Practical3 - Image Recognition with Deep Neural Networks

Implementation of ANN on MNIST dataset

In this task, we are required to construct an ANN model for MNIST dataset. This is a handwritte digit classificatio task. The input is the 28×28-pixel handwritten digits and the output is the number from 0 to 9. Therefore, the input layer and output layer are determined. We only have to build hidden layers. This task is quite straightforward. I used trial-and-error approach to find the optimal layers and neurons for each layers. We know that the past common practice is to form a pyramid (the number of neurons for each hidden layer is decreasing from top to bottom layer) while the new trend is to use the same number of neurons in all hidden layers. I used the old-fashioned structure to build the model and it worked pretty well.

Why ReLU?

Comparing to *Sigmoid* activation function, *ReLU* can solve problems regarding vanishing gradients

Why Adam?

Adam optimizer combines the best properties of the *AdaGrad* and *RMSProp* algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.

Implementation of CNN by using transfer learning on CIFAR-10 dataset

Introduction

In this task, we were asked to use transfer learning to build a CNN model for CIFAR-10 dataset. The first step is to pick a pre-trained CNN model from keras library as the base model. There are many pre-trained models available (https://keras.io/api/applications/). I used trial-and-error approach to narrow down the base model. I tried MobileNet, ResNet50, VGG19, even DenseModel201 (with more than 700 convolutional layers). I ended up using MobileNet as my base model for transfer learning.

Base Model for Transfer Learning

There are no more than 100 convolutional layers in MobileNet. Comparing to other available pre-trained models, MobileNet is a "lightweight" model, which can reduce the training time to some extent. We know that top layers only capture low-level structures and these structure are shared while bottom layers combine these low-level features to capture high-level structures. Since low-level structures are highly similar across different image recognition tasks, I dropped 10 bottom layers out of the pre-trained model. Afterwards, I froze all the layers from pre-trained model to avoid distroying any of the information they contain during future training rounds. I added some new trainable layers on top of the frozen layers, which would learn to turn the old features into predictions on the given CIFAR-10 dataset. I used trial-and-error approach to search for optimal trainable layers and it turned out that one *Flatten* layer, one *Dense* layer with 128 neurons and a dropout rate of 50% worked well in our setting. Why 128 neurons? Some studies sugguest that one should start with 2^n neurons for reason that powers of 2 is the tradition in computer science.

Training Procedure

I used *EarlyStopping* and *ReduceLearningRate* techniques in training procedure. I mainly focused on two hyperparameters - *Batch Size* and *Epochs*.
Batch Size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters. Larger batch size tends to generalize worse on test dataset.
Epochs is a hyperparameter that defines how many times the learning algorithm will work through the entire training procedure. More epochs are more likely to lead to overfitting.

Unfreeze Layers

Last step is to unfreeze layers. We know that weights of pre-trained model are frozen but the pre-trained model is not meant to perform well on the given dataset. We therefore need to unfreeze layers to improve model performance on CIFAR-10 dataset. A good practice is to unfreeze layers from bottom to top since we know that top layers only learn low-level structures and these low-level structures are highly similar across different image recognition tasks.