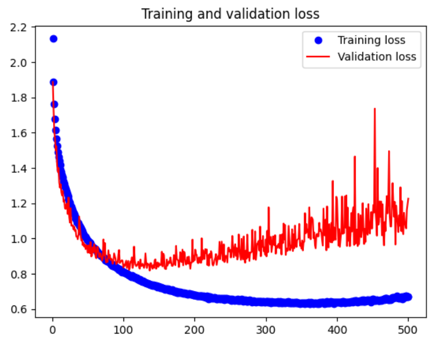
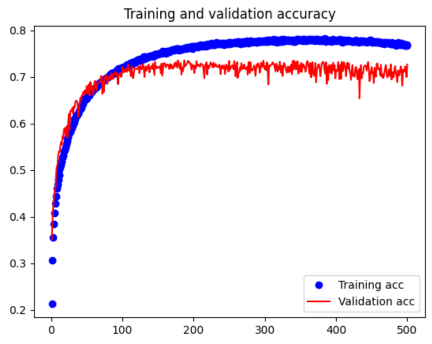
a) In this project, I have implemented four deep learning methods on CIFAR-10 dataset. The environment of this project is Windows 11, Tensorflow 2.6.0, Keras 2.6.0, Anaconda 3.

i) Simple CNN

In the first method, I construct a concurrent neural network with fewer layers. First, the model is defined using Keras's Sequential(), which is to define and add layer by layer, and finally compile. First the process of "convolutional layer-activation function-maximum pooling" is cycled three times. The convolutional layer is conv2D, the first parameter is the number of filters, the second parameter is the size of the convolution kernel, the third is the padding mode, and the fourth is the input size. The input is a 32*32 image, as for the 3, it means that the image is composed of three layers of RGB color. When Keras is once again reflected, except that the first layer requires us to define the input size, there is no need to define the latter. The framework will judge by itself that the output size of the previous layer is the input size of the next layer. After the convolutional layer is an activation function, ReLU is applied. After that is a pooling layer, max pooling is used. This is connected with a Flatten layer, which is flattened to a fully connected layer, and then connect to the general neural layer, which has 64 neurons, and then ReLU activation function is used, and then connects to a Dropout function, which means that half of the weights in the layer of 64 neurons needs to be discarded and reduce overfitting. Finally, the output layer is connected. The number of neurons in this layer is represented by the number of categories. In CIFAR -10, there are 10 categories. The final activation function is softmax, which is suitable for multi-classification tasks.

The following figures show the accuracy and loss data of training and validation respectively:

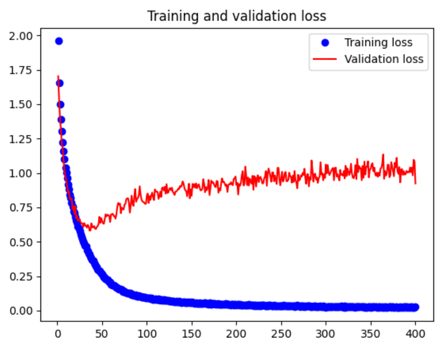


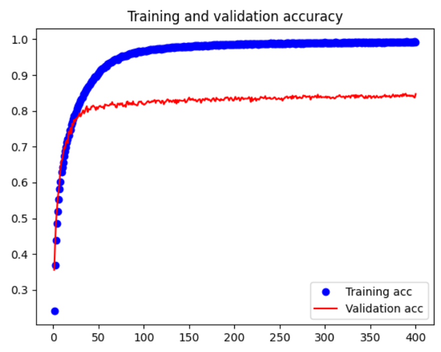


This method can achieve 78% of training accuracy and 72% of validation accuracy. Since there was little improvement in performance after 400 epochs, the following experiments took 400 epochs.

ii) Deep CNN

In this method, I made the neural network deeper into 40 layers. The following figures show the accuracy and loss data of training and validation respectively:

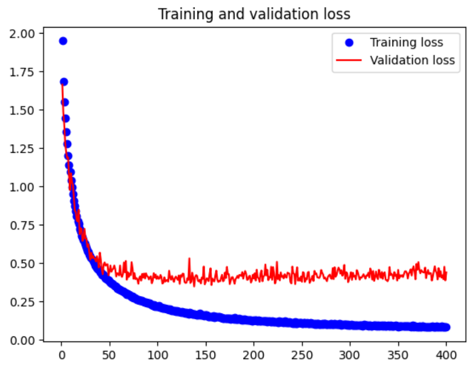


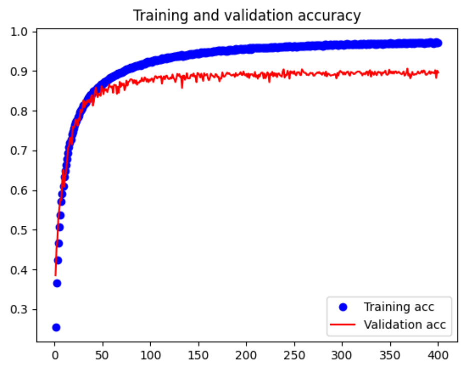


The training accuracy reached 99% and the validation reached 84.7%. From the result, we can see that making the network deeper is very effective.

iii) Deep CNN with data augmentation

Since the performance of deep learning method depends largely on the amount of data, I applied data augmentation to increase the amount of data.
Keras comes with a way to generate similar image data using the ImageDataGenerator instance, including moving the original image horizontally or vertically to a certain range, flip the image horizontally or vertically, enlarge the image to a certain range to achieve the purpose of generating new images of the same type. The following figures show the accuracy and loss data of training and validation respectively:

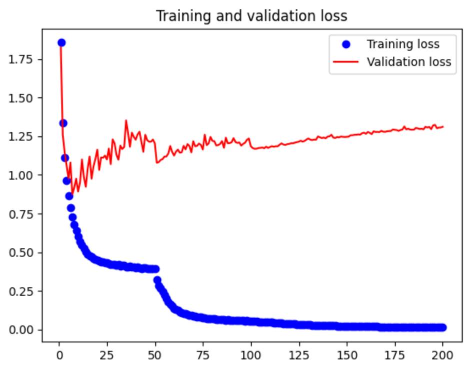


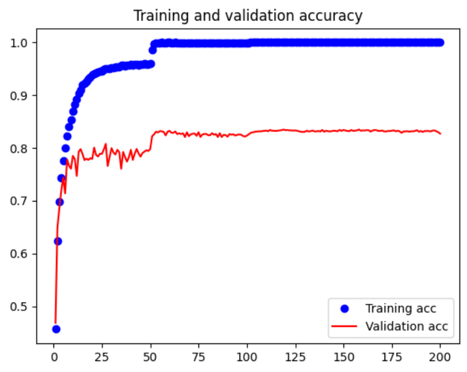


The validation accuracy further increased to 90.2% compared to the original deep CNN. Furthermore, it significantly mitigated the overfitting happened in the original model.

iv) Resnet

In this method. I have applied a small Resnet with dynamic learning rate. The following figures show the accuracy and loss data of training and validation respectively:





In the experiment, the accuracy basically does not improve within 60 epochs, the training accuracy reached 100%, and the validation accuracy reached 83.5%, which is close to the result of deep CNN without data augmentation. It shows that Resnet can achieve the performance close to deep CNN with fewer layers in less time. The reason why overfitting happened during training and testing process maybe because data augmentation was not implemented, and the number of epochs is cut down in this method since the training time would be too long.

b) In this project, noise, semantic, FGSM, PGD and C&W L2 attacks are implemented. The environment is Jupyter Notebook, pyTorch 1.11.0. The model is trained on a simple CNN similar to the first architecture used in question a)
The test and prediction results are listed in the notebook.
The results show that:
    1. Manage to implement five different attack models on a simple CNN trained on MNIST and successfully fooled the prediction
    2. Compare the performance drop of noise, FGSM and PGD attack with three different magnitudes. The accuracy of the classifier is significantly reduced from in FGSM model when the noise magnitude ϵ increases from 0.05 to 0.2.
    3. C&W attack overperforms among the five attack models implemented in the experiment.
    4. In this method simple feature squeezing based on the paper "**Feature Squeezing: Detecting Adversarial Examples in Deep Neural Network**" is implemented to FGSM and PGD attack model with the noise magnitude ϵ=0.2. Results tell that the mechanism slightly mitigated the attack by approximately 1% with the same noise magnitude

Due 05/11/2021