Deep Learning HW2

Classification using Convolutional Neural Networks

CNNs have revolutionized the field of computer vision, their unique architecture, inspired by the visual cortex, allows CNNs to learn intricate patterns in images—detecting edges, textures, and even complex features at multiple levels.

In this assignment we will explore the power of different CNN architectures and how to apply them for learning state of the art image classification algorithms.

This assignment is divided into two parts. In the first part, you will implement a simple CNN in different ways and compare the performance on image classification.
In the second part, we will use a more powerful network to achieve an almost perfect score on a real life dataset.

Part 1: (CNNs on CIFAR-10)

In this part of the assignment, you will design, implement, and experiment with Convolutional Neural Networks (CNNs) on the CIFAR-10 dataset.
The goal is to understand how different configurations (number of layers, kernel sizes, strides, and pooling techniques) impact model performance on image classification tasks.

The CIFAR-10 dataset consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images.

Load the dataset using torchvision-datasets library.
Split the dataset into training, validation and test sets if not already provided like that, and showcase some samples from different classes from the training set.

The parameter search should be done on the validation set and then used to test the best model found on the test set.

You are given 4 tasks where you will implement or add stuff on previous tasks, then compare the results and explain them.

Task 1:

Create a simple CNN with 2 convolutional layers and 1 fully connected layer with a softmax output such that:
Conv Layer 1 contains 16 filters of 3x3 kernels, stride 1, padding 1.
Conv Layer 2 contains 32 filters of 3x3 kernels, stride 1, padding 1.
Max Pooling after each Conv layer
Flattening followed by a fully connected layer
Use Relu as an activation function on the convolutional layers, Adam or SGD as an optimizer and Cross Entropy as a loss function.

Train the network on 10 epochs and test it.

Task 2:

Increase the depth of the network by adding 2 additional layers.
The first additional one with 64 filters and the second additional one with 128 with the same kernel sizes as before.

Train and test the network and compare it with the previous ones.

Task 3:

Remove the 2 new layers.

Increase the kernel sizes to 5x5 and the stride to 2.

Train and test the network and compare it with the previous.

Task 4:

Change Max pooling to average pooling on the **best** network so far and observe the changes.

Train and test the network and compare it with the previous best.

Part 2: (Weather Classification)

In this task, you'll explore using a powerful deep learning model for classifying weather conditions based on image data. Weather classification can be valuable in various applications, such as in automated weather monitoring, agricultural planning, and disaster response. By leveraging a powerful model that's known for its effectiveness in image recognition tasks, you'll gain experience with advanced architectures that achieve high accuracy on real world images.

The dataset consists of images depicting different weather conditions, each labeled with one of 4 distinct classes: Sunrise, Shine, Cloudy, Rain. Your goal is to train a model to recognize and classify these weather conditions with high accuracy.

Download the train and test datasets from the website.
The training data is given in 4 separate zip files due to space constraints in the website, download them and download the test data, preprocess it how you see fit and showcase a couple of images from each of the classes with it's corresponding label.

Use any one of the model architectures we talked about in class, load a pre trained version from pytorch and adapt it to classify the 4 weather classes. Fine-tune the model to maximize its performance on the dataset.

Loss Function:
In class, we showcased different types of loss function, create a custom class called:

```
Class MyLossFunction(*arguments*):
        *write your code here*
```

Write your custom loss function mathematically using Latex in a text cell above the code cell in the notebook and implement your own custom loss function, you may not use a pre-built loss function from pytorch, even if it's a regular CE/MSE function.

After training the model, evaluate it on the test set.

When evaluating on the test set, show a handful of images with their original label and the predicted label.

Also, show the results of the network on the test set using:
1) test set accuracy percentage (like we always do)
2) Confusion matrix

You may use any regularization techniques that you want, the goal is to achieve the highest accuracy that you can.

Plot the training, validation and testing accuracy as well as the loss function values as a function to the number of epochs.
The training and validation plots should be showcased in one plot together.

For the final step, use any projection algorithm we talked about in the class to showcase (visualize) how the model classified the data on a 2D plane.

Submission:-

For this assignment, make sure your submission is clear and well-organized.
You should include a notebook for each part of the assignment with all code cells run and explained.
There's no need to put the files in a zip file, just submit your 2 ipynb files with their names being

HW2_PT1_ID1_ID2.ipynb & HW2_PT2_ID1_ID2.ipynb

Add text cells to introduce each section, explaining what you did and why. Keep your notebooks tidy and easy to follow, with the names of all group members at the top.

In each notebook, include comments in your code and use text to describe what you're doing in each section. Visualize your results with plots that help explain the data and model performance.

Finally, print each model you use, showing the layer details and the total number of parameters.

Only the pytorch library can be used as a deep learning tool to work with the models, you can’t use keras or tensorflow under any circumstances, you may use other libraries for manipulating the dataset and preprocessing as long as it doesn’t affect the model definition within pytorch.