ET3112 Assignment 03 on Convolutional Neural Networks

Sampath Perera

April 10, 2024

1. Your task is to build an image classifier using convolutional neural network (CNN). You have the freedom to use any programming language and toolkit of your choice. You may use the Python programming language along with TensorFlow and Keras or PyTorch to construct the convolutional neural network (CNN).

2. Prepare your dataset: Download the CIFAR-10 dataset. This dataset contains 60,000 color images in 10 different classes. Documentation of this data set can be found in CIFAR-10 dataset. You may use listing 1 to load data. Further, apply suitable feature scaling. Refer this for more information "pytorch normalization".

3. Split the dataset into training, validation, and testing subsets using a ratio of 60% for training and 20% each for validation and testing sets[1]

```
#  for pytorch
import torchvision
import torchvision.transforms as transforms

# Define data transformations (optional, but recommended)
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

# Load the training dataset
train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True, transform=transform, download=True)

# Load the testing dataset
test_dataset = torchvision.datasets.CIFAR10(root='./data', train=False, transform=transform, download=True)

#  for keras

from keras.datasets import cifar10

# Load the CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

```

Listing 1: Data loading.

4. Build the CNN model: A common CNN design consists of interleaving convolutional and max-pooling layers, ending with a linear classification layer [4]. This pattern is illustrated in Figure 1[2], and it was inspired by Fukushima's neocognitron [1] and Hubel and Wiesel's work on human visual cortex [2]. Yann LeCun's LeNet model refined this approach in 1998, popularizing it through backpropagation and SGD [3].

[1] If data set is too large to process, you may use portion of it.

[2] Image adapted from https://blog.floydhub.com

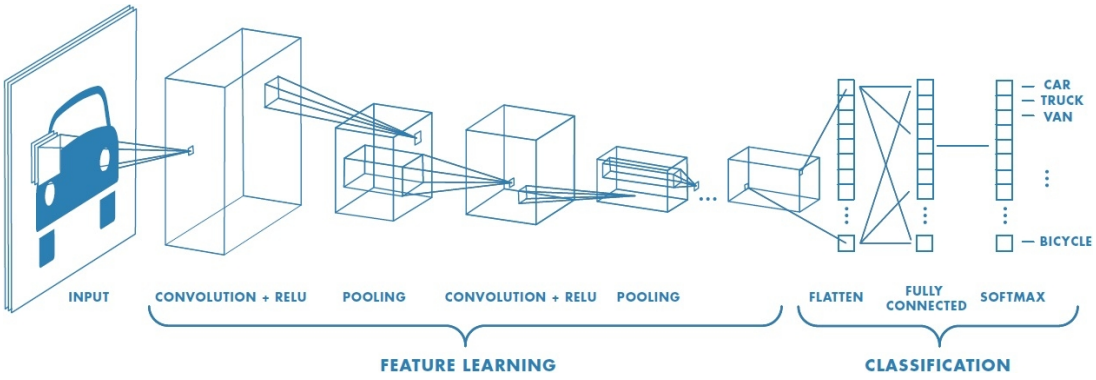

Figure 1: A simple CNN for image classification.

In this example (Figure 1), we exclude normalization layers from the CNN design because the model is relatively simple and not very deep. Here, the benefits of normalization layers may not be visible. However, in deep neural networks, normalization layers, such as batch normalization or layer normalization, are often used to improve training stability. They help mitigate issues like vanishing gradients and can make it easier for deep networks to converge during training. Refer tensorflow batch normalization and pytorch batch normalization for more information.

A basic CNN architecture is given below. Feel free to modify this network by adding more layers.

- A Convolutional layer with x_1 filters, a $k_1 \times k_1$ kernel, and 'relu' activation.
- A MaxPooling layer.
- Another Convolutional layer with x_2 filters, a $k_2 \times k_2$ kernel, and 'relu' activation.
- Another MaxPooling layer.
- Flatten the output.
- A fully connected layer with x_3 units and 'relu' activation.
- Add dropout with a rate of d to reduce overfitting.
- An output layer with 10 units (for 10 classes) and 'softmax' activation.

5. Determine the parameters of the above network such as kernel sizes, filter sizes, size of the fully connected layer and dropout rate.

6. Train the model: Train the model using the training data for 20 epochs and plot training and validation loss for with respect to epoch. Here, for the optimizer you may use adam and sparse categorical crossentropy as the loss function. Set a suitable learning rate. Refer this page to see available optimizers keras optimizers and pytorch optimizers. More information about optimizer can be found in https://cs231n.github.io/neural-networks.

7. Why we have chosen sparse categorical crossentropy as the loss function?
8. Evaluate the Model: After training, evaluate the model's performance on the testing dataset. Record the train/test accuracy, confusion matrix, precision and recall.
9. Plot training and validation loss for with respect to epoch for different learning rates such as 0.0001, 0.001, 0.01, and 0.1.

1 Additional Resources

1. MIT: Convolutional Neural Networks
2. MIT: Introduction to deep learning
3. Pytorch training a classifier
4. Pytorch build a model
5. keras image classification from scratch
6. keras image classification via fine-tuning with EfficientNet
7. Conv Nets: A Modular Perspective
8. Understanding Convolutions

GitHub Profile

- You must include the link to your GitHub (or some other SVN) profile, so that I can see that you have worked on this assignment over a reasonable duration. Therefore, make commits regularly. However, I will use only the pdf for grading to save time.

Submission

- Upload a report (eight pages or less) named as your_index_a03.pdf. Include the index number and the name within the pdf as well. The report must include important parts of code, image results, and comparison of results. The interpretation of results and the discussion are important in the report. Extra-page penalty is 2 marks per page.
- An extra penalty of 10% is applied for late submission.
- Pay careful attention to formatting such as font size, spacing, and margins.
- Include a title page with necessary information (e.g., title, author, date, index no).

- Use consistent and professional formatting throughout the document.
- Plagiarism will be checked and in cases of plagiarism, an extra penalty of 50% will be applied. In case of copying from each other, both parties involved will receive a grade of zero for the assignment. Academic integrity is of utmost importance, and any form of plagiarism1 or cheating will not be tolerated.

References

[1] Kunihiko Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological cybernetics*, 20(3-4):121–136, 1975.

[2] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106, 1962.

[3] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[4] Kevin P Murphy. *Probabilistic machine learning: an introduction*. MIT press, 2022.