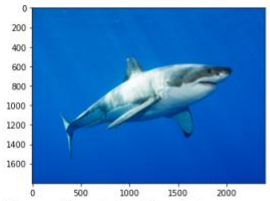
Brianna Solano
Project 1: Deep Learning for Recognition

1. Classify Image Net classes with ResNet50

Results:

Picture of a Great White Shark

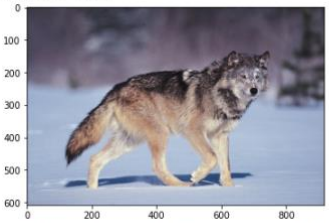


[('great white shark, white shark, man-eater, man-eating shark, Carcharodon carcharias', 84.42811584472656)]

Picture of a Great White Shark

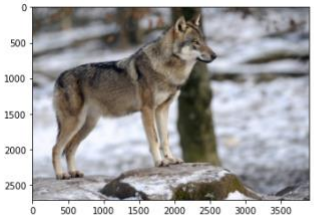
[('great white shark, white shark, man-eater, man-eating shark, Carcharodon carcharias', 72.13719177246094)]

Picture of a Wolf



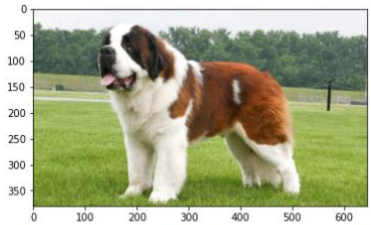
[('timber wolf, grey wolf, gray wolf, Canis lupus', 77.01177978515625)]

Picture of a Wolf



[('timber wolf, grey wolf, gray wolf, Canis lupus', 91.45182037353516)]

Picture of a Saint Bernard

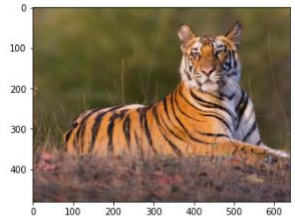


[('Saint Bernard, St Bernard', 99.8226547241211)]

Picture of a Saint Bernard

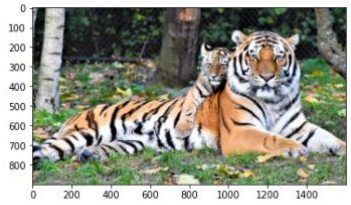
[('Saint Bernard, St Bernard', 92.01873016357422)]

Picture of a Tiger



[('tiger, Panthera tigris', 75.94995880126953)]

Picture of a Tiger



[('tiger, Panthera tigris', 75.94995880126953)]

Picture of an Artic Fox

[('Arctic fox, white fox, Alopex lagopus', 98.02201080322266)]

Picture of an Artic Fox

[('Arctic fox, white fox, Alopex lagopus', 98.95771789550781)]

Confusion matrix:

	Shark	Tiger	Saint Bernard	Wolf	Arctic Fox
Shark	2	0	0	0	0
Tiger	0	2	0	0	0
Saint Bernard	0	0	2	0	0
Wolf	0	0	0	2	0
Arctic Fox	0	0	0	0	2

Accuracy:

Out of the 10 images the program was able to output the correct label for all 10 images.

$$N = \#\ trials$$
$$S = of\ correct\ outputs$$
$$\frac{S}{N} = \ accuracy$$

$$\frac{10\ correct\ outputs}{10\ trails, 1\ for\ each\ image} = 100\%$$

Therefore the program had 100% accuracy

F-score:

$$P = \ precision$$
$$R = \ recall$$
$$F = \frac{2PR}{P+R}$$
$$F = \frac{2(1)(1)}{1+1} = \frac{2}{2} = 1$$

Precision:

True = Outputted the correct label

False = Outputted the wrong label

FP signifies that the program alerted True when it outputted the wrong label and TP signifies that the program alerted True when it outputted the correct label. The program accumulated a total of 10 TP and 0 FP.

$$FP = \#\ of\ false\ postive$$
$$TP = \#\ of\ true\ postives$$
$$Precision = \frac{TP}{TP+FP}$$
$$Precision = \frac{10}{10+0} = 100\%$$

Recall:

True = Outputted the correct label

False = Outputted the wrong label

FN signifies that the program alerted False when it outputted the correct label and TP signifies that the program alerted True when it outputted the correct label. The program accumulated a total of 10 TP and 0 FN.

$$FN = \#\ of\ false\ negatives$$
$$TP = \#\ of\ true\ postives$$

$$Recall = \frac{TP}{FN + TP}$$
$$Recall = \frac{10}{0 + 10} = 100\%$$

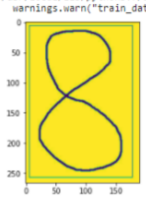
ResNet:

ResNet stands for residual network and is a type of neural network. It consists of residual blocks and varying layers. The layers feed into each other and typically consist of a convolution layer followed by a batch normalization layer and rectified linear unit. We are presented with the problem of vanishing gradient which occurs in the training process of deep neural networks. ResNet alleviates this issue though it's implemented of skipped network which allow it to skip some layers. These features allow ResNet to incorporate more layers while also minimizing the error percentage. ResNet is pre-trained on the ImageNet library. With a top-5 error rate of 3.57% in the ILSVRS 2015 classification completion, we can see why ResNet has become one of the most popular architectures in computer vision.

2. Classify MNIST classes with ResNet18

Results:

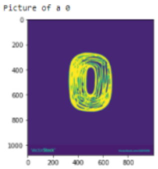
```
Picture of a 8
/usr/local/lib/python3.7/dist-packages/torchvision/datasets/mnist.py:62: UserWarning: train_data has been renamed data
  warnings.warn("train_data has been renamed data")
```



```
[('8', 76.69300079345703),
 ('0', 11.884601593017578),
 ('1', 5.898499011993408),
 ('5', 3.1556930541992188),
 ('3', 1.1414660215377808),
 ('4', 0.4879510998725891),
 ('6', 0.3912808895111084),
 ('9', 0.22406747937202454),
 ('7', 0.0689193457365036),
 ('2', 0.0545201301574707)]
```

```
print("Picture of a 0")
response3 = requests.get("https://cdn2.vectorstock.com/i/1000x1000/05/56/number-zero-hand-drawn-chalk-on-a-blackboard-vector-32870556.jpg")
img3 = Image.open(BytesIO(response3.content)).convert('L')
img3.resize((256,256))
imgplot = plt.imshow(img3)
plt.show()
img3 = data_transform(img3)
batch3 = torch.unsqueeze(img3,0)
batch3= batch3.cuda()
Answer3 = model(batch3)
_, indices = torch.sort(Answer3,descending = True)
precentage = torch.nn.functional.softmax(Answer3,dim=1)[0]*100
[(classes[idx],precentage[idx].item()) for idx in indices[0][:10]]
```

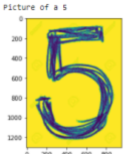
```
Picture of a 0
```



```
[('0', 89.83409118652344),
 ('9', 8.364124298095703),
 ('6', 1.0459314584732056),
 ('8', 0.6203619837760925),
 ('2', 0.10279686748981476),
 ('5', 0.0119206877425313),
 ('4', 0.009508761577308178),
 ('1', 0.009026501327753067),
 ('3', 0.0011245725909247994),
 ('7', 0.0011079489486292005)]
```

```
print("Picture of a 5")
response = requests.get("https://previews.123rf.com/images/aroas/aroas1704/aroas170400061/79321951-handwritten-sketch-black-number-5-on-whi
img = Image.open(BytesIO(response.content)).convert('L')
img.resize((256,256))
imgplot = plt.imshow(img)
plt.show()
img6 = data_transform(img)
batch6 = torch.unsqueeze(img6,0)
batch6= batch6.cuda()
Answer6 = model(batch6)
_, indices = torch.sort(Answer6,descending = True)
precentage = torch.nn.functional.softmax(Answer6,dim=1)[0]*100
[(classes[idx],precentage[idx].item()) for idx in indices[0][:10]]
```

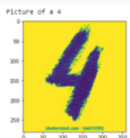
```
Picture of a 5
```



```
[('5', 71.8088607788086),
 ('8', 25.08469009399414),
 ('3', 1.2320839166641235),
 ('6', 0.778889298439059),
 ('1', 0.5166008472442627),
 ('0', 0.5093263983726501),
 ('4', 0.06008698418736458),
 ('2', 0.008939065039157867),
 ('7', 0.0004052283766213804),
 ('9', 0.00011547817848622799)]
```

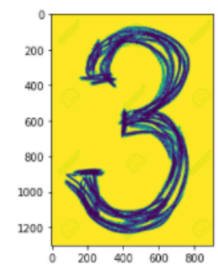
```
print("Picture of a 4")
response = requests.get("https://image.shutterstock.com/image-illustration/4-black-handwritten-number-on-260nw-166572392.jpg")
img = Image.open(BytesIO(response.content)).convert('L')
img.resize((256,256))
imgplot = plt.imshow(img)
plt.show()
img5 = data_transform(img)
batch5 = torch.unsqueeze(img5,0)
batch5= batch5.cuda()
Answer5 = model(batch5)
_, indices = torch.sort(Answer5,descending = True)
precentage = torch.nn.functional.softmax(Answer5,dim=1)[0]*100
[(classes[idx],precentage[idx].item()) for idx in indices[0][:10]]
```

```
Picture of a 4
```



```
[('4', 75.34496307373047),
 ('1', 22.196889877319336),
 ('3', 0.6111820936203003),
 ('7', 0.4874011576175697),
 ('0', 0.4510076344013214),
 ('6', 0.2350267469882965),
 ('8', 0.21385625004768372),
 ('2', 0.193826824426651),
 ('9', 0.1399412602186203),
 ('5', 0.12590935826301575)]
```

```
Picture of a 3
```



```
[('8', 92.86206817626953),
 ('3', 3.606292724609375),
 ('0', 1.716536521911621),
 ('6', 1.0797592401504517),
 ('5', 0.6877622008323669),
 ('1', 0.03991674259305),
 ('2', 0.0029790354892611504),
 ('4', 0.0028734118677675724),
 ('9', 0.001759019447490573),
 ('7', 4.138391886954196e-05)]
```

Confusion matrix:

	0	1	2	3	4	5	6	7	8	9
0	980	0	0	0	0	0	0	0	0	0
1	0	1134	0	0	0	0	0	1	0	0
2	0	0	1028	0	1	0	0	2	1	0
3	0	0	0	1007	0	1	0	1	0	1
4	0	0	0	0	973	0	0	0	0	9
5	0	0	0	3	0	887	1	0	0	1
6	2	4	0	0	1	0	949	0	2	0
7	0	2	2	0	1	0	0	1023	0	0
8	0	1	0	1	0	1	0	1	969	1
9	0	0	0	0	3	0	0	1	0	1005

Accuracy:

Out of the 10 images the program was able to output the correct label for all 10 images.

$$N = \#\ trials$$
$$S = of\ correct\ outputs$$
$$\frac{S}{N} = accuracy$$

$$\frac{9955\ correct\ outputs}{10{,}000\ trails} = 99.55\%$$

Therefore the program had 100% accuracy

F-score, Precision, and Recall

When calculating the precision, precision, and F-score through code we get an average of 98% for each category

Results and Methods

When constructing the code, I first imported the necessary libraries such that I was able to use PyTorch, have the ability to import the MNIST data and obtain google images to conduct my own tests. Then I decided to define important variables such as the classes which are 0 through 9, epochs, and batch size. Once that was done, I made sure to download the training and validation, images and labels from the MNIST database. These images were then transform to the required channel, size, and into a tensor to operate on RESNET. Since MNIST images are in greyscale therefore containing 1 channel, we had to transform these images into 3 channels such that RESNET would be able to interpret them. Once data was downloaded and we needed to define our pre-trained model which is ResNet18. Since MNIST only contains 10 classes, unlike ImageNet, we had to manually change the model's number of classes. Since, these computations are intense rather then using the CPU, we transfer the model onto the GPU. We define our loss function as CrossEntropyLoss and our optimizer.

Once everything is defined we move onto training and validating our model. For each epoch, we would conduct our training. For each epoch we would use the training data that we downloaded and pass it through the model. We would then compute the loss from the predicted output to the label. We would then compute Tensor.backward to compute the gradient and update the parameters using optimizer.step.

After training is complete for the epoch, we would then processed to the validation phase which required the model to be sent to eval(). The process of training and validating would be conducted on every epoch. When the complete process was over, images were taken from google which represented numbers from 0 to 9 and be evaluated on this fine tuned model. As a result, the product wasn't prefect but was able to identity 7 out of the 10 images that were presented. 8 was predicted correctly with a

97%, while 3 was predicted to be an 8 with a rate of 69%. When the model would output a prediction and it resulted to be wrong, usually the correct prediction would appear to be in 2nd or 3rd place. Such as in the example where it predicted a 3 to be an 8. Out of the list of predictions, 3 was second on the list. In certain cases, you could see the reason why the computer would interpret a 3 to be an 8. The 3 could be seen as an 8 since the gapes that form the 3 were small.