CS 7150: Deep Learning — Spring 2021— Paul Hand

HW 1

Due: Friday February 12, 2020 at 5:00 PM Eastern time via Gradescope

Names: [Gourang Patel, Sonal Jain]

You will submit this homework in groups of 2. You may consult any and all resources. Note that some of these questions are somewhat vague by design. Part of your task is to make reasonable decisions in interpreting the questions. Your responses should convey understanding, be written with an appropriate amount of precision, and be succinct. Where possible, you should make precise statements. For questions that require coding, you may either type your results with figures into this tex file, or you may append a pdf of output of a Jupyter notebook that is organized similarly. You may use code available on the internet as a starting point.

Question 1. *Is a 1×1 convolution operation the same as scaling the input by a single scalar constant? Explain. If the answer is sometimes yes, then make sure to explain when it is and when it isn't.*

Response:

Yes, but only sometimes. We can claim that 1×1 convolution is same as scaling the input by a single scalar constant . For an instance in the following figure below, we can see the convolution operation with single channel input and multi channel input.

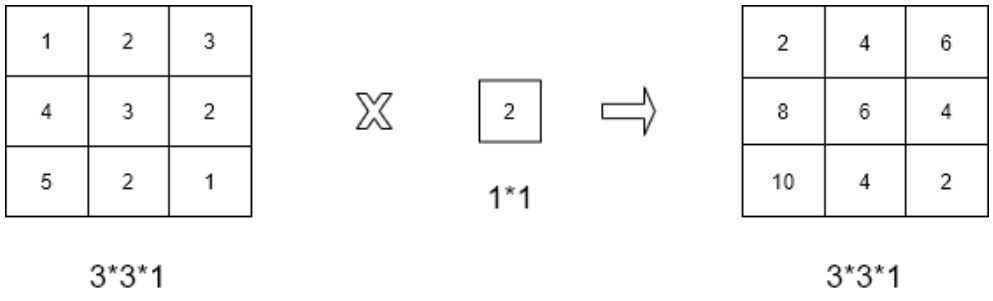


Fig 1(a) convolution operation with 1*1 filter with 1 channel

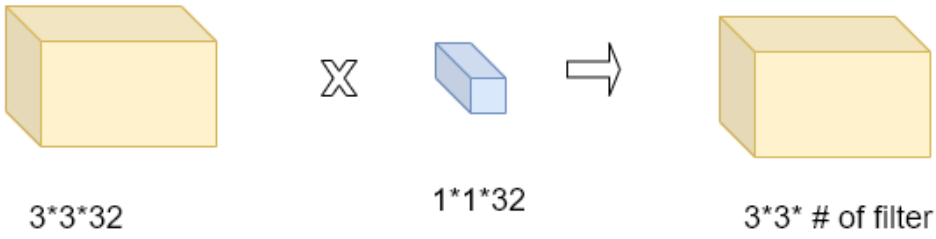


Fig 1(b) convolution operation with 1*1 filter with 32 channel

Figure 1: Convolution operation with 1*1 filter

In this figure 1(a) we can observe that when we use a single channel image with some width and height and convolute it with a 1*1 filter, it computes the dot product and scales the input by

the value.
In figure 1(b), we have multiple channels and in this case it will be different. Here, 1x1 convolution could be used to alter the depth of the input volume. As mentioned in the example about with 32 channels when we perform 1*1 convolution, our filter will convolve each channel in the input image and then a non-linearity is applied so we get the altered number of channels in the output. Thus, we can claim that with multiple channels, 1*1 filter is not scaling the inputs rather it aids to alter the number of channels of the input space. If we take reduced number of filter instead of 32 with 1*1 convolution we'll be able to reduce the number of channels after convolution.

Question 2. *In this problem, you will train a classifier on the MNIST dataset. You can find this dataset in TorchVision. Train a fully-connected neural network with 2 hidden layers and ReLU activations.*

(a) Clearly convey the architecture (by providing a figure) and training procedure you used.

Response:

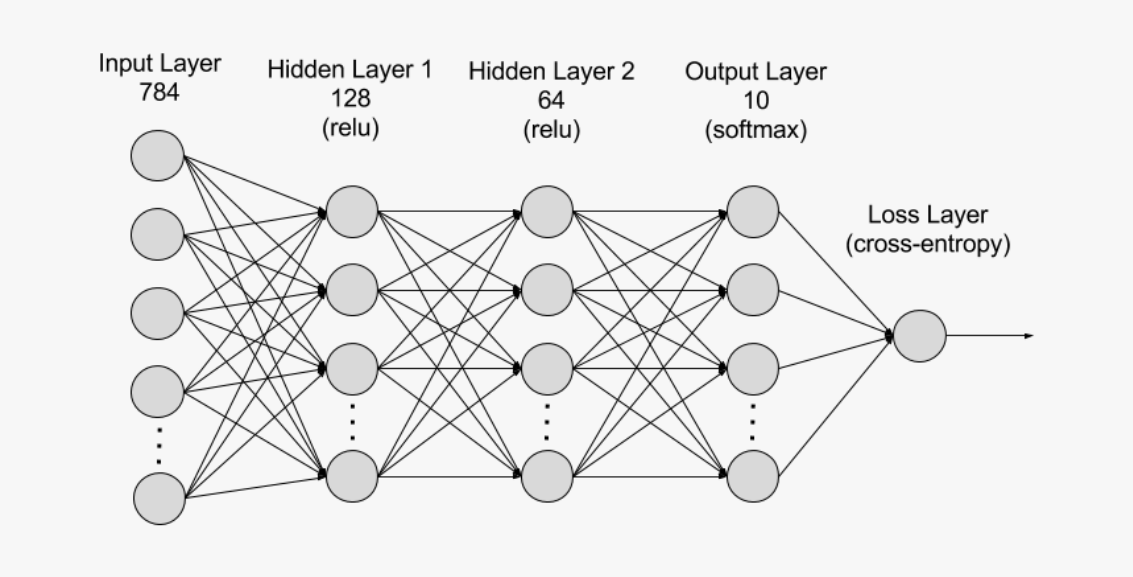


Figure 2: Architechture of fully connected two layer neural network

Training procedure:

In this particular problem we have created a 2 layer fully connected neural network with 10 classes at the output layer. We are training this neural network on MNIST data which is a very famous dataset for hand writing recognition.

The training procedure includes:

1. Load Data: MNSIT data-set is downloaded from Torchvision Library and splitted into train and test dataset.

2. Model Definition: In this step, we defined the necessary parameters such as input size, hidden layer size, batch size, number of epochs, learning rate and output classes. After

this, the train and test data was divided into batches and we built the neural network. The Neural network consist of input layer, 2 fully connected hidden layer with ReLu activation in between and a output layer. We chose Stochastic gradient descent as optimizer and Cross entropy loss function.

```
<bound method Module.parameters of NeuralNet(
  (fc1): Linear(in_features=784, out_features=128, bias=True)
  (relu): ReLU()
  (fc2): Linear(in_features=128, out_features=64, bias=True)
  (relu_2): ReLU()
  (fc3): Linear(in_features=64, out_features=10, bias=True)
)>
```

Figure 3: Model parameters

3. Training Phase: We start with the forward propagation defined by forward function (forward(self, x)) in the code where the input data passes through the network in such a way that all the neurons apply their transformation to the information they receive from the neurons of the previous layer and sending it to the neurons of the next layer. When the data passes through all the neurons, final layer will have the results of label prediction.

We will use a loss function to estimate the loss and compare the true and predicted labels. Once loss is calculated (criterion(output, labels)), information is propagated back, this is called backward propagation defined by loss.backward() in the code. Starting from the output layer, that loss information propagates to all the neurons in the hidden layer that contribute directly to the output.

We then adjust the weights of connections between the neurons to make this loss as close to zero. We have used Stochastic gradient descent for this in the code denoted by optim.SGD(). This is done in batches of data in the successive iterations (epochs) of all the dataset that we pass to the net work in each iteration.

4. We stored the loss and accuracy while training for the number of iterations and saved the model with the best accuracy.

After performing hyperparameter tuning we obtained the best train and test accuracy for:

$$INPUTSIZE = 784$$

$$HIDDENSIZE_1 = 128$$

$$HIDDENSIZE_2 = 64$$

$$BATCHSIZE = 60$$

$$NUMEPOCHS = 10$$

$$LEARNINGRATE = 0.005$$

$$Trainloss : 0.29$$

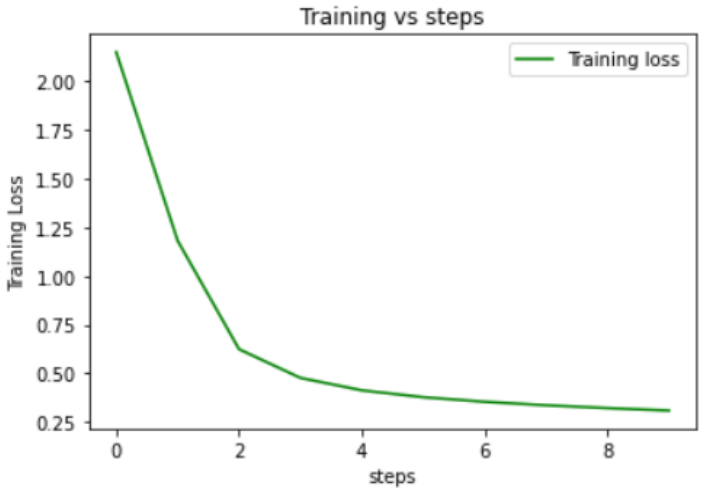


Figure 4: Training loss plot

The best train and test accuracy we got:

$$Trainaccuracy : 82$$

Figure 5: Training Accuracy plot

$$Testaccuracy : 86.26$$

(b) Plot training loss, training accuracy, and test accuracy vs. iteration count.

Response:

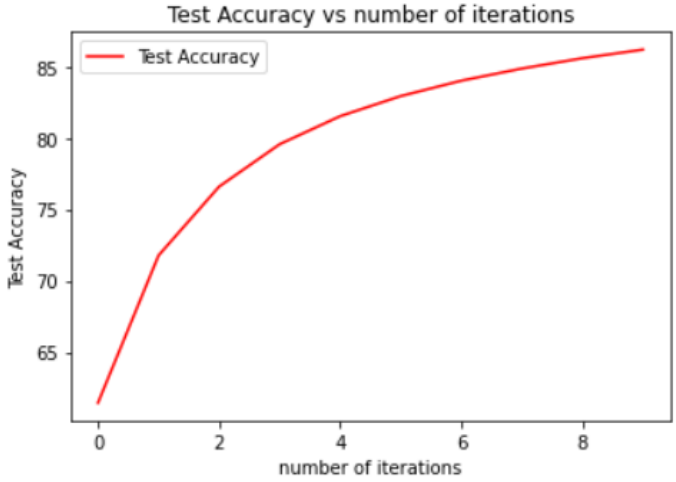


Figure 6: Test Accuracy plot

Question 3. *Download an AlexNet and a ResNet101 model that have been trained on ImageNet. You can find these models in TorchVision. Using a camera, take a picture of an object that belongs to one of the ImageNet classes. Use both models to classify the image. Output the top 5 predicted classes and their corresponding probabilities, according to each model. Were the models correct?*

Response:

We provided a image with a dog which was captured by us. The results of both models seems really good to us. As we observe the ouptuts of the models we can observe that both the models predicted the breed of the dog in the image. From the results of ResNet101 model, top 5 classes along with its probabilities(from high to low) were Pomerian, Pembroke, White Wolf, Chow Chow, Samoyed . From the AlexNet model, they are Pomerian, Sampoyed, Malamute, Chow Chow, Eskimo dog (husky), which are all included as well. It's surprising interesting that both the models were able to classify the dog with it's right breed that was Pomerian, however Resnet score for Pomerian was 61.12% which was more confident than the AlexNet which was 29.86%.

Results from the models –

Resnet 101

TOP 5 predictions from Resnet 101 :

[('259: 'Pomeranian',', 61.12522506713867),
('263: 'Pembroke, Pembroke Welsh corgi',', 19.54987907409668),
('270: 'white wolf, Arctic wolf, Canis lupus tundrarum',', 2.1717541217803955),
('260: 'chow, chow chow',', 2.150696039199829),
('258: 'Samoyed, Samoyede',', 1.5832675695419312)]

ALexnet Model

TOP 5 predictions from AlexNet :

Figure 7: Object Image

[(”259: ’Pomeranian’,”, 29.866588592529297),
(”258: ’Samoyed, Samoyede’,”, 21.46195411682129),
(”249: ’malamute, malemute, Alaskan malamute’,”, 13.845385551452637),
(”260: ’chow, chow chow’,”, 8.142885208129883),
(”248: ’Eskimo dog, husky’,”, 3.4929816722869873)]

Question 2. In this problem, you will train a classifier on the MNIST dataset.You can find this dataset in TorchVision. Train a fully-connected neural network with 2 hidden layers and ReLU activations.

```
import torchvision
import os


import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.autograd import Variable
from torchvision import transforms
import torch.utils.data as Data
import pandas as pd
import matplotlib.pyplot as plt
from pandas import DataFrame


if not(os.path.exists('./mnist/')) or not os.listdir('./mnist/'):
    DOWNLOAD_MNIST = True

train_data = torchvision.datasets.MNIST(
    root='./mnist/',
    train=True,
    transform=torchvision.transforms.ToTensor(),
    download=True,
)

test_data = torchvision.datasets.MNIST(
    root='./mnist/',
    train=False,
    transform=torchvision.transforms.ToTensor(),
    download=True,
)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./mnist/MNIST/raw/train-images-idx3-ubyte.gz
9920512/? [00:02<00:00, 3731498.34it/s]
Extracting ./mnist/MNIST/raw/train-images-idx3-ubyte.gz to ./mnist/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./mnist/MNIST/raw/train-labels-idx1-ubyte.gz
32768/? [00:02<00:00, 16019.58it/s]
Extracting ./mnist/MNIST/raw/train-labels-idx1-ubyte.gz to ./mnist/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./mnist/MNIST/raw/t10k-images-idx3-ubyte.gz
1%  16384/1648877 [00:00<00:10, 150317.89it/s]
Extracting ./mnist/MNIST/raw/t10k-images-idx3-ubyte.gz to ./mnist/MNIST/raw
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./mnist/MNIST/raw/t10k-labels-idx1-ubyte.gz
0/? [00:00<?, ?it/s]
Extracting ./mnist/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./mnist/MNIST/raw
Processing...
Done!
/usr/local/lib/python3.6/dist-packages/torchvision/datasets/mnist.py:480: UserWarning: The given NumPy array is not writeable, and PyTorch does not supp
  return torch.from_numpy(parsed.astype(m[2], copy=False)).view(*s)
```

```
print("Training Data Size: {}".format(train_data.data.size()))
print("Training Data Labels Size : {}".format(train_data.targets.size()))
plt.imshow(train_data.data[0].numpy(), cmap = 'gray')
plt.title(train_data.targets[0])
plt.show()
```

```
Training Data Size: torch.Size([60000, 28, 28])
Training Data Labels Size : torch.Size([60000])
```

```
## CNN with two hidden layers and Relu Activation for MNIST
## training parameters
INPUT_SIZE = 784
HIDDEN_SIZE_1 = 128
HIDDEN_SIZE_2 = 64
BATCH_SIZE = 60
NUM_EPOCHS = 10
LEARNING_R = 0.005
DROPOUT_P = 0.5
NUM_CLASSES = 10

seed = 1234
torch.manual_seed(seed)
```

```
<torch._C.Generator at 0x7fc2ba2cbb70>
```

```
train_data_loader = Data.DataLoader(
    dataset = train_data,
    batch_size = BATCH_SIZE,
    shuffle = True
    )
test_data_loader = Data.DataLoader(
    dataset = test_data,
    batch_size = BATCH_SIZE,
    shuffle = True
)
```

```
## Visualizing the test data
batch = next(iter(test_data_loader))
samples = batch[0][:5]
y_true = batch[1]

for i,sample in enumerate(samples):
  plt.subplot(1,5,i+1)
  plt.title("Number %i" %y_true[i])
  plt.imshow(sample.numpy().reshape((28,28)), cmap = 'gray')
  plt.axis("off")
```


```
## CNN Model with two fully connected Hidden Layers

class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size1,hidden_size2, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, HIDDEN_SIZE_1)
        self.fc2 = nn.Linear(HIDDEN_SIZE_1, HIDDEN_SIZE_2)
        self.fc3 = nn.Linear(HIDDEN_SIZE_2, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = F.relu(out)
        out = self.fc2(out)
        out = F.relu(out)
        out = self.fc3(out)
        return out
```

```
model = NeuralNet(INPUT_SIZE, HIDDEN_SIZE_1, HIDDEN_SIZE_2, NUM_CLASSES)
# Adam Optimizer
optimizer = optim.SGD(model.parameters(), lr = LEARNING_R)
# Categorical Test and check class is it
criterion = torch.nn.CrossEntropyLoss()
print(model.parameters)
```

```
<bound method Module.parameters of NeuralNet(
  (fc1): Linear(in_features=784, out_features=128, bias=True)
```

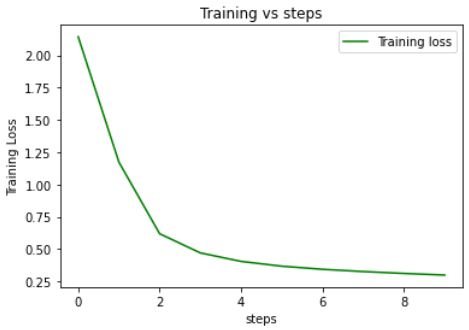
```
    (fc2): Linear(in_features=128, out_features=64, bias=True)
    (fc3): Linear(in_features=64, out_features=10, bias=True)
  )>
```

```
# CNN Model Training

loss_list = []
accuracy = []
iterations = []
running_loss = 0.0
total_train = 0.0

total_test = 0.0

test_accuracy_list  = []
correct_test = 0.0
correct_train = 0.0

for epoch in range(10):
  for i, (image, labels) in enumerate(train_data_loader):
    # print(i)

    train = image.reshape(-1, 28*28)
    labels = labels


    output = model(train)
    loss = criterion(output, labels)
    optimizer.zero_grad()

    loss.backward()
    optimizer.step()

    # print statistics
    running_loss += loss.item()
    _, predicted = torch.max(output.data,1)
    total_train += labels.size(0)
    correct_train += predicted.eq(labels.data).sum().item()
    train_accuracy = 100 * correct_train / total_train


  for i, (image, labels) in enumerate(test_data_loader):
      images = image.reshape(-1, 28*28)
      output = model(images)
      _, pred = torch.max(output,1)
      total_test += labels.size(0)
      correct_test += pred.eq(labels.data).sum().item()
  test_accuracy = 100 * correct_test / total_test


  print (f"\nEpoch {epoch+1}\nTraining Loss: {running_loss/1000}\nTraining Accuracy {train_accuracy}\nTesting Accuracy {test_accuracy}")

  # # print('[%d, %5d] Training loss: %.3f Training Accuracy: %.3f' %
  #       (epoch + 1, i + 1, running_loss/1000, train_accuracy))

  loss_list.append(running_loss/1000)
  accuracy.append(train_accuracy)
  test_accuracy_list.append(test_accuracy)
  running_loss = 0.0
```

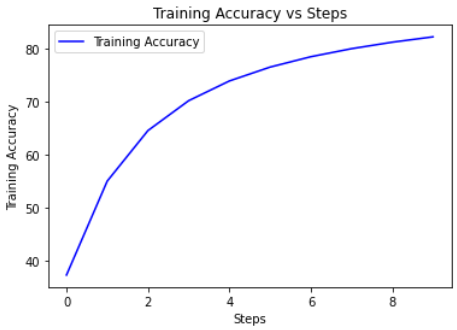
```
Epoch 1
Training Loss: 2.142834052801132
Training Accuracy 37.37833333333333
Testing Accuracy 61.44

Epoch 2
Training Loss: 1.1729536954164506
Training Accuracy 55.08583333333333
Testing Accuracy 71.355

Epoch 3
Training Loss: 0.6188344047665596
Training Accuracy 64.62333333333333
Testing Accuracy 76.30333333333333

Epoch 4
Training Loss: 0.4706866837888956
Training Accuracy 70.26125
Testing Accuracy 79.3125

Epoch 5
```

```
Training Loss: 0.40484689649939537
Training Accuracy 73.98
Testing Accuracy 81.364

Epoch 6
Training Loss: 0.3678727466464043
Training Accuracy 76.59722222222223
Testing Accuracy 82.85

Epoch 7
Training Loss: 0.3438453508019447
Training Accuracy 78.55285714285715
Testing Accuracy 83.98428571428572

Epoch 8
Training Loss: 0.32627104545384644
Training Accuracy 80.06958333333333
Testing Accuracy 84.88875

Epoch 9
Training Loss: 0.3117251776307821
Training Accuracy 81.29333333333334
Testing Accuracy 85.65

Epoch 10
Training Loss: 0.2992381045296788
Training Accuracy 82.3005
Testing Accuracy 86.267
```

```
plt.plot(np.array(loss_list), 'g', label='Training loss')
plt.title('Training vs steps')
plt.xlabel('steps')
plt.ylabel('Training Loss')
plt.legend()
plt.show()
```

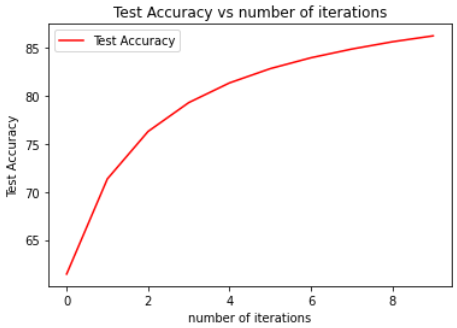



```
plt.plot( np.array(accuracy), 'b', label='Training Accuracy')
plt.title('Training Accuracy vs Steps')
plt.xlabel('Steps')
plt.ylabel('Training Accuracy')
plt.legend()
plt.show()
```




```
PATH = './mnist_net.pth'
torch.save(model.state_dict(), PATH)
```

```
plt.plot(np.array(test_accuracy_list), 'r', label='Test Accuracy')
plt.title('Test Accuracy vs number of iterations')
plt.xlabel('number of iterations')
plt.ylabel('Test Accuracy')
plt.legend()
plt.show()
```



Question 3

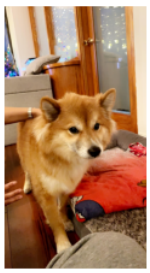
Part a

```
model_resnet = torch.hub.load('pytorch/vision:v0.6.0', 'resnet101', pretrained=True)
```

```
Downloading: "https://github.com/pytorch/vision/archive/v0.6.0.zip" to /root/.cache/torch/hub/v0.6.0.zip
Downloading: "https://download.pytorch.org/models/resnet101-5d3b4d8f.pth" to /root/.cache/torch/hub/checkpoints/resnet101-5d3b4d8f.pth
100%                                  170M/170M [03:28<00:00, 858kB/s]
```

```
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor()

])

from PIL import Image
img = Image.open('/content/dog.jpg')

plt.imshow(img)
plt.axis("off")
```

```
(-0.5, 1187.5, 2207.5, -0.5)
```



```
transform_img = transform(img)
batch_t = torch.unsqueeze(transform_img, 0)


def predict_class(output):
  with open('/content/imagenet1000_clsidx_to_labels.txt') as f:
    classes = [line.strip() for line in f.readlines()]
  _, index = torch.max(output, 1)

  percentage = torch.nn.functional.softmax(output, dim = 1)[0] * 100
  print("TOP 1 predictions {} and probability of : {}".format(classes[index[0]], percentage[index[0].item()]))
  # Finding index where we get the TOP 5 maximum score
  _, predicted = torch.sort(output, descending= True)
  print([(classes[idx], percentage[idx].item()) for idx in predicted[0][:5]])


model_resnet.eval()
output_resnet = model_resnet(batch_t)
predict_class(output_resnet)
```

```
TOP 1 predictions 259: 'Pomeranian', and probability of : 61.12522506713867
[("259: 'Pomeranian',", 61.12522506713867), ("263: 'Pembroke, Pembroke Welsh corgi',", 19.54987907409668), ("270: 'white wolf, Arctic wolf, Canis lupus
```

```
alexnet = torch.hub.load('pytorch/vision:v0.6.0', 'alexnet', pretrained=True)
```

```
Using cache found in /root/.cache/torch/hub/pytorch_vision_v0.6.0
Downloading: "https://download.pytorch.org/models/alexnet-owt-4df8aa71.pth" to /root/.cache/torch/hub/checkpoints/alexnet-owt-4df8aa71.pth
100%  233M/233M [00:03<00:00, 77.6MB/s]
```

```
alexnet.eval()
output_alexnet = alexnet(batch_t)
predict_class(output_alexnet)
```

```
TOP 1 predictions 259: 'Pomeranian', and probability of : 29.866588592529297
[("259: 'Pomeranian',", 29.866588592529297), ("258: 'Samoyed, Samoyede',", 21.46195411682129), ("249: 'malamute, malemute, Alaskan malamute',", 13.84538
```