Deep Learning CS577

Project Report

IMAGE CLASSIFICATION WITH VISION TRANSFORMER

BHOOMIKA PANDURANGA
A20503493
bpanduranga@hawk.iit.edu

MOHAMMAD AUSAF ALI HAQQANI
A20516413
malihaqqani@hawk.iit.edu

Computer Science department, Illinois Institute of Technology, Chicago-60616

Abstract: The transformers architecture is recognized for Natural language processing tasks but its applications for the computer vision has remained limited. In case of computer vision the convolution neural networks or certain components replacement keeping the overall structure is found to be used everywhere. This causes the reliance on CNNs for vision tasks. This reliance can be eradicated by using pure transformer when applied directly to sequences of image patches, pretrained on large amounts of data and transferred to multiple mid-sized or small image recognition benchmarks (ImageNet, CIFAR-100, VTAB, etc.), Vision Transformer attains excellent results compared to state-of -the-art convolution networks while requiring substantially fewer computational resources to train.

I Problem Statement:

The goal of this project is to build a Transformer for Image classification, motivated by the Transformer scaling triumphs in NLP that can minimize the dependency on convolution neural networks since they are not scaled for larger applications in computer vision. In case of transformers (Sequence length) the computational cost is inversely proportional to the patch size of the Image.

II Proposed solution:

We investigate with applying a typical Transformer straight to Images (CIFAR-100 dataset) with the least number of alterations. To do this, we divide an image into patches and feed the series of their linear embeddings into a Transformer. Like NLP application, image patches are handled in the same manner as tokens (words). We use supervised learning to train the model for Image classification.

III Implementation:

1. Method:
   We try to create our models as nearly as feasible to the actual Transformer (Vaswani et al., 2017).The fact that scalable NLP Transformer topologies and their effective implementations can be employed almost right out of the box is a benefit of this setup's deliberate simplicity.

   The first model implemented is a transformer with the following values of the hyperparameters.
   learning_rate = 0.001
   weight_decay = 0.0001

```
batch_size = 32
num_epochs = 30
image_size = 72
patch_size = 6
num_patches = (image_size // patch_size) ** 2
projection_dim = 128
num_heads = 8
transformer_units = [
    projection_dim * 2,
    projection_dim,
]
transformer_layers = 10
mlp_head_units = [2048, 1024]
```

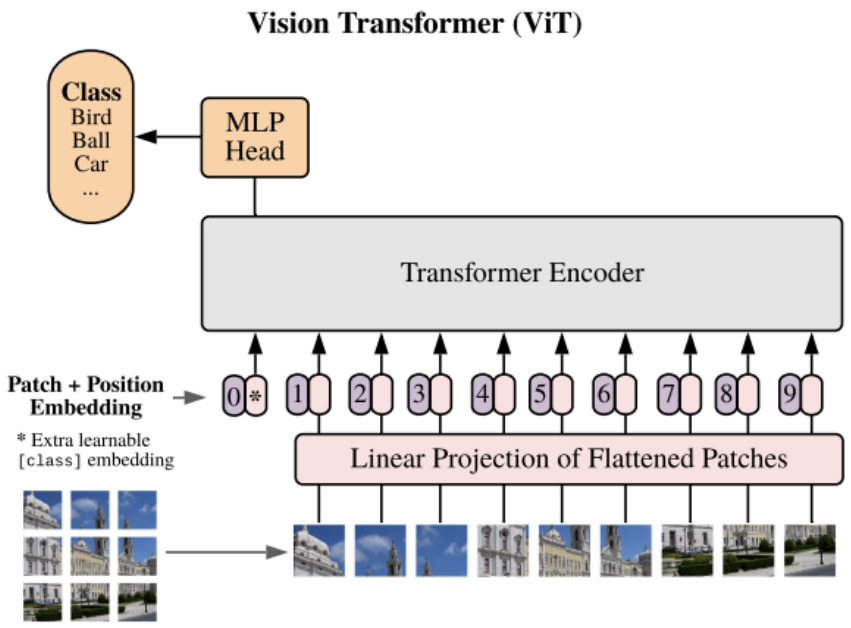
Position embeddings are added to the patch embeddings to retain positional information. We use standard learnable 1D position embeddings, since we have not observed significant performance gains from using more advanced 2D-aware position embeddings The resulting sequence of embedding vectors serves as input to the encoder.

1-dimensional positional embedding: Considering the inputs as a sequence of patches.

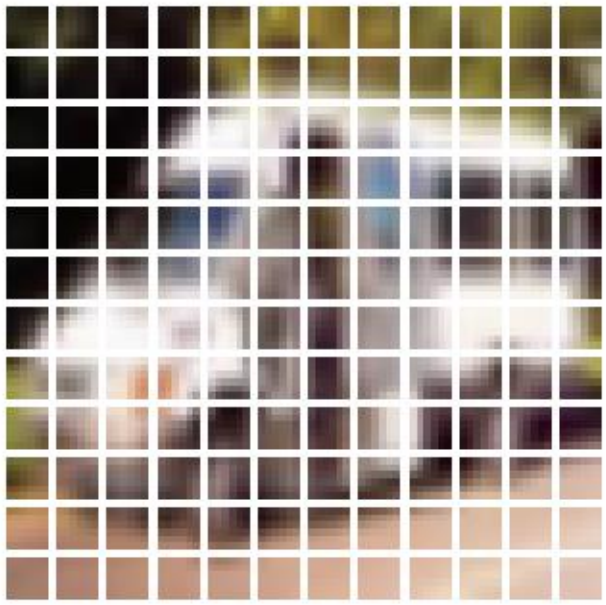
2-dimensional positional embedding: Considering the inputs as a grid of patches in two dimensions. In this case, two sets of embeddings are learned, each for one of the axes, X-embedding, and Y -embedding, each with size D/2. Then, based on the coordinate on the path in the input, we concatenate the X and Y embedding to get the final positional embedding for that patch.

Relative positional embeddings: Considering the relative distance between patches to encode the spatial information as instead of their absolute position. To do so, we use 1-dimensional Relative Attention, in which we define the relative distance all possible pairs of patches. Thus, for every given pair (one as query, and the other as key/value in the attention mechanism), Then, we simply run extra attention, where we use the original query (the content of query), but use relative positional embeddings as keys. We then use the logits from the relative attention as a bias term and add it to the log.

We note that Vision Transformer has much less image-specific inductive bias than CNNs. In CNNs, locality, two-dimensional neighborhood structure, and translation equivariance are baked into each layer throughout the whole model. In ViT, only MLP layers are local and translationally equivariant, while the self-attention layers are global. The two-dimensional neighborhood structure is used very sparingly: in the beginning of the model by cutting the image into patches and Other than that, the position embeddings at initialization time carry no information about the 2D positions of the patches and all spatial relations between the patches have to be learned from scratch.




```
Image size: 72 X 72
Patch size: 6 X 6
Patches per image: 144
Elements per patch: 108
```



The below is the architecture of the transformer model:

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 32, 32, 3)]  0           []

 data_augmentation (Sequential)  (None, 72, 72, 3)   7           ['input_1[0][0]']

 patches_1 (Patches)            (None, None, 108)    0           ['data_augmentation[0][0]']

 patch_encoder (PatchEncoder)   (None, 144, 128)     32384       ['patches_1[0][0]']

 layer_normalization (LayerNorm  (None, 144, 128)    256         ['patch_encoder[0][0]']
 alization)

 multi_head_attention (MultiHea  (None, 144, 128)    527488      ['layer_normalization[0][0]',
 dAttention)                                                      'layer_normalization[0][0]']
```

```
 add (Add)                      (None, 144, 128)     0           ['multi_head
_attention[0][0]',
                                                                  'patch_enco
der[0][0]']

 layer_normalization_1 (LayerNo  (None, 144, 128)    256         ['add[0][0]'
]
 rmalization)

 dense_1 (Dense)                (None, 144, 256)     33024       ['layer_norm
alization_1[0][0]']

 dropout (Dropout)              (None, 144, 256)     0           ['dense_1[0]
[0]']

 dense_2 (Dense)                (None, 144, 128)     32896       ['dropout[0]
[0]']

 dropout_1 (Dropout)            (None, 144, 128)     0           ['dense_2[0]
[0]']

 add_1 (Add)                    (None, 144, 128)     0           ['dropout_1[
0][0]',
                                                                  'add[0][0]'
]

 layer_normalization_2 (LayerNo  (None, 144, 128)    256         ['add_1[0][0
]']
 rmalization)

 multi_head_attention_1 (MultiH  (None, 144, 128)    527488      ['layer_norm
alization_2[0][0]',
 eadAttention)                                                    'layer_norm
alization_2[0][0]']

 add_2 (Add)                    (None, 144, 128)     0           ['multi_head
_attention_1[0][0]',
                                                                  'add_1[0][0
]']

 layer_normalization_3 (LayerNo  (None, 144, 128)    256         ['add_2[0][0
]']
 rmalization)
```

```
 dense_3 (Dense)                (None, 144, 256)     33024       ['layer_norm
alization_3[0][0]']

 dropout_2 (Dropout)            (None, 144, 256)     0           ['dense_3[0]
[0]']

 dense_4 (Dense)                (None, 144, 128)     32896       ['dropout_2[
0][0]']

 dropout_3 (Dropout)            (None, 144, 128)     0           ['dense_4[0]
[0]']

 add_3 (Add)                    (None, 144, 128)     0           ['dropout_3[
0][0]',
                                                                  'add_2[0][0
]']

 layer_normalization_4 (LayerNo  (None, 144, 128)    256         ['add_3[0][0
]']
 rmalization)

 multi_head_attention_2 (MultiH  (None, 144, 128)    527488      ['layer_norm
alization_4[0][0]',
 eadAttention)                                                    'layer_norm
alization_4[0][0]']

 add_4 (Add)                    (None, 144, 128)     0           ['multi_head
_attention_2[0][0]',
                                                                  'add_3[0][0
]']

 layer_normalization_5 (LayerNo  (None, 144, 128)    256         ['add_4[0][0
]']
 rmalization)

 dense_5 (Dense)                (None, 144, 256)     33024       ['layer_norm
alization_5[0][0]']

 dropout_4 (Dropout)            (None, 144, 256)     0           ['dense_5[0]
[0]']

 dense_6 (Dense)                (None, 144, 128)     32896       ['dropout_4[
0][0]']
```

```
 dropout_5 (Dropout)            (None, 144, 128)     0           ['dense_6[0]
[0]']

 add_5 (Add)                    (None, 144, 128)     0           ['dropout_5[
0][0]',
                                                                  'add_4[0][0
]']

 layer_normalization_6 (LayerNo  (None, 144, 128)    256         ['add_5[0][0
]']
 rmalization)

 multi_head_attention_3 (MultiH  (None, 144, 128)    527488      ['layer_norm
alization_6[0][0]',
 eadAttention)                                                    'layer_norm
alization_6[0][0]']

 add_6 (Add)                    (None, 144, 128)     0           ['multi_head
_attention_3[0][0]',
                                                                  'add_5[0][0
]']

 layer_normalization_7 (LayerNo  (None, 144, 128)    256         ['add_6[0][0
]']
 rmalization)

 dense_7 (Dense)                (None, 144, 256)     33024       ['layer_norm
alization_7[0][0]']

 dropout_6 (Dropout)            (None, 144, 256)     0           ['dense_7[0]
[0]']

 dense_8 (Dense)                (None, 144, 128)     32896       ['dropout_6[
0][0]']

 dropout_7 (Dropout)            (None, 144, 128)     0           ['dense_8[0]
[0]']

 add_7 (Add)                    (None, 144, 128)     0           ['dropout_7[
0][0]',
                                                                  'add_6[0][0
]']
```

```
 layer_normalization_8 (LayerNo  (None, 144, 128)    256         ['add_7[0][0
]']
 rmalization)

 multi_head_attention_4 (MultiH  (None, 144, 128)    527488      ['layer_norm
alization_8[0][0]',
 eadAttention)                                                   'layer_norm
alization_8[0][0]']

 add_8 (Add)                    (None, 144, 128)     0           ['multi_head
_attention_4[0][0]',
                                                                 'add_7[0][0
]']

 layer_normalization_9 (LayerNo  (None, 144, 128)    256         ['add_8[0][0
]']
 rmalization)

 dense_9 (Dense)                (None, 144, 256)     33024       ['layer_norm
alization_9[0][0]']

 dropout_8 (Dropout)            (None, 144, 256)     0           ['dense_9[0]
[0]']

 dense_10 (Dense)               (None, 144, 128)     32896       ['dropout_8[
0][0]']

 dropout_9 (Dropout)            (None, 144, 128)     0           ['dense_10[0
][0]']

 add_9 (Add)                    (None, 144, 128)     0           ['dropout_9[
0][0]',
                                                                 'add_8[0][0
]']

 layer_normalization_10 (LayerN  (None, 144, 128)    256         ['add_9[0][0
]']
 ormalization)

 multi_head_attention_5 (MultiH  (None, 144, 128)    527488      ['layer_norm
alization_10[0][0]',
 eadAttention)                                                   'layer_norm
alization_10[0][0]']
```

```
 add_10 (Add)                   (None, 144, 128)     0           ['multi_head
_attention_5[0][0]',
                                                                  'add_9[0][0
]']

 layer_normalization_11 (LayerN  (None, 144, 128)    256         ['add_10[0][
0]']
 ormalization)

 dense_11 (Dense)               (None, 144, 256)     33024       ['layer_norm
alization_11[0][0]']

 dropout_10 (Dropout)           (None, 144, 256)     0           ['dense_11[0
][0]']

 dense_12 (Dense)               (None, 144, 128)     32896       ['dropout_10
[0][0]']

 dropout_11 (Dropout)           (None, 144, 128)     0           ['dense_12[0
][0]']

 add_11 (Add)                   (None, 144, 128)     0           ['dropout_11
[0][0]',
                                                                  'add_10[0][
0]']

 layer_normalization_12 (LayerN  (None, 144, 128)    256         ['add_11[0][
0]']
 ormalization)

 multi_head_attention_6 (MultiH  (None, 144, 128)    527488      ['layer_norm
alization_12[0][0]',
 eadAttention)                                                    'layer_norm
alization_12[0][0]']

 add_12 (Add)                   (None, 144, 128)     0           ['multi_head
_attention_6[0][0]',
                                                                  'add_11[0][
0]']

 layer_normalization_13 (LayerN  (None, 144, 128)    256         ['add_12[0][
0]']
 ormalization)
```

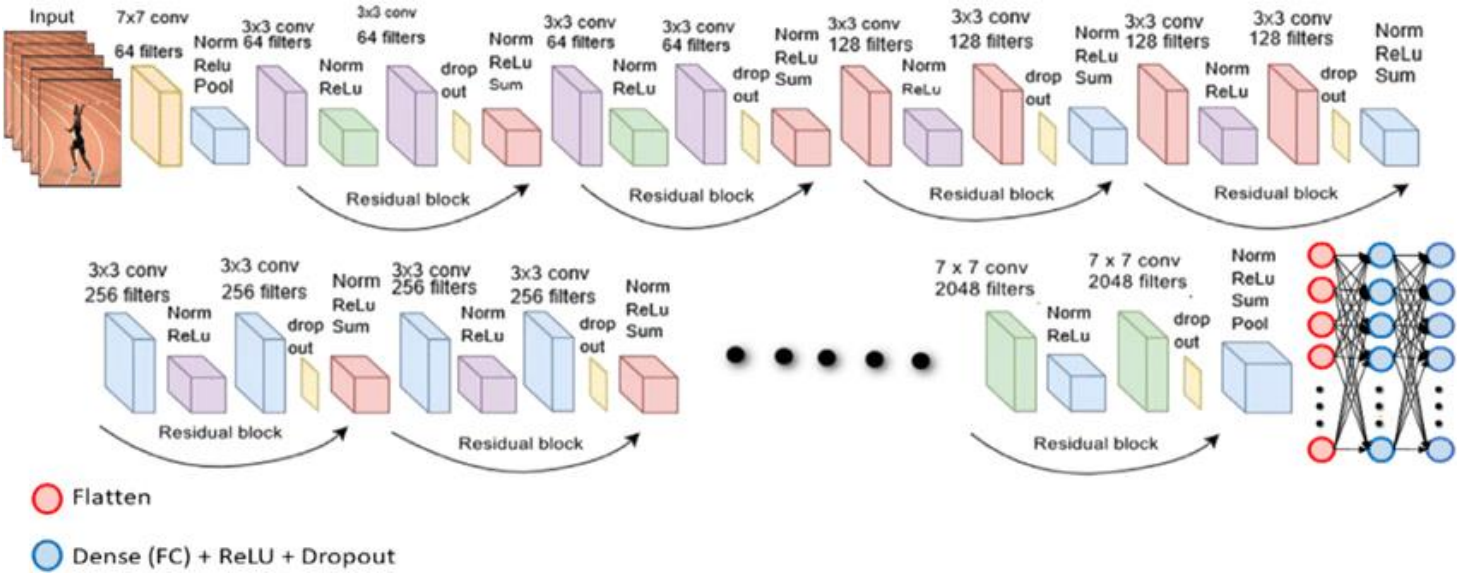
```
 dense_13 (Dense)               (None, 144, 256)     33024       ['layer_norm
alization_13[0][0]']

 dropout_12 (Dropout)           (None, 144, 256)     0           ['dense_13[0
][0]']

 dense_14 (Dense)               (None, 144, 128)     32896       ['dropout_12
[0][0]']

 dropout_13 (Dropout)           (None, 144, 128)     0           ['dense_14[0
][0]']

 add_13 (Add)                   (None, 144, 128)     0           ['dropout_13
[0][0]',
                                                                  'add_12[0][
0]']

 layer_normalization_14 (LayerN  (None, 144, 128)    256         ['add_13[0][
0]']
 ormalization)

 multi_head_attention_7 (MultiH  (None, 144, 128)    527488      ['layer_norm
alization_14[0][0]',
 eadAttention)                                                    'layer_norm
alization_14[0][0]']

 add_14 (Add)                   (None, 144, 128)     0           ['multi_head
_attention_7[0][0]',
                                                                  'add_13[0][
0]']

 layer_normalization_15 (LayerN  (None, 144, 128)    256         ['add_14[0][
0]']
 ormalization)

 dense_15 (Dense)               (None, 144, 256)     33024       ['layer_norm
alization_15[0][0]']

 dropout_14 (Dropout)           (None, 144, 256)     0           ['dense_15[0
][0]']

 dense_16 (Dense)               (None, 144, 128)     32896       ['dropout_14
[0][0]']
```

```
 dropout_15 (Dropout)           (None, 144, 128)     0           ['dense_16[0
][0]']

 add_15 (Add)                   (None, 144, 128)     0           ['dropout_15
[0][0]',
                                                                 'add_14[0][
0]']

 layer_normalization_16 (LayerN  (None, 144, 128)    256         ['add_15[0][
0]']
 ormalization)

 multi_head_attention_8 (MultiH  (None, 144, 128)    527488      ['layer_norm
alization_16[0][0]',
 eadAttention)                                                   'layer_norm
alization_16[0][0]']

 add_16 (Add)                   (None, 144, 128)     0           ['multi_head
_attention_8[0][0]',
                                                                 'add_15[0][
0]']

 layer_normalization_17 (LayerN  (None, 144, 128)    256         ['add_16[0][
0]']
 ormalization)

 dense_17 (Dense)               (None, 144, 256)     33024       ['layer_norm
alization_17[0][0]']

 dropout_16 (Dropout)           (None, 144, 256)     0           ['dense_17[0
][0]']

 dense_18 (Dense)               (None, 144, 128)     32896       ['dropout_16
[0][0]']

 dropout_17 (Dropout)           (None, 144, 128)     0           ['dense_18[0
][0]']

 add_17 (Add)                   (None, 144, 128)     0           ['dropout_17
[0][0]',
                                                                 'add_16[0][
0]']
```

```
 layer_normalization_18 (LayerN  (None, 144, 128)    256         ['add_17[0][
0]']
 ormalization)

 multi_head_attention_9 (MultiH  (None, 144, 128)    527488      ['layer_norm
alization_18[0][0]',
 eadAttention)                                       'layer_norm
alization_18[0][0]']

 add_18 (Add)                   (None, 144, 128)     0           ['multi_head
_attention_9[0][0]',
                                                     'add_17[0][
0]']

 layer_normalization_19 (LayerN  (None, 144, 128)    256         ['add_18[0][
0]']
 ormalization)

 dense_19 (Dense)               (None, 144, 256)     33024       ['layer_norm
alization_19[0][0]']

 dropout_18 (Dropout)           (None, 144, 256)     0           ['dense_19[0
][0]']

 dense_20 (Dense)               (None, 144, 128)     32896       ['dropout_18
[0][0]']

 dropout_19 (Dropout)           (None, 144, 128)     0           ['dense_20[0
][0]']

 add_19 (Add)                   (None, 144, 128)     0           ['dropout_19
[0][0]',
                                                     'add_18[0][
0]']

 layer_normalization_20 (LayerN  (None, 144, 128)    256         ['add_19[0][
0]']
 ormalization)

 flatten (Flatten)              (None, 18432)        0           ['layer_norm
alization_20[0][0]']

 dropout_20 (Dropout)           (None, 18432)        0           ['flatten[0]
[0]']
```

```
 dense_21 (Dense)                (None, 2048)         37750784    ['dropout_20
[0][0]']

 dropout_21 (Dropout)            (None, 2048)         0           ['dense_21[0
][0]']

 dense_22 (Dense)                (None, 1024)         2098176     ['dropout_21
[0][0]']

 dropout_22 (Dropout)            (None, 1024)         0           ['dense_22[0
][0]']

 dense_23 (Dense)                (None, 100)          102500      ['dropout_22
[0][0]']

==================================================================================
====================
Total params: 45,923,307
Trainable params: 45,923,300
Non-trainable params: 7
__________________________________________________________________________________
____________________
```

To compare and prove the that transformer architecture will not work good if it is not pre-trained on large dataset, The Resnet101V2 architecture was implemented with the following structure of the model:



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet101v2 (Functional)    (None, 1, 1, 2048)        42626560

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 512)               1049088

 gaussian_dropout (GaussianD  (None, 512)              0
 ropout)

 dense_1 (Dense)             (None, 1024)              525312

 gaussian_dropout_1 (Gaussia  (None, 1024)             0
 nDropout)

 dense_2 (Dense)             (None, 100)               102500

=================================================================
Total params: 44,303,460
Trainable params: 44,205,796
Non-trainable params: 97,664
_________________________________________________________________
```

The activation for the last dense layer is softmax and we have relu activation for the other two dense layers.

The above are the hyperparameter values for the Resnet101V2 model which was trained for 30 epochs.

The training and testing were not normalized and thereby using sparse categorical cross entropy as the loss function.

2. Dataset:
   To explore model scalability, we use the CIFAR-100 dataset with 100 classes and 60000 images. 50000 images were used as training data and 10000 as test data.
   The training data was split into 40000 and 10000. This was used for both the models.

   Data augmentation is also done for both the models:

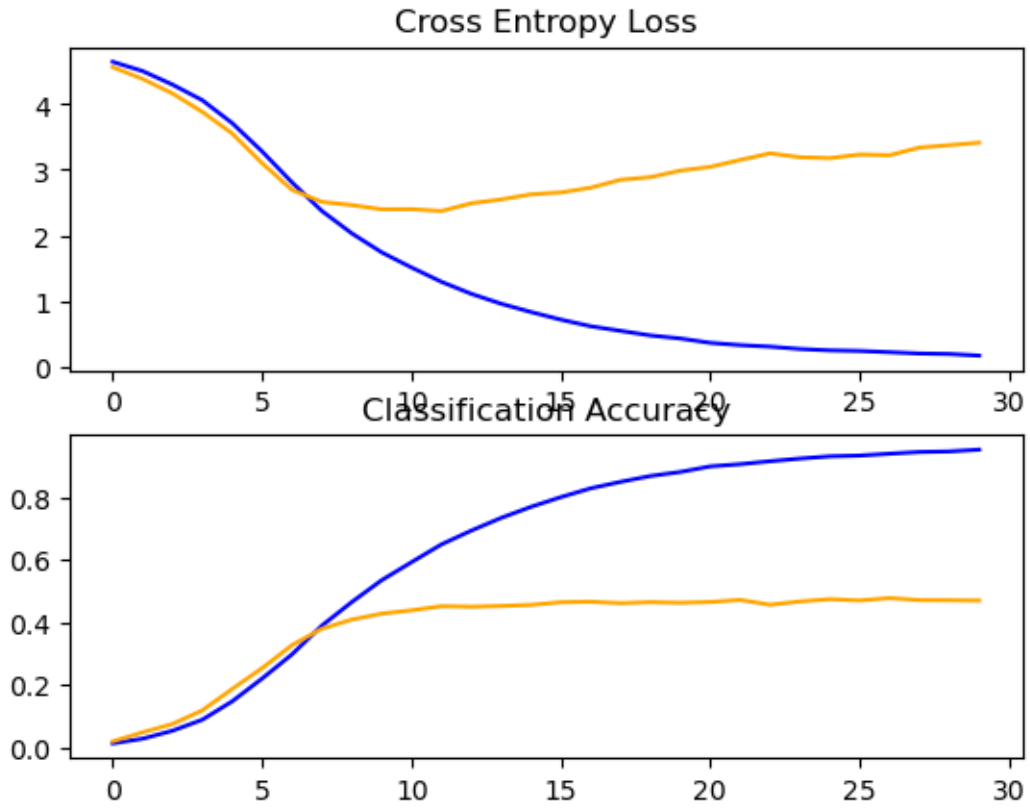
   1. For transformer:
      layers.Normalization layers.Normalization(),

layers.Resizing(image_size, image_size),
layers.RandomFlip("horizontal"),
layers.RandomRotation(factor=0.02),
layers.RandomZoom(height_factor=0.2, width_factor=0.2)

2. For Resnet101V2:
   rotation_range=20,
   width_shift_range=0.2,
   height_shift_range=0.2,
   shear_range=0.2,
   zoom_range=0.2,
   horizontal_flip=True

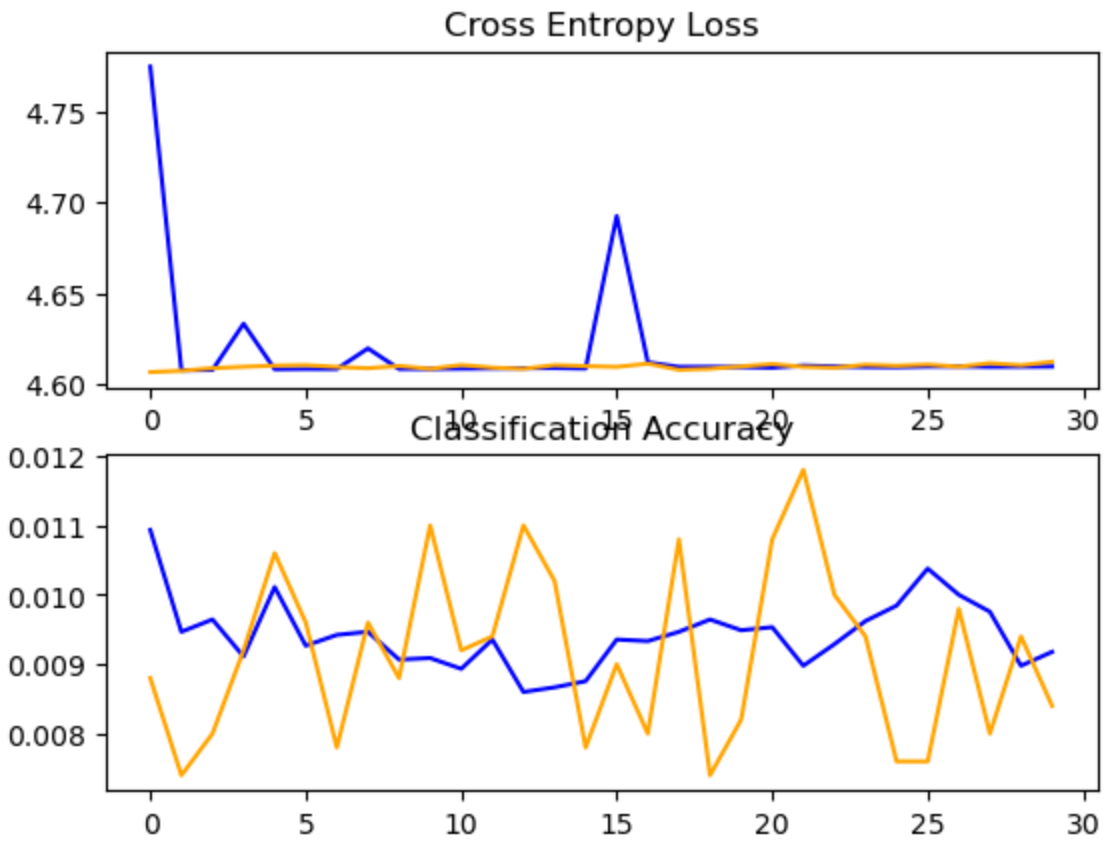
IV Results:

1. Resnet101V2 architecture results:



The test accuracy was: `313/313 [==============================] - 3s 10ms/step - loss: 3.3324 - accuracy: 0.4793`
`the accuracy is: 47.929999232292175`

2. Transformer Architecture result:




```
313/313 [==============================] - 5s 17ms/step - loss: 4.6079
- accuracy: 0.0100 - top-5-accuracy: 0.0500 Test accuracy: 1.0% Test
top 5 accuracy: 5.0%
```

V Conclusion:

Large dataset is needed for pretraining the transformer model to overcome the Inductive bias. Convolutional Inductive bias is useful for smaller datasets but for larger datasets the inductive bias inherent to CNN's is irrelevant as the Transformer learns the relevant patterns by itself. Requires high end TPUv3 in order to pretrain the network on large datasets (JFT-300). This model is Scalable and works well when pretrained on large datasets. Thus, vision transformers exceed or matches the well performed models of the image classification dataset even after being economical to pre train.

VI Future work:

Pretrain the model on large datasets(JFT-300).
Using ViT for segmentation and detection tasks for computer vision.
Explore self-supervised pre-training methods.
Further Scaling of ViT.

VII Steps to run the program:

1.Download the CIFAR100 dataset and load it to the program.

2. Run the program on Jupyter Notebook on GPU.

VIII Team Member Responsibilities

1. Bhoomika: Data Preparation, Data Augmentation and Patch Creation as a Layer.
2. Ausaf: Implement patch encoding layer and Build the Vision Transformer model.
3. Bhoomika and Ausaf: Report, Presentation, and Research.

XI References:

[1]. https://keras.io/examples/vision/image_classification_with_vision_transformer/
[2]. https://arxiv.org/abs/2010.11929
[3]. Bello, B. Zoph, Q. Le, A. Vaswani, and J. Shlens. Attention augmented convolutional networks. In ICCV, 2019.
[4]. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In NAACL, 2019.