Edison Chen, ebc5802
11/14/2023

Machine Learning
HW#5 Computer Vision using Convolutional NNs in PyTorch

Bonus Question:
Experiment with the hyperparameter patience (you may set it equal to 2 and 4 for instance) in the fit_with_early_stopping() function, and report your observations

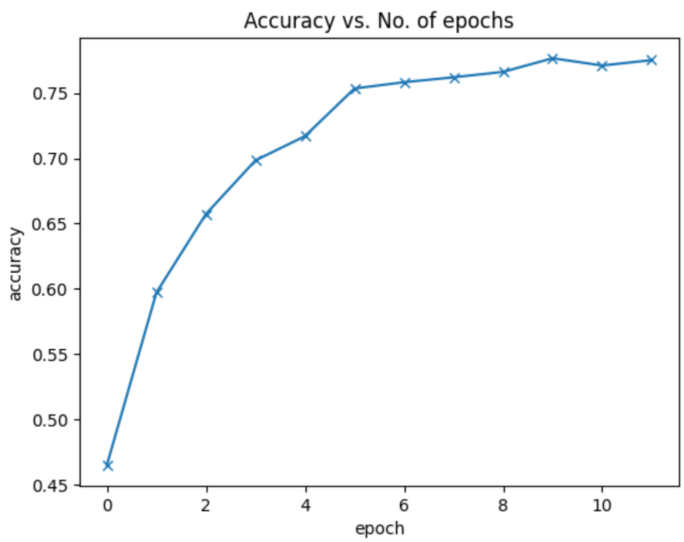
Answer:
The patience affects how much allowance the program gives for an increase in validation loss. For example, a patience of 1 stopped the program immediately after just 1 increase in validation loss. This resulted in a very ealy stop after just 2 epochs. The validation accuracy wasn't as good as when patience was > 1. **Patience = 2 produced the best results** with a validation accuracy of 0.7650390863418579 with the second-best accuracy being 0.764453113079071 with patience = 3.

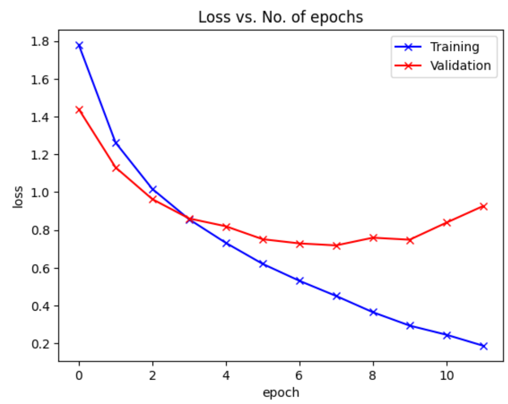
Patience = 1:
{'val_loss': 1.0575330257415771, 'val_acc': 0.7626953125}

Epoch [0], train_loss: 0.2081, val_loss: 0.9186, val_acc: 0.7729
Epoch [1], train_loss: 0.1446, val_loss: 1.0705, val_acc: 0.7663
Early stopping at epoch 1

Patience = 2:
{'val_loss': 0.9376282691955566, 'val_acc': 0.7650390863418579}

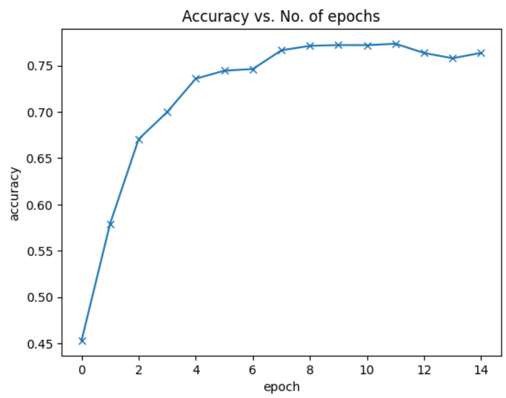
Epoch [0], train_loss: 1.7795, val_loss: 1.4387, val_acc: 0.4651
Epoch [1], train_loss: 1.2620, val_loss: 1.1309, val_acc: 0.5972
Epoch [2], train_loss: 1.0170, val_loss: 0.9628, val_acc: 0.6573
Epoch [3], train_loss: 0.8562, val_loss: 0.8607, val_acc: 0.6984
Epoch [4], train_loss: 0.7315, val_loss: 0.8193, val_acc: 0.7169
Epoch [5], train_loss: 0.6205, val_loss: 0.7509, val_acc: 0.7534
Epoch [6], train_loss: 0.5309, val_loss: 0.7286, val_acc: 0.7582
Epoch [7], train_loss: 0.4505, val_loss: 0.7181, val_acc: 0.7620
Epoch [8], train_loss: 0.3645, val_loss: 0.7590, val_acc: 0.7661
Epoch [9], train_loss: 0.2936, val_loss: 0.7482, val_acc: 0.7766
Epoch [10], train_loss: 0.2453, val_loss: 0.8385, val_acc: 0.7710
Epoch [11], train_loss: 0.1871, val_loss: 0.9264, val_acc: 0.7751
Early stopping at epoch 11

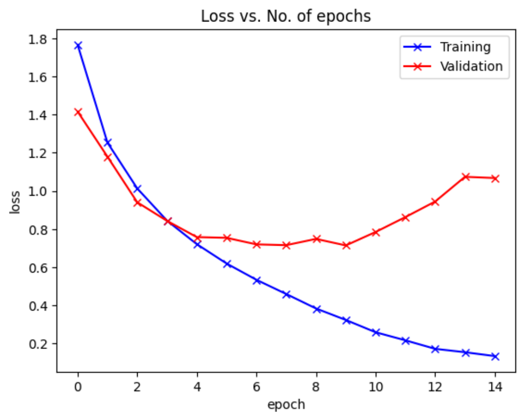




Patience = 3:
{'val_loss': 1.0728938579559326, 'val_acc': 0.764453113079071}

history_with_early_stopping = fit_with_early_stopping(num_epochs, lr, model3, train_dl, val_dl, opt_func)
output
Epoch [0], train_loss: 1.7655, val_loss: 1.4156, val_acc: 0.4529
Epoch [1], train_loss: 1.2532, val_loss: 1.1793, val_acc: 0.5792
Epoch [2], train_loss: 1.0130, val_loss: 0.9405, val_acc: 0.6706
Epoch [3], train_loss: 0.8440, val_loss: 0.8431, val_acc: 0.6999
Epoch [4], train_loss: 0.7211, val_loss: 0.7579, val_acc: 0.7359
Epoch [5], train_loss: 0.6195, val_loss: 0.7542, val_acc: 0.7445
Epoch [6], train_loss: 0.5343, val_loss: 0.7200, val_acc: 0.7463
Epoch [7], train_loss: 0.4599, val_loss: 0.7158, val_acc: 0.7665
Epoch [8], train_loss: 0.3841, val_loss: 0.7487, val_acc: 0.7714
Epoch [9], train_loss: 0.3239, val_loss: 0.7143, val_acc: 0.7722
Epoch [10], train_loss: 0.2595, val_loss: 0.7848, val_acc: 0.7721
Epoch [11], train_loss: 0.2172, val_loss: 0.8628, val_acc: 0.7737
Epoch [12], train_loss: 0.1725, val_loss: 0.9444, val_acc: 0.7636
Epoch [13], train_loss: 0.1548, val_loss: 1.0740, val_acc: 0.7580
Epoch [14], train_loss: 0.1346, val_loss: 1.0676, val_acc: 0.7638
Early stopping at epoch 14






```
[75] # -----------
     # Graded Cell
     # -----------
     dog_dir = "/train/dog"
     dog_files = os.listdir(data_dir + dog_dir)
     len_dog = len(dog_files)
     print('No. of training examples for dogs', len_dog)
     print(dog_files[:5])
```

```
# -----------
# Graded Cell
# -----------
cat_test_dir = "/test/cat"
cat_test_files = os.listdir(data_dir + cat_test_dir)
len_test_cat = len(cat_test_files)
print("No. of test examples for cats:", len_test_cat)
print(cat_test_files[:5])
```

```
No. of test examples for cats: 1000
['0414.png', '0646.png', '0328.png', '0129.png', '0628.png']
```

```
[80] # -----------
     # Graded Cell
     # -----------

     def length(training_data): # outputs the no. of images contained in the training set
       # one line of code
       return len(training_data)

     length(training_data)
```

```
50000
```

```
# -----------
# Graded Cell
# -----------

def shape(img): # outputs the shape of an image tensor
  # one line of code
  return img.shape

img, label = training_data[0]
img_size = shape(img)
print(img_size, label)
img
```

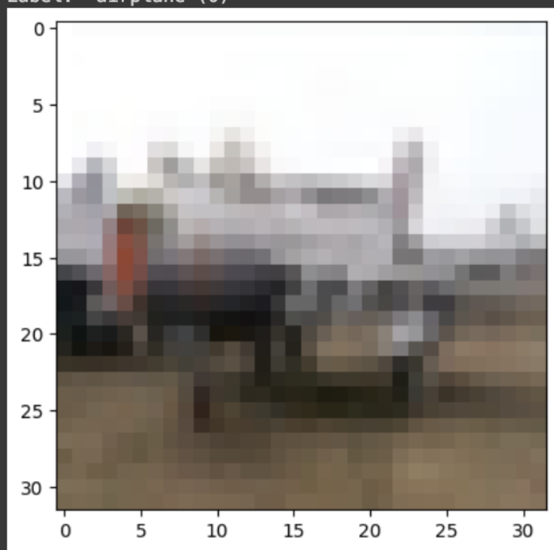
```
torch.Size([3, 32, 32]) 0
tensor([[[0.7922, 0.7922, 0.8000,  ..., 0.8118, 0.8039, 0.7961],
         [0.8078, 0.8078, 0.8118,  ..., 0.8235, 0.8157, 0.8078],
         [0.8235, 0.8275, 0.8314,  ..., 0.8392, 0.8314, 0.8235],
         ...,
         [0.8549, 0.8235, 0.7608,  ..., 0.9529, 0.9569, 0.9529],
         [0.8588, 0.8510, 0.8471,  ..., 0.9451, 0.9451, 0.9451],
         [0.8510, 0.8471, 0.8510,  ..., 0.9373, 0.9373, 0.9412]],

        [[0.8000, 0.8000, 0.8078,  ..., 0.8157, 0.8078, 0.8000],
         [0.8157, 0.8157, 0.8196,  ..., 0.8275, 0.8196, 0.8118],
         [0.8314, 0.8353, 0.8392,  ..., 0.8392, 0.8353, 0.8275],
         ...,
         [0.8510, 0.8196, 0.7608,  ..., 0.9490, 0.9490, 0.9529],
         [0.8549, 0.8471, 0.8471,  ..., 0.9412, 0.9412, 0.9412],
         [0.8471, 0.8431, 0.8471,  ..., 0.9333, 0.9333, 0.9333]],

        [[0.7804, 0.7804, 0.7882,  ..., 0.7843, 0.7804, 0.7765],
         [0.7961, 0.7961, 0.8000,  ..., 0.8039, 0.7961, 0.7882],
         [0.8118, 0.8157, 0.8235,  ..., 0.8235, 0.8157, 0.8078],
         ...,
         [0.8706, 0.8392, 0.7765,  ..., 0.9686, 0.9686, 0.9686],
         [0.8745, 0.8667, 0.8627,  ..., 0.9608, 0.9608, 0.9608],
         [0.8667, 0.8627, 0.8667,  ..., 0.9529, 0.9529, 0.9529]]])
```

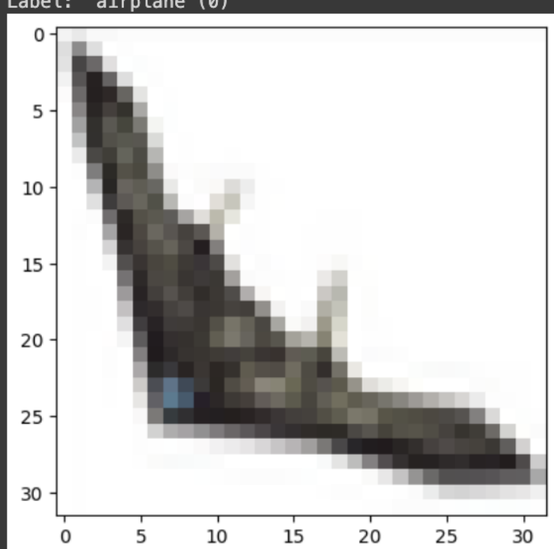
```
# -----------
# Graded Cell
# -----------
show_example(*training_data[1000])
```

Label: airplane (0)



```
[87] # -----------
# Graded Cell
# -----------
show_example(*training_data[2000])
```

Label: airplane (0)



```python
[89] # -----------
# Graded Cell
# -----------
def get_train_size(training_data, val_size): # outputs the size of the validation set
  train_length = length(training_data) - val_size
  return train_length


val_size = 5000
train_size = get_train_size(training_data, val_size)

train_ds, val_ds = random_split(training_data, [train_size, val_size])
len(train_ds), len(val_ds)
```

```
(45000, 5000)
```

```python
[94] # -----------
# Graded Cell
# -----------
def apply_filter(image, filter):
    img_rows, img_cols = image.shape  # image dimensions
    fil_rows, fil_cols = filter.shape  # filter dimensions
    out_rows, out_cols = (img_rows - fil_rows + 1, img_cols - fil_cols + 1)  # output dimensions
    output = torch.zeros([out_rows, out_cols])
    for i in range(out_rows):
        for j in range(out_cols):
            output[i,j] = torch.sum(image[i:i+fil_rows,j:j+fil_cols] * filter)
    return output
```

```python
# -----------
# Graded Cell
# -----------
class ImageClassificationBase(nn.Module):
    def training_step(self, batch):
        images, labels = batch
        ####TODO####
        # forward pass
        out = self(images)  # Generate predictions for all the images in the batch
        loss = F.cross_entropy(out, labels) # Calculate cross-entropy (CE) loss
        ############
        return loss

    def validation_step(self, batch):
        images, labels = batch
        ####TODO####
        # validation pass
        out = self(images)  # Generate predictions for all the images in the batch
        loss = F.cross_entropy(out, labels) # Calculate cross-entropy (CE) loss
        ############
        acc = accuracy(out, labels) # Calculate accuracy
        return {'val_loss': loss.detach().cpu(), 'val_acc': acc}

    def validation_epoch_end(self, outputs):
        batch_losses = [x['val_loss'] for x in outputs] # Creates a list that contains the validation
        epoch_loss = torch.stack(batch_losses).mean()   # Combines batch losses into a single tensor a
        batch_accs = [x['val_acc'] for x in outputs]    # Creates a list that contains the validation
        epoch_acc = torch.stack(batch_accs).mean()      # Combines batch accuracies into a single tens
        return {'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item()}

    def epoch_end(self, epoch, result):
        print("Epoch [{}], train_loss: {:.4f}, val_loss: {:.4f}, val_acc: {:.4f}".format(
            epoch, result['train_loss'], result['val_loss'], result['val_acc']))

def accuracy(outputs, labels):
    #####TODO####
    # generate predictions (see HINT below)
    _, preds = torch.max(outputs, dim=1) # Wrote this
    # HINT: Use torch.max(input, dim) specifying input and dim while having in mind that torch.max() r
    # along the specified dimension and the indices of those maximum values. Here, we are interested i
    ############
    return torch.tensor(torch.sum(preds == labels).item() / len(preds)).detach().cpu()
```

```python
# -----------
# Graded Cell
# -----------
class Cifar10CnnModel(ImageClassificationBase):
    def __init__(self):
        super().__init__()
        self.network = nn.Sequential(
            # input: 3 x 32 x 32
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
            # output: 32 x 32 x 32
            nn.ReLU(),
            # output: 32 x 32 x 32
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            # output: 64 x 32 x 32
            nn.ReLU(),
            # output: 64 x 32 x 32
            nn.MaxPool2d(2, 2),
            # output: 64 x 16 x 16

            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            # output: 128 x 16 x 16
            nn.ReLU(),
            # output: 128 x 16 x 16
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            # output: 128 x 16 x 16
            nn.ReLU(),
            # output: 128 x 16 x 16
            nn.MaxPool2d(2, 2),
            # output: 128 x 8 x 8

            nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
            # output: 256 x 8 x 8
            nn.ReLU(),
            # output: 256 x 8 x 8
            nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            # output: 256 x 8 x 8
            nn.ReLU(),
            # output: 256 x 8 x 8
            nn.MaxPool2d(2, 2),
            # output: 256 x 4 x 4

            nn.Flatten(),
            nn.Linear(4096, 1024),
            # output: 1024
            nn.ReLU(),
            # output: 1024
            nn.Linear(1024, 512),
            # output: 512
            nn.ReLU(),
            # output: 512
            nn.Linear(512, 10))
            # output: 10

    def forward(self, xb):
        return self.network(xb)
```

```python
# -----------
# Graded Cell
# -----------
@torch.no_grad()
def evaluate(model, val_loader):
    model.eval()
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)

def fit(epochs, lr, model, train_loader, val_loader, opt_func=torch.optim.SGD): # SGD refers to mini-b
    history = []
    optimizer = opt_func(model.parameters(), lr)
    for epoch in range(epochs):
        # Training Phase
        model.train()
        train_losses = []
        for batch in train_loader:
            loss = model.training_step(batch)
            ####TODO####
            # Arrange the following three lines in the correct order:
            #1.optimizer.step();  2. optimizer.zero_grad(); 3. loss.backward().
            loss.backward() # Calculate losses
            optimizer.step() # Step
            optimizer.zero_grad() # Reset gradients
            ############
            train_losses.append(loss.detach().cpu())
        # Validation phase
        result = evaluate(model, val_loader)
        result['train_loss'] = torch.stack(train_losses).mean().item()
        model.epoch_end(epoch, result)
        history.append(result) # displays train_loss, val_loss and val_acc at the end of each epoch
    return history
```

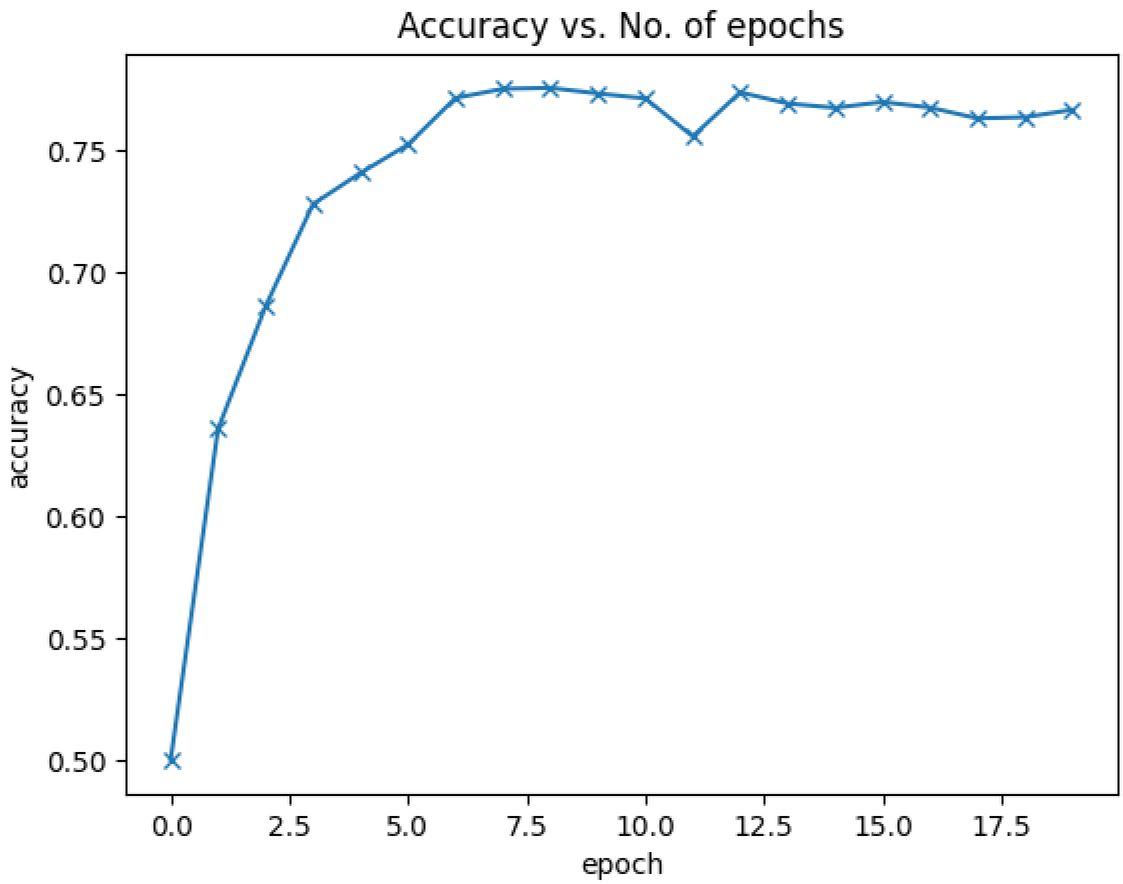
```python
[125] test_loader = DeviceDataLoader(DataLoader(test_dataset, batch_size*2), device)
      result = evaluate(model, test_loader)
      result
```

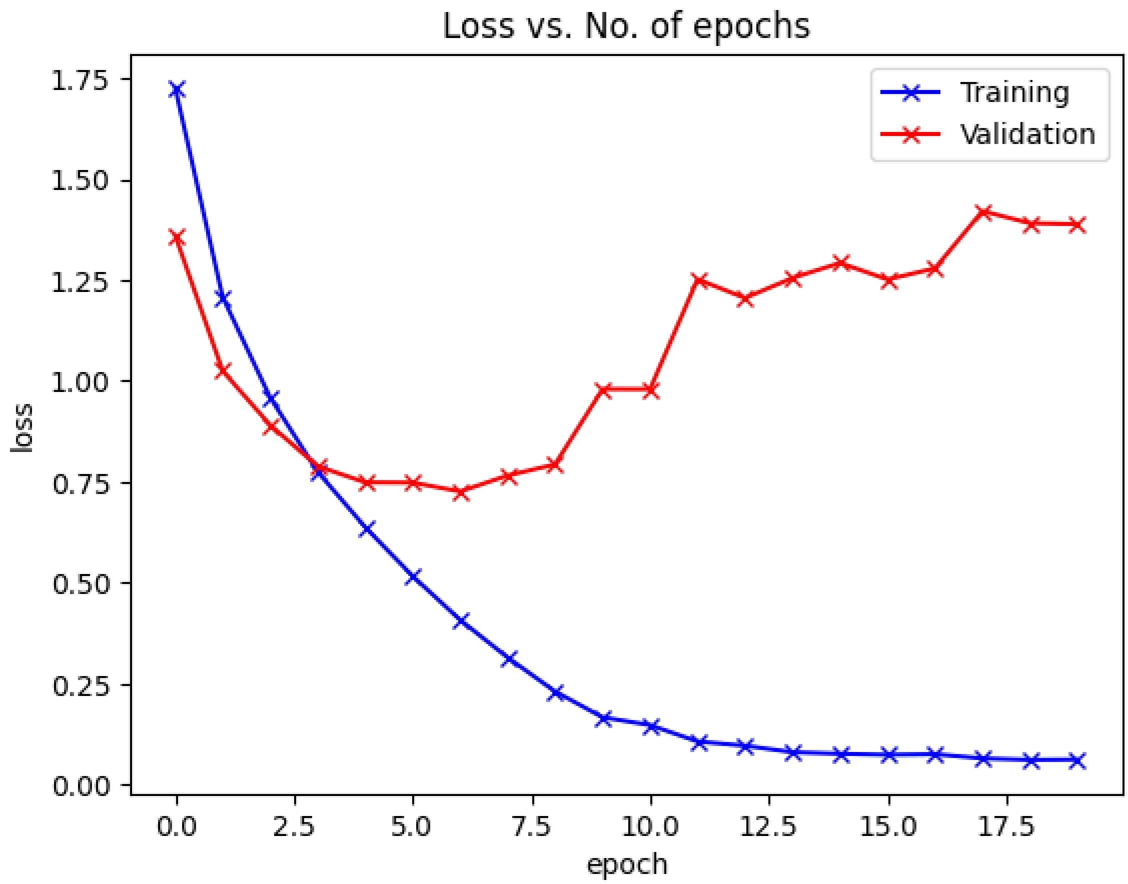
```
{'val_loss': 1.289367914199829, 'val_acc': 0.7705078125}
```

Just as a sanity check, let's verify that this model has the same loss and accuracy on the test set as before.

```python
evaluate(model2, test_loader)
```

```
{'val_loss': 1.289367914199829, 'val_acc': 0.7705078125}
```

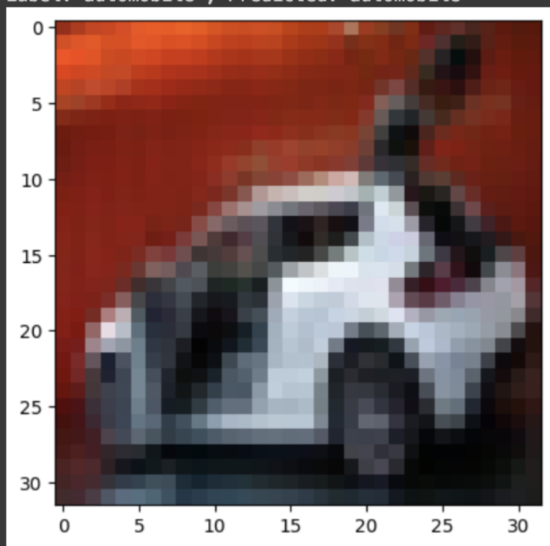

Accuracy vs. No. of epochs
accuracy
epoch
0.50
0.55
0.60
0.65
0.70
0.75
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5



Loss vs. No. of epochs
Training
Validation
loss
epoch
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

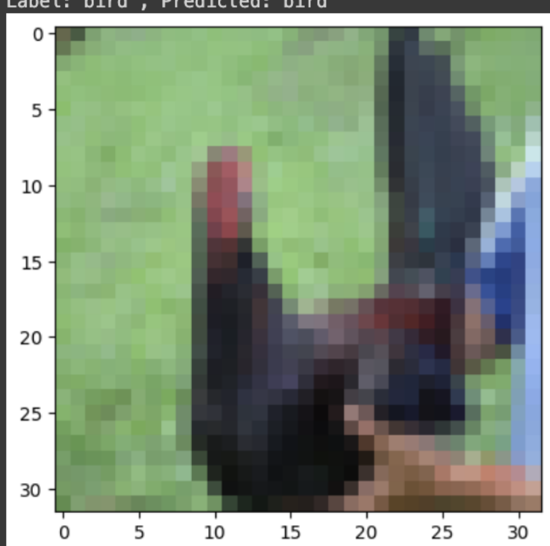
[122]
```
# -----------
# Graded Cell
# -----------
img, label = test_dataset[1000]
plt.imshow(img.permute(1, 2, 0))
print('Label:', training_data.classes[label], ', Predicted:', predict_image(img, model))
```

```
Label: automobile , Predicted: automobile
```




```
# -----------
# Graded Cell
# -----------
img, label = test_dataset[2000]
plt.imshow(img.permute(1, 2, 0))
print('Label:', training_data.classes[label], ', Predicted:', predict_image(img, model))
```

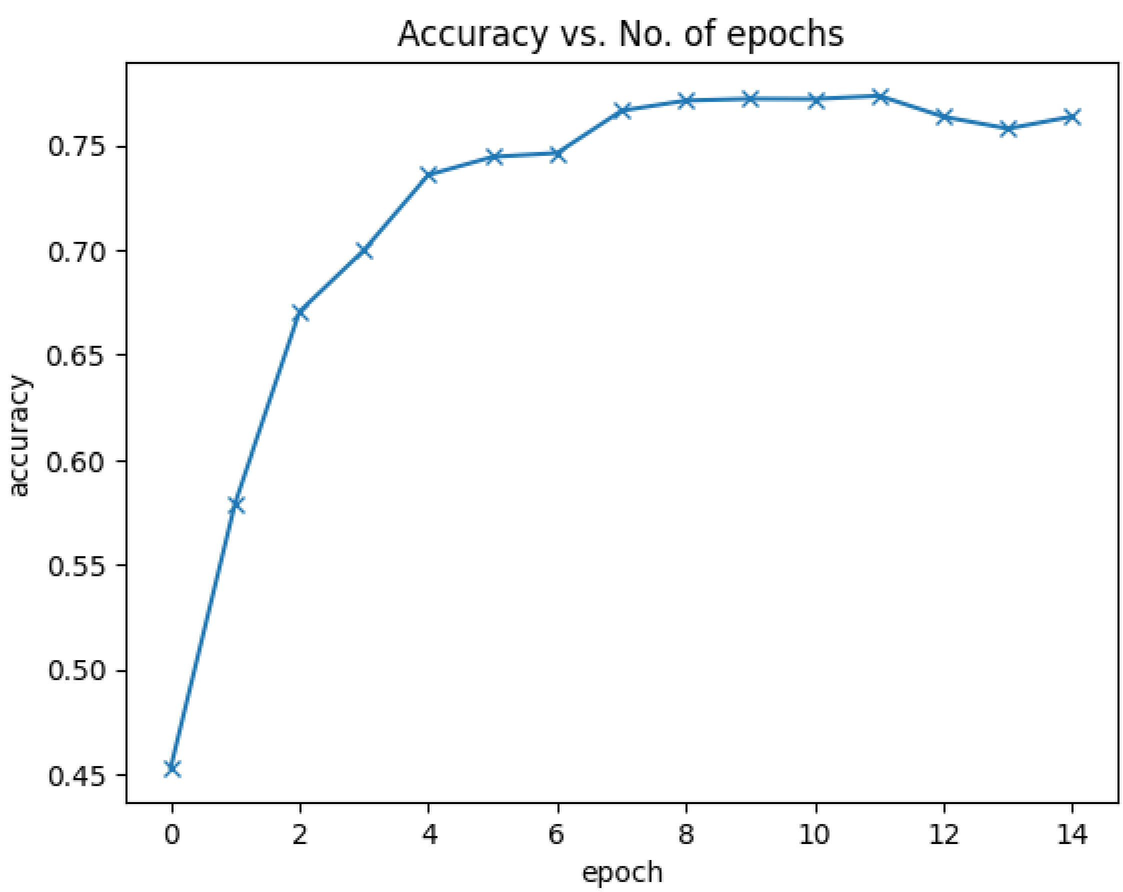
```
Label: bird , Predicted: bird
```

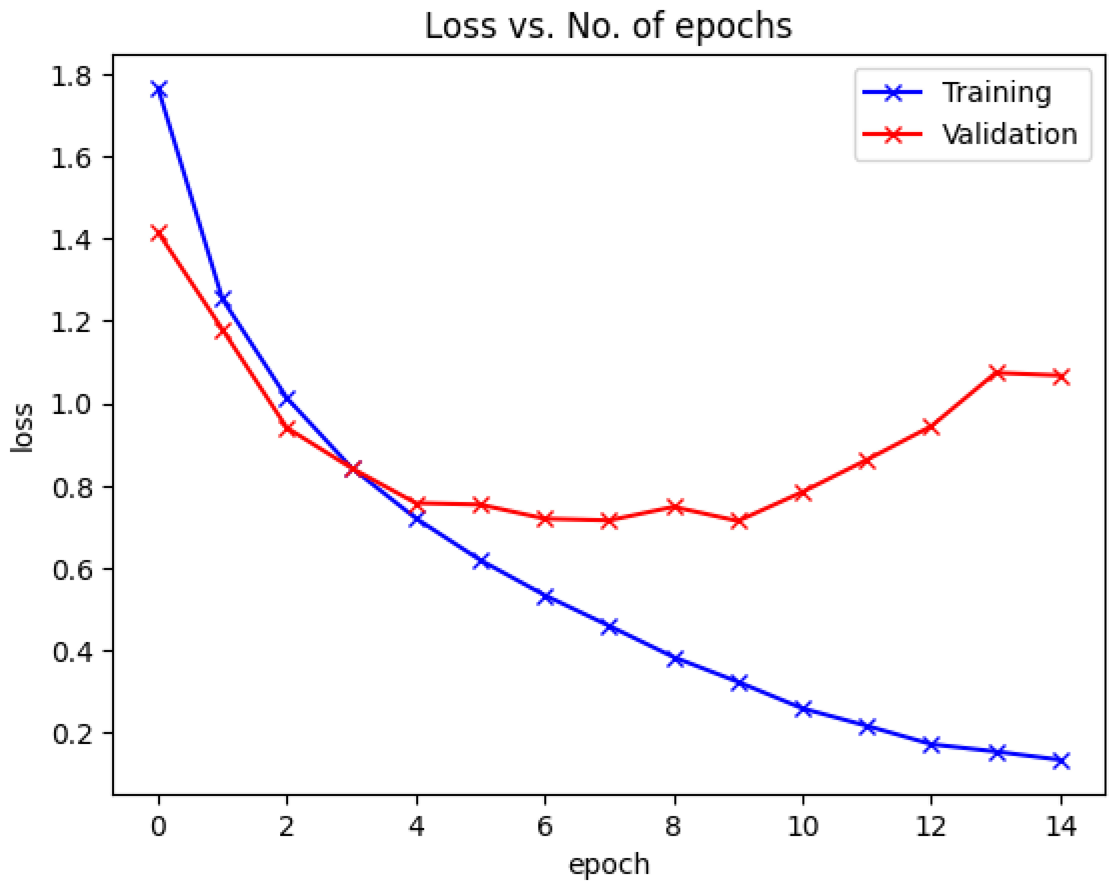


```
[132] history_with_early_stopping = fit_with_early_stopping(num_epochs, lr, model3, train_dl, val_dl, opt_fu
```

```
Epoch [0], train_loss: 1.7655, val_loss: 1.4156, val_acc: 0.4529
Epoch [1], train_loss: 1.2532, val_loss: 1.1793, val_acc: 0.5792
Epoch [2], train_loss: 1.0130, val_loss: 0.9405, val_acc: 0.6706
Epoch [3], train_loss: 0.8440, val_loss: 0.8431, val_acc: 0.6999
Epoch [4], train_loss: 0.7211, val_loss: 0.7579, val_acc: 0.7359
Epoch [5], train_loss: 0.6195, val_loss: 0.7542, val_acc: 0.7445
Epoch [6], train_loss: 0.5343, val_loss: 0.7200, val_acc: 0.7463
Epoch [7], train_loss: 0.4599, val_loss: 0.7158, val_acc: 0.7665
Epoch [8], train_loss: 0.3841, val_loss: 0.7487, val_acc: 0.7714
Epoch [9], train_loss: 0.3239, val_loss: 0.7143, val_acc: 0.7722
Epoch [10], train_loss: 0.2595, val_loss: 0.7848, val_acc: 0.7721
Epoch [11], train_loss: 0.2172, val_loss: 0.8628, val_acc: 0.7737
Epoch [12], train_loss: 0.1725, val_loss: 0.9444, val_acc: 0.7636
Epoch [13], train_loss: 0.1548, val_loss: 1.0740, val_acc: 0.7580
Epoch [14], train_loss: 0.1346, val_loss: 1.0676, val_acc: 0.7638
Early stopping at epoch 14
```

We can also plot the validation set accuracies to study how the model improved before it was halted due to the early stopping meta-algorithm. Moreover, we can plot the training and validation losses to study how the effect of overfitting was avoided when we introduced early stopping.






```
[135] # -----------
      # Graded Cell
      # -----------
      evaluate(model3, test_loader)

      {'val_loss': 1.0728938579559326, 'val_acc': 0.764453113079071}
```