Assignment 2 Report

Asmae Mouradi

Deep Learning

Supervised by: Dr. Lokesh Das

Table of figures:

I. Introduction

In this assignment, I will be using two different datasets to address each question. For Question 1, I have downloaded Tiny ImageNet, a subset of the ILSVRC 2010 dataset, from the GitHub link [1].

For question 2, I compare ReLU and Tanh as hidden-layer activations in a four-convolution-layer CNN trained on CIFAR-10 (60,000 images, 32×32 RGB, 10 classes). I keep the architecture and training identical across runs and change only the activation function. I stopped training when the training error is less or equal to 25% meaning training accuracy > 75%. I report two curves for each activation: training error vs. epochs and time per epoch vs. epochs. I also select a learning rate separately for each activation from a small candidate set, choosing the one that reaches the stop criterion fastest.

I have organized my repository as follows:
The ques1 folder contains the sub-dataset I created from Tiny ImageNet , I have compressed the dataset under the folder ques1 and pushed it to GitHub under the name data_subset_tiny_imagenet.zip, and the question_2 folder contains the Jupyter Notebook named q2_relu_vs_tanh_cifar10.ipynb where I ran the experiment. My repository is hosted on GitHub, and I set it to public visibility so that the professor can access it. The report is already uploaded to my GitHub repository under the name assignment 2 report.

Here is the link to my assignment: https://github.com/asmaemou/assignment2-DL

II. Methodology

a. Question 1

In my repository *tiny-imagenet-200* is the name of the original dataset tiny imagenet and *data_subset_tiny_imagenet* is the dataset I have created.

Tiny ImageNet Dataset Preparation

I downloaded the Tiny ImageNet dataset from the Stanford CS231n course website after encountering token issues when attempting to retrieve it from DeepLake [3]. I obtained the dataset using the link provided in the README section of the GitHub repository [1] using this link http://cs231n.stanford.edu/tiny-imagenet-200.zip
Tiny ImageNet is a smaller version of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). It contains 200 image classes. The dataset includes 100,000 images for training, 10,000 images for validation and 10,000 images for testing. The dataset includes images in RGB format, each 64x64 pixels, labels meaning tensors identifying the object class in each image, and bounding box annotations for object localization tasks, which are represented by tensors indicating the position of the object within the image in the format [xmin, ymin, xmax, ymax]. To create a subset of the

Tiny ImageNet dataset, I selected 100 classes from the original 200 classes. I started by using the train subset of the Tiny ImageNet dataset, as it is already labeled and contains enough images to generate a dataset. I randomly selected 100 classes instead of choosing the first 100 classes in sorted order using a fixed random seed to ensure reproducibility. For each class, I collected 500 images. I then split the images as follows, 300 images for the training set, 100 images for the validation set, and 100 images for the test set. As a result, I got a total of 30,000 training images, 10,000 validation images, and 10,000 test images.

Image Collection and Bounding Box Annotations

For each selected class, I collected exactly 500 images from the "images" subfolder in the Tiny ImageNet dataset. I also considered the bounding box annotations provided in the dataset, each class has a corresponding file, named <wnid>_boxes.txt, which contains bounding box information for some images. The WordNet ID (wnid) represents the class, and the bounding boxes are provided in the format [xmin, ymin, xmax, ymax] for each object in the image. While collecting the images from each class folder, I filtered the images to include only valid formats, such as *.jpg, .jpeg, and .png.* Images in Tiny ImageNet have bounding boxes.

Dataset Splitting

I have split the dataset into three parts, 300 images for training, 100 images for validation, and 100 images for testing. I split the images randomly using a fixed seed to ensure reproducibility.

Data Preparation

To prepare the data for my model, I followed the preprocessing steps described in Section 2 of the AlexNet paper [2]. I have performed two main steps resizing, cropping and normalization.

1. Resizing and Cropping:
    - I resized the images so that the shorter side became 256 pixels to ensure that all images have similar scale and size.
    - I applied center cropping to be certain the final image size is 224x224 pixels which matches the input size required by AlexNet.
2. Normalization:
    - I calculated the mean and standard deviation for each of the RGB channels (Red, Green, Blue) across all the training images. Then, I saved those statistics in a file named dataset_stats.json.
    - I will use the values inside dataset_stats.json file to normalize the images so that each channel has a zero mean and unit variance.

WordNet IDs and Class Mapping

Each image in the Tiny ImageNet dataset belongs to a specific class identified by a WordNet ID (wnid). For instance, the WordNet ID "n01443537" corresponds to the goldfish class. The dataset also provides a file called words.txt, which maps each WordNet ID to a readable class name. For example, n01443537 represents "goldfish", n01629819 represents "king snake". Additionally, I created JSON label files that map each WordNet ID for example "n01443537" to a numerical class ID such as 0, so the model can easily use numerical labels during training. This mapping allows the model to work with numeric class IDs instead of WordNet IDs during training. The file classes.txt contains a list of class identifiers each entry corresponds to a unique class in the Tiny ImageNet dataset.

Manifest Files

To organize the dataset, I created manifest files for each image in the training, validation, and test sets. Each manifest file contains eight columns:

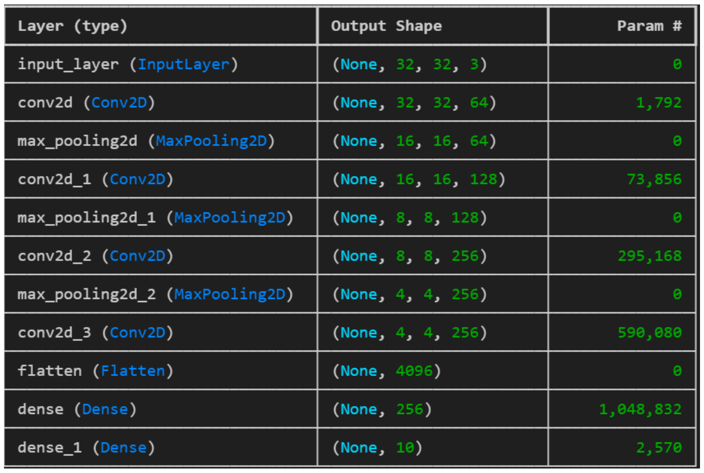
1. split: Indicates whether the image is in the training, validation, or test set.
2. relative_path: The relative path to the image file.
3. class_name: The name of the class the image belongs to.
4. class_id: A numerical label used by the model (mapped from the WordNet ID).
5. bbox_x1, bbox_y1, bbox_x2, bbox_y2: The bounding box coordinates

Each row in the manifest file represents one image and its associated metadata which are split, path, class name, class id, bounding box.

b. Question 2:

I load CIFAR-10 directly from Keras. The dataset is split into training set with 50,000 images and test set 10,000 images. From the original paper of imagenet [1] when they did the experiment on CIFAR-10 they explicitly mention that no regularization was used. I scale pixel values to [0,1] by dividing them by 255. I built a four-conv CNN using the same structure, the only difference between the two experiments is the activation function, one experiment used ReLu in all hidden layers and the other experiment used Tanh in all hidden layers. I have used softmax in the output layers since I have a multiclass classification. I use *sparse_categorical_crossentropy* loss because CIFAR-10 is a single-label multi-class task with integer labels, and this loss pairs correctly with a softmax output without needing one-hot encoding.

Regarding the network architecture both experiments used the same 4-layer convolutional network:



Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 32, 32, 3)	0
conv2d (Conv2D)	(None, 32, 32, 64)	1,792
max_pooling2d (MaxPooling2D)	(None, 16, 16, 64)	0
conv2d_1 (Conv2D)	(None, 16, 16, 128)	73,856
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 128)	0
conv2d_2 (Conv2D)	(None, 8, 8, 256)	295,168
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 256)	0
conv2d_3 (Conv2D)	(None, 4, 4, 256)	590,080
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 256)	1,048,832
dense_1 (Dense)	(None, 10)	2,570

Figure 1. Model Summary of the Four-Layer Convolutional Neural Network (CNN)

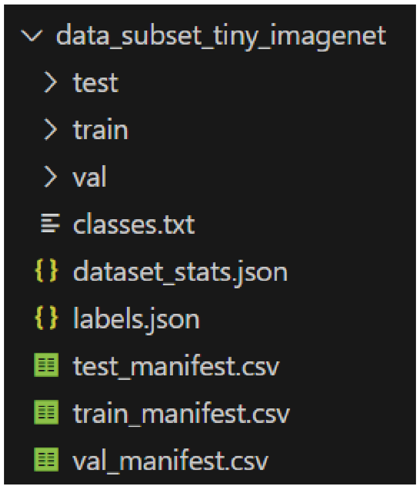
Figure 1 shows the use of four Conv2D layers, each followed by max pooling, then a flattening layer, one hidden dense layer, and a final output dense layer with a softmax activation.

As stated in the paper [1], *"The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed"*. I therefore tried a small range of learning rates and selected the one that reached the ≤25% training-error threshold in the fewest epochs. With this protocol, ReLU used LR = 0.2 and stopped at epoch 6, while Tanh used LR = 0.05 and stopped at epoch 8.

In the paper [2], they did not explicitly mention which gradient descent they have used when training on CIFAR-10. I have opted for stochastic gradient descent (SGD) as optimizer with batch size of 128. I have set the stopping criteria as soon as the training error is less than or equal 25%.

III. Experiments and Results

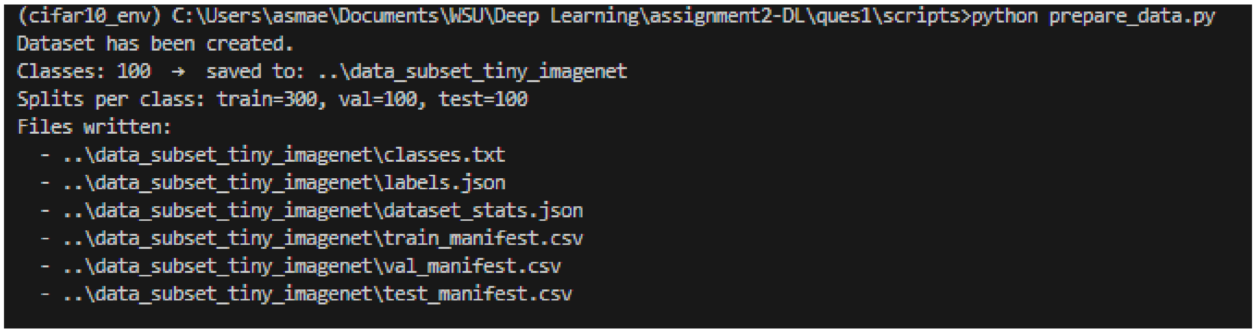
a. Question 1 results:



```
data_subset_tiny_imagenet
  test
  train
  val
  classes.txt
  dataset_stats.json
  labels.json
  test_manifest.csv
  train_manifest.csv
  val_manifest.csv
```

Figure 2. Structure of the dataset

Figure 2 is the dataset that I have prepared based on Tiny Imagenet dataset.



```
(cifar10_env) C:\Users\asmae\Documents\WSU\Deep Learning\assignment2-DL\ques1\scripts>python prepare_data.py
Dataset has been created.
Classes: 100  →  saved to: ..\data_subset_tiny_imagenet
Splits per class: train=300, val=100, test=100
Files written:
  - ..\data_subset_tiny_imagenet\classes.txt
  - ..\data_subset_tiny_imagenet\labels.json
  - ..\data_subset_tiny_imagenet\dataset_stats.json
  - ..\data_subset_tiny_imagenet\train_manifest.csv
  - ..\data_subset_tiny_imagenet\val_manifest.csv
  - ..\data_subset_tiny_imagenet\test_manifest.csv
```

Figure 3.Dataset Creation and File Generation for Tiny ImageNet Challenge

Figure 3 shows the output of the terminal after successfully creating my dataset based on Tiny dataset. Along with the six files that I have created. Here is a brief explanation of the six files:

- **classes.txt:** it has a list of unique class identifiers for example n01443537, n01629819 , n01641577 and so on
- **labels.json:** I created this file to use later during model training and evaluation. It allows me to refer to each class by its corresponding class ID instead of dealing with long class identifiers.
- **dataset_stats.json**: I have created the dataset_stats.json file to store important preprocessing information for the subset of the Tiny ImageNet dataset. This includes:
    - Mean RGB values and standard deviation of RGB values, which are used for normalizing the dataset before training the model. These values are calculated for each color channel (Red, Green, and Blue).
    - Total pixels counts across all images in the dataset, providing insight into the overall scale of the data.
    - Resize settings, where the shorter side of each image is resized to 256 pixels, ensuring consistent image dimensions.
    - Crop size of 224x224 pixels, which is used after resizing to crop the image and ensure consistent dimensions suitable for input into the model.
- **Manifest files**: Each of the manifest files (train_manifest.csv, val_manifest.csv, test_manifest.csv) contains 8 columns that organize and describe the image data.
    - Split: subset the image belongs to. This will be "train" for the training set, "validation" for the validation set, and "test" for the test set.
    - relative_path: location of each image file.
    - class_name: refers to the WordNet ID

- class_id: numerical ID assigned to the class
- bbox_x1: x-coordinate of the top-left corner of the bounding box
- bbox_y1: y-coordinate of the top-left corner of the bounding box
- bbox_x2: x-coordinate of the bottom-right corner of the bounding box
- bbox_y2: y-coordinate of the bottom-right corner of the bounding box

b. Question 2 results:

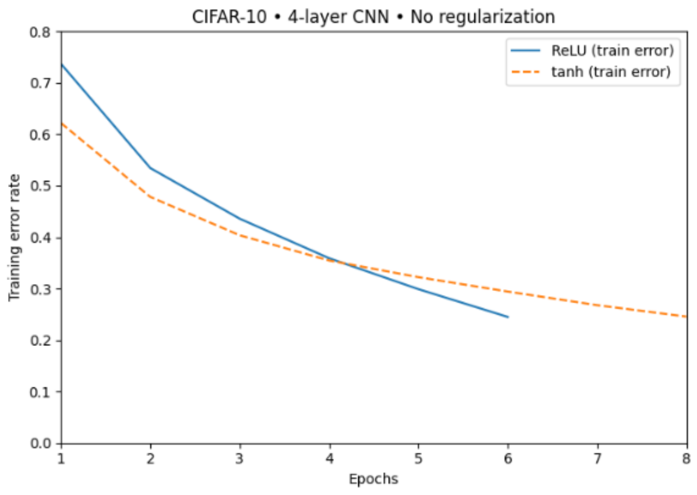


Figure 4. Training error vs. epochs (CIFAR-10, 4-layer CNN)

In figure 4, I plotted training error for ReLU with learning rate 0.2 and Tanh with learning rate 0.05. I stopped when training error is less or equal than 25%. From figure 4 the blue line shows ReLU activation function, and the dashed orange line is tanh activation function. From figure 4, we can see that ReLU reaches the threshold at epoch 6 while Tanh at epoch 8.

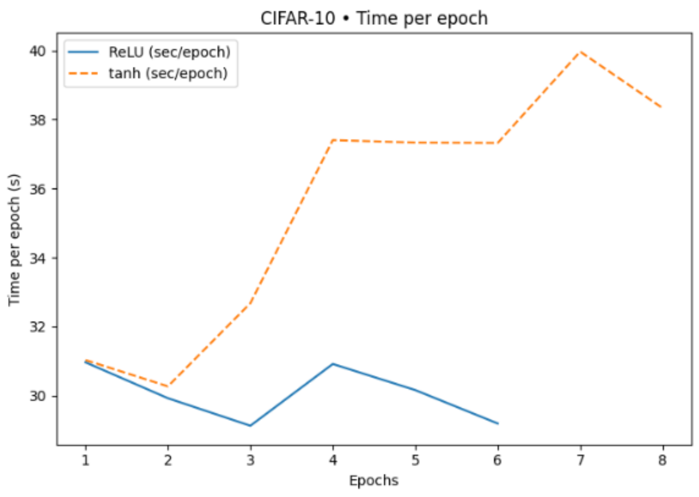


Figure 5. Time per epoch (ReLU vs. Tanh)

Figure 5 shows ReLU curve stays lower, meaning it trains faster per epoch. However, Tanh curve rises over time meaning that training with Tanh becomes slower as epochs progress. Based on the results I can conclude that ReLU activation allows a CNN to learn much faster compared to tanh.

IV. Conclusion

For Question 1 of this assignment, I downloaded the Tiny ImageNet dataset and selected 100 classes to create a custom subset. I then split the data into training, validation, and test sets using a fixed random seed to ensure reproducibility. The images were resized and cropped to meet the input requirements of AlexNet architecture. Next, I calculated the mean and standard deviation for the RGB channels and saved these statistics in the dataset_stats.json file to be used for image normalization during training. I also generated a labels.json file to map the WordNet IDs (class names) to numerical class IDs for efficient model training. Finally, I created manifest files that describe each image, its class label, and its bounding box annotations.

Regarding Question 2, ReLU converges faster because it does not saturate for positive inputs its gradient stays 1, so signals don't vanish during backpropagation. In contrast, tanh saturates in [−1, 1]; its gradient becomes close to 0, slowing learning this what lead to vanishing gradient problem. This is why the ReLU model reaches the 25% training-error threshold earlier than the tanh model. To sum up, CNN can learn much after using ReLU activation compared to tanh. It is highly recommended to use ReLU instead of tanh. So indeed, I was able to get the same results as figure 1 in the paper [2].

References:

[1] R. McCormick, *rmccorm4/Tiny-Imagenet-200*. (Sept. 29, 2025). Python. Accessed: Oct. 09, 2025. [Online]. Available: https://github.com/rmccorm4/Tiny-Imagenet-200

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.

[3] "Tiny ImageNet Dataset - Machine Learning Datasets." Accessed: Oct. 14, 2025. [Online]. Available: https://datasets.activeloop.ai/docs/ml/datasets/tiny-imagenet-dataset/