Deep Learning Report HW2

111511236_歐軒佑

1. MINST

1-1) Plot and Analyze

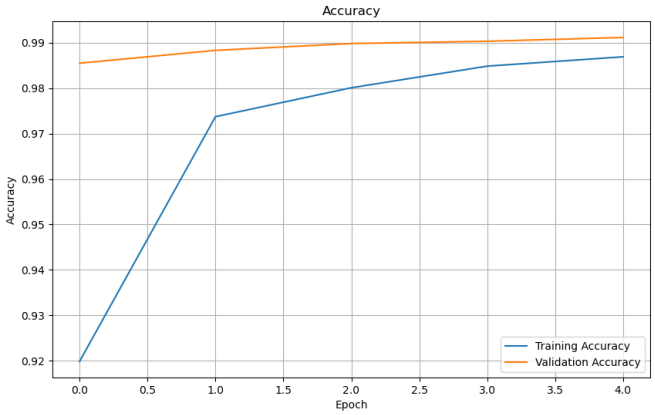
Filter size and stride size affect the features captured by the model. Smaller sizes capture more local details, which is particularly effective in MNIST, where precise handwriting recognition is essential.

In contrast, larger sizes capture broader region contours and can help improve computational efficiency. In my model, which is not that deep, (3,3) ~ (9,9) filters do not make huge difference. But when stride size increases, accuracy goes down.

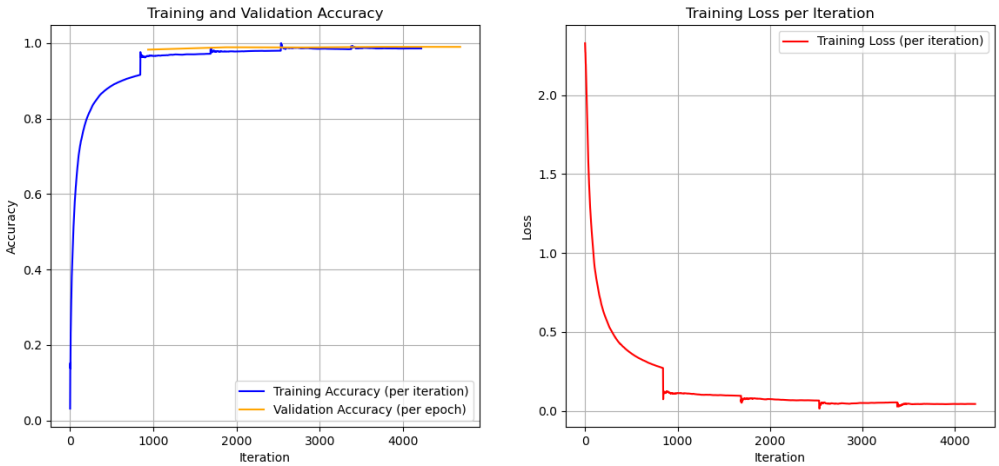
Accuracy:

```
844/844 [==============
313/313 [==============
Test accuracy: 0.9920
```

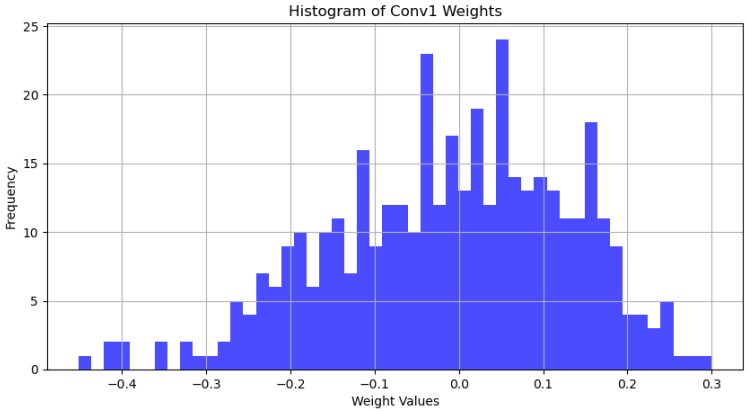
Training and Validation Accuracy(Per Epoch):



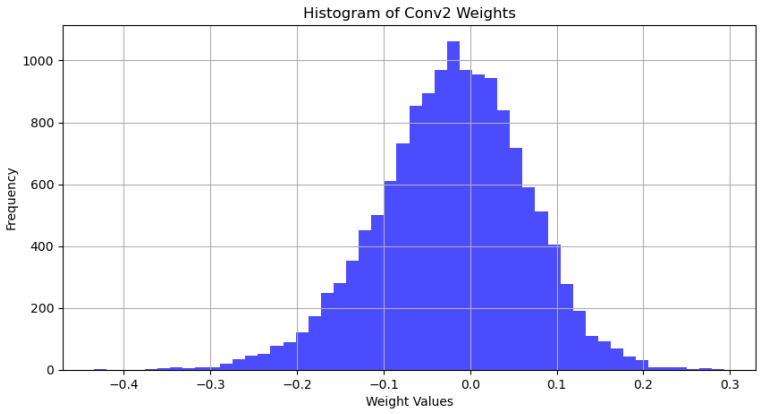
Accuracy(left) and Learning Curve (right) (Per Iteration):



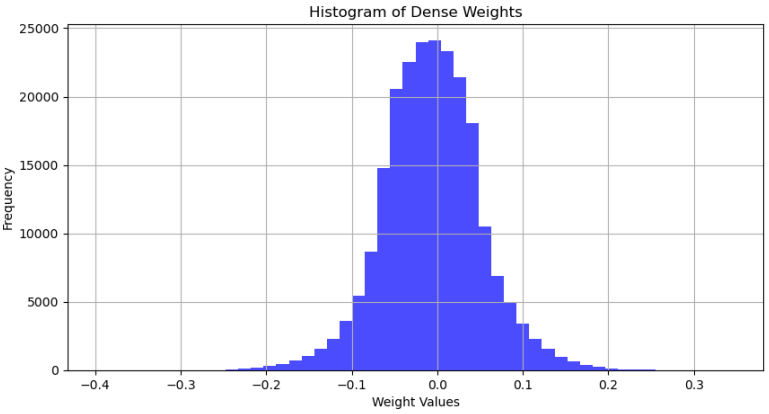
Histogram of Layers:



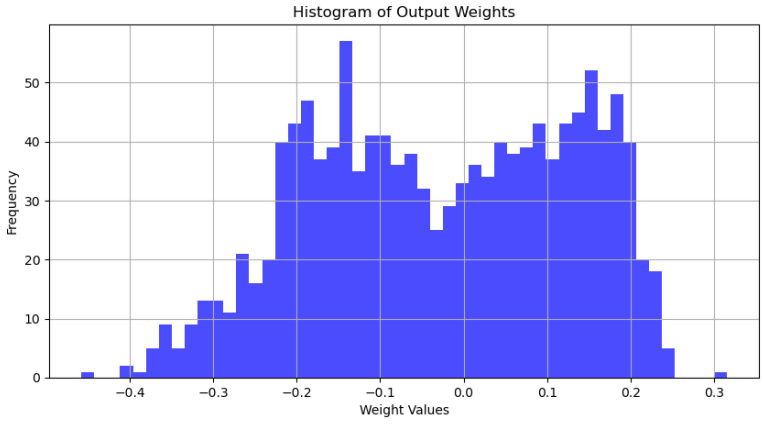
Conv1



Conv2



Dense

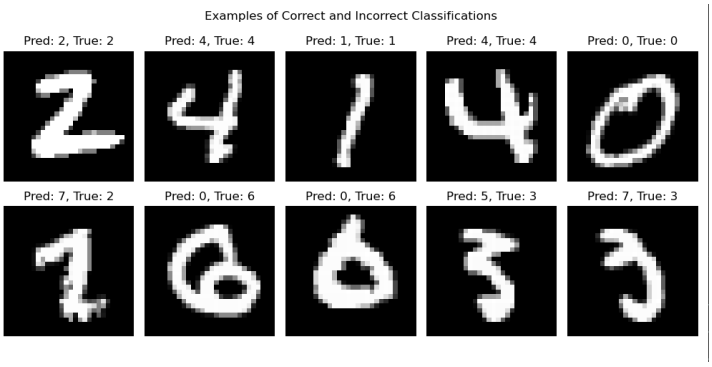


Output

1-2) Examples of correctly classified and miss-classified images

I think that the predicted results of misclassified images have contours similar to the actual digits, which indicates that the model does capture the basic shape features of the digits in certain situations but fails to accurately distinguish between classes by its details.

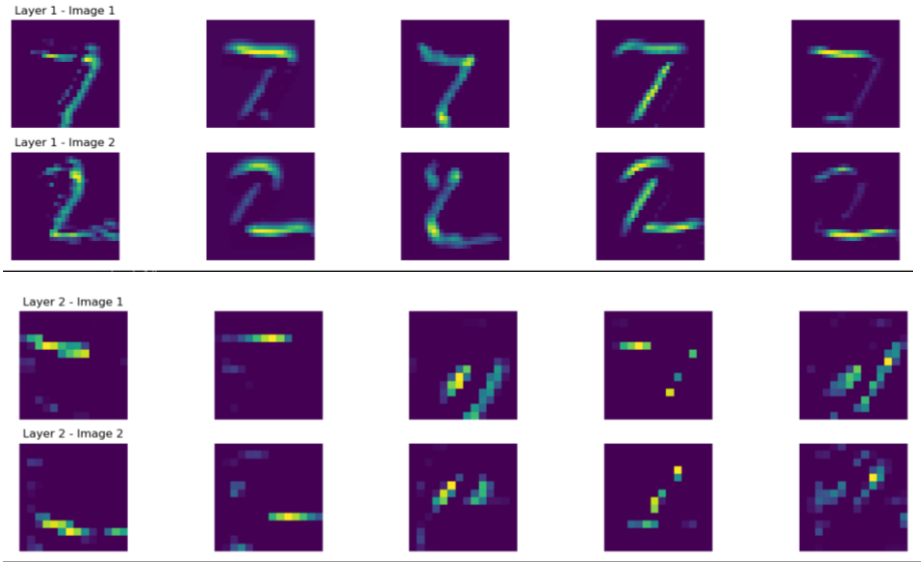
This situation often occurs when certain features of the digits are similar, such as the digits "5" and "3." There are even some images that even humans might struggle to differentiate correctly, like the images of "0" and "6" shown below.



1-3) Observe the feature maps from different convolutional layers

In Layer1, the feature maps retain much of the original image's structure, focusing on basic patterns such as edges and outlines. These maps capture lower-level features, preserving much of the shape of the digits.

While in Layer2, the feature maps become abstract, with details starting to fade and only certain parts of the original structure being highlighted. These layers extract more specific features, isolating patterns that are important for identifying the digits while filtering out less relevant details.



1-4) L2 Regulation

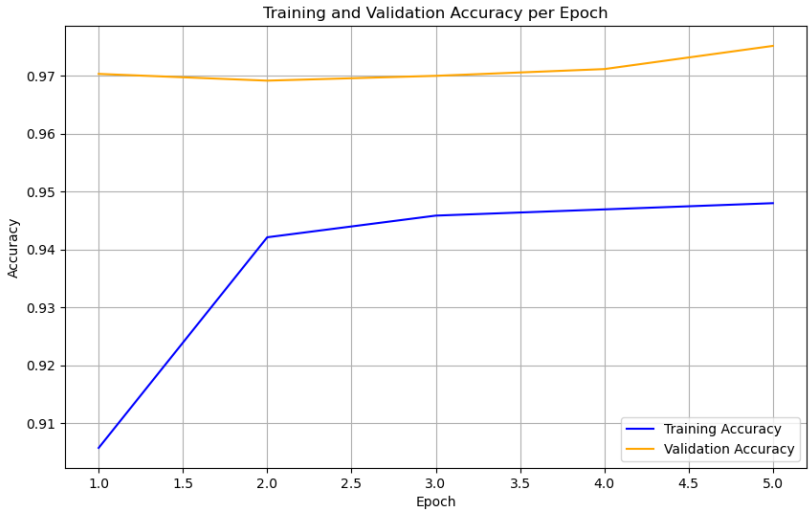
L2 regularization affects the distribution of feature weights. After applying L2 regularization, we can observe that the weight histogram is more concentrated around zero, indicating that many less important feature weights are forced closer to zero, effectively filtering out features with minimal contribution to the model's predictions.

However, for my model, applying L2 did not increase the accuracy, which remained at a level of 0.97.

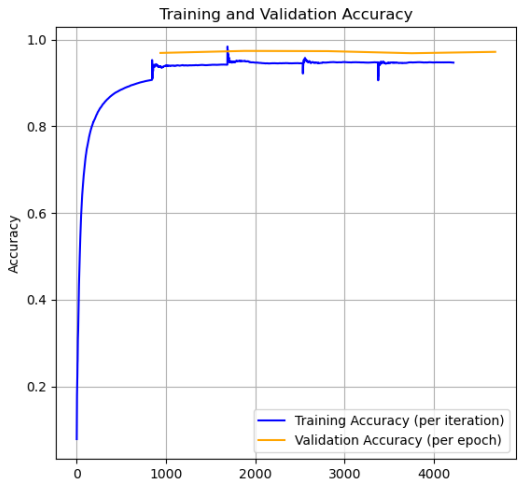
L2Accuracy:

```
Epoch 5/5
844/844 [==============
313/313 [==============
Test accuracy: 0.9720
```

L2Training and Validation Accuracy(Per Epoch):

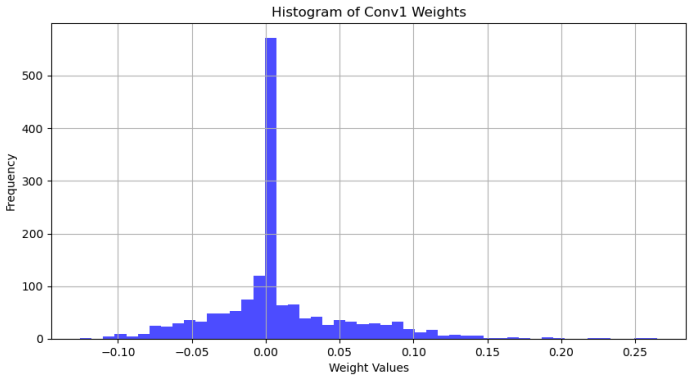


L2Accuracy(left) and Learning Curve (right) (Per Iteration):

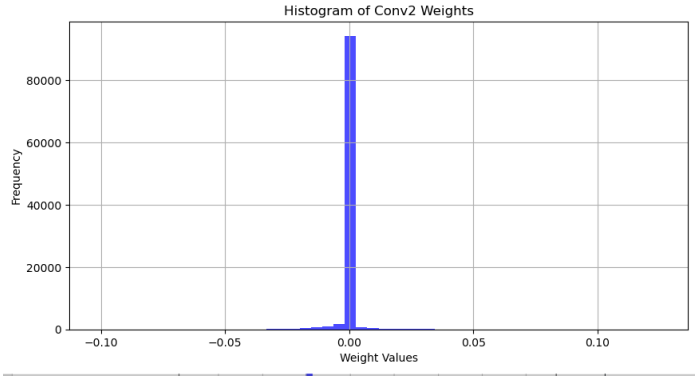


L2Histogram of Layers:

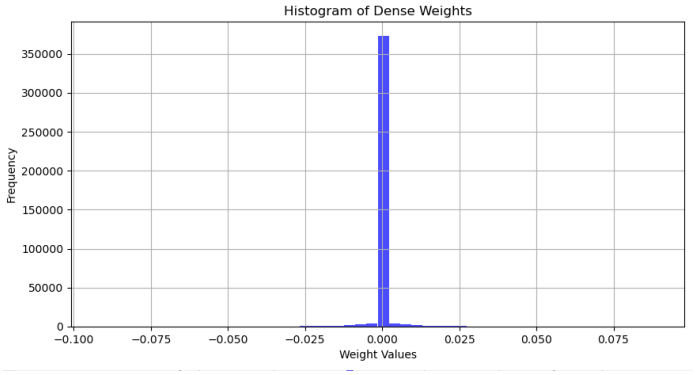
Conv1



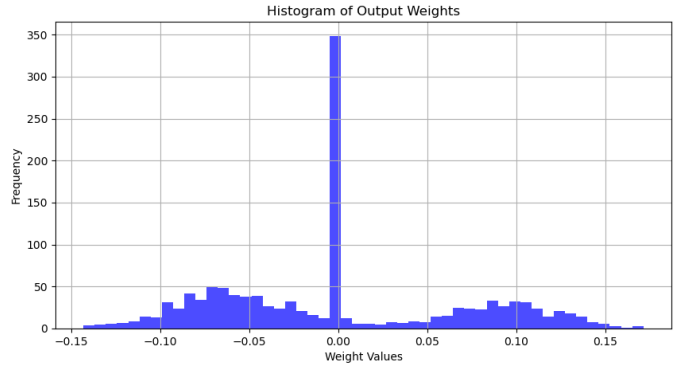
Conv2



Dense



Output



2. CIFAR-10

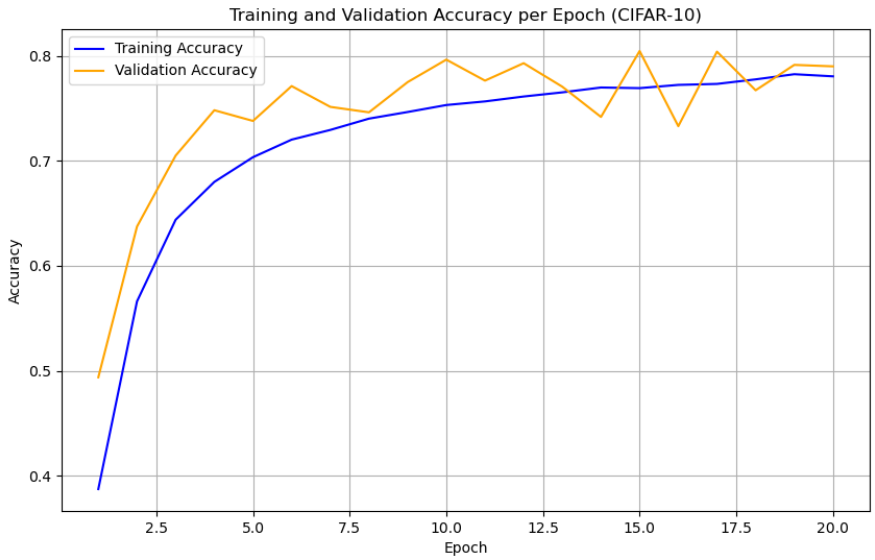
2-1) Plot

Unlike MNIST, the CIFAR-10 dataset contains more complex images, so training requires a deeper and wider layer structure as well as more epochs. When stride size and filter size goes up, accuracy decreases. So, in my model, I use filter = (3,3) and stride = (1,1) to optimize the accuracy.

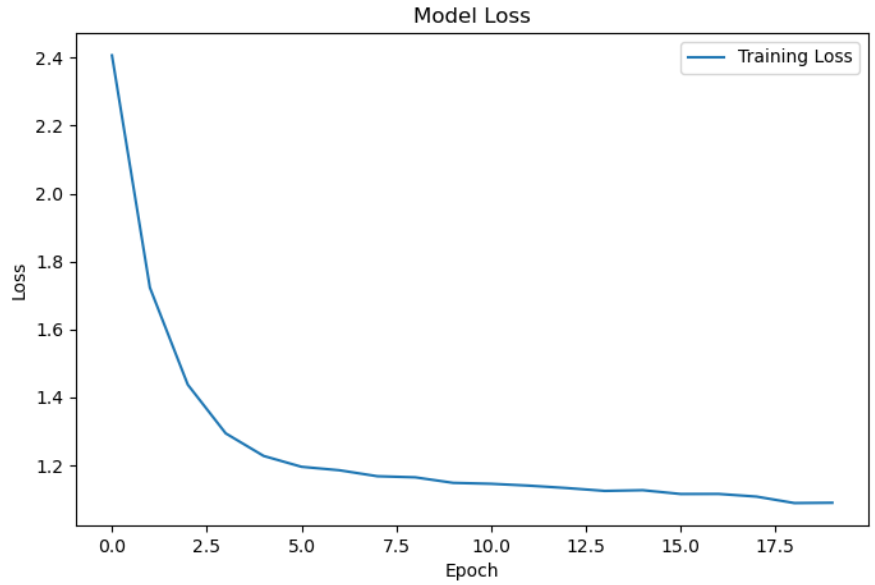
Accuracy:

```
313/313 [==============
Test accuracy: 0.8361
```

Training and Validation Accuracy(Per Epoch):

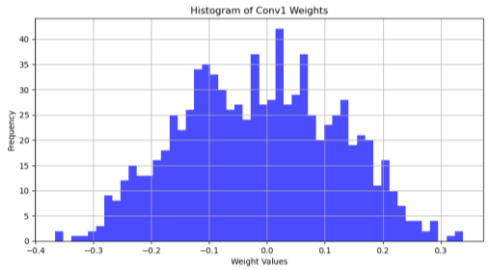


Learning Curve:

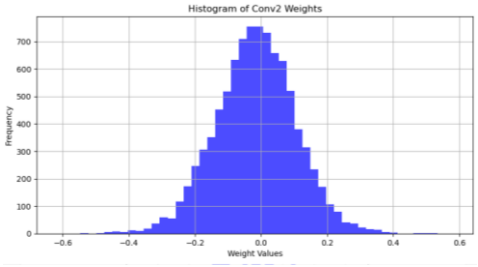


Histogram of Layers:

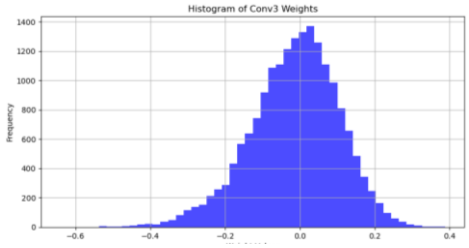
Conv1



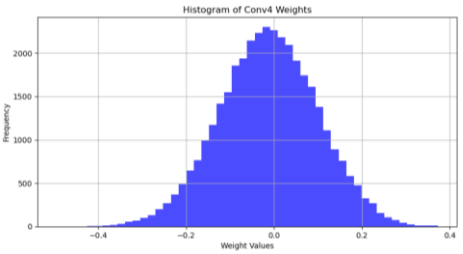
Conv2



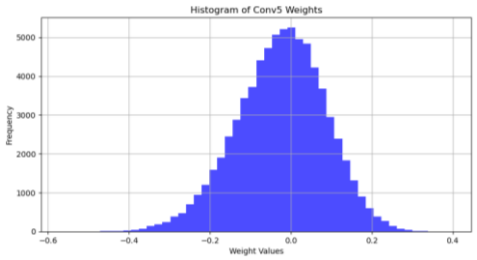
Conv3



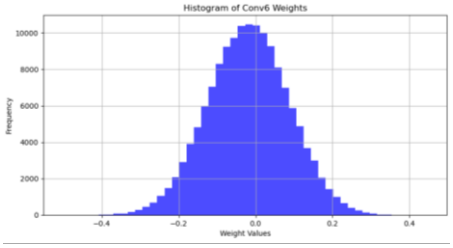
Conv4



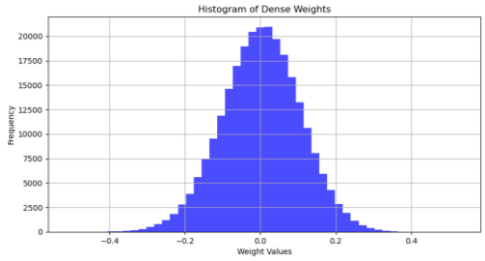
Conv5



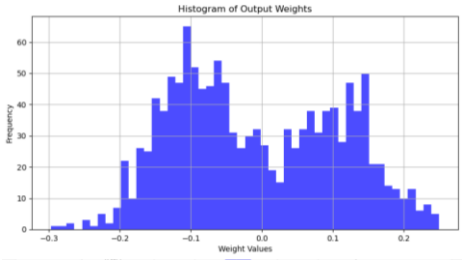
Conv6



Dense

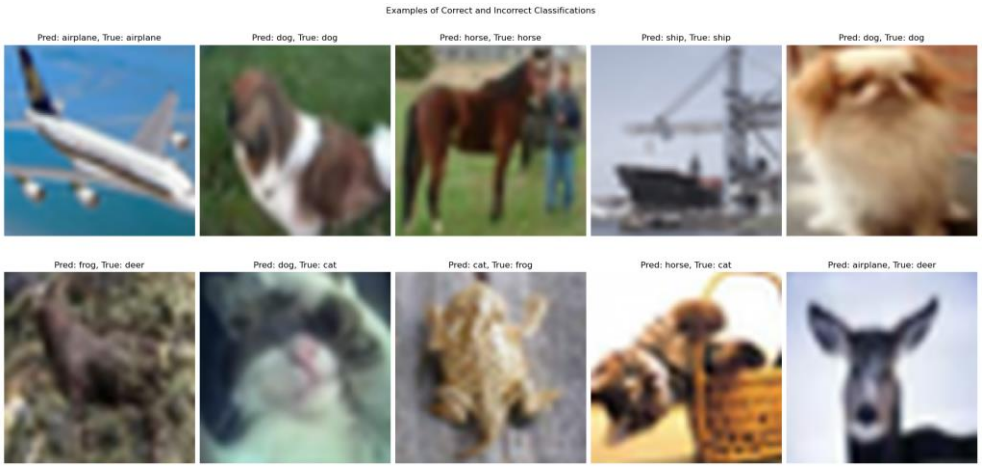


Output



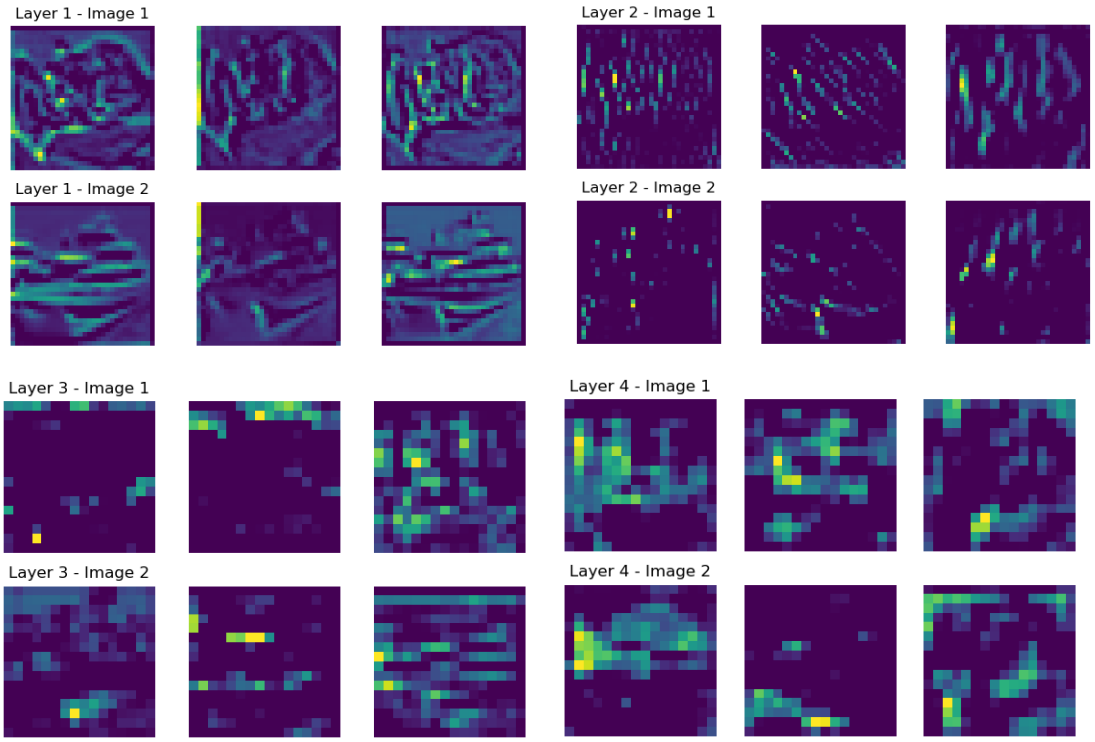
2-2) Examples of correctly classified and miss-classified images

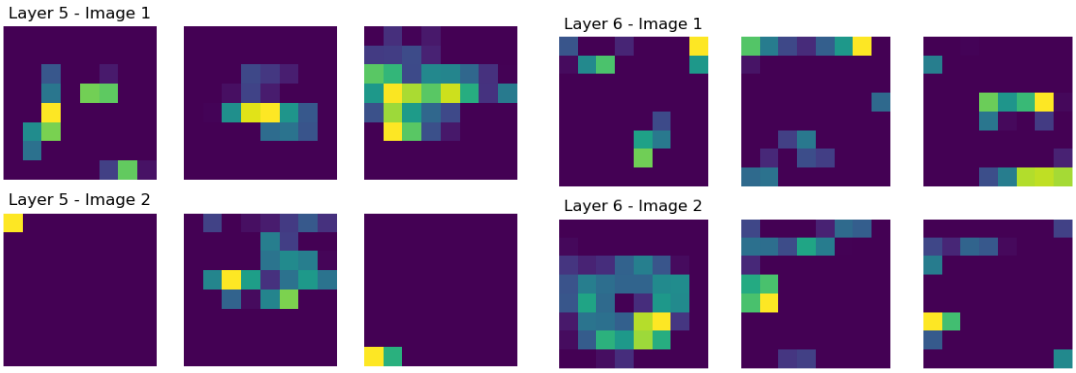
These misclassifications often occur due to similarities in shapes, textures, or ambiguous visual cues in the images. For example, the image classified as "frog" when it is actually a "deer" might have features (like dark, blurred regions) that the model mistakenly associates with the frog category.



2-3) Observe the feature maps from different convolutional layers

Image 1 is a frog and Image 2 is a ship. As the network depth increases, feature maps progress from representing low-level details (edges and textures) to high-level abstractions (object parts or complex textures). Also, its resolution drops. This hierarchical approach allows the model to capture diverse information at different levels, which is crucial for tasks like image recognition.





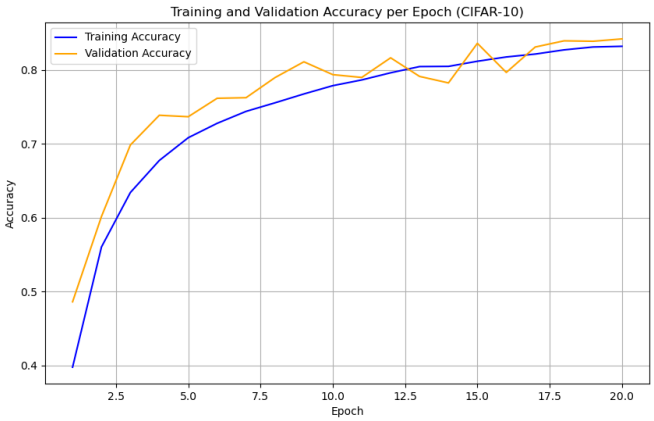
2-4) L2 Regulation

Similar to MNIST, applying L2 regularization did not improve my accuracy, although the weight distribution is not as concentrated as in MNIST, it still causes the weights to be concentrated within a smaller value range.

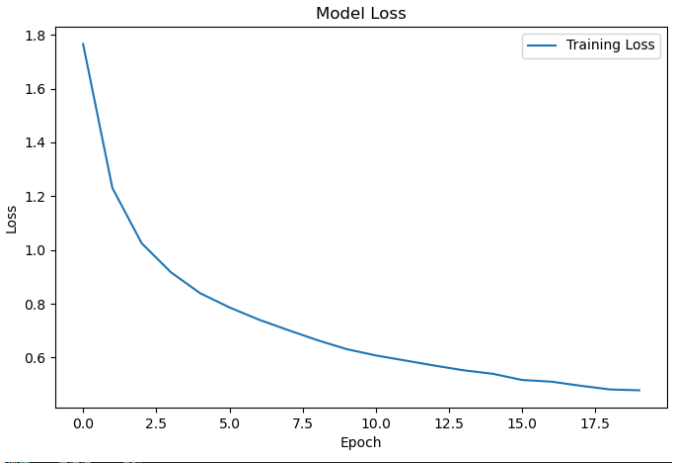
L2Accuracy:

```
313/313 [==============
Test accuracy: 0.7800
```

L2Training and Validation Accuracy(Per Epoch):

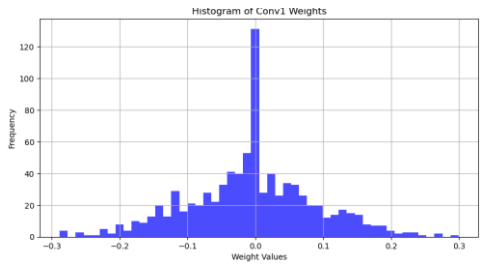


L2Accuracy(left) and Learning Curve (right) (Per Iteration):

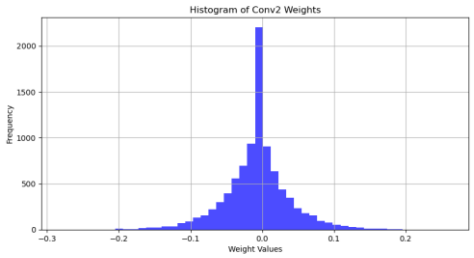


L2Histogram of Layers:

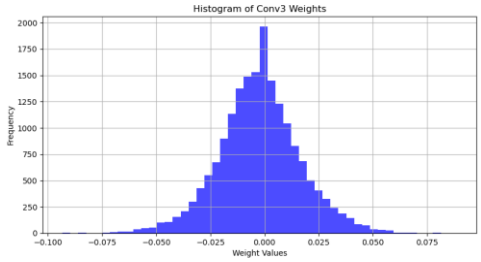
Conv1



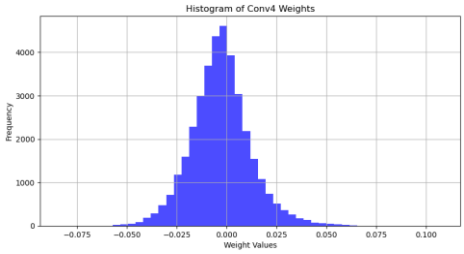
Conv2



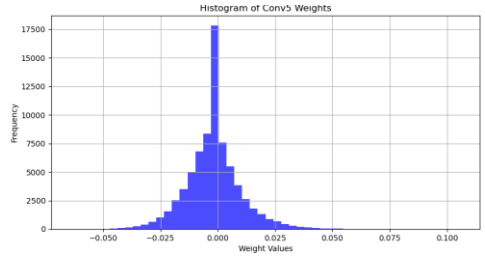
Conv3



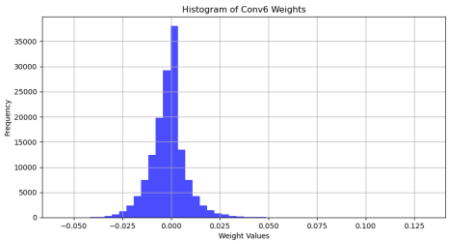
Conv4



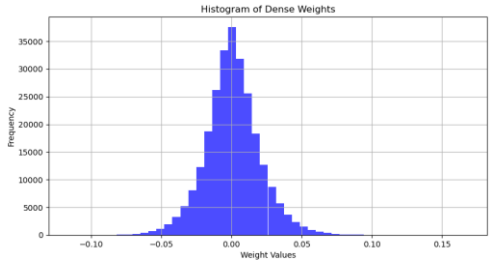
Conv5



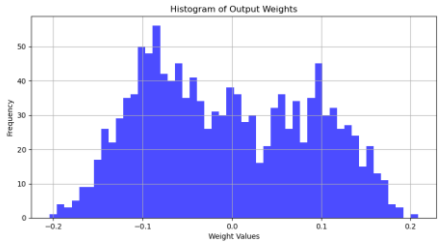
Conv6



Dense



Output



2-5) Preprocessing

2-5.1)Normalize the data

Scaling the pixel values to a [0, 1] range by dividing by 255. This helps in speeding up convergence during training.

```
# Normalize the data
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
```

2-5.2)One-hot encoding

CIFAR-10 has 10 classes, and labels are provided as integer values. To use these labels in a neural network, we need to convert them into one-hot encoding format.

```
# One-hot encoding
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

2-5.3) Data augmentation

Data augmentation increases the size of training dataset by applying random transformations (like rotations, flips, zooms) to the images. This helps improve generalization and reduce overfitting.

```
# Data augmentation
datagen = ImageDataGenerator(
    rotation_range=15,              # Randomly rotate images
    width_shift_range=0.1,          # Randomly shift images horizontally
    height_shift_range=0.1,         # Randomly shift images vertically
    shear_range=0.2,                # Randomly apply shearing transformations
    zoom_range=0.2,                 # Randomly zoom in on images
    horizontal_flip=True,           # Randomly flip images horizontally
    fill_mode='nearest'             # Replace missing pixels with the nearest pixel
)
# Apply data augmentation to the training data
datagen.fit(x_train)
```