EE456 Section 001 Mini Project 2 Report

CNN TO CLASSIFY CIFAR10 DATASET

Sahin, Efe
PENN STATE UNIVERSITY | DECEMBER 5, 2022

Overview

This Project is about using CNN to solve the image classification problem where the CIFAR10 dataset will be used. CIFAR10 is an image dataset of 32x32 images with 3 channels (R,G,B). There are in total 60000 images which consist of 10000 per class (therefore there are 10 classes). Each class represents a specific category like, bird, truck, ship etc. The CNN is trained based on a training set randomly augmented (for this project augmentations include random flips and random crops) and selected from the main dataset.

Implementation

It is important to mention that the code I ran "cifar10NN.vfinal.ipynb" uses python 3.8.15 with several other libraries like torch, torchvision, matplotlib and numpy inside of an Anaconda environment. I have installed the following versions to benefit the GPU of my Mac M1 PC.

- Torch: 1.14.0.dev20221130
- Torchvision: 0.15.0.dev20221130
- Matplotlib: (latest is fine)
- Numpy: 1.24.0rc1 (latest is fine)

The code is able to select the proper device to maximize the possible speed benefits.

```
if torch.backends.mps.is_available:
    processor='mps'
elif torch.cuda.is_available():
    processor='cuda0'
else:
    processor='cpu'
device = torch.device(device=processor)
print(f"device is set as: {device}")
```

✓ 0.1s

```
device is set as: mps
```

FIGURE 1: MPS IS SPECIAL TO M1 APPLE COMPUTERS, CUDA:0 IS THE OTHER OPTION, CPU IS THE LAST OPTION.

Then it is time to split the main dataset into train, validation, testing sets. I decided to use 50000 randomly picked and augmented images from the main dataset then split it into 35000 and 15000 images of train and validation data sets respectively. The remaining 10000 images will be used for testing after the Net has been trained.

```
1  normalized = transforms.Normalize((0.49139968, 0.48215827 ,0.44653124), (0.24703233, 0.24348505, 0.26158768))
2  flip = transforms.RandomHorizontalFlip()
3  crop = transforms.RandomCrop(size=32)
4
5  trainValidTransform = transforms.Compose([transforms.ToTensor(), normalized, flip, crop])
6  testTransform = transforms.Compose([transforms.ToTensor(), normalized])
7
8  trainvalidSet = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=trainValidTransform)
9  trainSet , validSet = data.random_split(trainvalidSet, [35000, 15000])
10 testSet = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=testTransform)
11
12 trainLoader = torch.utils.data.DataLoader(trainSet, batch_size=batchSize, shuffle=True, num_workers=2)
13 validLoader = torch.utils.data.DataLoader(validSet, batch_size=batchSize, shuffle=True, num_workers=2)
14 testLoader = torch.utils.data.DataLoader(testSet, batch_size=batchSize, shuffle=False, num_workers=2)
15
16 classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
17 print('Train data set:', len(trainSet))
18 print('Valid data set:', len(validSet))
19 print('Test data set:', len(testSet))
```

✓ 1.5s

```
Files already downloaded and verified
Files already downloaded and verified
Train data set: 35000
Valid data set: 15000
Test data set: 10000
```

FIGURE 2: FIRST NORMALIZATION OF CIFAR10, FLIP, CROP ARE DECIDED IN LINES 1-3. THEN TRAIN, VALID, TESTING SETS HAVE BEEN FORMED IN LINES 5-10. THEN THEY ARE LOADED IN LINES 12-14.

The Structure of the neural net consists of 3 convolutional layers, 2 pooling and followed by 3 fully connected linear layers. The Net is defined as the class ConvNet1 and made as the model object. Pytorch can show detailed summary for cuda0 or cpu processors but not for mps. Then for loss, CrossEntropyLoss is used since this net will classify multiple classes. As optimizer SGD i.e stochastic gradient descent is used to optimize the parameters of the net.

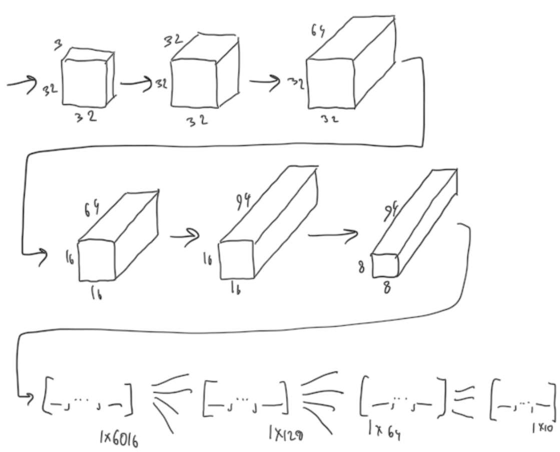

FIGURE 3: A CNN HEURISTIC.

```
import torch.nn as nn
import torch.nn.functional as F

class ConvNet1(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=3,out_channels=32, kernel_size=3, stride=1, padding=1) # 32,32,32
        self.conv2 = nn.Conv2d(in_channels=32,out_channels=64, kernel_size=3, stride=1, padding=1) #64,32,32
        self.pool1 = nn.MaxPool2d(kernel_size=2,stride=2,padding=0) #64,16,16
        self.conv3 = nn.Conv2d(in_channels=64,out_channels=94, kernel_size=3, stride=1, padding=1) #94,16,16
        self.pool2 = nn.MaxPool2d(kernel_size=2,stride=2,padding=0) #94,8,8

        self.fc1 = nn.Linear(in_features=94*8*8,out_features=128)
        self.fc2 = nn.Linear(in_features=128,out_features=64)
        self.fc3 = nn.Linear(in_features=64,out_features=10)

    def forward(self, x):
        x= F.relu(self.conv1(x))
        x= F.relu(self.conv2(x))
        x= self.pool1(x)
        x= F.relu(self.conv3(x))
        x= self.pool2(x)

        x= torch.flatten(x,start_dim=1)
        x= F.relu(self.fc1(x))
        x= F.relu(self.fc2(x))
        x= self.fc3(x)
        return x

model = ConvNet1().to(device)

if processor=='cpu' or processor=='cuda0':
    from torchsummary import summary
    summary(model, (3, 32, 32))

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=learnRate, momentum=0.9)
```

FIGURE 4: THE CLASS OF COVNET1.

Results/Conclusion

During the training process of the CNN for 15 epochs with 20 image batches, I displayed the trainingloss, validationloss, trainingaccuracy and validationaccuracies. It seemed from the values that 15 epochs were enough to achieve training accuracy above 90. I also realized that training accuracy surpassed validation accuracy. Similar relationship can be seen from the calculated loss values.

```
For epoch: 1 |---| trainloss=1.653847 validloss=1.238903 | trainaccuracy=39.545715 valid=55.360001
For epoch: 2 |---| trainloss=1.102419 validloss=1.069913 | trainaccuracy=60.237144 valid=63.366669
For epoch: 3 |---| trainloss=0.882153 validloss=0.813558 | trainaccuracy=68.842857 valid=71.366669
For epoch: 4 |---| trainloss=0.755343 validloss=0.804646 | trainaccuracy=73.371429 valid=72.546669
For epoch: 5 |---| trainloss=0.655973 validloss=0.813546 | trainaccuracy=77.405716 valid=72.139999
For epoch: 6 |---| trainloss=0.579830 validloss=0.780484 | trainaccuracy=79.902855 valid=73.986664
For epoch: 7 |---| trainloss=0.497934 validloss=0.805496 | trainaccuracy=82.534286 valid=74.146667
For epoch: 8 |---| trainloss=0.445979 validloss=0.735752 | trainaccuracy=84.494286 valid=76.153336
For epoch: 9 |---| trainloss=0.392063 validloss=0.876399 | trainaccuracy=86.491432 valid=74.366669
For epoch: 10 |---| trainloss=0.353841 validloss=0.790553 | trainaccuracy=88.011429 valid=75.453331
For epoch: 11 |---| trainloss=0.315596 validloss=0.879806 | trainaccuracy=89.245712 valid=74.693336
For epoch: 12 |---| trainloss=0.268274 validloss=0.910542 | trainaccuracy=90.785713 valid=75.566666
For epoch: 13 |---| trainloss=0.256548 validloss=1.030078 | trainaccuracy=91.314285 valid=74.546669
For epoch: 14 |---| trainloss=0.224269 validloss=1.036487 | trainaccuracy=92.540001 valid=75.040001
For epoch: 15 |---| trainloss=0.212034 validloss=1.004622 | trainaccuracy=92.868568 valid=75.306671
Finished Training
```

FIGURE 5: EPOCHS AND RESPECTIVE STATISTICS OF THE CNN DURING TRAINING

The following figures 6 and 7 show the plots of loss and accuracy for the CNN training.

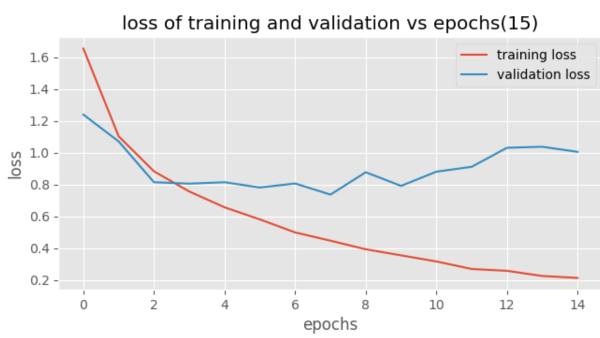

FIGURE 6

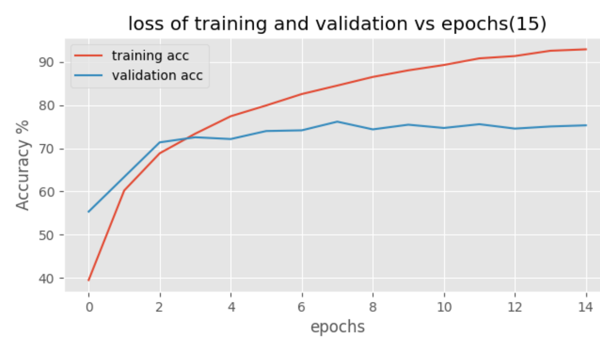

FIGURE 7

Later the CNN is put into the testing process with the testing dataset. The following accuracy of 75% has been observed, which is very similar accuracy percentage from the validation. The figure 8 also shows the accuracy levels of predicting each class correctly

```
Accuracy of the network on the 10000 test images: 75.1 %
Accuracy for class: plane is 80.4 %
Accuracy for class: car is 87.9 %
Accuracy for class: bird is 59.3 %
Accuracy for class: cat is 58.9 %
Accuracy for class: deer is 74.6 %
Accuracy for class: dog is 53.9 %
Accuracy for class: frog is 83.1 %
Accuracy for class: horse is 81.2 %
Accuracy for class: ship is 88.2 %
Accuracy for class: truck is 83.5 %
```

FIGURE 8:

Then I selected a random batch of 20 images to display with their labels vs what the neural net predicted.

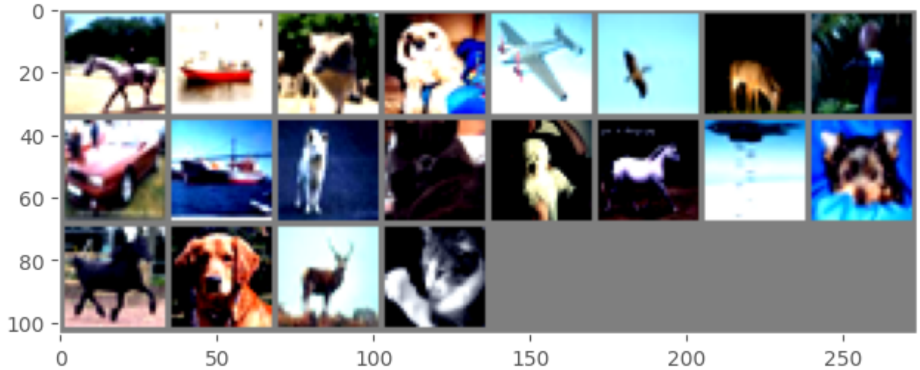

FIGURE 9: A BATCH OF IMAGES

```
Actualval: horse | ship  | bird  | dog   | plane | bird  | deer  | bird  | car   | ship  | dog   | cat   | dog   | horse | plane | dog
| horse | dog   | deer  | cat   |
Predicted: horse | ship  | bird  | dog   | plane | plane | deer  | bird  | car   | ship  | horse | cat   | dog   | horse | plane | dog
| horse | dog   | deer  | cat   |
```

FIGURE 10: THE BATCH OF IMAGES WITH THEIR RESPECTIVE ACTUAL LABELS AND PREDICTED VALUES.

Lastly the confusion matrix resulted a plot of Actual labels of classes vs the number of predictions based on the training set. It seems that the diagonally, meaning the correct predictions, have been the majority as those boxes were darker than others.

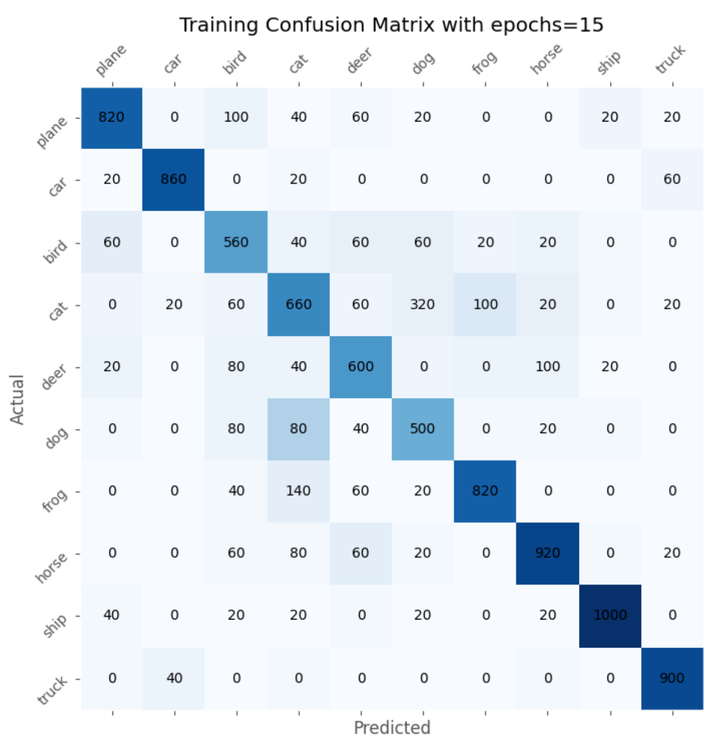

FIGURE 11: CONFUSION MATRIX OF CIFAR10 ACCORDING TO THE CNN TRAINED IN THIS PROJECT