Wensong Hu
EECS
University of Michigan

March 19, 2024

HW4 — Machine Learning

1 Section 1

1.1 Task 1: Implementing Computational Graphs

Code submitted to canvas.

```python
import math


"""
Defines forward and backward passes through different computational graphs.

Students should complete the implementation of all functions in this file.
"""

# Optional
def f1(x1, w1, x2, w2, b, y):
    """
    Computes the forward and backward pass through the computational graph f1
    from the homework PDF.

    A few clarifications about the graph:
    - The subtraction node in the graph computes d = y_hat - y
    - The ^2 node squares its input

    Inputs:
    - x1, w1, x2, w2, b, y: Python floats

    Returns a tuple of:
    - L: Python scalar giving the output of the graph
    - grads: A tuple (grad_x1, grad_w1, grad_x2, grad_w2, grad_b, grad_y)
    giving the derivative of the output L with respect to each input.
    """
    # Forward pass: compute loss
    L = None
    # ############################################################################
    # TODO: Implement the forward pass for the computational graph f1 shown    #
```

```python
    # in the homework description. Store the loss in the variable L.        #
    # ##########################################################################
    a1 = x1 * w1
    a2 = x2 * w2
    y_bar = a1 + a2 + b
    d = y_bar - y
    L = d ** 2
    # ##########################################################################
    #                              END OF YOUR CODE                          #
    # ##########################################################################

    # Backward pass: compute gradients
    grad_x1, grad_w1, grad_x2, grad_w2 = None, None, None, None
    grad_b, grad_y = None, None
    # ##########################################################################
    # TODO: Implement the backward pass for the computational graph f1 shown  #
    # in the homework description. Store the gradients for each input        #
    # variable in the corresponding grad variagbles defined above.           #
    # ##########################################################################
    grad_L = 1
    grad_d = grad_L * 2 * d
    grad_y = grad_d * -1
    grad_y_bar = grad_d
    grad_b = grad_y_bar
    grad_a2 = grad_y_bar
    grad_a1 = grad_y_bar
    grad_w2 = grad_a2 * x2
    grad_x2 = grad_a2 * w2
    grad_w1 = grad_a1 * x1
    grad_x1 = grad_a1 * w1
    # ##########################################################################
    #                              END OF YOUR CODE                          #
    # ##########################################################################

```

```
    grads = (grad_x1, grad_w1, grad_x2, grad_w2, grad_b, grad_y)
    return L, grads

# Optional
def f2(x):
    """
    Computes the forward and backward pass through the computational graph
    f2
    from the homework PDF.

    A few clarifications about this graph:
    - The "x2" node multiplies its input by the constant 2
    - The "+1" and "-1" nodes add or subtract the constant 1
    - The division node computes y = t / b

    Inputs:
    - x: Python float

    Returns a tuple of:
    - y: Python float
    - grads: A tuple (grad_x,) giving the derivative of the output y with
      respect to the input x
    """
    # Forward pass: Compute output
    y = None
    #
    ###########################################################################
    # TODO: Implement the forward pass for the computational graph f2
    shown   #
    # in the homework description. Store the output in the variable y.
        #
    #
    ###########################################################################
    d = 2 * x
    e = math.exp(d)
    t = e - 1
    b = e + 1
    y = t / b
    #
    ###########################################################################
    #                              END OF YOUR CODE
        #
    #
    ###########################################################################

    # Backward pass: Compute gradients
    grad_x = None
    #
    ###########################################################################
```

```
    # TODO: Implement the backward pass for the computational graph f2 shown   #
    # in the homework description. Store the gradients for each input          #
    # variable in the corresponding grad variagbles defined above.             #
    # ###########################################################################
    grad_y = 1
    grad_t = grad_y * (1 / b)
    grad_b = grad_y * (- t / (b ** 2))
    grad_e = grad_t + grad_b
    grad_d = grad_e * math.exp(d)
    grad_x = grad_d * 2
    # ###########################################################################
    #                              END OF YOUR CODE                            #
    # ###########################################################################

    return y, (grad_x,)

# Required
def f3(s1, s2, y):
    """
    Computes the forward and backward pass through the computational graph f3
    from the homework PDF.

    A few clarifications about the graph:
    - The input y is an integer with y == 1 or y == 2; you do not need to
      compute a gradient for this input.
    - The division nodes compute p1 = e1 / d and p2 = e2 / d
    - The choose(p1, p2, y) node returns p1 if y is 1, or p2 if y is 2.

    Inputs:
    - s1, s2: Python floats
    - y: Python integer, either equal to 1 or 2

    Returns a tuple of:
    - L: Python scalar giving the output of the graph
    - grads: A tuple (grad_s1, grad_s2) giving the derivative of the output L
    with respect to the inputs s1 and s2.
    """
    assert y == 1 or y == 2
    # Forward pass: Compute loss
    L = None
    # ###########################################################################
```

```
    # TODO: Implement the forward pass for the computational graph f3 shown    #
    # in the homework description. Store the loss in the variable L.           #
    # ##########################################################################
    e1 = math.exp(s1)
    e2 = math.exp(s2)
    d = e1 + e2
    p1 = e1 / d
    p2 = e2 / d
    if y == 1:
        p_plus = p1
    elif y ==2:
        p_plus = p2
    L = -math.log(p_plus)
    # ##########################################################################
    #                              END OF YOUR CODE                            #
    # ##########################################################################

    # Backward pass: Compute gradients
    grad_s1, grad_s2 = None, None
    # ##########################################################################
    # TODO: Implement the backward pass for the computational graph f3 shown   #
    # in the homework description. Store the gradients for each input          #
    # variable in the corresponding grad variagbles defined above. You do not  #
    # need to compute a gradient for the input y since it is an integer.       #
    #                                                                          #
    # HINT: You may need an if statement to backprop through the choose node   #
    # ##########################################################################
    grad_L = 1
    grad_p_plus = grad_L * (- 1 / p_plus)
    if y == 1:
        grad_p1 = grad_p_plus
        grad_p2 = 0
    elif y == 2:
        grad_p1 = 0
```

```
        grad_p2 = grad_p_plus
    grad_d = grad_p1 * (- e1 / d ** 2) + grad_p2 * (- e2 / d ** 2)
    grad_e1 = grad_d + grad_p1 * (1 / d)
    grad_e2 = grad_d + grad_p2 * (1 / d)
    grad_s1 = grad_e1 * math.exp(s1)
    grad_s2 = grad_e2 * math.exp(s2)
    # ###########################################################################
    #                              END OF YOUR CODE                              #
    # ###########################################################################

    grads = (grad_s1, grad_s2)
    return L, grads


def f3_y1(s1, s2):
    """
    Helper function to compute f3 in the case where y = 1

    Inputs:
    - s1, s2: Same as f3

    Outputs: Same as f3
    """
    return f3(s1, s2, y=1)


def f3_y2(s1, s2):
    """
    Helper function to compute f3 in the case where y = 2

    Inputs:
    - s1, s2: Same as f3

    Outputs: Same as f3
    """
    return f3(s1, s2, y=2)
```

1.2 Task 2: Modular Backprop API

1.2.1 Fully-connected layer

Code submitted to Canvas.

```
import numpy as np


def fc_forward(x, w, b):
    """
    Computes the forward pass for a fully-connected layer.
```

```
    The input x has shape (N, Din) and contains a minibatch of N
    examples, where each example x[i] has shape (Din,).

    Inputs:
    - x: A numpy array of shape (N, Din) giving input data
    - w: A numpy array of shape (Din, Dout) giving weights
    - b: A numpy array of shape (Dout,) giving biases

    Returns a tuple of:
    - out: output, of shape (N, Dout)
    - cache: (x, w, b)
    """
    out = None
    # ##########################################################################
    # TODO: Implement the forward pass. Store the result in out.               #
    # ##########################################################################
    out = x @ w + b
    # ##########################################################################
    #                              END OF YOUR CODE                            #
    # ##########################################################################
    cache = (x, w, b)
    return out, cache


def fc_backward(grad_out, cache):
    """
    Computes the backward pass for a fully-connected layer.

    Inputs:
    - grad_out: Numpy array of shape (N, Dout) giving upstream gradients
    - cache: Tuple of:
      - x: A numpy array of shape (N, Din) giving input data
      - w: A numpy array of shape (Din, Dout) giving weights
      - b: A numpy array of shape (Dout,) giving biases

    Returns a tuple of downstream gradients:
    - grad_x: A numpy array of shape (N, Din) of gradient with respect to x
    - grad_w: A numpy array of shape (Din, Dout) of gradient with respect to w
    - grad_b: A numpy array of shape (Dout,) of gradient with respect to b
    """
    x, w, b = cache
```

```python
    grad_x, grad_w, grad_b = None, None, None
    # ##########################################################################
    # TODO: Implement the backward pass for the fully-connected layer          #
    # ##########################################################################
    N, _ = x.shape
    grad_x = grad_out @ w.T
    grad_w = x.T @ grad_out
    grad_b = (grad_out.T @ np.ones((N, 1))).squeeze()
    # ##########################################################################
    #                              END OF YOUR CODE                            #
    # ##########################################################################
    return grad_x, grad_w, grad_b
```

1.2.2 ReLU nonlinearity

Code submitted to Canvas.

```python
def relu_forward(x):
    """
    Computes the forward pass for the Rectified Linear Unit (ReLU) nonlinearity

    Input:
    - x: A numpy array of inputs, of any shape

    Returns a tuple of:
    - out: A numpy array of outputs, of the same shape as x
    - cache: x
    """
    out = None
    # ##########################################################################
    # TODO: Implement the ReLU forward pass.                                   #
    # ##########################################################################
    out = np.copy(x)
    out[out < 0] = 0
    # ##########################################################################
```

```
    #                              END OF YOUR CODE
        #
    #
###########################################################################
    cache = x
    return out, cache


def relu_backward(grad_out, cache):
    """
    Computes the backward pass for a Rectified Linear Unit (ReLU)
nonlinearity

    Input:
    - grad_out: Upstream derivatives, of any shape
    - cache: Input x, of same shape as dout

    Returns:
    - grad_x: Gradient with respect to x
    """
    grad_x, x = None, cache
    #
###########################################################################
    # TODO: Implement the ReLU backward pass.
        #
    #
###########################################################################
    grad_x = np.copy(x)
    grad_x[grad_x >= 0] = 1
    grad_x[grad_x < 0] = 0
    grad_x = grad_x * grad_out
    #
###########################################################################
    #                              END OF YOUR CODE
        #
    #
###########################################################################
    return grad_x
```

1.2.3 Softmax Loss Function

Code submitted to Canvas.

```
def softmax_loss(x, y):
    """
    Computes the loss and gradient for softmax (cross-entropy) loss
function.

    Inputs:
```

```
    - x: Numpy array of shape (N, C) giving predicted class scores, where
      x[i, c] gives the predicted score for class c on input sample i
    - y: Numpy array of shape (N,) giving ground-truth labels, where
      y[i] = c means that input sample i has ground truth label c, where
      0 <= c < C.

    Returns a tuple of:
    - loss: Scalar giving the loss
    - grad_x: Numpy array of shape (N, C) giving the gradient of the loss
with
      with respect to x
    """
    loss, grad_x = None, None
    #
###########################################################################

    # TODO: Implement softmax loss
        #
    #
###########################################################################

    N, _ = x.shape
    y_one_hot = np.zeros_like(x)
    y_one_hot[np.arange(N), y] = 1

    shifted_x = x - np.max(x, axis=1, keepdims=True)
    log_probs = shifted_x - np.log( np.sum(np.exp(shifted_x), axis=1,
keepdims=True) )
    probs = np.exp(log_probs)
    loss = - np.sum(y_one_hot * log_probs) / N

    grad_x = (probs - y_one_hot) / N
    #
###########################################################################

    #                               END OF YOUR CODE
        #
    #
###########################################################################

    return loss, grad_x
```

1.2.4 L2 Regularization

Code submitted to Canvas.

```
def l2_regularization(w, reg):
    """
    Computes loss and gradient for L2 regularization of a weight matrix:

    loss = (reg / 2) * sum_i w_i^2

    Where the sum ranges over all elements of w.

```

```
    Inputs:
    - w: Numpy array of any shape
    - reg: float giving the regularization strength

    Returns:
    """
    loss, grad_w = None, None
    #
   ##########################################################################
    # TODO: Implement L2 regularization.
       #
    #
   ##########################################################################
    loss = (reg / 2) * np.sum(w ** 2)

    grad_w = reg * w
    #
   ##########################################################################
    #                              END OF YOUR CODE
       #
    #
   ##########################################################################
    return loss, grad_w
```

1.3 Task 3: Implementing a Two-layer Network

Code submitted to Canvas.

```
import numpy as np
from classifier import Classifier
from layers import fc_forward, fc_backward, relu_forward, relu_backward


class TwoLayerNet(Classifier):
    """
    A neural network with two layers, using a ReLU nonlinearity on its one
    hidden layer. That is, the architecture should be:

    input -> FC layer -> ReLU layer -> FC layer -> scores
    """
    def __init__(self, input_dim=3072, num_classes=10, hidden_dim=512,
                 weight_scale=1e-3):
        """
        Initialize a new two layer network.

        Inputs:
        - input_dim: The number of dimensions in the input.
        - num_classes: The number of classes over which to classify
        - hidden_dim: The size of the hidden layer
```

```python
        - weight_scale: The weight matrices of the model will be initialized
          from a Gaussian distribution with standard deviation equal to
          weight_scale. The bias vectors of the model will always be
          initialized to zero.
        """
        # ######################################################################
        # TODO: Initialize the weights and biases of a two-layer network. #
        # ######################################################################
        self.W1 = np.random.normal(0, weight_scale, size=(input_dim, hidden_dim))
        self.b1 = np.zeros(hidden_dim)
        self.W2 = np.random.normal(0, weight_scale, size=(hidden_dim, num_classes))
        self.b2 = np.zeros(num_classes)
        # ######################################################################
        #                           END OF YOUR CODE                           #
        # ######################################################################

    def parameters(self):
        params = None
        # ######################################################################
        # TODO: Build a dict of all learnable parameters of this model. #
        # ######################################################################
        params = {"W1" : self.W1,
                  "b1" : self.b1,
                  "W2" : self.W2,
                  "b2" : self.b2}
        # ######################################################################
        #                           END OF YOUR CODE                           #
        # ######################################################################
        return params

    def forward(self, X):
        scores, cache = None, None
        # ######################################################################
        # TODO: Implement the forward pass to compute classification scores   #
        # for the input data X. Store into cache any data that will be needed #
```

```
        # during the backward pass.                                          #
        # ######################################################################
        params = self.parameters()
        h1, cache1 = fc_forward(X, params["W1"], params["b1"])
        h2, cache2 = relu_forward(h1)
        scores, cache3 = fc_forward(h2, params["W2"], params["b2"])
        cache = (cache1, cache2, cache3)
        # ######################################################################
        #                             END OF YOUR CODE                         #
        # ######################################################################
        return scores, cache

    def backward(self, grad_scores, cache):
        grads = None
        # ######################################################################
        # TODO: Implement the backward pass to compute gradients for all      #
        # learnable parameters of the model, storing them in the grads dict   #
        # above. The grads dict should give gradients for all parameters in   #
        # the dict returned by model.parameters().                            #
        # ######################################################################
        cache1, cache2, cache3 = cache
        grad_h2, grad_W2, grad_b2 = fc_backward(grad_scores, cache3)
        grad_h1 = relu_backward(grad_h2, cache2)
        grad_X, grad_W1, grad_b1 = fc_backward(grad_h1, cache1)
        grads = {"W1" : grad_W1,
                 "b1" : grad_b1,
                 "W2" : grad_W2,
                 "b2" : grad_b2}
        # ######################################################################
        #                             END OF YOUR CODE                         #
        # ######################################################################
        return grads
```

1.4 Task 4: Training Two-Layer Networks

1.4.1 Training_step()

Code submitted to Canvas.

```
def training_step(model, X_batch, y_batch, reg):
```

```python
    """
    Compute the loss and gradients for a single training iteration of a model
    given a minibatch of data. The loss should be a sum of a cross-entropy loss
    between the model predictions and the ground-truth image labels, and
    an L2 regularization term on all weight matrices in the fully-connected
    layers of the model. You should not regularize the bias vectors.

    Inputs:
    - model: A Classifier instance
    - X_batch: A numpy array of shape (N, D) giving a minibatch of images
    - y_batch: A numpy array of shape (N,) where 0 <= y_batch[i] < C is the
      ground-truth label for the image X_batch[i]
    - reg: A float giving the strength of L2 regularization to use.

    Returns a tuple of:
    - loss: A float giving the loss (data loss + regularization loss) for the
      model on this minibatch of data
    - grads: A dictionary giving gradients of the loss with respect to the
      parameters of the model. In particular grads[k] should be the gradient
      of the loss with respect to model.parameters()[k].
    """
    loss, grads = None, None
    # #########################################################################
    # TODO: Compute the loss and gradient for one training iteration.        #
    # #########################################################################
    params = model.parameters()
    score, caches = model.forward(X_batch)
    reg_loss_W1, reg_grad_W1 = l2_regularization(params["W1"], reg)
    reg_loss_W2, reg_grad_W2 = l2_regularization(params["W2"], reg)
    loss, grad_score = softmax_loss(score, y_batch)
    loss = loss + reg_loss_W1 + reg_loss_W2

    grads = model.backward(grad_score, caches)
    grads["W1"] += reg_grad_W1
    grads["W2"] += reg_grad_W2
    # #########################################################################
    #                              END OF YOUR CODE                          #
    # #########################################################################
```

```
    return loss, grads
```

1.4.2 Result

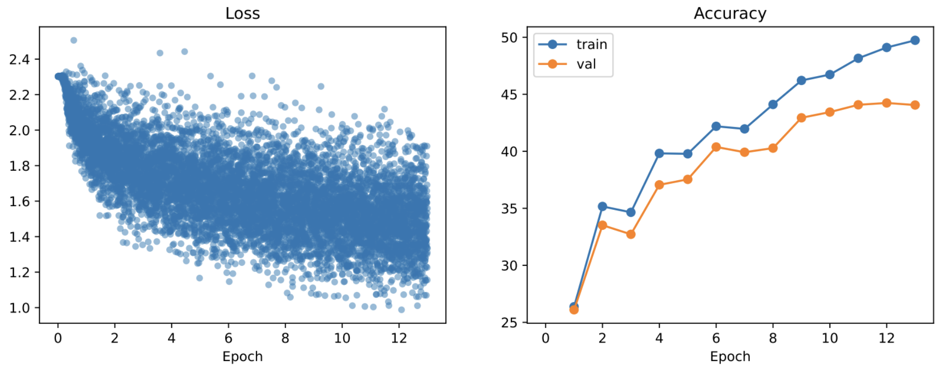


Figure 1: Best model plot

The hyperparameters are:

```
    # How much data to use for training
    num_train = 20000
    # Model architecture hyperparameters.
    hidden_dim = 200
    # Optimization hyperparameters.
    batch_size = 32
    num_epochs = 13
    learning_rate = 3e-2
    reg = 0.00
```

The final test set performance of this model is: 44.82%

1.4.3 Over-fit

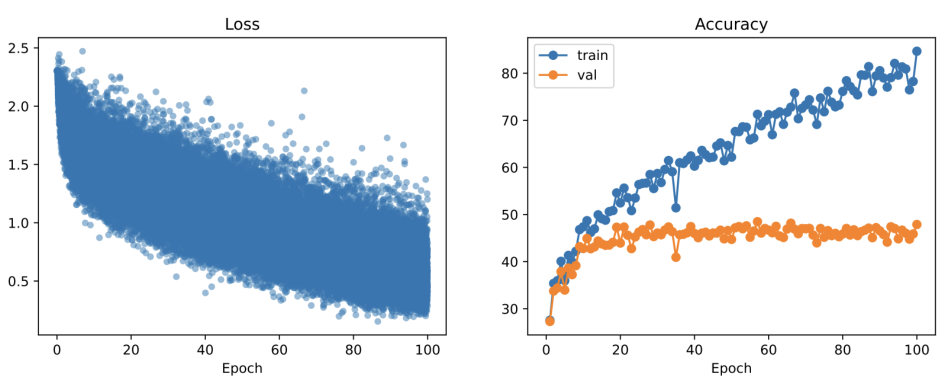


Figure 2: Overfit model plot

The hyperparameters are:

```
    # How much data to use for training
    num_train = 20000
    # Model architecture hyperparameters.
    hidden_dim = 200
    # Optimization hyperparameters.
    batch_size = 32
    num_epochs = 100
    learning_rate = 3e-2
    reg = 0.00
```

2 Section 2

2.1 Task 5: Train Your Own Classification Model

2.1.1 Code Implementation

Code submitted to Canvas.

Notebook pdf file see appendix.

2.1.2 Model structure, Hyperparameter, and Result

I'm using the similar structure as ResNet-18, but only adopting 2 non-bottleneck stage here. The model structure is:

```
Your network:
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 28, 28]           3,200
       BatchNorm2d-2           [-1, 64, 28, 28]             128
         MaxPool2d-3           [-1, 64, 14, 14]               0
            Conv2d-4           [-1, 64, 14, 14]          36,928
       BatchNorm2d-5           [-1, 64, 14, 14]             128
              ReLU-6           [-1, 64, 14, 14]               0
            Conv2d-7           [-1, 64, 14, 14]          36,928
       BatchNorm2d-8           [-1, 64, 14, 14]             128
              ReLU-9           [-1, 64, 14, 14]               0
           Conv2d-10           [-1, 64, 14, 14]          36,928
      BatchNorm2d-11           [-1, 64, 14, 14]             128
             ReLU-12           [-1, 64, 14, 14]               0
           Conv2d-13           [-1, 64, 14, 14]          36,928
      BatchNorm2d-14           [-1, 64, 14, 14]             128
             ReLU-15           [-1, 64, 14, 14]               0
           Conv2d-16            [-1, 128, 7, 7]           8,320
      BatchNorm2d-17            [-1, 128, 7, 7]             256
           Conv2d-18            [-1, 128, 7, 7]          73,856
      BatchNorm2d-19            [-1, 128, 7, 7]             256
             ReLU-20            [-1, 128, 7, 7]               0
           Conv2d-21            [-1, 128, 7, 7]         147,584
      BatchNorm2d-22            [-1, 128, 7, 7]             256
             ReLU-23            [-1, 128, 7, 7]               0
           Conv2d-24            [-1, 128, 7, 7]         147,584
```

```
      BatchNorm2d-25           [-1, 128, 7, 7]             256
             ReLU-26           [-1, 128, 7, 7]               0
           Conv2d-27           [-1, 128, 7, 7]         147,584
      BatchNorm2d-28           [-1, 128, 7, 7]             256
             ReLU-29           [-1, 128, 7, 7]               0
        AvgPool2d-30           [-1, 128, 1, 1]               0
           Linear-31                  [-1, 512]         66,048
             ReLU-32                  [-1, 512]              0
           Linear-33                   [-1, 10]          5,130
================================================================
Total params: 748,938
Trainable params: 748,938
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 2.69
Params size (MB): 2.86
Estimated Total Size (MB): 5.55
----------------------------------------------------------------
None
```

Hyperparameters:

```
    batch_size = 64
    learning_rate, weight_decay, num_epoch = 1e-4, 0.0, 5
```

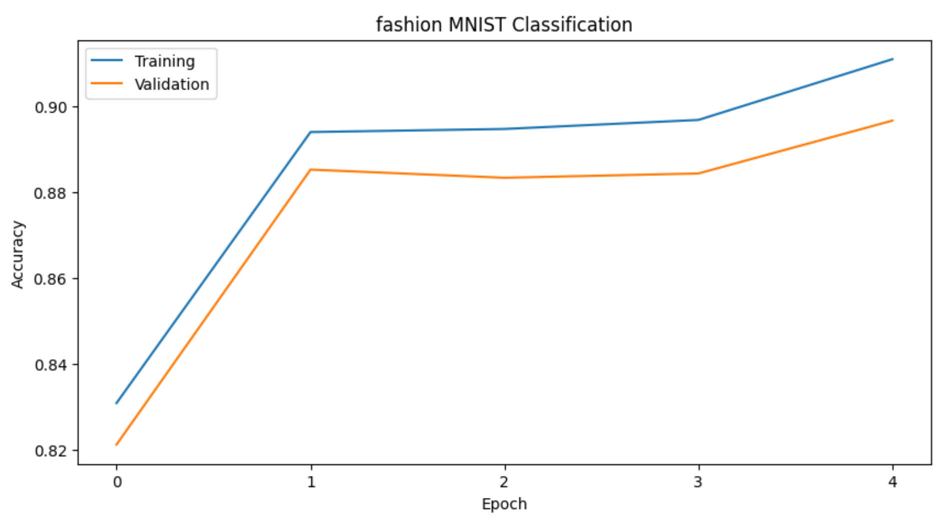


Figure 3: My CNN result on fashion MNIST classification

2.1.3 Best Result

The best accuracy is: 89.17%

2.2 Task 6: Pre-trained NN

Notebook pdf file see appendix

Figure 4: Cup 1

Figure 5: Cup 2

The first cup is correct, but the second cup is not a measuring cup because there is no marked scale. The pre-trained network tends to classify transparent object as measuring cup.

3 Appendix

part1

March 20, 2024

1 EECS 442 Homework 4: Fashion-MNIST Classification

In this part, you will implement and train Convolutional Neural Networks (ConvNets) in PyTorch to classify images. Unlike HW4 Secion 1, backpropagation is automatically inferred by PyTorch, so you only need to write code for the forward pass.

Before we start, please put your name and UMID in following format Firstname LASTNAME, #00000000 // e.g.) David FOUHEY, #12345678

Your Answer:
Wensong HU #24908654

1.1 Setup

[2]:
```
# Run the command in the terminal if it failed on local Jupyter Notebook,␣
↪remove "!" before each line
# !pip install torchsummary
```

[3]:
```
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm # Displays a progress bar

import torch
from torch import nn
from torch import optim
import torch.nn.functional as F
from torchsummary import summary
from torchvision import datasets, transforms
from torch.utils.data import Dataset, Subset, DataLoader, random_split
```

[4]:
```
if torch.cuda.is_available():
    print("Using the GPU. You are good to go!")
    device = 'cuda'
else:
    print("Using the CPU. Overall speed may be slowed down")
    device = 'cpu'
```

```
Using the GPU. You are good to go!
```

1.2 Loading Dataset

The dataset we use is Fashion-MNIST dataset, which is available at https://github.com/zalandoresearch/fashion-mnist and in torchvision.datasets. Fashion-MNIST has 10 classes, 60000 training+validation images (we have splitted it to have 50000 training images and 10000 validation images, but you can change the numbers), and 10000 test images.

[5]:
```python
# Load the dataset and train, val, test splits
print("Loading datasets...")
# Transform from [0,255] uint8 to [0,1] float,
# then normalize to zero mean and unit variance
FASHION_transform = transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize([0.2859], [0.3530])
                    ])
FASHION_trainval = datasets.FashionMNIST('.', download=True, train=True,
                                          transform=FASHION_transform)
FASHION_train = Subset(FASHION_trainval, range(50000))
FASHION_val = Subset(FASHION_trainval, range(50000, 60000))
FASHION_test = datasets.FashionMNIST('.', download=True, train=False,
                                     transform=FASHION_transform)
print("Done!")
```

```
Loading datasets…
Done!
```

Now, we will create the dataloder for train, val and test dataset. You are free to experiment with different batch sizes.

[6]:
```python
# Create dataloaders
##############################################################################
# TODO: Experiment with different batch sizes                                #
##############################################################################
batch_size = 64
##############################################################################
#                              END OF YOUR CODE                              #
##############################################################################
trainloader = DataLoader(FASHION_train, batch_size=batch_size, shuffle=True)
valloader = DataLoader(FASHION_val, batch_size=batch_size, shuffle=True)
testloader = DataLoader(FASHION_test, batch_size=batch_size, shuffle=True)
```

1.3 Model

Initialize your model and experiment with with different optimizers, parameters (such as learning rate) and number of epochs.

[7]:
```python
class Network(nn.Module):
    def __init__(self):
        super().__init__()
```

```
    ␣
↪###############################################################################
        # TODO: Design your own network, define layers here.                    ␣
↪        #
        # Here We provide a sample of two-layer fc network from HW4 Part3.      ␣
↪       #
        # Your solution, however, should contain convolutional layers.          ␣
↪       #
        # Refer to PyTorch documentations of torch.nn to pick your layers.      ␣
↪       #
        # (https://pytorch.org/docs/stable/nn.html)                             ␣
↪       #
        # Some common choices: Linear, Conv2d, ReLU, MaxPool2d, AvgPool2d,␣
↪Dropout    #
        # If you have many layers, use nn.Sequential() to simplify your code    ␣
↪       #
    ␣
↪###############################################################################
        # stem: 3*28*28 -> 64 * 14* 14
        self.stem = torch.nn.Sequential(torch.nn.Conv2d(1, 64, kernel_size=7,␣
↪stride=1, padding=3),
                                        torch.nn.BatchNorm2d(64),
                                        torch.nn.MaxPool2d(kernel_size=3,␣
↪stride=2, padding=1))
        # stage1: 64 * 14 * 14 -> 64 * 14 * 14
        self.resblock1 = torch.nn.Sequential(torch.nn.Conv2d(64, 64,␣
↪kernel_size=3, stride=1, padding=1),
                                             torch.nn.BatchNorm2d(64),
                                             torch.nn.ReLU(inplace=True),
                                             torch.nn.Conv2d(64, 64,␣
↪kernel_size=3, stride=1, padding=1),
                                             torch.nn.BatchNorm2d(64))
        self.resblock2 = torch.nn.Sequential(torch.nn.Conv2d(64, 64,␣
↪kernel_size=3, stride=1, padding=1),
                                             torch.nn.BatchNorm2d(64),
                                             torch.nn.ReLU(inplace=True),
                                             torch.nn.Conv2d(64, 64,␣
↪kernel_size=3, stride=1, padding=1),
                                             torch.nn.BatchNorm2d(64))
        # stage2: 64 * 14 * 14 -> 128 * 7 * 7
        self.resblock3 = torch.nn.Sequential(torch.nn.Conv2d(64, 128,␣
↪kernel_size=3, stride=2, padding=1),
                                             torch.nn.BatchNorm2d(128),
                                             torch.nn.ReLU(inplace=True),
                                             torch.nn.Conv2d(128, 128,␣
↪kernel_size=3, stride=1, padding=1),
```

```
                                             torch.nn.BatchNorm2d(128))
        self.resblock4 = torch.nn.Sequential(torch.nn.Conv2d(128, 128,␣
↪kernel_size=3, stride=1, padding=1),
                                             torch.nn.BatchNorm2d(128),
                                             torch.nn.ReLU(inplace=True),
                                             torch.nn.Conv2d(128, 128,␣
↪kernel_size=3, stride=1, padding=1),
                                             torch.nn.BatchNorm2d(128))
        # fc layer:
        self.pool = torch.nn.AvgPool2d(kernel_size=7)
        self.fc = torch.nn.Sequential(torch.nn.Linear(128, 512),
                                      torch.nn.ReLU(inplace=True),
                                      torch.nn.Linear(512, 10))

        # downsample
        self.projection = torch.nn.Sequential(torch.nn.Conv2d(64, 128,␣
↪kernel_size=1, stride=2),
                                              torch.nn.BatchNorm2d(128))
        #ReLU
        self.relu = torch.nn.ReLU(inplace=True)
       ␣
↪##############################################################################
        #                                END OF YOUR CODE                              ␣
↪       #
       ␣
↪##############################################################################

    def forward(self, x):
       ␣
↪##############################################################################
        # TODO: Design your own network, implement forward pass here                  ␣
↪       #
       ␣
↪##############################################################################
        # print(x.shape)
        N, C, H, W = x.shape

        # stem
        x = self.stem(x)
        # print(x.shape)

        # stage1
        x_res = torch.clone(x)
        x = self.resblock1(x)
        x += x_res
        x = self.relu(x)
```

```
        # print(x.shape)

        x_res = torch.clone(x)
        x = self.resblock2(x)
        x += x_res
        x = self.relu(x)
        # print(x.shape)

        # stage2
        x_res = self.projection(x)
        x = self.resblock3(x)
        x += x_res
        x = self.relu(x)
        # print(x.shape)

        x_res = torch.clone(x)
        x = self.resblock4(x)
        x += x_res
        x = self.relu(x)
        # print(x.shape)

        # fc layer
        x = self.pool(x)
        # print(x.shape)
        x = torch.flatten(x, start_dim=1)
        # print(x.shape)
        x = self.fc(x)

        return x
       ␣
 ↪###############################################################################
        #                              END OF YOUR CODE                              ␣
 ↪      #
       ␣
 ↪###############################################################################

model = Network().to(device)
criterion = nn.CrossEntropyLoss() # Specify the loss layer
print('Your network:')
print(summary(model, (1,28,28), device=device)) # visualize your model

###############################################################################
# TODO: Modify the lines below to experiment with different optimizers,      #
# parameters (such as learning rate) and number of epochs.                    #
###############################################################################
# Set up optimization hyperparameters
learning_rate, weight_decay, num_epoch = 1e-4, 0.0, 5
```

```
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate,␣
 ↪weight_decay=weight_decay)
##############################################################################
#                              END OF YOUR CODE                              #
##############################################################################
```

```
Your network:
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 28, 28]           3,200
       BatchNorm2d-2           [-1, 64, 28, 28]             128
         MaxPool2d-3           [-1, 64, 14, 14]               0
            Conv2d-4           [-1, 64, 14, 14]          36,928
       BatchNorm2d-5           [-1, 64, 14, 14]             128
              ReLU-6           [-1, 64, 14, 14]               0
            Conv2d-7           [-1, 64, 14, 14]          36,928
       BatchNorm2d-8           [-1, 64, 14, 14]             128
              ReLU-9           [-1, 64, 14, 14]               0
           Conv2d-10           [-1, 64, 14, 14]          36,928
      BatchNorm2d-11           [-1, 64, 14, 14]             128
             ReLU-12           [-1, 64, 14, 14]               0
           Conv2d-13           [-1, 64, 14, 14]          36,928
      BatchNorm2d-14           [-1, 64, 14, 14]             128
             ReLU-15           [-1, 64, 14, 14]               0
           Conv2d-16            [-1, 128, 7, 7]           8,320
      BatchNorm2d-17            [-1, 128, 7, 7]             256
           Conv2d-18            [-1, 128, 7, 7]          73,856
      BatchNorm2d-19            [-1, 128, 7, 7]             256
             ReLU-20            [-1, 128, 7, 7]               0
           Conv2d-21            [-1, 128, 7, 7]         147,584
      BatchNorm2d-22            [-1, 128, 7, 7]             256
             ReLU-23            [-1, 128, 7, 7]               0
           Conv2d-24            [-1, 128, 7, 7]         147,584
      BatchNorm2d-25            [-1, 128, 7, 7]             256
             ReLU-26            [-1, 128, 7, 7]               0
           Conv2d-27            [-1, 128, 7, 7]         147,584
      BatchNorm2d-28            [-1, 128, 7, 7]             256
             ReLU-29            [-1, 128, 7, 7]               0
        AvgPool2d-30            [-1, 128, 1, 1]               0
           Linear-31                  [-1, 512]          66,048
             ReLU-32                  [-1, 512]               0
           Linear-33                   [-1, 10]           5,130
================================================================
Total params: 748,938
Trainable params: 748,938
Non-trainable params: 0
----------------------------------------------------------------
```

```
Input size (MB): 0.00
Forward/backward pass size (MB): 2.69
Params size (MB): 2.86
Estimated Total Size (MB): 5.55
----------------------------------------------------------------
None
```

Run the cell below to start your training, we expect you to achieve over **85%** on the test set. A valid solution that meet the requirement take no more than **10 minutes** on normal PC Intel core CPU setting. If your solution takes too long to train, try to simplify your model or reduce the number of epochs.

[8]:
```
%%time
def train(model, trainloader, valloader, num_epoch=10):  # Train the model
    print("Start training...")
    trn_loss_hist = []
    trn_acc_hist = []
    val_acc_hist = []
    model.train()  # Set the model to training mode
    for i in range(num_epoch):
        running_loss = []
        print('-----------------Epoch = %d-----------------' % (i+1))
        for batch, label in tqdm(trainloader):
            batch = batch.to(device)
            label = label.to(device)
            optimizer.zero_grad()  # Clear gradients from the previous iteration
            # This will call Network.forward() that you implement
            pred = model(batch)
            loss = criterion(pred, label)  # Calculate the loss
            running_loss.append(loss.item())
            loss.backward()  # Backprop gradients to all tensors in the network
            optimizer.step()  # Update trainable weights
        print("\n Epoch {} loss:{}".format(i+1, np.mean(running_loss)))

        # Keep track of training loss, accuracy, and validation loss
        trn_loss_hist.append(np.mean(running_loss))
        trn_acc_hist.append(evaluate(model, trainloader))
        print("\n Evaluate on validation set...")
        val_acc_hist.append(evaluate(model, valloader))
    print("Done!")
    return trn_loss_hist, trn_acc_hist, val_acc_hist


def evaluate(model, loader):  # Evaluate accuracy on validation / test set
    model.eval()  # Set the model to evaluation mode
    correct = 0
    with torch.no_grad():  # Do not calculate grident to speed up computation
        for batch, label in tqdm(loader):
```

```
            batch = batch.to(device)
            label = label.to(device)
            pred = model(batch)
            correct += (torch.argmax(pred, dim=1) == label).sum().item()
        acc = correct/len(loader.dataset)
        print("\n Evaluation accuracy: {}".format(acc))
        return acc


trn_loss_hist, trn_acc_hist, val_acc_hist = train(model, trainloader,
                                                  valloader, num_epoch)

##############################################################################
# TODO: Note down the evaluation accuracy on test set                        #
##############################################################################
print("\n Evaluate on test set")
evaluate(model, testloader)
#5: 64, 1e-4, 0.95, 5: 0.7843
#6: 64, 1e-4, 0.98, 5: 0.7851
#7: 64, 1e-4, 0.00, 5: 0.8917
```

```
Start training…
-----------------Epoch = 1-----------------

  0%|          | 0/782 [00:00<?, ?it/s]100%|     | 782/782 [00:13<00:00,
59.46it/s]


 Epoch 1 loss:0.5725756988424779

100%|     | 782/782 [00:07<00:00, 99.74it/s]


 Evaluation accuracy: 0.83086

 Evaluate on validation set…

100%|     | 157/157 [00:01<00:00, 96.58it/s]


 Evaluation accuracy: 0.8212
-----------------Epoch = 2-----------------

100%|     | 782/782 [00:12<00:00, 61.51it/s]


 Epoch 2 loss:0.38106703539105025

100%|     | 782/782 [00:07<00:00, 101.50it/s]


 Evaluation accuracy: 0.89394
```

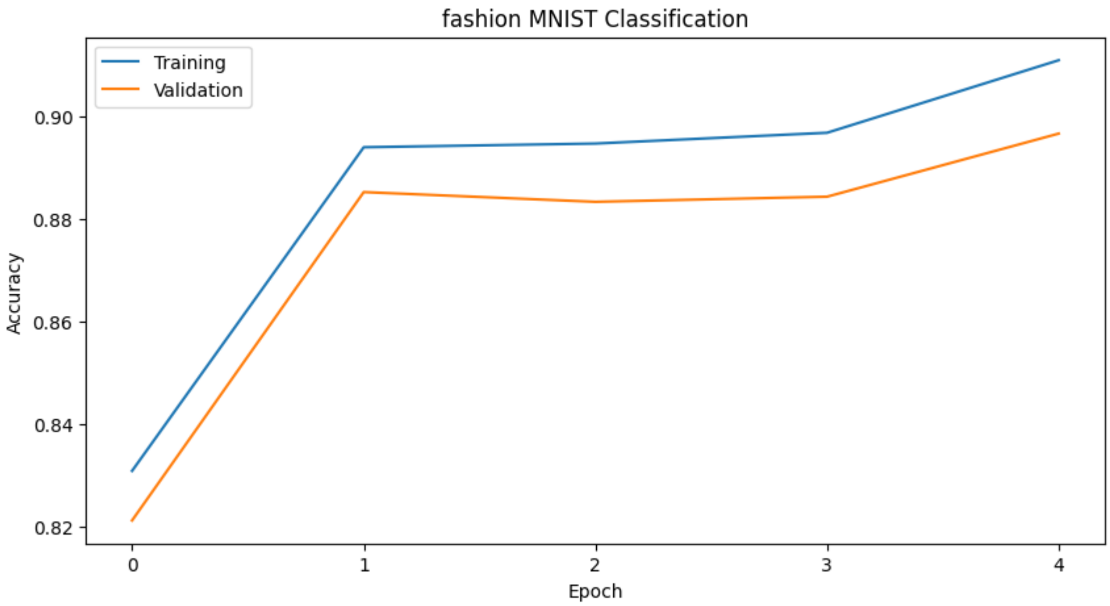
```
 Evaluate on validation set…
100%|      | 157/157 [00:01<00:00, 98.84it/s]

 Evaluation accuracy: 0.8852
-----------------Epoch = 3-----------------
100%|      | 782/782 [00:12<00:00, 61.09it/s]

 Epoch 3 loss:0.31754977807707496
100%|      | 782/782 [00:07<00:00, 102.77it/s]

 Evaluation accuracy: 0.89466

 Evaluate on validation set…
100%|      | 157/157 [00:01<00:00, 97.87it/s]

 Evaluation accuracy: 0.8833
-----------------Epoch = 4-----------------
100%|      | 782/782 [00:12<00:00, 61.65it/s]

 Epoch 4 loss:0.2882935323983507
100%|      | 782/782 [00:07<00:00, 101.61it/s]

 Evaluation accuracy: 0.89676

 Evaluate on validation set…
100%|      | 157/157 [00:01<00:00, 101.90it/s]

 Evaluation accuracy: 0.8843
-----------------Epoch = 5-----------------
100%|      | 782/782 [00:12<00:00, 61.54it/s]

 Epoch 5 loss:0.2685324148086788
100%|      | 782/782 [00:07<00:00, 99.73it/s]

 Evaluation accuracy: 0.9109

 Evaluate on validation set…
```

```
100%|          | 157/157 [00:01<00:00, 101.33it/s]


 Evaluation accuracy: 0.8966
Done!

 Evaluate on test set

100%|          | 157/157 [00:01<00:00, 103.19it/s]


 Evaluation accuracy: 0.8917
CPU times: user 1min 54s, sys: 512 ms, total: 1min 54s
Wall time: 1min 52s
```

[8]:
```
0.8917
```

Once your training is complete, run the cell below to visualize the training and validation accuracies across iterations.

[9]:
```
##############################################################################
# TODO: Submit the accuracy plot                                             #
##############################################################################
# visualize the training / validation accuracies
x = np.arange(num_epoch)
# train/val accuracies for MiniVGG
plt.figure()
plt.plot(x, trn_acc_hist)
plt.plot(x, val_acc_hist)
plt.legend(['Training', 'Validation'])
plt.xticks(x)
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('fashion MNIST Classification')
plt.gcf().set_size_inches(10, 5)
plt.savefig('part1.png', dpi=300)
plt.show()
```


fashion MNIST Classification
Training
Validation
0.90
0.88
0.86
0.84
0.82
Accuracy
0
1
2
3
4
Epoch

part2

March 20, 2024

1 EECS 442 Homework 4 - PyTorch ConvNets

In this notebook we will explore how to use a pre-trained PyTorch convolution neural network (ConvNet).

Before we start, please put your name and UMID in following format Firstname LASTNAME, #00000000 // e.g.) David FOUHEY, #12345678

Your Answer:
Wensong HU #24908654

1.1 Setup

[1]:
```
import os
import json
import torch
import torchvision
import torchvision.transforms as T
import random
import numpy as np
from scipy.ndimage.filters import gaussian_filter1d
import matplotlib.pyplot as plt
SQUEEZENET_MEAN = torch.tensor([0.485, 0.456, 0.406], dtype=torch.float)
SQUEEZENET_STD = torch.tensor([0.229, 0.224, 0.225], dtype=torch.float)
from PIL import Image

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'
```

```
/tmp/ipykernel_1781330/770024508.py:8: DeprecationWarning: Please import
`gaussian_filter1d` from the `scipy.ndimage` namespace; the
`scipy.ndimage.filters` namespace is deprecated and will be removed in SciPy
2.0.0.
  from scipy.ndimage.filters import gaussian_filter1d
```

[2]:
```
if torch.cuda.is_available():
    print("Using the GPU. You are good to go!")
```

```
    device = 'cuda'
else:
    print("Using the CPU. Overall speed may be slowed down")
    device = 'cpu'
```

```
Using the GPU. You are good to go!
```

For all of our experiments, we will start with a convolutional neural network which was pretrained to perform image classification on ImageNet [1]. We can use any model here, but for the purposes of this assignment we will use SqueezeNet [2], which achieves accuracies comparable to AlexNet but with a significantly reduced parameter count and computational complexity.

Using SqueezeNet rather than AlexNet or VGG or ResNet means that we can easily perform all experiments without heavy computation. Run the following cell to download and initialize your model.

[1] Olga Russakovsky, *Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. IJCV, 2015

[2] Iandola et al, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5MB model size", arXiv 2016

[3]:
```
print('Download and load the pretrained SqueezeNet model.')
model = torchvision.models.squeezenet1_1(pretrained=True).to(device)

# We don't want to train the model, so tell PyTorch not to compute gradients
# with respect to model parameters.
for param in model.parameters():
    param.requires_grad = False

# Make sure the model is in "eval" mode
model.eval()

# you may see warning regarding initialization deprecated, that's fine,
# please continue to next steps
```

```
Download and load the pretrained SqueezeNet model.
```

```
/home/umhws/anaconda3/envs/eecs442/lib/python3.10/site-
packages/torchvision/models/_utils.py:208: UserWarning: The parameter
'pretrained' is deprecated since 0.13 and may be removed in the future, please
use 'weights' instead.
  warnings.warn(
/home/umhws/anaconda3/envs/eecs442/lib/python3.10/site-
packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a
weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed
in the future. The current behavior is equivalent to passing
`weights=SqueezeNet1_1_Weights.IMAGENET1K_V1`. You can also use
`weights=SqueezeNet1_1_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

[3]:
```
SqueezeNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=True)
    (3): Fire(
      (squeeze): Conv2d(64, 16, kernel_size=(1, 1), stride=(1, 1))
      (squeeze_activation): ReLU(inplace=True)
      (expand1x1): Conv2d(16, 64, kernel_size=(1, 1), stride=(1, 1))
      (expand1x1_activation): ReLU(inplace=True)
      (expand3x3): Conv2d(16, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      (expand3x3_activation): ReLU(inplace=True)
    )
    (4): Fire(
      (squeeze): Conv2d(128, 16, kernel_size=(1, 1), stride=(1, 1))
      (squeeze_activation): ReLU(inplace=True)
      (expand1x1): Conv2d(16, 64, kernel_size=(1, 1), stride=(1, 1))
      (expand1x1_activation): ReLU(inplace=True)
      (expand3x3): Conv2d(16, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
      (expand3x3_activation): ReLU(inplace=True)
    )
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=True)
    (6): Fire(
      (squeeze): Conv2d(128, 32, kernel_size=(1, 1), stride=(1, 1))
      (squeeze_activation): ReLU(inplace=True)
      (expand1x1): Conv2d(32, 128, kernel_size=(1, 1), stride=(1, 1))
      (expand1x1_activation): ReLU(inplace=True)
      (expand3x3): Conv2d(32, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (expand3x3_activation): ReLU(inplace=True)
    )
    (7): Fire(
      (squeeze): Conv2d(256, 32, kernel_size=(1, 1), stride=(1, 1))
      (squeeze_activation): ReLU(inplace=True)
      (expand1x1): Conv2d(32, 128, kernel_size=(1, 1), stride=(1, 1))
      (expand1x1_activation): ReLU(inplace=True)
      (expand3x3): Conv2d(32, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (expand3x3_activation): ReLU(inplace=True)
    )
    (8): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=True)
    (9): Fire(
```

```
      (squeeze): Conv2d(256, 48, kernel_size=(1, 1), stride=(1, 1))
      (squeeze_activation): ReLU(inplace=True)
      (expand1x1): Conv2d(48, 192, kernel_size=(1, 1), stride=(1, 1))
      (expand1x1_activation): ReLU(inplace=True)
      (expand3x3): Conv2d(48, 192, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (expand3x3_activation): ReLU(inplace=True)
    )
    (10): Fire(
      (squeeze): Conv2d(384, 48, kernel_size=(1, 1), stride=(1, 1))
      (squeeze_activation): ReLU(inplace=True)
      (expand1x1): Conv2d(48, 192, kernel_size=(1, 1), stride=(1, 1))
      (expand1x1_activation): ReLU(inplace=True)
      (expand3x3): Conv2d(48, 192, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (expand3x3_activation): ReLU(inplace=True)
    )
    (11): Fire(
      (squeeze): Conv2d(384, 64, kernel_size=(1, 1), stride=(1, 1))
      (squeeze_activation): ReLU(inplace=True)
      (expand1x1): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1))
      (expand1x1_activation): ReLU(inplace=True)
      (expand3x3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (expand3x3_activation): ReLU(inplace=True)
    )
    (12): Fire(
      (squeeze): Conv2d(512, 64, kernel_size=(1, 1), stride=(1, 1))
      (squeeze_activation): ReLU(inplace=True)
      (expand1x1): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1))
      (expand1x1_activation): ReLU(inplace=True)
      (expand3x3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (expand3x3_activation): ReLU(inplace=True)
    )
  )
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Conv2d(512, 1000, kernel_size=(1, 1), stride=(1, 1))
    (2): ReLU(inplace=True)
    (3): AdaptiveAvgPool2d(output_size=(1, 1))
  )
)
```

Next, we will download the imagenet labels which we are going to use in the notebook. ImageNet labels are stored in the `idx2label` dictionary of `{index(int): label(str)}`.

[4]:
```
# Loading the imagenet class labels
# If this cell failed due to wget probelm, you can put the link below into your␣
↪browser to download the file directly
# Put the downloaded file under the same directory as this jupyter notebook
!wget https://s3.amazonaws.com/deep-learning-models/image-models/
↪imagenet_class_index.json

class_idx = json.load(open("imagenet_class_index.json"))
idx2label = {k:class_idx[str(k)][1] for k in range(len(class_idx))}
idx2label
```

```
--2024-03-20 13:42:07--  https://s3.amazonaws.com/deep-learning-models/image-
models/imagenet_class_index.json
Resolving s3.amazonaws.com (s3.amazonaws.com)… 52.217.200.40, 54.231.172.152,
52.216.113.205, …
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.217.200.40|:443…
connected.
HTTP request sent, awaiting response… 200 OK
Length: 35363 (35K) [application/octet-stream]
Saving to: ‘imagenet_class_index.json.1’

imagenet_class_inde 100%[===================>]  34.53K  --.-KB/s    in 0.03s

2024-03-20 13:42:07 (1.29 MB/s) - ‘imagenet_class_index.json.1’ saved
[35363/35363]
```

[4]:
```
{0: 'tench',
 1: 'goldfish',
 2: 'great_white_shark',
 3: 'tiger_shark',
 4: 'hammerhead',
 5: 'electric_ray',
 6: 'stingray',
 7: 'cock',
 8: 'hen',
 9: 'ostrich',
 10: 'brambling',
 11: 'goldfinch',
 12: 'house_finch',
 13: 'junco',
 14: 'indigo_bunting',
 15: 'robin',
 16: 'bulbul',
 17: 'jay',
 18: 'magpie',
 19: 'chickadee',
```

```
20: 'water_ouzel',
21: 'kite',
22: 'bald_eagle',
23: 'vulture',
24: 'great_grey_owl',
25: 'European_fire_salamander',
26: 'common_newt',
27: 'eft',
28: 'spotted_salamander',
29: 'axolotl',
30: 'bullfrog',
31: 'tree_frog',
32: 'tailed_frog',
33: 'loggerhead',
34: 'leatherback_turtle',
35: 'mud_turtle',
36: 'terrapin',
37: 'box_turtle',
38: 'banded_gecko',
39: 'common_iguana',
40: 'American_chameleon',
41: 'whiptail',
42: 'agama',
43: 'frilled_lizard',
44: 'alligator_lizard',
45: 'Gila_monster',
46: 'green_lizard',
47: 'African_chameleon',
48: 'Komodo_dragon',
49: 'African_crocodile',
50: 'American_alligator',
51: 'triceratops',
52: 'thunder_snake',
53: 'ringneck_snake',
54: 'hognose_snake',
55: 'green_snake',
56: 'king_snake',
57: 'garter_snake',
58: 'water_snake',
59: 'vine_snake',
60: 'night_snake',
61: 'boa_constrictor',
62: 'rock_python',
63: 'Indian_cobra',
64: 'green_mamba',
65: 'sea_snake',
66: 'horned_viper',
```

```
67: 'diamondback',
68: 'sidewinder',
69: 'trilobite',
70: 'harvestman',
71: 'scorpion',
72: 'black_and_gold_garden_spider',
73: 'barn_spider',
74: 'garden_spider',
75: 'black_widow',
76: 'tarantula',
77: 'wolf_spider',
78: 'tick',
79: 'centipede',
80: 'black_grouse',
81: 'ptarmigan',
82: 'ruffed_grouse',
83: 'prairie_chicken',
84: 'peacock',
85: 'quail',
86: 'partridge',
87: 'African_grey',
88: 'macaw',
89: 'sulphur-crested_cockatoo',
90: 'lorikeet',
91: 'coucal',
92: 'bee_eater',
93: 'hornbill',
94: 'hummingbird',
95: 'jacamar',
96: 'toucan',
97: 'drake',
98: 'red-breasted_merganser',
99: 'goose',
100: 'black_swan',
101: 'tusker',
102: 'echidna',
103: 'platypus',
104: 'wallaby',
105: 'koala',
106: 'wombat',
107: 'jellyfish',
108: 'sea_anemone',
109: 'brain_coral',
110: 'flatworm',
111: 'nematode',
112: 'conch',
113: 'snail',
```

```
114: 'slug',
115: 'sea_slug',
116: 'chiton',
117: 'chambered_nautilus',
118: 'Dungeness_crab',
119: 'rock_crab',
120: 'fiddler_crab',
121: 'king_crab',
122: 'American_lobster',
123: 'spiny_lobster',
124: 'crayfish',
125: 'hermit_crab',
126: 'isopod',
127: 'white_stork',
128: 'black_stork',
129: 'spoonbill',
130: 'flamingo',
131: 'little_blue_heron',
132: 'American_egret',
133: 'bittern',
134: 'crane',
135: 'limpkin',
136: 'European_gallinule',
137: 'American_coot',
138: 'bustard',
139: 'ruddy_turnstone',
140: 'red-backed_sandpiper',
141: 'redshank',
142: 'dowitcher',
143: 'oystercatcher',
144: 'pelican',
145: 'king_penguin',
146: 'albatross',
147: 'grey_whale',
148: 'killer_whale',
149: 'dugong',
150: 'sea_lion',
151: 'Chihuahua',
152: 'Japanese_spaniel',
153: 'Maltese_dog',
154: 'Pekinese',
155: 'Shih-Tzu',
156: 'Blenheim_spaniel',
157: 'papillon',
158: 'toy_terrier',
159: 'Rhodesian_ridgeback',
160: 'Afghan_hound',
```

```
161: 'basset',
162: 'beagle',
163: 'bloodhound',
164: 'bluetick',
165: 'black-and-tan_coonhound',
166: 'Walker_hound',
167: 'English_foxhound',
168: 'redbone',
169: 'borzoi',
170: 'Irish_wolfhound',
171: 'Italian_greyhound',
172: 'whippet',
173: 'Ibizan_hound',
174: 'Norwegian_elkhound',
175: 'otterhound',
176: 'Saluki',
177: 'Scottish_deerhound',
178: 'Weimaraner',
179: 'Staffordshire_bullterrier',
180: 'American_Staffordshire_terrier',
181: 'Bedlington_terrier',
182: 'Border_terrier',
183: 'Kerry_blue_terrier',
184: 'Irish_terrier',
185: 'Norfolk_terrier',
186: 'Norwich_terrier',
187: 'Yorkshire_terrier',
188: 'wire-haired_fox_terrier',
189: 'Lakeland_terrier',
190: 'Sealyham_terrier',
191: 'Airedale',
192: 'cairn',
193: 'Australian_terrier',
194: 'Dandie_Dinmont',
195: 'Boston_bull',
196: 'miniature_schnauzer',
197: 'giant_schnauzer',
198: 'standard_schnauzer',
199: 'Scotch_terrier',
200: 'Tibetan_terrier',
201: 'silky_terrier',
202: 'soft-coated_wheaten_terrier',
203: 'West_Highland_white_terrier',
204: 'Lhasa',
205: 'flat-coated_retriever',
206: 'curly-coated_retriever',
207: 'golden_retriever',
```

```
208: 'Labrador_retriever',
209: 'Chesapeake_Bay_retriever',
210: 'German_short-haired_pointer',
211: 'vizsla',
212: 'English_setter',
213: 'Irish_setter',
214: 'Gordon_setter',
215: 'Brittany_spaniel',
216: 'clumber',
217: 'English_springer',
218: 'Welsh_springer_spaniel',
219: 'cocker_spaniel',
220: 'Sussex_spaniel',
221: 'Irish_water_spaniel',
222: 'kuvasz',
223: 'schipperke',
224: 'groenendael',
225: 'malinois',
226: 'briard',
227: 'kelpie',
228: 'komondor',
229: 'Old_English_sheepdog',
230: 'Shetland_sheepdog',
231: 'collie',
232: 'Border_collie',
233: 'Bouvier_des_Flandres',
234: 'Rottweiler',
235: 'German_shepherd',
236: 'Doberman',
237: 'miniature_pinscher',
238: 'Greater_Swiss_Mountain_dog',
239: 'Bernese_mountain_dog',
240: 'Appenzeller',
241: 'EntleBucher',
242: 'boxer',
243: 'bull_mastiff',
244: 'Tibetan_mastiff',
245: 'French_bulldog',
246: 'Great_Dane',
247: 'Saint_Bernard',
248: 'Eskimo_dog',
249: 'malamute',
250: 'Siberian_husky',
251: 'dalmatian',
252: 'affenpinscher',
253: 'basenji',
254: 'pug',
```

```
255: 'Leonberg',
256: 'Newfoundland',
257: 'Great_Pyrenees',
258: 'Samoyed',
259: 'Pomeranian',
260: 'chow',
261: 'keeshond',
262: 'Brabancon_griffon',
263: 'Pembroke',
264: 'Cardigan',
265: 'toy_poodle',
266: 'miniature_poodle',
267: 'standard_poodle',
268: 'Mexican_hairless',
269: 'timber_wolf',
270: 'white_wolf',
271: 'red_wolf',
272: 'coyote',
273: 'dingo',
274: 'dhole',
275: 'African_hunting_dog',
276: 'hyena',
277: 'red_fox',
278: 'kit_fox',
279: 'Arctic_fox',
280: 'grey_fox',
281: 'tabby',
282: 'tiger_cat',
283: 'Persian_cat',
284: 'Siamese_cat',
285: 'Egyptian_cat',
286: 'cougar',
287: 'lynx',
288: 'leopard',
289: 'snow_leopard',
290: 'jaguar',
291: 'lion',
292: 'tiger',
293: 'cheetah',
294: 'brown_bear',
295: 'American_black_bear',
296: 'ice_bear',
297: 'sloth_bear',
298: 'mongoose',
299: 'meerkat',
300: 'tiger_beetle',
301: 'ladybug',
```

```
302: 'ground_beetle',
303: 'long-horned_beetle',
304: 'leaf_beetle',
305: 'dung_beetle',
306: 'rhinoceros_beetle',
307: 'weevil',
308: 'fly',
309: 'bee',
310: 'ant',
311: 'grasshopper',
312: 'cricket',
313: 'walking_stick',
314: 'cockroach',
315: 'mantis',
316: 'cicada',
317: 'leafhopper',
318: 'lacewing',
319: 'dragonfly',
320: 'damselfly',
321: 'admiral',
322: 'ringlet',
323: 'monarch',
324: 'cabbage_butterfly',
325: 'sulphur_butterfly',
326: 'lycaenid',
327: 'starfish',
328: 'sea_urchin',
329: 'sea_cucumber',
330: 'wood_rabbit',
331: 'hare',
332: 'Angora',
333: 'hamster',
334: 'porcupine',
335: 'fox_squirrel',
336: 'marmot',
337: 'beaver',
338: 'guinea_pig',
339: 'sorrel',
340: 'zebra',
341: 'hog',
342: 'wild_boar',
343: 'warthog',
344: 'hippopotamus',
345: 'ox',
346: 'water_buffalo',
347: 'bison',
348: 'ram',
```

```
349: 'bighorn',
350: 'ibex',
351: 'hartebeest',
352: 'impala',
353: 'gazelle',
354: 'Arabian_camel',
355: 'llama',
356: 'weasel',
357: 'mink',
358: 'polecat',
359: 'black-footed_ferret',
360: 'otter',
361: 'skunk',
362: 'badger',
363: 'armadillo',
364: 'three-toed_sloth',
365: 'orangutan',
366: 'gorilla',
367: 'chimpanzee',
368: 'gibbon',
369: 'siamang',
370: 'guenon',
371: 'patas',
372: 'baboon',
373: 'macaque',
374: 'langur',
375: 'colobus',
376: 'proboscis_monkey',
377: 'marmoset',
378: 'capuchin',
379: 'howler_monkey',
380: 'titi',
381: 'spider_monkey',
382: 'squirrel_monkey',
383: 'Madagascar_cat',
384: 'indri',
385: 'Indian_elephant',
386: 'African_elephant',
387: 'lesser_panda',
388: 'giant_panda',
389: 'barracouta',
390: 'eel',
391: 'coho',
392: 'rock_beauty',
393: 'anemone_fish',
394: 'sturgeon',
395: 'gar',
```

```
396: 'lionfish',
397: 'puffer',
398: 'abacus',
399: 'abaya',
400: 'academic_gown',
401: 'accordion',
402: 'acoustic_guitar',
403: 'aircraft_carrier',
404: 'airliner',
405: 'airship',
406: 'altar',
407: 'ambulance',
408: 'amphibian',
409: 'analog_clock',
410: 'apiary',
411: 'apron',
412: 'ashcan',
413: 'assault_rifle',
414: 'backpack',
415: 'bakery',
416: 'balance_beam',
417: 'balloon',
418: 'ballpoint',
419: 'Band_Aid',
420: 'banjo',
421: 'bannister',
422: 'barbell',
423: 'barber_chair',
424: 'barbershop',
425: 'barn',
426: 'barometer',
427: 'barrel',
428: 'barrow',
429: 'baseball',
430: 'basketball',
431: 'bassinet',
432: 'bassoon',
433: 'bathing_cap',
434: 'bath_towel',
435: 'bathtub',
436: 'beach_wagon',
437: 'beacon',
438: 'beaker',
439: 'bearskin',
440: 'beer_bottle',
441: 'beer_glass',
442: 'bell_cote',
```

```
443: 'bib',
444: 'bicycle-built-for-two',
445: 'bikini',
446: 'binder',
447: 'binoculars',
448: 'birdhouse',
449: 'boathouse',
450: 'bobsled',
451: 'bolo_tie',
452: 'bonnet',
453: 'bookcase',
454: 'bookshop',
455: 'bottlecap',
456: 'bow',
457: 'bow_tie',
458: 'brass',
459: 'brassiere',
460: 'breakwater',
461: 'breastplate',
462: 'broom',
463: 'bucket',
464: 'buckle',
465: 'bulletproof_vest',
466: 'bullet_train',
467: 'butcher_shop',
468: 'cab',
469: 'caldron',
470: 'candle',
471: 'cannon',
472: 'canoe',
473: 'can_opener',
474: 'cardigan',
475: 'car_mirror',
476: 'carousel',
477: "carpenter's_kit",
478: 'carton',
479: 'car_wheel',
480: 'cash_machine',
481: 'cassette',
482: 'cassette_player',
483: 'castle',
484: 'catamaran',
485: 'CD_player',
486: 'cello',
487: 'cellular_telephone',
488: 'chain',
489: 'chainlink_fence',
```

```
490: 'chain_mail',
491: 'chain_saw',
492: 'chest',
493: 'chiffonier',
494: 'chime',
495: 'china_cabinet',
496: 'Christmas_stocking',
497: 'church',
498: 'cinema',
499: 'cleaver',
500: 'cliff_dwelling',
501: 'cloak',
502: 'clog',
503: 'cocktail_shaker',
504: 'coffee_mug',
505: 'coffeepot',
506: 'coil',
507: 'combination_lock',
508: 'computer_keyboard',
509: 'confectionery',
510: 'container_ship',
511: 'convertible',
512: 'corkscrew',
513: 'cornet',
514: 'cowboy_boot',
515: 'cowboy_hat',
516: 'cradle',
517: 'crane',
518: 'crash_helmet',
519: 'crate',
520: 'crib',
521: 'Crock_Pot',
522: 'croquet_ball',
523: 'crutch',
524: 'cuirass',
525: 'dam',
526: 'desk',
527: 'desktop_computer',
528: 'dial_telephone',
529: 'diaper',
530: 'digital_clock',
531: 'digital_watch',
532: 'dining_table',
533: 'dishrag',
534: 'dishwasher',
535: 'disk_brake',
536: 'dock',
```

```
537: 'dogsled',
538: 'dome',
539: 'doormat',
540: 'drilling_platform',
541: 'drum',
542: 'drumstick',
543: 'dumbbell',
544: 'Dutch_oven',
545: 'electric_fan',
546: 'electric_guitar',
547: 'electric_locomotive',
548: 'entertainment_center',
549: 'envelope',
550: 'espresso_maker',
551: 'face_powder',
552: 'feather_boa',
553: 'file',
554: 'fireboat',
555: 'fire_engine',
556: 'fire_screen',
557: 'flagpole',
558: 'flute',
559: 'folding_chair',
560: 'football_helmet',
561: 'forklift',
562: 'fountain',
563: 'fountain_pen',
564: 'four-poster',
565: 'freight_car',
566: 'French_horn',
567: 'frying_pan',
568: 'fur_coat',
569: 'garbage_truck',
570: 'gasmask',
571: 'gas_pump',
572: 'goblet',
573: 'go-kart',
574: 'golf_ball',
575: 'golfcart',
576: 'gondola',
577: 'gong',
578: 'gown',
579: 'grand_piano',
580: 'greenhouse',
581: 'grille',
582: 'grocery_store',
583: 'guillotine',
```

```
584: 'hair_slide',
585: 'hair_spray',
586: 'half_track',
587: 'hammer',
588: 'hamper',
589: 'hand_blower',
590: 'hand-held_computer',
591: 'handkerchief',
592: 'hard_disc',
593: 'harmonica',
594: 'harp',
595: 'harvester',
596: 'hatchet',
597: 'holster',
598: 'home_theater',
599: 'honeycomb',
600: 'hook',
601: 'hoopskirt',
602: 'horizontal_bar',
603: 'horse_cart',
604: 'hourglass',
605: 'iPod',
606: 'iron',
607: "jack-o'-lantern",
608: 'jean',
609: 'jeep',
610: 'jersey',
611: 'jigsaw_puzzle',
612: 'jinrikisha',
613: 'joystick',
614: 'kimono',
615: 'knee_pad',
616: 'knot',
617: 'lab_coat',
618: 'ladle',
619: 'lampshade',
620: 'laptop',
621: 'lawn_mower',
622: 'lens_cap',
623: 'letter_opener',
624: 'library',
625: 'lifeboat',
626: 'lighter',
627: 'limousine',
628: 'liner',
629: 'lipstick',
630: 'Loafer',
```

```
631: 'lotion',
632: 'loudspeaker',
633: 'loupe',
634: 'lumbermill',
635: 'magnetic_compass',
636: 'mailbag',
637: 'mailbox',
638: 'maillot',
639: 'maillot',
640: 'manhole_cover',
641: 'maraca',
642: 'marimba',
643: 'mask',
644: 'matchstick',
645: 'maypole',
646: 'maze',
647: 'measuring_cup',
648: 'medicine_chest',
649: 'megalith',
650: 'microphone',
651: 'microwave',
652: 'military_uniform',
653: 'milk_can',
654: 'minibus',
655: 'miniskirt',
656: 'minivan',
657: 'missile',
658: 'mitten',
659: 'mixing_bowl',
660: 'mobile_home',
661: 'Model_T',
662: 'modem',
663: 'monastery',
664: 'monitor',
665: 'moped',
666: 'mortar',
667: 'mortarboard',
668: 'mosque',
669: 'mosquito_net',
670: 'motor_scooter',
671: 'mountain_bike',
672: 'mountain_tent',
673: 'mouse',
674: 'mousetrap',
675: 'moving_van',
676: 'muzzle',
677: 'nail',
```

```
678: 'neck_brace',
679: 'necklace',
680: 'nipple',
681: 'notebook',
682: 'obelisk',
683: 'oboe',
684: 'ocarina',
685: 'odometer',
686: 'oil_filter',
687: 'organ',
688: 'oscilloscope',
689: 'overskirt',
690: 'oxcart',
691: 'oxygen_mask',
692: 'packet',
693: 'paddle',
694: 'paddlewheel',
695: 'padlock',
696: 'paintbrush',
697: 'pajama',
698: 'palace',
699: 'panpipe',
700: 'paper_towel',
701: 'parachute',
702: 'parallel_bars',
703: 'park_bench',
704: 'parking_meter',
705: 'passenger_car',
706: 'patio',
707: 'pay-phone',
708: 'pedestal',
709: 'pencil_box',
710: 'pencil_sharpener',
711: 'perfume',
712: 'Petri_dish',
713: 'photocopier',
714: 'pick',
715: 'pickelhaube',
716: 'picket_fence',
717: 'pickup',
718: 'pier',
719: 'piggy_bank',
720: 'pill_bottle',
721: 'pillow',
722: 'ping-pong_ball',
723: 'pinwheel',
724: 'pirate',
```

```
725: 'pitcher',
726: 'plane',
727: 'planetarium',
728: 'plastic_bag',
729: 'plate_rack',
730: 'plow',
731: 'plunger',
732: 'Polaroid_camera',
733: 'pole',
734: 'police_van',
735: 'poncho',
736: 'pool_table',
737: 'pop_bottle',
738: 'pot',
739: "potter's_wheel",
740: 'power_drill',
741: 'prayer_rug',
742: 'printer',
743: 'prison',
744: 'projectile',
745: 'projector',
746: 'puck',
747: 'punching_bag',
748: 'purse',
749: 'quill',
750: 'quilt',
751: 'racer',
752: 'racket',
753: 'radiator',
754: 'radio',
755: 'radio_telescope',
756: 'rain_barrel',
757: 'recreational_vehicle',
758: 'reel',
759: 'reflex_camera',
760: 'refrigerator',
761: 'remote_control',
762: 'restaurant',
763: 'revolver',
764: 'rifle',
765: 'rocking_chair',
766: 'rotisserie',
767: 'rubber_eraser',
768: 'rugby_ball',
769: 'rule',
770: 'running_shoe',
771: 'safe',
```

```
772: 'safety_pin',
773: 'saltshaker',
774: 'sandal',
775: 'sarong',
776: 'sax',
777: 'scabbard',
778: 'scale',
779: 'school_bus',
780: 'schooner',
781: 'scoreboard',
782: 'screen',
783: 'screw',
784: 'screwdriver',
785: 'seat_belt',
786: 'sewing_machine',
787: 'shield',
788: 'shoe_shop',
789: 'shoji',
790: 'shopping_basket',
791: 'shopping_cart',
792: 'shovel',
793: 'shower_cap',
794: 'shower_curtain',
795: 'ski',
796: 'ski_mask',
797: 'sleeping_bag',
798: 'slide_rule',
799: 'sliding_door',
800: 'slot',
801: 'snorkel',
802: 'snowmobile',
803: 'snowplow',
804: 'soap_dispenser',
805: 'soccer_ball',
806: 'sock',
807: 'solar_dish',
808: 'sombrero',
809: 'soup_bowl',
810: 'space_bar',
811: 'space_heater',
812: 'space_shuttle',
813: 'spatula',
814: 'speedboat',
815: 'spider_web',
816: 'spindle',
817: 'sports_car',
818: 'spotlight',
```

```
819: 'stage',
820: 'steam_locomotive',
821: 'steel_arch_bridge',
822: 'steel_drum',
823: 'stethoscope',
824: 'stole',
825: 'stone_wall',
826: 'stopwatch',
827: 'stove',
828: 'strainer',
829: 'streetcar',
830: 'stretcher',
831: 'studio_couch',
832: 'stupa',
833: 'submarine',
834: 'suit',
835: 'sundial',
836: 'sunglass',
837: 'sunglasses',
838: 'sunscreen',
839: 'suspension_bridge',
840: 'swab',
841: 'sweatshirt',
842: 'swimming_trunks',
843: 'swing',
844: 'switch',
845: 'syringe',
846: 'table_lamp',
847: 'tank',
848: 'tape_player',
849: 'teapot',
850: 'teddy',
851: 'television',
852: 'tennis_ball',
853: 'thatch',
854: 'theater_curtain',
855: 'thimble',
856: 'thresher',
857: 'throne',
858: 'tile_roof',
859: 'toaster',
860: 'tobacco_shop',
861: 'toilet_seat',
862: 'torch',
863: 'totem_pole',
864: 'tow_truck',
865: 'toyshop',
```

```
866: 'tractor',
867: 'trailer_truck',
868: 'tray',
869: 'trench_coat',
870: 'tricycle',
871: 'trimaran',
872: 'tripod',
873: 'triumphal_arch',
874: 'trolleybus',
875: 'trombone',
876: 'tub',
877: 'turnstile',
878: 'typewriter_keyboard',
879: 'umbrella',
880: 'unicycle',
881: 'upright',
882: 'vacuum',
883: 'vase',
884: 'vault',
885: 'velvet',
886: 'vending_machine',
887: 'vestment',
888: 'viaduct',
889: 'violin',
890: 'volleyball',
891: 'waffle_iron',
892: 'wall_clock',
893: 'wallet',
894: 'wardrobe',
895: 'warplane',
896: 'washbasin',
897: 'washer',
898: 'water_bottle',
899: 'water_jug',
900: 'water_tower',
901: 'whiskey_jug',
902: 'whistle',
903: 'wig',
904: 'window_screen',
905: 'window_shade',
906: 'Windsor_tie',
907: 'wine_bottle',
908: 'wing',
909: 'wok',
910: 'wooden_spoon',
911: 'wool',
912: 'worm_fence',
```

```
913: 'wreck',
914: 'yawl',
915: 'yurt',
916: 'web_site',
917: 'comic_book',
918: 'crossword_puzzle',
919: 'street_sign',
920: 'traffic_light',
921: 'book_jacket',
922: 'menu',
923: 'plate',
924: 'guacamole',
925: 'consomme',
926: 'hot_pot',
927: 'trifle',
928: 'ice_cream',
929: 'ice_lolly',
930: 'French_loaf',
931: 'bagel',
932: 'pretzel',
933: 'cheeseburger',
934: 'hotdog',
935: 'mashed_potato',
936: 'head_cabbage',
937: 'broccoli',
938: 'cauliflower',
939: 'zucchini',
940: 'spaghetti_squash',
941: 'acorn_squash',
942: 'butternut_squash',
943: 'cucumber',
944: 'artichoke',
945: 'bell_pepper',
946: 'cardoon',
947: 'mushroom',
948: 'Granny_Smith',
949: 'strawberry',
950: 'orange',
951: 'lemon',
952: 'fig',
953: 'pineapple',
954: 'banana',
955: 'jackfruit',
956: 'custard_apple',
957: 'pomegranate',
958: 'hay',
959: 'carbonara',
```

```
960: 'chocolate_sauce',
961: 'dough',
962: 'meat_loaf',
963: 'pizza',
964: 'potpie',
965: 'burrito',
966: 'red_wine',
967: 'espresso',
968: 'cup',
969: 'eggnog',
970: 'alp',
971: 'bubble',
972: 'cliff',
973: 'coral_reef',
974: 'geyser',
975: 'lakeside',
976: 'promontory',
977: 'sandbar',
978: 'seashore',
979: 'valley',
980: 'volcano',
981: 'ballplayer',
982: 'groom',
983: 'scuba_diver',
984: 'rapeseed',
985: 'daisy',
986: "yellow_lady's_slipper",
987: 'corn',
988: 'acorn',
989: 'hip',
990: 'buckeye',
991: 'coral_fungus',
992: 'agaric',
993: 'gyromitra',
994: 'stinkhorn',
995: 'earthstar',
996: 'hen-of-the-woods',
997: 'bolete',
998: 'ear',
999: 'toilet_tissue'}
```

1.1.1 Helper Functions

Our pretrained model was trained on images that had been preprocessed by subtracting the per-color mean and dividing by the per-color standard deviation. We define a few helper functions for performing and undoing this preprocessing. You don't need to do anything in this cell.

[5]:
```
def preprocess(img, size=224):
    transform = T.Compose([
        T.Resize((size, size)),
        T.ToTensor(),
        T.Normalize(mean=SQUEEZENET_MEAN.tolist(),
              std=SQUEEZENET_STD.tolist()),
        T.Lambda(lambda x: x[None]),
    ])
    return transform(img)

def deprocess(img, should_rescale=True):
    transform = T.Compose([
        T.Lambda(lambda x: x[0]),
        T.Normalize(mean=[0, 0, 0], std=(1.0 / SQUEEZENET_STD).tolist()),
        T.Normalize(mean=(-SQUEEZENET_MEAN).tolist(), std=[1, 1, 1]),
        T.Lambda(rescale) if should_rescale else T.Lambda(lambda x: x),
        T.ToPILImage(),
    ])
    return transform(img)

def rescale(x):
    low, high = x.min(), x.max()
    x_rescaled = (x - low) / (high - low)
    return x_rescaled

def blur_image(X, sigma=1):
    X_np = X.cpu().clone().numpy()
    X_np = gaussian_filter1d(X_np, sigma, axis=2)
    X_np = gaussian_filter1d(X_np, sigma, axis=3)
    X.copy_(torch.Tensor(X_np).type_as(X))
    return X
```

1.2 Task 6 - Pre-trained Convolution Network

In order to get a better sense of the classification decisions made by convolutional networks, your job is now to experiment by running whatever images you want through a model pretrained on ImageNet. These can be images from your own photo collection, from the internet, or somewhere else but they **should belong to one of the ImageNet classes**. Look at the `idx2label` dictionary for all the ImageNetclasses.

You need to find: 1. One image (`img1`) where the SqueezeNet model gives reasonable predictions, and produces a category label that seems to correctly describe the content of the image 2. One image (`img2`) where the SqueezeNet model gives unreasonable predictions, and produces a category label that does not correctly describe the content of the image.

You can upload images in Colab by using the upload button on the top left. For more details about using Colab, please see our Colab tutorial.

[17]:

```
###############################################################################
# TODO: Upload your image and run the forward pass to get the ImageNet class. #
# This code will crash when you run it, since the maxresdefault.jpg image is  #
# not found. You should upload your own images to the Colab notebook and edit #
# these lines to load your own image.                                         #
###############################################################################
img1 = Image.open('cup1.png').convert('RGB')
img2 = Image.open('cup2.png').convert('RGB')
names = ['cup1', 'cup2']
##############################################################################
#               END OF YOUR CODE                                             #
##############################################################################
for i, img in enumerate([img1, img2]):
    X = preprocess(img).to(device)
    pred_class = torch.argmax(model(X)).item()
    plt.figure(figsize=(6,8))
    plt.imshow(img)
    plt.title('Predicted Class: %s' % idx2label[pred_class])
    plt.axis('off')
    plt.savefig(f'{names[i]}_pred.jpg')
    plt.show()
```

Predicted Class: cup

Predicted Class: measuring_cup