EQ2425 Analysis and Search of Visual Data
Project 3 Report: Image Classification using CNNs

Jingxuan Mao jmao@kth.se
Yuqi Zheng yuqizh@kth.se
Aarati Medehal medehal@kth.se

October 16, 2022

Summary

The following Project focuses mainly on building and training a 3-layer Convolutional Network following the given configuration, and ultimately choosing the preferred configuration following parameter modification.
The authors collaborated via Google Colab and used the provided CIFAR-10 dataset which consisted of 50,000 training examples and 10,000 test examples of images from 10 classes as training and testing data. In order to be able to perform evaluation of performance, the labeled test images are used as queries, and the criterion followed is the average top-1 recall rate.

1 Data Pre-Processing

The first step was to perform data pre-processing in order to facilitate smooth analysis of data. In our case, the data is of course the images available in the dataset. We are required to to cast and normalize all the pixel values to the range [-0.5,0.5], which has been depicted below after downloading and storing the dataset locally, following Equation 1, where x is the pixel value.

$$x = \frac{x - mean}{std} \tag{1}$$

```
    train_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[1, 1, 1])
])
test_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[1, 1, 1])
])

# Set the train transform
train_dataset.transform = train_transform
# Set the test transform
test_dataset.transform = test_transform
```

2 Build the Default Model

2.1 Default Network Structure

A three-layer Convolutional Neural Network is built following the stated configuration, consisting of 3 convolutional layers, ReLu, pooling layers, 2 fully

connected layers and a Softmax classifier.

- Convolutional Layer: The major bulk of the computation occurs here, where the filters convolve over the input image resulting in respective feature maps for each filter.
- Stride: denotes the number of pixel shifts the filter has to perform after each convolution computation.
- ReLu: Rectified Linear Unit Function that outputs positive for positive inputs else a zero, helps to avoid exponential computation in a neural network.
- Pooling layer: Reduces the image size to reduce the number of required computations by summarizing the features in the input image (downsampling).
- Fully connected layer: the Feed forward neural network, that performs the classification using the flatten version of input matrix.
- Softmax classifier: Helps to assign probabilities to the classification process. In this project, we train the model with the *CrossEntropyLoss* criterion to compute the cross entropy loss between the input and target, with a Softmax classifier already integrated in the function itself. Therefore, we do not need to add a Softmax layer in the network structure. Otherwise it will affect the accuracy of the model.

The CNN is constructed as below and Figure 1 shows the structure of the network and the number of parameters of each layer with a 32×32 input of 3 channels.

```
def __init__(self, encoded_space_dim, fc2_input_dim):
        super().__init__()

        ### Convolutional section
        self.cnn = nn.Sequential(
            # First convolutional layer
            nn.Conv2d(3, 24, 5, stride=1, padding='valid'),
            nn.ReLU(),
            nn.MaxPool2d(2, stride=2),
            # Second convolutional layer
            nn.Conv2d(24, 48, 3, stride=1, padding='valid'),
            nn.ReLU(),
            nn.MaxPool2d(2, stride=2),
            # Third convolutional layer
            nn.Conv2d(48, 96, 3, stride=1, padding='valid'),
            nn.MaxPool2d(2, stride=2)
        )

        ### Flatten layer
        self.flatten = nn.Flatten(start_dim=1)

        ### Linear section
        self.lin = nn.Sequential(
            # First linear layer
            nn.Linear(2 * 2 * 96, fc2_input_dim),
            nn.ReLU(),
            # Second linear layer
            nn.Linear(fc2_input_dim, encoded_space_dim)
        )

    def forward(self, x):
        # Apply convolutions
        x = self.cnn(x)
```

```
        # Flatten
        x = self.flatten(x)
        # Apply linear layers
        x = self.lin(x)
        return x
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 24, 28, 28]           1,824
              ReLU-2          [-1, 24, 28, 28]               0
         MaxPool2d-3          [-1, 24, 14, 14]               0
            Conv2d-4          [-1, 48, 12, 12]          10,416
              ReLU-5          [-1, 48, 12, 12]               0
         MaxPool2d-6            [-1, 48, 6, 6]               0
            Conv2d-7            [-1, 96, 4, 4]          41,568
         MaxPool2d-8            [-1, 96, 2, 2]               0
           Flatten-9                 [-1, 384]               0
           Linear-10                 [-1, 512]         197,120
             ReLU-11                 [-1, 512]               0
           Linear-12                  [-1, 10]           5,130
================================================================
Total params: 256,058
Trainable params: 256,058
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.47
Params size (MB): 0.98
Estimated Total Size (MB): 1.46
----------------------------------------------------------------
```

Figure 1: Depiction of output shape and number of parameters of each layer

2.2 Default Training Parameters

The built network is optimized using the stochastic gradient algorithm with back propagation. We set the learning rate to 0.001, batch size 64 to and training epochs to 300 by default. The training function is as below.

```
### Training function
def train_cnn(model, device, dataloader, optimizer):
    # Set train mode for CNN
    model.train()
    train_loss = []

    # Iterate the dataloader (we need the label values, this is
                                        supervised learning)
    for image_batch, labels in dataloader:
        # Move tensor to the proper device
        image_batch = image_batch.to(device)
        labels = labels.to(device)
        # Apply CNN to predict
        predictions = model(image_batch)

        # Evaluate loss with Cross Entropy Loss
        loss_fn = nn.CrossEntropyLoss()
        loss = loss_fn(predictions, labels)

        # Backward pass
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        train_loss.append(loss.detach().cpu().numpy())
```

```
    return np.mean(train_loss)
```

The training loss and the testing result of the default CNN are as follows.

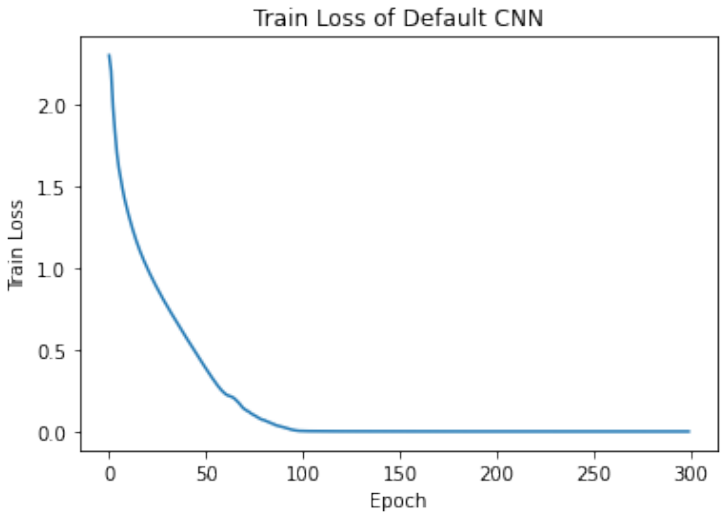


Figure 2: Illustration of Training Loss versus Epoch, recall rate: 0.6932

The achieved recall rate for the default CNN is 0.6932.

3 Network Structure

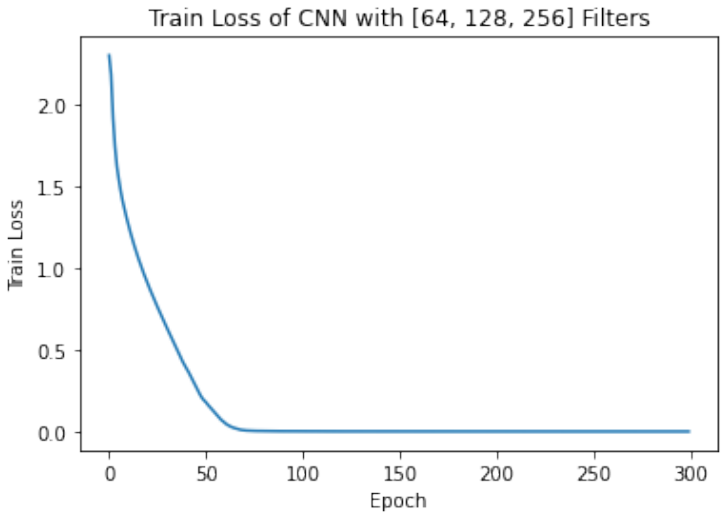
3.1 Number of layers vs. Number of filters

Given tasks are:

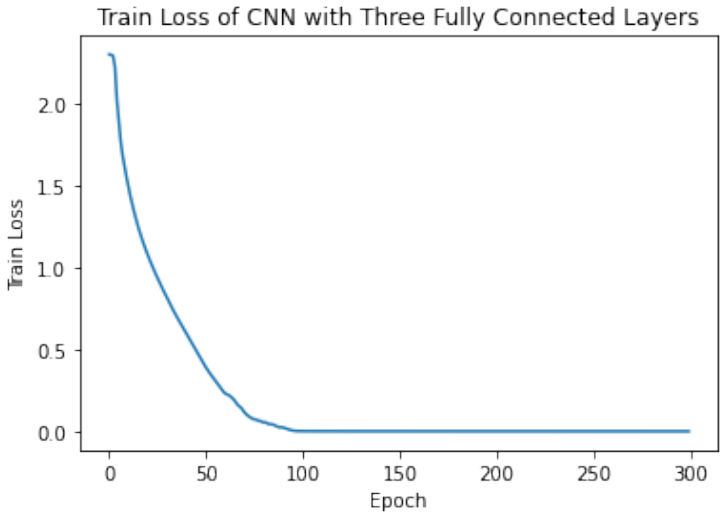
1. Change the default number of filters of three convolutional layers from [24, 48, 96] to [64, 128, 256].

2. Add a fully connected layer between the original fully connected layer 1 and 2 with the number of output units 128, followed by a ReLu activation function.

The results are as follows. We can see that CNN with [64, 128, 256] filters has a higher prediction accuracy of 0.7146 while the addition of the fully connected layer did not improve the model.



(a) Recall rate of CNN with (64, 128, 256) filters: 0.7146



(b) Recall rate of CNN with three fully connected layers: 0.7018

Figure 3: Depiction of training loss against epoch to arrive at the recall rate

3.2 Filter Size

Based on the CNN with [64, 128, 256] filters, the filter size of convolutional layer 1 and 2 is changed to 7×7 and 5×5, respectively. The results are shown below. The recall rate of this CNN is 0.7018, which is not improved.

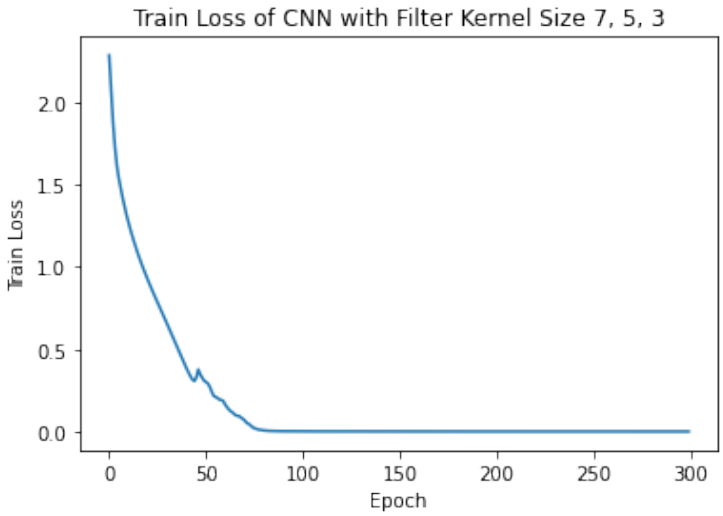


Figure 4: Recall rate of CNN with filter kernel size 7, 5, 3: 0.7003

3.3 ReLu vs. Leaky ReLu

The utilization of Leaky ReLu helps to avoid the vanishing gradient problem (saturating at 0). It works by leaking some positive values to 0 if they are close enough to zero, so as to avoid dividing with 0 and also having the "mean activation" be close to 0 makes training faster. The figure depicts the recall rates on changing all activation functions to leaky ReLu. (Figure 5) The introduction of Leaky ReLu did improve the accuracy of the model to 0.7199.

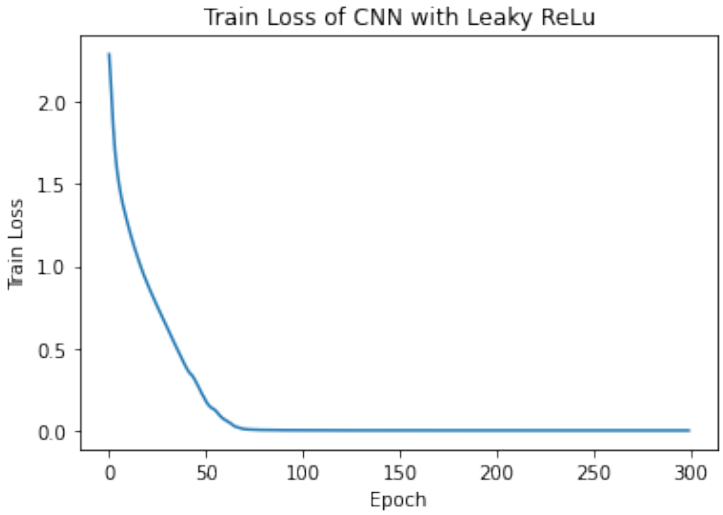


Figure 5: Recall rate of CNN with Leaky ReLu: 0.7199

3.4 Dropout vs. without Dropout

Based on the best configuration above, a dropout regulation is added between the fully connected layer 1 and 2. The dropout rate is set to 0.3.

Dropout randomly drops some neurons in training which helps to prevent overfitting of the model. The recall rate improved noticeably to 0.7358. (Figure 6)

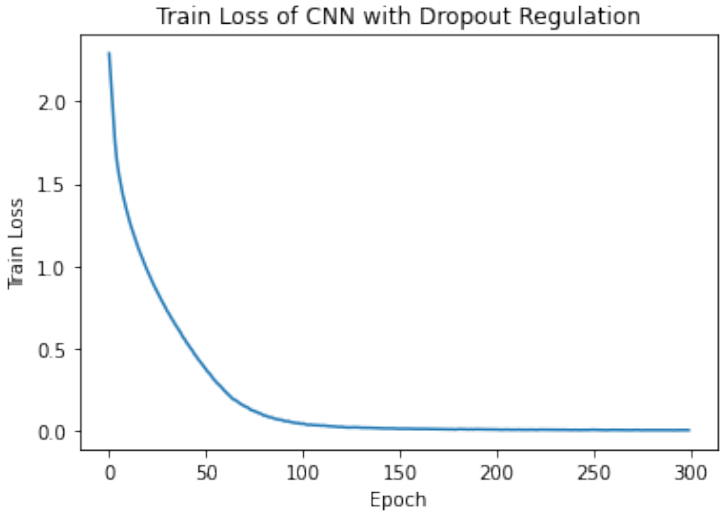


Figure 6: Recall rate of CNN with dropout regulation: 0.7358

3.5 Batch Normalization Layer

> Add batch normalization layer after each activation function ReLu/Leaky ReLu.

Batch normalization keeps the input of each network layer in the same distribution, which avoids the problem of gradient vanishing. The training and testing results are shown below. We can see that the convergence of the training loss is significantly faster than before, with the recall rate improved to 0.7487.

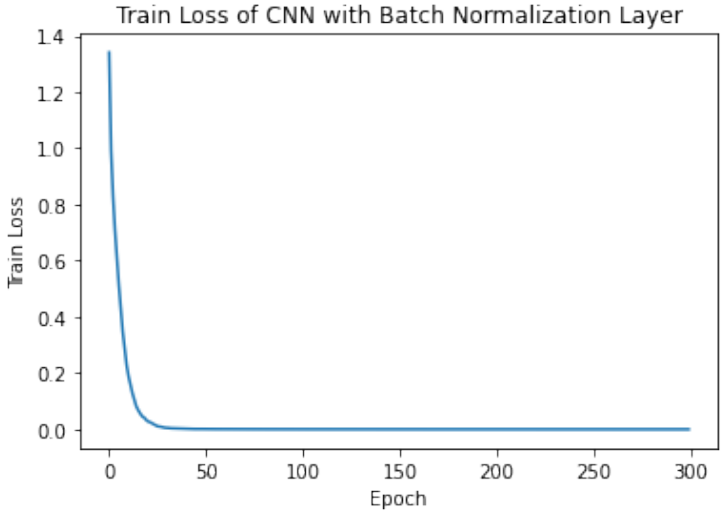


Figure 7: Recall rate of CNN with batch normalization layer: 0.7487

4 Training Settings

Based on the recall rate in section 3, we select the network structure of: filter number[64,128,256], filter size of convolution layer 1, 2 and 3 as 5×5, 3×3 and 3×3, Leaky ReLu with dropout and batch normalization after.

4.1 Batch size

We change batch size from 64 to 256, and the recall rate is shown in Figure 8. The training time was reduced from 1h 15m 44s to 1h 9m 24s, since the batch size became larger. However, a larger training batch size did not result in a better model.

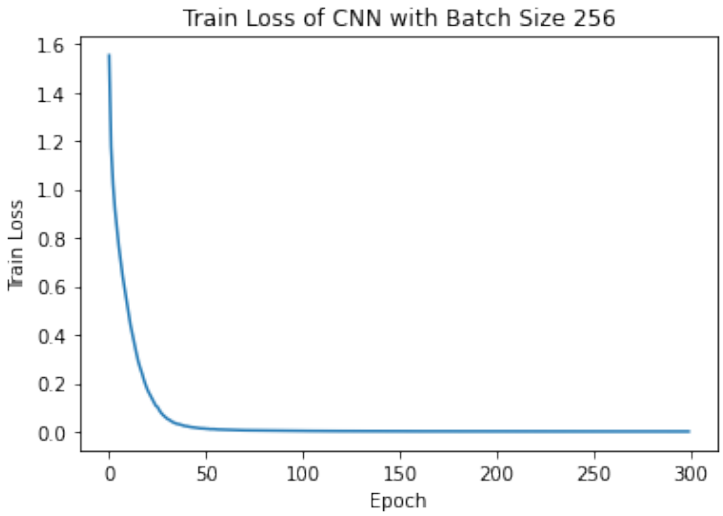


Figure 8: Recall rate of CNN with batch size 256: 0.7236

4.2 Learning rate

We change learning rate from 0.001 to 0.1, and the recall rate is shown in Figure 9. We can see that the training loss converged even faster, but it fluctuated slightly afterwards.

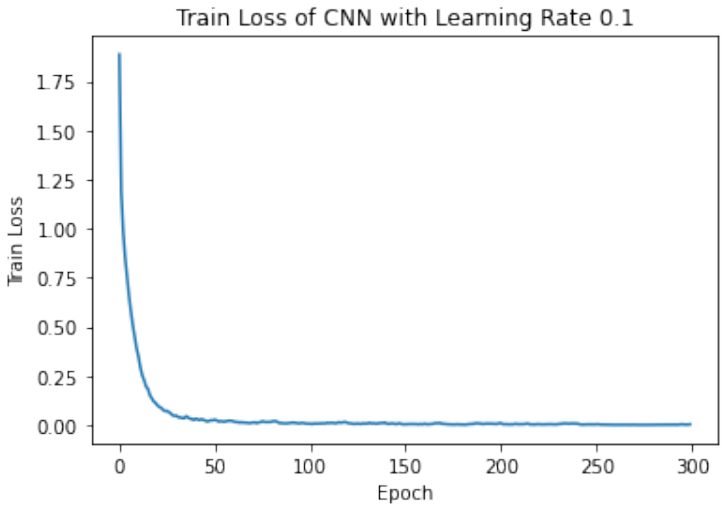


Figure 9: Recall rate of CNN with learning rate 0.1: 0.7391

4.3 Data shuffling

We shuffle the data in each epoch, and the recall rate is shown in Figure 10. The recall rate of 0.7479 is only a bit lower than the highest of 0.7487.

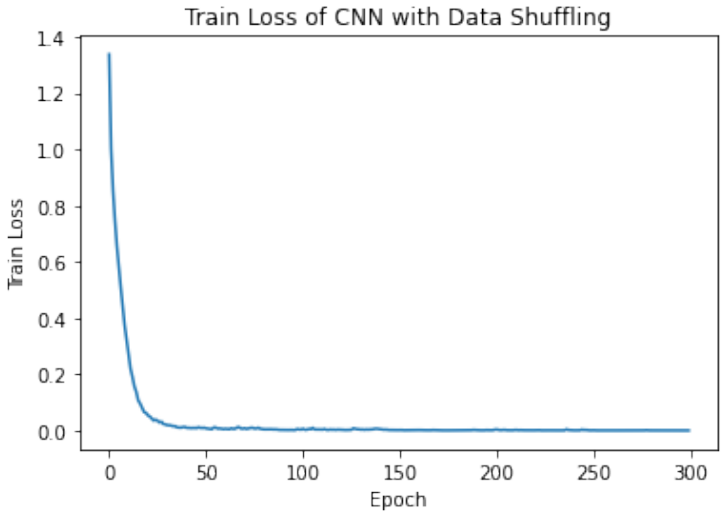


Figure 10: Recall rate of CNN with data shuffling: 0.7479

5 Conclusions

From the results above we can conclude that the network structure of: filter number[64,128,256], filter size of convolution layer 1, 2 and 3 as 5×5, 3×3 and 3×3, Leaky ReLu with dropout and batch normalization is the best structure, while the best training settings are batch size 64 and learning rate 0.001. The largest performance improvement happens when dropout layers are added. This is because the size of the training dataset is small and therefore leads to overfitting. With dropout layer to drop neurons randomly in training process, the model becomes less complex and the problem of overfitting is alleviated.

Appendix

Who Did What

Jingxuan Mao and Yuqi Zheng were involved in the major part of the coding aspect and all 3 members contributed in equal amounts along with helping to write the report.