Machine_Learning_Homework3

January 18, 2019

DATA IMPORT

The goal of this homework is to understand and become familiar with neural networks.

More specifically, we are going to start from a pre-defined code that implements a basic NN and CNN, the data loading, the training phase and the evaluation phase and see how the network responds to the change of different parameters.

The dataset used for this purpose is CIFAR-100, which contains 100 image classes, with 600 images per class. Each image is 32x32x3 (3 color), and the 600 images are divided into 500 training, and 100 test for each class.

For each training, the graphs of accuracy and loss behavior are provided.

```
trainset = torchvision.datasets.CIFAR100(root='./data', train=True,
                                        download=True, transform=transform_train)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=256,
                                          shuffle=True, num_workers=4,drop_last=True)
```

If not specified, parameters used for almost all of trials are:

- Epochs = 20
- Resolution 32x32
- Adam solver Learning Rate = 0.0001
- Batch Size = 256

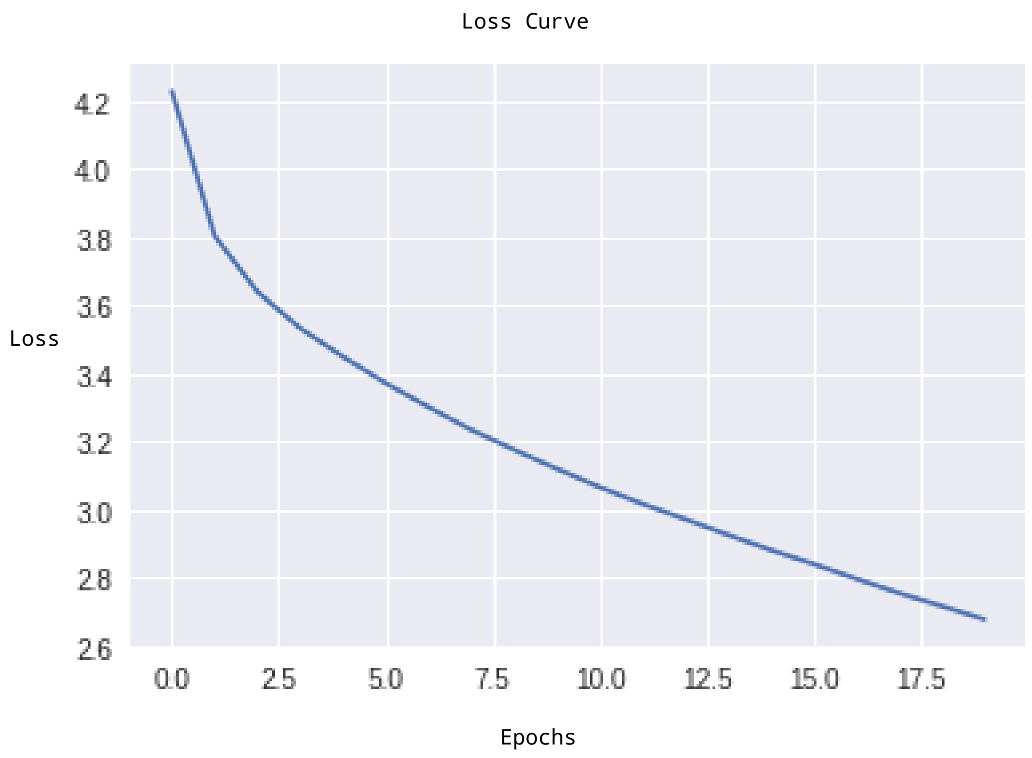
TRAINING OF TRADITIONAL 2 HIDDEN LAYER PLUS LAST FC LAYER NETWORK

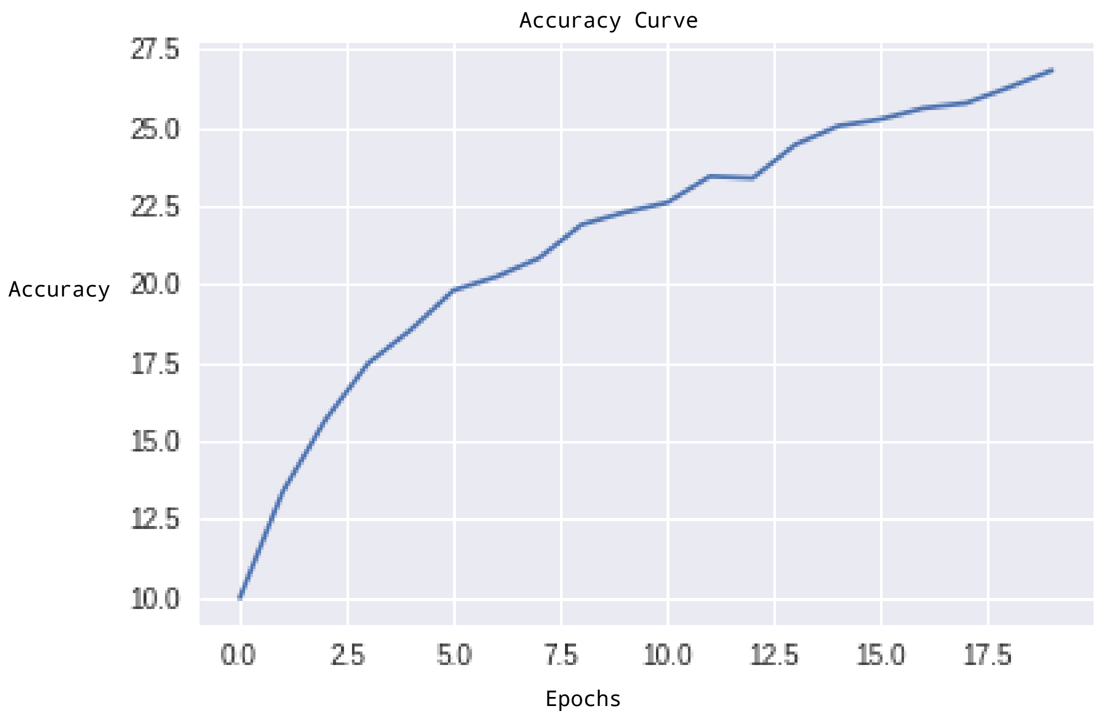
The principal difference between NN and CNN is about features and architecture.

The first usually use hand-crafted features, while the second crafts the features by through its deep architecture.

```
class old_nn(nn.Module):
    def __init__(self):
        super(old_nn, self).__init__()
        self.fc1 = nn.Linear(32*32*3, 4096)
        self.fc2 = nn.Linear(4096, 4096)
        self.fc3 = nn.Linear(4096, n_classes) #last FC for classification

    def forward(self, x):
        x = x.view(x.shape[0], -1)
        x = F.sigmoid(self.fc1(x))
        x = F.sigmoid(self.fc2(x))
        x = self.fc3(x)
        return x
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz to ./data/cifar-100-python.tar.gz
Files already downloaded and verified
[1,   195] loss: 4.228
Accuracy of the network on the test set: 9 %
[2,   195] loss: 3.803
Accuracy of the network on the test set: 13 %
[3,   195] loss: 3.639
Accuracy of the network on the test set: 15 %
[4,   195] loss: 3.533
Accuracy of the network on the test set: 17 %
[5,   195] loss: 3.449
Accuracy of the network on the test set: 18 %
[6,   195] loss: 3.371
Accuracy of the network on the test set: 19 %
[7,   195] loss: 3.301
Accuracy of the network on the test set: 20 %
[8,   195] loss: 3.234
Accuracy of the network on the test set: 20 %
[9,   195] loss: 3.176
Accuracy of the network on the test set: 21 %
[10,   195] loss: 3.120
Accuracy of the network on the test set: 22 %
[11,   195] loss: 3.066
Accuracy of the network on the test set: 22 %
[12,   195] loss: 3.016
Accuracy of the network on the test set: 23 %
[13,   195] loss: 2.971
Accuracy of the network on the test set: 23 %
[14,   195] loss: 2.925
Accuracy of the network on the test set: 24 %
[15,   195] loss: 2.881
Accuracy of the network on the test set: 25 %
[16,   195] loss: 2.840
Accuracy of the network on the test set: 25 %
[17,   195] loss: 2.796
Accuracy of the network on the test set: 25 %
[18,   195] loss: 2.754
Accuracy of the network on the test set: 25 %
[19,   195] loss: 2.716
Accuracy of the network on the test set: 26 %
[20,   195] loss: 2.677
Accuracy of the network on the test set: 26 %
Finished Training.
Max accuracy: 26.81 %
Execution time: 00:03:52.18
```





The loss and the accuracy have opposite trend. As a matter of fact, we can see from the plots above that the curve of the loss decreases while the accuracy one grows.

This happens because of the overfitting coadaptation phenomenon: when a large feedforward neural network is trained on a small training set, it typically performs poorly on held-out test data.

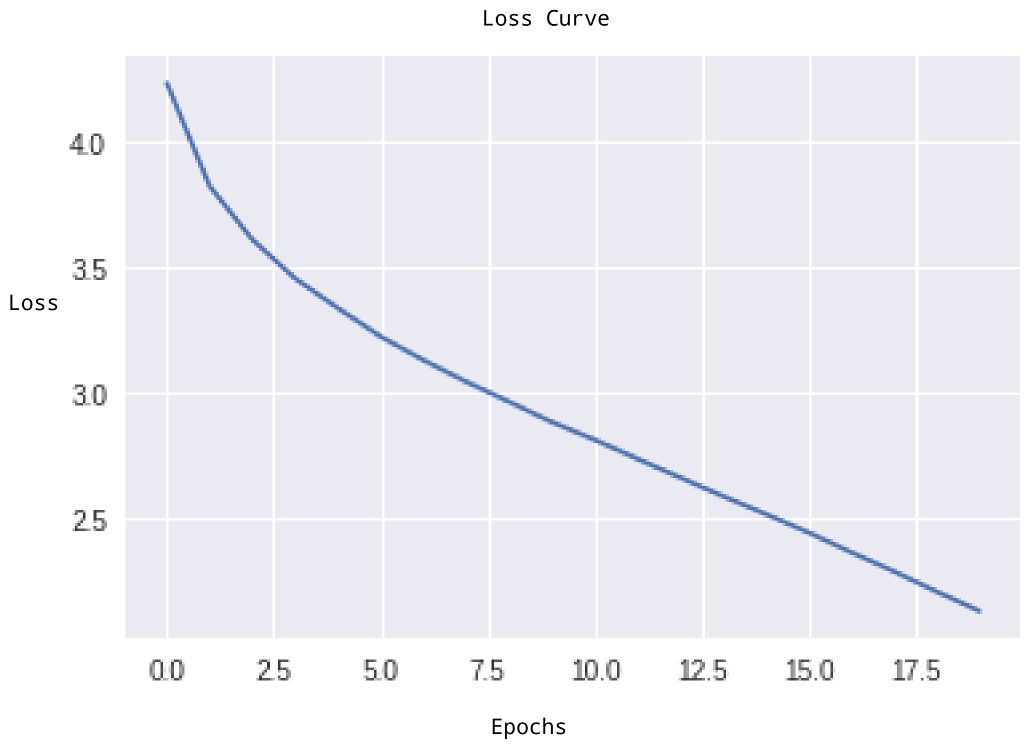
We will see that this phenomenon will occur in every network proposed.

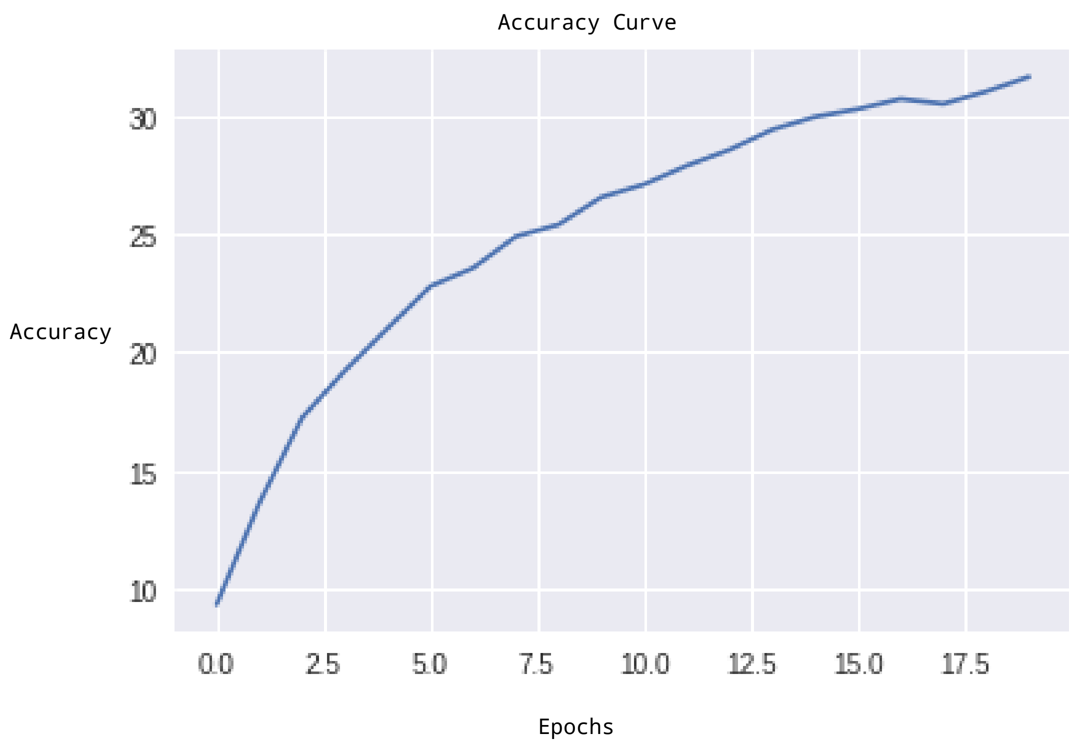
Convolutional Neural Network 32/32/32/64

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=5, stride=2, padding=0)
        self.conv2 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=0)
        self.conv3 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=0)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.conv_final = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=0)
        self.fc1 = nn.Linear(64 * 4 * 4, 4096)
        self.fc2 = nn.Linear(4096, n_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = F.relu(self.pool(self.conv_final(x)))
        x = x.view(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```
Files already downloaded and verified
Files already downloaded and verified
[1,   195] loss: 4.232
Accuracy of the network on the test set: 9 %
[2,   195] loss: 3.822
Accuracy of the network on the test set: 13 %
[3,   195] loss: 3.608
Accuracy of the network on the test set: 17 %
[4,   195] loss: 3.453
Accuracy of the network on the test set: 19 %
[5,   195] loss: 3.335
Accuracy of the network on the test set: 21 %
[6,   195] loss: 3.221
Accuracy of the network on the test set: 22 %
[7,   195] loss: 3.127
Accuracy of the network on the test set: 23 %
[8,   195] loss: 3.040
Accuracy of the network on the test set: 24 %
[9,   195] loss: 2.961
Accuracy of the network on the test set: 25 %
[10,   195] loss: 2.879
Accuracy of the network on the test set: 26 %
[11,   195] loss: 2.808
Accuracy of the network on the test set: 27 %
[12,   195] loss: 2.733
Accuracy of the network on the test set: 27 %
[13,   195] loss: 2.658
Accuracy of the network on the test set: 28 %
[14,   195] loss: 2.584
Accuracy of the network on the test set: 29 %
[15,   195] loss: 2.512
Accuracy of the network on the test set: 29 %
```

```
[16,   195] loss: 2.439
Accuracy of the network on the test set: 30 %
[17,   195] loss: 2.359
Accuracy of the network on the test set: 30 %
[18,   195] loss: 2.283
Accuracy of the network on the test set: 30 %
[19,   195] loss: 2.202
Accuracy of the network on the test set: 31 %
[20,   195] loss: 2.124
Accuracy of the network on the test set: 31 %
Finished Training.
Max accuracy: 31.63 %
Execution time: 00:03:35.96
```



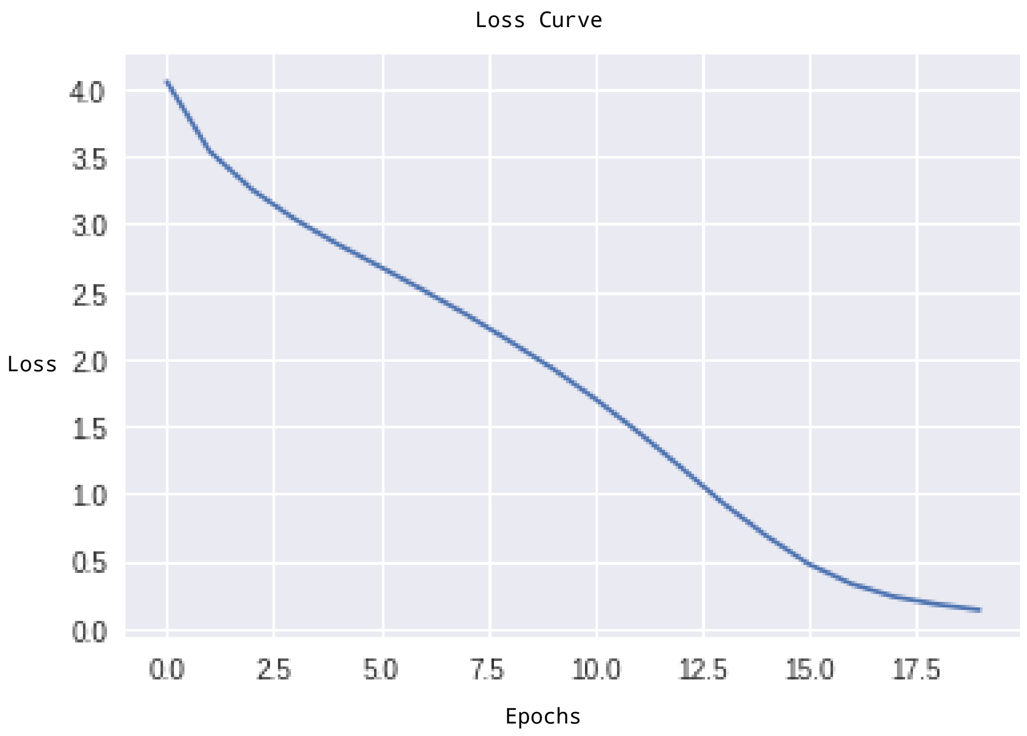


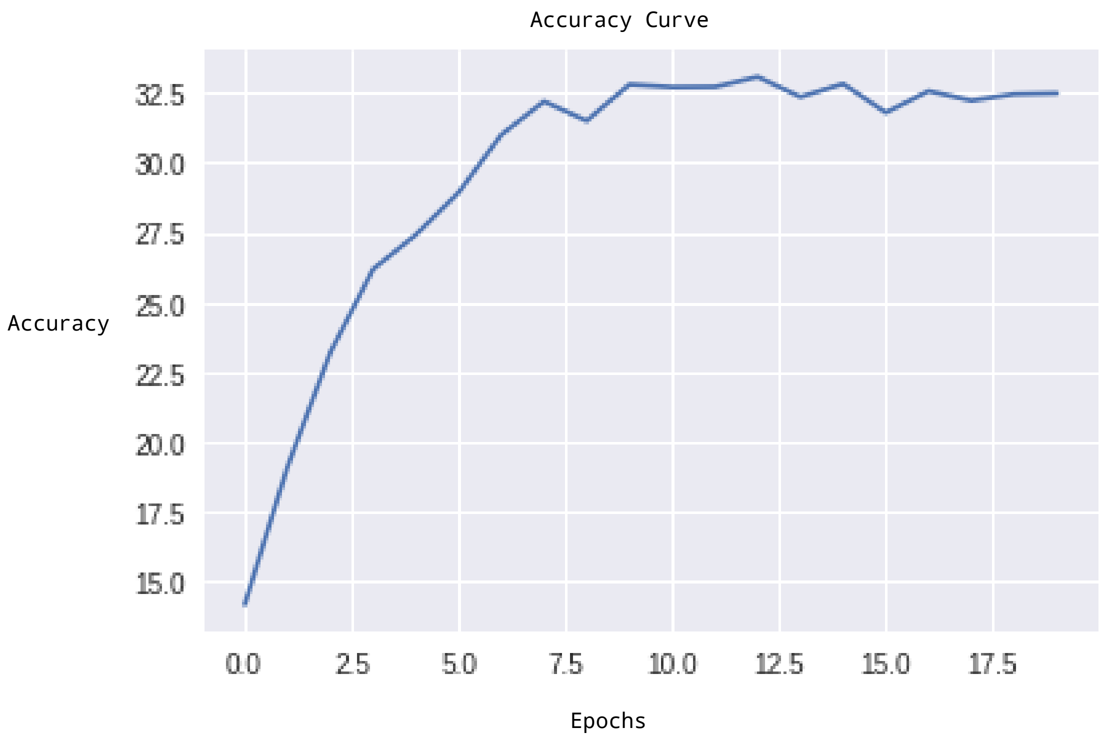
Convolutional Neural Network 128/128/128/256

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 128, kernel_size=5, stride=2, padding=0)
        self.conv2 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=0)
        self.conv3 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=0)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.conv_final = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=0)
        self.fc1 = nn.Linear(256 * 4 * 4, 4096)
        self.fc2 = nn.Linear(4096, n_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = F.relu(self.pool(self.conv_final(x)))
        x = x.view(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```
Files already downloaded and verified
Files already downloaded and verified
[1,   195] loss: 4.053
Accuracy of the network on the test set: 14 %
[2,   195] loss: 3.539
Accuracy of the network on the test set: 19 %
[3,   195] loss: 3.253
```

```
Accuracy of the network on the test set: 23 %
[4,   195] loss: 3.034
Accuracy of the network on the test set: 26 %
[5,   195] loss: 2.849
Accuracy of the network on the test set: 27 %
[6,   195] loss: 2.680
Accuracy of the network on the test set: 28 %
[7,   195] loss: 2.509
Accuracy of the network on the test set: 30 %
[8,   195] loss: 2.330
Accuracy of the network on the test set: 32 %
[9,   195] loss: 2.136
Accuracy of the network on the test set: 31 %
[10,   195] loss: 1.933
Accuracy of the network on the test set: 32 %
[11,   195] loss: 1.708
Accuracy of the network on the test set: 32 %
[12,   195] loss: 1.463
Accuracy of the network on the test set: 32 %
[13,   195] loss: 1.203
Accuracy of the network on the test set: 33 %
[14,   195] loss: 0.933
Accuracy of the network on the test set: 32 %
[15,   195] loss: 0.692
Accuracy of the network on the test set: 32 %
[16,   195] loss: 0.481
Accuracy of the network on the test set: 31 %
[17,   195] loss: 0.333
Accuracy of the network on the test set: 32 %
[18,   195] loss: 0.237
Accuracy of the network on the test set: 32 %
[19,   195] loss: 0.180
Accuracy of the network on the test set: 32 %
[20,   195] loss: 0.138
Accuracy of the network on the test set: 32 %
Finished Training.
Max accuracy: 33.07 %
Execution time: 00:06:19.03
```



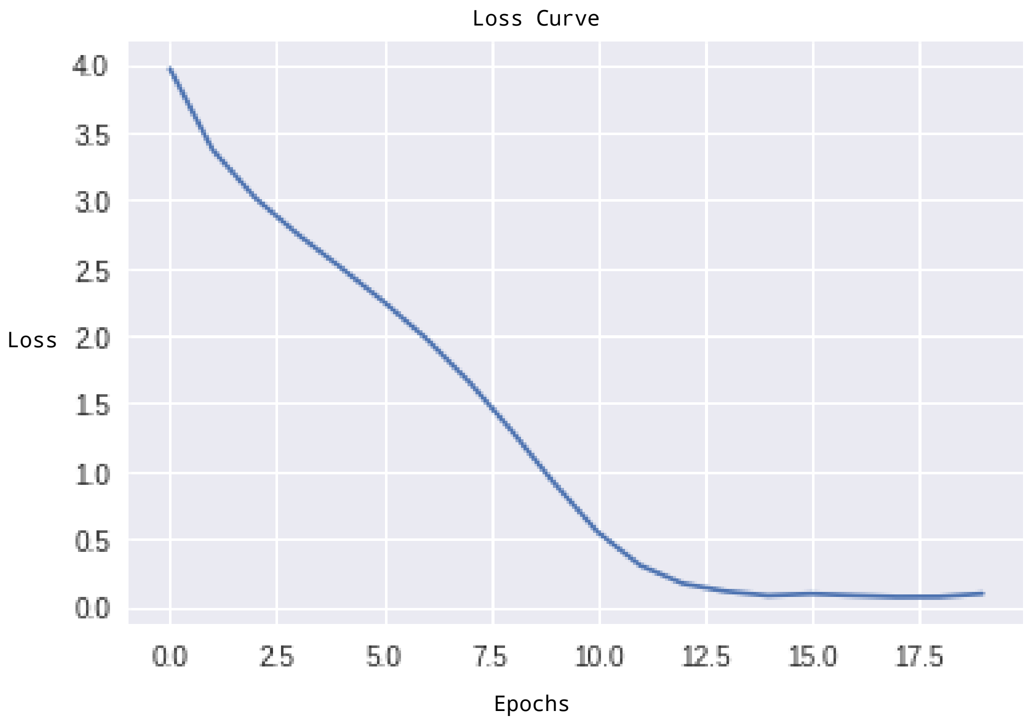


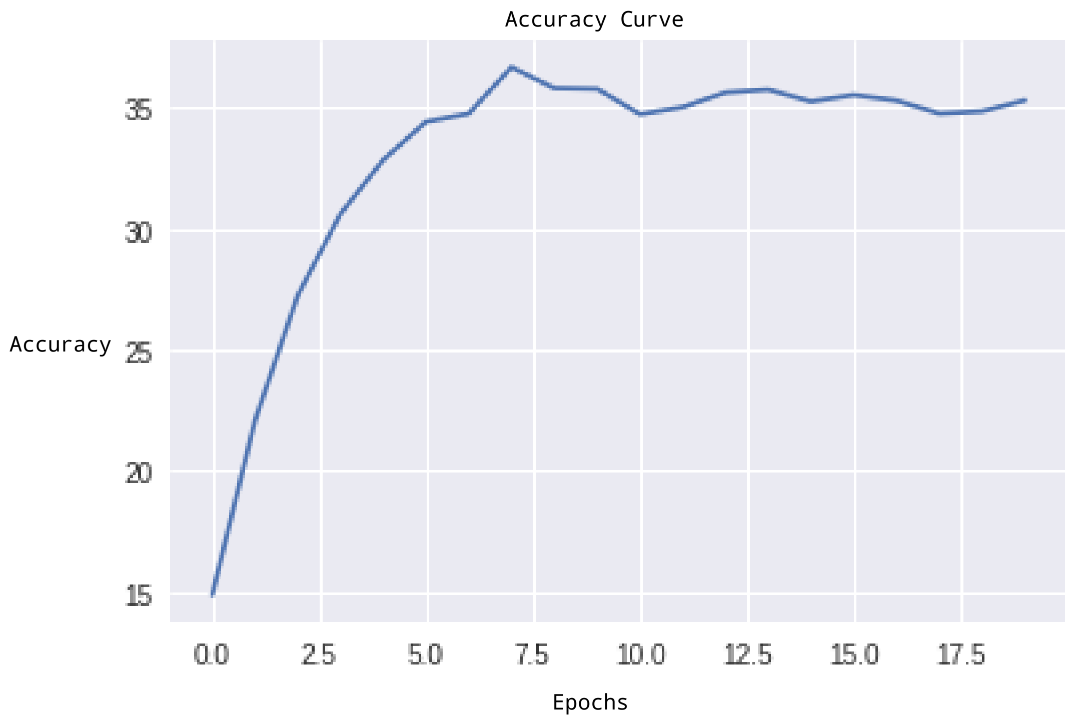
Convolutional Neural Network 256/256/256/512

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 256, kernel_size=5, stride=2, padding=0)
        self.conv2 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=0)
        self.conv3 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=0)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.conv_final = nn.Conv2d(256, 512, kernel_size=3, stride=1, padding=0)
        self.fc1 = nn.Linear(512 * 4 * 4, 4096)
        self.fc2 = nn.Linear(4096, n_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = F.relu(self.pool(self.conv_final(x)))
        x = x.view(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```
Files already downloaded and verified
Files already downloaded and verified
[1,   195] loss: 3.966
Accuracy of the network on the test set: 14 %
[2,   195] loss: 3.370
Accuracy of the network on the test set: 22 %
[3,   195] loss: 3.015
Accuracy of the network on the test set: 27 %
[4,   195] loss: 2.745
Accuracy of the network on the test set: 30 %
[5,   195] loss: 2.502
Accuracy of the network on the test set: 32 %
[6,   195] loss: 2.250
Accuracy of the network on the test set: 34 %
[7,   195] loss: 1.978
Accuracy of the network on the test set: 34 %
[8,   195] loss: 1.659
Accuracy of the network on the test set: 36 %
[9,   195] loss: 1.296
Accuracy of the network on the test set: 35 %
[10,   195] loss: 0.909
Accuracy of the network on the test set: 35 %
[11,   195] loss: 0.553
Accuracy of the network on the test set: 34 %
[12,   195] loss: 0.306
Accuracy of the network on the test set: 35 %
[13,   195] loss: 0.171
Accuracy of the network on the test set: 35 %
[14,   195] loss: 0.114
Accuracy of the network on the test set: 35 %
[15,   195] loss: 0.083
Accuracy of the network on the test set: 35 %
[16,   195] loss: 0.096
Accuracy of the network on the test set: 35 %
[17,   195] loss: 0.083
Accuracy of the network on the test set: 35 %
```

```
[18,   195] loss: 0.073
Accuracy of the network on the test set: 34 %
[19,   195] loss: 0.074
Accuracy of the network on the test set: 34 %
[20,   195] loss: 0.097
Accuracy of the network on the test set: 35 %
Finished Training.
Max accuracy: 36.67 %
Execution time: 00:15:38.74
```



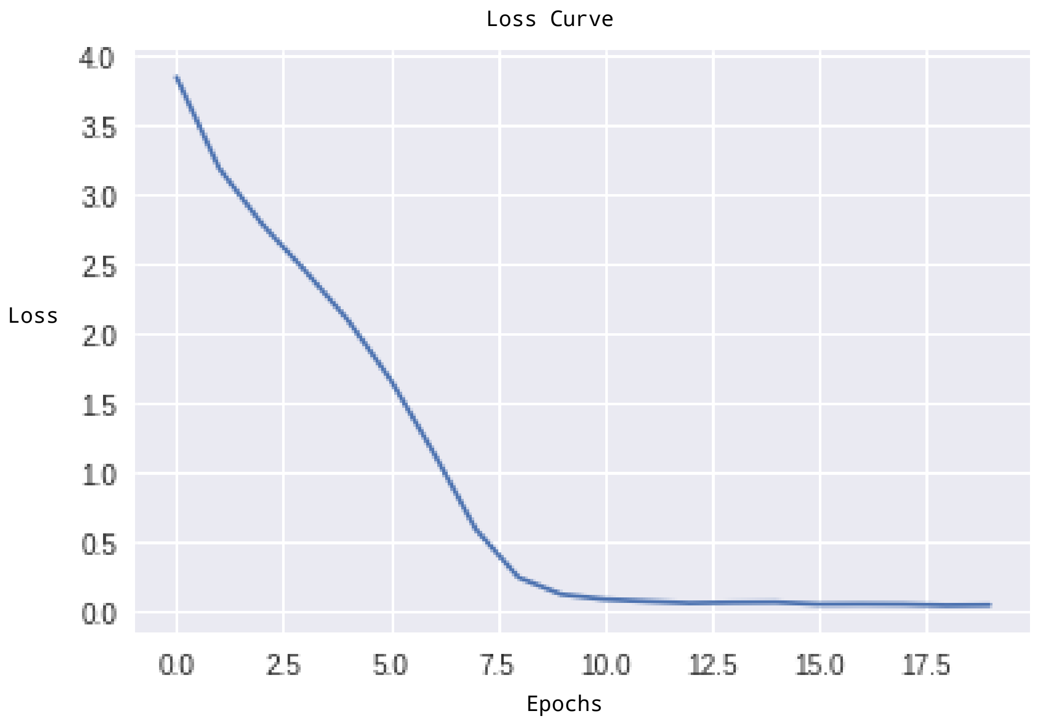


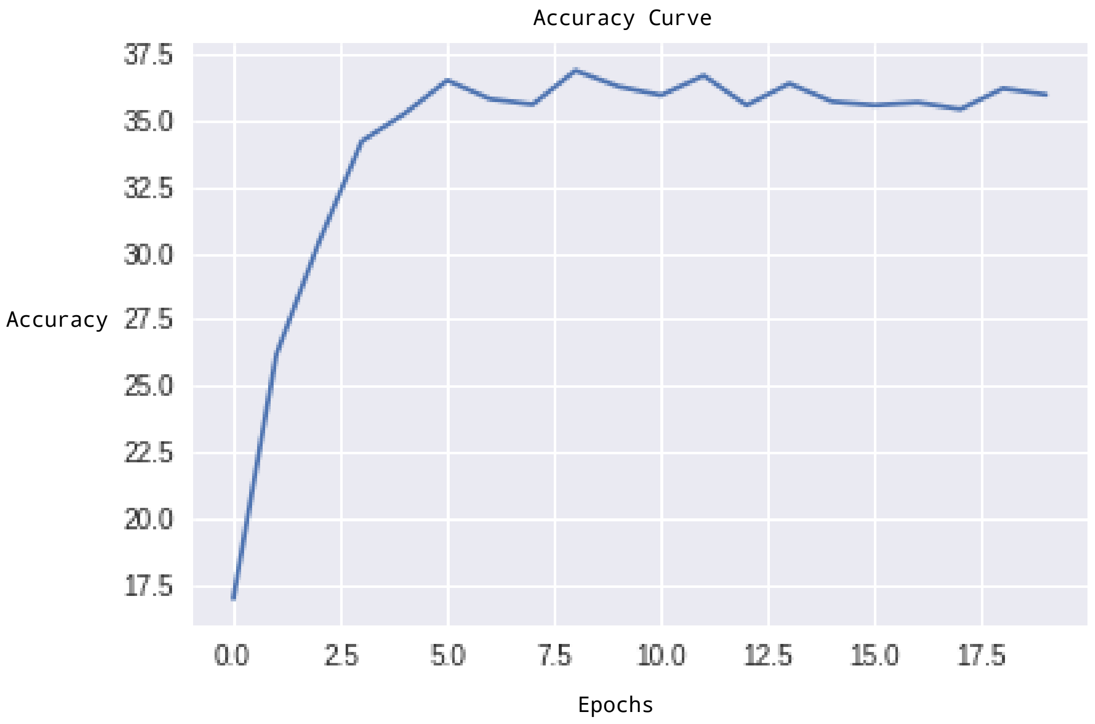
Convolutional Neural Network 512/512/512/1024

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 512, kernel_size=5, stride=2, padding=0)
        self.conv2 = nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=0)
        self.conv3 = nn.Conv2d(512, 512, kernel_size=3, stride=1, padding=0)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.conv_final = nn.Conv2d(512, 1024, kernel_size=3, stride=1, padding=0)
        self.fc1 = nn.Linear(1024 * 4 * 4, 4096)
        self.fc2 = nn.Linear(4096, n_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = F.relu(self.pool(self.conv_final(x)))
        x = x.view(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```
Files already downloaded and verified
Files already downloaded and verified
[1,   195] loss: 3.850
Accuracy of the network on the test set: 16 %
[2,   195] loss: 3.194
Accuracy of the network on the test set: 26 %
[3,   195] loss: 2.793
```

```
Accuracy of the network on the test set: 30 %
[4,   195] loss: 2.461
Accuracy of the network on the test set: 34 %
[5,   195] loss: 2.106
Accuracy of the network on the test set: 35 %
[6,   195] loss: 1.675
Accuracy of the network on the test set: 36 %
[7,   195] loss: 1.153
Accuracy of the network on the test set: 35 %
[8,   195] loss: 0.596
Accuracy of the network on the test set: 35 %
[9,   195] loss: 0.249
Accuracy of the network on the test set: 36 %
[10,   195] loss: 0.127
Accuracy of the network on the test set: 36 %
[11,   195] loss: 0.092
Accuracy of the network on the test set: 35 %
[12,   195] loss: 0.076
Accuracy of the network on the test set: 36 %
[13,   195] loss: 0.064
Accuracy of the network on the test set: 35 %
[14,   195] loss: 0.069
Accuracy of the network on the test set: 36 %
[15,   195] loss: 0.071
Accuracy of the network on the test set: 35 %
[16,   195] loss: 0.056
Accuracy of the network on the test set: 35 %
[17,   195] loss: 0.057
Accuracy of the network on the test set: 35 %
[18,   195] loss: 0.056
Accuracy of the network on the test set: 35 %
[19,   195] loss: 0.048
Accuracy of the network on the test set: 36 %
[20,   195] loss: 0.051
Accuracy of the network on the test set: 35 %
Finished Training.
Max accuracy: 36.89 %
Execution time: 00:47:59.03
```





Here we see how different filter dimension affects the network's accuracy and loss.

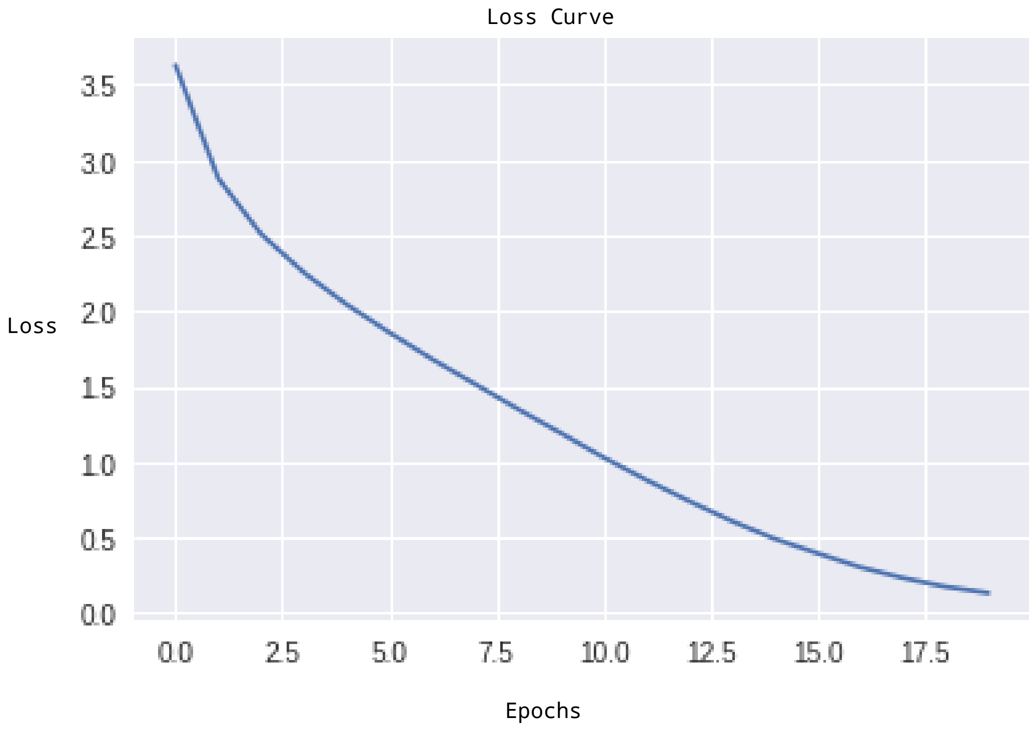
while we can see that the values of accuracy and loss do not change much in absolute terms, we notice how the speed with which they reach these values changes with increasing filter size.

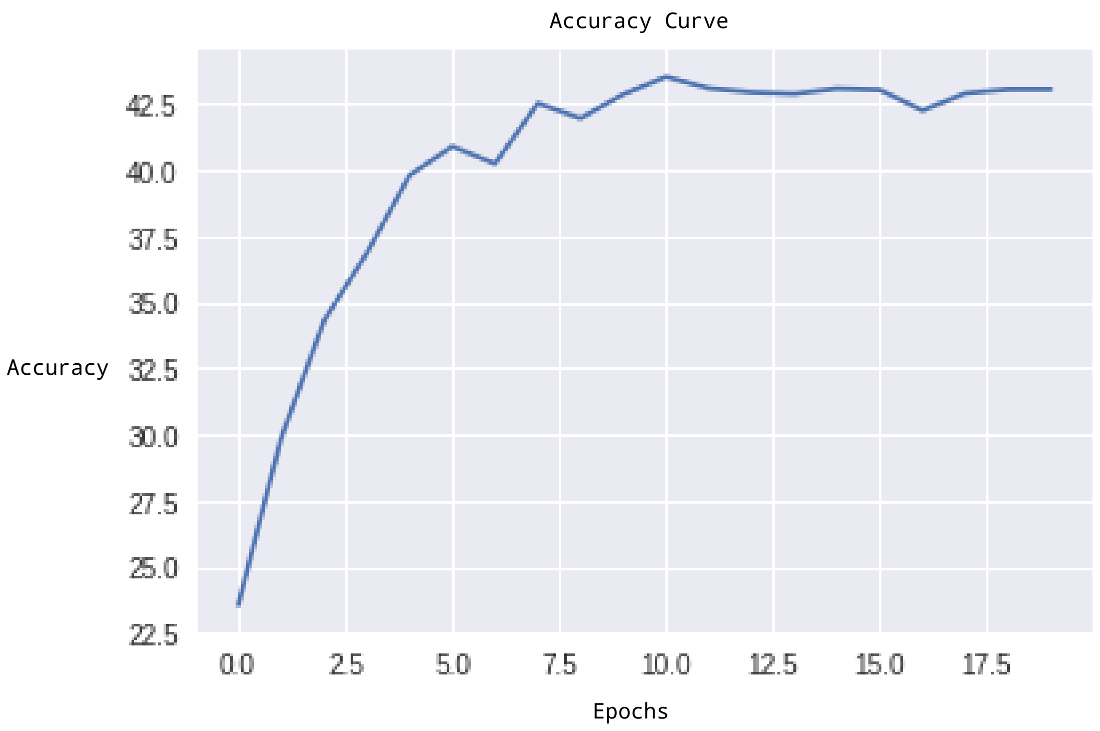
CNN with Batch Normalization on every convolutional layer

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 128, kernel_size=5, stride=2, padding=0)
        self.conv1_bn = nn.BatchNorm2d(128)
        self.conv2 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=0)
        self.conv2_bn = nn.BatchNorm2d(128)
        self.conv3 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=0)
        self.conv3_bn = nn.BatchNorm2d(128)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.conv_final = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=0)
        self.conv_final_bn = nn.BatchNorm2d(256)
        self.fc1 = nn.Linear(256 * 4 * 4, 4096)
        self.fc2 = nn.Linear(4096, n_classes)

    def forward(self, x):
        x = F.relu(self.conv1_bn(self.conv1(x)))
        x = F.relu(self.conv2_bn(self.conv2(x)))
        x = F.relu(self.conv3_bn(self.conv3(x)))
        x = F.relu(self.pool(self.conv_final_bn(self.conv_final(x))))
        x = x.view(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```
Files already downloaded and verified
Files already downloaded and verified
[1,   195] loss: 3.627
Accuracy of the network on the test set: 23 %
[2,   195] loss: 2.879
Accuracy of the network on the test set: 29 %
[3,   195] loss: 2.511
Accuracy of the network on the test set: 34 %
[4,   195] loss: 2.256
Accuracy of the network on the test set: 36 %
[5,   195] loss: 2.045
Accuracy of the network on the test set: 39 %
[6,   195] loss: 1.859
Accuracy of the network on the test set: 40 %
[7,   195] loss: 1.682
Accuracy of the network on the test set: 40 %
[8,   195] loss: 1.518
Accuracy of the network on the test set: 42 %
[9,   195] loss: 1.353
Accuracy of the network on the test set: 41 %
[10,   195] loss: 1.194
Accuracy of the network on the test set: 42 %
[11,   195] loss: 1.033
Accuracy of the network on the test set: 43 %
[12,   195] loss: 0.884
Accuracy of the network on the test set: 43 %
[13,   195] loss: 0.744
Accuracy of the network on the test set: 42 %
```

```
[14,   195] loss: 0.611
Accuracy of the network on the test set: 42 %
[15,   195] loss: 0.493
Accuracy of the network on the test set: 43 %
[16,   195] loss: 0.397
Accuracy of the network on the test set: 43 %
[17,   195] loss: 0.306
Accuracy of the network on the test set: 42 %
[18,   195] loss: 0.234
Accuracy of the network on the test set: 42 %
[19,   195] loss: 0.175
Accuracy of the network on the test set: 43 %
[20,   195] loss: 0.134
Accuracy of the network on the test set: 43 %
Finished Training
Max accuracy: 43.51 %
Execution time: 00:06:30.11
```





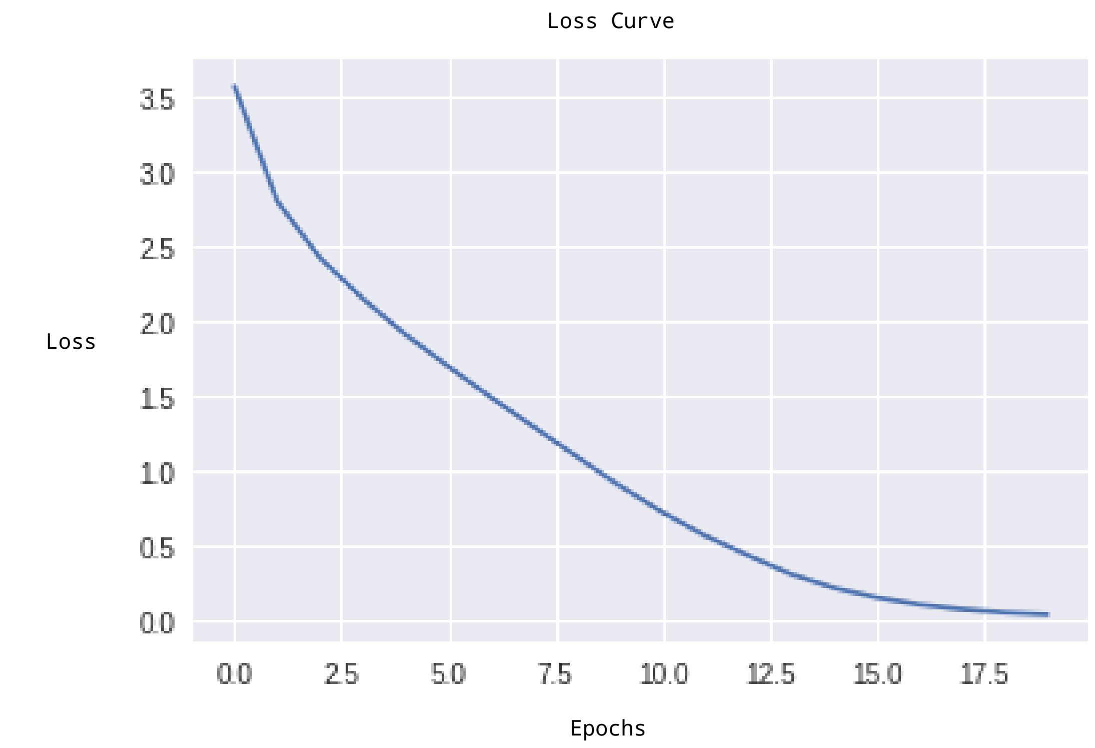
Batch normalization is a technique for improving the performance and stability of artificial neural networks normalizing the input layer by adjusting and scaling the activations.

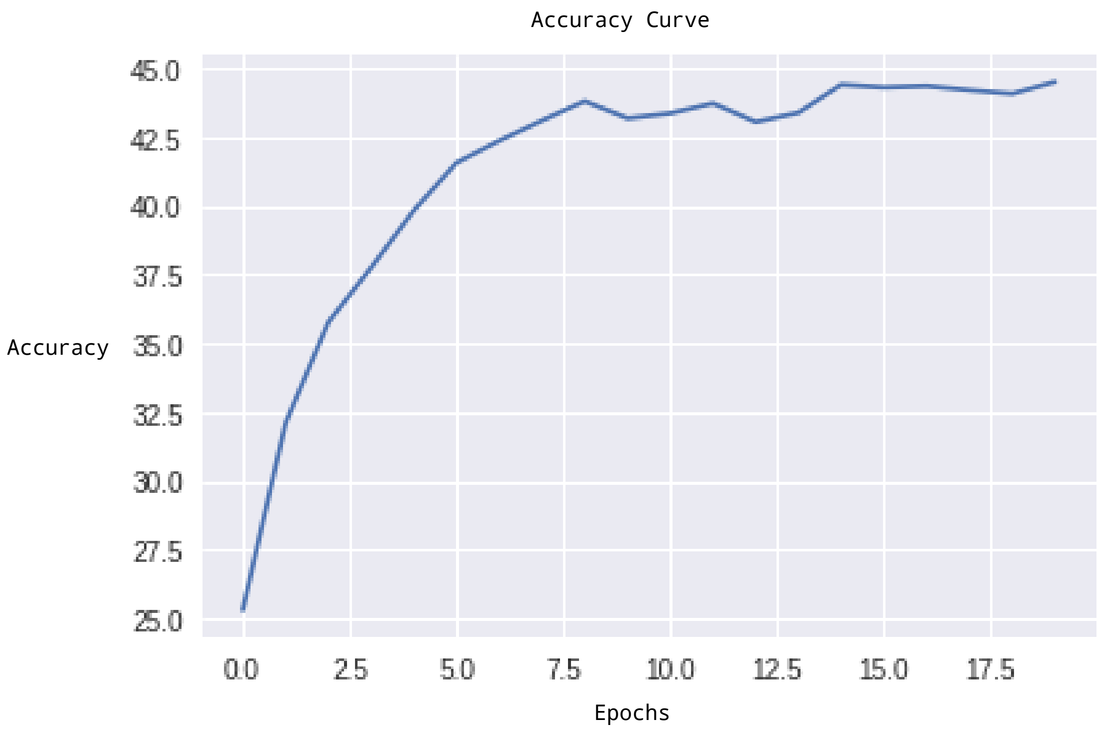
The graphs shows how both accuracy and loss are improved.

CNN with Batch Normalization on every convolutional layer and 8192 neurons

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 128, kernel_size=5, stride=2, padding=0)
        self.conv1_bn = nn.BatchNorm2d(128)
        self.conv2 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=0)
        self.conv2_bn = nn.BatchNorm2d(128)
        self.conv3 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=0)
        self.conv3_bn = nn.BatchNorm2d(128)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.conv_final = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=0)
        self.conv_final_bn = nn.BatchNorm2d(256)
        self.fc1 = nn.Linear(256 * 4 * 4, 8192)
        self.fc2 = nn.Linear(8192, n_classes)

    def forward(self, x):
        x = F.relu(self.conv1_bn(self.conv1(x)))
        x = F.relu(self.conv2_bn(self.conv2(x)))
        x = F.relu(self.conv3_bn(self.conv3(x)))
        x = F.relu(self.pool(self.conv_final_bn(self.conv_final(x))))
        x = x.view(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```
Files already downloaded and verified
Files already downloaded and verified
[1,   195] loss: 3.565
Accuracy of the network on the test set: 25 %
[2,   195] loss: 2.799
Accuracy of the network on the test set: 32 %
[3,   195] loss: 2.421
Accuracy of the network on the test set: 35 %
[4,   195] loss: 2.150
Accuracy of the network on the test set: 37 %
[5,   195] loss: 1.911
Accuracy of the network on the test set: 39 %
[6,   195] loss: 1.698
Accuracy of the network on the test set: 41 %
[7,   195] loss: 1.491
Accuracy of the network on the test set: 42 %
[8,   195] loss: 1.295
Accuracy of the network on the test set: 43 %
[9,   195] loss: 1.098
Accuracy of the network on the test set: 43 %
[10,   195] loss: 0.904
Accuracy of the network on the test set: 43 %
[11,   195] loss: 0.727
Accuracy of the network on the test set: 43 %
[12,   195] loss: 0.570
Accuracy of the network on the test set: 43 %
[13,   195] loss: 0.439
Accuracy of the network on the test set: 43 %
[14,   195] loss: 0.315
Accuracy of the network on the test set: 43 %
[15,   195] loss: 0.223
Accuracy of the network on the test set: 44 %
[16,   195] loss: 0.158
Accuracy of the network on the test set: 44 %
[17,   195] loss: 0.112
Accuracy of the network on the test set: 44 %
[18,   195] loss: 0.080
Accuracy of the network on the test set: 44 %
[19,   195] loss: 0.059
Accuracy of the network on the test set: 44 %
[20,   195] loss: 0.045
Accuracy of the network on the test set: 44 %
Finished Training
Max accuracy: 44.49 %
Execution time: 00:07:48.65
```





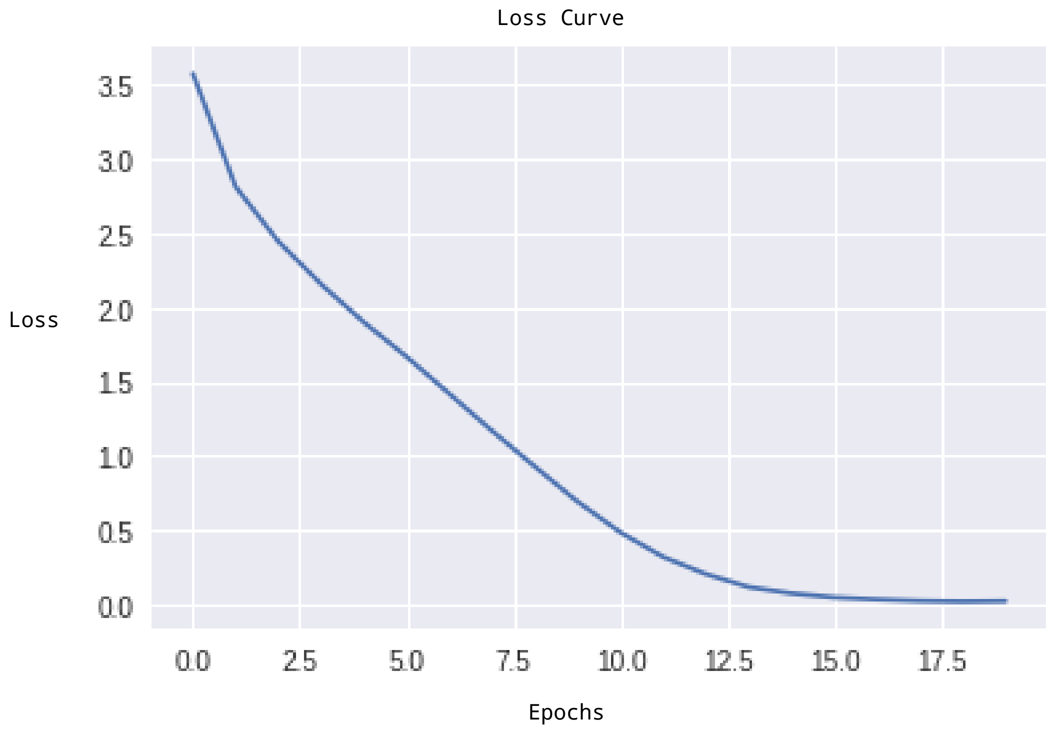
Using more neurons we see how the values increase further to the detriment of the execution time.

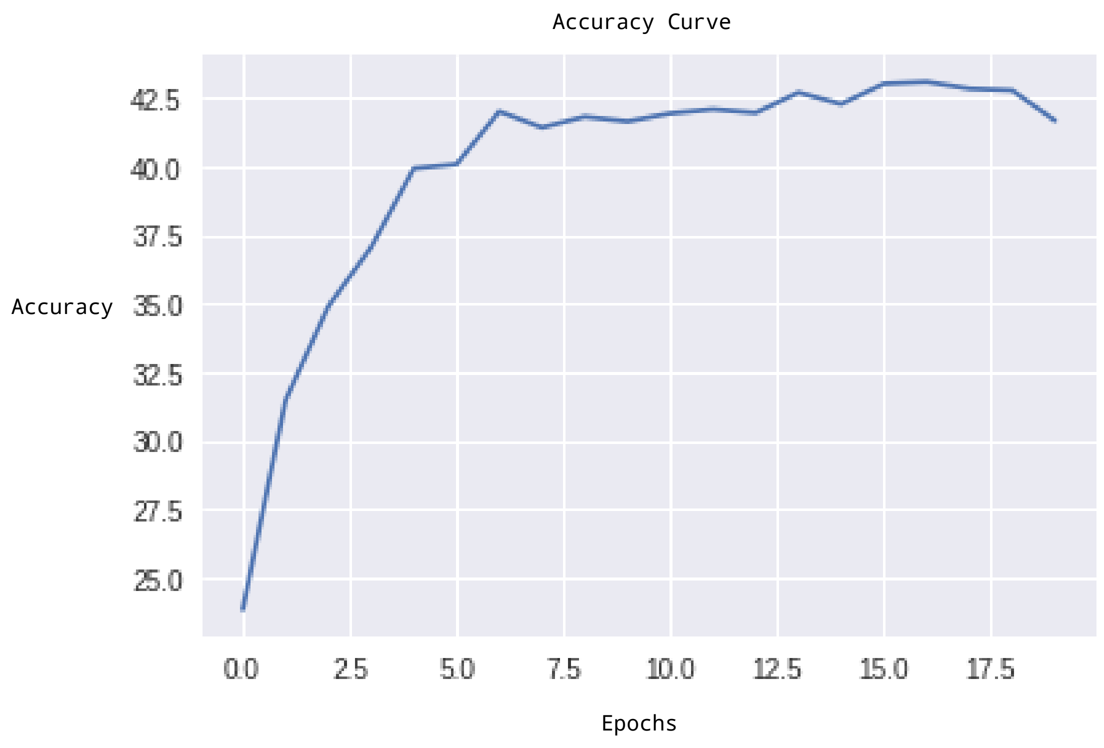
CNN with Batch Normalization and Dropout on FC1

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 128, kernel_size=5, stride=2, padding=0)
        self.conv1_bn = nn.BatchNorm2d(128)
        self.conv2 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=0)
        self.conv2_bn = nn.BatchNorm2d(128)
        self.conv3 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=0)
        self.conv3_bn = nn.BatchNorm2d(128)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.conv_final = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=0)
        self.conv_final_bn = nn.BatchNorm2d(256)
        self.fc1 = nn.Linear(256 * 4 * 4, 4096)
        self.fc2 = nn.Linear(4096, n_classes)

    def forward(self, x):
        x = F.relu(self.conv1_bn(self.conv1(x)))
        x = F.relu(self.conv2_bn(self.conv2(x)))
        x = F.relu(self.conv3_bn(self.conv3(x)))
        x = F.relu(self.pool(self.conv_final_bn(self.conv_final(x))))
        x = x.view(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        x = F.dropout2d(self.fc1(x), p=0.5)
        x = self.fc2(x)
        return x
```

```
Files already downloaded and verified
Files already downloaded and verified
[1,   195] loss: 3.570
Accuracy of the network on the test set: 23 %
[2,   195] loss: 2.810
Accuracy of the network on the test set: 31 %
[3,   195] loss: 2.442
Accuracy of the network on the test set: 34 %
[4,   195] loss: 2.155
Accuracy of the network on the test set: 37 %
[5,   195] loss: 1.899
Accuracy of the network on the test set: 39 %
[6,   195] loss: 1.666
Accuracy of the network on the test set: 40 %
[7,   195] loss: 1.417
Accuracy of the network on the test set: 42 %
[8,   195] loss: 1.167
Accuracy of the network on the test set: 41 %
[9,   195] loss: 0.928
Accuracy of the network on the test set: 41 %
[10,   195] loss: 0.691
Accuracy of the network on the test set: 41 %
[11,   195] loss: 0.485
Accuracy of the network on the test set: 41 %
[12,   195] loss: 0.320
Accuracy of the network on the test set: 42 %
[13,   195] loss: 0.204
Accuracy of the network on the test set: 41 %
[14,   195] loss: 0.117
Accuracy of the network on the test set: 42 %
```

```
[15,   195] loss: 0.074
Accuracy of the network on the test set: 42 %
[16,   195] loss: 0.047
Accuracy of the network on the test set: 43 %
[17,   195] loss: 0.033
Accuracy of the network on the test set: 43 %
[18,   195] loss: 0.024
Accuracy of the network on the test set: 42 %
[19,   195] loss: 0.020
Accuracy of the network on the test set: 42 %
[20,   195] loss: 0.023
Accuracy of the network on the test set: 41 %
Finished Training
Max accuracy: 43.09 %
Execution time: 00:07:27.50
```



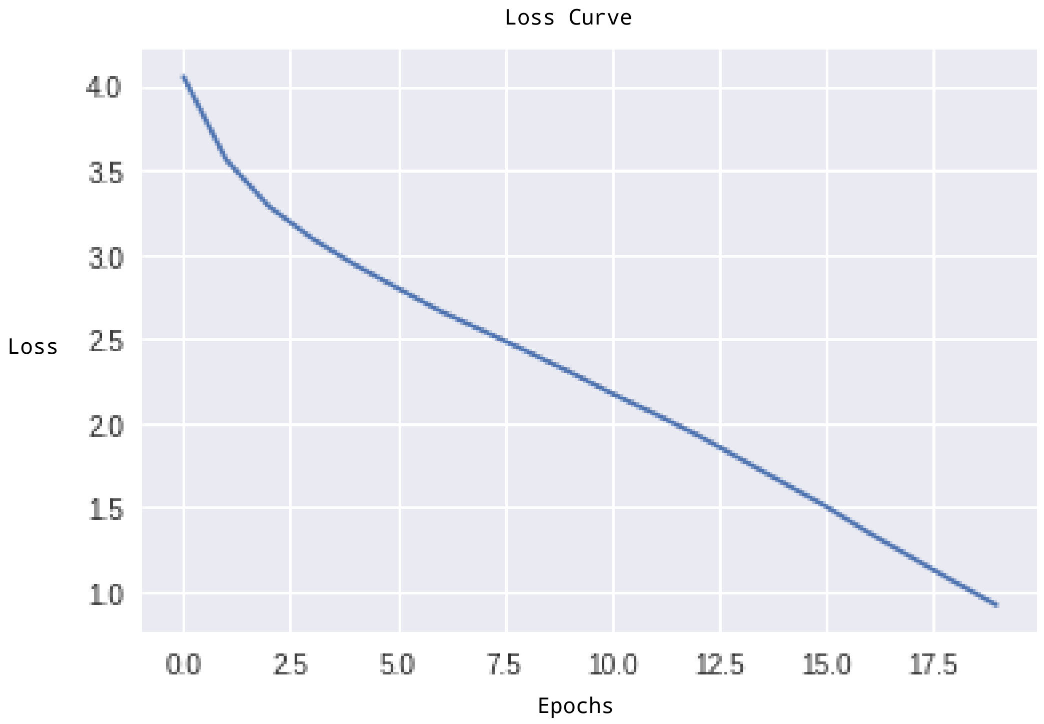


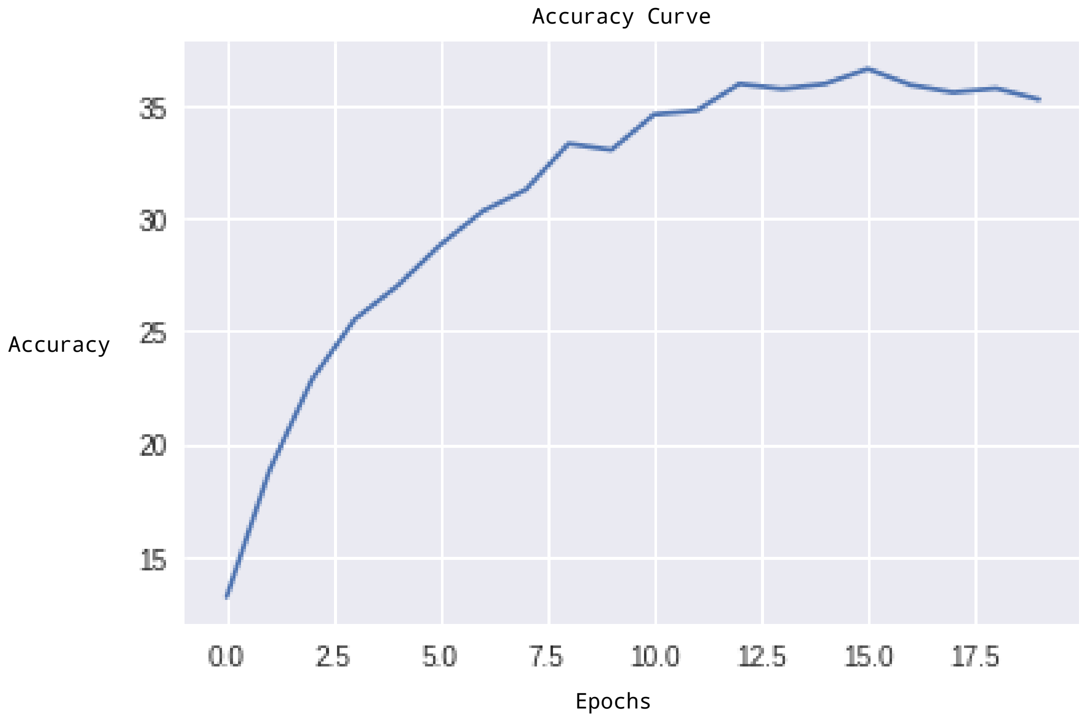
Dropout is a regularization technique patented by Google for reducing overfitting in neural networks. As a matter of fact, this is the most efficient Convolutional Neural Network tried until this point.

CNN with Random Horizontal Flipping

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 128, kernel_size=5, stride=2, padding=0)
        self.conv2 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=0)
        self.conv3 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=0)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.conv_final = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=0)
        self.fc1 = nn.Linear(256 * 4 * 4, 4096)
        self.fc2 = nn.Linear(4096, n_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = F.relu(self.pool(self.conv_final(x)))
        x = x.view(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

transform_train = transforms.Compose(
    [
     transforms.RandomHorizontalFlip(),
     transforms.Resize((32,32)),
     transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
    ])
```

```
Files already downloaded and verified
Files already downloaded and verified
[1,   195] loss: 4.054
Accuracy of the network on the test set: 13 %
[2,   195] loss: 3.562
Accuracy of the network on the test set: 18 %
[3,   195] loss: 3.285
Accuracy of the network on the test set: 22 %
[4,   195] loss: 3.097
Accuracy of the network on the test set: 25 %
[5,   195] loss: 2.940
Accuracy of the network on the test set: 26 %
[6,   195] loss: 2.801
Accuracy of the network on the test set: 28 %
[7,   195] loss: 2.664
Accuracy of the network on the test set: 30 %
[8,   195] loss: 2.547
Accuracy of the network on the test set: 31 %
[9,   195] loss: 2.428
Accuracy of the network on the test set: 33 %
[10,   195] loss: 2.305
Accuracy of the network on the test set: 33 %
[11,   195] loss: 2.176
Accuracy of the network on the test set: 34 %
[12,   195] loss: 2.055
Accuracy of the network on the test set: 34 %
[13,   195] loss: 1.929
Accuracy of the network on the test set: 35 %
[14,   195] loss: 1.790
Accuracy of the network on the test set: 35 %
[15,   195] loss: 1.648
Accuracy of the network on the test set: 35 %
[16,   195] loss: 1.508
Accuracy of the network on the test set: 36 %
[17,   195] loss: 1.351
Accuracy of the network on the test set: 35 %
[18,   195] loss: 1.205
Accuracy of the network on the test set: 35 %
[19,   195] loss: 1.060
Accuracy of the network on the test set: 35 %
[20,   195] loss: 0.919
Accuracy of the network on the test set: 35 %
Finished Training
Max accuracy: 36.60 %
Execution time: 00:06:10.33
```





Data augmentation is a way to increase the performance of CNNs and can reduce the probability of overfitting, in addition to dropout, on small databases.

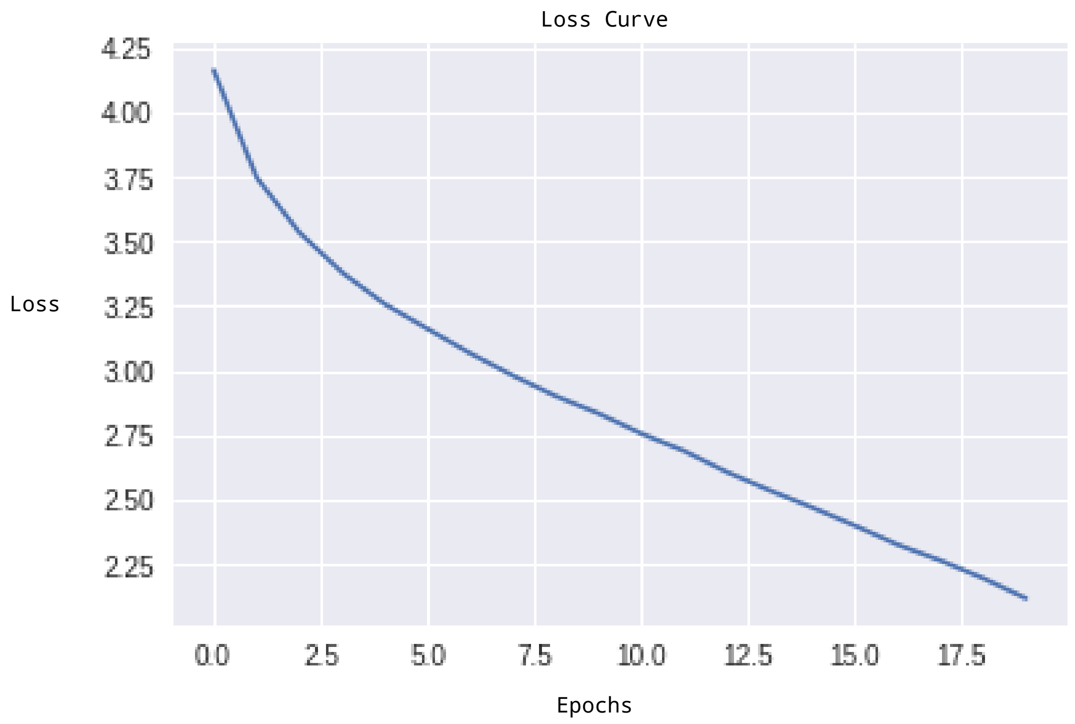
In fact we can see a slight increase in accuracy and loss values.

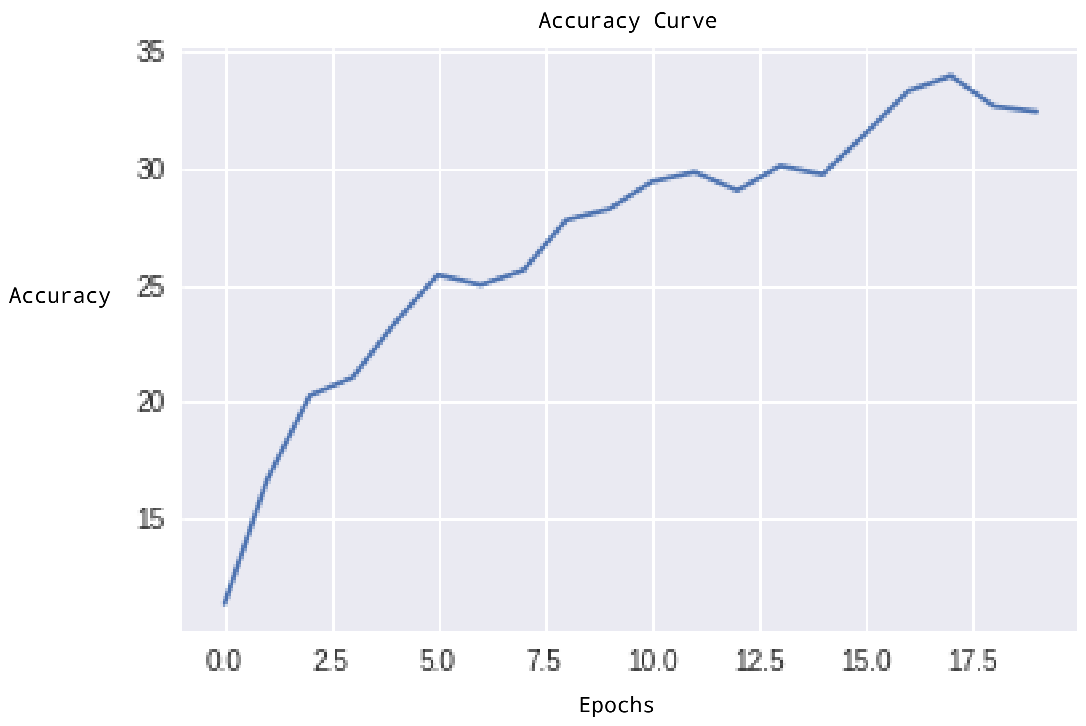
CNN with Random Crop

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 128, kernel_size=5, stride=2, padding=0)
        self.conv2 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=0)
        self.conv3 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=0)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.conv_final = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=0)
        self.fc1 = nn.Linear(256 * 4 * 4, 4096)
        self.fc2 = nn.Linear(4096, n_classes)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = F.relu(self.pool(self.conv_final(x)))
        x = x.view(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

transform_train = transforms.Compose(
    [
     transforms.Resize((40,40)),
     transforms.RandomCrop(32),
     transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
    ])
```

```
Files already downloaded and verified
Files already downloaded and verified
[1,   195] loss: 4.161
Accuracy of the network on the test set: 11 %
[2,   195] loss: 3.747
Accuracy of the network on the test set: 16 %
[3,   195] loss: 3.534
Accuracy of the network on the test set: 20 %
[4,   195] loss: 3.380
Accuracy of the network on the test set: 21 %
[5,   195] loss: 3.257
Accuracy of the network on the test set: 23 %
[6,   195] loss: 3.161
Accuracy of the network on the test set: 25 %
[7,   195] loss: 3.066
Accuracy of the network on the test set: 25 %
[8,   195] loss: 2.980
Accuracy of the network on the test set: 25 %
[9,   195] loss: 2.901
Accuracy of the network on the test set: 27 %
[10,   195] loss: 2.834
Accuracy of the network on the test set: 28 %
[11,   195] loss: 2.756
Accuracy of the network on the test set: 29 %
```

```
[12,   195] loss: 2.689
Accuracy of the network on the test set: 29 %
[13,   195] loss: 2.607
Accuracy of the network on the test set: 29 %
[14,   195] loss: 2.537
Accuracy of the network on the test set: 30 %
[15,   195] loss: 2.470
Accuracy of the network on the test set: 29 %
[16,   195] loss: 2.400
Accuracy of the network on the test set: 31 %
[17,   195] loss: 2.326
Accuracy of the network on the test set: 33 %
[18,   195] loss: 2.264
Accuracy of the network on the test set: 33 %
[19,   195] loss: 2.196
Accuracy of the network on the test set: 32 %
[20,   195] loss: 2.118
Accuracy of the network on the test set: 32 %
Finished Training
Max accuracy: 33.96 %
Execution time: 00:06:30.11
```





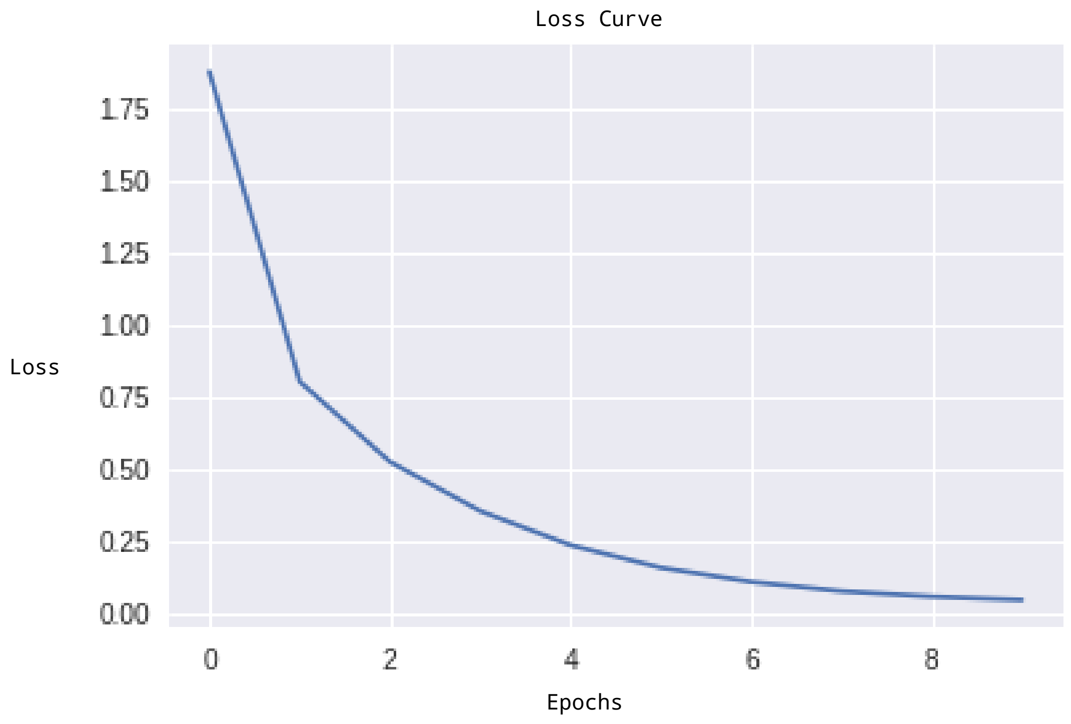
This type of data augmentation performs worse than Horizontal Flipping. This method, compared to the version without data augmentation, we notice that although tha values do not change, these are reached before.

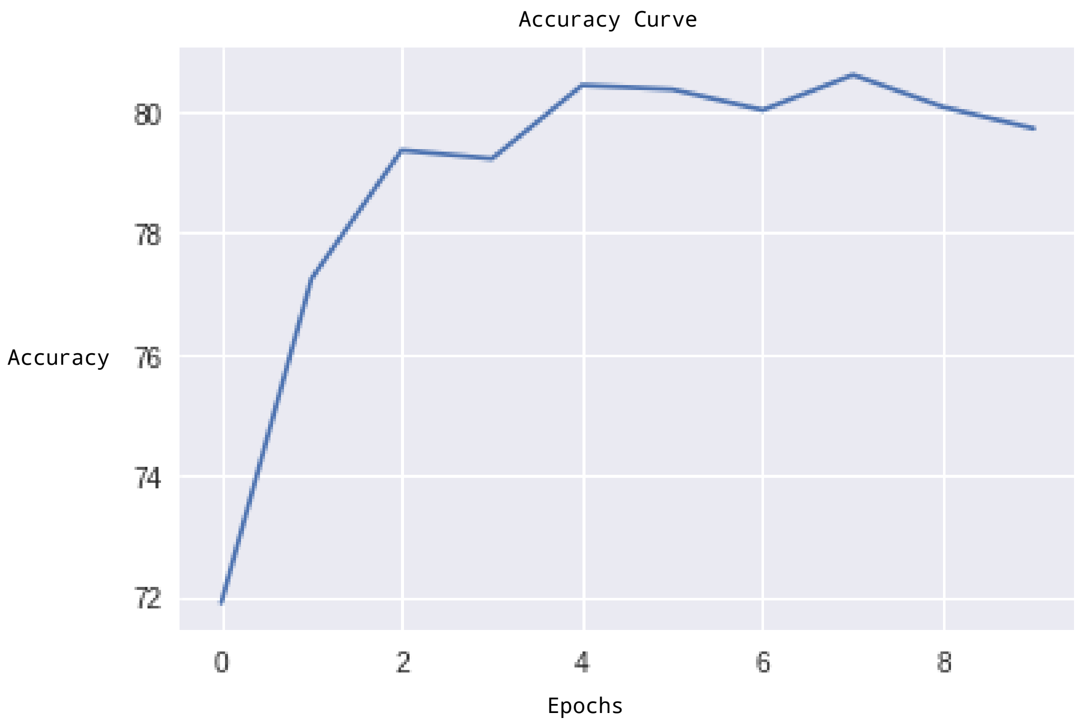
ResNet18

```
transform_train = transforms.Compose(
    [
     transforms.RandomHorizontalFlip(),
     transforms.Resize((224,224)),
     transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
    ])

transform_test = transforms.Compose(
    [
     transforms.Resize((224,224)),
     transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
     ])

trainset = torchvision.datasets.CIFAR100(root='./data', train=True,
                                        download=True, transform=transform_train)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=128,
                                          shuffle=True, num_workers=4,drop_last=True)

testset = torchvision.datasets.CIFAR100(root='./data', train=False,
                                       download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=128,
                                         shuffle=False, num_workers=4,drop_last=True)
```

```python
dataiter = iter(trainloader)

#for Residual Network:
net = models.resnet18(pretrained=True)
net.fc = nn.Linear(512, n_classes)
```

```
Files already downloaded and verified
Files already downloaded and verified
[1,   390] loss: 1.874
Accuracy of the network on the test set: 71 %
[2,   390] loss: 0.800
Accuracy of the network on the test set: 77 %
[3,   390] loss: 0.522
Accuracy of the network on the test set: 79 %
[4,   390] loss: 0.351
Accuracy of the network on the test set: 79 %
[5,   390] loss: 0.233
Accuracy of the network on the test set: 80 %
[6,   390] loss: 0.154
Accuracy of the network on the test set: 80 %
[7,   390] loss: 0.105
Accuracy of the network on the test set: 80 %
[8,   390] loss: 0.073
Accuracy of the network on the test set: 80 %
[9,   390] loss: 0.054
Accuracy of the network on the test set: 80 %
[10,   390] loss: 0.043
Accuracy of the network on the test set: 79 %
Finished Training
Max accuracy: 80.61 %
Execution time: 01:00:55.23
```





We clearly see how this type of network is the best view so far.

This is due to to the fact that there are a large number of additional layers, and it is also the reason why the execution time is very high.