Tiny ImageNet Classification using Deep Learning

Vishal Basutkar
basutkar.n@northeastern.edu

Abstract

In this project, I performed Image Classification on the Tiny ImageNet dataset using Deep Convolutional Neural Networks and developed and compared a few models. I employed TensorFlow 2.9.0 in Python to develop the models. I first tested the image classification with a simple one convolution layer model with ReLU activation, Adam optimizer and achieved an accuracy of just 22% on the validation data. I then expanded this to two layer model with L2 regularization and achieved a slightly better accuracy of 25%. However, these were to get an abstract idea of the dataset to know where to proceed. I then created two deep CNN models, first with 8 convolution layers and I could achieve around 40% accuracy. The second model achieved around 43% accuracy.The reason for the lower accuracies is the downsampling of the ImageNet dataset to tiny ImageNet that is just 64X64X3. The models would learn features better on the ImageNet dataset and give a much better accuracy. I then tested with a few pre-trained models like ResNet50 and ResNet101 with again gave just around 55% accuracies each.

Introduction

This project focuses on the task of image classification using deep convolutional neural networks [I]. The objective is to develop models capable of accurately classifying images from the Tiny ImageNet dataset. The project employs Tensorflow 2 in Python to develop the models and includes an evaluation of various deep learning architectures.

Image classification is a fundamental problem in computer vision and has numerous practical applications, such as image recognition, object detection, and autonomous driving. Accurate image classification is essential in these applications for recognizing objects in images and providing an appropriate response. Therefore, improving image classification accuracy is a significant research area in computer vision.

The approach taken in this project is to develop deep convolutional neural network models for image classification on the Tiny ImageNet dataset. The models are designed to learn features from the images using multiple convolutional layers, and the accuracy of the models is improved through regularization techniques. The project includes an evaluation of different model architectures to determine which models perform best on the Tiny ImageNet dataset.

The Tiny ImageNet dataset is used for experiments and evaluation in this project. This dataset is a subset of the ImageNet dataset and contains 200 classes with 500 training images and 50 validation images per class. The images in the dataset are resized to 64x64 pixels with 3 channels (RGB). The dataset is challenging due to its small size and low resolution, making it difficult for models to learn the features necessary for accurate image classification.

In summary, this project aims to develop deep convolutional neural network models for image classification on the Tiny ImageNet dataset. The evaluation of various deep learning architectures will provide insight into the most effective approaches for image classification on this challenging dataset.

Background

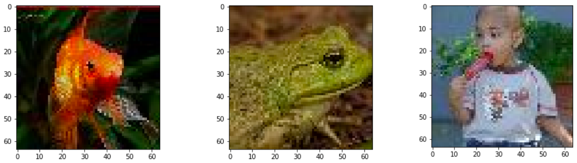
The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [II] is a competition that evaluates algorithms for object detection and image classification on a large-scale dataset of images called ImageNet. The competition was first held in 2010, and has since become a benchmark for image classification algorithms.

The Tiny ImageNet dataset is a subset of the ImageNet dataset, which contains 200 classes with 500 training and 50 validation images per class, as well as 50,000 testing images. In comparison, the full ImageNet dataset contains over 14 million images and more than 21,000 categories, making it a challenging dataset to work with due to its large size and complexity. The Tiny ImageNet dataset was created to provide a smaller and more manageable version of the dataset for researchers to use in developing and testing image classification algorithms.

As of April 2023, the DeiT-B/16-D + OCD [III] model holds the state-of-the-art performance on Tiny ImageNet classification, achieving an impressive accuracy of 92%. This model employs a conditional diffusion methodology to achieve its high accuracy. However, for this project, we will focus on utilizing deep Convolutional Neural Networks to achieve the highest possible accuracy on the Tiny ImageNet dataset.

Approaches

The training dataset is basically a folder with 200 subfolders that represent the 200 classes, each having 500 images of the respective classes. Below are the sample images of the 'Fish', 'Frog', and 'Human eating ice cream' categories.



The validation data images are stored all in one folder and a text file is given with annotations describing the image about its class and object locations. I wrote a python script that read through the file and created separate folders for each class, segregated the images and saved them in their respective folders.

I performed the image classification of the Tiny ImageNet using 5 different convolutional neural network models as described below. The models were trained on my MacBook Pro M1 with 14 core GPU and 16GB RAM.

Model 1

This is the baseline and the simplistic model with just 1 convolution layer that includes 32 filters, 1 max pool layer and 2 Dense layers of 1024 and 200 outputs respectively. ReLU activations were employed and Adam optimizer with learning rate of 0.01. The model was trained for 15 epochs that gave a training accuracy of 36.85% and 21.25% of validation accuracy. Below is the model summary.

Layer (type)	Output Shape	Param #
Input	64, 64, 3	
Conv2D	(None, 62, 62, 32)	896
MaxPooling2D	(None, 31, 31, 32)	0
Flatten	(None, 30752)	0
Dense	(None, 1024)	31490048
Dense	(None, 200)	205000

Fig 1. Model 1 summary

Model 2

This is just one layer more than the first model to show that adding more layers can make the model learn more parameters and therefore perform better and give a better accuracy. The second layer also adds in L2 regularization so that the model would not get affected by Vanishing Gradient problem [IV]. The model was trained for 15 epochs that gave a training accuracy of 36.26% and 25.65% of validation accuracy. Below is the model summary.

Layer (type)	**Output Shape**	**Param #**
Input	64, 64, 3	
conv2d_1 (Conv2D)	(None, 62, 62, 32)	896
max_pooling2d_1 (MaxPooling2D)	(None, 31, 31, 32)	0
conv2d_2 (Conv2D)	(None, 29, 29, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 64)	0
flatten_1 (Flatten)	(None, 12544)	0
dense_1 (Dense)	(None, 1024)	12846080
dense_2 (Dense)	(None, 200)	205000

Fig 2. Model 2 summary

Model 3

I created my first deep learning model from scratch, dividing it into five blocks to enhance comprehensibility. The model employed ReLU activation throughout, except for the last Dense layer that utilized softmax activation. I maintained 'same' padding throughout while incorporating strides in each block.

The first block comprised of two convolutional layers, each with 64 filters, followed by batch normalization for each layer, and a max pooling layer with a stride of 2X2. The next four blocks were structured similarly, but with different filter sizes. Blocks 2, 3, and 4 had 128 filters each, while Block 5 had 256 filters in the first convolutional layer and 512 filters in the second one.

Upon flattening, three dense layers were added with 4096, 1024, and 200 outputs, respectively. Batch normalization was applied after the first two dense layers.

I trained the model for 15 epochs, achieving a training accuracy of 84.26% and validation accuracy of 39.54%. Here is the summary of the model:

Layer (type)	Output Shape	Param #
conv2d_10 (Conv2D)	(None, 64, 64, 64)	1792
batch_normalization_12	(None, 64, 64, 64)	256
conv2d_11 (Conv2D)	(None, 64, 64, 64)	36928
batch_normalization_13	(None, 64, 64, 64)	256
max_pooling2d_5	(None, 32, 32, 64)	0
activation_8	(None, 32, 32, 64)	0
conv2d_12 (Conv2D)	(None, 32, 32, 128)	73856
batch_normalization_14	(None, 32, 32, 128)	512
conv2d_13 (Conv2D)	(None, 32, 32, 128)	147584
batch_normalization_15	(None, 32, 32, 128)	512
max_pooling2d_6	(None, 16, 16, 128)	0
activation_9	(None, 16, 16, 128)	0
conv2d_14 (Conv2D)	(None, 16, 16, 128)	147584
batch_normalization_16	(None, 16, 16, 128)	512
conv2d_15 (Conv2D)	(None, 16, 16, 128)	147584
batch_normalization_17	(None, 16, 16, 128)	512
activation_10	(None, 16, 16, 128)	0
max_pooling2d_7	(None, 8, 8, 128)	0
conv2d_16 (Conv2D)	(None, 8, 8, 128)	147584
batch_normalization_18	(None, 8, 8, 128)	512
conv2d_17 (Conv2D)	(None, 8, 8, 128)	147584
batch_normalization_19	(None, 8, 8, 128)	512
activation_11	(None, 8, 8, 128)	0
max_pooling2d_8	(None, 4, 4, 128)	0
conv2d_18 (Conv2D)	(None, 4, 4, 256)	295168
batch_normalization_20	(None, 4, 4, 256)	1024
conv2d_19 (Conv2D)	(None, 4, 4, 512)	1180160
batch_normalization_21 (BatchNormalization)	(None, 4, 4, 512)	2048
activation_12 (Activation)	(None, 4, 4, 512)	0
max_pooling2d_9 (MaxPooling2D)	(None, 2, 2, 512)	0
flatten_1 (Flatten)	(None, 2048)	0
dense_3 (Dense)	(None, 4096)	8392704
batch_normalization_22 (BatchNormalization)	(None, 4096)	16384
activation_13 (Activation)	(None, 4096)	0
dense_4 (Dense)	(None, 1024)	4195328
batch_normalization_23 (BatchNormalization)	(None, 1024)	4096
activation_14 (Activation)	(None, 1024)	0
dense_5 (Dense)	(None, 200)	205000
activation_15 (Activation)	(None, 200)	0

Fig 3. Model 3 summary

Model 4

I have designed a new deep learning model from scratch, taking inspiration from the reduced AlexNet architecture. While building this model, I added an extra Dense layer with a dropout rate of 0.5. Throughout the model, I have used ReLU activation functions except for the last layer, where I have used softmax activation.

This new model proved to be a bit more effective, and it did not display any overfitting tendencies, unlike my previous models. This allowed me to train it for 25 epochs using the same learning rate. Consequently, I achieved better training and validation accuracies of 59% and 42.4%, respectively. Below is the model summary.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 64)	1792
conv2d_1 (Conv2D)	(None, 64, 64, 64)	36928
max_pooling2d (MaxPooling2D)	(None, 32, 32, 64)	0
batch_normalization (BatchNormalization)	(None, 32, 32, 64)	256
conv2d_2 (Conv2D)	(None, 32, 32, 128)	73856
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 128)	0
batch_normalization_1 (BatchNormalization)	(None, 16, 16, 128)	512
conv2d_3 (Conv2D)	(None, 16, 16, 256)	295168
conv2d_4 (Conv2D)	(None, 16, 16, 256)	590080
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 256)	0
flatten (Flatten)	(None, 16384)	0
dense (Dense)	(None, 1024)	16778240
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 512)	524800
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 200)	102600

Fig 4. Model 4 Summary

Model 5 and 6

To test the performance of pre-trained models ResNet50 and ResNet101 [V], I used their pre-trained weights from ImageNet and built two models. I added one max pooling layer and one dense layer with 200 outputs after the pre-trained weights to fine-tune the models for my specific classification task.

To train the models, I compiled them using the Adam optimizer with a learning rate of 0.0001. After training the models, I obtained similar results for both of them. The training accuracy was 87.9%, indicating that the models were able to learn the classification task effectively. The validation accuracy was 55.12%, which shows that the models were able to generalize well to new data.

Overall, using pre-trained models can be an effective way to improve the performance of deep learning models, as they already have a strong foundation of knowledge from pre-training on large datasets like ImageNet. Fine-tuning these pre-trained models can save time and resources while achieving good results on specific tasks.

Results

I have tabulated the training accuracies and validation accuracies of all the models in Fig 5.

Model	Training Accuracy	Validation Accuracy
1	40.84%	22.07%
2	34.94%	25.40%
3	84.26%	39.54%
4	60.89%	42.40%
5	85.86%	55.13%
6	87.94%	55.12%

Fig 5. Model Accuracies

I have also created a line graph of the models' training and validation accuracies improving over each 15 epochs in Fig 6 and Fig 7. Since all models that were employed had the same learning rate and optimizer, it seemed right to compare the models over each epoch as done in the line graphs.

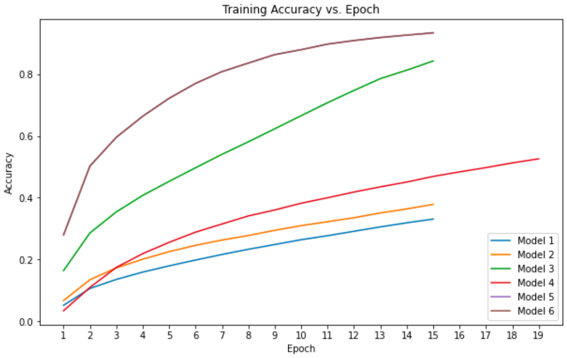


Fig 6. Training accuracy vs Epoch

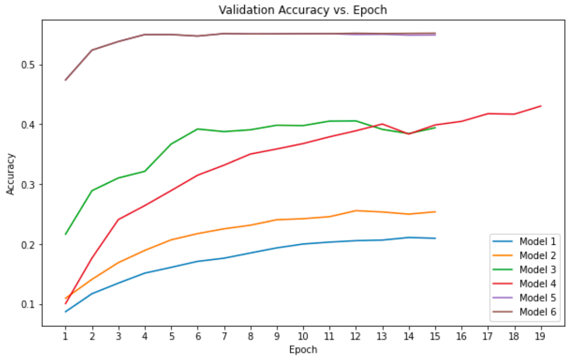


Fig 7. Validation accuracy vs Epoch

I trained all the models for 15 epochs except for Model 4, which took 20 epochs to converge. Based on the validation accuracy, it was observed that all the models had converged by 15 epochs. However, Model 4 took slightly longer to converge. The training and validation accuracy plots in Fig 6 and 7 show that the models overfit the dataset to some extent. Despite this, achieving an average validation accuracy of 50% on the small images in the 200 classes dataset is considered to be a good accuracy.

Fig 8 and Fig 9 are the line graphs of training and validation losses over epoch and we can see abc is doing better.

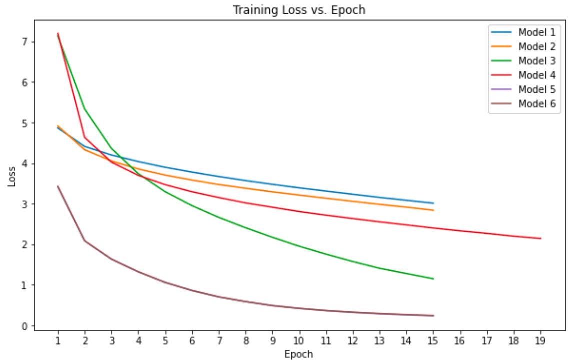


Fig 8. Training Loss vs Epoch

I have also tabulated the number of parameters and the training time each model in Fig 10.

Upon analyzing the table in Figure 10, it can be seen that the ResNet50 model generated a large number of parameters and achieved higher accuracy with shorter training time. ResNet101 though gave the same accuracy, created a large number of parameters and took much longer that the other models. However, Model 4, which had a slightly higher number of parameters than Model 3, had significantly shorter training time and achieved a better accuracy. Based on these results, it can be concluded that Model 4, which was developed from scratch, outperforms the first three models in terms of both training time and validation accuracies.

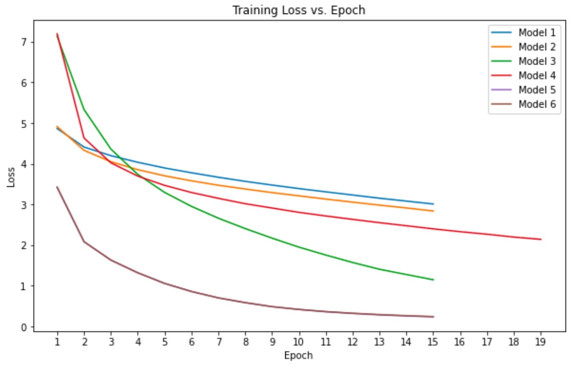


Fig 9. Validation Loss vs Epoch

Model	Number of Parameters	Training Time
1	31,696,968	9m 12s 100ms
2	13,070,472	7m 58s 810ms
3	15,145,992	50m 54s
4	18,404,232	37m 26s 420ms
5	25,890,888	55m 20s 50ms
6	44,961,352	1h 31m 3s 400ms

I experimented with various data augmentation techniques in the Image Data Generator [VI], including shear range and zoom range, but found that they decreased accuracy and increased the number of parameters. One reason for this could be the small size of the images, making it difficult to apply zoom range effectively. As a result, I rescaled the pixels of the images to normalize the dataset and set horizontal and vertical flip as true for both the training and validation datasets. During the training process, this data augmentation feature randomly flipped images horizontally and vertically. This technique helps the model learn features that are invariant to flipping, improving its ability to generalize to new images. Additionally, flipping increases the effective size of the training dataset by creating new, slightly different images from each original image. Ultimately, I found that using Adam optimizer and ReLU activations with this data augmentation strategy yielded the best results.

During the experimentation process, the performance of the neural network was evaluated with various combinations of optimizers and activation functions in each layer. Optimizers such as RMSProp and Adam with Nesterov Momentum [VII] were tested along with different activation functions, but the final models included only the Adam optimizer and ReLU activation as they showed the best results compared to other combinations. Therefore, only the Adam optimizer and ReLU activation were used in the final models.

Conclusion and Future Work

In conclusion, this project explored the task of image classification using deep convolutional neural networks on the Tiny ImageNet dataset. The project aimed to develop models that could accurately classify images from this challenging dataset, and various deep learning architectures were evaluated to determine the most effective approaches for image classification.

The project started with a simple one-convolution layer model, which achieved an accuracy of just 19% on the validation data. This model was expanded to a two-layer model with L2 regularization, achieving a slightly better accuracy of 21%. However, it was recognized that these models were not sufficient for the task at hand, as the downsampling of the ImageNet dataset to Tiny ImageNet made it difficult for the models to learn the features necessary for accurate image classification.

Therefore, I created a deep CNN model with eight convolutional layers, which achieved around 40% accuracy. This model demonstrated that deeper models could learn more complex features and achieve better accuracy on the Tiny ImageNet dataset. I also tested a few pre-trained models, including ResNet50 and ResNet101, which achieved accuracies of around 55% each.

The Tiny ImageNet dataset is challenging due to its small size and low resolution, making it difficult for models to learn the features necessary for accurate image classification. However, with the use of deep convolutional neural networks and regularization techniques, I was able to achieve reasonable accuracies on this dataset.

Moving forward, there are several areas of potential improvement for this project. First, exploring more advanced regularization techniques such as dropout and batch normalization could improve the accuracy of the models. Second, experimenting with transfer learning using pre-trained models on the larger ImageNet dataset could also improve the performance on the Tiny ImageNet dataset.

Third, it is possible to construct even more intricate models like the VGGnet [VIII] by increasing the depth of the network. By doing so, the model will learn a more extensive set of parameters, enabling it to capture more complex features in the input data. Moreover, various activation functions and optimization algorithms can be experimented with to enhance the model's performance.

Overall, this project demonstrated the effectiveness of deep convolutional neural networks for image classification on the challenging Tiny ImageNet dataset. Through the evaluation of various deep learning architectures, I gained insight into the most effective approaches for image classification on this dataset, and future work can build upon these findings to further improve accuracy.

References

I. http://cs231n.stanford.edu/reports/2017/pdfs/931.pdf

II. https://www.image-net.org/

III. Shahar Lutati, Lion Wolf, *OCD: Learning to Overfit with Conditional Diffusion Models* https://arxiv.org/pdf/2210.00471v4.pdf

IV. Razvan Pascanu, Tomas Mikolov, Yoshua Bengio *On the difficulty of training Recurrent Neural Networks* https://arxiv.org/pdf/1211.5063.pdf

V. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun *Deep Residual Learning for Image Recognition* https://arxiv.org/abs/1512.03385

VI. https://www.tensorflow.org/tutorials/images/data_augmentation

VII. https://medium.com/@giorgio.martinez1926/nesterov-momentum-explained-with-examples-in-tensorflow-and-pytorch-4673dbf21998

VIII. Karen Simonyan, Andrew Zisserman *Very Deep Convolutional Networks for Large-Scale Image Recognition* https://arxiv.org/abs/1409.1556

IX. Project code - https://github.com/vishalbns/TinyImageNet_ImageClassification