COMP SCI 7318 Deep Learning Fundamentals - Assignment 2

Lalitphan Sae-teoh (a1932456)

University of Adelaide

Abstract

Image classification is one of the Artificial Intelligence (AI) applications that is essential and widely used in modern days. This assignment focuses on implementing the Convolutional Neural Network (CNN), which is one of deep learning technique for computer vision, to classify object in the image. To experiment and analysis the implementation of a simple CNN model on the CIFER-10 dataset and compare it with some tuning experiments and the pre-trained model ResNet18.

1. Introduction

We live in the digital age. With the accessibility of the internet, a huge amount of data volume is generated every day by society in various forms, especially images and videos. It is essential to develop Artificial Intelligence (AI) applications to deal with that data, One such application is image classification.

Image classification capabilities have drawn much attention lately in various sectors, for example, healthcare, transportation, manufacturing, etc. It involves healthcare analysis of medical images such as X-rays, MRIs, and CT scans and provides remarkably accurate diagnosis support. In transportation, self-driven automobiles can't operate without image classification to detect vehicles, road tracks, and traffic. Image classification is also an impactful tool used in manufacturing to improve product quality and inspecting processes which helps reduce the number of human workers and reduce cost.

This assignment focuses on implementing a Convolutional Neural Network (CNN), a special form of neuron network with specific architecture layers, from scratch to classify images from the CIFER-10 dataset. Then, experiment with model tuning to improve performance and explore the effect of hyperparameters. Also, trained with a pre-trained ResNet18 model to compare the results. The objective is to assess the effectiveness of the Convolutional Neural Network model in image classification applications.

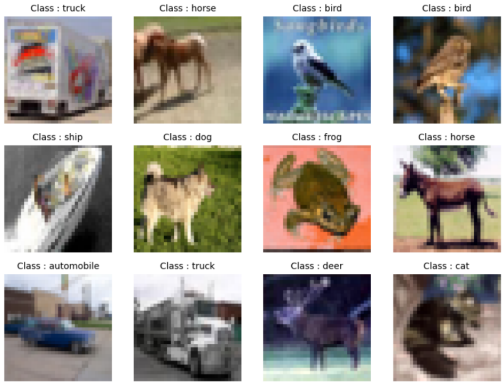

Figure 1. Example of CIPHER-10 images

1.1. Dataset

The dataset for model training is CIFER-10, a well-known dataset in the computer vision machine learning field. There are 10 object classes in this dataset, a total of 60,000 color images with size 32x32 pixels. The object classes are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each class contains 6,000 images, separated into 5,000 images for training, and 1,000 images for testing. Fig. 1 Despite the famous, the challenge of this dataset is the low resolution of images compared to other datasets for classification.

1.2. Convolution Neural Network (CNN)

Convolutional Neural Network (CNN) is a type of deep learning network. It has been applied to process and make predictions from different data types including text, images, and audio. It operates with these data by applying convolutional operations, a mathematical operation, that optimizes images with filters (or kernels) allowing CNN to capture important features and learn efficiently from them. [2] CNN has been widely used in many applications such as facial recognition, image classification, speech recognition, object detection, etc.

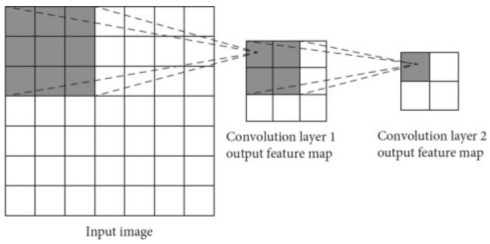

Figure 2. Convolution - Feature Maps

1.2.1 CNN Architecture

Compared to a regular neural network, the structure of CNNs is different. While each layer in a regular neural network consists of a set of neurons and connects to all neurons in the previous layer, in CNNs all neurons in a particular layer are not connected to the previous layer neurons. Instead, CNNs have 3-dimensional layers in a width, height, and depth manner in which a layer is only connected to a small portion of neurons in the previous layer. A typical CNN consists of the following layers that learn together:

Convolutional Layers The key part of the CNN architecture is the convolutional layer, it plays an important role in extracting the features. It performs feature extraction by applying multiple filters, sometimes called kernels, to the input images. These filters extract features such as edges, textures, and shapes into different feature maps from the input image by moving across the image and performing element-wise multiplication (convolution operation) to learn complex visual patterns. Fig. 2 [4] Then, the multiple convolutional layers are stacked in sequence with some non-linearity and become a convolutional neural network (CNN).

Activation Layers The activation function is applied in this layer, it introduces non-linearity into the model. The activation function plays a significant role in approximating complex functions beyond normal linear transformation. There are several activation functions such as Sigmoid, Tahn, and ReLU (Rectified Linear Unit).

The ReLU function is $f(x) = max(0, x)$. [1] It is simple, fast to compute, and helps solve vanish gradients issues. ReLU is a popular activation function in CNN implementation because it was not differentiable at the point zero, unlike Sigmoid and Tanh which suffer saturation over time, then lead to vanishing gradients.

Pooling Layers The objective of the pooling layer is to downsample the feature maps. It helps control the model's complexity, reduce overfitting, improve computation, and reduce parameters, while the important features are maintained. The operation of pooling involves sliding a 2-dimensional filter over the feature map channel and summarising the features. There are many types of pooling layers such as max pooling, average pooling, and global pooling. Max pooling is a pooling operation that selects the maximum value in the covered region of the feature map.

Fully Connected Layers At the last layers of CNN are fully connected layers which work like normal multi-layer perceptron. It accepted the flattened input from convolutional and pooling layers and passed through the fully connected layer to combine the extracted features and classify the input image into specific classes. [3]

1.3. Optimizer

Optimizers are essential algorithms in deep learning because they dynamically adjust a model's parameters during training to minimize a predetermined loss function. By iteratively adjusting the weights and biases in response to feedback from the data, these specialized algorithms help neural networks learn. Well-known deep learning optimizers include Stochastic Gradient Descent (SGD), Adaptive Moment Estimation (Adam), and RMSprop. Each of these optimizers has its own set of update rules, learning rates, and momentum strategies, all of which are designed to find and converge on the best model parameters and improve overall performance.

1.3.1 Stocastic Gradient Descent (SGD)

In Stochastic Gradient Descent (SGD), the batches of data are randomly selected for each iteration rather than processing the complete dataset. This suggests that only a small number of dataset samples are considered at once, enabling more effective and computationally feasible deep learning model optimization.

1.3.2 Adaptive Moment Estimation (Adam)

The Adam optimizer, which stands for Adaptive Moment Estimation optimizer, is a popular deep learning optimization technique. It is intended to update a neural network's weights as it is being trained and is an extension of the Stochastic Gradient Descent (SGD) algorithm. The term "adaptive moment estimation", emphasizes its capacity to adaptively modify the learning rate for every network weight separately. The Adam optimizer dynamically calculates individual learning rates depending on the previous gradients and their second moments, in contrast to SGD, which keeps a single learning rate during training.

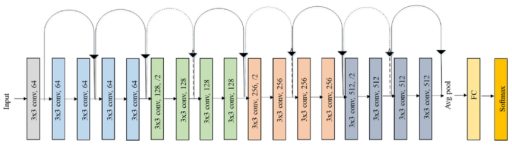

Figure 3. ResNet-18 Architecture

1.4. Transfer Learning

A deep learning technique called transfer learning uses a model that has been trained on one task as the foundation for a model on a different, related task. This can be accomplished by either fine-tuning the training model's weights for the new job or by employing the trained model as a fixed feature extractor. Despite training a model from scratch, transfer learning can save time and money while enhancing the new model's performance at the same time. CNN has been pre-trained on huge picture datasets for example, ImageNet are frequently employed in computer vision for transfer learning.

1.4.1 ResNet-18

Over a million photos from the ImageNet collection were used to train the convolutional neural network ResNet-18. The network has consequently acquired rich feature representations for a variety of images. Images can be categorized by the network into 1000 different item types.

The well-known vanishing gradient is one of the issues that ResNets resolves. This is because the gradients from which the loss function is computed readily shrink to zero after multiple chain rule applications when the network is too deep. As a result, no learning is taking place because the weights never update their values. Gradients from later layers to initial filters can flow straight through the skip connections when using ResNets. Fig. 3 [5]

2. Code

The implementation code for this assignment can be found as follows source-codes.

According to the experiments performed in this assignment, there are 4 notebooks as follows:

1. CNN base model
2. CNN base model transform tuning
3. CNN base model optimizer tuning
4. ResNet-18 model

3. Base Model Implementation

This section explains the process for building the CNN model from scratch as a base model: data preprocessing, model training, and evaluation process.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
Cifar10CnnModel                          [1, 10]                   --
├─Sequential: 1-1                        [1, 10]                   --
│    └─Conv2d: 2-1                       [1, 32, 32, 32]           896
│    └─ReLU: 2-2                         [1, 32, 32, 32]           --
│    └─MaxPool2d: 2-3                    [1, 32, 16, 16]           --
│    └─Conv2d: 2-4                       [1, 64, 16, 16]           18,496
│    └─ReLU: 2-5                         [1, 64, 16, 16]           --
│    └─MaxPool2d: 2-6                    [1, 64, 8, 8]             --
│    └─Conv2d: 2-7                       [1, 128, 8, 8]            73,856
│    └─ReLU: 2-8                         [1, 128, 8, 8]            --
│    └─MaxPool2d: 2-9                    [1, 128, 4, 4]            --
│    └─Flatten: 2-10                     [1, 2048]                 --
│    └─Linear: 2-11                      [1, 256]                  524,544
│    └─ReLU: 2-12                        [1, 256]                  --
│    └─Linear: 2-13                      [1, 128]                  32,896
│    └─ReLU: 2-14                        [1, 128]                  --
│    └─Linear: 2-15                      [1, 10]                   1,290
==========================================================================================
Total params: 651,978
Trainable params: 651,978
Non-trainable params: 0
Total mult-adds (M): 10.94
==========================================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 0.46
Params size (MB): 2.61
Estimated Total Size (MB): 3.08
==========================================================================================
```

Figure 4. Base Model Summary

3.1. Data Preprocess

From a total of 60,000 color images with a size of 32x32 pixels, the dataset is separated into 50,000 images for the training set and 10,000 images for the testing set. For this assignment, the validation set is randomly split from the training set of 5,000 images for hyperparameter tuning where each class is stratified, so the training set is reduced to 45,000 images.

For the base model, the training data is set to 128 batch size (samples per batch to load), and to reshuffle the data at every epoch, while the validation data is set to double the batch size of training data and no shuffle. The shuffle option is helpful for model training to ensure that the model learns a mix of class data and shuffles to prevent some majority classes is sequential in the sample.

3.2. Model Training

After preprocessing the data, the CNN base model was implemented from scratch. The goal was to train the model to classify 10 classes of the CIFER-10 dataset. The training was repeated for 10 epochs iterations with a learning rate of 0.001 and Adam optimizer with a default setting. The architecture of the CNN base model is constructed as follows: Fig. 4

- **Three Convolutional Layers**:
  1. **First Convolutional Layer**: 32 3x3 filters with the ReLu nonlinear activation, 2x2 max pooling
  2. **Second Convolutional Layer**: 64 3x3 filters with the ReLu nonlinear activation, 2x2 max pooling
  3. **Third Convolutional Layer**: 128 3x3 filters with the ReLu nonlinear activation, 2x2 max pooling
- **Fully Connected Layer**: 2 hidden layers with the ReLu nonlinear activation

3.3. Evaluation

For the evaluation process, the cross entropy is used to calculate the loss, and Accuracy is used for measurement.

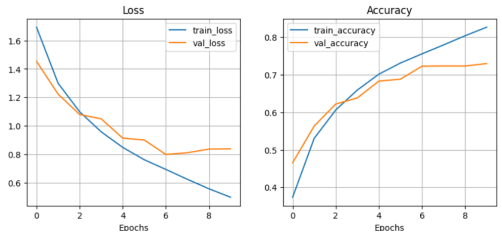

Figure 5. Base Model Loss and Accuracy

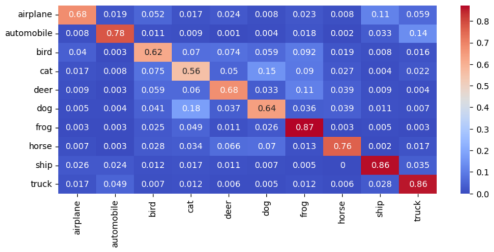

Figure 6. Base Model Test Accuracy

3.3.1 Cross Entropy

Cross-entropy is a popular loss function used in machine learning to measure the performance of a classification model. In particular, it calculates the difference between the expected values and the probability distribution that a classification model has found. The categorical cross entropy is used for multiple label classification, the formula is:

$$H(y, \hat{y}) = -\sum_{i} y_i log(\hat{y}_i)$$

where,

- **y**: the true probability distribution (usually in the form of one-hot encoded vectors)
- $\hat{y}$: the predicted probability distribution (output of softmax)

3.3.2 Accuracy

The accuracy is calculated as the ratio of correctly predicted samples to the total number of samples in the test set:

$$Accuracy = \frac{Numberofcorrectpredictions}{Totalnumberofpredictions}$$

3.3.3 Base Model Result

The training and validation result of the base model can be found in Fig. 5. Overall, test accuracy is 73.26%, and the base model performs well on predicting class frog, truck, and ship with accuracy 87%, 86%, and 86% respectively. It is still not good at predicting cats which get only 56% accuracy. Fig. 6

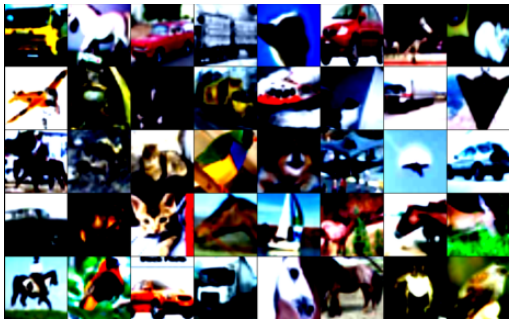

Figure 7. Example of Transformed Input Images

4. Experiments and Analysis

There are 3 experiments performed to compare with the base model in this assignment:

1. Data Transformation Tuning in Base Model
2. Optimizer Tuning in Base Model
3. ResNet-18 Model Training

4.1. Data Transformation Tuning

The input images for the base model didn't apply any transformation, so this experiment would like to try whether data transformation can improve model performance. The input images were transformed as follows: Fig. 7

1. Resize to 256x256 pixel
2. Crop center to 224x224 pixel
3. Normalize

As the input size is changed, the new architecture of the CNN tuning model is constructed as follows: Fig. 4

- **Five Convolutional Layers**:
  1. **First Convolutional Layer**: 32 3x3 filters with the ReLu nonlinear activation, 2x2 max pooling
  2. **Second Convolutional Layer**: 64 3x3 filters with the ReLu nonlinear activation, 2x2 max pooling
  3. **Third Convolutional Layer**: 128 3x3 filters with the ReLu nonlinear activation, 2x2 max pooling
  4. **Fourth Convolutional Layer**: 256 3x3 filters with the ReLu nonlinear activation, 2x2 max pooling
  5. **Fifth Convolutional Layer**: 512 3x3 filters with the ReLu nonlinear activation, 2x2 max pooling
- **Fully Connected Layer**: 2 hidden layers with the ReLu nonlinear activation and apply the drop out to regularize

The training and validation result of the data transformation model can be found in Fig. 8. Overall, test accuracy is 74.25%, and the data transformation model performs well on predicting class automobile, truck, ship, and frog with accuracy 86%, 85%, 85%, and 82% respectively. It is still not good at predicting cats and birds which get accuracy of 52% and 54% respectively. Fig. 9

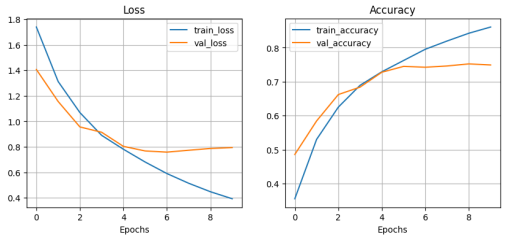

Figure 8. Data Transformation Loss and Accuracy

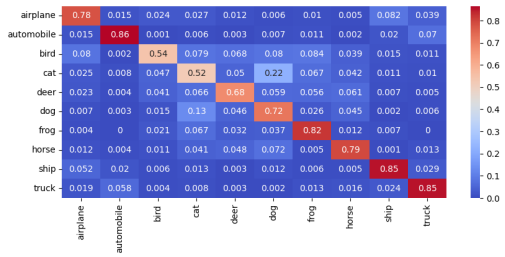

Figure 9. Data Transformation Test Accuracy

Hyperparameters	Range
batch size	16-64
learning rate	1e-5 - 1e-1
momentum	0.8 - 0.99
weight decay	1e-5 - 1e-1

Table 1. Hyperparameter Range for Tuning

4.2. Optimizer Tuning

The optimizer of the base model is Adam with a default setting, so this experiment would like to do hyperparameter tuning whether it can improve model performance. Not only Adam optimizer, but Stochastic Gradient Descent also performs in this tuning to compare results with Adam. The hyperparameter range for random search operation is shown in Tab. 1

The search of hyperparameters took 10 trials for both optimizers. The result of Stochastic Gradient Descent is not good, it got only 57.64% for best accuracy, while the result of Adam's best accuracy is 73.6% and so far the optimized parameters for Adam are 48 batch size, learning rate 0.0005, and weight decay 5.4e-5 from the random search operation.

4.3. ResNet-18 Model Training

The ResNet-18 model is implemented with the same hyperparameters setting as the base model (10 epochs, 0.001 learning rate, and Adam optimizer) and fed with the same input images as the base model (no data transformation).

The training and validation result of the ResNet-18 model can be found in Fig. 10. Overall, test accuracy is

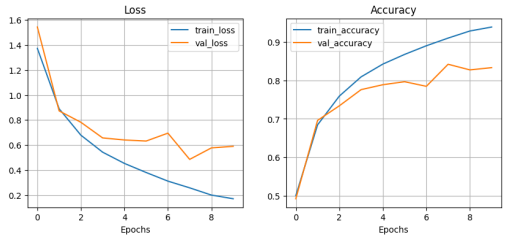

Figure 10. ResNet-18 Model Loss and Accuracy

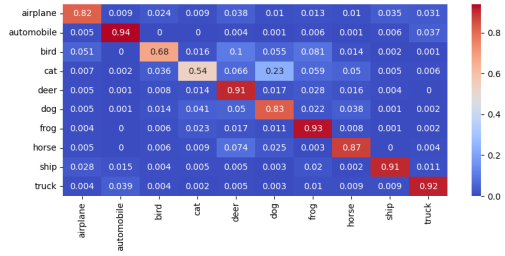

Figure 11. ResNet-18 Model Test Accuracy

83.27%, and the base model performs well on predicting class automobile, frog, truck, ship, and deer with accuracy 94%, 93%, 92%, 91%, and 91% respectively. It is still not good at predicting cats which get accuracy of 54%. Fig. 11

4.4. Experiments Analysis

From the experiments, we found that data transformation (DT model) can improve some class accuracy performance from the base model such as the accuracy of airplanes, automobiles, and dogs increased by 10%, 8%, and 8%. Resnet-18 proved that it has the best performance in this experiment with the overall test accuracy up to 10% increased from the base model. However, all models in this experiment have extremely poor performance in classifying cat class which is around 55% of accuracy. The comparison table can be found at Tab. 2.

5. Summary

In this assignment, a base model is created by implementing simple Convolutional Neural Networks (CNNs) from scratch to perform image classification on the CIFER-10 dataset. There are a total of 10 classes to classify consisting of airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The data has been split into 3 parts which are the training set for model training, the validation set for model tuning, and the test set for evaluation. The base model was initialized and its performance was evaluated by cross entropy and accuracy. The test accuracy of the base model is 73.26% and the top-3 best performance classification classes are frog (87%), Ship (86%), and Truck (86%).

Class	Base Model	DT Model	Resnet-18 Model
Overall	73.26%	74.25%	83.27%
Airplane	68%	78%	82%
Automobile	78%	86%	94%
Bird	62%	54%	68%
Cat	56%	52%	54%
Deer	68%	68%	91%
Dog	64%	72%	83%
Frog	87%	82%	93%
Horse	76%	79%	87%
Ship	86%	85%	91%
Truck	86%	85%	92%

Table 2. Test Accuracy Comparison Table

There are 3 experiments were conducted to test model performance. First, data transformation was introduced to the model by resizing, cropping, and normalizing the input image, resulting in improving the accuracy of some classes, such as airplane (78%), automobile (86%), and dog (72%), up to 8%. Second, the optimizer of the model was switched to Stochastic Gradient Descent and conducted a search operation to find the best hyperparameters, but the results were poor with an accuracy of 57.64%. Therefore, the Adam optimizer is switched back and finds the optimized hyperparameters on the searching operation. However, the result is not satisfactory due to the limitation of resources and few trials conducted. The last experiment is implementing the pre-train ResNet-18 model. The result of ResNet-18 is outstanding with an overall test accuracy of 83.27%, and some classes that achieve accuracy more than 90% such as automobile (94%), frog (93%), truck (92%), Ship (91%), and deer (91%).

Overall, the experiments showed that some tuning can improve model performance remarkably. However, some classes like cat and bird images still not classified well in these experiments. There is room for further improvements, such as applying data augmentation to create more samples, especially cat and bird images.

References

[1] Tobias Alt-Veit, Karl Schrader, Matthias Augustin, Pascal Peter, and Joachim Weickert. Connections between numerical algorithms for pdes and neural networks. *Journal of Mathematical Imaging and Vision*, 65:1–24, 2022. 2

[2] Aaron Courville Ian Goodfellow, Yoshua Bengio. *Deep Learning*. The MIT Press, 2016. 1

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 2012. 2

[4] Xiaoyao Liang. *Ascend AI Processor Architecture and Programming: Principles and Applications of CANN*. Elsevier, 2020. 2

[5] Farheen Ramzan, Muhammad Usman Khan, Asim Rehmat, Sajid Iqbal, Tanzila Saba, Amjad Rehman, and Zahid Mehmood. A deep learning approach for automated diagnosis and multi-class classification of alzheimer's disease stages using resting-state fmri and residual neural networks. *Journal of Medical Systems*, 44, 2019. 3