AML Convolutional Neural Network for MNIST

Based on https://github.com/Atcold/pytorch-Deep-Learning

Data and Libraries

In [3]:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import numpy, random


# set the PseudoRandom Generator Seeds for better reproducibility
# see here for more: https://pytorch.org/docs/stable/notes/randomness.html
torch.manual_seed(99)
random.seed(99)
numpy.random.seed(99)


# this 'device' will be used for training our model
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda:0
```

Load the MNIST dataset

Observe that we set `shuffle=True`, which means that data is randomized

In [4]:

```
input_size  = 28*28   # images are 28x28 pixels
output_size = 10      # there are 10 classes

train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=64, shuffle=True)

test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=False, transform=transforms.Compo
se([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ])),
    batch_size=1000, shuffle=True)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ub
yte.gz to ../data/MNIST/raw/train-images-idx3-ubyte.gz
```

```
Extracting ../data/MNIST/raw/train-images-idx3-ubyte.gz to ../dat
a/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ub
yte.gz to ../data/MNIST/raw/train-labels-idx1-ubyte.gz
```

```
Extracting ../data/MNIST/raw/train-labels-idx1-ubyte.gz to ../dat
a/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-uby
te.gz to ../data/MNIST/raw/t10k-images-idx3-ubyte.gz
```

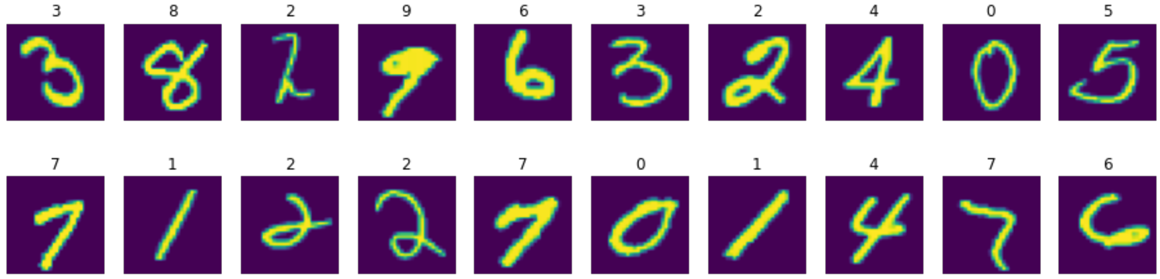
```
Extracting ../data/MNIST/raw/t10k-images-idx3-ubyte.gz to ../dat
a/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-uby
te.gz to ../data/MNIST/raw/t10k-labels-idx1-ubyte.gz
```

```
Extracting ../data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ../dat
a/MNIST/raw
Processing...
Done!
```

```
/opt/conda/lib/python3.7/site-packages/torchvision/datasets/mnis
t.py:480: UserWarning: The given NumPy array is not writeable, an
d PyTorch does not support non-writeable tensors. This means you
can write to the underlying (supposedly non-writeable) NumPy arra
y using the tensor. You may want to copy the array to protect its
data or make it writeable before converting it to a tensor. This
type of warning will be suppressed for the rest of this program.
(Triggered internally at  /pytorch/torch/csrc/utils/tensor_nump
y.cpp:141.)
  return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*
s)
```

In [5]:

```
# show some training images
plt.figure(figsize=(16, 4))

# fetch a batch of train images; RANDOM
image_batch, label_batch = next(iter(train_loader))

for i in range(20):
    image = image_batch[i]
    label = label_batch[i].item()
    plt.subplot(2, 10, i + 1)
    #image, label = train_loader.dataset.__getitem__(i)
    plt.imshow(image.squeeze().numpy())
    plt.axis('off')
    plt.title(label)
```



Helper functions for training and testing

In [6]:

```
# function to count number of parameters
def get_n_params(model):
    np=0
    for p in list(model.parameters()):
        np += p.nelement()
    return np

accuracy_list = []
# we pass a model object to this trainer, and it trains this model for
one epoch
def train(epoch, model):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        # send to device
        data, target = data.to(device), target.to(device)

        optimizer.zero_grad()
        output = model(data)
        #loss = F.nll_loss(output, target)
        loss = F.cross_entropy(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 100 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.f
ormat(
                epoch, batch_idx * len(data), len(train_loader.datase
t),
                100. * batch_idx / len(train_loader), loss.item()))

def test(model):
    model.eval()
    test_loss = 0
    correct = 0
    for data, target in test_loader:
        # send to device
        data, target = data.to(device), target.to(device)

        output = model(data)
        #test_loss += F.nll_loss(output, target, reduction='sum').item
() # sum up batch loss
        test_loss += F.cross_entropy(output, target, reduction='su
m').item() # sum up batch loss
```

```
        pred = output.data.max(1, keepdim=True)[1] # get the index of the max log-probability
        correct += pred.eq(target.data.view_as(pred)).cpu().sum().item()

    test_loss /= len(test_loader.dataset)
    accuracy = 100. * correct / len(test_loader.dataset)
    accuracy_list.append(accuracy)
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        accuracy))
```

The Convolutional Network

In [7]:

```
class CNN(nn.Module):
    def __init__(self, input_size, output_size):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=12, kernel_size=3,padding=0)
        self.conv2 = nn.Conv2d(in_channels=12, out_channels=24, kernel_size=6,padding=0)
        self.conv3 = nn.Conv2d(in_channels=24, out_channels=32, kernel_size=6,padding=0)
        self.fc1 = nn.Linear(8*4*4, 200)
        self.fc2 = nn.Linear(200, 10)

    def forward(self, x, verbose=False):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, kernel_size=2)
        x = self.conv3(x)
        x = F.relu(x)
        x = F.max_pool2d(x, kernel_size=2)
        x = x.view(-1, 8*4*4)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.log_softmax(x, dim=1)
        return x
```

Train the Network

In [8]:

```
print("Training on ", device)
model_cnn = CNN(input_size, output_size)
model_cnn.to(device)
optimizer = optim.SGD(model_cnn.parameters(), lr=0.01, momentum=0.5)
print('Number of parameters: {}'.format(get_n_params(model_cnn)))

for epoch in range(0, 10):
    train(epoch, model_cnn)
    test(model_cnn)
```

```
Training on  cuda:0
Number of parameters: 66002
Train Epoch: 0 [0/60000 (0%)]	Loss: 2.314376
Train Epoch: 0 [6400/60000 (11%)]	Loss: 1.247057
Train Epoch: 0 [12800/60000 (21%)]	Loss: 0.543811
Train Epoch: 0 [19200/60000 (32%)]	Loss: 0.348373
Train Epoch: 0 [25600/60000 (43%)]	Loss: 0.439647
Train Epoch: 0 [32000/60000 (53%)]	Loss: 0.334617
Train Epoch: 0 [38400/60000 (64%)]	Loss: 0.239256
Train Epoch: 0 [44800/60000 (75%)]	Loss: 0.030815
Train Epoch: 0 [51200/60000 (85%)]	Loss: 0.211453
Train Epoch: 0 [57600/60000 (96%)]	Loss: 0.032946

Test set: Average loss: 0.1618, Accuracy: 9515/10000 (95%)

Train Epoch: 1 [0/60000 (0%)]	Loss: 0.281891
Train Epoch: 1 [6400/60000 (11%)]	Loss: 0.091353
Train Epoch: 1 [12800/60000 (21%)]	Loss: 0.082387
Train Epoch: 1 [19200/60000 (32%)]	Loss: 0.112644
Train Epoch: 1 [25600/60000 (43%)]	Loss: 0.094412
Train Epoch: 1 [32000/60000 (53%)]	Loss: 0.074808
Train Epoch: 1 [38400/60000 (64%)]	Loss: 0.309685
Train Epoch: 1 [44800/60000 (75%)]	Loss: 0.043199
Train Epoch: 1 [51200/60000 (85%)]	Loss: 0.149641
Train Epoch: 1 [57600/60000 (96%)]	Loss: 0.073493

Test set: Average loss: 0.0841, Accuracy: 9752/10000 (98%)

Train Epoch: 2 [0/60000 (0%)]	Loss: 0.091149
Train Epoch: 2 [6400/60000 (11%)]	Loss: 0.173618
Train Epoch: 2 [12800/60000 (21%)]	Loss: 0.119580
Train Epoch: 2 [19200/60000 (32%)]	Loss: 0.051772
Train Epoch: 2 [25600/60000 (43%)]	Loss: 0.042329
Train Epoch: 2 [32000/60000 (53%)]	Loss: 0.045354
Train Epoch: 2 [38400/60000 (64%)]	Loss: 0.080187
Train Epoch: 2 [44800/60000 (75%)]	Loss: 0.030847
Train Epoch: 2 [51200/60000 (85%)]	Loss: 0.112225
Train Epoch: 2 [57600/60000 (96%)]	Loss: 0.096882

Test set: Average loss: 0.0717, Accuracy: 9764/10000 (98%)

Train Epoch: 3 [0/60000 (0%)]	Loss: 0.041779
```

```
Train Epoch: 3 [6400/60000 (11%)]	Loss: 0.026723
Train Epoch: 3 [12800/60000 (21%)]	Loss: 0.079777
Train Epoch: 3 [19200/60000 (32%)]	Loss: 0.073896
Train Epoch: 3 [25600/60000 (43%)]	Loss: 0.041413
Train Epoch: 3 [32000/60000 (53%)]	Loss: 0.018171
Train Epoch: 3 [38400/60000 (64%)]	Loss: 0.132102
Train Epoch: 3 [44800/60000 (75%)]	Loss: 0.055729
Train Epoch: 3 [51200/60000 (85%)]	Loss: 0.094866
Train Epoch: 3 [57600/60000 (96%)]	Loss: 0.062711

Test set: Average loss: 0.0562, Accuracy: 9826/10000 (98%)

Train Epoch: 4 [0/60000 (0%)]	Loss: 0.185694
Train Epoch: 4 [6400/60000 (11%)]	Loss: 0.033204
Train Epoch: 4 [12800/60000 (21%)]	Loss: 0.009174
Train Epoch: 4 [19200/60000 (32%)]	Loss: 0.065939
Train Epoch: 4 [25600/60000 (43%)]	Loss: 0.015719
Train Epoch: 4 [32000/60000 (53%)]	Loss: 0.131551
Train Epoch: 4 [38400/60000 (64%)]	Loss: 0.013528
Train Epoch: 4 [44800/60000 (75%)]	Loss: 0.029064
Train Epoch: 4 [51200/60000 (85%)]	Loss: 0.032624
Train Epoch: 4 [57600/60000 (96%)]	Loss: 0.038630

Test set: Average loss: 0.0560, Accuracy: 9823/10000 (98%)

Train Epoch: 5 [0/60000 (0%)]	Loss: 0.008610
Train Epoch: 5 [6400/60000 (11%)]	Loss: 0.072935
Train Epoch: 5 [12800/60000 (21%)]	Loss: 0.036093
Train Epoch: 5 [19200/60000 (32%)]	Loss: 0.014054
Train Epoch: 5 [25600/60000 (43%)]	Loss: 0.002158
Train Epoch: 5 [32000/60000 (53%)]	Loss: 0.054400
Train Epoch: 5 [38400/60000 (64%)]	Loss: 0.101882
Train Epoch: 5 [44800/60000 (75%)]	Loss: 0.019061
Train Epoch: 5 [51200/60000 (85%)]	Loss: 0.007275
Train Epoch: 5 [57600/60000 (96%)]	Loss: 0.027882

Test set: Average loss: 0.0479, Accuracy: 9853/10000 (99%)

Train Epoch: 6 [0/60000 (0%)]	Loss: 0.027948
Train Epoch: 6 [6400/60000 (11%)]	Loss: 0.092156
Train Epoch: 6 [12800/60000 (21%)]	Loss: 0.103358
Train Epoch: 6 [19200/60000 (32%)]	Loss: 0.029406
Train Epoch: 6 [25600/60000 (43%)]	Loss: 0.020894
```

```
Train Epoch: 6 [32000/60000 (53%)]	Loss: 0.018326
Train Epoch: 6 [38400/60000 (64%)]	Loss: 0.011901
Train Epoch: 6 [44800/60000 (75%)]	Loss: 0.029184
Train Epoch: 6 [51200/60000 (85%)]	Loss: 0.013231
Train Epoch: 6 [57600/60000 (96%)]	Loss: 0.054908

Test set: Average loss: 0.0464, Accuracy: 9859/10000 (99%)

Train Epoch: 7 [0/60000 (0%)]	Loss: 0.035456
Train Epoch: 7 [6400/60000 (11%)]	Loss: 0.017647
Train Epoch: 7 [12800/60000 (21%)]	Loss: 0.123160
Train Epoch: 7 [19200/60000 (32%)]	Loss: 0.012506
Train Epoch: 7 [25600/60000 (43%)]	Loss: 0.004673
Train Epoch: 7 [32000/60000 (53%)]	Loss: 0.017831
Train Epoch: 7 [38400/60000 (64%)]	Loss: 0.014590
Train Epoch: 7 [44800/60000 (75%)]	Loss: 0.016720
Train Epoch: 7 [51200/60000 (85%)]	Loss: 0.110885
Train Epoch: 7 [57600/60000 (96%)]	Loss: 0.062288

Test set: Average loss: 0.0410, Accuracy: 9874/10000 (99%)

Train Epoch: 8 [0/60000 (0%)]	Loss: 0.023103
Train Epoch: 8 [6400/60000 (11%)]	Loss: 0.034757
Train Epoch: 8 [12800/60000 (21%)]	Loss: 0.009517
Train Epoch: 8 [19200/60000 (32%)]	Loss: 0.014460
Train Epoch: 8 [25600/60000 (43%)]	Loss: 0.029975
Train Epoch: 8 [32000/60000 (53%)]	Loss: 0.025051
Train Epoch: 8 [38400/60000 (64%)]	Loss: 0.010092
Train Epoch: 8 [44800/60000 (75%)]	Loss: 0.013215
Train Epoch: 8 [51200/60000 (85%)]	Loss: 0.036333
Train Epoch: 8 [57600/60000 (96%)]	Loss: 0.019535

Test set: Average loss: 0.0428, Accuracy: 9858/10000 (99%)

Train Epoch: 9 [0/60000 (0%)]	Loss: 0.025493
Train Epoch: 9 [6400/60000 (11%)]	Loss: 0.116543
Train Epoch: 9 [12800/60000 (21%)]	Loss: 0.001867
Train Epoch: 9 [19200/60000 (32%)]	Loss: 0.014267
Train Epoch: 9 [25600/60000 (43%)]	Loss: 0.127119
Train Epoch: 9 [32000/60000 (53%)]	Loss: 0.009173
Train Epoch: 9 [38400/60000 (64%)]	Loss: 0.118213
Train Epoch: 9 [44800/60000 (75%)]	Loss: 0.004635
Train Epoch: 9 [51200/60000 (85%)]	Loss: 0.010975
```

```
Train Epoch: 9 [57600/60000 (96%)]      Loss: 0.005096

Test set: Average loss: 0.0474, Accuracy: 9849/10000 (98%)
```

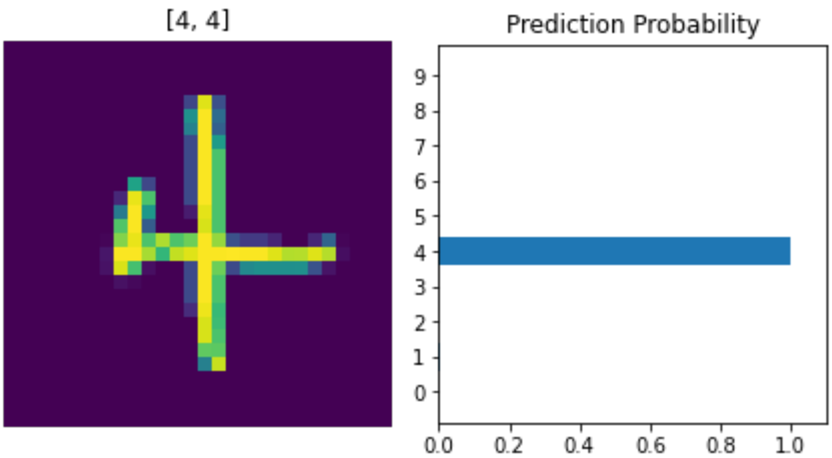
Show some predictions of the test network

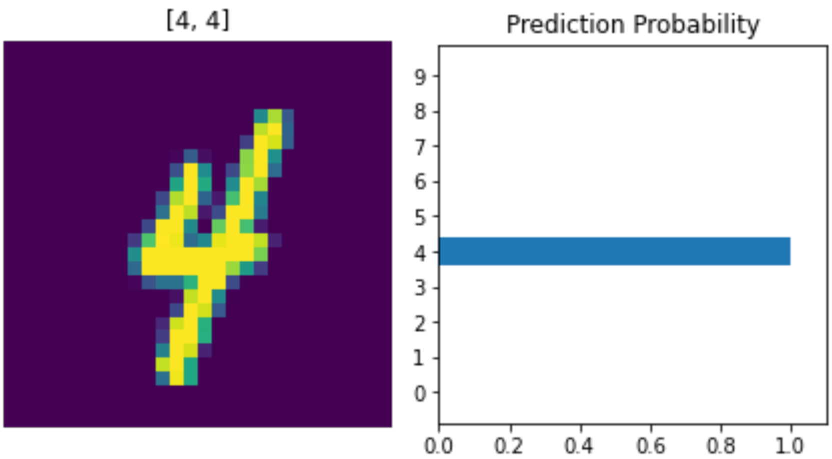
In [9]:

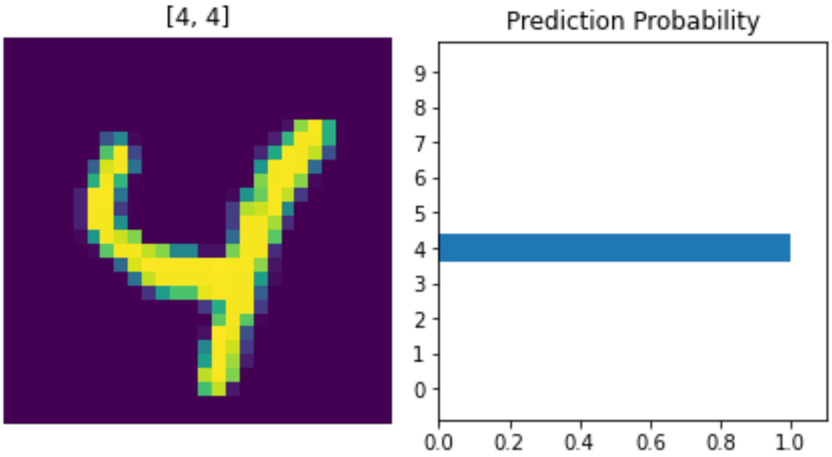
```
def visualize_pred(img, pred_prob, real_label):
    ''' Function for viewing an image and it's predicted classes.
    '''
    #pred_prob = pred_prob.data.numpy().squeeze()

    fig, (ax1, ax2) = plt.subplots(figsize=(6,9), ncols=2)
    ax1.imshow(img.numpy().squeeze())
    ax1.axis('off')
    pred_label = numpy.argmax(pred_prob)
    ax1.set_title([real_label, pred_label])

    ax2.barh(numpy.arange(10), pred_prob)
    ax2.set_aspect(0.1)
    ax2.set_yticks(numpy.arange(10))
    ax2.set_yticklabels(numpy.arange(10))
    ax2.set_title('Prediction Probability')
    ax2.set_xlim(0, 1.1)
    plt.tight_layout()
```

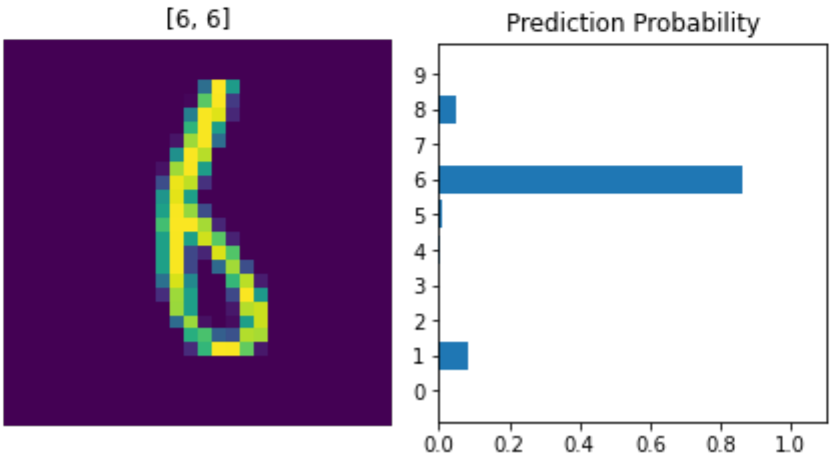
In [10]:

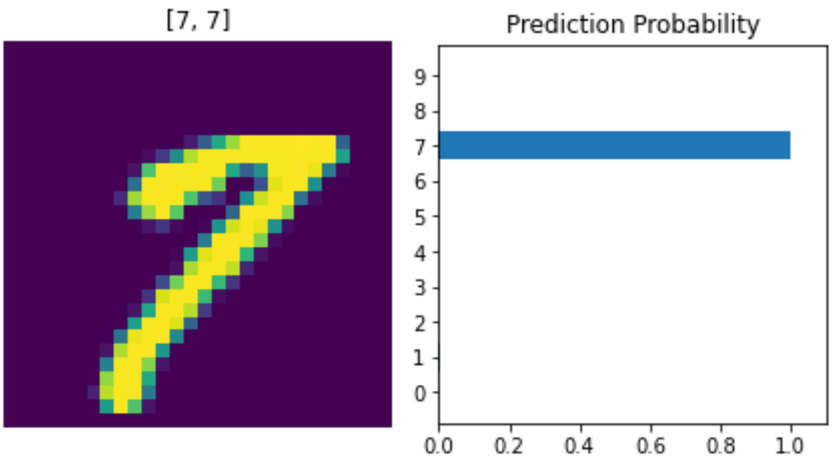
```
model_cnn.to('cpu')

# fetch a batch of test images
image_batch, label_batch = next(iter(test_loader))

# Turn off gradients to speed up this part
with torch.no_grad():
    log_pred_prob_batch = model_cnn(image_batch)
for i in range(10):
    img = image_batch[i]
    real_label = label_batch[i].item()
    log_pred_prob = log_pred_prob_batch[i]
    # Output of the network are log-probabilities, need to take expone
ntial for probabilities
    pred_prob = torch.exp(log_pred_prob).data.numpy().squeeze()
    visualize_pred(img, pred_prob, real_label)
```

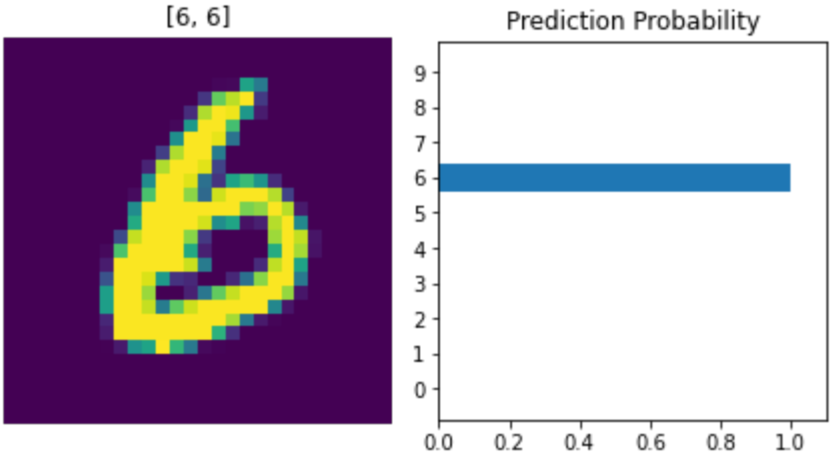


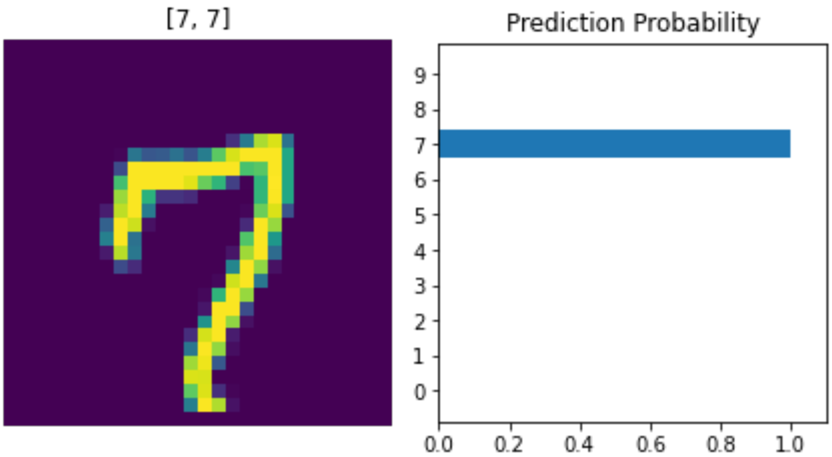


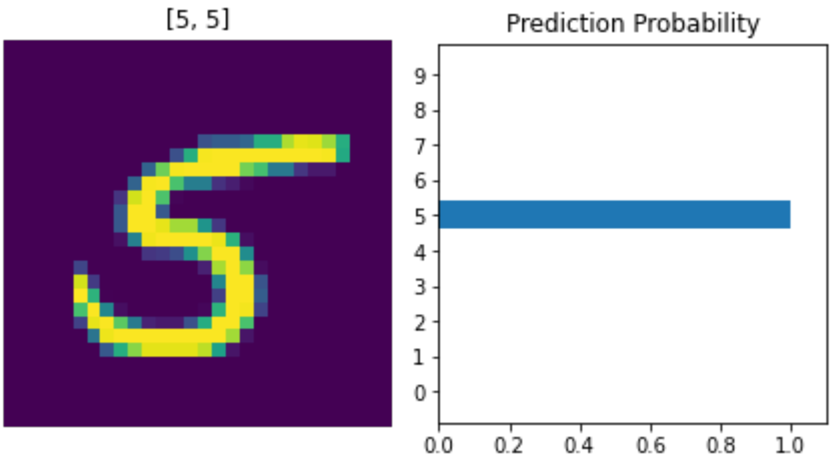

[4, 4]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0

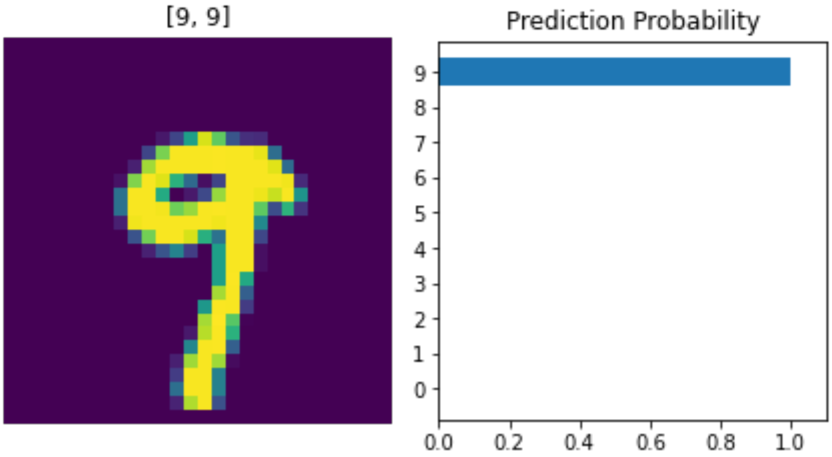

[6, 6]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0

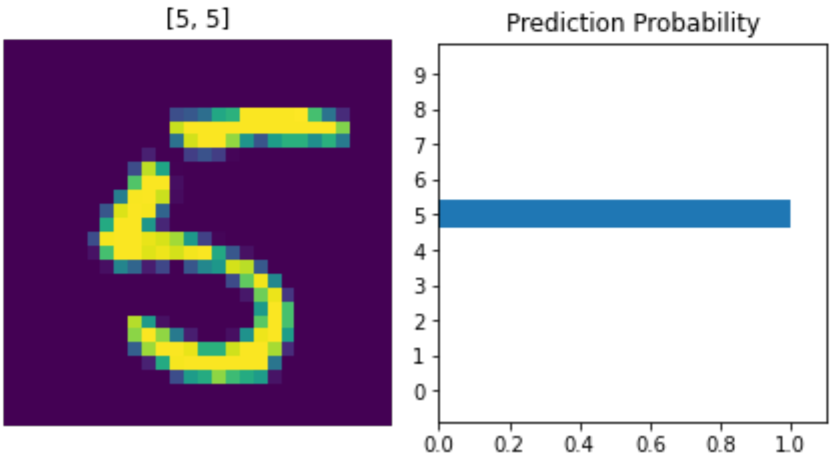

[7, 7]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0


[6, 6]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0


[7, 7]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0


[5, 5]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0





Network with Dropout

In [11]:

```
class CNNDropout(nn.Module):
    def __init__(self, input_size, output_size):
        super(CNNDropout, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=12, kernel
_size=3,padding=0)
        self.conv2 = nn.Conv2d(in_channels=12, out_channels=24, kerne
l_size=6,padding=0)
        self.conv3 = nn.Conv2d(in_channels=24, out_channels=32, kerne
l_size=6,padding=0)
        self.fc1 = nn.Linear(8*4*4, 200)
        self.do1 = nn.Dropout2d(p=0.8)
        self.fc2 = nn.Linear(200, 10)

    def forward(self, x, verbose=False):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, kernel_size=2)
        x = self.conv3(x)
        x = F.relu(x)
        x = F.max_pool2d(x, kernel_size=2)
        x = x.view(-1, 8*4*4)
        x = self.fc1(x)
        x = self.do1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.log_softmax(x, dim=1)
        return x
```

In [12]:

```
print("Training on ", device)
model_2 = CNNDropout(input_size, output_size)
model_2.to(device)
optimizer = optim.SGD(model_2.parameters(), lr=0.01, momentum=0.5)
print('Number of parameters: {}'.format(get_n_params(model_2)))

for epoch in range(0, 10):
    model_2.train() # model in training mode. Turns on dropout, batch-
norm etc during training
    train(epoch, model_2)
    model_2.eval() # model in evaluation mode. Turn off dropout, batc
h-norm etc during validation/testing
    test(model_2)
```

```
Training on  cuda:0
Number of parameters: 66002
Train Epoch: 0 [0/60000 (0%)]	Loss: 2.306069
Train Epoch: 0 [6400/60000 (11%)]	Loss: 1.758997
Train Epoch: 0 [12800/60000 (21%)]	Loss: 0.908702
Train Epoch: 0 [19200/60000 (32%)]	Loss: 0.667039
Train Epoch: 0 [25600/60000 (43%)]	Loss: 0.543066
Train Epoch: 0 [32000/60000 (53%)]	Loss: 0.635493
Train Epoch: 0 [38400/60000 (64%)]	Loss: 0.378995
Train Epoch: 0 [44800/60000 (75%)]	Loss: 0.287397
Train Epoch: 0 [51200/60000 (85%)]	Loss: 0.194406
Train Epoch: 0 [57600/60000 (96%)]	Loss: 0.298970

Test set: Average loss: 0.1656, Accuracy: 9474/10000 (95%)

Train Epoch: 1 [0/60000 (0%)]	Loss: 0.419913
Train Epoch: 1 [6400/60000 (11%)]	Loss: 0.202465
Train Epoch: 1 [12800/60000 (21%)]	Loss: 0.164826
Train Epoch: 1 [19200/60000 (32%)]	Loss: 0.509709
Train Epoch: 1 [25600/60000 (43%)]	Loss: 0.188931
Train Epoch: 1 [32000/60000 (53%)]	Loss: 0.233923
Train Epoch: 1 [38400/60000 (64%)]	Loss: 0.360609
Train Epoch: 1 [44800/60000 (75%)]	Loss: 0.147047
Train Epoch: 1 [51200/60000 (85%)]	Loss: 0.263204
Train Epoch: 1 [57600/60000 (96%)]	Loss: 0.132362

Test set: Average loss: 0.0937, Accuracy: 9713/10000 (97%)

Train Epoch: 2 [0/60000 (0%)]	Loss: 0.199842
Train Epoch: 2 [6400/60000 (11%)]	Loss: 0.225595
Train Epoch: 2 [12800/60000 (21%)]	Loss: 0.074334
Train Epoch: 2 [19200/60000 (32%)]	Loss: 0.081297
Train Epoch: 2 [25600/60000 (43%)]	Loss: 0.124935
Train Epoch: 2 [32000/60000 (53%)]	Loss: 0.157687
Train Epoch: 2 [38400/60000 (64%)]	Loss: 0.153288
Train Epoch: 2 [44800/60000 (75%)]	Loss: 0.160657
Train Epoch: 2 [51200/60000 (85%)]	Loss: 0.126917
Train Epoch: 2 [57600/60000 (96%)]	Loss: 0.260541

Test set: Average loss: 0.0687, Accuracy: 9797/10000 (98%)

Train Epoch: 3 [0/60000 (0%)]	Loss: 0.114873
```

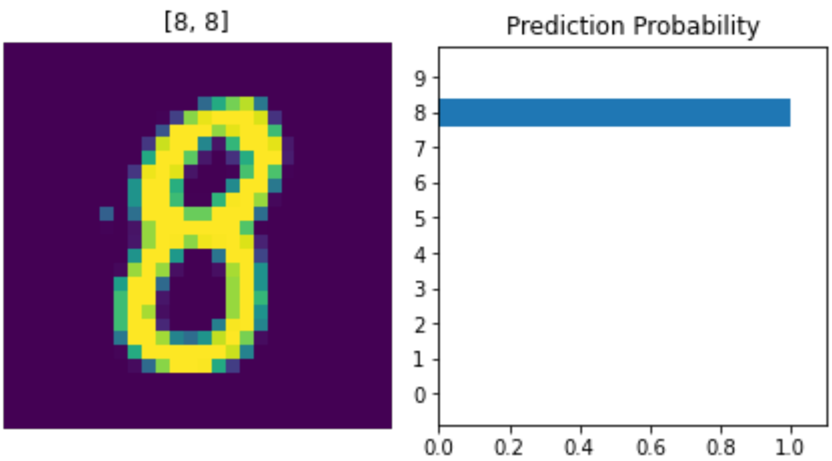
```
Train Epoch: 3 [6400/60000 (11%)]	Loss: 0.140660
Train Epoch: 3 [12800/60000 (21%)]	Loss: 0.181974
Train Epoch: 3 [19200/60000 (32%)]	Loss: 0.152856
Train Epoch: 3 [25600/60000 (43%)]	Loss: 0.221724
Train Epoch: 3 [32000/60000 (53%)]	Loss: 0.050747
Train Epoch: 3 [38400/60000 (64%)]	Loss: 0.060080
Train Epoch: 3 [44800/60000 (75%)]	Loss: 0.043175
Train Epoch: 3 [51200/60000 (85%)]	Loss: 0.142447
Train Epoch: 3 [57600/60000 (96%)]	Loss: 0.059221

Test set: Average loss: 0.0579, Accuracy: 9829/10000 (98%)

Train Epoch: 4 [0/60000 (0%)]	Loss: 0.093558
Train Epoch: 4 [6400/60000 (11%)]	Loss: 0.012862
Train Epoch: 4 [12800/60000 (21%)]	Loss: 0.041361
Train Epoch: 4 [19200/60000 (32%)]	Loss: 0.124632
Train Epoch: 4 [25600/60000 (43%)]	Loss: 0.094132
Train Epoch: 4 [32000/60000 (53%)]	Loss: 0.009214
Train Epoch: 4 [38400/60000 (64%)]	Loss: 0.257953
Train Epoch: 4 [44800/60000 (75%)]	Loss: 0.037816
Train Epoch: 4 [51200/60000 (85%)]	Loss: 0.117457
Train Epoch: 4 [57600/60000 (96%)]	Loss: 0.208364

Test set: Average loss: 0.0532, Accuracy: 9837/10000 (98%)

Train Epoch: 5 [0/60000 (0%)]	Loss: 0.234095
Train Epoch: 5 [6400/60000 (11%)]	Loss: 0.034321
Train Epoch: 5 [12800/60000 (21%)]	Loss: 0.039650
Train Epoch: 5 [19200/60000 (32%)]	Loss: 0.073291
Train Epoch: 5 [25600/60000 (43%)]	Loss: 0.082997
Train Epoch: 5 [32000/60000 (53%)]	Loss: 0.084169
Train Epoch: 5 [38400/60000 (64%)]	Loss: 0.061226
Train Epoch: 5 [44800/60000 (75%)]	Loss: 0.098424
Train Epoch: 5 [51200/60000 (85%)]	Loss: 0.015311
Train Epoch: 5 [57600/60000 (96%)]	Loss: 0.065204

Test set: Average loss: 0.0501, Accuracy: 9853/10000 (99%)

Train Epoch: 6 [0/60000 (0%)]	Loss: 0.165180
Train Epoch: 6 [6400/60000 (11%)]	Loss: 0.027101
Train Epoch: 6 [12800/60000 (21%)]	Loss: 0.234293
Train Epoch: 6 [19200/60000 (32%)]	Loss: 0.152707
Train Epoch: 6 [25600/60000 (43%)]	Loss: 0.032201
```

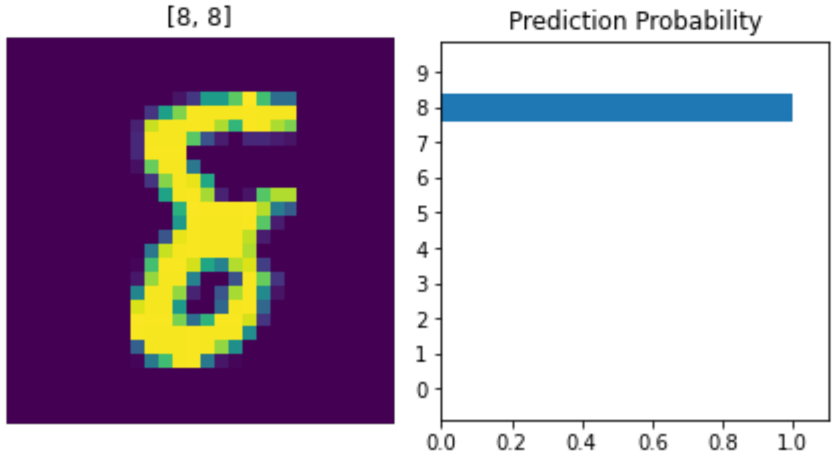
```
Train Epoch: 6 [32000/60000 (53%)]	Loss: 0.038543
Train Epoch: 6 [38400/60000 (64%)]	Loss: 0.144795
Train Epoch: 6 [44800/60000 (75%)]	Loss: 0.070931
Train Epoch: 6 [51200/60000 (85%)]	Loss: 0.095175
Train Epoch: 6 [57600/60000 (96%)]	Loss: 0.084652

Test set: Average loss: 0.0468, Accuracy: 9859/10000 (99%)

Train Epoch: 7 [0/60000 (0%)]	Loss: 0.075135
Train Epoch: 7 [6400/60000 (11%)]	Loss: 0.031033
Train Epoch: 7 [12800/60000 (21%)]	Loss: 0.101963
Train Epoch: 7 [19200/60000 (32%)]	Loss: 0.134962
Train Epoch: 7 [25600/60000 (43%)]	Loss: 0.068243
Train Epoch: 7 [32000/60000 (53%)]	Loss: 0.052485
Train Epoch: 7 [38400/60000 (64%)]	Loss: 0.175256
Train Epoch: 7 [44800/60000 (75%)]	Loss: 0.054473
Train Epoch: 7 [51200/60000 (85%)]	Loss: 0.068356
Train Epoch: 7 [57600/60000 (96%)]	Loss: 0.086020

Test set: Average loss: 0.0394, Accuracy: 9877/10000 (99%)

Train Epoch: 8 [0/60000 (0%)]	Loss: 0.085675
Train Epoch: 8 [6400/60000 (11%)]	Loss: 0.046595
Train Epoch: 8 [12800/60000 (21%)]	Loss: 0.095389
Train Epoch: 8 [19200/60000 (32%)]	Loss: 0.167578
Train Epoch: 8 [25600/60000 (43%)]	Loss: 0.084118
Train Epoch: 8 [32000/60000 (53%)]	Loss: 0.131149
Train Epoch: 8 [38400/60000 (64%)]	Loss: 0.019214
Train Epoch: 8 [44800/60000 (75%)]	Loss: 0.103678
Train Epoch: 8 [51200/60000 (85%)]	Loss: 0.047586
Train Epoch: 8 [57600/60000 (96%)]	Loss: 0.121740

Test set: Average loss: 0.0419, Accuracy: 9874/10000 (99%)

Train Epoch: 9 [0/60000 (0%)]	Loss: 0.080053
Train Epoch: 9 [6400/60000 (11%)]	Loss: 0.035413
Train Epoch: 9 [12800/60000 (21%)]	Loss: 0.121057
Train Epoch: 9 [19200/60000 (32%)]	Loss: 0.091089
Train Epoch: 9 [25600/60000 (43%)]	Loss: 0.081844
Train Epoch: 9 [32000/60000 (53%)]	Loss: 0.151014
Train Epoch: 9 [38400/60000 (64%)]	Loss: 0.038320
Train Epoch: 9 [44800/60000 (75%)]	Loss: 0.072954
Train Epoch: 9 [51200/60000 (85%)]	Loss: 0.010996
```

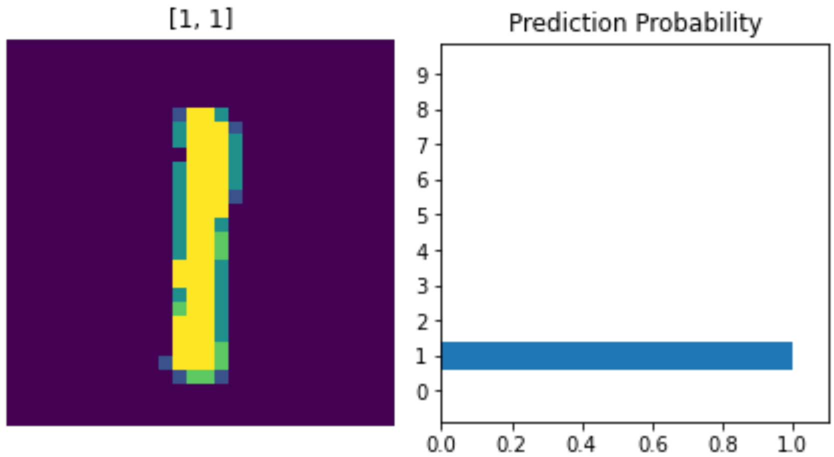
```
Train Epoch: 9 [57600/60000 (96%)]	Loss: 0.163306

Test set: Average loss: 0.0388, Accuracy: 9890/10000 (99%)
```

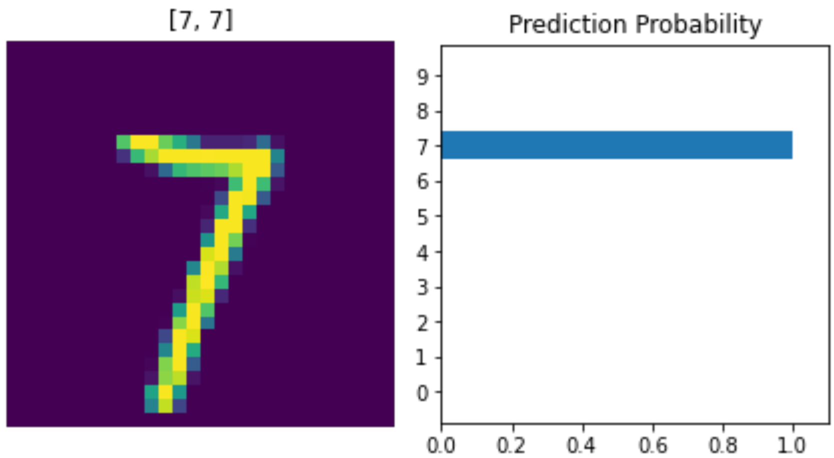
In [13]:

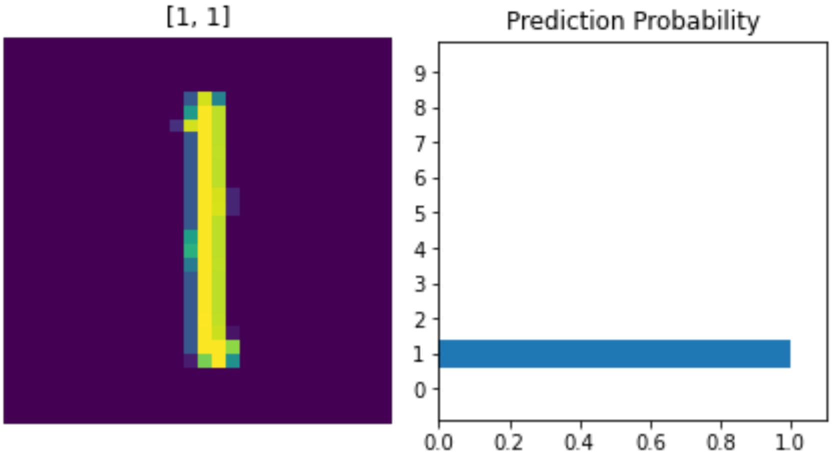
```
model_2.to('cpu')

# fetch a batch of test images
image_batch, label_batch = next(iter(test_loader))

# Turn off gradients to speed up this part
with torch.no_grad():
    log_pred_prob_batch = model_2(image_batch)
for i in range(10):
    img = image_batch[i]
    real_label = label_batch[i].item()
    log_pred_prob = log_pred_prob_batch[i]
    # Output of the network are log-probabilities, need to take expone
ntial for probabilities
    pred_prob = torch.exp(log_pred_prob).data.numpy().squeeze()
    visualize_pred(img, pred_prob, real_label)
```

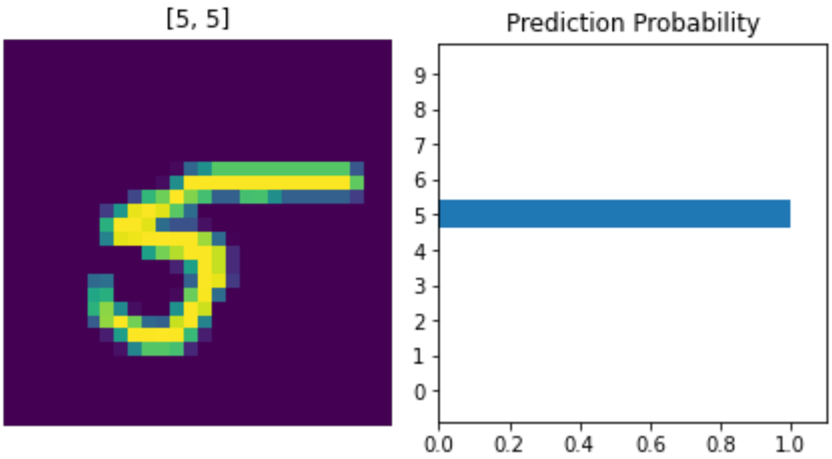


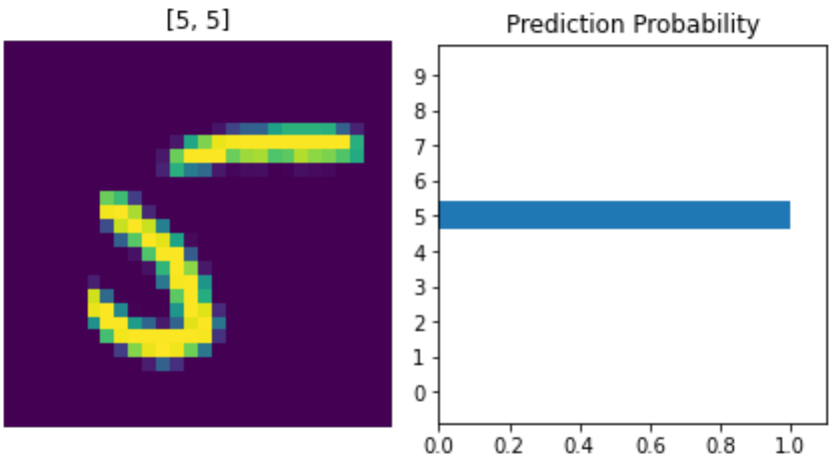

[8, 8]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0

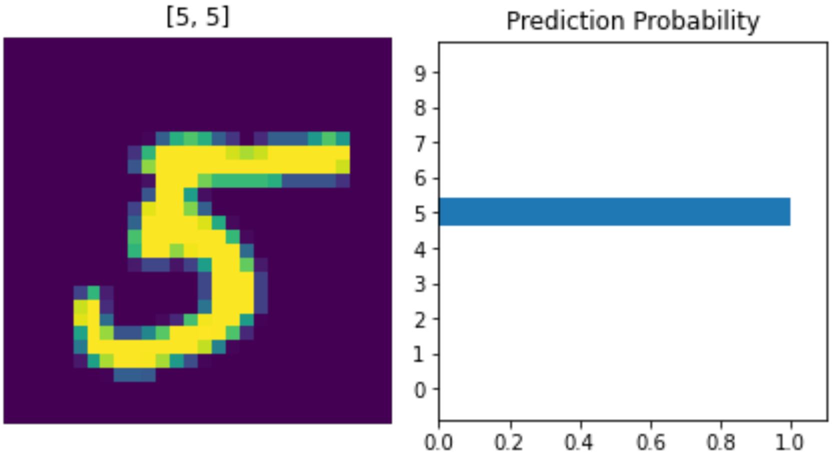

[1, 1]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0

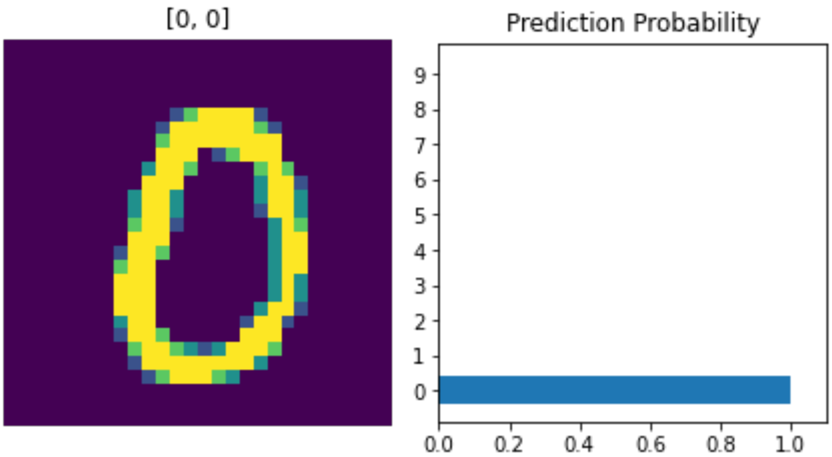

[7, 7]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0

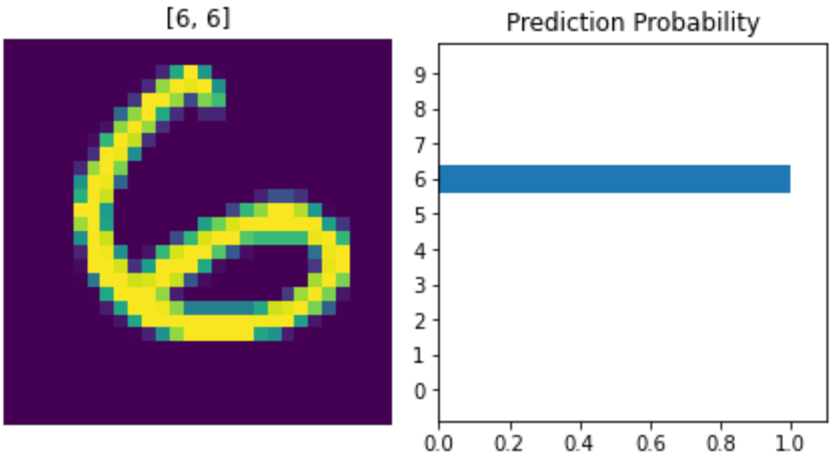

[1, 1]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0


[5, 5]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0


[5, 5]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0



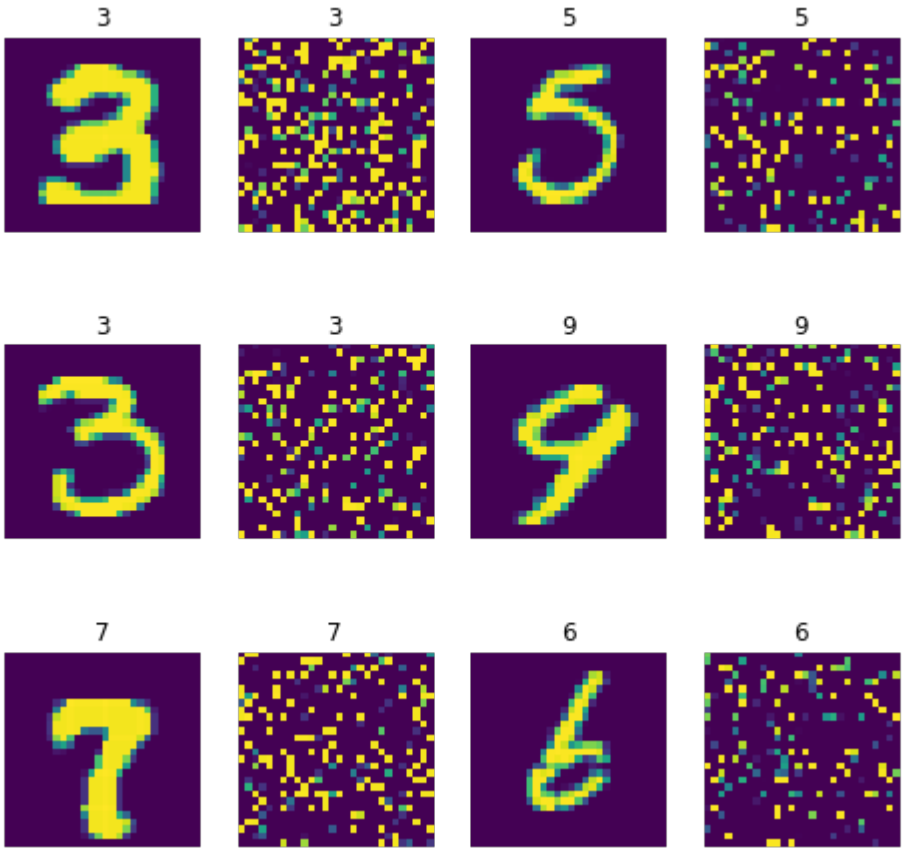




Does the CNN use "Visual Information" ?

In [14]:

```
fixed_perm = torch.randperm(784) # Fix a permutation of the image pix
els; We apply the same permutation to all images

# show some training images
plt.figure(figsize=(8, 8))

# fetch a batch of train images; RANDOM
image_batch, label_batch = next(iter(train_loader))

for i in range(6):
    image = image_batch[i]
    image_perm = image.view(-1, 28*28).clone()
    image_perm = image_perm[:, fixed_perm]
    image_perm = image_perm.view(-1, 1, 28, 28)

    label = label_batch[i].item()
    plt.subplot(3,4 , 2*i + 1)
    #image, label = train_loader.dataset.__getitem__(i)
    plt.imshow(image.squeeze().numpy())
    plt.axis('off')
    plt.title(label)
    plt.subplot(3, 4, 2*i+2)
    plt.imshow(image_perm.squeeze().numpy())
    plt.axis('off')
    plt.title(label)
```


3
3
5
5
3
3
9
9
7
7
6
6

In [15]:

```
accuracy_list = []

def scramble_train(epoch, model, perm=torch.arange(0, 784).long()):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        # send to device
        data, target = data.to(device), target.to(device)

        # permute pixels
        data = data.view(-1, 28*28)
        data = data[:, perm]
        data = data.view(-1, 1, 28, 28)

        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 100 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.f
ormat(
                epoch, batch_idx * len(data), len(train_loader.datase
t),
                100. * batch_idx / len(train_loader), loss.item()))

def scramble_test(model, perm=torch.arange(0, 784).long()):
    model.eval()
    test_loss = 0
    correct = 0
    for data, target in test_loader:
        # send to device
        data, target = data.to(device), target.to(device)

        # permute pixels
        data = data.view(-1, 28*28)
        data = data[:, perm]
        data = data.view(-1, 1, 28, 28)

        output = model(data)
        test_loss += F.nll_loss(output, target, reduction='sum').item
() # sum up batch loss
        pred = output.data.max(1, keepdim=True)[1] # get the index of
```

```
the max log-probability
            correct += pred.eq(target.data.view_as(pred)).cpu().sum().ite
m()

    test_loss /= len(test_loader.dataset)
    accuracy = 100. * correct / len(test_loader.dataset)
    accuracy_list.append(accuracy)
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0
f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        accuracy))
```

In [16]:

```
print("Training on ", device)
model_cnn_3 = CNN(input_size, output_size)
model_cnn_3.to(device)
optimizer = optim.SGD(model_cnn_3.parameters(), lr=0.01, momentum=
0.5)
print('Number of parameters: {}'.format(get_n_params(model_cnn_3)))

for epoch in range(0, 10):
    scramble_train(epoch, model_cnn_3, fixed_perm)
    scramble_test(model_cnn_3, fixed_perm)
```

```
Training on  cuda:0
Number of parameters: 66002
Train Epoch: 0 [0/60000 (0%)]	Loss: 2.311506
Train Epoch: 0 [6400/60000 (11%)]	Loss: 2.266764
Train Epoch: 0 [12800/60000 (21%)]	Loss: 2.184983
Train Epoch: 0 [19200/60000 (32%)]	Loss: 1.470032
Train Epoch: 0 [25600/60000 (43%)]	Loss: 0.816377
Train Epoch: 0 [32000/60000 (53%)]	Loss: 0.680872
Train Epoch: 0 [38400/60000 (64%)]	Loss: 0.736304
Train Epoch: 0 [44800/60000 (75%)]	Loss: 0.592037
Train Epoch: 0 [51200/60000 (85%)]	Loss: 0.344290
Train Epoch: 0 [57600/60000 (96%)]	Loss: 0.339610

Test set: Average loss: 0.3624, Accuracy: 8907/10000 (89%)

Train Epoch: 1 [0/60000 (0%)]	Loss: 0.551571
Train Epoch: 1 [6400/60000 (11%)]	Loss: 0.334081
Train Epoch: 1 [12800/60000 (21%)]	Loss: 0.285482
Train Epoch: 1 [19200/60000 (32%)]	Loss: 0.352685
Train Epoch: 1 [25600/60000 (43%)]	Loss: 0.188578
Train Epoch: 1 [32000/60000 (53%)]	Loss: 0.412186
Train Epoch: 1 [38400/60000 (64%)]	Loss: 0.207640
Train Epoch: 1 [44800/60000 (75%)]	Loss: 0.190874
Train Epoch: 1 [51200/60000 (85%)]	Loss: 0.406359
Train Epoch: 1 [57600/60000 (96%)]	Loss: 0.244149

Test set: Average loss: 0.2367, Accuracy: 9329/10000 (93%)

Train Epoch: 2 [0/60000 (0%)]	Loss: 0.319895
Train Epoch: 2 [6400/60000 (11%)]	Loss: 0.293301
Train Epoch: 2 [12800/60000 (21%)]	Loss: 0.217626
Train Epoch: 2 [19200/60000 (32%)]	Loss: 0.387113
Train Epoch: 2 [25600/60000 (43%)]	Loss: 0.077615
Train Epoch: 2 [32000/60000 (53%)]	Loss: 0.212487
Train Epoch: 2 [38400/60000 (64%)]	Loss: 0.193970
Train Epoch: 2 [44800/60000 (75%)]	Loss: 0.185607
Train Epoch: 2 [51200/60000 (85%)]	Loss: 0.168193
Train Epoch: 2 [57600/60000 (96%)]	Loss: 0.247540

Test set: Average loss: 0.1769, Accuracy: 9451/10000 (95%)

Train Epoch: 3 [0/60000 (0%)]	Loss: 0.068273
```

```
Train Epoch: 3 [6400/60000 (11%)]	Loss: 0.502430
Train Epoch: 3 [12800/60000 (21%)]	Loss: 0.084381
Train Epoch: 3 [19200/60000 (32%)]	Loss: 0.066023
Train Epoch: 3 [25600/60000 (43%)]	Loss: 0.059836
Train Epoch: 3 [32000/60000 (53%)]	Loss: 0.052095
Train Epoch: 3 [38400/60000 (64%)]	Loss: 0.194041
Train Epoch: 3 [44800/60000 (75%)]	Loss: 0.069777
Train Epoch: 3 [51200/60000 (85%)]	Loss: 0.226673
Train Epoch: 3 [57600/60000 (96%)]	Loss: 0.025219

Test set: Average loss: 0.1480, Accuracy: 9539/10000 (95%)

Train Epoch: 4 [0/60000 (0%)]	Loss: 0.096207
Train Epoch: 4 [6400/60000 (11%)]	Loss: 0.203058
Train Epoch: 4 [12800/60000 (21%)]	Loss: 0.065872
Train Epoch: 4 [19200/60000 (32%)]	Loss: 0.100554
Train Epoch: 4 [25600/60000 (43%)]	Loss: 0.132901
Train Epoch: 4 [32000/60000 (53%)]	Loss: 0.122994
Train Epoch: 4 [38400/60000 (64%)]	Loss: 0.166694
Train Epoch: 4 [44800/60000 (75%)]	Loss: 0.124768
Train Epoch: 4 [51200/60000 (85%)]	Loss: 0.061354
Train Epoch: 4 [57600/60000 (96%)]	Loss: 0.096026

Test set: Average loss: 0.1665, Accuracy: 9481/10000 (95%)

Train Epoch: 5 [0/60000 (0%)]	Loss: 0.138119
Train Epoch: 5 [6400/60000 (11%)]	Loss: 0.079209
Train Epoch: 5 [12800/60000 (21%)]	Loss: 0.029018
Train Epoch: 5 [19200/60000 (32%)]	Loss: 0.066754
Train Epoch: 5 [25600/60000 (43%)]	Loss: 0.019635
Train Epoch: 5 [32000/60000 (53%)]	Loss: 0.138224
Train Epoch: 5 [38400/60000 (64%)]	Loss: 0.135656
Train Epoch: 5 [44800/60000 (75%)]	Loss: 0.092600
Train Epoch: 5 [51200/60000 (85%)]	Loss: 0.080379
Train Epoch: 5 [57600/60000 (96%)]	Loss: 0.089517

Test set: Average loss: 0.1244, Accuracy: 9599/10000 (96%)

Train Epoch: 6 [0/60000 (0%)]	Loss: 0.171076
Train Epoch: 6 [6400/60000 (11%)]	Loss: 0.040048
Train Epoch: 6 [12800/60000 (21%)]	Loss: 0.072116
Train Epoch: 6 [19200/60000 (32%)]	Loss: 0.039751
Train Epoch: 6 [25600/60000 (43%)]	Loss: 0.020567
```

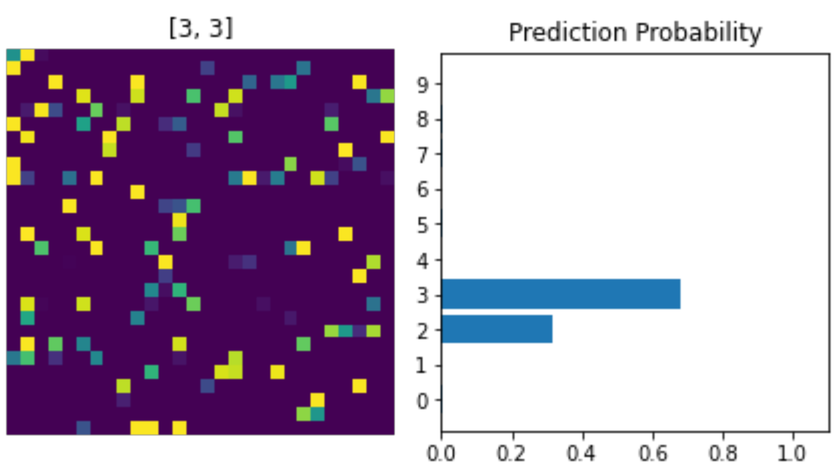
```
Train Epoch: 6 [32000/60000 (53%)]	Loss: 0.063507
Train Epoch: 6 [38400/60000 (64%)]	Loss: 0.133681
Train Epoch: 6 [44800/60000 (75%)]	Loss: 0.119641
Train Epoch: 6 [51200/60000 (85%)]	Loss: 0.037868
Train Epoch: 6 [57600/60000 (96%)]	Loss: 0.067937

Test set: Average loss: 0.1275, Accuracy: 9593/10000 (96%)

Train Epoch: 7 [0/60000 (0%)]	Loss: 0.205356
Train Epoch: 7 [6400/60000 (11%)]	Loss: 0.057234
Train Epoch: 7 [12800/60000 (21%)]	Loss: 0.060806
Train Epoch: 7 [19200/60000 (32%)]	Loss: 0.028336
Train Epoch: 7 [25600/60000 (43%)]	Loss: 0.077426
Train Epoch: 7 [32000/60000 (53%)]	Loss: 0.004114
Train Epoch: 7 [38400/60000 (64%)]	Loss: 0.079707
Train Epoch: 7 [44800/60000 (75%)]	Loss: 0.095361
Train Epoch: 7 [51200/60000 (85%)]	Loss: 0.015836
Train Epoch: 7 [57600/60000 (96%)]	Loss: 0.144897

Test set: Average loss: 0.1237, Accuracy: 9623/10000 (96%)

Train Epoch: 8 [0/60000 (0%)]	Loss: 0.016177
Train Epoch: 8 [6400/60000 (11%)]	Loss: 0.066097
Train Epoch: 8 [12800/60000 (21%)]	Loss: 0.024603
Train Epoch: 8 [19200/60000 (32%)]	Loss: 0.067242
Train Epoch: 8 [25600/60000 (43%)]	Loss: 0.014626
Train Epoch: 8 [32000/60000 (53%)]	Loss: 0.051929
Train Epoch: 8 [38400/60000 (64%)]	Loss: 0.050840
Train Epoch: 8 [44800/60000 (75%)]	Loss: 0.172051
Train Epoch: 8 [51200/60000 (85%)]	Loss: 0.023358
Train Epoch: 8 [57600/60000 (96%)]	Loss: 0.135856

Test set: Average loss: 0.1345, Accuracy: 9599/10000 (96%)

Train Epoch: 9 [0/60000 (0%)]	Loss: 0.088040
Train Epoch: 9 [6400/60000 (11%)]	Loss: 0.055426
Train Epoch: 9 [12800/60000 (21%)]	Loss: 0.038393
Train Epoch: 9 [19200/60000 (32%)]	Loss: 0.008204
Train Epoch: 9 [25600/60000 (43%)]	Loss: 0.041335
Train Epoch: 9 [32000/60000 (53%)]	Loss: 0.015474
Train Epoch: 9 [38400/60000 (64%)]	Loss: 0.071670
Train Epoch: 9 [44800/60000 (75%)]	Loss: 0.051915
Train Epoch: 9 [51200/60000 (85%)]	Loss: 0.054322
```

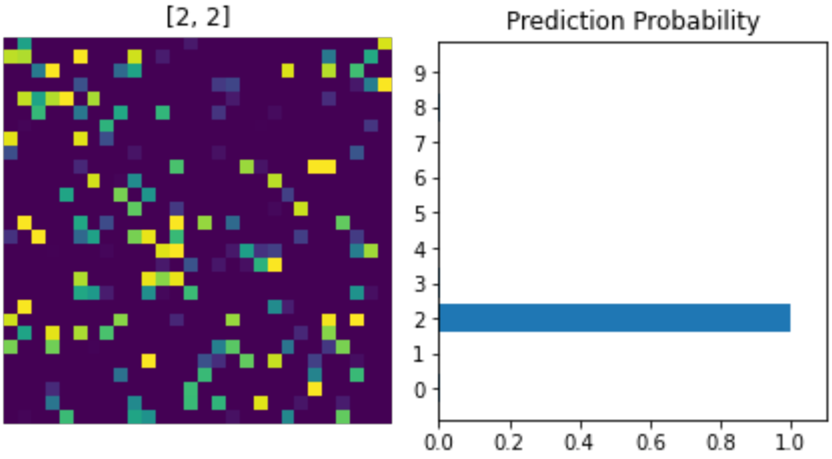
```
Train Epoch: 9 [57600/60000 (96%)]      Loss: 0.005000

Test set: Average loss: 0.1261, Accuracy: 9645/10000 (96%)
```

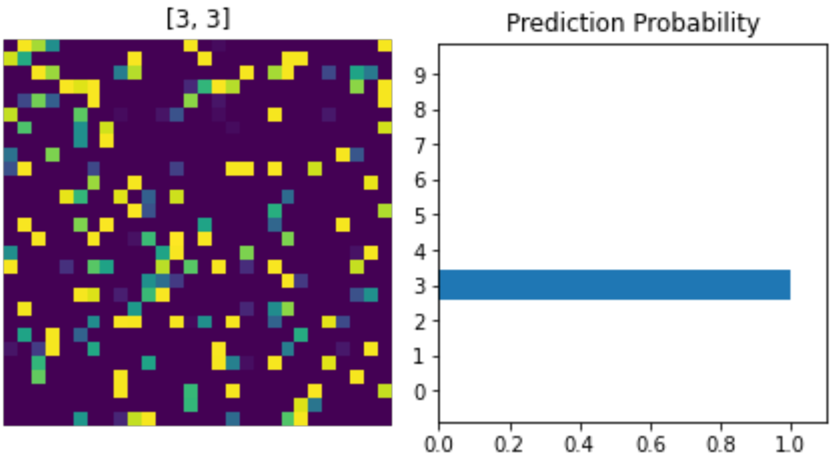
Performance decreased from 99% to 96%

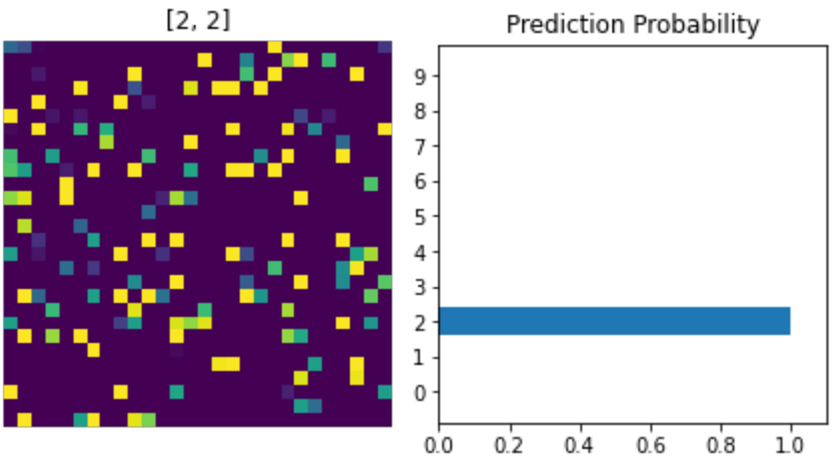
In [17]:

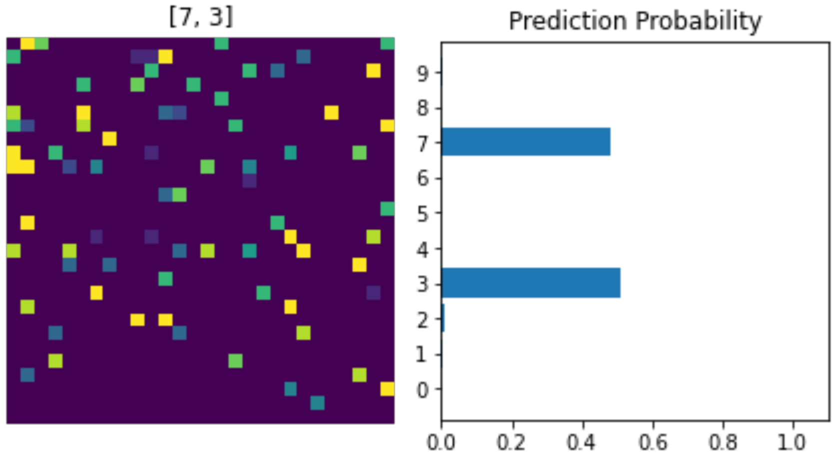
```python
model_cnn_3.to('cpu')

# fetch a batch of test images
image_batch, label_batch = next(iter(test_loader))
image_batch_scramble = image_batch.view(-1, 28*28)
image_batch_scramble = image_batch_scramble[:, fixed_perm]
image_batch_scramble = image_batch_scramble.view(-1, 1, 28, 28)
# Turn off gradients to speed up this part
with torch.no_grad():
    log_pred_prob_batch = model_cnn_3(image_batch_scramble)
for i in range(10):
    img = image_batch[i]
    img_perm = image_batch_scramble[i]
    real_label = label_batch[i].item()
    log_pred_prob = log_pred_prob_batch[i]
    # Output of the network are log-probabilities, need to take expone
ntial for probabilities
    pred_prob = torch.exp(log_pred_prob).data.numpy().squeeze()
    visualize_pred(img_perm, pred_prob, real_label)
```



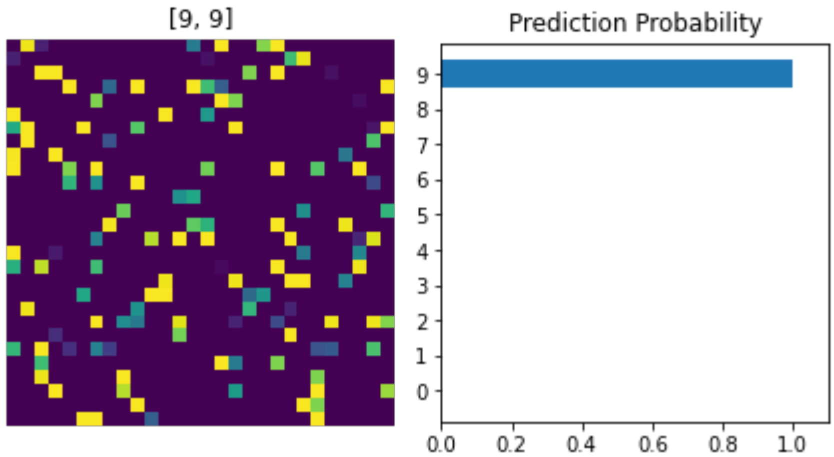

[2, 2]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0


[3, 3]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0

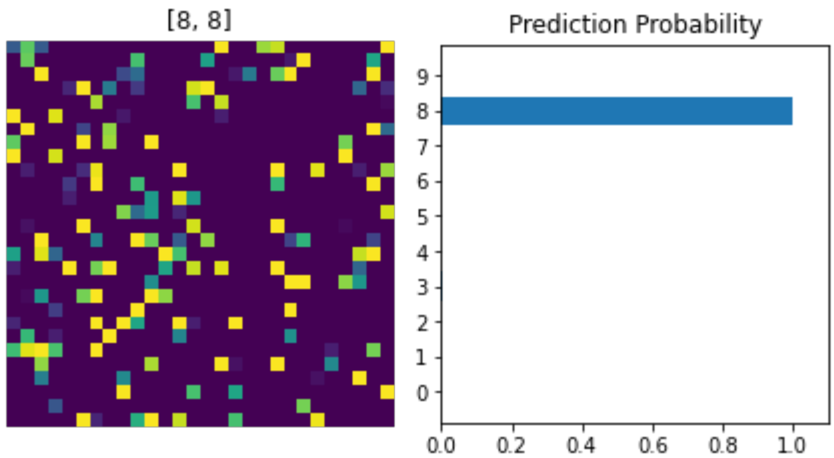

[2, 2]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0



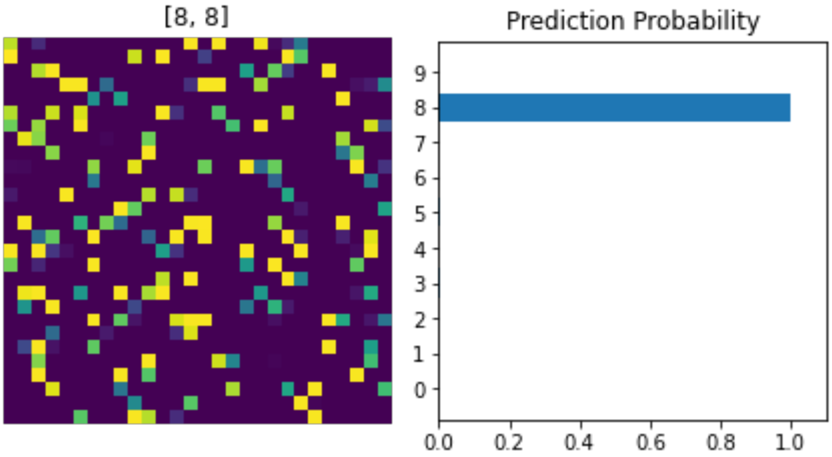
[7, 3]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0

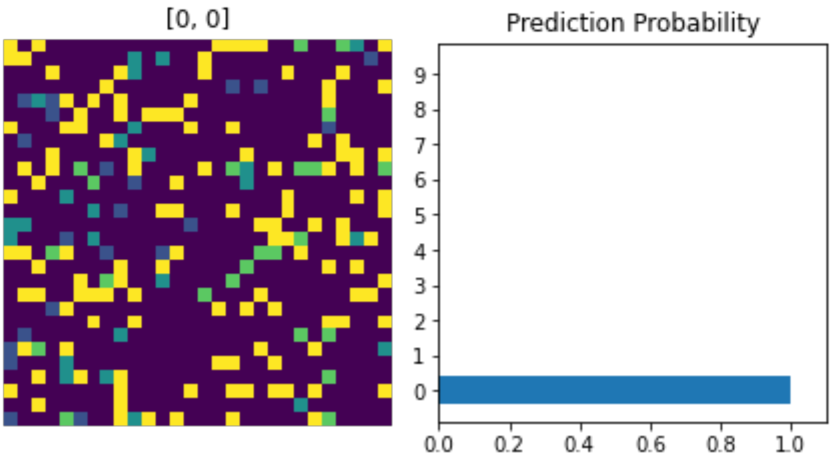


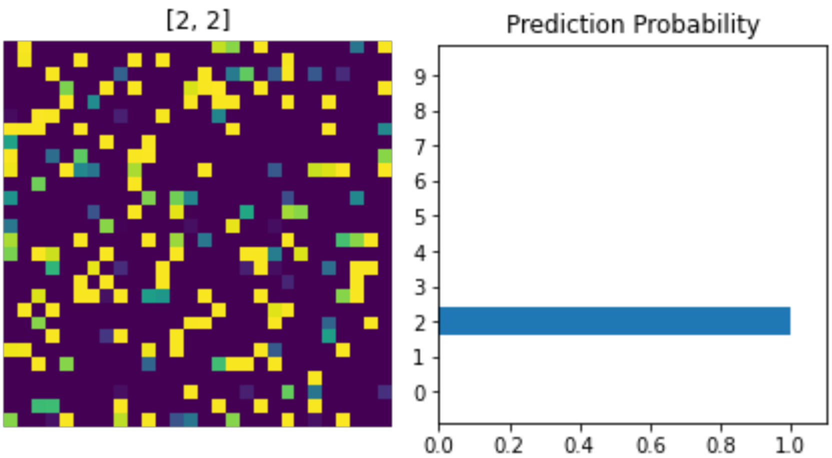
[9, 9]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0



[8, 8]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0


[8, 8]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0


[0, 0]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0


[2, 2]
Prediction Probability
9
8
7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0