Image Classification using CNNs

EQ2425, Project 3

Ata Yavuzyilmaz
atay@kth.se

Shashang Murali
shashang@kth.se

October 14, 2023

Summary

In this project different neural network architectures and training settings are investigated on CIFAR-10 dataset, and the network and training setups are evaluated using the recall metric. If the change made on the network architecture or the training setting increases the recall, we keep this setting to find the configuration with highest success. Our final model configuration got a recall of 0.7844, on the test data.

1 Introduction

The CIFAR-10 dataset consists of many images and each of these images are associated with a class [1]. This dataset has 10 different classes, and we have to find a way to classify the images to their associated labels. You can see some images with their classes in Figure 1.

2 Problem Description

To handle this problem, Convolutional Neural Networks (CNNs) will be used [2].

2.1 Data Preprocessing

We first start with preprocessing the images by normalizing their pixel values between $[-0.5, 0.5]$, so that bigger integers of pixel values don't disrupt the learning process.

Figure 1: Some of the images and their classes from the CIFAR-10 dataset.

Then, the labels are converted from integers to one-hot vectors, so that the output from the CNN can be compared with the original class label.

2.2 Convolutional Neural Network

Convolutional Neural Networks are a type of Neural Networks, but the weights are 2D filters instead of 1D vectors and they are convolved with the input matrix [2]. This filter layer is called the Convolutional layer. After the Convolutional layer, the CNN has a Pooling layer. This layer is basically performs subsampling but unlike subsampling, it takes the maximum value within a specified region. After the Pooling layer, the convolved matrix gets smaller. CNNs have multiple Convolutional layers and Pooling layers back to back, so that the network can capture some of the 2D features within the image.

After a certain number of Convolutional and Pooling layers, the CNN has Fully Connected layers. This is just the normal Neural Network architecture, used for classification. The matrix is flattened to a vector before the Fully Connected layer.

Our model in this assignment has 3 Convolutional and Pooling layers. The Convolutional layers have 24 filters with size $5x5$, 48 filters with size $3x3$ and, 96 filters with size $3x3$ respectively. ReLU function is used after each Conv layer. The pooling layers are after each Conv layer, and their filter size is $2x2$, meaning that they divide the matrix into $2x2$ blocks and get the maximum value in that block to obtain a smaller matrix.

After that, our model has 2 Fully Connected layers with 512 and 10 neurons respectively. The first layer also uses ReLU as activation function and the last layer uses Softmax function so that it can perform multi-class classification. You can see the details of this model in Figure 2.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 28, 28, 24)        1824

 max_pooling2d_3 (MaxPoolin  (None, 14, 14, 24)        0
 g2D)

 conv2d_4 (Conv2D)           (None, 12, 12, 48)        10416

 max_pooling2d_4 (MaxPoolin  (None, 6, 6, 48)          0
 g2D)

 conv2d_5 (Conv2D)           (None, 4, 4, 96)          41568

 max_pooling2d_5 (MaxPoolin  (None, 2, 2, 96)          0
 g2D)

 flatten_1 (Flatten)         (None, 384)               0

 dense_2 (Dense)             (None, 512)               197120

 dense_3 (Dense)             (None, 10)                5130

=================================================================
Total params: 256058 (1000.23 KB)
Trainable params: 256058 (1000.23 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 2: The initial model summary written in Tensorflow and Keras.

2.3 Training and Evaluation

The dataset has 50000 training images and 10000 test images and the performance will be evaluated on the recall of test images after the training. The training is performed with Stochastic Gradient Descent algorithm with learning

rate $10^{-}3$ and with loss function Categorical Cross-entropy, since the target vector is one-hot encoded. The batch size is 64 and the algorithm is ran for 300 epochs.

Then, the network structure and the training parameters will be modified and the recall will be observed. Based on the result, the modification will be kept. We will present the results in next section.

3 Results

The default model given in Section 2 has a test recall of **0.6262**. Now we will change some parameters in the model and see how that changes the test recall. The change will be kept if it improves the model.

3.1 Network Architecture

- **Number of Filters:** Convolutional layer filters are increased from 24, 48, 96 to 64, 128, 256. The test recall increased to **0.6564** so we will proceed with these extra filters.
- **Number of Layers:** An extra Fully Connected layer with 128 neurons is added between the original Fully Connected layers. In this case, our network becomes too complex and overfits, therefore the test recall decreases to **0.6347**. This modification is not kept.
- **Filter size:** The filter sizes of first 2 Convolutional layers are increased from $5x5$ and $3x3$ to $7x7$ and $5x5$, resulting a test recall of **0.6050**.
- **Leaky ReLU:** ReLU activation functions are changed to Leaky ReLU, but the recall has decreased to **0.6227**. ReLU function is 0 when the input is negative, which can be used to eliminate the neurons not affecting the result. This can cause regularization and increase the performance of the model.
- **Dropout:** A Dropout layer is added between the Fully Connected layers. This layer randomly sets some of the input values to 0, in our case with probability 0.3. This also causes regularization and increases the recall to **0.6605**.
- **Batch Normalization:** A Batch Normalization layer is added after each ReLU activation function. This layer normalizes the values by centering and scaling. In this case, the model converges faster and the recall becomes **0.6962**.

The network with best performance is given in Figure 3.

3.2 Training Parameters

- **Batch size:** The batch size is increased from 64 to 256. The recall has decreased to **0.6685**, however the training time also decreased from 2005 seconds to 1225.
- **Learning rate:** Learning rate is increased to 0.1. The model converges even faster in this case with recall **0.7808**.
- **Data shuffle:** Data is shuffled at each epoch, resulting in a recall of **0.7844**.

We obtain better performance when we increase the learning rate and shuffle the data.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 28, 28, 64)        4864

 batch_normalization_4 (Bat  (None, 28, 28, 64)        256
 chNormalization)

 max_pooling2d_3 (MaxPoolin  (None, 14, 14, 64)        0
 g2D)

 conv2d_4 (Conv2D)           (None, 12, 12, 128)       73856

 batch_normalization_5 (Bat  (None, 12, 12, 128)       512
 chNormalization)

 max_pooling2d_4 (MaxPoolin  (None, 6, 6, 128)         0
 g2D)

 conv2d_5 (Conv2D)           (None, 4, 4, 256)         295168

 batch_normalization_6 (Bat  (None, 4, 4, 256)         1024
 chNormalization)

 max_pooling2d_5 (MaxPoolin  (None, 2, 2, 256)         0
 g2D)

 flatten_1 (Flatten)         (None, 1024)              0

 dense_2 (Dense)             (None, 512)               524800

 batch_normalization_7 (Bat  (None, 512)               2048
 chNormalization)

 dropout_1 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 10)                5130

=================================================================
Total params: 907658 (3.46 MB)
Trainable params: 905738 (3.46 MB)
Non-trainable params: 1920 (7.50 KB)
```

Figure 3: The model summary with best performance.

4 Conclusions

In this project, we have written our own Convolutional Neural Network and we have experimented several ways to increase its performance. We have seen that some of the changes increase the performance, while some changes decrease the performance or result in overfitting. Overall, we have seen that increasing the number of filters in the Convolutional layer works for our benefit. Also, some methods like Dropout and Batch Normalization increased the performance. Increasing the learning rate significantly increased the performance since it resulted in faster convergence, and we have also seen that data shuffleing is useful. While designing a Neural Network model, it is important to experiment some different architectures and parameters and to compare the results. Usually we don't have a clear answer on how to design the model, so this project has shown us that we can try until we get the best results.

Appendix

You can see our code and results here.

Who Did What

Ata Yavuzyilmaz - Python code, Project report review
Shashang Murali - Project report, Python code review

References

[1] https://www.cs.toronto.edu/ kriz/cifar.html

[2] A. Krizhevsky, I. Sutskever, G.E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, NIPS, 2012.