STAT542 Midterm Project Report

Image Classification with CIFAR-10 Dataset

Group 18: Name: Yilin Zhang **NetId:** yilin19

Abstract:

Nowadays, image recognition and classification are widely used in the world, and the algorithms in this technology are constantly being improved and upgraded. Delightfully, the technology has realized a high level of accuracy. This project aims to investigate image classification on the CIFAR-10 dataset, comparing a traditional Convolutional Neural Network (CNN) with a novel approach that integrates CNN with Batch Normalization (BN) and Dropout. Initial sections introduce the task and describe the algorithms, followed by a comparison of their performance based on simulation results. Despite challenges in debugging and parameter tuning, the hybrid CNN-BN/D model demonstrates significantly improved accuracy and reduced loss over the standalone CNN, underscoring the potential of BN and Dropout methods in enhancing neural network predictions. The related code is integrated in github: https://github.com/YilinZhang0101/stat542_mid/tree/main

1. Machine Learning Task Introduction

This project aims to leverage more cutting edge algorithms and statistical learning techniques, specifically Convolutional Neural Network (CNN) and a novel approach that integrates CNN with Batch Normalization (BN) and Dropout, to process image recognition and classification. I decided to build a model capable of classifying different types of objects and match them with the correct Labels. To make the project more challenging, I chose the standard vision dataset CIFAR-10, which consists of 60000 32x32 color images in 10 classes, with 6000 images per class.

Fig.1. Example of images and classes in Dataset CIFAR-10

The goal is to build two models capable of accurately classifying 10 types of objects (10 target categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck): https://www.cs.toronto.edu/~kriz/cifar.html and match them with the correct Labels. Using

‘PyTorch’ in this project, I gained much familiarity with the PyTorch library and NumPy libraries. Finally, I want to reach an accuracy of around 60%.

2. Two Algorithm Description

2.1 Convolutional Neural Network (CNN)

2.1.1 Definition and Function

Convolutional Neural Networks (CNN) are a class of deep neural networks most commonly used to analyze visual images. Great progress has been made in the fields of image and video recognition, image classification, and medical image analysis using convolutional neural networks. It has two important features: it effectively reduces the large data of images, and it can effectively preserve image features during image processing.

Convolutional Neural Networks have convolutional operations at their core. A typical CNN architecture consists of several layers, each with a specific function:

a. Convolutional Layer: It mainly performs convolutional operations on the input image to filter the image and extract features. It learns from the image during the filtering process, summarizes the input image features, and creates a feature map.
b. Activation Layer: Each convolutional layer is generally followed by an activation layer, which introduces nonlinearity into the system and enables the network to learn complex patterns. In this project the Revised Linear Unit (ReLU) is used for activation.
c. Pooling layer: reduces the spatial size of the feature map, making feature detection invariant to changes in scale and orientation and reducing the computation required to process the data.
d. Fully Connected Layer: This part is responsible for mapping the extracted features to the final output class or prediction.
e. Output layer: softmax function for classification problems and linear function for regression problems.

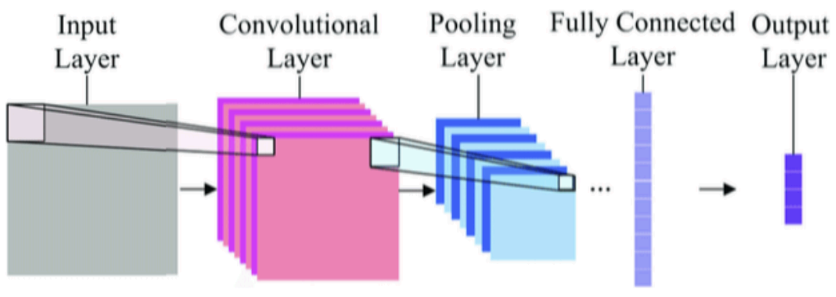
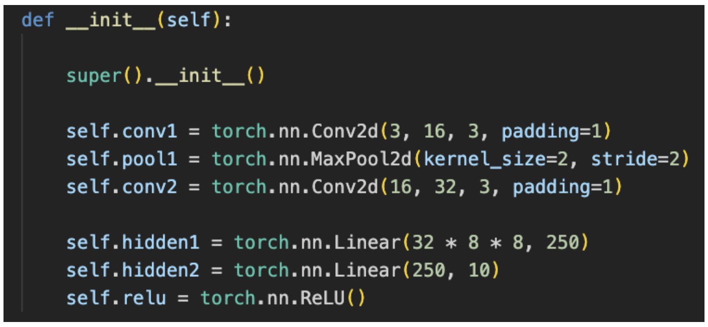


Fig.2. The Basic Architecture of CNN

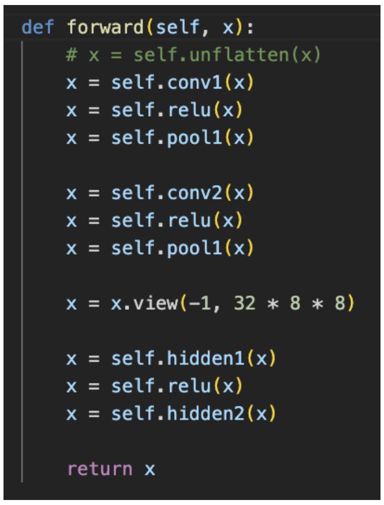
2.1.2 Implementation in Project

Use ‘PyTorch’, which is one of the most commonly used machine learning frameworks, and NumPy libraries to build CNN models. With ‘dataLoader’s, we can iterate through the datasets easily. To build a suitable ‘dataLoader’ function, I need to set a suitable batch size. Each iteration returns a batch of train_features and train_labels (in this project, I set up a batch size equal to 64, so each batch contains 64 feature and label tensors respectively).

When building a neural network class to build a model, I need to write an ‘initial’ function and a ‘forward’ function, which is the core code of CNN algorithm. (The codes are attached to the appendix. Some of the core code is as Fig.3)



```python
def __init__(self):

    super().__init__()

    self.conv1 = torch.nn.Conv2d(3, 16, 3, padding=1)
    self.pool1 = torch.nn.MaxPool2d(kernel_size=2, stride=2)
    self.conv2 = torch.nn.Conv2d(16, 32, 3, padding=1)

    self.hidden1 = torch.nn.Linear(32 * 8 * 8, 250)
    self.hidden2 = torch.nn.Linear(250, 10)
    self.relu = torch.nn.ReLU()
```



```python
def forward(self, x):
    # x = self.unflatten(x)
    x = self.conv1(x)
    x = self.relu(x)
    x = self.pool1(x)

    x = self.conv2(x)
    x = self.relu(x)
    x = self.pool1(x)

    x = x.view(-1, 32 * 8 * 8)

    x = self.hidden1(x)
    x = self.relu(x)
    x = self.hidden2(x)

    return x
```

Fig.3. The Core Code to Build CNN Model

CNNs automatically detect important features without any human supervision, using backpropagation algorithms for training.

2.2 Batch Normalization (BN) and Dropout

2.2.1 Definition and Function

Batch Normalization (BN) and Dropout is the advanced algorithm for CNN, it can help CNN improve its accuracy and efficiency.

2.2.1.1 Batch Normalization (BN):

Batch Normalization is a technique for normalizing the inputs of a particular layer in a neural network across different small batches. It was introduced to address the problem of internal covariate bias, where the distribution of inputs to each layer changes as the parameters of the previous layers change during training.

The main function of batch normalization is to stabilize and accelerate the training process of deep neural networks. It normalizes the output of the previous activation layer by subtracting the batch mean and dividing it by the batch standard deviation. It has several benefits: Improves Gradient Flow, allows Higher Learning Rates, Reduces Sensitivity to Initialization, Acts as a Regularizer.

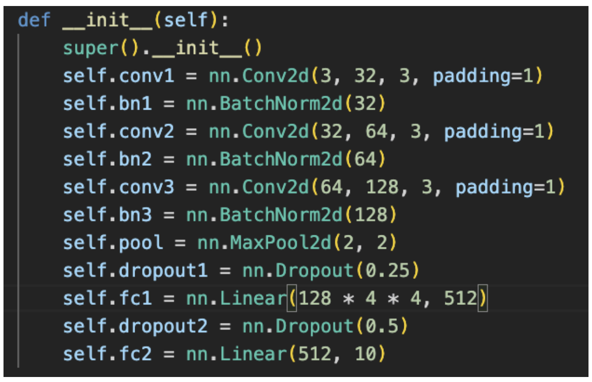
2.2.1.2 Dropout:

Dropout is a regularization technique to prevent overfitting in neural networks. It works by randomly "dropping out" (i.e., setting to zero) a number of output features of the layer during training at each update to the model.

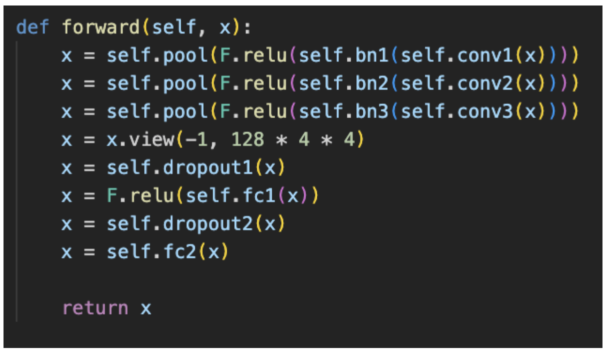
The key function of Dropout is to improve the generalization of the model by preventing complex co-adaptations on the training data. It effectively forces the network to learn more robust features that are useful in conjunction with many different random subsets of the other neurons. Its benefits are: Reduces Overfitting, Ensemble Interpretation, Model Robustness.

2.1.2 Implementation in Project

Batch Normalization (BN) and Dropout algorithm are added into code when building the model. (The codes are attached to the appendix. Some of the core code is as Fig.4)



```python
def __init__(self):
    super().__init__()
    self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
    self.bn1 = nn.BatchNorm2d(32)
    self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
    self.bn2 = nn.BatchNorm2d(64)
    self.conv3 = nn.Conv2d(64, 128, 3, padding=1)
    self.bn3 = nn.BatchNorm2d(128)
    self.pool = nn.MaxPool2d(2, 2)
    self.dropout1 = nn.Dropout(0.25)
    self.fc1 = nn.Linear(128 * 4 * 4, 512)
    self.dropout2 = nn.Dropout(0.5)
    self.fc2 = nn.Linear(512, 10)
```



```python
def forward(self, x):
    x = self.pool(F.relu(self.bn1(self.conv1(x))))
    x = self.pool(F.relu(self.bn2(self.conv2(x))))
    x = self.pool(F.relu(self.bn3(self.conv3(x))))
    x = x.view(-1, 128 * 4 * 4)
    x = self.dropout1(x)
    x = F.relu(self.fc1(x))
    x = self.dropout2(x)
    x = self.fc2(x)

    return x
```

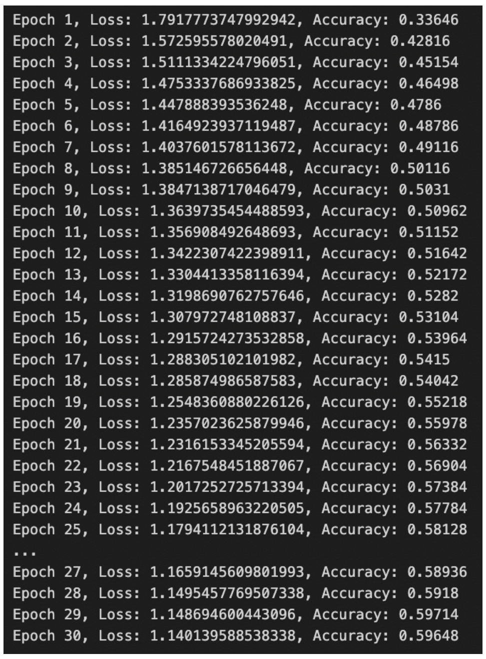
Fig.4. The Core Code for Batch Normalization (BN) and Dropout

3. Results and Comparison

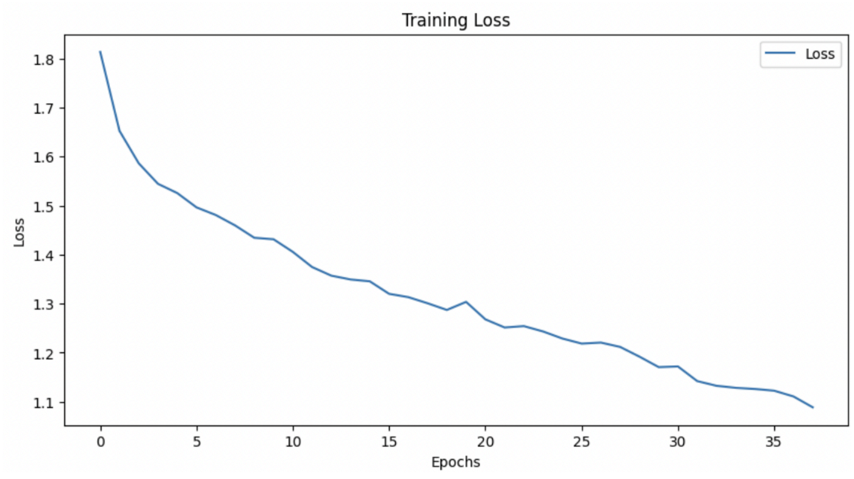
3.1 Convolutional Neural Network (CNN)

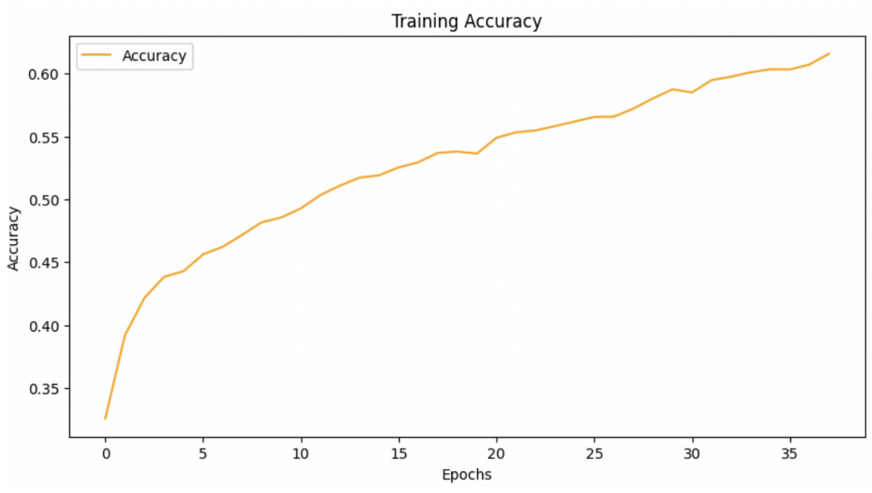
The loss and accuracy data of every epoch are shown below, and I made two figures to indicate the trend of training loss and accuracy. The loss fluctuated downward, with a steep and pronounced decline at the beginning of the curve, stable in the latter regions. Contradictorily, the accuracy fluctuated upward, also with a steep and pronounced increase at the beginning of the curve, stable in the latter regions.

I set 30 epochs, than we can see the loss became 1.140, accuracy became 59.6%, which is nearly 60%.



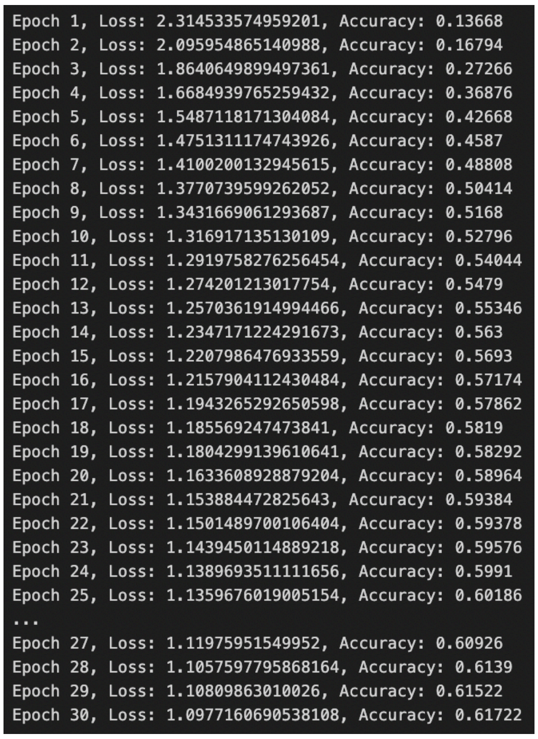
```
Epoch 1, Loss: 1.7917773747992942, Accuracy: 0.33646
Epoch 2, Loss: 1.572595578020491, Accuracy: 0.42816
Epoch 3, Loss: 1.5111334224796051, Accuracy: 0.45154
Epoch 4, Loss: 1.4753337686933825, Accuracy: 0.46498
Epoch 5, Loss: 1.447888393536248, Accuracy: 0.4786
Epoch 6, Loss: 1.4164923937119487, Accuracy: 0.48786
Epoch 7, Loss: 1.4037601578113672, Accuracy: 0.49116
Epoch 8, Loss: 1.385146726656448, Accuracy: 0.50116
Epoch 9, Loss: 1.3847138717046479, Accuracy: 0.5031
Epoch 10, Loss: 1.3639735454488593, Accuracy: 0.50962
Epoch 11, Loss: 1.356908492648693, Accuracy: 0.51152
Epoch 12, Loss: 1.3422307422398911, Accuracy: 0.51642
Epoch 13, Loss: 1.3304413358116394, Accuracy: 0.52172
Epoch 14, Loss: 1.3198690762757646, Accuracy: 0.5282
Epoch 15, Loss: 1.307972748108837, Accuracy: 0.53104
Epoch 16, Loss: 1.2915724273532858, Accuracy: 0.53964
Epoch 17, Loss: 1.288305102101982, Accuracy: 0.5415
Epoch 18, Loss: 1.285874986587583, Accuracy: 0.54042
Epoch 19, Loss: 1.2548360880226126, Accuracy: 0.55218
Epoch 20, Loss: 1.2357023625879946, Accuracy: 0.55978
Epoch 21, Loss: 1.2316153345205594, Accuracy: 0.56332
Epoch 22, Loss: 1.2167548451887067, Accuracy: 0.56904
Epoch 23, Loss: 1.2017252725713394, Accuracy: 0.57384
Epoch 24, Loss: 1.1925658963220505, Accuracy: 0.57784
Epoch 25, Loss: 1.1794112131876104, Accuracy: 0.58128
...
Epoch 27, Loss: 1.1659145609801993, Accuracy: 0.58936
Epoch 28, Loss: 1.1495457769507338, Accuracy: 0.5918
Epoch 29, Loss: 1.148694600443096, Accuracy: 0.59714
Epoch 30, Loss: 1.140139588538338, Accuracy: 0.59648
```



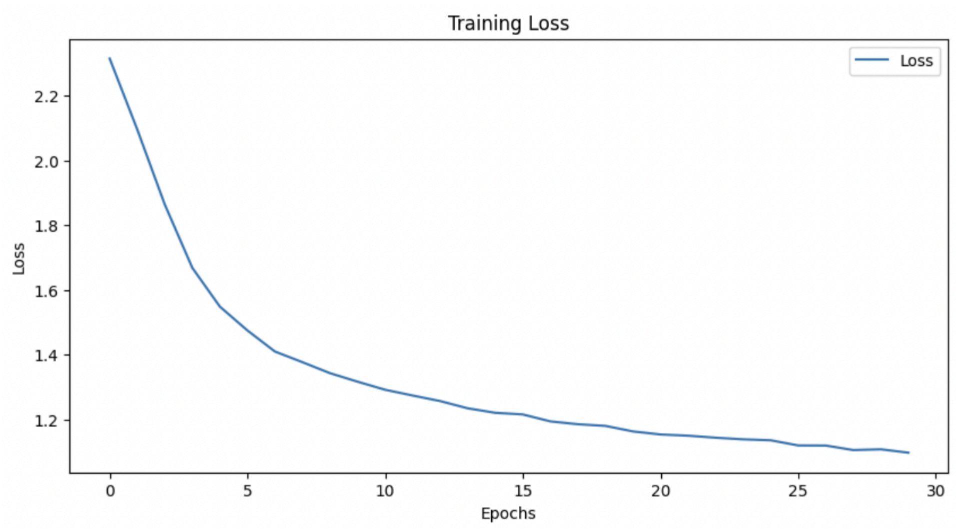


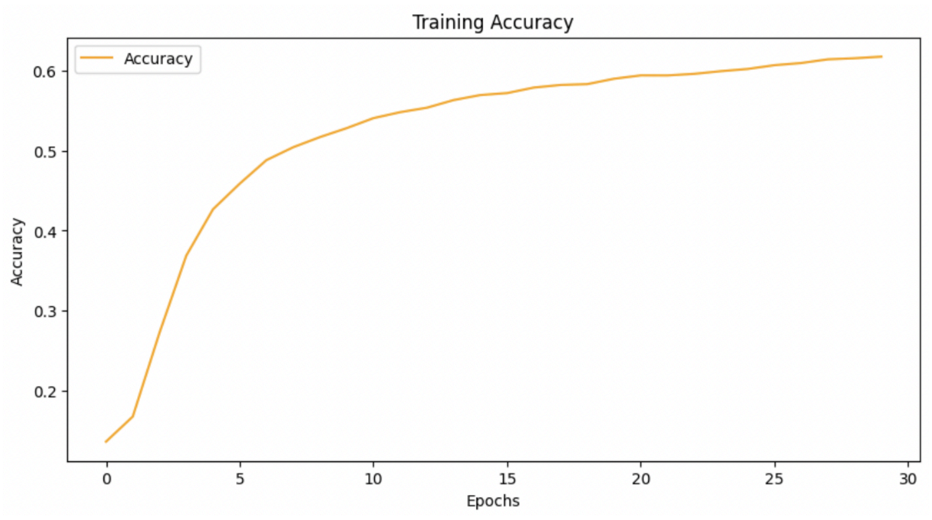
3.2 Batch Normalization (BN) and Dropout

Two figures indicated that the trend of training loss and accuracy are similar to CNN. However, the curves are steeper than CNN, but the loss starts at 2.31, which is higher than CNN, and ends at 1.09, which is less than CNN. The accuracy is contradictory.



```
Epoch 1, Loss: 2.314533574959201, Accuracy: 0.13668
Epoch 2, Loss: 2.095954865140988, Accuracy: 0.16794
Epoch 3, Loss: 1.8640649899497361, Accuracy: 0.27266
Epoch 4, Loss: 1.6684939765259432, Accuracy: 0.36876
Epoch 5, Loss: 1.5487118171304084, Accuracy: 0.42668
Epoch 6, Loss: 1.4751311174743926, Accuracy: 0.4587
Epoch 7, Loss: 1.4100200132945615, Accuracy: 0.48808
Epoch 8, Loss: 1.3770739599262052, Accuracy: 0.50414
Epoch 9, Loss: 1.3431669061293687, Accuracy: 0.5168
Epoch 10, Loss: 1.316917135130109, Accuracy: 0.52796
Epoch 11, Loss: 1.2919758276256454, Accuracy: 0.54044
Epoch 12, Loss: 1.274201213017754, Accuracy: 0.5479
Epoch 13, Loss: 1.2570361914994466, Accuracy: 0.55346
Epoch 14, Loss: 1.2347171224291673, Accuracy: 0.563
Epoch 15, Loss: 1.2207986476933559, Accuracy: 0.5693
Epoch 16, Loss: 1.2157904112430484, Accuracy: 0.57174
Epoch 17, Loss: 1.1943265292650598, Accuracy: 0.57862
Epoch 18, Loss: 1.185569247473841, Accuracy: 0.5819
Epoch 19, Loss: 1.1804299139610641, Accuracy: 0.58292
Epoch 20, Loss: 1.1633608928879204, Accuracy: 0.58964
Epoch 21, Loss: 1.153884472825643, Accuracy: 0.59384
Epoch 22, Loss: 1.1501489700106404, Accuracy: 0.59378
Epoch 23, Loss: 1.1439450114889218, Accuracy: 0.59576
Epoch 24, Loss: 1.1389693511111656, Accuracy: 0.5991
Epoch 25, Loss: 1.1359676019005154, Accuracy: 0.60186
...
Epoch 27, Loss: 1.11975951549952, Accuracy: 0.60926
Epoch 28, Loss: 1.1057597795868164, Accuracy: 0.6139
Epoch 29, Loss: 1.10809863010026, Accuracy: 0.61522
Epoch 30, Loss: 1.0977160690538108, Accuracy: 0.61722
```





3.3 Comparison and Test

Thus, we can draw the conclusion that for more than 5 epochs, the CNN with Batch Normalization and Dropout can have a better accuracy.

To testify, use a random function to find one image from the dataset, both algorithms can match them with correct labels.

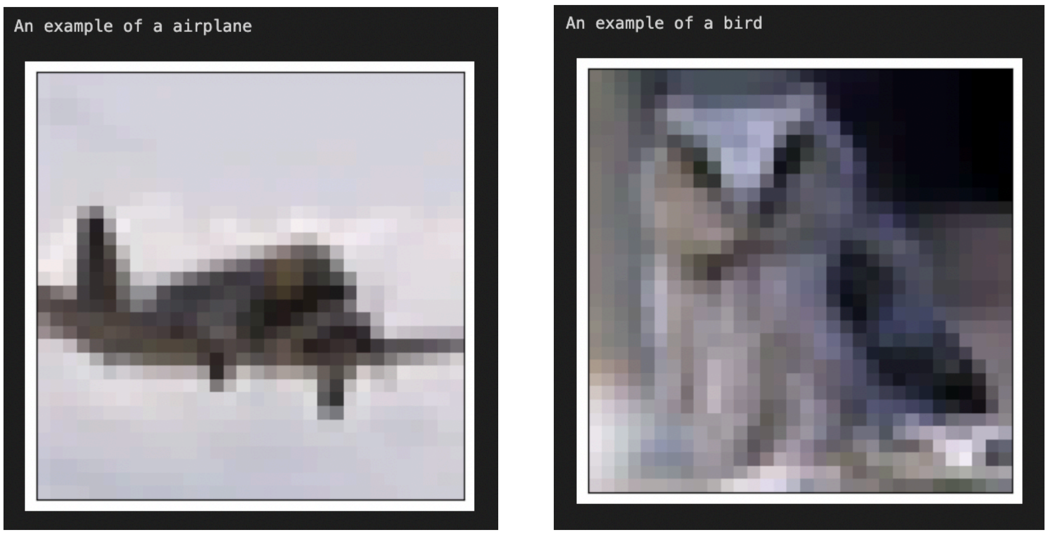


Fig.5. The Test Results of each algorithm

4. Difficulties and Solutions

4.1 Process the Dataset.

CIFAR-10 dataset is a challenging and complex dataset, and it is needed for well-developed data-processing structures, including ‘unpickle’ and ‘reshape’ functions. When debugging the code, I’ve met several error:

a. IndexError: Dimension out of range (expected to be in range of [-1, 0], but got 1)

It’s an issue with tensor dimensionality in your PyTorch model. This error typically occurs during operations that expect tensors of a specific shape or during reshaping operations, but the actual tensor doesn't meet these expectations. Here, I needed to ‘unflatten’, which is used to reshape a flat tensor into a specified shape, and it requires the specified dimensions to exist in the tensor.

b. ValueError: operands could not be broadcast together with shapes (0,) (10000,3072)

It’s occurring because I attempted to use the += operator to append numpy arrays to a list in CIFAR10 class. When doing self.image_data += data_dict[b'data'], Python attempts to add the arrays element-wise instead of appending them. Here, I need to initialize ‘self.image_data’ and ‘self.image_labels’ as empty lists and then append each array using ‘list.extend()’ or accumulate arrays and then use ‘numpy.vstack()’ (or ‘numpy.concatenate()’ for 1D arrays like labels) after the loop.

4.2 Build the Model.

Setting parameters for the CNN model can be tricky. Errors always happened when sizes weren’t matched.

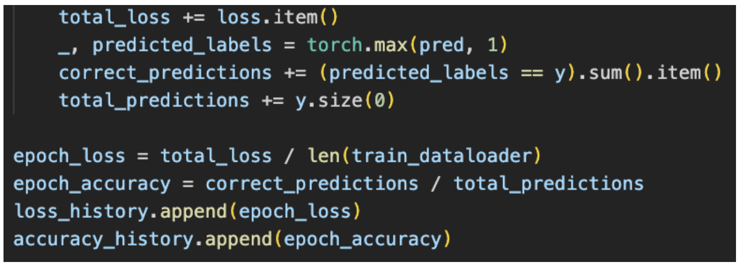
a. RuntimeError: unflatten: Provided sizes [3, 32, 32] don't multiply up to the size of dim 1 (3) in the input tensor

It suggested an issue in the forward method of the model, specifically with the unflatten operation. The error occurred because the operation attempted to reshape an input tensor into dimensions that didn't match its total number of elements. It needed to be unflatten at the beginning of the forward method because the ‘DataLoader’ should already provide the images in the correct shape, [batch_size, nels, height, width]. Added ‘x = x.view(-1, 32 * 8 * 8)’ before the hidden layer.

4.3 Draw the Figures for Loss and Accuracy

It’s difficult to calculate their loss function and accuracy function. The figures are supposed to be clear and comparable as epochs increased.

We need to use ‘.item()’ to get the loss value as a Python float and use division to get loss for each epoch.



```
    total_loss += loss.item()
    _, predicted_labels = torch.max(pred, 1)
    correct_predictions += (predicted_labels == y).sum().item()
    total_predictions += y.size(0)

epoch_loss = total_loss / len(train_dataloader)
epoch_accuracy = correct_predictions / total_predictions
loss_history.append(epoch_loss)
accuracy_history.append(epoch_accuracy)
```

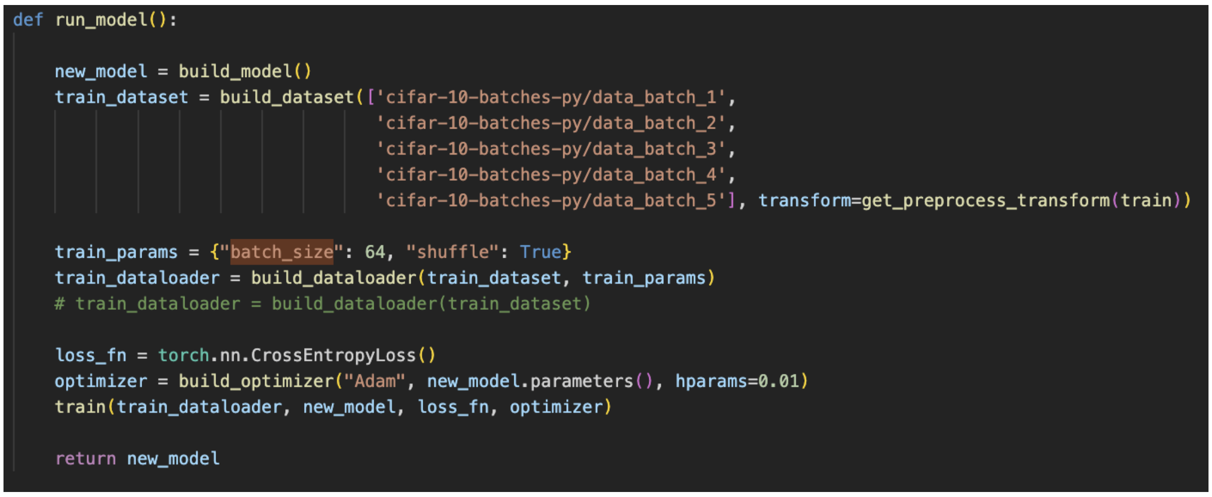
Fig.6. Calculate The Loss and Accuracy Functions

5 References

[1] Dataset: https://www.cs.toronto.edu/~kriz/cifar.html

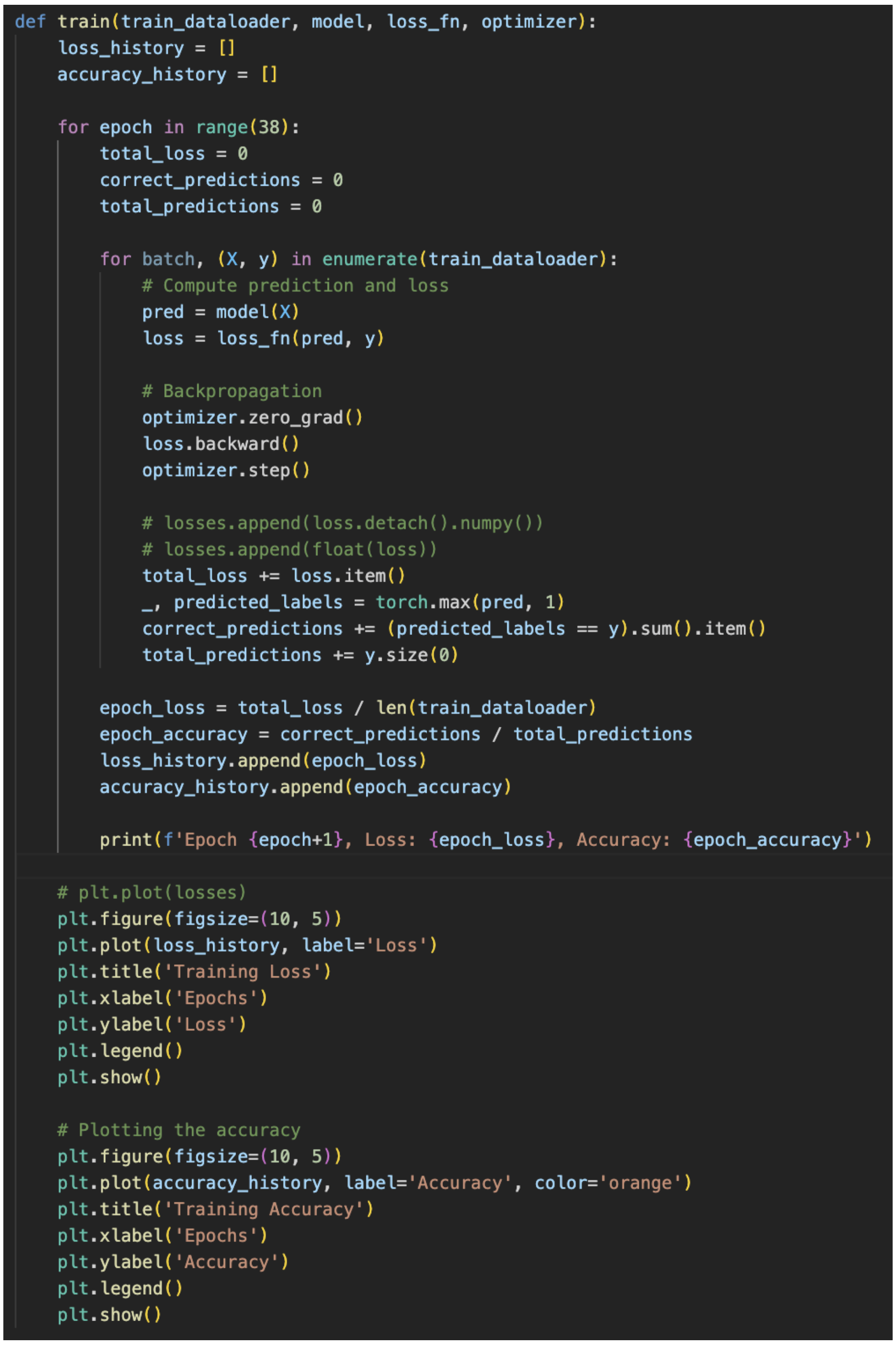
[2] Exam and polish code: https://chat.openai.com/c/5e2164fc-eda3-4c00-8456-0b6406e96f5f

6 Report Appendix (Two).



```
def run_model():

    new_model = build_model()
    train_dataset = build_dataset(['cifar-10-batches-py/data_batch_1',
                                   'cifar-10-batches-py/data_batch_2',
                                   'cifar-10-batches-py/data_batch_3',
                                   'cifar-10-batches-py/data_batch_4',
                                   'cifar-10-batches-py/data_batch_5'], transform=get_preprocess_transform(train))

    train_params = {"batch_size": 64, "shuffle": True}
    train_dataloader = build_dataloader(train_dataset, train_params)
    # train_dataloader = build_dataloader(train_dataset)

    loss_fn = torch.nn.CrossEntropyLoss()
    optimizer = build_optimizer("Adam", new_model.parameters(), hparams=0.01)
    train(train_dataloader, new_model, loss_fn, optimizer)

    return new_model
```

Fig.7. Run_model Function



```python
def train(train_dataloader, model, loss_fn, optimizer):
    loss_history = []
    accuracy_history = []

    for epoch in range(38):
        total_loss = 0
        correct_predictions = 0
        total_predictions = 0

        for batch, (X, y) in enumerate(train_dataloader):
            # Compute prediction and loss
            pred = model(X)
            loss = loss_fn(pred, y)

            # Backpropagation
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # losses.append(loss.detach().numpy())
            # losses.append(float(loss))
            total_loss += loss.item()
            _, predicted_labels = torch.max(pred, 1)
            correct_predictions += (predicted_labels == y).sum().item()
            total_predictions += y.size(0)

        epoch_loss = total_loss / len(train_dataloader)
        epoch_accuracy = correct_predictions / total_predictions
        loss_history.append(epoch_loss)
        accuracy_history.append(epoch_accuracy)

        print(f'Epoch {epoch+1}, Loss: {epoch_loss}, Accuracy: {epoch_accuracy}')

    # plt.plot(losses)
    plt.figure(figsize=(10, 5))
    plt.plot(loss_history, label='Loss')
    plt.title('Training Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

    # Plotting the accuracy
    plt.figure(figsize=(10, 5))
    plt.plot(accuracy_history, label='Accuracy', color='orange')
    plt.title('Training Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()
```

Fig.8. Train Function