IMAGE RECOGNITION 2.0:
VISUAL TRANSFORMERS

By Chimaobi Chikezie, Riki Fameli, Ethan Asis, and Isabelle Towle

Github:
https://github.com/ethanga12/dl-final-project

Paper:
Visual Transformers: Token-based Image Representation and Processing for Computer Vision

Introduction

For our project, we implemented a Cornell University paper that created a token-based image recognition model that is more efficient than the traditional CNN model. We felt that creating a transformer model equipped for image classification would be an interesting topic to explore because CNN models regard every pixel in the image to be of equal importance and struggle to relate distant elements in the images, whereas visual transformers instead function in a semantic token space that allows them to intelligently hone in on features found in the image based on context.

Methodology

DISCLAIMER THAT WE DIDNT MAKE IT

Since our project is based on image classification the datasets that we used to train and test our model were CIFAR-10 and CIFAR-100. The architecture for the Visual Transformer we originally found takes advantage of the strengths of traditional convolution networks and combines them with concepts from machine translation. It should be noted however, that due to issues in implementation we opted for a model that is simpler and did not take advantage of convolution at all.

Preprocessing:
Thanks to keras, we were able to prepocess our data with a simple command:

```
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
```

Architecture:
For a given image passed through a full featured VT model, we first apply a couple of convolution layers to extract the low-level features from the image. Following the intuition that a short sentence should be sufficient to describe the high-level features in our images, we use spatial attention to convert our feature maps into sets of semantic tokens. After that, we deploy a transformer to model the relationships between the tokens. Finally, these tokens are used to either directly classify an image, or they can be projected back to the feature map for semantic segmentation.

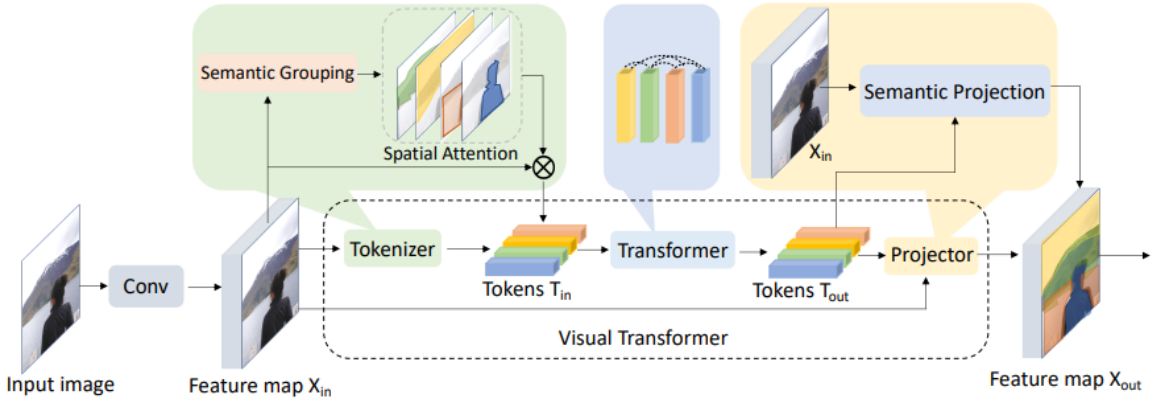

Figure 1: DIagram of Visual Transformer Architecture (Visual Transformers)

Training:
A full visual transformer needs to be trained in a specific way to avoid overfitting. To this end we have to incorporate more training epochs, greater data augmentation, stronger regularization, and distillation. To be more specific, we train the VT-ResNet model for 400 epochs. During that time we start with a learning rate of 0.01, and then over 5 warmup epochs, raise that learning rate to 0.16. After the warmup, we decrease the learning rate by a factor of 0.9875 per epoch. Our batch size is 2048. We must also set a stochastic depth survival probability of 0.9 and a dropout ratio of 0.2

The training recipe that was used on our model was much simpler. We trained our model for only 10 epochs with a constant learning rate of 1e-3 and with a much

smaller batch size of 256. Our model had a dropout ratio of 0.1 but did not make use of a stochastic survival probability.

	From the Paper	Our Model
Epochs	400	10
Dropout Ratio	0.2	0.1
Batch Size	2046	256
Stochastic Survial Prob	0.9	N/A
Learning Rate	Varys (see above)	0.001

Loss:
While the original paper used an negative loss likelihood for their loss function, we opted for the tutorial's version, which was just sparse categorical loss. In code, this meant that our loss was calculated with

```
loss = tf.experimental.nn.losses.negloglik(logits, batch_labels)
```

rather than

```
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

Results
As a benchmark we implemented an image classification model based off the traditonal CNN architecture. Over 10 epochs and with a batch size of 256, the CNN model concluded training with a test accuracy of 69.37%. We were unable to implement the initial VT model that was based on a PyTorch implementation. For the sake of being able to compare results with a CNN model, we created a new VT implementation based off of a Tensorflow Keras tutorial and ended up with a model with an accuracy of 70.45% over 10 epochs (and the same batch size of 256).

Our results indicate that our VT model implementation was basically equivalent in performance to the CNN, despite significantly longer runtimes.

Our CNN model's accuracy was 70%, so our goal for our VT model was to surpass that. Unfortunately, the VT model that we implemented failed to run due to shape errors. For the sake of being able to compare our CNN with a VT model, we based a new implementation off the tensorflow tutorial for VT models and ended up with a model that boasted an accuracy of 70.45%.

Challenges:

In the process of completing this project and building our model we ran into a number of difficulties. First, when we were implementing our benchmark CNN model using keras, for some reason our model was not learning and had its accuracy stuck at around 10% on the CIFAR-10 dataset. Our visualization was showing that the model was guessing the same label over and over. To remedy this, we based our CNN off of the tensorflow keras tutorial for a CNN as opposed to attempting to translate the CNN we wrote in homework 2 into a keras implementation. The next issue that we ran into was understanding the concepts from the paper, specifically how the things we learned during the NLP unit were being used in the context of images instead of words. To overcome this issue we just had to read the paper more closely and compare what it was saying to our notes and the slides from class. The final issue that we ran into was translating the pytorch implementation that we had originally found for the VT model into tensorflow. This problem was two pronged. First, we struggled with actually writing a VT model in tensorflow following the structure of the pytorch implementation we found. This was due to the fact that tensorflow and pytorch oftentimes have different parameters for the same methods so trying to translate these things was quite confusing and made us have to look very closely at the documentation for both frameworks to try and make sense of what was going on. The second issue arose after we finally put together what we thought was a tensorflow translation of the pytorch implementation. It simply would not run despite our best efforts. No matter what we changed, we kept on getting different shape errors stemming from different parts of the code, so eventually we cut our losses and pivoted to basing our VT model off of the official tensorflow tutorial.

Reflection:

While we are proud of our efforts, we believe it's safe to say that we were unable to meet our target goal, though one could argue we did reach our base goal, which was to have running code. Despite this, we were all able to learn a lot and enjoyed delving into the industry standard for image classification's rising competitor. In trying to implement our model from pytorch, our group also began to understand the differences between pytorch and tensorflow. Unfortunately, we learned some of the larger differences once we began to debug the model, and realized that pytorch handles shapes in a way we were not anticipating after a semester of tensorflow. If we were to have more time, we would try to combine the keras implementation and the repo implementation together to take the strengths from both and hybridize them into their own unique model. In order to do this, we were planning on starting with comparing the patches and blocks classes to help with tokenization and shape issues, as this seemed to be where many of our issues were coming from when comparing models. What was most interesting to our group overall was the use of transformers for computer vision, something that we had previously focused on with regards to machine translation. This seemingly unintuitive implementation of a deep learning strategy has left us excited to see what other ways we may be able to cross reference designs to further deep learning models. It should also be noted that our VT took significantly longer to train, and its environmental impact when compared with CNNs was much greater, with little to no tangible gain. If we were to continue to develop this model, we would also focus heavily on balancing the run time and accuracy of the model to truly balance out it's energy use.