Image Classification Using CNN from Optimization Perspective

Mohammad Mehdi Hosseini, Farida Far Poor

Abstract

Nowadays, convolutional neural networks (CNNs) have grown in many fields of image processing and computer vision to tackle the problem of feature extraction by preserving the strength of neural networks. In this study, we aim to run a CNN model on two popular datasets for image classification from scratch and analyze the accuracy of the model on different optimizer and loss functions.

1. Project

1.1. Introduction

Neural networks: The idea of neural networks stems from the structure of the human brain, where millions of neurons are connected to each other. As Figure 1 shows, a neuron receives voltages from its neighbors through Dendrites, analyze it inside the Nucleus, and generates a voltage by an Axon toward other neurons. In an analogous way, artificial neural networks are structured to receive, analyze, and generate data from the first layer toward the last layer. By having raw data and their expected outputs we can generate a network which can predict the new unseen data. The learning methodology of neural networks is based on back-propagation which refers to passing data from the initial layer toward the last layer (feed-forward), getting an error through calculating the loss function, which is a metric for updating our network weights (in a backward process) for the purpose of updating the weights i a way to be more accurate in the next feed forward step. One of the most important benefits of neural networks in comparison with the other traditional machine learning models, is that it can keep the ability of training the machine with some new data without the essence of the previously used data. However, it could not tackle the problem of feature extraction. To be more specific, just like the other machine learning methods, we should extract some features (like SIFT, HOG, RGB, HSV) to feed to the neural network. This process is boring, time-and-memory consuming, and there is no guarantee about the accuracy. Now, the question is that what if we let the network extract the features.

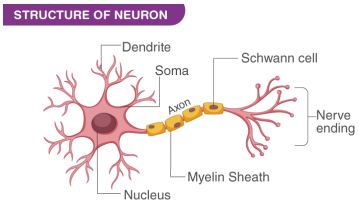

Figure 1. Brain neuron structure (link).

Convolutional Neural Networks: To overcome the drawbacks of feature extraction, convolutional neural networks (CNNs) emerged. They were initially used for image processing tasks, however, nowadays they are useful in several fields. Scientists believe that rather than extracting information from raw data by humans, it is better to let the neural networks learn feature extraction. The idea originates from the convolution process which is popular in image processing. Therefore, researchers merged the idea of convolution with the idea of neural networks to get rid of the feature engineering process along with utilizing the ability of neural networks. The biggest advantage of the features extracted using this method is that they are shift and space-invariant. In other words, the rotation of image is not an obstacle here, and the location of the object is not important (because it looks at the data locally). Figure 2 shows the structure of a CNN model.

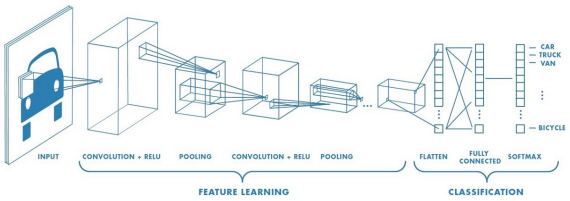

Figure 2. Convolutional neural networks structure (link).

1.2. Image Classification Using CNN

One of the most challenging applications in image processing and machine learning is image classification. Many machine learning approaches, such as SVM, Bayesian, and Neural Networks are proposed to solve its challenges to

some extent. In this study, we are going to design a CNN classifier for image classification and investigate the effect of different loss functions and optimizers on the accuracy of the model. Fortunately, many open-access datasets have been proposed to train and test the classifiers. On the other hand, python libraries (Tensorflow and PyTorch) have facilitated the implementation of different networks.

1.3. Optimization Problem

Any machine learning task is an optimization problem. The goal of machine learning is to minimize the error between the ground truth and predicted label. There are many different methods of loss calculation. We explain three popular loss functions which are used in the experiments in this project:

- Mean Squared Error (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{1}$$

- Cross Entropy:

$$CE = -\frac{1}{n}\sum_{i=1}^{n}[y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)] \tag{2}$$

- Hinge:

$$H(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y \cdot \hat{y}) \tag{3}$$

where y_i is the label of the sample x_i and $\hat{y}_i$ is the predicted label. On the other hand, two of the most popular optimizers in machine learning are:

- Stochastic Gradient Descent (SGD):

$$\theta_{t+1} = \theta_t - \eta\nabla J(\theta_t; x_i, y_i) \tag{4}$$

- ADAM:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{5}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{6}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{7}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla J(\theta_t))^2 \tag{8}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla J(\theta_t) \tag{9}$$

where θ is parameter vector, η is the learning rate, ∇ is the gradient of the model output and the real labels. m_t and v_t are the first and second moment estimates, respectively. $\hat{m}_t$ and $\hat{v}_t$ are bias-corrected estimate. Finally, β_1 and β_2 are the exponential decay rates for the moment estimates.

2. Details and Experiments

2.1. Introduction

Combining the knowledge of machine learning with python libraries lead to significantly good results for image classification. To this end, we also need to collect data and an appropriate hardware equipment. Fortunately, many datasets have been proposed for this goal, and GPUs are tuned to process data as fast as possible. In the following, we initially describe the datasets, talk about our base model, and finally analyze the effect of different loss functions and optimizers on our network.

2.2. Datasets

Two of the most popular datasets in image classification task are *Caltech-101* (Fei-Fei et al., 2004) and *Cifar-100* (Krizhevsky et al., 2009). Caltech-101 consists of pictures of 102 object classes. Each class contains between 40 to 800 images with a variable size, in the range of 200-300 pixels. Cifar-100 has 100 classes containing 600 images per class. The drawback of this dataset is the size of images. Table 1 depicts the information about these datasets. We categorize the data into two sets, including 70 percent as training samples, as well as 30 percent for the test.

Table 1. Caltech-101 and Cifar-100 datasets information.

Dataset	#Classes	#Images	Size
Caltech-101	102	9146	200-300
Cifar-100	100	60000	32×32

2.3. Base Model

As Figure 3 illustrates, we utilized a network with input image size of $256 \times 256 \times 3$ as the input layer. Furthermore, we equipped our network with three convolutional layers with the same filter sizes as 3×3. The first and the third convolutional layers have 64 filters, while the middle one has 128 filters. The output of the convolutional layer is flattened and fed into a dense layer with 256 neurons, before being passed to the last layer with 102 nodes. It is noteworthy that this structure is for Caltech dataset. For Cifar dataset, the initial and the last layers would change to $32 \times 32 \times 3$ and 100, respectively. Figure 4 demonstrates more information about the base model. The number of trainable parameters in this network is around 15 millions. In this network we used ReLU activation function, Sparse Categorical Cross Entropy loss function, max-pooling, Adam optimizer (with learning rate 3×10^{-4}), batch size of 32, and 20 epochs as training parameters of our model. The convergence of this network on Caltech dataset is clarified in Figure 5. The accuracy on the train set is nearly 99 percent, while on

the test set is 86 percent. For the next dataset, Cifar, the accuracy reduces due to the lower quality of the images. On this dataset, our model reached to approximately 95 and 71 percent, on train and test sets, respectively (see Figure 6). Based on the x-axis of Figure 6, more iterations are needed for convergence on Cifar dataset. The reason, as we mentioned earlier, is related to the size (quality) of the input images.

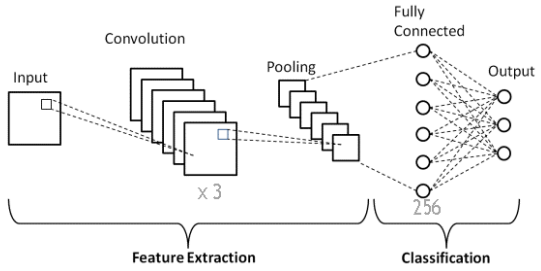

Figure 3. Base model architecture.

Layer (type)	Output Shape	Param #
rescaling (Rescaling)	(None, 256, 256, 3)	0
conv2d (Conv2D)	(None, 254, 254, 64)	1792
max_pooling2d (MaxPooling2D)	(None, 127, 127, 64)	0
conv2d_1 (Conv2D)	(None, 125, 125, 128)	73856
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 128)	0
conv2d_2 (Conv2D)	(None, 60, 60, 64)	73792
max_pooling2d_2 (MaxPooling2D)	(None, 30, 30, 64)	0
flatten (Flatten)	(None, 57600)	0
dense (Dense)	(None, 256)	14745856
dense_1 (Dense)	(None, 102)	26214

Total params: 14,921,510
Trainable params: 14,921,510
Non-trainable params: 0

Figure 4. Base model information

2.4. Parameter Analysis

In this part we changed the parameters that may affect the network performance or computational complexity. Indeed, we analyze the model performance subject to changes in basic parameters of the network.

Learning rate: One of the most critical parameters in machine learning tasks is learning rate. Choosing an inappropriate learning rate may cause your model never converge. A high or a very low learning rate affects computational complexity and model's accuracy. In this experiment, we

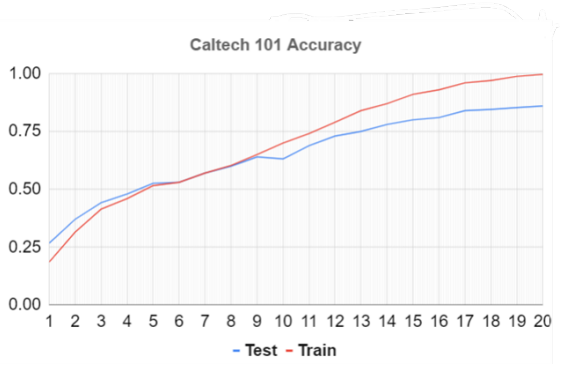

Figure 5. Caltech-101 accuracy after 20 epochs.

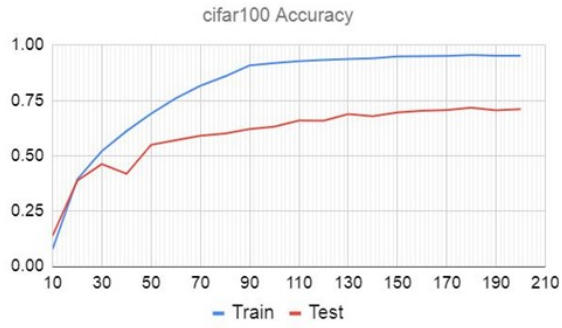

Figure 6. Cifar-100 accuracy after 200 epochs.

divided our basic learning rate (3×10^{-4}) by 10 and investigated the convergence speed through Figure 7. Obviously, smaller learning rate leads to lower speed of convergence. In order to select a good learning rate, we utilized dynamic

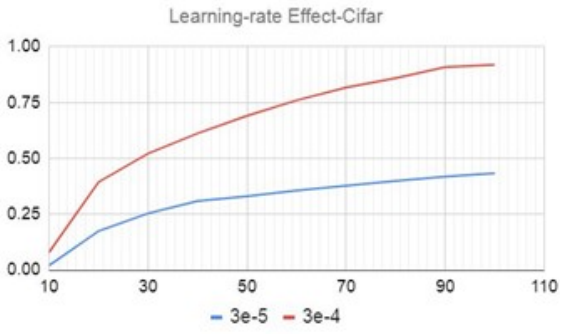

Figure 7. Impact of learning rate on model convergence after 100 epochs on Cifar dataset (optimizer is Adam).

learning rates in the range of [0.5 0.00005] and let the model train only one epoch on each of these learning rates. Figures

?? and 9 show the best range of learning rate on the ADAM optimizer.

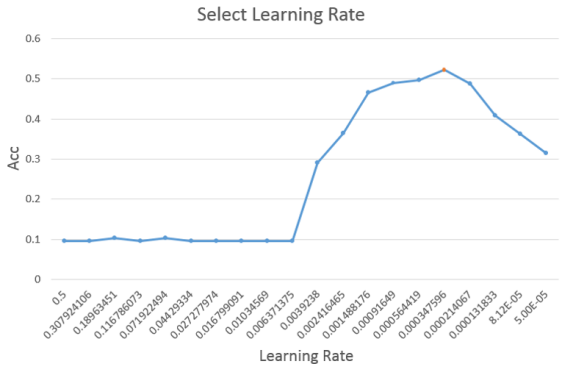

Figure 8. The best learning rate for the Caltech dataset.

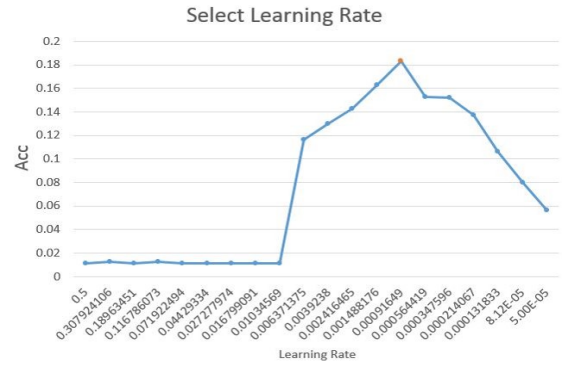

Figure 9. The best learning rate for the Cifar dataset.

Optimizer: Two of the most common optimizers are 'Adam' and 'SGD'. We changed our optimizer from Adam (with learning rate 3×10^{-4}) to SGD (with learning rate 0.01) and as Figure 10 depicts, the final results are very close. However, due to the larger learning rate, SGD converged faster than Adam.

Loss Function: As the final experiment, we tested different loss functions including Cross-Entropy (CE), Mean-Squared (MSE) and Hinge loss on our model. CE is one of the most popular loss functions in classification tasks. However, Hinge loss is designed to tackle the problem of overfitting. Lastly, MSE is a good option for regression tasks, that's why it does not lead to a good accuracy in classification tasks. Figure 11 shows that Cross-Entropy works better in comparison with Hinge loss for our problem. It is notable that due to the aforementioned results, MSE did not provide good results and we ignored it in this figure.

Activation Function: Our base model utilizes ReLU as an activation function. In this experiment we also analyzed the performance on Sigmoid. As Figure 12 reveals, the accuracy of the model is quite the same for both activation functions.

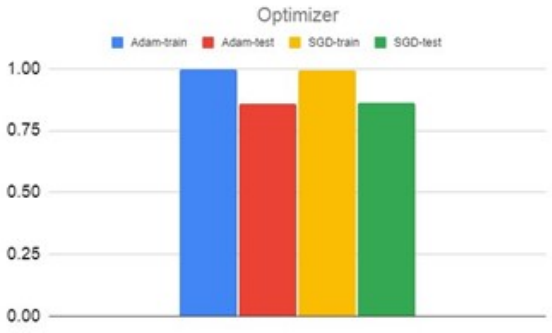

Figure 10. Impact of optimization algorithm on Caltech accuracy.

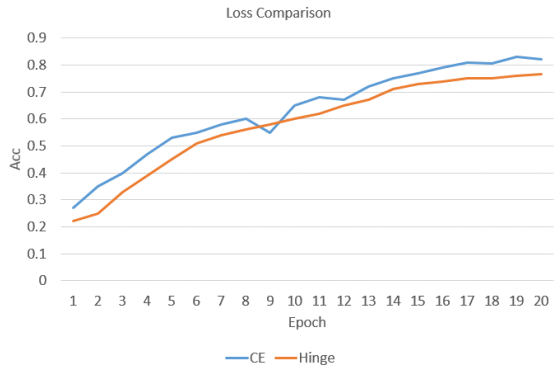

Figure 11. Different loss functions and their accuracy.

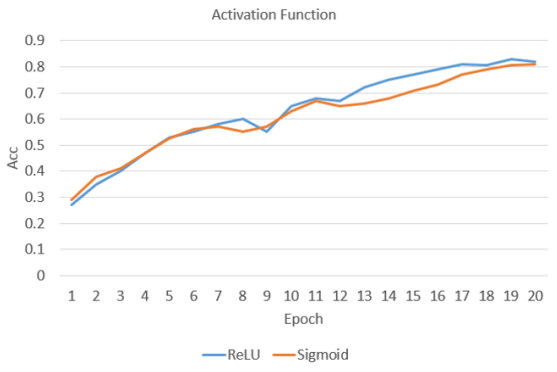

Figure 12. Impact of two activation functions on Caltech.

3. Conclusion

This experiment shows that convolutional neural networks can extract strong features and classify images by an acceptable accuracy. The challenge in this problem is selecting an appropriate loss function, optimizer, and learning rate. Based on the results of this study, Cross-Entropy loss function is a reasonable choice for image classification on caltech and cifar datasets.

References

Fei-Fei, L., Fergus, R., and Perona, P. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Pattern Recognition Workshop*, 2004.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.