Transfer Learning Techniques on Tiny-ImageNet: Final Report

Soham De

July 2020

Abstract

The nature of the scaled downed aspect of the Tiny ImageNet [1] dataset makes Transfer Learning an ideal approach to classify the dataset. This paper reports the progress of experiments conducted with various Transfer Learning Methods on the Tiny Imagenet, using state-of-the-art models pre-trained on the original ImageNet [2] dataset. This report lays a brief background of the dataset, possible approaches and common issues one is expected to face while classifying this dataset. It also presents the results of the use of certain Transfer Learning approaches in classifying the images in this dataset.

1 Introduction

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [2], started in 2010, has become the standard benchmark for testing the performance of state-of-the-art CNN-based Image Classification architectures. The Tiny ImageNet [1] is a scaled-down version of the original ImageNet dataset, consisting of 200 image classes, a training dataset of 90,000 images, a validation dataset of 10,000 images, and a test dataset of 10,000 images. All images are of size (64,64), which allows for shorter training times while still maintaining a reasonable level of difficulty.

The scaled down nature of the Tiny ImageNet dataset makes it an interesting target for transfer learning. We already have highly accurate CNN-based Classifiers trained to produce over 95% (Top-5) test accuracy [3] [4] on the full ImageNet dataset. Direct application of these models on the Tiny ImageNet will of course produce sub-par results, due to the reduction in image resolution and the number of classes. Thus, I try different methods of Transfer Learning, such as Fine-Tuning, Layer Reduction and Feature Extraction to analyse how they perform on the Tiny ImageNet dataset. I also suggest measures which are very likely to improve the current results and report persistent issues that have appeared in the process. Using the best experimental combination of CNN architecture, data augmentation, hyper-parameters and size of input tensors, I have achieved a **training accuracy of over 80%, while validation and test accuracy plateaus at around 75%. The test accuracy of my final submission on Kaggle was 74.8%.**

2 Hardware Constraints

Without access to a GPU, I was limited to free online GPU accelerators on Kaggle, [5] which have a weekly usage limit of 30 hours and a meagre 9 hours limit on continuous sessions. Thus, training a full deep CNN for large epochs wasn't feasible. This was the main reason why I decided to adopt a Transfer Learning approach to start with (if time and resources permit, I shall explore training models from scratch as well).

At a later stage, I will try to utilize the very recent TPU acceleration feature provided by Kaggle and hopefully decrease training time. It may be noted, that the Input Pipeline I am using (ImageDataGenerator) isn't currently supported for TPU usage, and thus, it will have to be modified. I tried to load the entire dataset into the session memory, but it exceeded limit. Therefore, all the approaches involved in this report rely only on GPU acceleration (and not TPU support).

3 Feasible Approaches

My biggest challenge was low computational power, as Kaggle allows a maximum continuous run-time of 9 hours, and I wanted to create a model using freely available online resources only. I also did not use vanilla versions of any pre-trained models for test predictions, as that would be unfair.

My ideal approach was to leverage models pre-trained on the full ImageNet dataset to produce a 1-D vector from an input image, which represents its features. Then, using a relatively smaller Dense Neural Network,

I use these feature vectors as inputs and create a classifier over 200 classes. To summarize, I experimented with the following:

- Random Guessing
- Single Layer NN
- Naive CNN
- Using pre-trained CNNs (XceptionNet, ResNet, VGGNet) as Feature Extractors
- Fine-tuned pre-trained weights for incremental increase in accuracy

The first 3 approaches were mostly to set naive benchmarks, in order to see an improvement with increase in complexity of the model. The results of these aren't reported and was mostly done to check the integrity of the code.

The final Fine-Tuning is an experiment which must be performed with extreme caution with very small learning rates. Larger learning rates is very likely to cause the Model to over-fit and end up nullifying all the benefits of pre-trained convolution layers. I have been successful in Fine-Tuning the XceptionNet, but applying similar methods on the ResNet resulted in immediate over-fitting and loss of accuracy.

4 Methods

4.1 Choice of CNN architecture

I experimented with 3 different CNN architectures - VGGNet-16 [6], ResNet-50 [4] and XceptionNet [3]. Each of these were pre-trained on the ImageNet dataset. I replaced the last layer in each of these models with an average pooling layer, 40% dropout and a dense 'softmax'-activated classifier with 200 neurons. For the size of the input tensor, I have experimented with the sizes (80,80), (100,100), (128,128) and (150,150). There seems to be a steady increase in accuracy with increase in input-scaling.

XceptionNet provides the highest accuracy after 10 epochs (refer to Results), which is why I decided to proceed with that architecture and fine-tune it end-to-end. All layers were unfrozen and the whole model was trained with a lower learning rate of 1e-05, using the Adam optimizer.

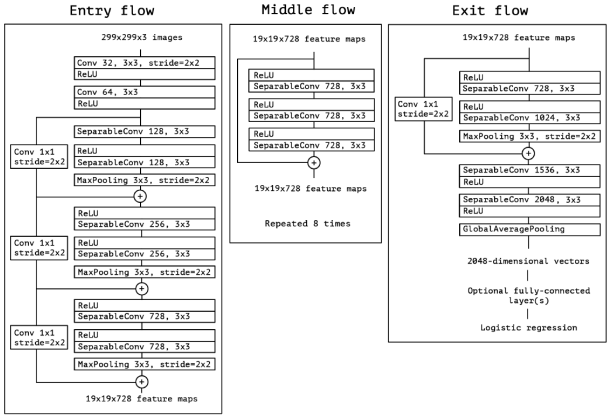


Figure 1: The Xception Net Architecture

4.2 Choice of Optimizer

Used Adam for Xception (lr=1e-03 for training, 1e-05 for fine-tuning) and VGGNet (lr=1e-03). However, Stochastic Gradient Descent for ResNet (nesterov momentum = 0.9, lr=0.01, decay= 1e-06) was used as it resulted in faster convergence. The choice of momentum was influenced by standard practice and wasn't tuned.

4.3 Regularisation and Data Augmentation

In the pre-processing, data augmentation has also been used to 'fool' the network in believing that there is more training data. The main augmentation techniques used were horizontal flip, rotation, shear and brightness adjustments. An ablation study will be conducted further on. The choice of augmentations did not involve change in hues or saturation as they often do not yield significant benefit. [7].

5 Results

Results have been promising throughout. Using the best experimental combination of data augmentation, hyperparameters and size of input tensors, I have achieved a **training accuracy of** $> 80\%$**, while validation and test accuracy plateaus at around** 75%**.** This may certainly be improved with further experimentation, some ideas of which are stated in the next section.

The first round of experimentation was done by simply removing the last layer from a pre-trained XceptionNet, adding an average pooling layer and a 200 node dense classifier. This resulted in a training accuracy of

60.8% and validation accuracy of 54.8% after only 10 epochs.

As the previous attempt seemed to over-fit very quickly on training data, I decided to add 40% dropout layer aon top of the Xception-based network. This resulted in a significantly higher validation accuracy of 58.2% after 10 epochs (and reduced over-fitting on training data).

I ran the same process using 3 different CNN architectures [3] [4] [6] and noted the results in Table 1

CNN	Train_Acc	Val_Acc
VGGNet-16	0.4488	0.4879
ResNet-50	0.4820	0.5677
XceptionNet	0.6111	0.5821

Table 1: Results of training different CNN architectures after 10 epochs, with the last layer replaced by a 40% dropout and a dense softmax classifier

I decided to therefore proceed with the ResNet and XceptionNet-based architectures for further training. The results of these experiments are noted in Table 2. ResNet took much longer to converge with an Adam optimizer, which was improved by the use of SGD with Nesterov momentum. However, after convergence, the ResNet over-fitted rapidly on fine-tuning. Thus, I decided to proceed with a fine-tuned XceptionNet-based architecture as the final model (refer to Table 3). On training it for 30 epochs, it gave a training accuracy of 85.8% and validation accuracy of 74.5%. Thus, to reduce this over-fitting, data augmentation was increased, and 2 dense layers were added, with increased dropouts (65% and 25%). The resulting model gave a 80.3% training accuracy and 75.7% validation accuracy after 30 epochs. While the improvement was only incremental, it was successful in reducing variance (over-fitting).

CNN	Epochs	Train_Acc	Val_Acc
ResNet	10	0.4820	0.5677
ResNet	15	0.5108	0.5842
ResNet	20	0.6117	0.6231
XceptionNet	10	0.6111	0.5821

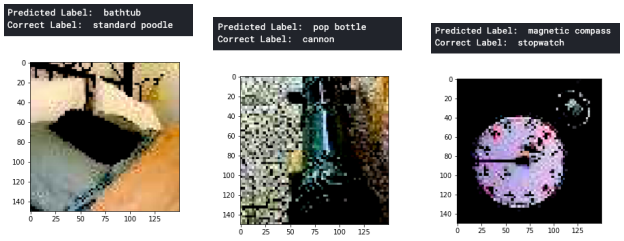
Table 2: Comparative performance analysis of ResNet and XceptionNet on training with and Average Pooling Layer, 40% dropout and a dense (200 node) classifier

Dropout (%)	Epochs	Train_Acc	Val_Acc
40	18	0.7876	0.7326
40	30	0.8578	0.7446
60,25*	30*	0.8031*	0.7568*

Table 3: Comparative performance analysis of ResNet and XceptionNet on training with and Average Pooling Layer, 40% dropout and a dense (200 node) classifier

5.1 Error Analysis

I randomly sampled 1000 images from the validation dataset and used my final Xception model to predict on them. Using these predictions, I performed an error analysis by comparing them with the provided labels. Classes with most inaccurate predictions were 'scorpion', 'miniskirt' and 'mashed potato'. I also noticed that the resolution was often too low for even humans to reasonably predict the subject-matter of the image, which can be seen in Figure 2.



(a) Standard Poodle (b) Cannon (c) Stopwatch

Figure 2: Examples of incorrectly classified images by the final model. We see that the labels are ambiguous (stopwatch vs metal compass) sometimes and the images are of a very low resolution which makes it impossible for even a human to classify some of them.

While there is no feasible way to measure Bayes' Accuracy for this particular dataset, from the error analysis, it is safe to assume that the final model is reasonably close to (perhaps even better than) human-level performance. Thus, it made more sense for me to prevent over-fitting than increasing the number of layers or training further. In terms of accuracy, this meant a greater focus on reducing the difference between training and validation accuracy, than on increasing training accuracy as much as possible. While increasing dropout and data augmentation certainly helped, further regularisation techniques may be desired.

6 Further Improvements

One persistent issue that I have faced is the under-utilisation of the GPU on the Kaggle kernels. On average, I am achieving around 40-60% GPU utilisation using Keras libraries. My suspicion is that the input pipeline using the Data Generators (flowing directly from directory) applies the data augmentation on the CPU, causing a bottleneck. However, there have been instances of the GPU utilisation metric showing inaccurate results on Kaggle, so it remains unclear as to what process was causing the bottleneck. In either case, there was a massive reduction in training time while using GPUs over CPUs on Kaggle.

Without access to more data, I will have to resort to better data augmentation and regularisation techniques (such as L1, L2 or introducing some dropout) to further reduce variance (difference in training and validation loss). While the CNN architectures I have used are among the best performing on the ImageNet dataset, sometimes, other neural network architectures may give better results, especially given the lower resolution of the Tiny ImageNet dataset. Thus, one may also experiment with DenseNet or even MobileNet architectures.

There are also certain methods such as Multi-Crop during test time and Ensembling the outputs from multiple trained model, which may help increase test accuracy by a small margin. However, these techniques are only limited to use in online competitions and are seldom used in real deployable products.

References

[1] Li Fei-Fei & Krishna Ranjay (Stanford University CS231n). *Tiny ImageNet Visual Recognition Challenge*. URL: `https://tiny-imagenet.herokuapp.com/`.

[2] *ImageNet Large Scale Visual Recognition Challenge*. URL: `http://www.image-net.org/`.

[3] Francois Chollet. "Xception: Deep Learning with Depthwise Separable Convolutions". In: (). URL: `https://arxiv.org/pdf/1610.02357.pdf`.

[4] Kaiming He ; Xiangyu Zhang ; Shaoqing Ren ; Jian Sun. "Deep Residual Learning for Image Recognition". In: *IEEE* (). URL: `https://ieeexplore.ieee.org/document/778045`.

[5] *Kaggle - Your Machine Learning and Data Science Community*. URL: `www.kaggle.com/`.

[6] Shuying Liu ; Weihong Deng. "Very deep convolutional neural network based image classification using small training sample size". In: *IEEE* (). URL: `https://ieeexplore.ieee.org/document/748659`.

[7] Zoheb Abai & Nishad Rajmalwar. "DenseNet Models for Tiny ImageNet Classification". In: (). URL: `https://arxiv.org/pdf/1904.10429.pdf`.