Università
della
Svizzera
italiana

**Institute of
Computing
CI**

Deep Learning Lab **2021**

Student: Naga Venkata Sai Jitin Jami

Solution for Assignment 2

1. Image Classification Using ConvNets

1.1. Dataset

In this assignment we are working with the CIFAR10 dataset. It consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images, we further divide the 5000 training images into 49000 training images set and 1000 images for validation.

The CIFAR10 dataset is available in PyTorch using the `torchvision.datasets.CIFAR10` command. A sample image from the dataset can be seen on figure 1.

1.1.1. Normalize

We first normalize the whole dataset for all the images but seperately for each of the 3 channels, namely *red*, *green* and *blue*. We use the NumPy library to calculate the mean and standard deviation of the 3 channels.

We use this knowledge to normalize our dataset using `torchvision.transforms.Normalize` function to normalize the related tensor data for all 60000 images. The mean and standard deviation values can be found on table 1

1.2. Model

For this assignment we have created 2 different Convolutional networks. The details of the first ConvNet are given in a sequential manner, as follows:

- Convolutional Layer 1, 32 filters, 3 × 3, ReLU activation. Output: 32 × 30 × 30

Channel	Mean	Std
Red	0.49139968	0.24703233
Green	0.48215827	0.24348505
Brown	0.44653124	0.26158768

Table 1: Mean and Std values

- Convolutional Layer 1, 32 filters, 3 × 3, ReLU activation. Output: 32 × 28 × 28
- Max Pooling Layer, 2 × 2. Output: 32 × 14 × 14
- Convolutional Layer 1, 64 filters, 3 × 3, ReLU activation. Output: 64 × 12 × 12
- Convolutional Layer 1, 64 filters, 3 × 3, ReLU activation. Output: 64 × 10 × 10
- Max Pooling Layer, 2 × 2. Output: 64 × 5 × 5
- Fully Connected Feed Forward Layer, ReLU activation, Input: 64 × 5 × 5 = 1600. Output: 512
- Fully Connected Feed Forward Layer, ReLU activation, Input: 512. Output: 10
- Softmax output layer

The details of the second ConvNet are given in a sequential manner, as follows:

- Convolutional Layer 1, 32 filters, 3 × 3, ReLU activation. Output: 32 × 30 × 30
- Convolutional Layer 1, 32 filters, 3 × 3, ReLU activation. Output: 32 × 28 × 28
- Max Pooling Layer, 2 × 2. Output: 32 × 14 × 14
- 2D Dropout Layer. Output: 32 × 14 × 14
- Convolutional Layer 1, 64 filters, 3 × 3, ReLU activation. Output: 64 × 12 × 12
- Convolutional Layer 1, 64 filters, 3 × 3, ReLU activation. Output: 64 × 10 × 10
- Max Pooling Layer, 2 × 2. Output: 64 × 5 × 5
- 2D Dropout Layer. Output: 64 × 5 × 5
- Fully Connected Feed Forward Layer, ReLU activation, Input: 64 × 5 × 5 = 1600. Output: 512
- 1D Dropout Layer, Input: 512. Output: 512
- Fully Connected Feed Forward Layer, ReLU activation, Input: 512. Output: 10
- Softmax output layer

1.3. Training and Validation

A training pipeline was created for both CNNs. For the first CNN, the training was done for various learning rates. The second CNN, the training was done for various learning rates and various dropout probabilities. The following are the *hyperparameters* for the first CNN:

- Learning Rates: [0.001,0.0015,0.002,0.0025,0.003,0.035,0.004]
- Momentum: 0.9
- Batch Size: 32
- Number of epochs: 20
- Loss: Cross Entropy Loss

The following are the *hyperparameters* for the second CNN with varying learning rates:

- Learning Rates: [0.001,0.0015,0.002,0.0025,0.003,0.035,0.004]

- Dropout probability: 0.05
- Momentum: 0.9
- Batch Size: 32
- Number of epochs: 30
- Loss: Cross Entropy Loss

The following are the *hyperparameters* for the second CNN with varying dropout probabilities:

- Learning Rate: 0.002
- Dropout probabilities: [0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5]
- Momentum: 0.9
- Batch Size: 32
- Number of epochs: 30
- Loss: Cross Entropy Loss

1.4. Results

1.4.1. CNN1

In the first case, we have multiple learning rates with 20 epochs. We now plot the evolution of training accuracy, validation accuracy and also the testing accuracy for all different learning rates. These plots can be found in figure 2.

It can be seen that for learning rate = 0.0035, it can be seen that the training and validation accuracy don't increase as the model evolves. This could be because the model might have missed the local minima and it kept overshooting the target while optimizing. But as expected, in all the other cases the training accuracy increases as the mode evolves. The accuracy reaches 100% fastest with learning rate = 0.004, but the difference seems negligible to the naked eye.
The validation accuracy on the other hand increases with training accuracy upto a certain point and flat lines at around 73% indicating that the model is clearly overfit to the training data since training accuracy is constantly increases where as the validation accuracy tops out at 73%. In figure 2c, we can see that performance for learning rate = 0.004 has the best performance.

1.4.2. CNN2

In the second case, we have multiple learning rates with 30 epochs. We now plot the evolution of training accuracy, validation accuracy and also the testing accuracy for all different learning rates. These plots can be found in figure 3.

It can be seen that for learning rate = 0.0035, it can be seen that the training and validation accuracy don't increase as the model evolves. This could be because the model might have missed the local minima and it kept overshooting the target while optimizing. But as expected, in all the other cases the training accuracy increases as the mode evolves. The accuracy reaches 100% fastest with learning rate = 0.004, but the difference seems negligible to the naked eye.
The validation accuracy on the other hand increases with training accuracy upto a certain point and flat lines at around 75% indicating that the model is clearly overfit to the training data since training accuracy is constantly increases where as the validation accuracy tops out at 75%. In figure 5c, we can see that performance for learning rate = 0.003 has the best performance.

1.4.3. CNN1 vs CNN2

Here we want to compare the testing accuracy and see if we had any performance gain by implementing dropout layers in the second CNN. The results can be seen in 4.

As can be seen, there is a slight improvement in performance due to inclusion of a dropout layer for all learning rates. The advantages of dropout layers for overfit model are discussed later in the report.

1.4.4. CNN3

In the third case, we have multiple dropout probability values with 30 epochs. We now plot the evolution of training accuracy, validation accuracy and also the testing accuracy for all different dropout probability values. These plots can be found in figure 5.

In all the cases the training accuracy increases as the mode evolves. The accuracy reaches 100% fastest with p = 0.2. The validation accuracy on the other hand increases with training accuracy upto a certain point and flat lines at around 75% indicating that the model is clearly overfit to the training data since training accuracy is constantly increases where as the validation accuracy tops out at 75%. In figure **??**, we can see that performance for p = 0.2 has the best performance, but the performance is pretty high for lower p values and drops off significantly as p value increases.

1.4.5. Best Hyper-parameters

Following the investigations conducted in the previous sections, the following hyper-parameters have been chosen for the best case scenario:

- Learning Rates: 0.003
- Dropout probability: 0.2
- Momentum: 0.9
- Batch Size: 32
- Number of epochs: 30
- Loss: Cross Entropy Loss

The validation and training accuracy can be found on the plot on figure 6. The testing accuracy for the best case scenario was **75.87%**.

2. Questions

2.1. Question1: Soft-Max Layer

The softmax function is a function that turns a vector of K real values into a vector of K real values that sum to 1. The input values can be positive, negative, zero, or greater than one, but the softmax transforms them into values between 0 and 1, so that they can be interpreted as probabilities.
The softmax layer is usually the last layer as it helps with classification especially in multi-class classification since it boils down the output of a NN layer to probabilities, the neuron with the highest probability will get the classification for that sample.

$$\sigma(\vec{z})_i = \frac{\exp z_i}{\sum_{j=1}^{K} \exp z_i} \tag{1}$$

2.2. Question2: Momentum in SGD

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another, which are common around local optima. In these scenarios, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum.
Momentum is a method that helps accelerate SGD in the relevant direction and dampens oscillations. It does this by adding a fraction γ of the update vector of the past time step to the current update vector:

$$\nu_t = \gamma\nu_{t-1} + \eta\nabla_\theta J(\theta) \tag{2}$$

$$\theta = \theta - \nu_t \tag{3}$$

The γ in equation 2, is the momentum parameter in `torch.optim.SGD`.

2.3. Question3: CNN vs FFNN

In a feed forward neural network, all the parameters are connected to all neurons in an NN layer. This leads to a structure agnostic property, i.e. there are no assumptions made about the input.

In a CNN, there is an explicit assumption that the inputs are images. This assumption allows us to model certain properties into the model, like : Convolutional Layers, Pooling Layers and Dropout layers.

2.4. Question4: Applications of 1D CNNs

1D CNNs have the applications in time series problems. The following are 2 such problems:

- Real-time Electrocardiogram (ECG) Monitoring
- Vibration-Based Structural Damage Detection in Civil Infrastructure

2.5. Question5: Dropout

Dropout is a machine learning technique where you remove (or "drop out") units in a neural net to simulate training large numbers of architectures simultaneously. Importantly, dropout can drastically reduce the chance of overfitting during training.

3. References

1. https://stackoverflow.com/questions/66678052/how-to-calculate-the-mean-and-the-std-of-
2. https://deepai.org/machine-learning-glossary-and-terms/softmax-layer
3. https://ruder.io/optimizing-gradient-descent/index.html#fn4
4. https://medium.com/swlh/fully-connected-vs-convolutional-neural-networks-813ca7bc6ee5
5. https://arxiv.org/pdf/1905.03554.pdf
6. https://wandb.ai/authors/ayusht/reports/Implementing-Dropout-in-PyTorch-With-Example--

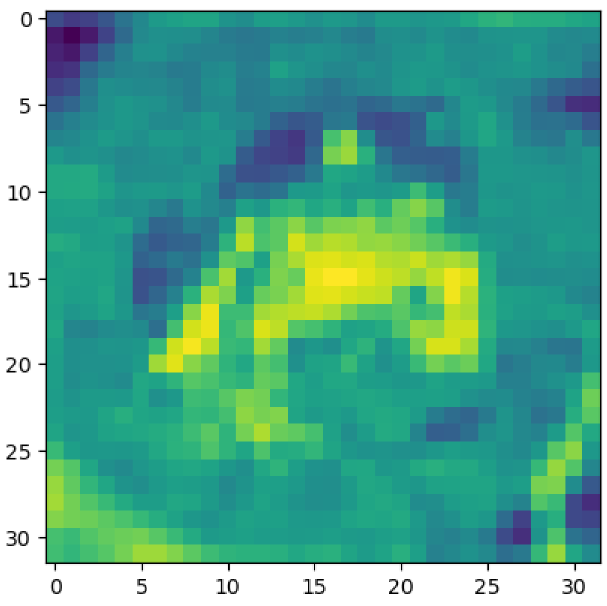
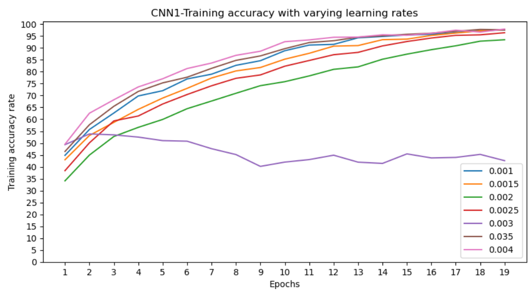
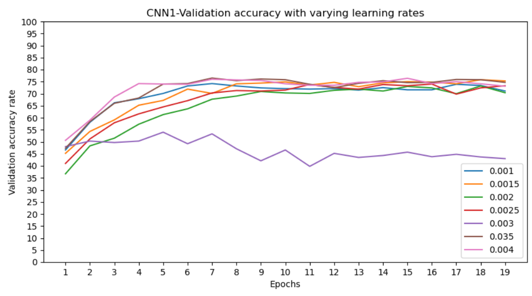


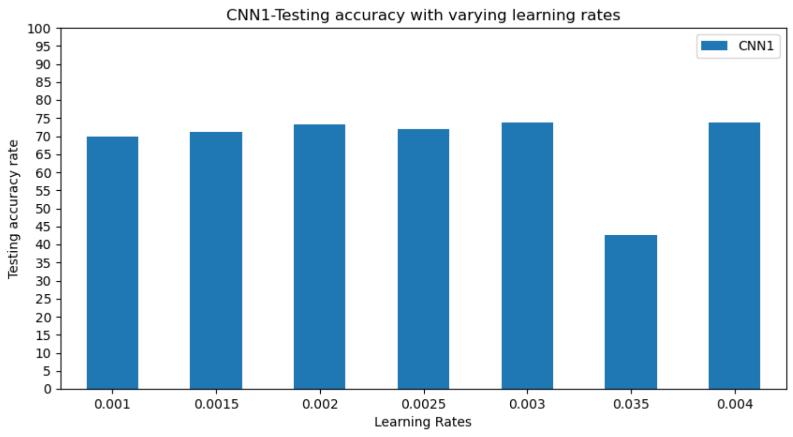
Figure 1: Sample Image



(a) Training Accuracy of first CNN

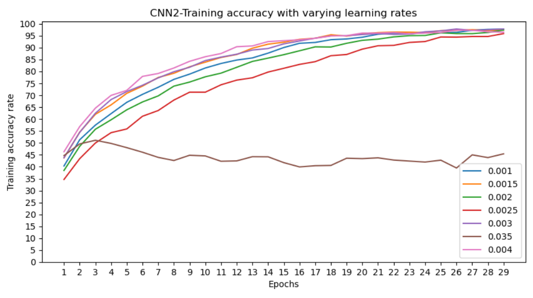


(b) Validation Accuracy of first CNN

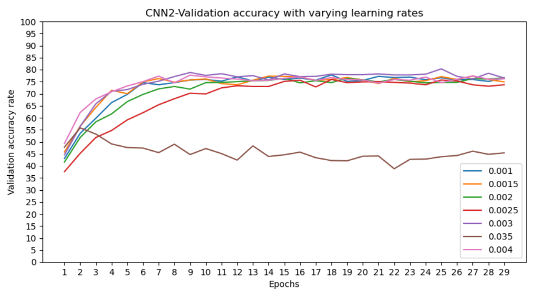


(c) Testing Accuracy of first CNN

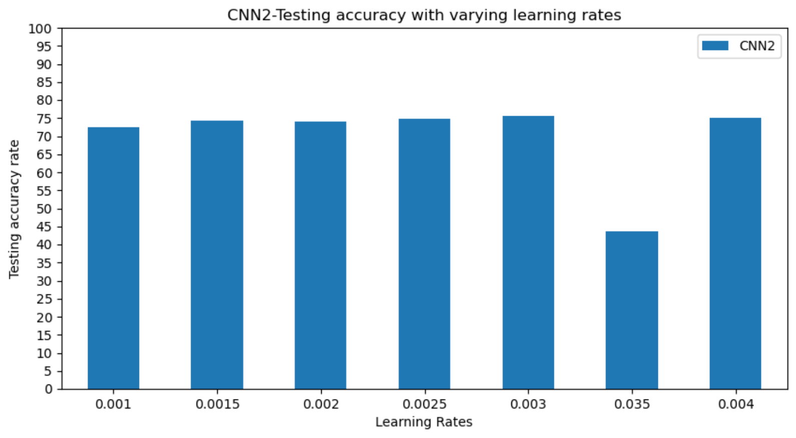
Figure 2: CNN1



(a) Training Accuracy of second CNN



(b) Validation Accuracy of second CNN



(c) Testing Accuracy of second CNN

Figure 3: CNN2

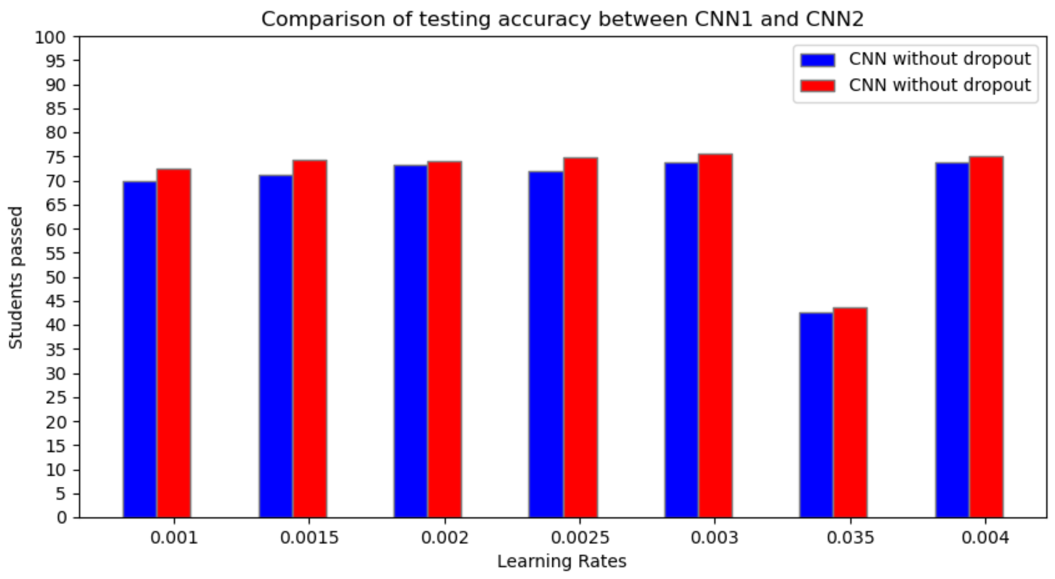
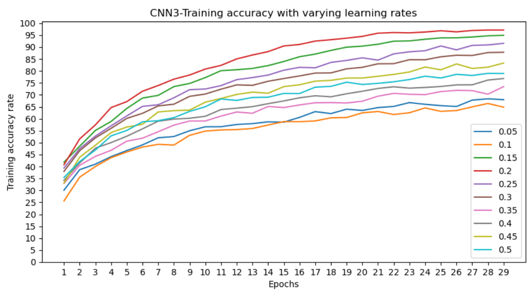
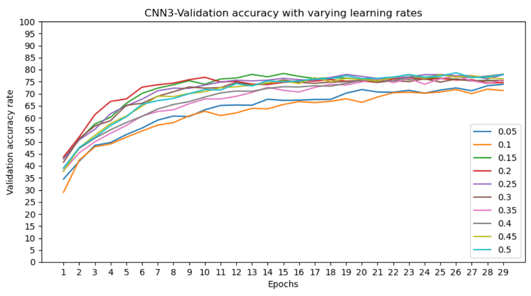


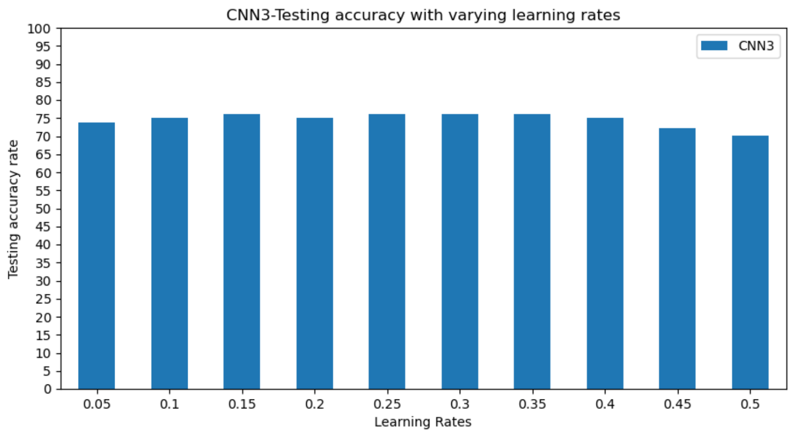
Figure 4: Comparison between CNN1 and CNN2



(a) Training Accuracy of third CNN



(b) Validation Accuracy of third CNN



(c) Testing Accuracy of third CNN

Figure 5: CNN3

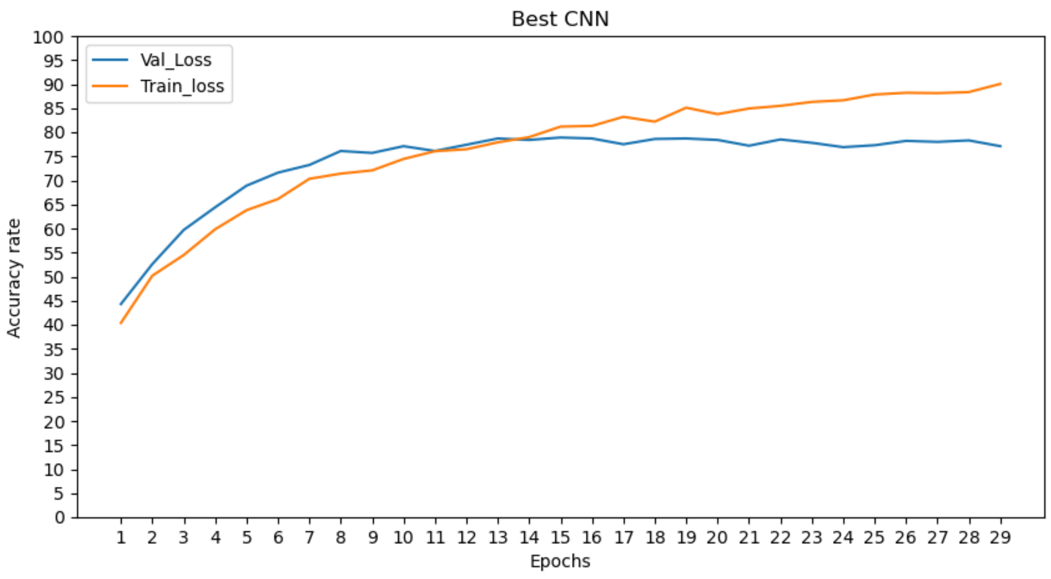


Figure 6: Best CNN