Handwriting Company Logos Recognition

1st Chih-Hao (Andy) Tsai
Department of Electrical, Computer and Energy Engineering
Arizona State University
Tempe, United States
ctsai67@asu.edu

I. Abstract

The use of artificial neural network, especially the field of machine vision, became viral over the years. In this project, I will design a few different image-based recognition neural network structures to recognize the logo designs of different brands from the past to present. The system allows the users to input graphic or hand drawing through a user interface (UI), then, the neural network can accurately classify the input to associated company logo and display the image with highest prediction probability.

II. Introduction

In recent years, the need for applications of artificial intelligence has grown exponentially. Especially in the field of image-based recognition systems, a lot of techniques are implemented over the years such as image classification, object detection, image style transferring, etc. This project focuses on developing an image recognition system using deep learning techniques, such as convolutional neural network (CNNs), to recognize hand drawing image from a predefined logo dataset. Throughout this project, a few different structure of CNN would be built as a comparison of existing models, such as AlexNet, ResNet34, etc.

The goal of this project is to train a deep neural network that is good enough and can recognize the input image, which is from users' handwriting or drawing, and output a most similar image from a predefined dataset-*Logo-2K+*. This dataset consists of all the logos of various companies from the past to the present.

To achieve the goal, a CNN architecture has to be built or defined before training on the dataset. Once the network's architecture is built, training process and evaluation of the model can be performed. After the model is trained and evaluated, we can extract feature vectors from the dataset in order to implement the similarity search. With the feature vectors, we now can get user's hand drawing as input image and out put the image with highest similarity.

In this project, I will customize a neural network based on my knowledge of deep learning. Also, I will introduce some pretrained models (Swin Transformer, AlexNet, etc.) as comparison to my own model and analyze the final result.

III. Dataset

This dataset includes over 160,000 256×256 RGB images, 10 root categories(food, clothes, etc.) and over 2300 different brand's logos.

In this project, I apply image augmentation in order to improve testing accuracy.(figure 2)
The original source of this dataset: *Jing Wang, and Weiqing Min, and Sujuan Hou, and Shengnan Ma, and Yuanjie Zheng, and Haishuai Wang, and Shuqiang Jiang, " Logo-2K+: A Large-Scale Logo Dataset for Scalable Logo Classification"*, 2020

Figure 1. A glance of sample images with their labels in the dataset.

Figure 2. Sample images after implementing image augmentation.

IV. Related Work

A. Similrity Search

Similarity Search is a method for estimating the similarity between two or more objects. To achieve similarity search, labeling data in the dataset with its own features is vital. Normally, labels of data would be their feature vectors (or latent vectors). Then, we can calculate the distance between different vectors, which indicates the similarity between data points. Lastly, we can perform searching with different approach. For example, the popular algorithm-K Nearest Neighbors can find certain number of nearest vectors (K vectors) for the input vector.

There are some methods to estimate the similarity with given vectors, such as using Euclidean Distance between vectors. The most common way is to calculate the *Cosine Similarity* between vectors (equation 2). The A and B in the equation represent the feature (latent) vectors of images, $\|\mathbf{A}\|$ and $\|\mathbf{B}\|$ represent their magnitude.

In *Learning similarity with cosine similarity ensemble* [10], the team propose a cosine similarity ensemble method that allows the original cosine similarity search be more adaptive to noises. In this project, I also tried the method in the paper, however, it doesn't make much difference for using original cosine similarity and cosine ensemble similarity to predict images in my dataset.
The equation for calculating Cosine Similarity:

$$\text{Cosine Similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} \tag{1}$$

$$= \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2}$$

B. Vision Transformer-Swin Transformer

To understand *Swin transformer*, we have to understand *Vision Transformers (ViTs)* first. ViTs is a structure that introduce transformer architecture into computer vision that outperform the original deep learning structures such as CNN and ResNet [13]. However, the computational complexity of ViTs is always a problem because of its global self-attention mechanism. Also, the lack of the hierarchical representations in ViTs structure makes it less powerful when it comes to dense prediction tasks.

In *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows* [12], the team proposed another vision transformer model-*Swin Transformer*. In contrast to ViTs, Swin Transformer use *shifted window attention* (figure 3) making the computational complexity more efficient and address the hierarchical feature problem. These characteristic make Swin Transformer suitable for some computer vision tasks such as object detection, image classification, and image segmentation.

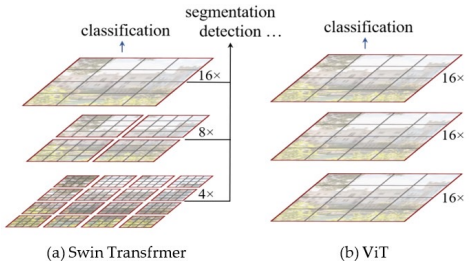

Figure 3. The comparison between the mechanism of original ViTs (global self-attention) and Swin Transformer (shifted window attention). From *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows* (p.1) by Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, Baining Guo (2021, 17 August) [12]

C. Convolutional Neural Network (CNNs)

A Convolutional Neural Network (CNNs) is a type of Deep Learning architecture that is mainly used to perform image classification. Similar to other kind of neural network, CNNs also contain input layer, hidden layers, output layer. What makes CNNs so special is that the hidden layers of CNNs consist of convolutional layers, pooling layers, etc. Before explaining how convolutional layers work, we have to define some terminology:

- Kernel: A kernel is also called a weight matrix or a filter. It essentially is a weights matrix that

can later be used for creating feature maps. Normally, it moves around the whole image (steps by steps) and extracts features from the input (either input image or other feature maps).
- Stride: Steps that decide how the kernel move around the image. It is used to reduce the size of output feature maps and reduce the computational complexity. Larger stride means the kernel moves faster and faster the computation. Thus, it may lead to smaller output feature map and may skip some regional features. By contrast, smaller stride leads to slower kernel movement, slower computation, larger output feature map with more detail.
- Padding: A process that add additional pixels to the boundary of input. It is meant to prevent information loss at the edges of the image and maintain spatial dimension of feature maps. Typically, we use zero padding, which is adding zeros to the boundary of input.
- Depth: The number of channels from the input, for example, RGB images have depth of 3 for the input convolutional layers. Each depth has one corresponding kernel (filter).
- Feature Maps: A feature map is the result from implementing a filter on the input image. Feature maps are usually a tensor that is used to represents the features extracted by filters.

A classic Convolutional Neural Network usually consists of different layers, where each layer do different operations:

- Convolutional Operation and activation function (convolutional layer): This layer implement the kernels to the input based on the parameters (such as padding and stride) that the layer predefined. The outputs of convolutional operation are multiple feature maps. After convolutional operation, an activation function (usually is ReLU) is applied element-wise to the generated feature maps.
- Pooling layer: Pooling layer is used to reduce the dimension of the output of convolutional layers. Depending on the pooling method, pooling procedure is applied on elements in a region covered by kernel (filter). Normally, this layer summarize the features in the output feature maps. The most common method for pooling is Max Pooling, Average Pooling, etc.
- Flattening layer: This process aims to convert the generated 2-dimensional arrays (feature maps) into 1-dimensional vectors so that the following fully connected layers can classify the data.
- Fully connected layer (Dense layer) an d Softmax: This layer serves as classifier for the output of previous layers. The Softmax function can convert raw prediction from fully connected layer into probability distribution. Therefore, Softmax layer is the last layer before output layer of the whole CNN. The input of dense layer is vectors from the flattening layer.

To improve the performance of the neural network, some methods also being introduced:

- Batch Normalization: As its name implies, This technique normalizes the input activation of each layer across a mini-batch. This technique is used to accelerate and stabilize the training process of the neural network by reducing internal covariate shift.
- Dropout: This technique temporarily ignore certain randomly selected neurons during training session, which can prevent the network from overfitting.

D. Residual Networks (ResNet)

The architecture of Residual Neural Network is consist of multiple residual blocks. This structure aims to solve the problem of diminishing gradient over the course of neural network. In the structure of a residual block, a skipping connection (or a shortcut) is added from the input to the output as can be seen in figure 4. In the field of image recognition, the weighted layers of residual blocks usually are replaced by concolutional layers. Depending on the design of residual blocks and the number of residual connections, different architectures are created. For example, the most common one-ResNet34 has 16 residual blocks with 2 layer convolutional layers each.

V. METHODS

A. Create Graphical User Interface (GUI)

First step to complete this project is to build the graphical user interface with some functions.

The most important function in my case is to build a predict function that can extract the feature (latent) vector from user's input image and use it on similarity search, in order to output the image with the highest similarity in the dataset. Also, other functions are such as choosing drawing color, saving input image, clear the canvas, etc. To perform these, I use Python built-in package-Tkinter.

B. Create my models-CNN Autoencoder

I built up a convolutional autoencoder (CNN autoencoder) as feature extracting model comparing to some other the existing models. Normally, a CNN autoencoder consists of an encoder, a bottleneck latent space, and a decoder. The encoder is made up of multiple convolutional layers and pooling layers (the conv1l̃ast Max Pool block in figure 5), which is designated to extract latent feature from the input image. The latent layer in the structure is meant to extract and encode the most important features from the extracted feature while discarding redundant information (the latent layer in figure 5). This layer normally reduce the dimensionality to a smaller feature space. Lastly, the structure of decoder reconstruct the compressed features from latent layer, which is meant to provide reconstruction feedback to encoder for optimization.

The structure for my own CNN autoencoder consist of approximately 50 layers of convolutional layers with different sizes of kernel size followed by Max Pooling layers and ReLU activations. My decoder structure mirrors the architecture of encoder but using transposed convolutional layers.

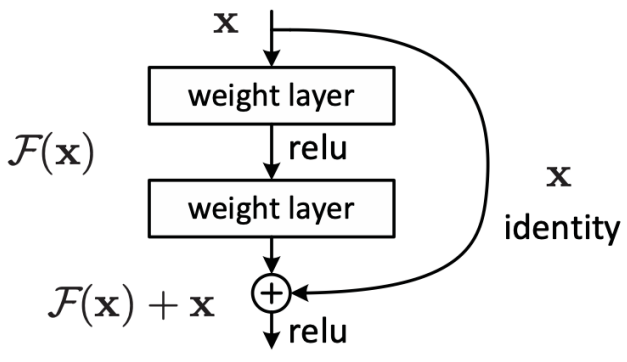

Figure 4. The structure of a Residual block. From *Deep Residual Learning for Image Recognition* (p. 2), by Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, 2015, December 10. [9]

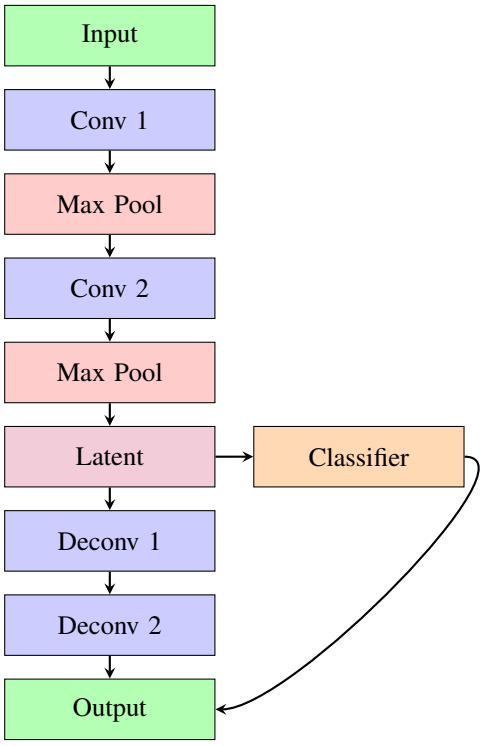

Figure 5. Autoencoder CNN structure with a classifier.

C. Implement Training

Throughout this project, I implement different training depending on different models. The model I use include AlexNet, ResNet50, EfficientNet-b0, Swin Transformer-Tiny, and my own CNN autoencoder.

At first, I use learning rate=0.000001, batch size=64, Adam as optimizer, Cross Entropy Loss function as loss function, and trained for 5l̃0 epochs for all models. Then, I modified the parameters for different models according to the output loss from the first training result. For example, I adjust the learning rate to 0.0001 and batch size to 128 for AlexNet. To get better testing accuracy, I also implement some image augmentation techniques to the dataset. For example, I randomly flip the images vertically and horizontally by 50% or randomly rotated the images by 0 to 45 degree.

D. Apply Similarity Search to the GUI

To perform similarity search to the GUI, a map of feature embeddings of the images in the dataset should be define first. Thus, I chose the model with the highest testing accuracy and modified the

structure into feature extracting only (frozen the classifier layers in the model). With the feature extractor, I create another dataset by implementing feature extraction and labeling for each images in the original dataset. Therefore, all latent vectors along with their labels are stored in the created dataset for the use of mapping and searching.

I can then apply Cosine similarity search with given input image and the feature-vector dataset. Also, I corporate Cosine similarity search method into the GUI (figure 7).

VI. Results

A. Graphical User Interface

I successfully create a GUI that allows users to draw and save images (figure6, 7). Furthermore, it can make prediction and display the predicted image using trained models.

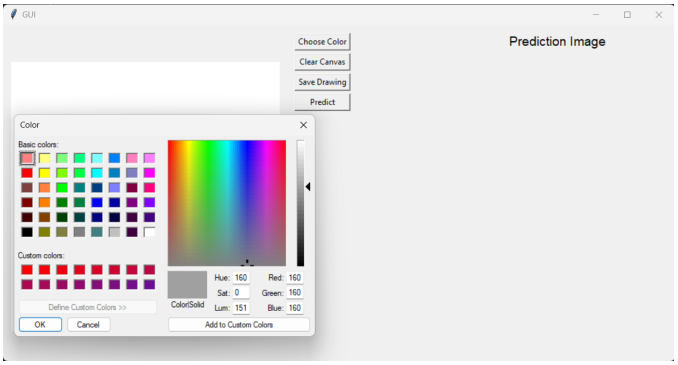

Figure 6. The users can choose their color to draw on the canvas.

Figure 7. The interface of the GUI, users can draw image and get the image with the highest similarity base on the input image and the latent-vector dataset.

Model Name	Best accuracy for the first training attempt	Best accuracy with adjusted parameters	Best accuracy with image augmentation
AlexNet	38.01%	40.472%	0.088%
ResNet-50	1.1%	11.7%	0.09%
My Model	34.69%	0.06%	0.002%
EfficientNet-B0	56.3%	62.1%	0.23%
Swin Transformer-Tiny	25.57%		0.074%

Figure 8. All testing accuracy for all models.

B. Model Training

- **First training attempt**
  In the first training attempt, I used 0.000001 as learning rate, batch size=64, Adam as optimizer, Cross Entropy Loss function as my loss function, and trained the models for 5 to 10 epochs. The reason I chose Adam as optimizer is that it performs better on deep neural network. As for the reason of using Cross Entropy Loss Function is because of its nature of having better performance when it comes to multi-classes dataset.
  However, the testing accuracy is surprisingly bad (first column in figure 8). Nearly all models' testing accuracy are under 50%. And the best accuracy is only around 56% for EfficientNet-b0. As the training loss graph shows in figure 9, the training loss of those model during training session barely converge to 0. What's even worse is that my model cannot even see the pattern of converging.

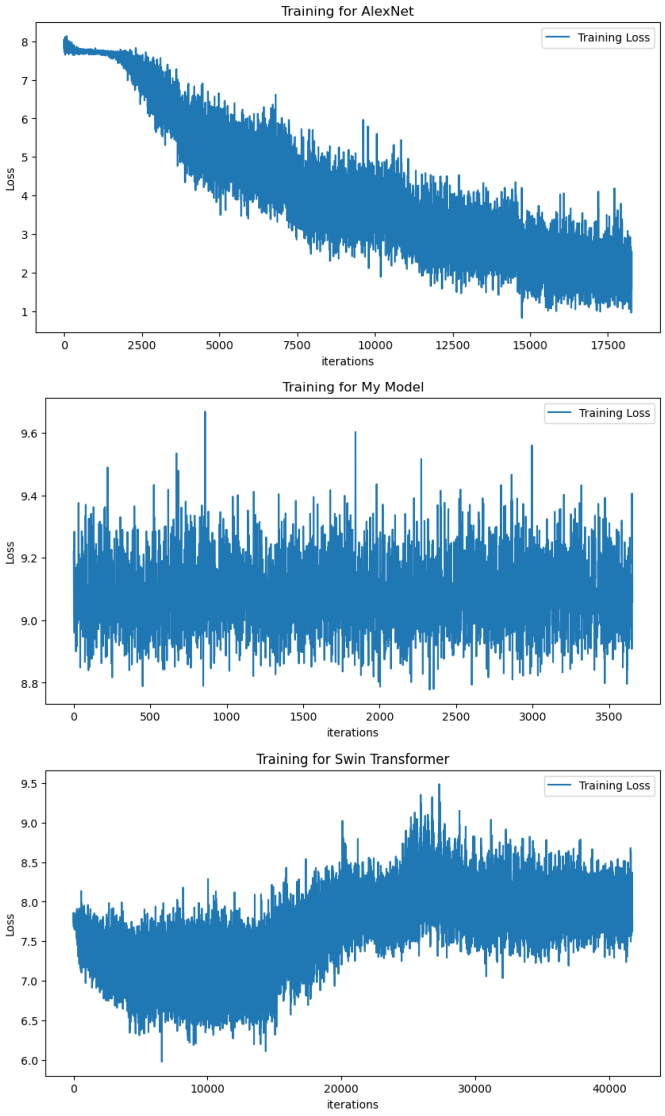

Figure 9. These are the graph of training loss over each iterations during training session. They barely converge to 0 when implement training for AlexNet, My Model, and Swin Transformer.

- **Adjusting parameters**
  With the output training loss from the first training attempt, I adjusted the parameters such as learning rate, batch size, etc, based on different type of models. For example, for ResNet-50, I adjust batch size to 64, learning rate to 0.001, and training for 30 epochs. Although the accuracy is still very low, it finally improved to near 12% (figure 10). Furthermore, I adjusted learning rate to 0.001 and trained for 50 epochs when training AlexNet. The testing accuracy increased to around 40% (figure 11)

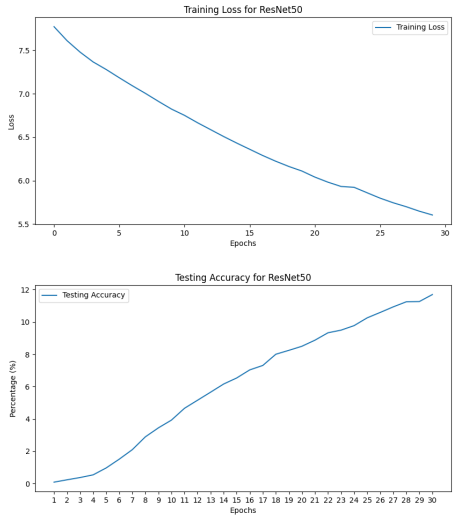

Figure 10. This image shows the training loss versus each epochs for ResNet-50,and it seems to converge.

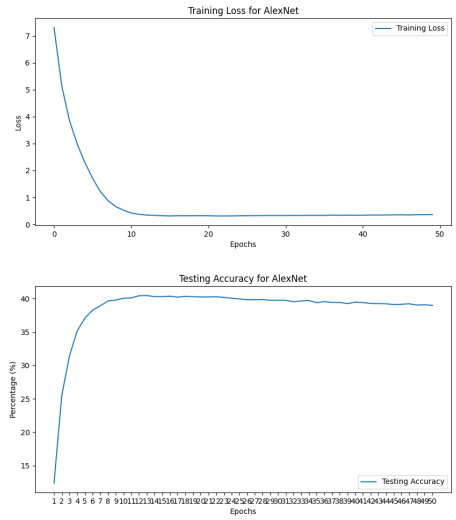

Figure 11. For AlexNet, I adjusted learning rate to 0.001 and trained for 50 epochs, the accuracy increased to 40%.

- **Implement image augmentation**
  Although some models' accuracy got improved after adjusting parameters, the overall performance still not ideal. Therefore, I tried implementing image augmentation to the original dataset. To my surprise, the testing accuracy of all models plummet to nearly 0% (figure 12).

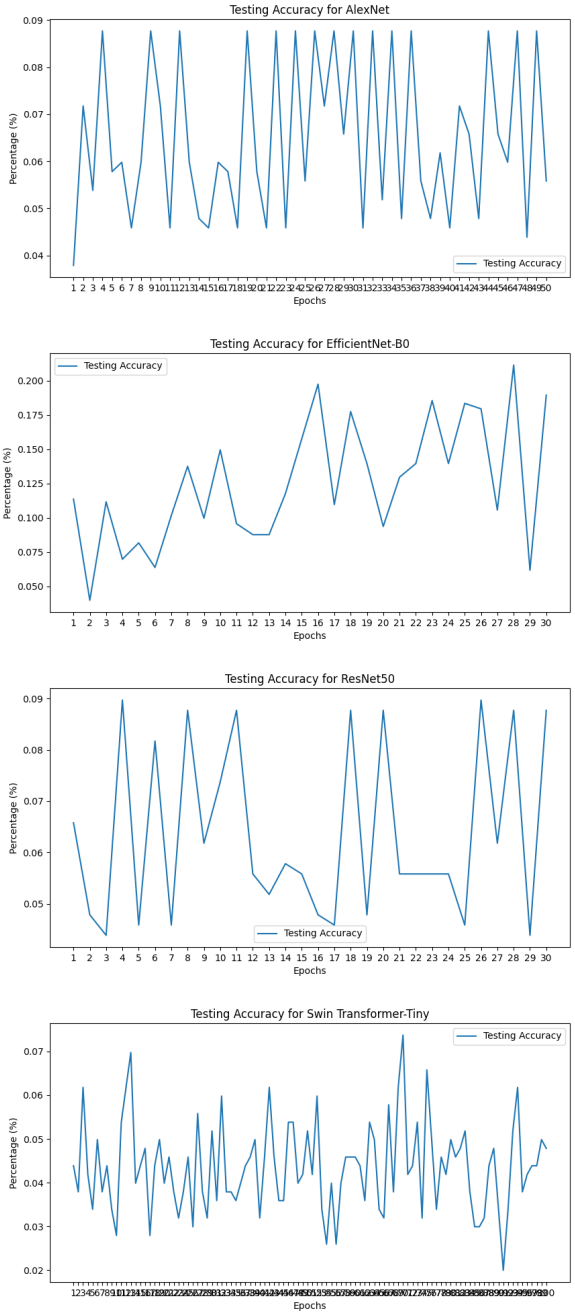

Figure 12. All testing accuracy drop dramatically to near 0% after implementing image augmentation.

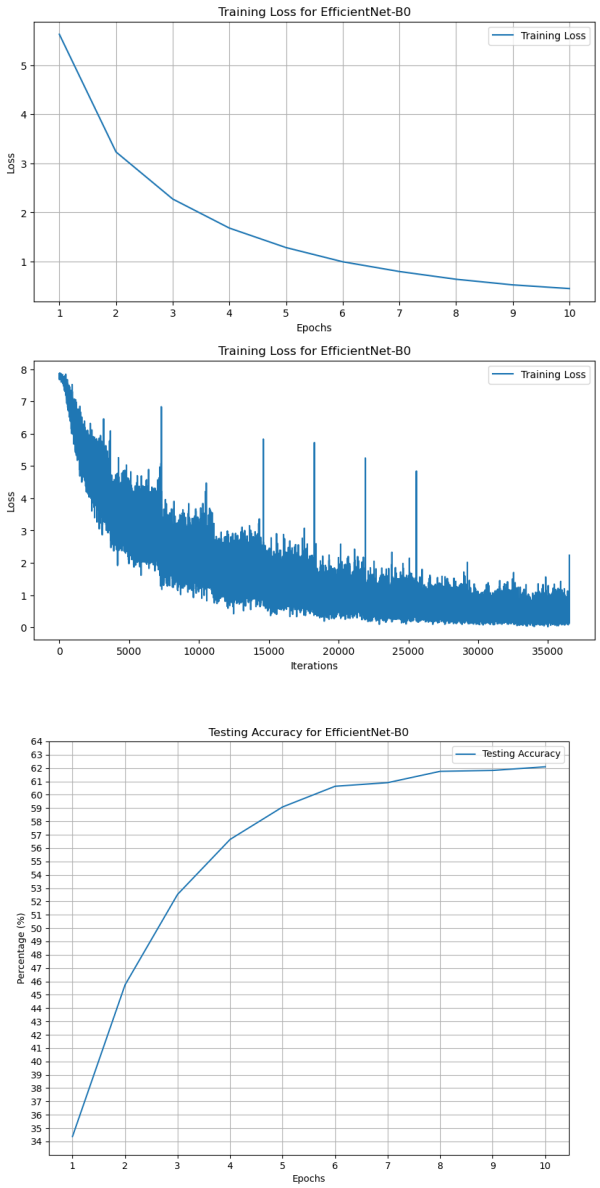

Figure 13. These images are the training result for the best model I got: EfficientNet-b0. The top image is the graph of the training loss over epochs. The middle image is the graph of training loss over each iterations. The image at the bottom is the overall testing accuracy.

- **Best model: EfficientNet-b0**
  With all the training attempts and experiments, the best model I get is EfficientNet-b0 using learning rate=0.0001, batch size=32, and train for 10 epochs. As top two image in figure 13 shows, the training loss of the model converge to nearly 0 after 10 epochs, and the best testing accuracy reaches 60%.

VII. Conclusion

Through this project, I successfully created a graphical user interface that finds the image with the highest similarity according to a dataset. Also, I test some famous deep learning models comparing to my own model.

During this project, the most difficult part in my case is definitely training different models at the same time because of the model size. For example, it took nearly a day only training on Swin Trainsformer and ResNet.

Although this project seems complete, I think there's some improvement to be done in the future. First, I believe that there are more computer vision models that can make more accurate result instead of only 60% of testing accuracy as my result. Second, I think it will be much better if I can add in another function that generate images based on the dataset with my own CNN autoencoder model.

References

[1] Aditi Rastogi (2022, March 14) *ResNet50*

[2] Rajat Tripathi (2023, June 30) *What is Similarity Search?*

[3] Akhsayarka almmyan deka (2023, September 04) *What is Padding in Convolutional Neural Networks (CNN)?*

[4] Dhruv Rawat (2024, March 26) *Stride in Convolutional Neural Network (CNN)*

[5] Bibhu Pala (2018, June 10) Dictionary of CNN

[6] Adrian Rosebrock (2021, May 14) *Convolutional Neural Networks (CNNs) and Layer Types*

[7] PingCAP Team (2014, July 16) *Understanding the Impact of Batch Normalization on CNNs*

[8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov (2014, November 13) *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun (2015, December 10) *Deep Residual Learning for Image Recognition* Microsoft Research

[10] Tuomo Korenius, Jorma Laurikkala, Martti Juhola (2006, November 1) *On principal component analysis, cosine and Euclidean measures in information retrieval*

[11] S. Bunrit, N. Kerdprasop, and K. Kerdprasop (2020, November 4) *Improving the Representation of CNN Based Features by Autoencoder for a Task of Construction Material Image Classification*

[12] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, Baining Guo (2021, 17 August)*Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby (2020, 22 October) *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*