Machine Learning
Home Assignment 6

Carlo Rosso rkm957

Contents

1 Logistic regression in PyTorch

```
class LogisticRegressionPytorch(nn.Module):
    def __init__(self, d, m):
      """
      d: input dimensionality
      m: number of classes (output dimensionality)
      """
      super(LogisticRegressionPytorch, self).__init__()
      self.fc = nn.Linear(d, m)                    # added
      self.activation_fn = nn.Sigmoid()            # added

    def forward(self, x):
      """
      x: input data
      """
```

```
        return self.activation_fn(self.fc(x))          # added

# definition of the loss function
criterion = nn.MultiMarginLoss()                       # added

# definition of the training
for epoch in range(no_epochs):  # Loop over the dataset multiple times
    # Zero the parameter gradients
    optimizer.zero_grad()

    # Forward + backward + optimize
    outputs = logreg_pytorch(X_train_T)                # added
    loss = criterion(outputs, y_train_T)               # added
    loss.backward()                                    # added
    optimizer.step()

# getting the biases
bs_torch = logreg_pytorch.fc.bias.detach().numpy()  # added
```

While I don't get the same accuracy on the training and test data, I think the implementation is correct, because the losses aren't too different and the weights are initialized to random values.

2 Convolutional Neural Networks

2.1 Sobel filter

```
# define convolutional layer
conv = nn.Conv2d(1, 2, W, padding = 0, bias=False)

# define the weights
simmetry = np.array([1, 0, -1])
g_x = torch.from_numpy(np.array([simmetry, 2 * simmetry, simmetry]))
params = torch.from_numpy(np.array([g_x, g_x.T])).reshape((2, 1, W, W))
conv.weight = torch.nn.Parameter(params.float(), requires_grad=False)

# apply the convolution
c = conv(x)
```

```
# combine the two channels to the final feature map
x2 = torch.sqrt(torch.sum(torch.square(c), dim=1))
```

So I defined the convolutional layer with 2 output channels. Considering the input image Figure 1, applying the convolutional layer results in two feature maps, one for each channel, respectively Figure 2 and Figure 3. Finally, I combined the two channels to obtain the final feature map Figure 4, and thus the Sobel filter.

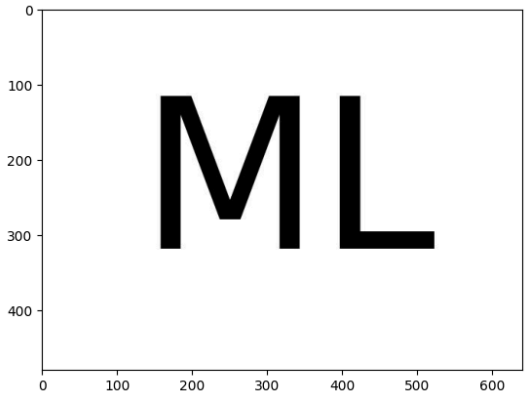

Figure 1: Input image

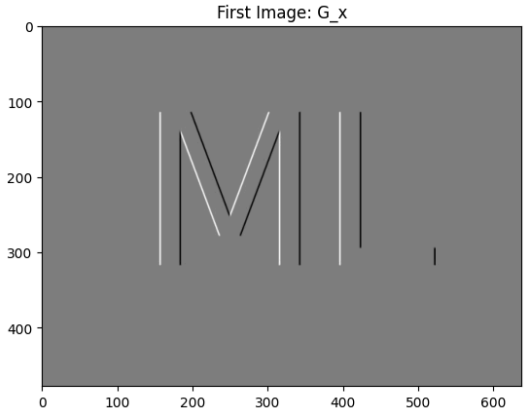

Figure 2: First channel of the feature map

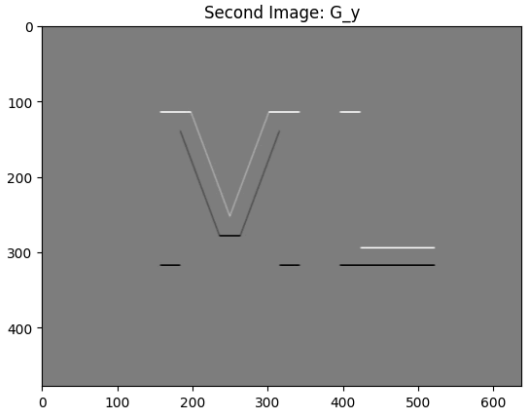

Figure 3: Second channel of the feature map

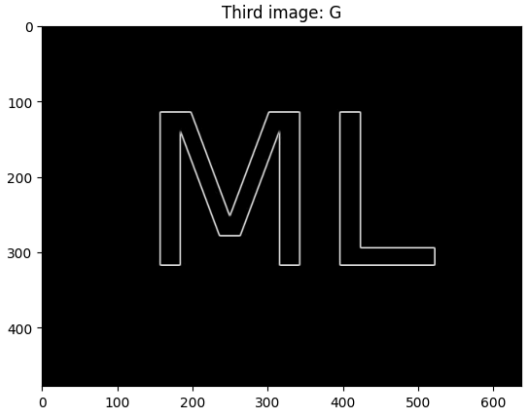

Figure 4: Combined feature map

2.2 Convolutional neural networks

```
class Net(nn.Module):
  def __init__(self, img_size=28):
    super(Net, self).__init__()
    self.conv1 = nn.Conv2d(3, 64, 5)
    self.pool1 = nn.MaxPool2d(2, 2)
    self.conv2 = nn.Conv2d(64, 64, 5)
    self.pool2 = nn.MaxPool2d(2, 2)
    self.fc1 = nn.Linear(1024, 43)

  def forward(self, x):
    x = self.pool1(F.elu(self.conv1(x)))
    x = self.pool2(F.elu(self.conv2(x)))
    x = torch.flatten(x, 1)
```

```
    x = F.elu(self.fc1(x))
    return x
```

2.3 Augmentation

Which transformations are applied to the input images during training?

The `Data` class in `GTSRBTrafficSigns.py` applies the following transformations to augment the dataset:

1. **Resize**: which makes all images the same size, 32x32 pixels.
2. **RandomAffine**: applies slight random rotation to the images.
3. **RandomCrop**: extracts random sections of the images.
4. **CenterCrop**: focuses on the center of the images.
5. **ColorJitter**: adjusts the brightness and contrast of the images.
6. **RandomHorizontalFlip**: flips the images horizontally.