Implementing BEiT, a BERT-like Transformer model for Image Classification

Lilesh Kurella
University of Illinois Urbana-Champaign
lileshk2

Kavin Krishnasami
University of Illinois Urbana-Champaign
kavinsk2

Abstract

This research paper delves into the application and evaluation of the BEiT [4] (BERT Pre-Training of Image Transformers) model, particularly focusing on its fine-tuning process using the CIFAR-10 and CIFAR-100 datasets [8]. The primary goal is to replicate and analyze the fine-tuning methodology presented in the original BEiT paper, thereby providing a deeper understanding of the model's viability and practical effectiveness in image classification tasks.

The purpose of this project is to comprehensively assess the impact and utility of the BEiT model. The study aims to validate the results reported in the BEiT paper by employing this model on widely recognized and diverse datasets like CIFAR-10 and CIFAR-100 [8]. However, more importantly, this paper looks to explore the model's limitations and strengths in a controlled experimental setting.

Link to our video: https://youtu.be/KOPAX_nJ32o

1. Introduction

In the rapidly evolving field of machine learning, the quest for models that offer not only high accuracy but also versatility and adaptability in various scenarios is a significant endeavor. In the quest to find a more efficient model, Google Researchers released *Attention is All You Need* [9] proposing a transformer architecture to replace recurrent and convolutional neural networks. This paper highlighted the attention mechanism, which helps researchers grasp a greater understanding of long-range dependencies between two different objects. Applying this transformer architecture has given researchers the ability to parallelize training processes, better interpret long-range dependencies, optimize memory, and much more. Transformers took over the NLP sphere and have made their way to computer vision. To utilize the benefits of the transformer architecture, researchers have developed Vision Transformers, one of them being BEiT (BERT Pre-Training of Image Transformers) [4].

1.1. Motivation

Bidirectional Encoder representation from Image Transformers (BEiT) [4] is a trained transformer model that is important to the growth of transformers in computer vision. BEiT takes an innovative approach by making use of the BERT [5] architecture to utilize attention for classification and object detection. Compared to a traditional vision transformer, BEiT (due to the BERT architecture) uses self-supervised learning rather than supervised learning. Creating a self-supervised pre-trained model pushes researchers away from the gigantic heaps of preprocessed data necessary for training neural networks. BERT [5] had success with randomized mask tokenization with encoder recovery. This strategy and many others have not been explored in the field of vision transformers. Utilizing the significant benefits of BERT's architecture, BEiT [4] can bring many positives to the field of computer vision.

The **goal** of this project is to replicate the finetuning process in the BEiT paper [4] to understand the viability and validity of the model.

The purpose of this project is to understand the impact and usefulness of BEiT. This project can help validate the results of the BEiT paper as well as help researchers understand the limitations of the model.

1.2. Background and Related Work

The BEiT paper [4] represents a significant advancement in the field of artificial intelligence, particularly in image processing and understanding. The paper introduces a novel approach to pre-train Vision Transformers, which is inspired by the BERT(Pre-training of Deep Bidirectional Transformers for Language Understanding) [5] model. The core concept of BEiT is Masked Image Modeling, which involves randomly masking out parts of an input image and then training the model to predict the missing pixels. BEiT also utilizes a tokenization scheme inspired by OpenAI's DALL-E, a model that generates images from textual descriptions. BEiT tokenizes the pixels of an image into visual tokens and uses these tokens for pre-training. Adopting this pre-training method allows BEiT to significantly improve the model's understanding of visual content [4]. This leads

to better performance on tasks such as image classification, object detection, and semantic segmentation. The BEiT paper has influenced a significant amount of research in the field of computer vision and AI. Here are some examples:

1. **MAE(Masked Autoencoders) [6]:** Inspired by the success of BEiT, the concept of Masked Autoencoders for Self-supervised Learning in Vision was developed. MAE further refines the idea of masked image modeling by proposing an asymmetric encoder-decoder architecture, where a lightweight decoder is used for reconstruction tasks.

2. **Data2Vec 3.0 [3]:** A unified framework for self-supervised learning, applicable to various methods, including vision, speech, and text.

3. **SimMIM [11]:** This research involves a simple masked image modeling framework for pre-training vision transformers. It simplifies the pre-training pipeline and shows effectiveness in a range of vision tasks.

2. Approach

Our approach to fine-tuning the BEiT model [4] for image classification involved several key steps, incorporating data selection, model adaptation, and training pipeline development. The datasets we used required specific adjustments to the BEiT model initially pre-trained on ImageNet [1].

2.1. DataSet Selection

To understand what a pre-trained BEiT model could achieve, this report delves into testing different and complex image classification objects. The pre-trained BEiT model has been trained on the ImageNet dataset [1]. Datasets such as Tiny ImageNet [10](a dataset that contained one hundred thousand images with two hundred classes created as a subsection of ImageNet), and a couple of others were considered. Although testing the learning and accuracy of the BEiT pre-trained model on a dataset it has already seen would be redundant. Instead, this paper researches the learning validity of the BEiT model on CIFAR-10 and CIFAR-100 datasets [8]. To address the reasoning behind the image classification datasets:

1. **Diversity and Complexity of Images:** CIFAR-10 and CIFAR-100 offer a wide range of images across different classes. CIFAR-10 consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. CIFAR-100 has 100 classes containing 600 images each [8].

2. **Benchmarking and Comparative Analysis:** CIFAR-10 and CIFAR-100 are widely recognized benchmarks in the field of image classification. This is crucial for objectively assessing the advancements BEiT brings to image classification tasks.

3. **Computational Requirements:** Despite being diverse and complex, the CIFAR datasets are relatively small compared to datasets like ImageNet. This makes it feasible for researchers to train and fine-tune models without requiring excessive computational resources.

The selection of CIFAR-10 and CIFAR-100 for this project align with this paper's goals of evaluating the BEiT model's performance in image classification and benchmarking it against established standards in the field. These datasets offer a balance of complexity, manageability, and relevance, making them a great fit for our research.

2.2. Baseline Adaptation

Adapting the BEiT model for both CIFAR datasets was a crucial step in this fine-tuning process.

1. **Model Instantiation and Importing:** The model adapted for this research was the standard huggingface transformers import called BeitForImageClassification. The Configuration object for this model was also imported from the same location, named BeitConfig. This approach utilized the checkpoint `microsoft/beit-base-patch16-224` which provides a BEiT model pre-trained in a self-supervised fashion on ImageNet-21k (14 million images, 21,841 classes) at resolution 224x224, and fine-tuned on ImageNet 2012 (1 million images, 1,000 classes) [7].

2. **Classifier Head Modification:** To accommodate the different number of classes this model would be trained on from the original pre-trained model's features, the classifier head was manipulated. The model classifier was replaced with a fully connected linear layer that took the current model's input features and returned an output the length of the number of classes (depending on which dataset):

```
model.classifier = torch.nn.Linear(
    model.classifier.in_features,
    num_classes).to(device)
```

3. **Computational Considerations:** Furthermore, considering the computational demands of training a sophisticated model like BEiT, this project leverages a CUDA-enabled GPU environment. This decision was driven by the need for efficient computation and faster training times, which are critical in deep learning experiments.

2.3. Training Setup & Preprocessing

There were two ways to gather and utilize the data from the CIFAR datasets, local downloads with scripting as well as loading libraries with large datasets. Given the size of the CIFAR datasets, loading the data from the hugging face library `datasets` was more efficient than adding a step to our Preproccessing pipeline.

1. **Feature Extraction:** After gathering the CIFAR datasets from the hugging face import, the data must be organized and interpret-able for the checkpointed BEiT model [4] itself. Utilizing the `BeitFeatureExtractor` library [7], this pipeline initializes a feature extractor object which helps standardize the dataset by normalizing the pixels, as well as resizing the images into the same size expected by the model (224x224). These basic transformations enhance the model's ability to learn effectively from the dataset. As part of the feature extractor process, the label array from the dataset is also converted to a torch tensor for convenience later on in the training process.

2. **Data Collation:** To best process the data through the actual training pipeline, the transformed datasets are then batched for efficient GPU usage. The training dataset, the validation dataset, and the testing dataset are all processed into DataLoader objects with a batch size of 32 and a designed collate function that separates and creates tensors per batch. The batch size being 32 was due to the CUDA limitations while running the GPU.

2.4. Pipeline

The training setup included Cross-Entropy Loss as the loss function. This choice was based on its effectiveness in classification tasks, particularly in scenarios involving multiple classes, as in CIFAR-10. Additionally, the setup employed the Adam optimizer, known for its efficiency in handling sparse gradients and adaptive learning rate capabilities. This combination of loss function and optimizer was anticipated to facilitate robust learning and convergence of the model during training.

Algorithm 1 gives a basic understanding of the training workflow. The train function takes in the model, cross-entropy loss object (criterion), the AdamOptimizer object, as well as the total number of epochs to be trained. The function then iterates through each epoch, and for every 10 epochs saves the model for a checkpointing standard. Then for each epoch, a running loss variable is initialized. After specifying that the model is training, the pre-processed data is accessed through a DataLoader object. The embedded loop extracts the images and the labels to calculate loss and update the running loss for this particular epoch.

Algorithm 1 Training Pipeline

function TRAIN($model, criterion, optimizer, epochs$)
  for $epoch = 0$ **to** $num\_epochs - 1$ **do**
    if $epoch \mod 10 == 0$ **then**
      Save model
    end if
    $run\_loss \leftarrow 0.0$
    $model.train()$
    Initialize progress bar for $train\_loader$
    for each $(images, labels)$ in $progress\_bar$ **do**
      $optimizer.zero\_grad()$
      Compute outputs using $model$
      $loss \leftarrow criterion(outputs, labels)$
      $run\_loss \leftarrow run\_loss + loss.item()$
      Update progress bar with current loss
    end for
    $epoch\_loss \leftarrow run\_loss/len(train\_loader)$
    Perform validation
  end for
end function

Algorithm 2 Validation

function VALIDATE($model, data\_loader, criterion$)
  $model.eval()$
  $val\_loss \leftarrow 0.0$
  $correct \leftarrow 0.0$
  $total \leftarrow 0.0$
  Initalize progress bar for $data\_loader$
  for each $(images, labels)$ in $progress\_bar$ **do**
    $outputs \leftarrow model(images)$
    $curr\_loss \leftarrow criterion(outputs.logits, labels).item()$
    $val\_loss \leftarrow val\_loss + curr\_loss$
    $predicted \leftarrow torch.max(outputs.logits, 1)[1]$
    Update $correct, total$
  end for
  $val\_loss \leftarrow val\_loss/len(data\_loader)$
  $val\_accuracy \leftarrow correct/total$
  return $val\_loss, val\_accuracy$
end function

At the end of each training epoch iteration, the training pipeline calls Validation specified in Algorithm 2. In this portion of the code, the model is switched to evalua-

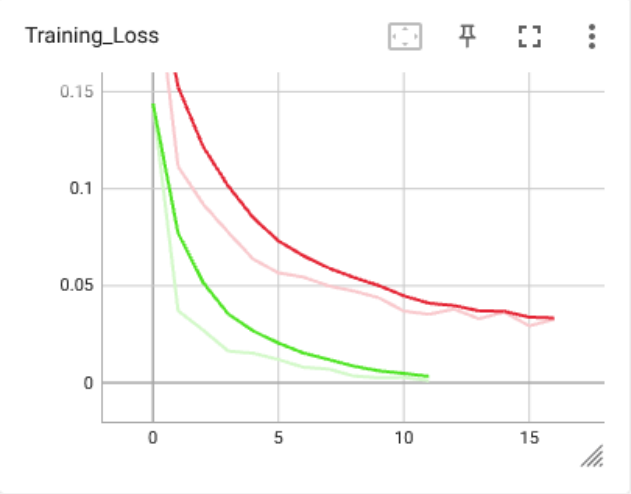

Figure 1. CIFAR10 finetuned models Training Loss over time. The red line represents the baseline model and the green line represents the finetuned model.

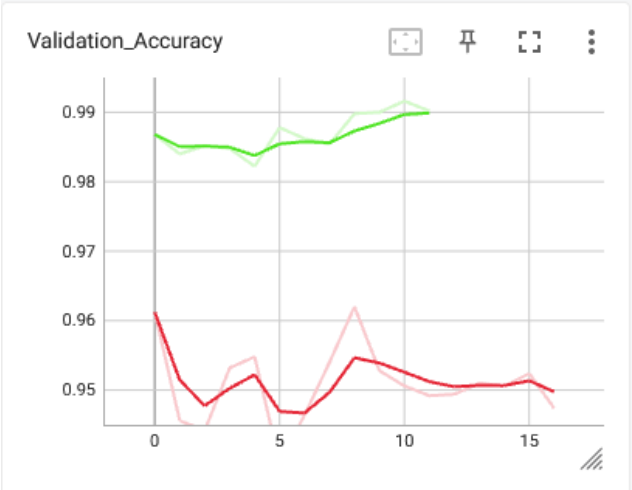

Figure 2. CIFAR10 finetuned models Validation Accuracy over time. The red line represents the baseline model and the green line represents the finetuned model.

tion and calculates the correct values based on labels in the data_loader. Since there is no need to track the gradients during validation, there is also a line in the actual implementation stating `with torch.no_grad()`. Both validation loss and validation accuracy are tracked and returned for further evaluation.

Metrics tracked for baseline testing By utilizing the tensorBoardX [2] python library, the pipeline included a writer variable that kept track of statistics every epoch in order to get a graphical understanding of the model's performance. This writer variable kept track of the Validation Loss, Validation Accuracy, as well as Learning Rate.

2.5. Finetuning Experiments and Strategies

This paper experiments with many strategies and hyperparameters to help determine what is best to interpret the CIFAR datasets. The pipeline saw transformations to the original preprocessed datasets, adding a learning rate scheduler, as well as manually tweaking starting and ending schedulers. There were some limitations to what could be tweaked and what could not given hardware and cost limitations.

3. Results

This section covers the outcomes of the experiments where the BEiT model was applied to the CIFAR-10 and CIFAR-100 datasets for image classification. The evaluation includes both qualitative and quantitative assessments to comprehensively understand the model's performance.

3.1. Experimental Protocols

1. **Evaluation Metrics:** The model's performance was evaluated using accuracy, precision, recall, and F1 score. These metrics provide a comprehensive understanding of the model's strengths and weaknesses in classifying images. The loss function trends, specifically in the cross-entropy loss, were tracked during training and validation phases to monitor the learning efficacy and convergence behavior of the model.

2. **Computational Environment:** Two different GPU's were used for this project. Initially, the graphics card that was used was an NVIDIA GeForce RTX 2070 with Max-Q design. While this GPU was perfectly fine for the initial stages of fine-tuning, it eventually could not handle the tasks necessary. The next GPU that was used was Google Colab's T4. This GPU was better suited to deep learning tasks due to its high computational power and efficiency.

3. **Hyperparameters:** Hyperparameters were carefully chosen based on preliminary trials to ensure the best possible performance of the BEiT model on the CIFAR-10 and CIFAR-100 datasets.

   The learning rate was initially set to 1e-4. After continuous training and testing, the more the learning rate decreased the better the accuracy was returning. The final initial learning rate ended up being 2e-5. To effectively manage the learning rate throughout the training process, a step cosine learning rate schedule was employed with an end learning rate being 1e-6. This approach involves reducing the learning rate following a

Dataset	Test Accuracy
CIFAR10 Baseline Model	0.9528
CIFAR100 Baseline Model	0.8554
CIFAR10 Modified Model	0.9855
CIFAR100 Modified Model	0.9714

Table 1. Testing Accuracy from our baseline models in comparison to the Modified Models due to finetuning

cosine curve, which helps in fine-tuning the model during later stages of training and avoids abrupt changes that could destabilize the learning process.

The batch size was chosen as 32, which provided a balance between computational efficiency and the ability for the model to learn from a sufficiently diverse set of examples in each iteration. The model was trained for 15 epochs, allowing for deep learning without risking overfitting

To enhance the model's ability to generalize and prevent overfitting, two key data augmentation techniques were tested in the training pipeline:

(a) **Gaussian Noise:** This technique helps the model become more robust to variations and noise in real-world data by simulating such conditions during training.

(b) **Random Horizontal Flipping:** This augmentation involved randomly flipping images horizontally. It increased the diversity of the training data and helped in learning more orientation-invariant features

An interesting discovery made was that the transformations had slight to no sway on testing accuracy. The transformations however did have an impact on the validation accuracy. The selection and optimization of these hyperparameters played a crucial role in the performance of the BEiT model on the CIFAR datasets.

3.2. Quantitative Results

The results shown in this section are from the baseline and updated models that were evaluated on around 15 epochs. The reason for this is that the computational resources that were used could only handle this limited number of epochs. Some of the graphs end between 10 to 15 epochs as the GPU used could not handle the computational load to even reach 15 epochs. While this may be a limitation, the graphs still paint a clear picture to compare the baseline and updated models.

1. **Performance Metrics:** Fig. 2 shows the validation accuracy of the baseline model(red) and the updated model(green). The updated model is far more accurate through the first 10-15 epochs. Clearly from the start, there is a significant improvement in image understanding over multiples batches.

2. **Loss Function Analysis:** Fig. 1 shows the train-

Figure 3. CIFAR10 Baseline Confusion Matrix

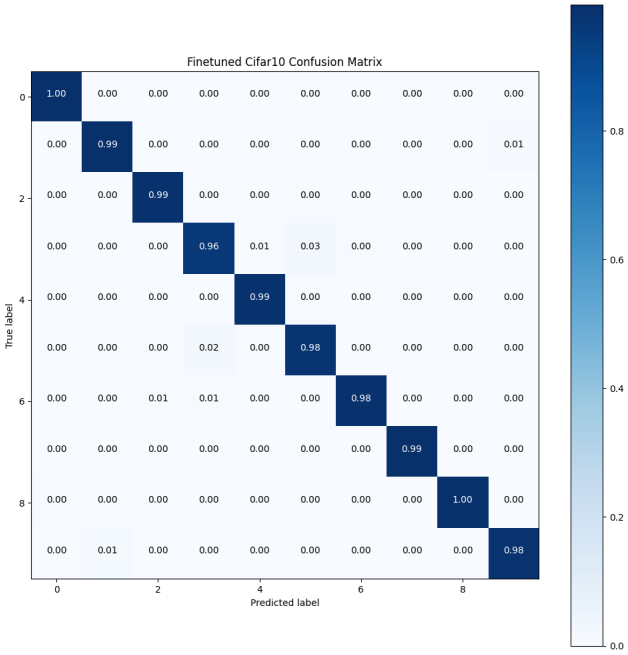

Figure 4. CIFAR10 Finetuned Confusion Matrix

ing loss of the baseline model(red) and the updated model(green). We can see the updated model follows a similar curve but consistently beats the baseline model.

3. **Confusion Matrices:** Fig. 3 and Fig. 4 show the Normalized Confusion Matrix and the Finetuned Confusion Matrix, respectively. The y-axis represents the true label of the image that is being classified and the x-axis is the model's predicted label for the image. The diagonal values are the accuracy in which the model predicts the label for the image and the darker shades of blue represent better accuracy. The fine-tuned matrix shows consistently better performance. There are particular classes in which the finetuned model sees enormous improvements. For example, the predicted label 3 in the baseline confusion matrix is much worse than in the finetuned confusion matrix. Both Confusion matrices have good results but there is a significant improvement in each class being more accurately classified.

4. Discussion and Conclusions

4.1. Summary of Findings

1. **Performance Metrics:** The BEiT model achieved great accuracy on both datasets, with slightly higher performance on CIFAR-10. We could attribute this to the simplicity of the dataset, although the improved fine-tuned model had much better growth in accuracy for CIFAR-100 showing the learning potential of this BEiT modelTab. 1. The model's success on CIFAR emphasizes its capability to handle diverse image categories with a relatively lower level of complexity. On CIFAR-100, while the accuracy was slightly lower than CIFAR-10's, it was still within the competitive range. This indicated the model's potential to deal with more granular classifications.

2. **Fine-Tuning:** The fine-tuning process was critical in adapting the pre-trained BEiT model to the specific characteristics of the CIFAR datasets. The choice of hyperparameters and the implementation of a step cosine learning rate schedule significantly improved the model's performance.

3. **Data Augmentation Impact:** The incorporation of data augmentation techniques, such as Gaussian noise and random horizontal flipping, did not significantly impact the model's performance. Resizing and normalization were important techniques for the BEiT model to understand and properly interpret the images.

4. **Model Limitations:** Despite the promising results, certain limitations were observed. In CIFAR-100, the model sometimes struggled to differentiate between classes with subtle differences. This pointed towards potential areas for improvement in feature discrimination capabilities.

4.2. Final Remarks

The goal of this research project was to understand the use of transformer-based models in image classification, specifically analyzing the viability and validity of the BEiT model. While there are certain limitations, the results of this project indicate that the BEiT model is a great tool for image classification, especially with the fine-tuning process.

Based on these observations, future research could explore modifications to the BEiT model to enhance its ability to distinguish between similar classes. Investigating alternative data augmentation techniques could further improve model performance. Extending this research to larger and more diverse datasets would provide deeper insights into the scalability and adaptability of BEiT in various contexts.

To conclude, this research emphasizes the potential of the BEiT model as a powerful tool in image classification. While it excels in many aspects, there are areas for improvement, providing a roadmap for future advancements in this field.

5. Individual Contributions

1. **Lilesh Kurella:** Model research, Created Training and Evaluation pipeline, Researched and implemented finetuning, Formatted and wrote research paper

2. **Kavin Krishnasami:** Model research, Dataset collection, Research fine-tuning methods, Formatting and writing research paper

References

[1] Imagenet. https://www.image-net.org/. Accessed: [2023]. 2

[2] Tensorboard: Tensorflow's visualization toolkit. https://www.tensorflow.org/tensorboard. 4

[3] Alexei Baevski, Wei-Ning Hsu, Qiantong Xu, Arun Babu, Jiatao Gu, and Michael Auli. data2vec: A general framework for self-supervised learning in speech, vision and language. *CoRR*, abs/2202.03555, 2022. 2

[4] Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*, 2021. 1, 2, 3

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 1

[6] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross B. Girshick. Masked autoencoders are scalable vision learners. *CoRR*, abs/2111.06377, 2021. 2

[7] Hugging Face Team. Beit base patch16 224. https://huggingface.co/microsoft/beit-base-patch16-224, 2021. 2, 3

[8] Alex Krizhevsky and Geoffrey Hinton. Cifar-10 and cifar-100 datasets. https://www.cs.toronto.edu/~kriz/cifar.html, 2009. Accessed: [insert date here]. 1, 2

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1

[10] Jiayu Wu, Qixiang Zhang, and Guoxi Xu. Tiny imagenet challenge. Technical report, Stanford University, 2017. 2

[11] Zhenda Xie, Zheng Zhang, Yue Cao, Yutong Lin, Jianmin Bao, Zhuliang Yao, Qi Dai, and Han Hu. Simmim: A simple framework for masked image modeling, 2022. 2