Machine Perception Project I
Multi-class classification with CNNs and ViTs

Ojas Taskar
MSE Robotics
Johns Hopkins University
otaskar1@jh.edu

Vaibhav Mahapatra
MSE Robotics
Johns Hopkins University
vmahapa1@jh.edu

Abstract

In this project, we aim to build a deep learning-based classifier for multiclass classification on the CIFAR-10 Dataset. The CIFAR-10 dataset consists of 60000 32×32 color images of 10 classes, with 6000 images per class. We first build Deep Convolutional Neural Networks (CNNs) for this task. After experimenting with multiple architectures, we choose the best CNN classfier and subject it to an adversarial attack. With the same workflow, we train a Vision Transformers (ViTs) for the same task. This report documents our observations and results in detail. Our code implementation can be found in our GitHub repository.

Contents

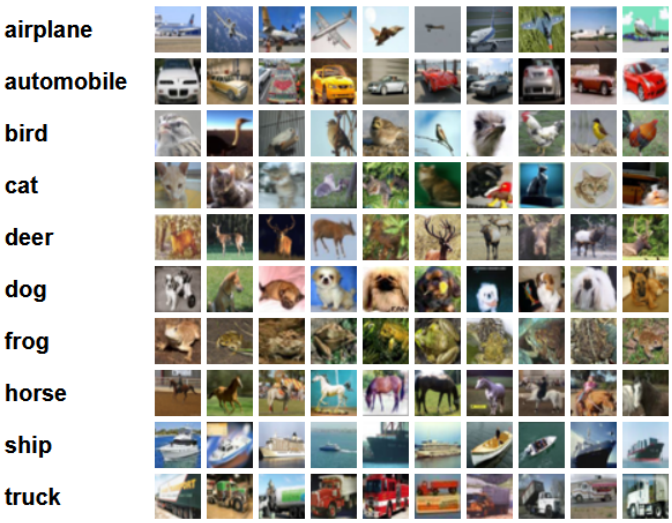

Figure 1. CIFAR10 Samples

I. The Dataset

The CIFAR-10 [1] Dataset dataset consists of 60000 32×32 color images of 10 classes, with 6000 images per class. Key points about it are:

- **Content**: The dataset contains 60,000 color images in 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck), representing common objects.
- **Resolution**: Each image is small, with a resolution of 32x32 pixels.
- **Split**: The data is divided into 50,000 training images and 10,000 test images.
- **Use Case**: CIFAR-10 is typically used for developing and benchmarking image classification models due to its manageable size and balanced class distribution (each class has 6,000 images).

A few samples of images from the dataset can be seen in Fig. 1. The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

II. Evaluation Metrics

1) CrossEntropyLoss: For a given input image x and target label y, the multiclass cross-entropy loss can be written as:

$$\mathcal{L} = -\sum_{i=1}^{C} y_i \log(\hat{p}_i)$$

Here, C is the total number of classes, y_i is the binary indicator (0 or 1) if class i is the correct classification for the sample, $\hat{p}_i$ is the predicted probability for class i, usually obtained through a softmax layer.

2) Accuracy: the fraction of correctly classified samples across N samples.

$$\text{Accuracy} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{1}(\hat{y}_n = y_n)$$

3) F1 score (macro): calculated by first computing the F1 score independently for each class, then averaging these scores.

$$\text{Macro F1} = \frac{1}{C}\sum_{i=1}^{C} \text{F1}_i$$

$$\text{F1}_i = \frac{2 \cdot \text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}$$

III. Convolutional Neural Network

In this section, we fit multiple Convolutional Neural Networks (CNNs) to the CIFAR10 data. Our experiments and conclusions are detailed in each of the following subsections:

- Building our own CNN
- Transfer Learning - Pre-trained SOTA Models

A. Building our own CNNs

In this section, I built 3 CNN architectures from scratch. In each architecture, I used a combination of methods, such as batch normalization, dropout, denser feature maps, varied activation functions etc, to experiment with the performance of the model. The models were evaluated on three learning rates with the following fixed parameters:

- Batch size: 32
- Optimizer: Adam
- Train and Validation data: 80-20 split
- Epochs: 10

In each of the architectures, I tuned hyperparameters for three separate learning rates - 0.001, 0.0005, 0.00005. The training loss, validation loss, test accuracy and F1-score have been detailed in later sections.

1) Net 1 - LeNet5: This architecture is a modified version of LeNet5, with batch norm added for smooth training. There are 2 convolutional and 3 fully connected layers. The activation function used is ReLU. Generally, we observe the most stable performance from this model out of the three trained in total.

2) Net 2: This architecture is an extended version of the first architecture. Here, we add another convolutional layer for denser feature maps. Also, I have introduced two dropout layers, one during the convolutional stage and one during the

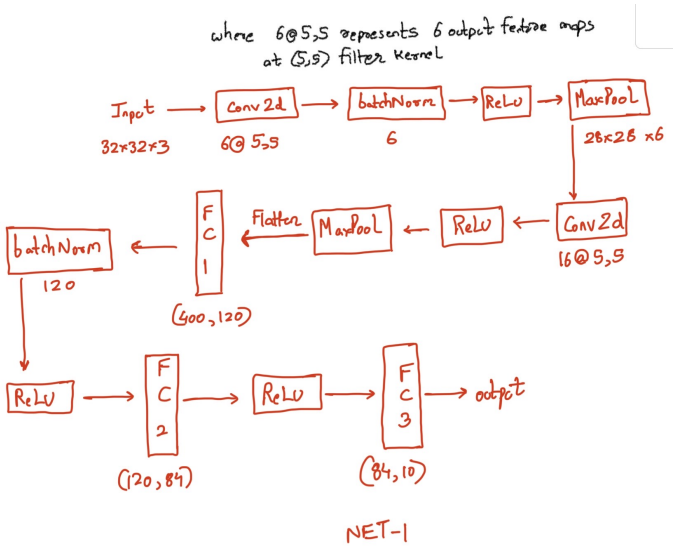

Figure 2. Net-1

fully connected stage. Additionally, I experimented with a sigmoid activation function in the second last fully connected layer. While this does give slightly smoother plots of training and validation loss, there is no significant improvement in accuracy.

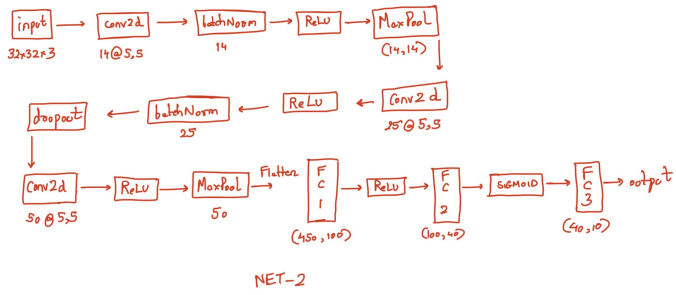

Figure 3. Net-2

3) Net 3: This architecture combines batch norm, dropout and multiple convolutional layers. Here, we change the filter size from 5x5 in the previous models to 3x3. In order to keep track of feature map size, the 'padding' window in conv2d layer is set to 'same'. Only ReLU is used as activation. Batch Norm is performed in the second last fully connected layer. This architecture is the most accurate of the 3. Reducing filter size also leads to a decrease in training time as compared to Net 2.

Training times for all the models are described below, on average, for the three different learning rates. The duration of training is 10 epochs and the models are trained on a Google Colab environment with T4 GPU.

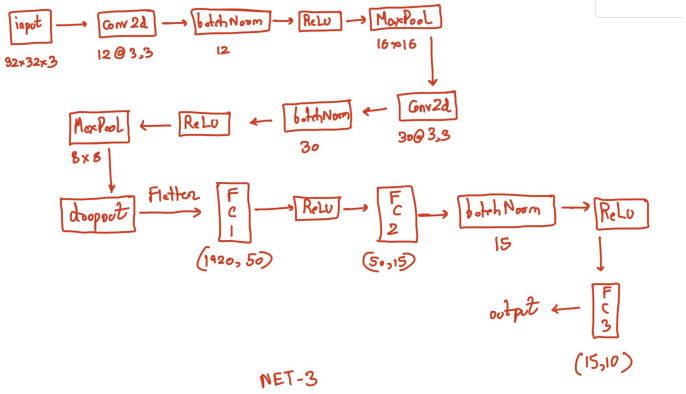

Figure 4. Net-3

- Net 1: **158.5** seconds
- Net 2: **288** seconds
- Net 3: **240** seconds

The largest training time for Net 2 can be attributed to the fact that it has the most layers, most convolutions and most batch normalization layers. Net 3 is slightly more complex than Net 1 and hence takes a bit more time to train.

Net 1

Learning Rate	Training Loss	Validation Loss	Test Accuracy (%)	Test F1 Score
0.001	**1.029**	**1.111**	**61.95**	**0.571**
0.0005	1.064	1.215	57.78	0.529
0.00005	1.389	1.435	49.53	0.45

Net 2

Learning Rate	Training Loss	Validation Loss	Test Accuracy (%)	Test F1 Score
0.001	1.349	1.395	51.66	0.466
0.0005	**1.34**	**1.384**	**50.67**	**0.452**
0.00005	1.671	1.705	41.17	0.353

Net 3

Learning Rate	Training Loss	Validation Loss	Test Accuracy (%)	Test F1 Score
0.001	**0.644**	**0.978**	**66.32**	**0.62**
0.0005	0.688	0.999	66.32	0.623
0.00005	1.006	1.164	61.56	0.57

Figure 5. Test Performance of all the models

B. Transfer Learning - Pre-trained SOTA Models

In this section, I imported pretrained state of the art (SOTA) CNN models from PyTorch and performed transfer learning with them to fit out CIFAR-10 dataset. I experimented with 3 architectures, namely Resnet50, VGG19 and Inceptionv3. The broad setup that I used for each set of experiments are:

- Batch size: 256
- Optimizer: Adam
- Train and Validation data: 80-20 split
- Epochs: 10

For each of the architecture, I essentially froze the majority of the pre-trained weights, and only trained additional custom

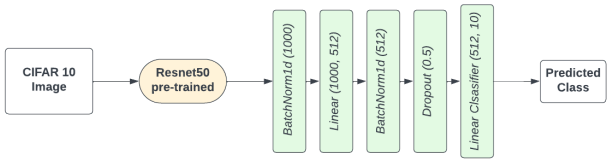

Figure 6. Transfer Learning with Resnet50

layers that I added. I hyperparameter tuned 4 learning rates: [0.0005, 0.0001, 0.00005, 0.00001]. Each architecture's performance along with training time are detailed in the upcoming sections.

1) Resnet50: ResNet-50 is a 50-layer deep convolutional neural network known for its *residual learning* technique, which uses skip connections to mitigate vanishing gradients and enable efficient training of very deep networks. It consists of an initial convolutional layer, followed by four stages of residual blocks, each with bottleneck layers (1x1, 3x3, and 1x1 convolutions) to reduce parameters. The final layers include global average pooling and a fully connected layer for classification.

For Transfer Learning, I added the following layers to the frozen pre-trained network. Fig. 6 diagrammatically showcases my Transfer Learning architecture. The orange cell denotes the frozen network whereas the green ones denote the trainable layers.

1) BatchNorm1d(1000)
2) Linear(1000, 512)
3) BatchNorm1d(512)
4) Dropout(0.5) - to prevent overfitting
5) Linear Classifier (512, 10) - final layer to classify into 10 classes

On the T4 x2 GPU, training time was 3 mins per epoch.

After training this architecture with 4 different learning rates, the training arcs of each model can be seen in Fig. 10. Table. I also details the performance metrics achieved after training all the models.

Learning Rate	Training Loss	Validation Loss	Test Accuracy (%)	Test F1 Score
0.0005	0.539	0.643	**78.0**	textbf0.841
0.0001	0.614	0.691	77.3	0.762
0.00005	0.705	0.724	76.1	0.750
0.00001	0.999	0.971	70.1	0.690

Table I
RESNET50 PERFORMANCE METRICS AFTER TRAINING FOR 10 EPOCHS

In conclusion, after comparing all the Resnet50 models, the one I trained with a learning rate of 5e-4 (0.0005) performs the best with 78% accuracy and a training time of 28 mins. Additionally, from the loss curve, there seems to be a little more potential to train the model further.

2) VGG19: VGG19 is a Deep Convolutional Neural Network that contains 19 layers with learnable parameters, organized into five convolutional blocks followed by three fully connected layers. Each block comprises multiple convolutional layers with small 3x3 filters, followed by max pooling layers that reduce spatial dimensions. VGG19 is known for its depth and simplicity, which allows it to capture complex features effectively. Though parameter-intensive, VGG19 achieved high accuracy in image classification tasks and was a top performer in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2014.

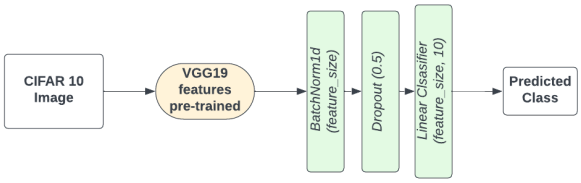

Figure 11. Transfer Learning with VGG19

For Transfer Learning with VGG19, I essentially froze the feature extractor and substituted the pretrained classifier with my own network. Assuming the flattened features have a size *feature_size*, the layers I added have the dimensions illustrated in Fig. 11:

- BatchNorm1d(feature_size)
- Dropout(0.5)
- Linear Classifier (feature_size, 10)

On the T4 x2 GPU, training time was longer than ResNet50 at 5 mins per epoch.

After training this architecture on 4 different learning rates, the training arcs of each model can be seen in Fig. 12. Table. II also details the performance metrics achieved after training all the models.

Learning Rate	Training Loss	Validation Loss	Test Accuracy (%)	Test F1 Score
0.0005	0.131	0.372	87.0	0.867
0.0001	0.157	0.357	87.0	0.867
0.00005	0.219	0.378	**87.0**	**0.867**
0.00001	0.462	0.489	84.4	0.841

Table II
VGG19 PERFORMANCE METRICS AFTER TRAINING FOR 10 EPOCHS

In conclusion, after comparing all the VGG19 models, the one I trained with a learning rate of 5e-5 (0.00005) performs the best with 87% accuracy and a training time of 48 mins. Additionally, from the loss curve in Fig. 12, there seems to

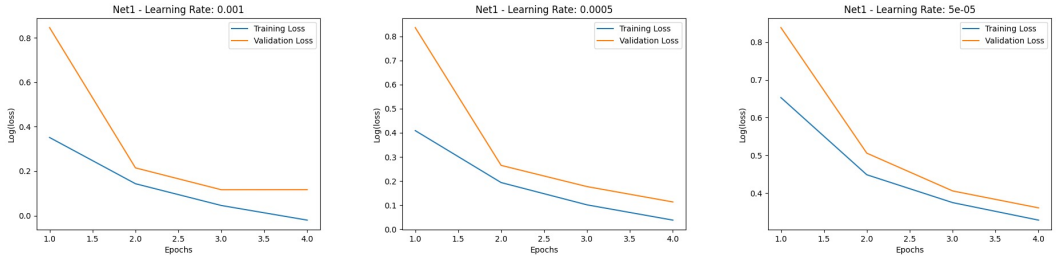

Figure 7. Training performance of Net1 over 10 epochs for lr = 0.001,0.005,0.00005

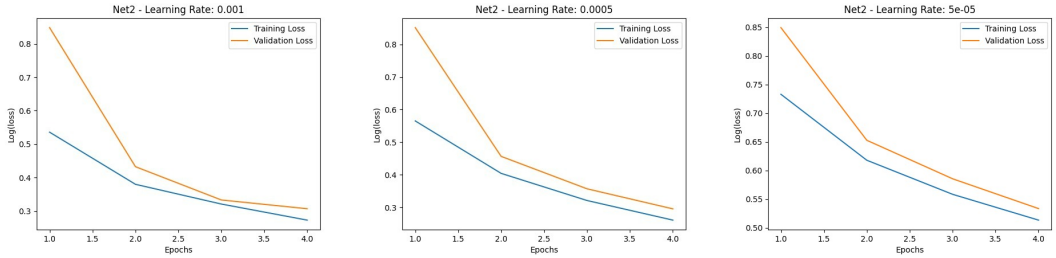

Figure 8. Training performance of Net 2 over 10 epochs for lr = 0.001,0.0005,0.00005

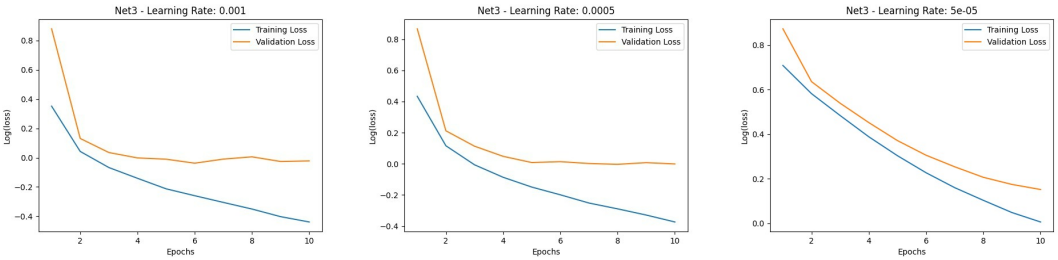

Figure 9. Training performance of Net3 over 10 epochs for lr = 0.001,0.0005.0.00005

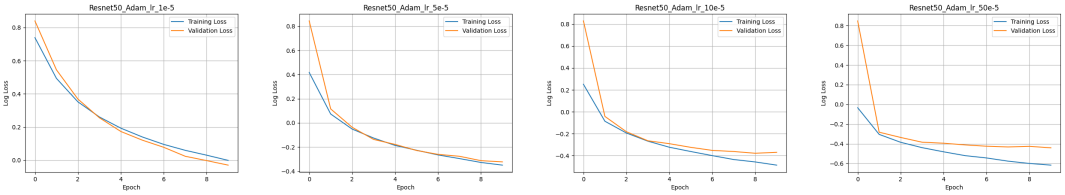

Figure 10. Training performance of Resnet50 over 10 epochs

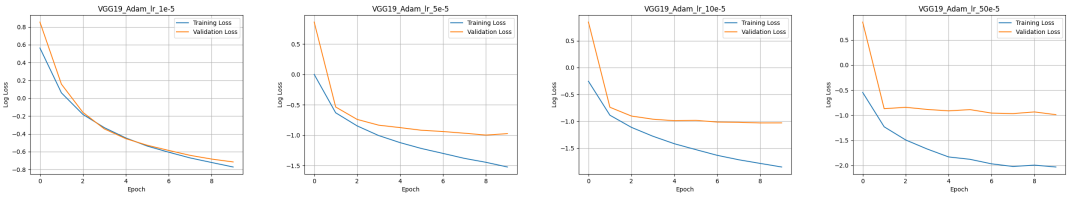

Figure 12. Training performance of VGG19 over 10 epochs

be a little more potential to train the model further with better regularization as it seems to overfit a bit on the training data.

3) Inception_v3: Inception_v3 is a deep convolutional neural network architecture from Google's Inception family, known for its efficiency and strong performance in image classification tasks. It uses the Inception module, which applies multiple filters (1x1, 3x3, and 5x5) in parallel, capturing features at various scales and reducing computational cost with 1x1 convolutions. Inception v3 also includes factorized convolutions, auxiliary classifiers for improved gradient flow, and label smoothing for regularization. With 48 layers, it achieved high accuracy on the ImageNet dataset while balancing depth and computational efficiency.

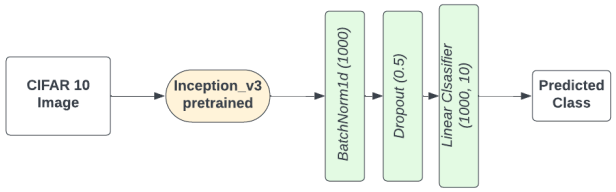

Figure 13. Transfer Learning with Inception_v3

For Transfer Learning with Inception_v3, I essentially froze the pretrained network and added my own classifier head on top of it. Fig. 13 shows the the custom classifier head that is trainable with the following architecture:

- BatchNorm1d(1000)
- Dropout(0.5)
- Linear Classifier (1000, 10)

On the T4 x2 GPU, training time was at 4 mins per epoch.

After training this architecture on 4 different learning rates, the training arcs of each model can be seen in Fig. 14. Table. III also details the performance metrics achieved after training all the models.

Learning Rate	Training Loss	Validation Loss	Test Accuracy (%)	Test F1 Score
0.0005	0.841	0.802	**73.2**	**0.723**
0.0001	0.942	0.918	70.8	0.697
0.00005	1.051	1.025	68.6	0.676
0.00001	1.547	1.539	57.5	0.564

Table III
INCEPTION_V3 PERFORMANCE METRICS AFTER TRAINING FOR 10 EPOCHS

In conclusion, after comparing all the Inception_v3 models, the one I trained with a learning rate of 5e-4 (0.0005) performs the best with 73.2% accuracy and a training time of 38 mins. Additionally, from the loss curve in Fig. 14, the model seems to have saturated without much scope for improvement. I believe changing the complexity of the network might improve the performance, but the current architecture has reached its limit.

In Fig 15, I compared the confusion matrices generated by two of the trained models:

1) VGG19 trained with 5e-5 lr - our best CNN
2) Inception_v3 with 5e-4 lr - a comparable model

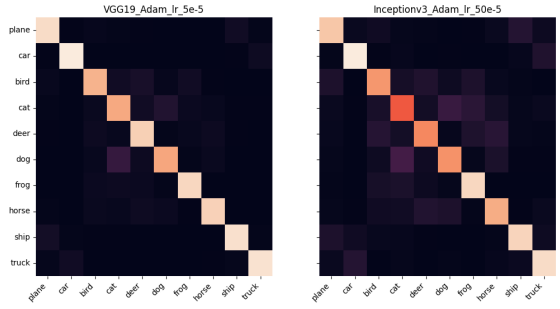

Figure 15. Confusion Matrices produced by CNNs

From the confusion matrix, it is evident that the prediction of the more accurate model (VGG19) are centered around the major diagonal, whereas we see a few more misclassifications for the Inception_v3 model.

IV. VISION TRANSFORMER

In this section, we fit multiple Vision Transformers (ViTs) to the CIFAR10 data. Our experiments and conclusions are detailed in each of the following subsections:

- Building our own ViT
- Transfer Learning - Using pre-trained ViT Models

A. Building our own ViT

I built my own Vision Transformer (ViT) in accordance to the block diagram illustrated in Fig. 18. Before running the experiments, I fixed the following parameters:

- AdamW Optimizer - lr = 5e-4
- Epochs = 20
- Num_Heads = 12 - No. of heads in the MultiHeadSelfAttention segment in the Transformer Encoder

With the above setup, I experimented with 18 architectures, where each I iterated over each of the following hyperparameters:

- patch_size [4, 8] - Patch sizes into which the input 3 x 32 x 32 images are split into

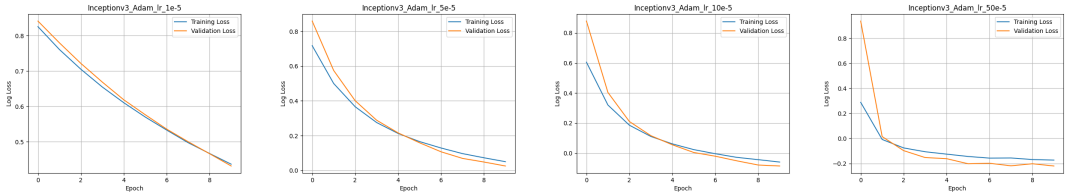

Figure 14. Training performance of Inception_v3 over 10 epochs

Exp No	Patch_size	Embed_dim	Num_layers	Parameters (K)	Train Loss	Val Loss	Test Accuracy (%)	Test F1 Score
1	4	48	4	119.19	1.151	1.158	58.62	0.561
2	4	48	8	232.28	1.042	1.075	61.65	0.591
3	4	48	12	345.37	0.999	1.035	63.12	0.607
4	4	96	4	459.56	0.911	1.018	64.3	0.618
5	4	96	8	906.92	0.815	0.977	66.25	0.64
6	4	96	12	1354.28	0.784	0.961	67.39	0.651
7	4	192	4	1803.85	0.693	1.045	67.95	0.657
8	4	192	8	3583.31	0.637	1.018	68.24	0.658
9	4	192	12	5362.76	0.614	0.949	**69.14**	**0.668**
10	8	48	4	123.8	1.182	1.196	57.03	0.547
11	8	48	8	236.89	1.107	1.149	59.03	0.565
12	8	48	12	349.98	1.076	1.124	59.77	0.574
13	8	96	4	468.78	0.971	1.1	61.52	0.593
14	8	96	8	916.14	0.896	1.087	63.02	0.607
15	8	96	12	1363.5	0.871	1.07	63.05	0.607
16	8	192	4	1822.28	0.785	1.132	63.98	0.618
17	8	192	8	3601.74	0.719	1.104	64.54	0.623
18	8	192	12	5381.19	0.73	1.1	64.55	0.62

Table IV
VIT EXPERIMENTS OVER 20 EPOCHS

- embed_dim [48, 96, 192] - Dimension of the embeddings. Note, this value has to be a multiple of num_heads
- num_layers [4, 8, 12] - No. of sequential transformer encoders

I trained each setup with the P100 GPU provided by Kaggle [2]. On an average, each architecture took 15 mins to train. Table. IV details the performance metrics achieved after training all the models, while simultaneously mentioning the no of parameters in each architecture. Fig. 16 shows the training and validation curves of experiments 6-9, best ViT Models I was able to train. I chose to display only the 4 most relevant plots out of my 18 experiments.

Through my experiments I made the following observations:

1) The bigger the ViT architecture, the more the parameters. As a result, the model seems to overfit to the data much quicker wherein the training loss drops below 1 but the validation loss oscillates around 1.
2) ViTs typically need large amount of data to learn effectively. CIFAR10 is a small dataset with just 50,000 images. As a result, I believe the ViT is underperforming compared to the CNNs we have seen in the past.
3) Encoding spatial information - ViTs encode spatial information with positional embeddings, whereas CNNs do so with spatial filters. For localization on small datasets, CNN's methodology of capturing information is more effective.
4) The best possible way to improve performance would be to transfer learn from a ViT that's already been trained on a large amount of image data. The next section delves into this approach.

In Fig 17, I compared the confusion matrices generated by our best and worst custom ViTs:

1) ViT patch 4 embed_dim 192 num_layers 12 - best ViT
2) ViT patch 4 embed_dim 48 num_layers 4 - worst ViT

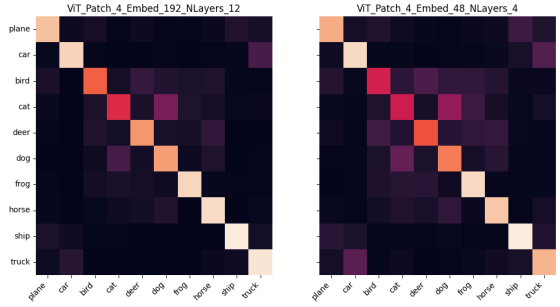

Figure 17. Confusion Matrices produced by ViTs

Similar to our previous confusion matrix comparison, we

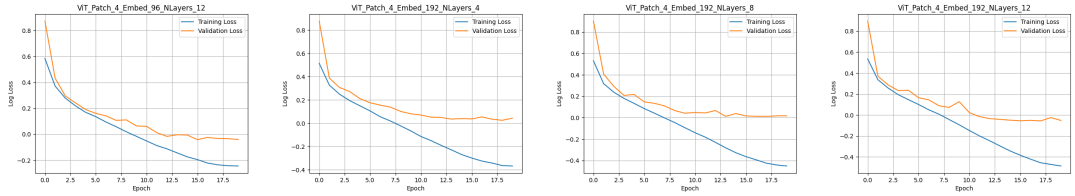

Figure 16. Performance of the 4 best ViTs over 20 epochs

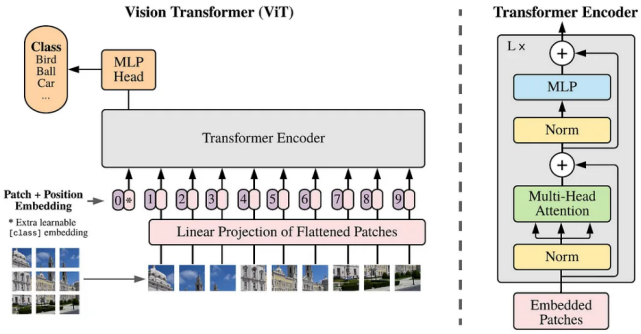

Figure 18. Base ViT Architecture

can see more true positives along the major diagonal for the better model.

B. Transfer Learning - Pre-trained ViT

In this section, I took two pre-trained Vision Transformer models from PyTorch. These models are the VITB-16 and VITB-32 from the original Vision Transformers paper [3] Keeping the pre-trained heads of these transformer models fixed, I added a single fully connected layer as the classification layer corresponding to the correct class outputs. I then observed the performance of these two models for three learning rate values: **0.005, 0.001 and 0.0005**.

The architectures of the following module are given in Fig. 18

As is evidently visible, the two models have an estimated parameter count of 86M for ViTB16 and 88M for ViTB32. The single fully connected layer adds another 7690 parameters. Due to freezing of the overall network, the training times are manageable, yet needed acceleration. For this, the networks were trained on an NVIDIA RTX3060 GPU with Intel i7 CPU. Training times, on average, were about 25mins for 5 epochs, or **5min/epoch** for ViTB16 and 35min for 5 epochs, or **7min/epoch** for ViTB32.

The ViTB16 and ViTB32 architectures differ in the input patch size. ViTB16 takes a 16x16 patch, whereas ViTB32 takes a 32x32 patch. Due to this, we can expect to see a little different behaviour from both, as both the models will find slightly different contextual information about the images. It is also worth noting that these models have been pretrained on ImageNet dataset, which points to a significantly good expected performance.

I made the following observations:

1) Significantly large learning rates, such as 0.005 or 0.001 seem to perform really well. This can be attributed to the fact that only the last layer is being trained, which indicates that a higher learning rate pushes the layer to work faster.
2) Training and validation both take equally long to execute, this means that the overall network size is the main bottleneck to training.
3) The accuracies are quite high, however we observe that the models slightly overfit between epochs 3-5, while training loss reduces slightly. This means that the model has already learnt enough and we are at the boundary between optimal training and high variance. Owing to this, only 5 epochs were chosen.
4) Despite adding just a single learnable layer, performance is splendid. Hence, we do not experiment further with other types of/additional layers.

acc@1 (on ImageNet-1K)	81.072
acc@5 (on ImageNet-1K)	95.318
categories	tench, goldfish, great white shark, ... (997 omitted)
num_params	86567656
min_size	height=224, width=224
recipe	link
GFLOPS	17.56
File size	330.3 MB

Figure 19. ViTB16

V. Adversarial Attack - FGSM

At its core, FGSM is a white-box attack, meaning it requires knowledge of the model's architecture and parameters. The idea is to perturb the input data by adding a small amount of noise based on the gradient of the loss with respect to the input. The simplicity of FGSM lies in its effectiveness in creating adversarial examples with minimal computational cost. Fig. 21 details the design of the attack.

ViT_B-16

Learning Rate	Training Loss	Validation Loss	Test Accuracy (%)
0.005	0.102	0.152	95.15
0.001	0.117	0.149	95.19
0.0005	0.137	0.142	95.06

ViT_B-32

Learning Rate	Training Loss	Validation Loss	Test Accuracy (%)
0.005	0.118	0.171	94.14
0.001	0.14	0.15	94.58
0.0005	0.152	0.168	94.3

Figure 20. Transfer Learning with VITB16 and VITB32

$$x^{adv} = x + \varepsilon.sign(\bigtriangledown_x J(x, y_{true}))$$

$where$

x – Clean Input Image

x^{adv} – Adversarial Image

J – Loss Function

y_{true} – Model Output for x

ε – Tunable Parameter

Figure 21. FGSM design

A. FGSM on our best CNN

After subjecting out best VGG to varying levels of FGSM attacks (denoted by the value of ϵ), we observed massive drops in accuracy and F1 Score. Table. V

Epsilon (ϵ)	Test Acc (%)	Test F1 Score
0 (pre-attack)	87.0	0.867
0.05	23.9	0.169
0.15	20.2	0.143
0.3	17.5	0.112

Table V
PERFORMANCE OF OUR BEST CNN MODEL UNDER FGSM ATTACK

B. FGSM on our best ViT

After subjecting out best ViT to varying levels of FGSM attacks (denoted by the value of ϵ), we observed massive drops in accuracy and F1 Score. Table. VI

Epsilon (ϵ)	Test Acc (%)	Test F1 Score
0 (pre-attack)	95.19	0.949
0.05	25.25	0.221
0.15	20.53	0.159
0.3	15.47	0.096

Table VI
PERFORMANCE OF OUR BEST VIT MODEL UNDER FGSM ATTACK

VI. CONCLUSION

In this project we have extensively researched and built multiple Deep Learning classifiers for multiclass classification of the CIFAR10 dataset. After experimenting with multiple CNNs and ViTs, our best models end up being transfer-learning architectures, highlighting the importance of utilizing models that have learnt to generalize well on tons of image data. However, our experiment with FGSM also highlighted how these data-driven models are susceptible to adversarial attacks. Hence, we have the power of CNNs and ViTs for large scale image classification tasks.

VII. CONTRIBUTIONS

- Building our own CNNs - Ojas
- Transfer Learning: Pre-trained SOTA models: Vaibhav
- Building our own ViT: Vaibhav
- Transfer Learning: Pre-Trained ViT: Ojas
- Adversarial Attack: Vaibhav
- Code Repository and Collation: Ojas
- Report Making: 50-50, respective sections

REFERENCES

[1] Cifar-10 and cifar-100 datasets. https://www.cs.toronto.edu/~kriz/cifar.html.

[2] Kaggle: A google platform for data science and ml. https://www.kaggle.com/.

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.