# Tiny ImageNet Challenge

Yinbin Ma
Stanford University
yinbin@stanford.edu

## Abstract

*In this project we classify the images in the Tiny ImageNet dataset. We use residual networks on GPU using PyTorch. We first train a residual network from scratch, exploring the effect of different weight initialization and activation function during the training process. We then attempt to increase the validation and test accuracy starting from a pre-trained model. We use ResNet-18 to achieve* $31.6\%$ *test error on the evaluation server.*

## 1. Introduction

Convolution neural network (CNN) is a powerful tool for image classification [7]. With the development of residual networks, training a deep network becomes feasible [7, 8], and the accuracy has surpassed human interpreters. In this project, we use residual network to study the training process on the tiny ImageNet dataset.

In section 2, we briefly discuss the background and related work. In section 3, we show the Tiny ImageNet dataset which is used to train the model. In section 4, the methods for this work is explained including the configuration of residual network, optimization procedure and technique to overcome overfitting problems. In section 5, we show numerical experiments we have done during this project. We first train the models to overfit a small subset of the training dataset. We then explore the effect of weight initialization and activation function during the training procedure. Training a classifier from scratch has been tried and we achieve $\sim 40\%$ validation accuracy. In section 6, we attempt build a classifier using transfer learning. We fine tune the training process, and we achieved $31.6\%$ test accuracy computed by the evaluation server using ResNet-18 configuration. In section 7, we study the mislabeled images from the validation set.

## 2. Background and related work

The basic frame work for this project is the deep residual network [6, 8, 7] proposed in 2015. The key idea is to add an identity mapping to the convolution layers. With the identity mapping, training deep neural network becomes possible. Many Variation of the residual network has been proposed including using different identity mapping function [9]. In this project, we use a simple residual network (ResNet-18) to classify the Tiny ImageNet dataset.

The training procedure is affected by the weight initialization [4, 11, 8]. and the choice of activation [7, 3, 5]. In this project we use different weight initialization and rectifier during the training procedure.

Limited by the size and quality of the training set, training a classifier from scratch is challenging in this project. An efficient way to train a model is to use transfer learning technique [10] to extract features from a pre-trained model. In this work, we build a classifier starting from a pre-trained ResNet-18 model with the help of existing code [2, 1].

## 3. Tiny ImageNet Dataset

The Tiny ImageNet dataset contains images with 200 different categories. The training set has $10^5$ images and each category contains 500 images. The validation set and test set has $10^4$ images (50 images per category). Each image is $64 \times 64$ in size. The Tiny ImageNet dataset comes from ILSVRC benchmark test but with fewer categories and lower resolution.

In Figure 1 we plot a few images from 7 categories. The resolution of the images limits our ability to achieve high accuracy. In the numerical experiments, we can achieve $50\% \sim 60\%$ accuracy if the model is trained from scratch, and $30\% \sim 35\%$ accuracy when transfer learning is applied. In either case, the accuracy is lower than the reported results on ILSVRC benchmark using deep residual networks [8].

## 4. Methods

In this section we describe the deep residual network for the image classification problem, the optimization process, and the evaluation methods.

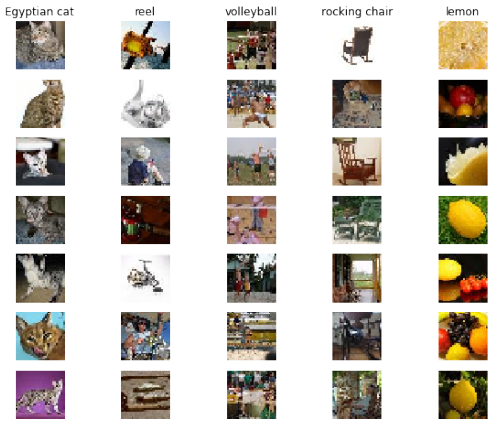


Figure 1. Samples of images from the Tiny ImageNet dataset

### 4.1. Residual Network

We use the residual network developed recently [8]. Deep residual networks with different depth have been explored, from ResNet-18 to ResNet-152 [8]. The network with 152 layers achieved $3.5\%$ error rate on the ILSVRC benchmark test in 2015 [7].

Due to the limited number of resource for the project, we used the simplest configuration ResNet-18 in this project. Training from scratch and transfer learning are both explored in this study.

The basic building block for residual network is shown in Figure 2. Comparing with CNN building blocks, there is an identity mapping on top of the nonlinear layers. With the identity mapping, we can skip arbitrary layers in principle. The identity mapping linearizes the problem which simplify the optimization procedure.

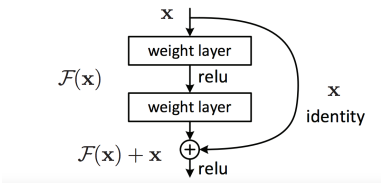


Figure 2. building block for residual learning. Image from [8].

In Figure 3 we show the configuration of ResNet-18 model. The input images are first convolved with 64 filters with size $7 \times 7$, and then followed by batch normalization, rectifier, and max pooling. 8 residual learning building blocks are applied afterward with $3 \times 3$ filters. Finally a fully connected layer with size $512 \times 200$ are used to compute the scores for each class and to compute the loss function.

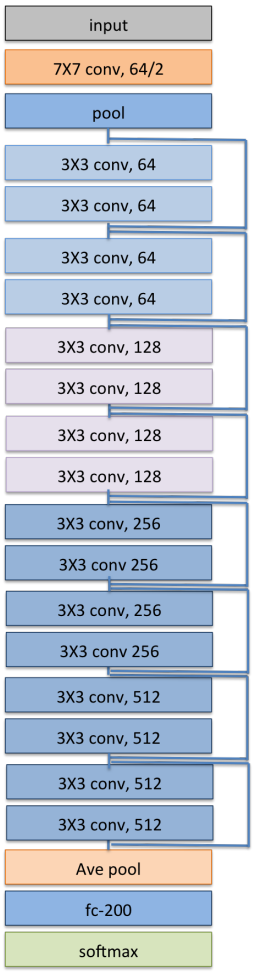


Figure 3. ResNet-18 configuration. It has a $7 \times 7$ convolution followed by 8 residual network building blocks with $3 \times 3$ convolution. Fully connected layer with 200 output is used to compute the label and loss function at the end.

### 4.2. Optimization procedure

We use softmax classifier to predict the labels, and compute the loss function using cross entropy,

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) \tag{1}$$

where $L_i$ the loss function for image $i$, and $s_j$ are the output from the fc-200 layer.

To minimized the loss function, we can use different optimization tool including Stochastic gradient descent (SGD), SGD+momentum, RMSProp, etc. For this work, we use Adam algorithm, which utilizes the first and second moments in the optimization procedure.

The optimization procedure is also affected by the choice of hyperparameters, such as learning rate, weight decay, dropout rate, etc.

In addition, the weight initialization and the choice of activation function play important roles in the training procedure. Proper weight initialization such as random Gaussian initialization or Xavier initialization have better performance comparing with uniform random initialization or zero initialization. ReLU is used in the ResNet-18 as the default rectifier, and it may suffer from gradient saturation problem. LeakyLU and ELU have been proposed to overcome the saturation problem. In the numerical experiments section, we are going to explore the effect of weight initialization and activation function.

### 4.3. Overfitting problem

We have 500 images per category for the Tiny ImageNet dataset. Considering the fact the ResNet-18 is designed for the original ImageNet Dataset with 1000 categories, it can easily overfit the Tiny ImageNet dataset. To increase validation accuracy and test accuracy, we need to overcome the overfitting problem.

Dropout and model regularization are commonly used to prevent overfitting. For dropout, we randomly set neurons to zero and only use part of the network to predict the label. It forces the network to have a redundant representation. The regularization technique add terms to the loss function which depends on the weight of the parameters. In case of L2 norm regularization, it is equivalent to adding a weight decay during training process. Since we have tested dropout and regularization in the assignments, they are not studied in this work.

Data augmentation is another useful technique to prevent overfitting. By simply flipping the images horizontally, we can double the size of training dataset although the flipped images has the same statistics (mean, variance, etc). Random crops and scales are also used for data augmentation. We use data augmentation in section 6 to achieve low validation and test error rate.

## 5. Numerical Experiments

In this section we show several numerical experiments on the Tiny ImageNet dataset using residual network.

### 5.1. Overfitting a Small Dataset

As a sanity check, we want to overfit a small dataset using the residual network. We choose 100 images from the training set. Model from scratch and pre-trained model are both tested. In Figure 4, we show the training accuracy as a function of epochs, where learning rate $10^{-4}$ is used. For the pre-trained model, we have reasonable convolution layers, and we can overfit within 10 iterations. When we train from scratch, it takes around 40 iterations to overfit.

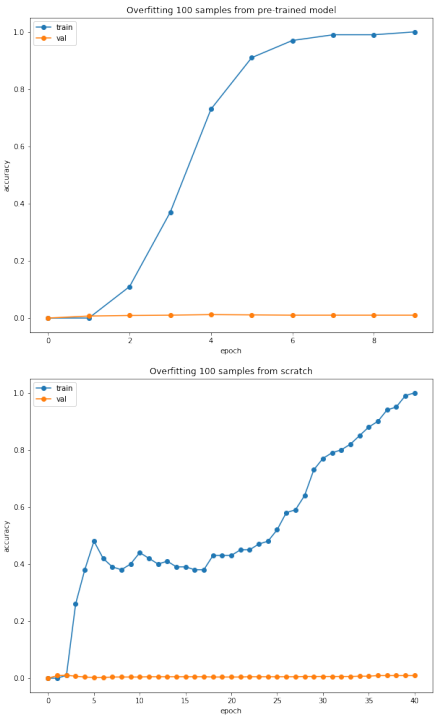


Figure 4. Top: accuracy using pre-trained model. Bottom: accuracy with model trained from scratch. We use model ResNet-18 configuration, learning rate $10^{-4}$.

### 5.2. Weight Initialization

The residual network has only one fully connected layer (not followed by activation layer). The default constructor use uniform random number for the weighting function, and we compare it with Xavier initialization, where the weight distributions are shown in Figure 5. We train the models from scratch for 1 epoch (around 390 iterations), and the loss functions are shown in Figure 6.

The initial loss function for Xavier initialization is higher than the default uniform initialization and it decays faster. After iteration 30 they are indistinguishable. At the end of the first epoch, we achieve 0.1723 accuracy for the training set, 0.148 for the validation set using Xavier initialization. As a comparison, default initialization has 0.147 training accuracy and 0.119 validation accuracy. No conclusion can be made at stage.

### 5.3. Activation Function

Activation function introduces nonlinearity in the neural networks. ReLU is commonly used for CNNs, and it is also the default rectifier for ResNet-18. Other popular rectifiers includes Leaky ReLU and ELU as shown in Figure 7. The ReLU suffers from gradient saturation at $x < 0$ while Leaky ReLU does not.

We use 3 different activation function, ReLU, leaky

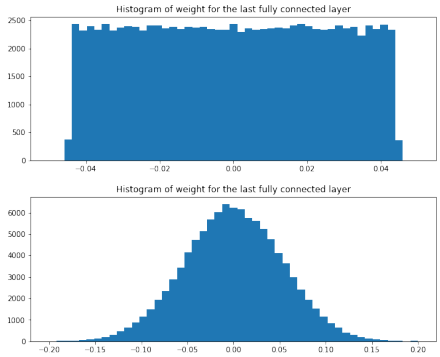


Figure 5. Top: histogram of default weight initialization for the fully connected layer. Bottom: histogram of Xavier initialization.

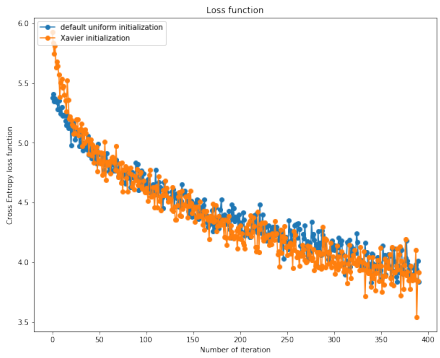


Figure 6. Loss function for different weight initialization. Xavier initialization has higher initial loss and the loss function decays faster.

ReLU and ELU in the residual network. We train the models from scratch with learning rate $10^{-4}$. After 1 epoch, the loss functions are shown in Figure 8. The loss function for ReLU and leaky ReLU are overlapping with each other and no observable difference has been found. The loss for ELU is slight better than ReLU for the first 200 iterations.

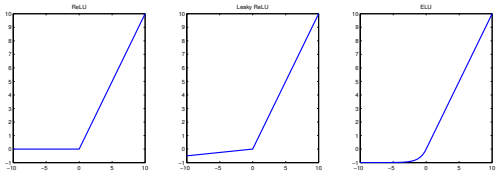


Figure 7. Activation function. Left: ReLU. Middle: leaky ReLU. Right: ELU.

### 5.4. Train the Model From Scratch

After we have gained experience on the residual network and the dataset, we train a model from scratch to achieve low error rate. We use all the training images ($10^5$), mini-

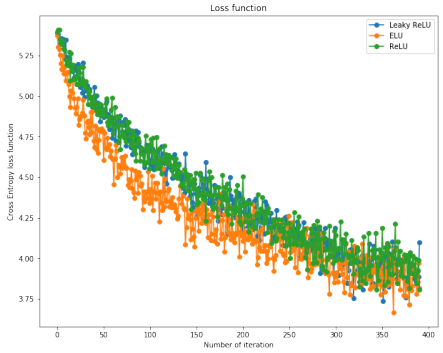


Figure 8. Loss function with different activation function. The loss function suggests that ELU is slightly better than ReLU.

batch size is set to 256 (390 iteration per epoch), learning rate is set to $10^{-4}$ for the first 7 epochs, and we use weight decay $10^{-4}$ during the training process.

The loss function is shown in Figure 13. The loss function is decaying as a function of number of iteration, as expected. The accuracy as a function of epochs is shown in Figure 11. After 10 epochs, we get $\sim 40\%$ validation accuracy.

The validation accuracy is significant lower than the case of using learning transfer which we will discuss in the next section. To understand the difference, we visualize the filters at Figure 9. We start from filters with random weight, and after 10 epochs, the filters shown different illumination but no clear pattern can be observed. As a comparison, the filters from the pre-trained model is shown in Figure 9, where they can be interpreted as edge detection filter, etc.

## 6. Use Pre-trained model to Increase Accuracy

In this section we try to increase the validation and test accuracy using learning transfer. We import the convolution filters from a pre-trained ResNet-18 model. The pre-trained model has 1000 classes while we have 200, therefore we need to train the last fully connected layer.

We use the following procedure to get 0.316 test accuracy:

1. For epoch 1 and 2, we fix the convolution layers and only train the last fully connected layer with learning rate $10^{-3}$.

2. For epoch 3 to 5. We train all the parameters in the network, with learning rate $10^{-4}$, weight decay $10^{-4}$. At the end of epoch 5, the training accuracy reaches 0.976 and we could not improve the validation accuracy anymore.

3. For epoch 6, we use random crop and random horizontal flip on the training set, with learning rate $10^{-5}$. The validation accuracy decreases at this step.

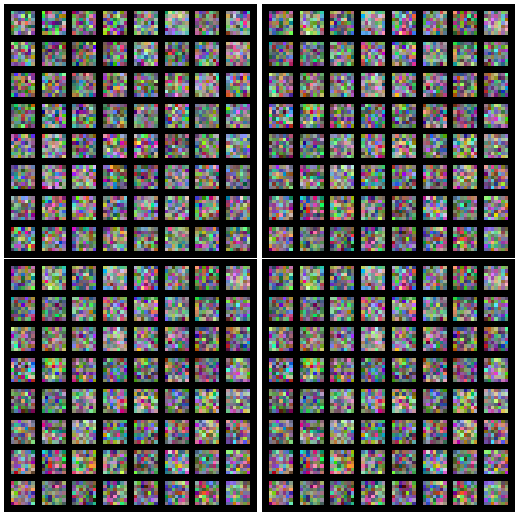

Figure 9. Top left: convolution filters of the first layer after initialization. Top right: convolution filters of the first layer after 2 epochs. Bottom left: convolution filters of the first layer after 6 epochs. Top right: convolution filters of the first layer after 10 epochs.

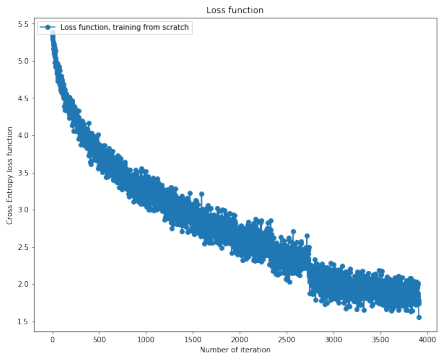


Figure 10. Loss function when we train the model from scratch.

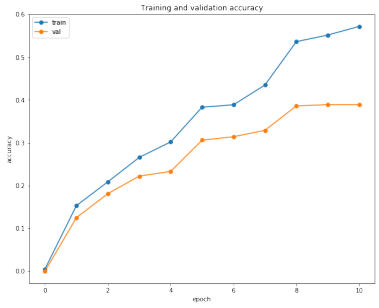


Figure 11. Training and validation accuracy when we train the model from scratch.

4. For epoch 7, we train all the parameters with learning rate $10^{-5}$. At the end of epoch 7, we get 0.067 training error, 0.257 validation error, and 0.316 test error.

5. After epoch 7, we try different strategies, such as using a subset of the training set and for a few epochs. The idea is to test if we can find different local minimum of the loss function that achieves better test accuracy. No further improvement has been observed.

In Figure 13 we show the loss function and in Figure 14 we show the train and validation accuracy. Notice that data augmentation is applied at epoch 6 (random crop and random horizontal flip) which reduces the accuracy.

The convolution filters at the first layer are shown in Figure 12. On the left of Figure 12 is the filters from pre-trained model, and on the right is the filters after our training process. It is interesting to notice that while most of the filters remain unchanged, some filters have significant update (the filter at row 1, col 3, and the filter at row 5, col 7, for example).

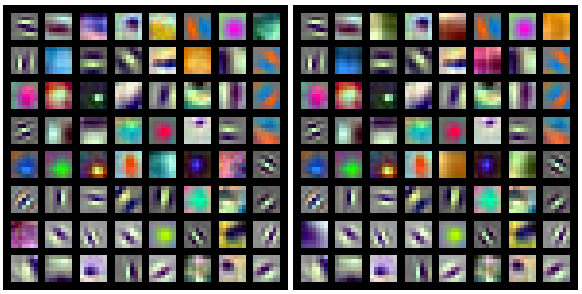

Figure 12. Left: convolution filters from the pre-trained model. Right: convolution filters after our training process.

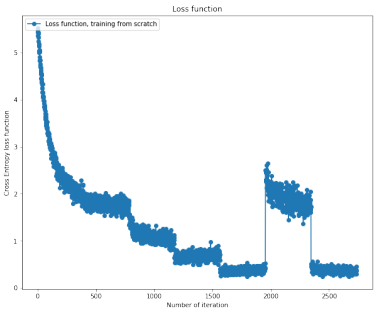


Figure 13. Loss function during the training process. Before iteration 780 we only train the last fully connected layer. From iteration 780 to iteration 1950 we train all the parameters, using the same data processing for training and validation set. From iteration 1950 to 2340 we use different data processing for the training set to prevent overfitting. After iteration 2340, the same data processing is applied.

## 7. Error Analysis

In this section we briefly discuss the validation and test error. The best test error rate we have obtained is 31.6%

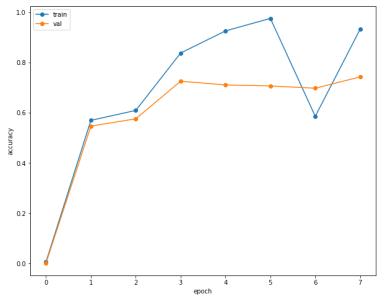


Figure 14. Training and validation accuracy during the training process. Decrease of training accuracy at epoch 6 is observed because we use random crop and random horizontal flip.

which is higher than the reported error rate in the ImageNet Challenge [7]. Since we have reached high accuracy on the training set ($> 0.97$), we do not expect significant further improvement on the test dataset.

To understand the error, we selected a couple of mislabeled images from the validation dataset as shown in Figure 15. One of the mislabeled image (left image in Figure 15) belongs to n02085620 (Chihuahua) while our model predict it to be n04399382 (teddy bear). The error is expected because of the low resolution, and the chihuahua has cloth on which is uncommon in the training set. Another mislabeled image (right image in Figure 15) belongs to n04118538 (football) while we predict it to be n04540053 (volleyball). Again the error is not surprising due to the color of the ball (yellow-green is common in volleyball but not in football), and the action of the player (looks like he is spiking the ball).

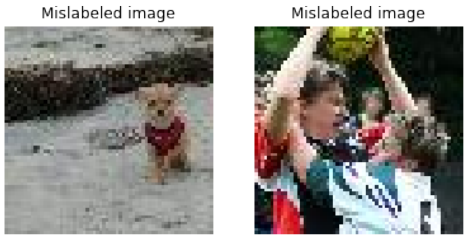


Figure 15. Mislabeled image from the validation dataset. Left: predicted label n04399382 (teddy bear) and actual label n02085620 (chihuahua). Right: predicted label n04540053 (volleyball) and actual label n04118538 (football).

## 8. Conclusions

In conclusion, we classify the images in the Tiny ImageNet dataset in this project. We train deep residual network on GPU using PyTorch library. We study the effect of different weight initialization and activation function during the training process. Training a classifier from scratch is challenging due to the low resolution and low quantity of training images. We use a pre-trained model ResNet-18 and fine tune the training procedure to achieve $31.6\%$ test error and $25.6\%$ validation error for the Tiny ImageNet challenge.

## References

[1] Pre-trained resnet-18. `https://download.pytorch.org/models/resnet18-5c106cde.pth`.

[2] Pytorch tutorial for data loading and fine tuning. `https://gist.github.com/jcjohnson/6e41e8512c17eae5da50aebef3378a4c`.

[3] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015.

[4] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[5] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout Networks. *ArXiv e-prints*, Feb. 2013.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *ArXiv e-prints*, Feb. 2015.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.

[10] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *CoRR*, abs/1403.6382, 2014.

[11] D. Sussillo and L. F. Abbott. Random Walk Initialization for Training Very Deep Feedforward Networks. *ArXiv e-prints*, Dec. 2014.