Results :

Epoch 1/10
350/350 [==============================] - 344s 919ms/step - loss: 1.6690 - accuracy: 0.6036 - val_loss: 1.1545 - val_accuracy: 0.7235
Epoch 2/10
350/350 [==============================] - 320s 915ms/step - loss: 0.2549 - accuracy: 0.9217 - val_loss: 1.1098 - val_accuracy: 0.7425
Epoch 3/10
350/350 [==============================] - 306s 874ms/step - loss: 0.0692 - accuracy: 0.9788 - val_loss: 1.0732 - val_accuracy: 0.7672
.
.

Epoch 9/10
350/350 [==============================] - 306s 874ms/step - loss: 0.0069 - accuracy: 0.9986 - val_loss: 1.0677 - val_accuracy: 0.7890
Epoch 10/10
350/350 [==============================] - 306s 873ms/step - loss: 0.0062 - accuracy: 0.9987 - val_loss: 1.0616 - val_accuracy: 0.7929

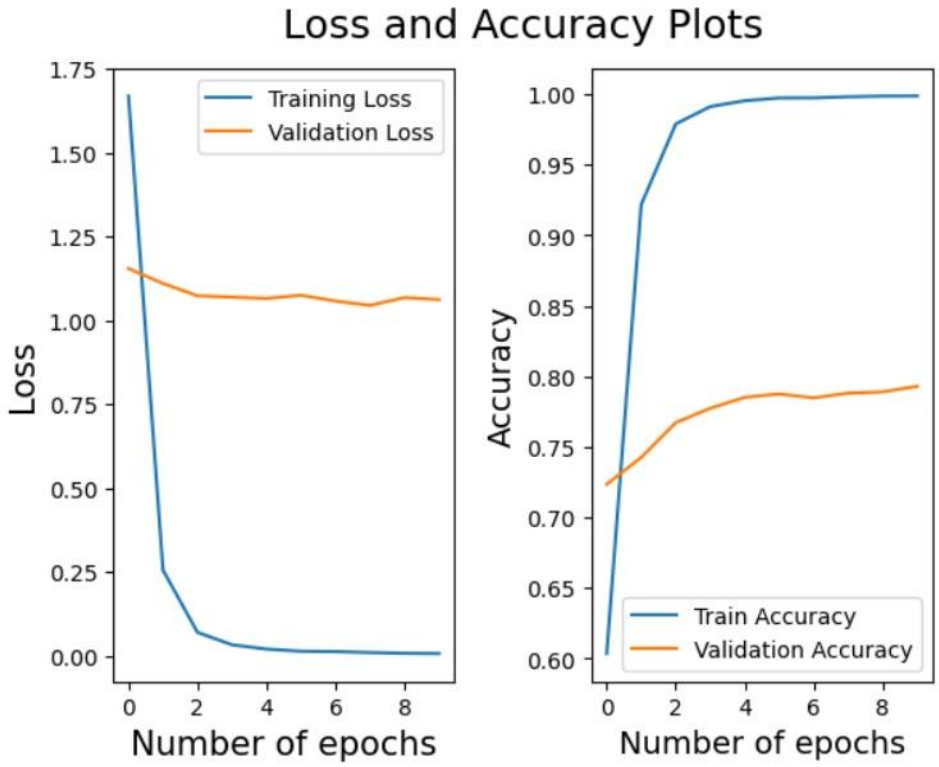
313/313 [==============================] - 25s 68ms/step - loss: 1.0434 -

accuracy: 0.7907

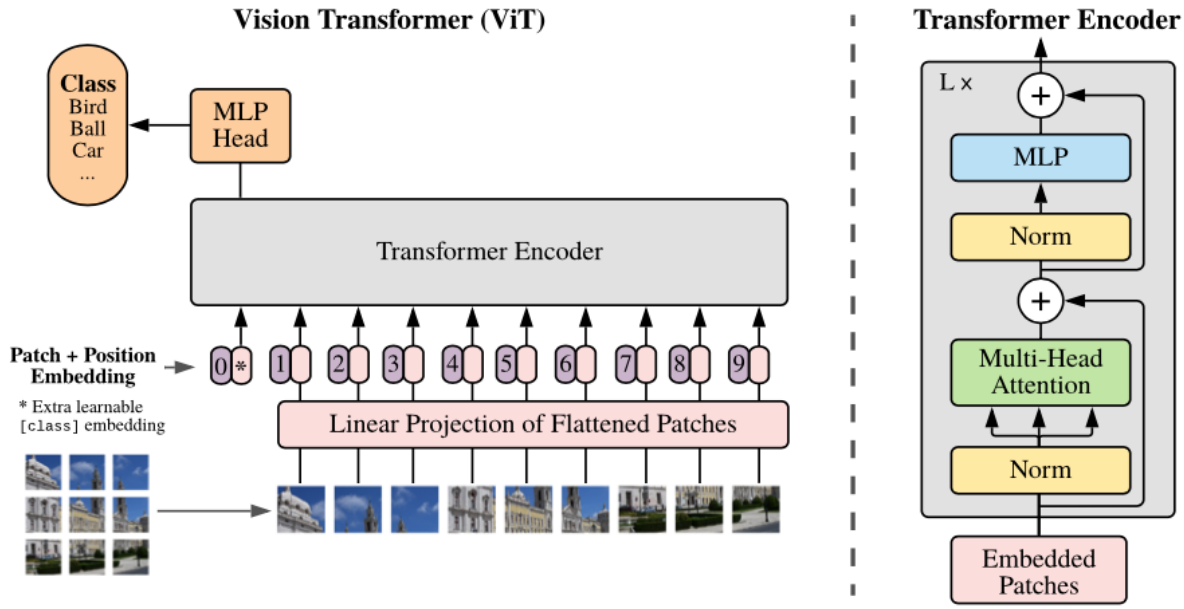
Accuracy for test data is >**79.070%**
Accuracy score: 0.790700
Precision score: 0.790797
Recall score: 0.790700
F1 score : 0.789501
Cohens Kappa score: 0.788586
Confusion Matrix :
[[89 0 0 ... 0 0 0]
[0 92 0 ... 0 0 0]
[0 0 65 ... 0 4 0]
...
[0 0 0 ... 79 0 0]
[0 0 5 ... 0 64 0]
[0 0 0 ... 0 0 86]]

Execution:
The code completed execution in 54 minutes using 1 GPU resource on Google Colab

Plots:



VISION TRANSFORMER ARCHITECTURE



Parameters:
Learning rate : 0.001
Momentum: 0.9
Batch Size: 32
Epochs : 5

Input Data:
Train : X=(35000, 32, 32, 3), Y=(35000, 1)
Validation : X=(15000, 32, 32, 3), Y=(15000, 1)
Test : X=(10000, 32, 32, 3), Y=(10000, 1)

Results :
Epoch 1/5
1094/1094 [==============================] - 2226s 2s/step - loss: 0.4945 - accuracy: 0.8564 - val_loss: 0.4139 - val_accuracy: 0.8797
Epoch 2/5
1094/1094 [==============================] - 2199s 2s/step - loss: 0.3154 - accuracy: 0.9063 - val_loss: 0.3665 - val_accuracy: 0.8949
Epoch 3/5
1094/1094 [==============================] - 2199s 2s/step - loss: 0.2368 - accuracy: 0.9304 - val_loss: 0.3444 - val_accuracy: 0.9020

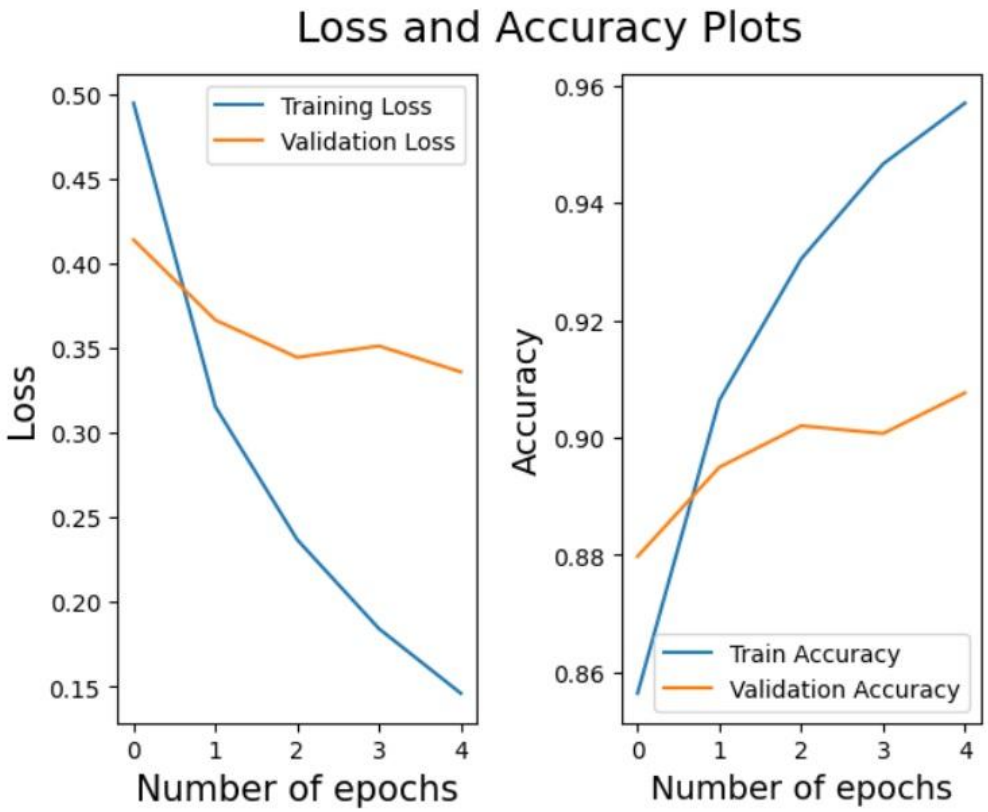
Epoch 4/5
1094/1094 [==============================] - 2199s 2s/step - loss: 0.1843 - accuracy: 0.9466 - val_loss: 0.3512 - val_accuracy: 0.9007
Epoch 5/5
1094/1094 [==============================] - 2199s 2s/step - loss: 0.1461 - accuracy: 0.9570 - val_loss: 0.3359 - val_accuracy: 0.9076

313/313 [==============================] - 180s 576ms/step - loss: 0.3328 - accuracy: 0.9088

Accuracy for test data is >**90.880%**
Accuracy score: 0.908800
Precision score: 0.910000
Recall score: 0.908800
F1 score : 0.908548
Cohens Kappa score: 0.907879
Confusion Matrix :
[[97 0 0 ... 0 0 0]
[0 96 0 ... 0 0 0]
[0 0 78 ... 0 0 0]
...
[0 0 0 ... 95 0 0]
[0 0 1 ... 0 86 0]
[0 0 0 ... 0 0 94]]

Execution:
The code completed execution in 3 hours 45 minutes using 1 GPU resource on Google Colab

Plots:


Loss and Accuracy Plots
Training Loss
Validation Loss
Loss
Number of epochs
Train Accuracy
Validation Accuracy
Accuracy
Number of epochs

CONCLUSION

Model	Accuracy of model	Accuracy w.r.t. Test data
CNN with standard softmax	**91.75%**	**57.09%**
CNN with Gumbel softmax	**97.30%**	**59.50%**
EfficientNet	**99.97%**	**79.07%**
Vision Transformer	**95.70%**	**90.88%**

As can be seen clearly, for the task of image classification for the CIFAR100 dataset, Vision Transformer Architecture outscores both the CNN and the EfficientNet Architectures by a wide margin, although it does come at the cost of taking a huge computing time on Google Colab GPU.