

EN3150 Assignment 03: Simple convolutional neural network to perform classification.

Sampath K. Perera

October 1, 2024

1 CNN for image classification

In this assignment, you will create a simple image classifier using convolutional neural network (CNN). You have the freedom to use any programming language and toolkit of your choice, but it is recommended to use the **Python** programming language along with **TensorFlow** and **Keras** or **PyTorch** to construct the convolutional neural network (CNN).

1. Set up your environment: Ensure that you have installed all the required software packages.
2. Prepare your dataset: Choose a data set from **UCI Machine Learning Repository** that is appropriate for classification (**Do not use CIFAR-10 dataset**). Download the selected dataset.
3. Split the dataset into training, validation, and testing subsets using a ratio of 60% for training and 20% each for validation and testing sets¹.

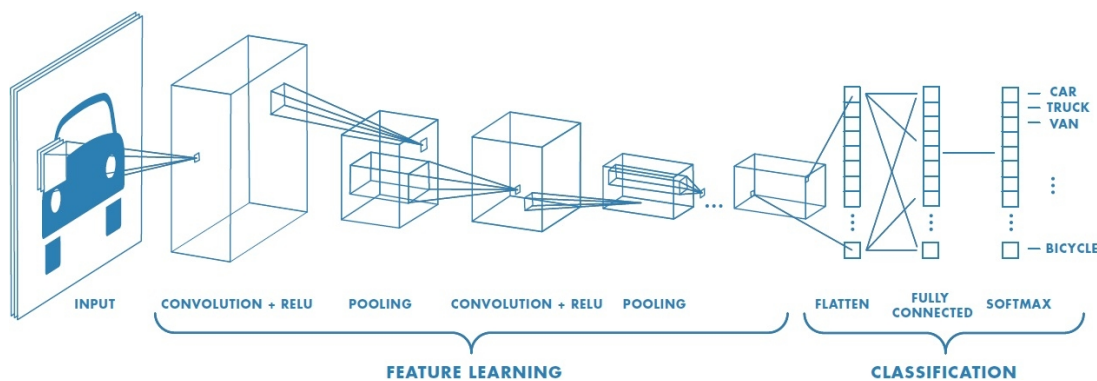


Figure 1: A simple CNN for image classification.

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

4. Build the CNN model: A common CNN design consists of interleaving convolutional and max-pooling layers, ending with a linear classification layer [1]. This pattern is illustrated in Figure 1², and it was inspired by Fukushima's neocognitron [2] and Hubel and Wiesel's work on human visual cortex [3]. Yann LeCun's LeNet model refined this approach in 1998, popularizing it through backpropagation and SGD [4]. 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 $m_1 \times m_1$ kernel, and a suitable activation function.
 - A MaxPooling layer.
 - Another Convolutional layer with x_2 filters, a $m_2 \times m_2$ kernel, and a suitable activation function.
 - Another MaxPooling layer.
 - Flatten the output.
 - A fully connected layer with x_3 units and a suitable activation function.
 - Add dropout with a rate of d to reduce overfitting.
 - An output layer with K units (for K classes) and 'softmax' activation.
5. Determine the parameters of the above network such as activation functions, kernel sizes, filter sizes, size of the fully connected layer and dropout rate.
 6. Provide justifications for activation functions selections.
 7. 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>.
 8. Why we have chosen adam optimizer over SGD?
 9. Why we have chosen sparse categorical crossentropy as the loss function?

²Image adapted from <https://blog.floydhub.com>

10. Evaluate the Model: After training, evaluate the model's performance on the testing dataset. Record the train/test accuracy, confusion matrix, precision and recall.
11. 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. Comment on your results and select a learning rate with a justification.

2 Compare your network with state-of-the-art networks

"In practice, training an entire convolutional network from scratch (starting with random initialization) is a rare occurrence. This is primarily due to the scarcity of datasets of sufficient size. Instead, a common approach is to first pretrain a CNN on an extensive dataset, such as ImageNet, which comprises 1.2 million images across 1000 categories. Subsequently, this pretrained CNN can be utilized as either an initialization point or a fixed feature extractor when tackling a specific task of interest. This is known as transfer learning"³.

12. Choose two state-of-the-art pre-trained model or architecture like [ImageNet](#), [ResNet](#), [GoogLeNet](#), [AlexNet](#), [DenseNet](#) and [VGG](#).
13. Load the pre-trained model and fine-tune it for the your dataset.
14. Train the fine-tuned model using the same training and testing data splits as your custom CNN model.
15. Record training and validation loss values for each epoch.
16. Evaluate the fine-tuned model on the testing dataset and calculate the test accuracy.
17. Compare the test accuracy of your custom CNN model with that of the fine-tuned state-of-the-art model.
18. Discuss trade-offs, advantages, and limitations of using a custom model versus a pre-trained model.

3 Additional Resources

1. [MIT: Convolutional Neural Networks](#)
2. [MIT: Introduction to deep learning](#)
3. [Pytorch training a classifier](#)
4. [Pytorch build a model](#)

³<https://cs231n.github.io/transfer-learning/>

5. [Deep learning with Pytorch: A 60 minute blitz](#)
6. [keras image classification from scratch](#)
7. [keras image classification via fine-tuning with EfficientNet](#)
8. [Conv Nets: A Modular Perspective](#)
9. [Understanding Convolutions](#)
10. [pytorch resnet](#)
11. [pytorch googlenet](#)
12. [Pytorch models and pre-trained weights](#)
13. [Transfer learning for computer vision tutorial](#)
14. [Stanford: Transfer learning](#)

4 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.

5 Submission

- Upload a report and your codes as a zip file named as "EN3150_your_groupno_A03.zip". Include the index numbers and the names of the group members within the report. Only one submission is required from a group.
- The interpretation of results and the discussion are important in the report.
- 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 plagiarism⁴ or cheating will not be tolerated.
- An extra penalty of 15% is applied for late submission.

References

- [1] Kevin P Murphy, *Probabilistic machine learning: an introduction*, MIT press, 2022.
- [2] Kunihiro Fukushima, "Cognitron: A self-organizing multilayered neural network," *Biological cybernetics*, vol. 20, no. 3-4, pp. 121–136, 1975.
- [3] David H Hubel and Torsten N Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, pp. 106, 1962.
- [4] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

⁴<https://en.wikipedia.org/wiki/Plagiarism>