UDACITY

Logout

‹ Return to Classroom

Train a Character Recognition System with MNIST

REVIEW | HISTORY

Meets Specifications

Congrats

You have successfully passed the project. The set of steps have been coded rightly.

Section 1: Data Loading and Exploration

✓ Data is preprocessed and converted to a tensor, either using the .ToTensor() transform from `torchvision.transforms` or simply manually with torch.Tensor.

Nice Work

The data preprocessing steps are rightly coded.
A bit of improvement can be made by using augmentation transforms with the training data. This will increase the model's generalizing power. But make sure to use it only with the training data.

✓ A `DataLoader` object for both train and test sets has been created using the train and test sets loaded from `torchvision`.

The data loaders are rightly used to load the data of training and testing set.

✓ Notebook contains code which shows the size and shape of the training and test data.

The provided function or some other method (e.g. plt.imshow) is used to print one or more images from the dataset

The code to plot images in the training data loader is all correct. The purpose of plotting training images was to see the changes in images due to transformations that have been performed while preprocessing.

```
## This cell contains a function for showing 5 images from a dataloader – DO NOT CHANGE THE CONTE
NTS! ##
def show5(img_loader):
    dataiter = iter(img_loader)

    batch = next(dataiter)
    labels = batch[1][0:5]
    images = batch[0][0:5]
    for i in range(5):
        print(int(labels[i].detach()))

        image = images[i].numpy()
        plt.imshow(image.T.squeeze().T)
        plt.show()
```

✓ The submission contains a justification of necessary preprocessing steps (e.g. flattening, converting to tensor, normalization) in a comment or (preferably) a markdown block.

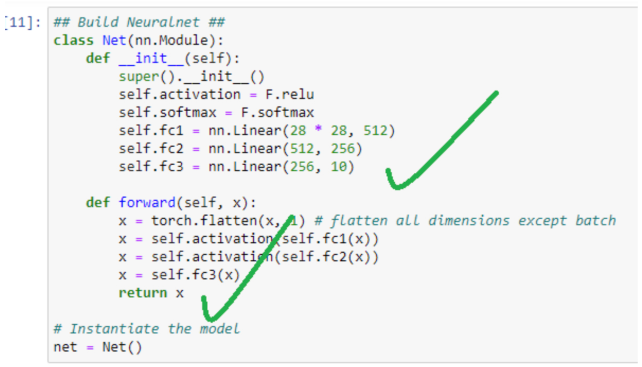
Section 2: Model Design and Training

✓ A `Model` or `Sequential` class is created with at least two hidden layers and implements a `forward` method that outputs a prediction probability for each of the 10 classes using softmax.

Awesome

The network architecture is coded correctly.
The classifier network is coded rightly using 3 Linear layers. You have rightly used the ReLU activation function with the layers.

It would be great if you try using the dropout layers as well. This will help to reduce the chances of overfitting.



```
[11]: ## Build Neuralnet ##
      class Net(nn.Module):
          def __init__(self):
              super().__init__()
              self.activation = F.relu
              self.softmax = F.softmax
              self.fc1 = nn.Linear(28 * 28, 512)
              self.fc2 = nn.Linear(512, 256)
              self.fc3 = nn.Linear(256, 10)

          def forward(self, x):
              x = torch.flatten(x, 1) # flatten all dimensions except batch
              x = self.activation(self.fc1(x))
              x = self.activation(self.fc2(x))
              x = self.fc3(x)
              return x

      # Instantiate the model
      net = Net()
```

Specify a loss function and an optimizer, and instantiate the model.

✓ A loss function that works for classification tasks is specified.

```
criterion = nn.CrossEntropyLoss()
```

The CrossEntropy loss function is perfectly used.

✓ Any optimizer from `torch.optim` is used to minimize the loss function.

Section 3: Model Testing and Evaluation

✓ The test `DataLoader` is used to get predictions from the neural network and compare the predictions to the true labels.

The testloader are coded correctly. You have used the transforms and loaded the data rightly.

✓ Hyperparameters are modified to attempt to improve accuracy and the model achieves at least 90% classification accuracy.

Perfect

The hyperparameter values are well chosen. Models is trained perfectly and it's able to learn the patterns.

```
Test Accuracy of     5: 96% (125/130)
Test Accuracy of     6: 99% (125/126)
Test Accuracy of     7: 96% (153/158)
Test Accuracy of     8: 97% (132/135)
Test Accuracy of     9: 95% (137/143)

Test Accuracy (Overall): 97% (1397/1432)
```

✓ The `torch.save()` function is used to save the weights of the trained model.

Saving step of model is coded rightly. ✌🏽

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

START