Implementation of DeiT - Data-Efficient Image Transformer: A Comparative Study with Vision Transformers (ViT)

Aravind Balaji Srinivasan
A20563386
Contributed to DeiT and implemented Data Augmentation.
Collaborated on Project report and presentation.

Vignesh Ram Ramesh Kutti
A20548747
Developed Vit and contributed to DeiT.
Collaborated on Project report and presentation.

Abstract

The project focuses on Vision Transform (ViT) and Data efficient image Transformer (DeiT) as an alternative for the traditional Convolution Neural Networks (CNN) for the image classification tasks. While the ViT offers a very competitive accuracy, it can be trained only with very large datasets. To avoid such a scenario, we use DeiT on CIFAR-10 dataset, which uses transfer learning to avoid this problem. In this study, we explore DeiT's performance further by incorporating advanced data augmentation methods—CutMix, MixUp, Random Erasing, and Repeated Augmentation—to enhance generalization.

We compare the three models—ViT, DeiT, and DeiT with Augmentation—across key metrics including Top-1 Accuracy, Top-5 Accuracy, AUC, F1 Score, Precision, and Recall. Our results show that DeiT generally performs better than ViT on CIFAR-10, with DeiT's performance further boosted by data augmentation. This study highlights DeiT's suitability for smaller datasets and the effectiveness of augmentation in improving model robustness and generalization for resource-limited applications.

1 Project Information

The main paper selected for this project is: - Touvron et al., **"Training Data-Efficient Image Transformers & Distillation through Attention"**. For this project, we implemented the Data-Efficient Image Transformer (DeiT) and substantial modifications were made to incorporate advanced data augmentation techniques such as CutMix, MixUp, Random Erasing, and Repeated Augmentation.

2 Problem Statement

While transformers have shown tremendous success in natural language processing (NLP), their application to image classification has been data-intensive. Vision Transformers (ViT) need large-scale datasets (like ImageNet-21k) to achieve competitive performance, which is not feasible in many real-world scenarios. DeiT addresses this challenge by introducing transfer learning through a distillation mechanism reducing the dependency on vast datasets, enabling efficient training even on smaller datasets. The project aims to implement DeiT and evaluate its performance.

3 Proposed Solution and Implementation Details

Our proposed solution leverages Data-Efficient Image Transformers (DeiTs), which improve upon ViTs by using knowledge distillation techniques. Additionally, we incorporated advanced data augmentation methods such as:

- CutMix
- MixUp
- Random Erasing

3.1 Basic Transformer Architecture

To Understand Transformers, we need to delve into the concept of Attention. What is **Attention** exactly? Attention in transformer focuses on specific parts of an input be it a sequence of words or a sequence of patches of an image. This mechanism assigns importance to various different parts of the input which enables the model to capture relationships between the input sequences and their context effectively. Transformers are models which use the self-attention techniques, to improve the ability to grasp long-range relationships, enable parallel computation, and increase the efficiency of processing sequences. They are made of two primary components: **Encoder** and **Decoder**.

3.1.1 Encoder Stack

The Encoder stack is usually made of N identical layer with two major components :

- **Multi-Head Attention**: This component allows the model to look at each position in the input sequence and attend to all other positions. By doing this, the model can capture complex relationships between different parts of the input. The ”multi-head” aspect means that the model processes different parts of the sequence in parallel, attending to multiple representation subspaces at once, which helps it learn different kinds of patterns simultaneously.
- **Feed Forward Network**: After the attention mechanism, each position’s output is passed through a fully connected feed-forward network. This network processes the output independently for each position in the sequence, helping the model to refine the representations further

3.1.2 Decoder Stack

- **Masked Multi-Head Attention**: This layer takes the previous inputs into account while processing attention but leaves out future positions this is called masking. Before applying softmax function we will initialize all future input positions with $-\infty$ so that after applying softmax function they become 0, without disturbing the probability distribution.

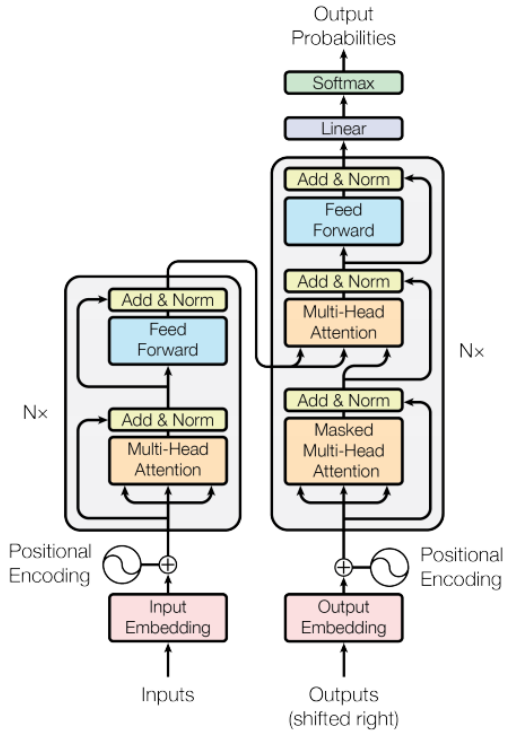

Figure 1: Basic Transformer Architecture showing encoder-decoder structure with multi-head attention mechanisms and Feed Forward Network

- The Encoder and Decoder layers are connected to each other. This enables the decoder in learning to attend to specific positions in the input sequence that are most relevant for generating the current output token.

3.1.3 How to Calculate Attention?

Attention is calculated using formula:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Keys (K), Queries(Q), Values(V): Let X be the input matrix containing the input values. $X \in R^{T\times d}$. Where T is the length of the sequence be it tokens or patches, d is Dimensionality of embeddings. $Q = XW_Q, \quad K = XW_K, \quad V = XW_V$, where W_Q, W_K, W_V are learnable weight matrices, they get updated after each iteration. $Q,K,V \in R^{T\times d_k}$ where $d_k = \frac{d}{h}$ where h is the number of attention heads. Let us look at a sample calculation:

The input sequence X has shape (4×512) (4 tokens, 512-dimensional embeddings).

The weight matrices W_Q, W_K, W_V have shape (512×64), where $d_k = 64$.

1. Compute Q: $Q = XW_Q$ Resulting Q has shape (4×64).
2. Similarly, compute K and V: $K = XW_K, \quad V = XW_V$

   Both K and V will also have shape (4×64).
3. Use Q,K,V to calculate attention: $\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$

3.2 Vision Transformer

Now let us look at Vision Transformers (ViT). ViTs apply the same concept of transformers but for images. The image is divided into patches and these patches are flattened and fed as input to the transformer.

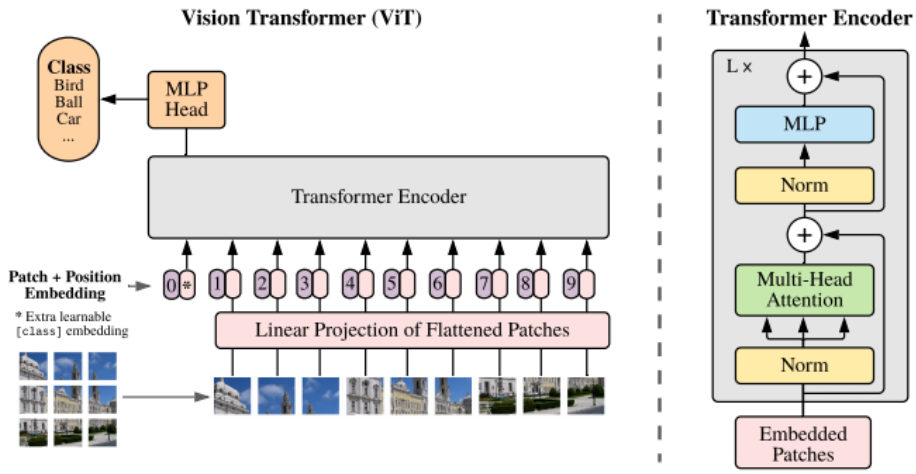

Figure 2: The images is divided into fixed-size patches, which are then linearly embedded. Positional embeddings are added to these patches, and the resulting sequence of vectors is passed through a standard Transformer encoder. For classification, a learnable "classification token" is appended to the sequence, following the standard approach.

3.2.1 Vision Transformer Architecture

The input to the model is patches of an image. The patches must not overlap. The patches are flattened before feeding to the transformer layer forming a vector.

- **Patch Embeddings:** The vector which is a result of flattening is then linearly projected into a feature space of dimension d. If the input has size $H \times W$ and patch size $P \times P$, number of patches is given by: $N = \frac{H \times W}{P^2}$
- **Positional Embeddings:** To keep a record of the order of sequence of patches we include positional embeddings are added to patch embeddings to encode spatial information.
- **Classification Token:** A classification token is added to the patch embeddings. This token is learnable as model trains. After the model finishes training, this token represents the class of the image.
- Similar to basic transformer, the sequence of patches and class tokens are passed through a Transformer encoder. This encoder also contains the Multi-head self attention, which captures the relation between patches globally and the Feed-Forward Network. The normalization layer and residual (skip) connections are applied as well.
- **Classification Head** is the output of class token when it is processed through a fully-connected layer for classification.

3.2.2 Parameters for ViT and DeiT

- **Input Parameters:**
  - `image_size` = size of input image (32×32) pixels.
  - `patch_size` = number of Patches calculated using formula for N = 4
- **Embedding Parameters:**
  - `embedding_dim`: dimension of each patch vector represented by $d = 4 \times 4 \times 4$.
  - `positional_embedding`: this parameter assists performing positional encoding. This has the same dimension as `embedding_dim` $= 4 \times 4 \times 4$.
- **Transformer Encoder Parameters:**
  - `num_layers`: number of layers of the Transformer Encoder. This is the depth of the model and depth $\propto$ performance = 7.
  - `num_heads`: number of attention heads given by $\frac{\texttt{embedding\_dim}}{head\_dim} = \frac{64}{8} = 8$.
  - `mlp_dim`: dimensions of feed-forward network in each encoder layer. Increasing this more than `embedding_dim` can capture better representations = 256.
- **Classification Parameters:**
  - `num_classes`: number of output classes for classification. this determines the size of output layer in the classification head = 10.
  - `cls_token` and `distillation_token`: this are learnable parameters appended in front of the sequence of patches (dimension = $4 \times 4 \times 4$).
- **Dropout and Regularization:**
  - `dropout_rate`: This rate determines the amount of neurons to randomly exclude while processing. This will help in reducing overfitting the model. In our case `dropout_rate` = 0.1 .
- **Optimization Parameters:**
  - `learning_rate`: This rate determines the step size for the optimizer during training. This is set by trial and error method. ($\eta = 0.001$)
  - `batch_size`: number of images processed together during training. higher batch size requires more computational power. (Batch size = 64)
  - `epochs`: number of times the entire dataset is passed through the model. (epochs = 20)
  - `optimizer`: Adam (Adaptive Moment Estimation) is used as optimizer.
  - `criterion`: Cross-Entropy Loss is used.

To achieve competitive performance of ViT we need massive amounts of data which is widely available but computationally expensive, to tackle this problem we introduce DeiT.

3.3 Data-Efficient Image Transformer (DeiT)

DeiT (Data-efficient Image Transformer) is a type of vision transformer which is designed with focus on achieving high performance on image classification with smaller datasets. This is done because ViTs are data-hungry. Original Vision Transformer is trained on JFT-300M dataset from Google, it has 300 million images. DeiT addresses this nature to implement **Distillation** a special type of transfer learning where a teacher model (Pre-trained model) will help in training our DeiT student model. The parameters for DeiT is same as ViT with addition of training with `Distillation_Loss`.

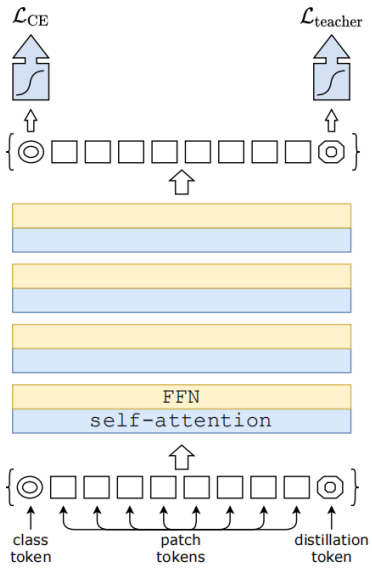

Figure 3: A new distillation token is added which interacts with the class and patch tokens through the self-attention layers

Implementation Flow:

- **Input Image**:The 32×32 input image is divided into 4×4 non-overlapping patches instead of normal 16x16 in vanilla vision transformer because our input dataset image size is reduced from 224x224 to 32x32. These patches are flattened and projected into a 64-dimensional embedding, forming patch tokens.
- **Dual Token System**:
  - **Classification Token [CLS]**: A learnable classification token ([CLS]) is appended in front of the patch tokens, which will represent the class the image belongs to.
  - **Distillation Token**: A distillation token is added alongside the class token, used to transfer knowledge from the teacher model during training.
- **Patch Embedding**: Positional embeddings are added to each patch token, classification token, and distillation token to maintain the positional relationships between patches.
- **Transformer Encoder**: The sequence of patch embeddings, along with the classification and distillation tokens, is passed through 7 transformer encoder layers. Each layer contains an 8-head self-attention mechanism and a feed-forward network, learning dependencies between the patches and tokens.
- **Knowledge Distillation:**: **Knowledge Distillation** or Distillation for short is a method where a small "student" model in our case the DeiT learns from a large pre-trained "teacher" model in our case the ResNet-50. The original teacher model is RegNetY-16 used by the authors of DeiT paper, In our case such a huge pre-trained model is unnecessary. Resnet-50 is more fine tuned towards training in CIFAR-10 dataset, so it would be the best choice for our implementation.The student learns from teacher's predictions on top of learning from the labeled data, teacher's predictions have more rich and nuanced information. This process, known as knowledge distillation, typically involves minimizing a loss function that balances the student's accuracy on true labels and its similarity to the teacher's output. Distillation allows the student model to achieve better performance with reduced resources, making it efficient for tasks where a lightweight model is advantageous.

  There are mainly two types of distillations:

- **Hard distillation** trains the student to match only the teacher's final class predictions, focusing on exact labels.
- **Soft distillation** uses the teacher's full probability distribution across classes, giving the student richer information on class relationships.

- **Distillation Objective:**:
  - In our model we use **soft distillation** as it captures subtleties in data, often improving student model performance and generalization.
  - **The teacher model** (ResNet50) provides logits (Z_t) for the patches.
  - **The student model** generates logits (Z_s) for the patches.
  - **The distillation loss** is calculated by combining the KL divergence between the teacher's and student's logits (with temperature scaling, T = 3.0) and a cross-entropy loss (L_{CE}) on the ground truth labels (y).
  - Where **KL Divergence** measures the difference between two probability distributions, quantifying the "extra" information needed when using one to approximate the other.

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

  - α is the coefficient balancing the Kullback–Leibler divergence loss (KL), ψ is softmax function and τ is the temperature.

  The general soft distillation equation:

$$L_{\text{global}} = (1-\alpha) L_{\text{CE}}(\psi(Z_s), y) + \alpha \tau^2 \mathrm{KL}\left(\psi\left(Z_s/\tau\right), \psi\left(Z_t/\tau\right)\right)$$

  The global loss consists of a weighted sum of the classification loss (LCE) and the distillation loss (KL divergence) between the teacher and student logits.

- **Output**: After processing through the transformer, the [CLS] token's final embedding is passed through an MLP head to output the predicted class probabilities for image classification.

3.4 Impact of Different Tokens

The number and type of tokens passed through the transformer architecture significantly influence model performance.

- **Single [CLS] Token**: When only one [CLS] token is passed, the model performs traditional image classification, where the class information is embedded into this single token that is , the class token is used to predict the actual label (ground truth).
- **Two [CLS] Tokens**: When two [CLS] tokens are passed, both the token handles the class prediction. Though they are initialized randomly and independently, after passing through the final layer, their cosine would almost close around 1 , which would not provide any significant imrpovement.
- **[CLS] and [Distil] Tokens**: When two tokens are passed, the [CLS] token handles the class prediction, while the [Distil] token is used to improve the distillation process. The [Distil] token aids in learning from the teacher model by passing knowledge from the teacher's output during training, thus enabling the student model to perform well even with limited data. Similar to that of the [CLS]-[CLS] token, after passing through the last layer, their cosine goes near 1, but is always lesser than that of [CLS]-[CLS] token.

By including additional tokens like the [Distil] token, DeiT benefits from knowledge transfer, improving generalization and robustness, especially for smaller datasets.

3.5 Data Augmentation

As the name suggest Data Augmentation is simply augmenting the data, in our case augmenting the image. It is used to provide variety of a dataset by applying various transformations like rotations, flips, cropping, color changes, or more complex ones like MixUp and CutMix to the dataset. The final dataset after augmentation should be a diverse version of same data for making the model robust to variation to face the real-world scenarios. We apply Augmentation because the validation loss of DeiT is high.

We use data augmentation to:

- **Prevent Overfitting**: The repeated data with augmentation helps the model to avoid memorizing the training set and generalize it to face real-world scenarios.
- **Increase Dataset Diversity**: It simulates various real-world variations (e.g., lighting conditions, object rotations, occlusions), making the model more adaptable to different inputs.
- **Enhance Model Performance**: With more diverse data, models tend to perform better on unseen data, improving accuracy and robustness.

In our model :

- **Random Crop**:Randomly cropping a portion of the image ensures object robustness despite variations in object position and scale
- **Random Horizontal Flip**:Flipping the image horizontally (with a 50% probability) introduces different object orientations, helping the model generalize across various image flips.
- **Color Jitter**:Simulation of lighting changes in the image by altering its contrast, saturation and brightness.
- **Random Erasing (Cutout)**:Masks portions of the image to simulate occlusion.
- **MixUp**:Combines two images by taking weighted averages of both the images and their labels, increasing dataset diversity and improving model robustness.
- **CutMix**:Similar to MixUp, but instead of blending entire images, it cuts a patch from one image and pastes it into another, creating new examples with different regions from different images.

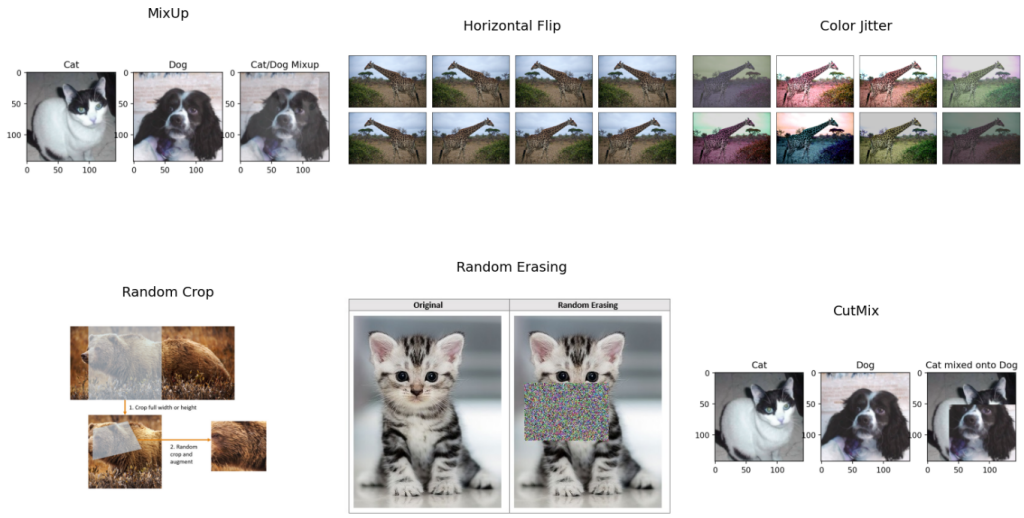

Figure 4: Data Augmentation Examples

3.6 Instructions on how to run the Program

- Required software and libraries: Python, Jupyter Notebook (or any Python IDE), Pytorch, torchmetrics.
- transformer_models.py and transformer_models_deit.py contain the model's required classses and functions.
- pytorch_project.ipynb is a jupyter notebook which has the training and validation of the models. those files are run in the notebook before the rest of the execution.
- We just have to make sure all the files are present in the same working directory.
- Now we can run the program in just jupyter notebook by starting the kernel and running all the cells.

4 Datasets

CIFAR-10 is one the fundamental benchmark dataset available for computer Vision tasks. It was created by Canadian Institute For Advanced Research and has 60,000 color images of size 32x32 pixels. The image have been distributed across 10 classes like cars, birds, cats etc. The data split is - 50,000 training and 10,000 testing , making it suitable for machine learning model's development and evaluations. Its moderate size and complexity position it perfectly between simpler datasets like MNIST and more complex ones like ImageNet, making it a popular choice for research and educational purposes.

Available at **https://www.cs.toronto.edu/ kriz/cifar.html**

5 Results and Discussion

We evaluated ViT, DeiT, and DeiT with augmentation on the CIFAR-10 dataset. The following metrics were used to assess the performance:

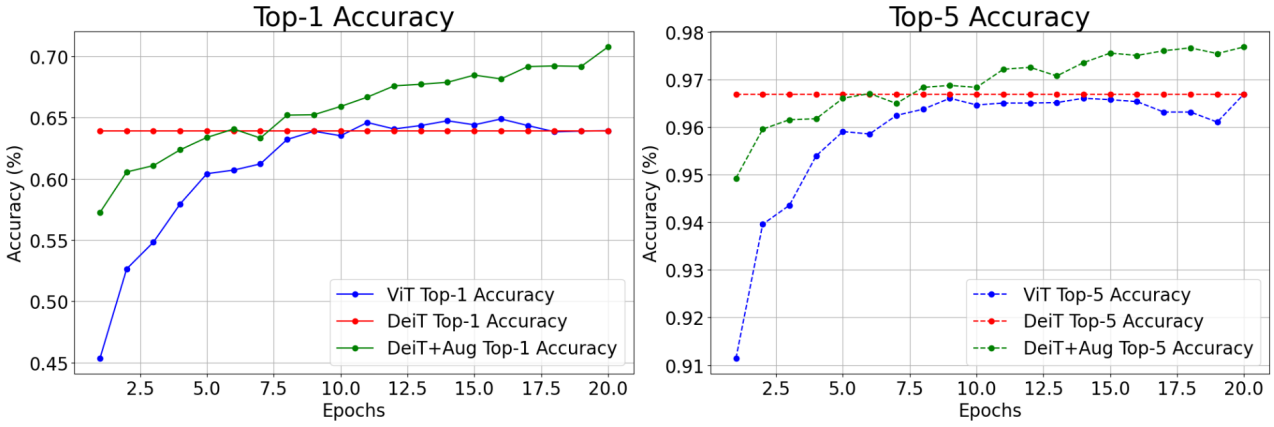

Figure 5: Top 1 Accuracy vs Epochs and Top 5 Accuracy vs Epochs

- **Top-1 Accuracy**:
  - Measures the percentage of times the model's top prediction matches the correct label.
  - The ViT starts with an Top- 1 accuracy of 45.3% and constantly increases to 63.9%, whereas the DeiT remains almost constant at 63.9%. However the DeiT with data augmentation progressively increases to 70.8%, implying its significance on performance and generalization.

- **Top-5 Accuracy**:
  - Indicates the percentage of times the correct label is within the model's top five predictions, useful in multi-class settings with many classes.
  - Unlike Top-1 , Top-5 accuracy of ViT starts with 91.1% and goes up steadily like Top-1 to 96.7%, which is also the Deit's accuracy through out. The Top-5 accuracy of Deit with augmentation shows the best result of 97.7%, implying the significance of augmentation.

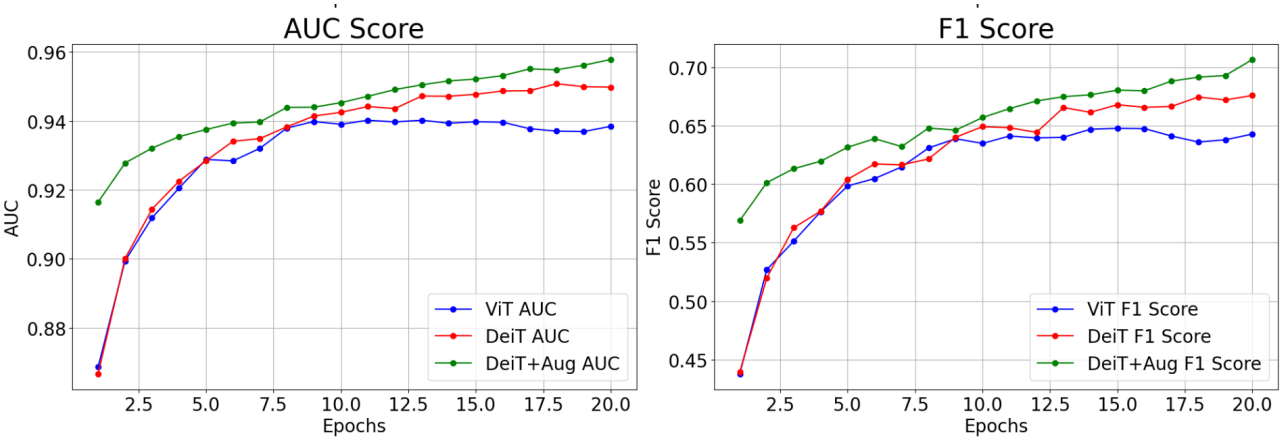

Figure 6: AUC vs Epochs and F1 score vs Epochs

- **AUC**(Area Under the Curve):
  - Summarizes the model's ability to distinguish between classes by plotting true positive rate against false positive rate across thresholds, with higher values indicating better discrimination.
  - Like the accuracy metrics (Top-1 and Top-5), the DeiT with augmentation has the highest value the in the AUC metric as well, around 95.8%, showing better discrimination capability. However, both ViT and DeiT show slower gains, proving data augmentations enhancement in models performance..
- **F1 Score**:
  - Combines precision and recall into a single metric by calculating their harmonic mean, providing a balanced measure of the model's accuracy, especially when classes are imbalanced.
  - The ViT and DeiT have increasing F1 score progression, but the DeiT with augmentation reaches around 70.6% demonstrating its enhanced capability in balancing precision and recall.

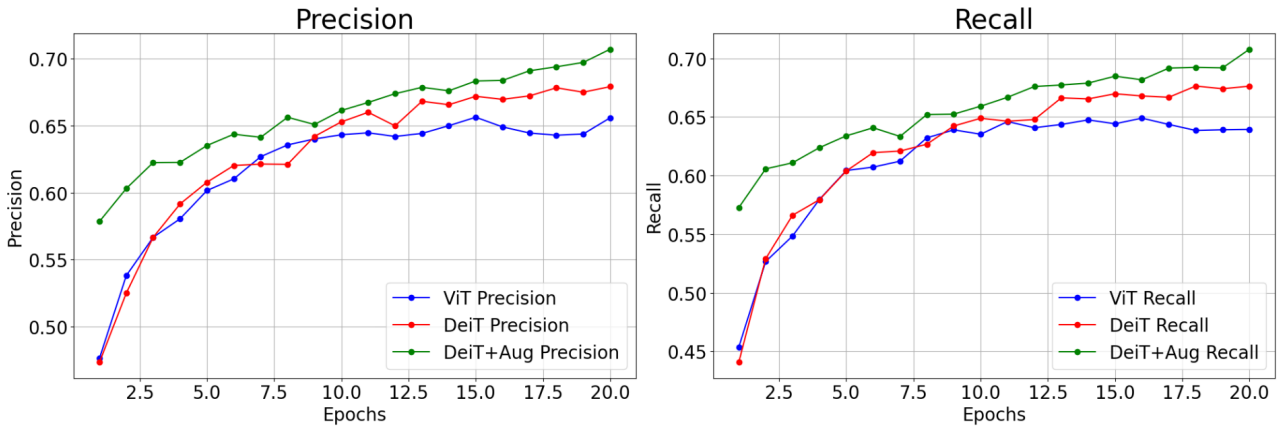

Figure 7: Precision vs Epochs and Recall vs Epochs

- **Precision**:
  - Precision is the calculation of proportion of correctly predicted positive instances and all the instance predicted positive.
  - The DeiT with Augmentation achieves the a precision around 70.7%, where as the ViT and DeiT achieve a precision of 63.9% and 67.6% respectively. This indicates that augmentation increase the accuracy in predicting positive instances.
- **Recall**
  - Recall is again a calculation of proportions but here it is between actual positive instances predicted by the model and its ability to capture all the relevant cases.
  - The ViT's recall increases gradually but gets outperformed by the improvement of the standard Deit.. But DeiT with Augmentation reaches the highest recall at approximately 70.8%, indicating its superior ability to identify actual positive instances.

Table 1 presents the results.

Model	Top-1 Accuracy	Top-5 Accuracy	F1 Score	AUC	Precision	Recall
ViT	63.9%	96.7%	0.642	0.938	0.655	0.639
DeiT	63.9%	96.7%	0.675	0.949	0.679	0.676
DeiT + Augmentation	70.7%	97.6%	0.706	0.957	0.707	0.707

Table 1: Performance comparison of ViT, DeiT, and DeiT with augmentation on CIFAR-10

Our results show that DeiT outperforms ViT in most metrics, with the augmentation-enhanced DeiT providing the best results, demonstrating the effectiveness of advanced data augmentation techniques.

6 Conclusion

The results from the implementation of the Vision Transformer and Data-efficient Image Transformer on the CIFAR-10 dataset show that both the ViT and DeiT achieved identical Top-1 accuracy (63.9%) and Top-5 accuracy (96.7%), suggesting that the additional training strategies employed in DeiT did not significantly enhance these particular metrics over ViT. However, when examining other metrics such as F1 score, AUC, precision, and recall, DeiT consistently outperformed ViT. Notable improvements include an F1 score of 0.675 versus 0.642 and an AUC of 0.949 compared to 0.938, indicating DeiT's more balanced and robust performance, especially in precision and recall, where it scored 0.679 and 0.676 against ViT's 0.655 and 0.639.

Moreover adding augmentation to the data for feeding into DeiT had a remarkable positive impact on all the 6 metrics. Top-1 accuracy rose to 70.7%, while F1 score, AUC, precision, and recall also experienced significant boosts, reaching 0.706, 0.957, 0.707, and 0.707, respectively. These result shows us the importance of data augmentation in enhancing the model generalization having the augmented DeiT outperformed both Deit and ViT.

To conclude, DeiT itself has a better performance than ViT but adding augmentation gives it a significant increase in performance for image classification tasks. The superior results on the CIFAR-10 dataset demonstrate that DeiT, especially with augmentation, is well-suited for real-world applications requiring high accuracy and balanced performance across multiple evaluation metrics.

References

[1] Dosovitskiy, A., et al. (2020). *Image classification with convolutional neural networks*. Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR).

[2] Touvron, H., et al. (2020). *Training data-efficient image transformers & distillation through attention*. arXiv preprint arXiv:2012.12877.

[3] Zhang, H., Cisse, M., Dauphin, Y. N., & Lopez-Paz, D. (2017). mixup: Beyond Empirical Risk Minimization. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1710.09412

[4] Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., & Yoo, Y. (2019). CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1905.04899

[5] Affine. (2021, October 27). Data Augmentation For Deep Learning Algorithms - Affine. Affine. https://affine.ai/data-augmentation-for-deep-learning-algorithms

[6] Anand. (2021, August 4). FastAI – Image Classification – [Chapter 5]. Learn. Share. Repeat. https://aprakash.wordpress.com/2021/07/28/fastai-image-classification-chapter-5

[7] Image augmentation. (n.d.). https://mxnet.apache.org/versions/1.5.0/tutorials/gluon/data_augmentation.html

[8] ResNet-50: https://pytorch.org/ vision/stable/models.html