ECE449 Small Project

Name: Kong Zitai ID:3190110306

Preprocessing

3.1 load data

3.1.1 Dimension of Training Set and Test Set

Dimension of training set:

(40000, 32, 32, 3), means 40000 pictures, each with 32×32 size in RGB 3 channels.

Dimension of test set:

(10000, 32, 32, 3), means 40000 pictures, each with 32×32 size in RGB 3 channels.

3.1.2 Ratio of training set to validation set

Ratio of training set to validation set = 0.8:0.2 = 4:1

3.1.3 Function of class CIFAR10_from_array

This class provides 3 functions which act 3 different usages. For __getitem__, it will read the indexed X and Y values from data and preprocess it by the transform method defined (including augmentation transform, conversion to tensor and normalization, and finally it returns the image and label combined in a pair. For __len__, it indicates the total size of the dataset. For plot_image, it plots the image of given data image.

3.2 data augmentation

3.2.1 Augmentation Methods Used in This Project

I try to test more methods provided by transforms.Compose(). These includes

(1) resize it to a given size 40×40
(2) crop the image back to 32×32 at a random place
(3) Flip the image horizontally with 0.5 probability
(4) Flip the image vertically with 0.5 probability
(5) Rotate the image

```
train_transform_aug = transforms.Compose([
###
#insert your code here
#implement some data augmentation methods here
#for example, transforms.Resize((40, 40)),

###
    transforms.Resize((40, 40)),       #First resize it to a given size 40by40
    transforms.RandomCrop((32, 32)),   #Then cut the image back to 32by32 at a random place
    transforms.RandomHorizontalFlip(), #Flip the image horizontally with 0.5 probability
    transforms.RandomVerticalFlip(),   #Flip the image vertically with 0.5 probability
    transforms.RandomRotation(15),     #Rotate the image

    transforms.ToTensor(),
    normalize_dataset(X_train)
])
```

3.2.2 Benefits of Data Augmentation

With data augmentation, we can extend our dataset greatly and redue costs of collecting and labeling data. This is especially useful for a small raw dataset. Also, data augmentation improves the model prediction accuracy because it can add more training data into the models, prevent data scarcity for better models, reduces data overfitting (i.e. an error in statistics, it means a function corresponds too closely to a limited set of data points) and create variability in data. Besides, it can increase generalization ability of the models and help to resolve class imbalance issues in classification

3.3 dataset normalization

3.3.1 Dataset Normalized Method

In this project, each channel of RGB of the picture is normalized with img = img - mean / std. The mean is taken by calculating means on each channel and finally divide it by 255. The standard is taken by calculating standards on each channel and finally divide it by 255. By doing these we can change the [0,255] values to [-1,1].

```
# Normalize for R, G, B with img = img - mean / std
def normalize_dataset(data):
    mean = data.mean(axis=(0,1,2)) / 255.0
    std = data.std(axis=(0,1,2)) / 255.0
    normalize = transforms.Normalize(mean=mean, std=std)
    return normalize
```

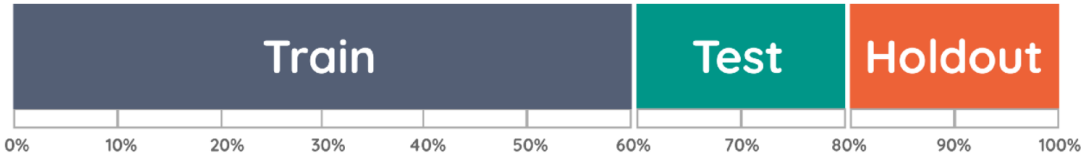
3.3.2 Benefit of Normalization

The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information. Normalization improves the numerical stability of the model and often reduces training time. It avoids the problem caused by data in very different scales.

Train

3.4 Validation

This project uses Train/test split or holdout method. We first split the dataset into two parts, the training set and the testing set, with 40000 and 10000 pictures in our project. And we further split 20% of the training set as the validation set randomly. In our program, we set the random state to 42.




```
# split the validation set out
from sklearn.model_selection import train_test_split
X_train_split, X_val_split, Y_train_split, Y_val_split = train_test_split(
    X_train, Y_train, test_size=0.2, random_state=42)
```

3.5 build model

According to the hint of the program file, I use the LeNet-5 CNN structure. The overall input is a 32×32×3 matrix, and the overall output is a length-10 vector.

For the first convolution layer, the input is a 32×32×3 matrix and output is a 28×28×16 matrix.

For the first pooling layer, the input is a 28×28×16 matrix and output is a 14×14×16 matrix.

For the second convolution layer, the input is a 14×14×16 matrix and output is a 10×10×32 matrix.

For the first pooling layer, the input is a 10×10×32 matrix and output is a 5×5×32 matrix.

For the first full connection layer, the input is a 5×5×32 length vector and the output is a 120 length vector.

For the second full connection layer, the input is a 120 length vector and the output is a 84 length vector.

For the third full connection layer, the input is an 84 length vector and the output is a 10 length vector.

```
# Build the model
class Net(nn.Module):
        #implement your own network here
        #you can use any simple CNN structure learned from class
        #for example, LeNet

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=5)
        self.pool1 = nn.MaxPool2d(kernel_size=(2,2))
        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5)
        self.pool2 = nn.MaxPool2d(kernel_size=(2,2))
        self.fc1 = nn.Linear(32 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)



    def forward(self, x):
        x = self.pool1(F.relu(self.conv1(x)))
        x = self.pool2(F.relu(self.conv2(x)))
        x = x.view(-1, 32 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

3.6 loss&optimizer

According the suggestions in the lab document, I use Cross Entropy loss and Adam optimizer.

```
#insert your code here
#implement criterion(also known as loss funtion) and optimizer here
#we suggest you to use CrossEntropyLoss and Adam

criterion = nn.CrossEntropyLoss() #loss function
optimizer = optim.Adam(net.parameters(), lr=learning_rate)
```

The loss calculation is:

$$\text{CrossEntropy(P, Q)} = -\sum_i [y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})]$$

Test

3.7 confusion matrix

A confusion matrix is a technique for summarizing the performance of a classification algorithm. Shown below, there are two dimensions of a confusion matrix, one is the prediction coordinate and another is the ground truth coordinate. Each point on the coordinate is stand for a certain class. Each point in the matrix is the number of elements for a kind of prediction that is in a kind of ground truth. From this , we can see that only the numbers on the diagonal are those with right predictions. Thus, with this matrix we can calculate the accuracy of prediction of each class.