In [ ]:
```
#Is It Bread? Grain Identifier
#By: Alicia Jin, Alice Liu, Jemima Chong & Bethel Yared

#Training a convolutional neural network to classify images of bread into specfic catagories
```

Colab Link: https://colab.research.google.com/drive/1pkFJD7-7jm_adH-XucnpVEmR2OimRc_x?usp=sharing

In [ ]:
```
#importing libraries
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

#useful libraries - extra options
import time
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
from torchvision import datasets, models, transforms
import time
import os
import matplotlib.pyplot as plt # Plotting graphs
from torch.utils.data import TensorDataset
```

In [ ]:
```
#Mounting Google Drive as many files will be generated
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)
```

```
Mounted at /content/gdrive
```

In [ ]:
```
#for splitting folders into train, validation and test set
! pip install split-folders
import splitfolders
```

```
Collecting split-folders
  Downloading split_folders-0.5.1-py3-none-any.whl (8.4 kB)
Installing collected packages: split-folders
Successfully installed split-folders-0.5.1
```

In [ ]:
```
#Defining the path to the dataset
data_directory = '/content/gdrive/MyDrive/APS360 Project Group/Data/Images'
#test data will be 80% of all data, validaion and test 10% each
splitfolders.ratio(data_directory, 'all_data_splitted', seed=1337, ratio = (0.8, 0.1, 0.1))
```

```
Copying files: 1883 files [01:02, 30.35 files/s]
```

In [ ]:
```
new_path = '/content/all_data_splitted'
train_dir = os.path.join(new_path, 'train/')
test_dir = os.path.join(new_path, 'test/')
val_dir = os.path.join(new_path, 'val/')


#Transform the images
data_transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize the images to a consistent size
    transforms.ToTensor(),  # Convert images to PyTorch tensors
])


train_data = datasets.ImageFolder(train_dir, transform=data_transform)
val_data = datasets.ImageFolder(val_dir, transform=data_transform)
test_data = datasets.ImageFolder(test_dir, transform=data_transform)

#Classes: A list of strings denoting the bread depicted in each image
classes = ('Bagel', 'Baguette', 'Brioche', 'Challah', 'Cheese Bread', 'Ciabatta', 'Cinnamon Raisin', 'Focaccia', 'Wheat Grain Bread', 'White Bread')

# define dataloader parameters
batch_size  = 256
num_workers = 0

# prepare data loaders
train_loader = torch.utils.data.DataLoader(train_data, batch_size=batch_size,
                                           num_workers=num_workers, shuffle=True)
val_loader = torch.utils.data.DataLoader(val_data, batch_size=batch_size,
                                         num_workers=num_workers, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_data, batch_size=batch_size,
                                          num_workers=num_workers, shuffle=True)
```

In [ ]:
```
#for reference, total number of images in each set
print(len(train_data))
print(len(test_data))
print(len(val_data))
```

```
1500
196
184
```

In [ ]:
```
#Display some data examples
dataiter = iter(train_loader)
images, labels = next(dataiter)
images = images.numpy()

fig = plt.figure(figsize=(25, 4))
for idx in np.arange(10):
    ax = fig.add_subplot(2, int(10/2), idx+1, xticks=[], yticks=[])
    plt.imshow(np.transpose(images[idx], (1, 2, 0)))
    ax.set_title(classes[labels[idx]])
```

Baseline Model

The Support Vector Machine (SVM) will be used to validate the performace of the primary model. Method is referenced from https://medium.com/analytics-vidhya/image-classification-using-machine-learning-support-vector-machine-svm-dc7a0ec92e01, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html, and tutorial 2.

In [ ]:
```
from sys import float_info
import pandas as pd
import os
from skimage.io import imread
import numpy as np
import matplotlib.pyplot as plt
from skimage.transform import rescale, resize, downscale_local_mean

from PIL import Image
Image.MAX_IMAGE_PIXELS = None
import cv2

flat_data_arr = []
target_arr = []

classes = ('Bagel', 'Baguette', 'Brioche', 'Challah', 'Cheese Bread', 'Ciabatta', 'Cinnamon Raisin', 'Focaccia', 'Wheat Grain Bread', 'White Bread')

#load all data

data_directory = '/content/gdrive/MyDrive/APS360 Project Group/Data/Images'

for n in classes:
  print(f'loading... class : {n}')
  path = os.path.join(data_directory, n)
  for img in os.listdir(path):
    img_array = np.array(cv2.imread(os.path.join(path,img))) #load image data into img_array
    img_array = img_array.astype('float64')
    img_resized = resize(img_array, (20,20,3)) #rezise all images to 25*25 in RGB
    flat_data_arr.append(img_resized.flatten()) #add flattened images to flat_data_arr
    target_arr.append(classes.index(n)) #add classes as numbers to target_arr
  print(f'loaded class:{n} succesfully')
```

In [ ]:
```
#convert flat_data_arr and target_arr to numpy arrays
flat_data = np.array(flat_data_arr)
target = np.array(target_arr)

#define dataframe
df = pd.DataFrame(flat_data)
df = df.fillna(0) #relpce NaN values wiht 0
df['Target'] = target
x=df.iloc[:,:-1] #input data
y=df.iloc[:,-1] #output data
```

In [ ]:
```
#accelerate training
! pip install scikit-learn-intelex
from sklearnex import patch_sklearn
patch_sklearn()
from sklearn.svm import SVC
```

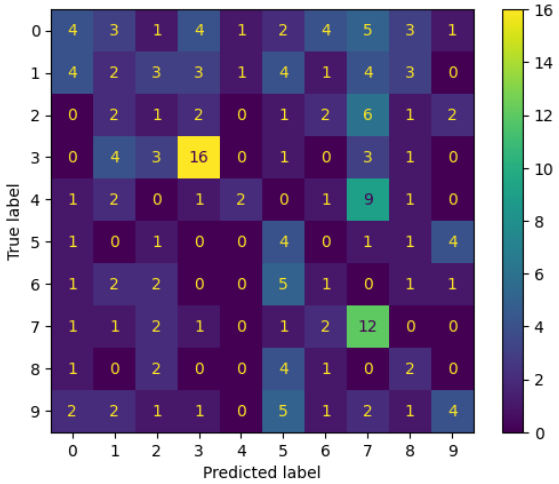
In [ ]:
```
#define svm model
from sklearn import svm
from sklearn.model_selection import GridSearchCV
param_grid={'C':[0.1,1,10,100],'gamma':[0.0001,0.001,0.1,1],'kernel':['rbf', 'poly']} #either rbf or poly kernal will be used for better accuracy
svc=svm.SVC(probability=True)
model=GridSearchCV(svc,param_grid, error_score='raise') #gridsearchcv chooses the best performing parameters defined in param_grid
```

In [ ]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,train_size = 0.8, shuffle = True) #split to 80% and 20%
x_val, x_test2, y_val, y_test2 = train_test_split(x_test, y_test, test_size=0.5, shuffle = True)#further split 20% into val and test

# normalize data between 0 and 1
from sklearn import preprocessing
norm_x_train = preprocessing.normalize(x_train)
norm_x_val = preprocessing.normalize(x_val)
norm_x_test2 = preprocessing.normalize(x_test2)
```

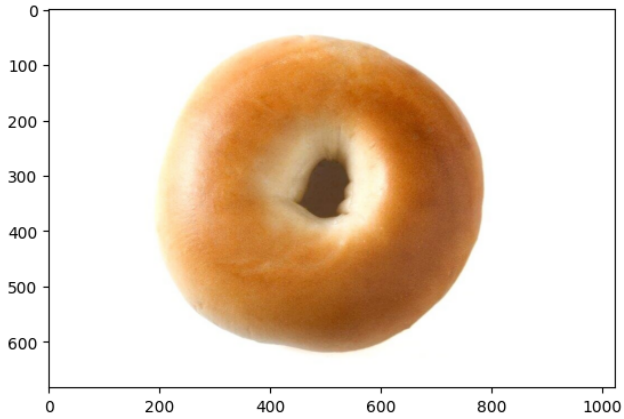
In [ ]:
```
# train in batches to reduce memory
batch_size = 251
#batch size should be divisible by length of train data for better accuracy, which ensures all classes are in each batch

#training loop to irerate through batches
for i in range(0, len(norm_x_train), batch_size):
     x_train_batch = norm_x_train[i:i+batch_size]
     y_train_batch = y_train[i:i+batch_size]
     #train model with tarining data
     model.fit(x_train_batch, y_train_batch)
```

In [ ]:
```
#validate and test model
y_pred_val = model.predict(norm_x_val) #prediction with model using validation data
y_pred_test = model.predict(norm_x_test2) #prediction with model using test data

from sklearn import metrics
from sklearn.metrics import accuracy_score
#obtain accuracy
val_accuracy = accuracy_score(y_val, y_pred_val) #compare accuracy of real vs predicted labels
test_accuracy = accuracy_score(y_test2, y_pred_test)
print(f'The validation accuracy for this model is {val_accuracy*100}%')
print(f'The test accuracy for this model is {test_accuracy*100}%')

#plot confucion matrix for testing data
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test2, y_pred_test)
disp = ConfusionMatrixDisplay(confusion_matrix = cm)
disp.plot()
plt.show()
```

```
The validation accuracy for this model is 23.93617021276596%
The test accuracy for this model is 25.396825396825395%
```



In [ ]:
```
#qualitative results
#choose one random image and obtain file path to be predicted

test_image_path = '/content/gdrive/MyDrive/APS360 Project Group/Data/Images/Bagel/plain+bagel.jpg'
img=imread(test_image_path)
plt.imshow(img)
plt.show()
img_resize=resize(img,(20,20,3)) #has to be the same dimension as images fed into the model to predict the label
l=[img_resize.flatten()]
probability=model.predict_proba(l)
for ind,val in enumerate(classes):
    print(f'{val} = {probability[0][ind]*100}%') #predicted probability of classes the image is
print("The predicted image is : "+ classes[model.predict(l)[0]]) #predict the first most likely class of the image
```



```
Bagel = 8.161424174701217%
Baguette = 11.48045944535891%
Brioche = 22.86355032550705%
Challah = 19.290898862260615%
Cheese Bread = 5.722914256571526%
Ciabatta = 4.309797125733436%
Cinnamon Raisin = 8.09074177568979%
Focaccia = 5.056570326226335%
Wheat Grain Bread = 12.337912631605068%
White Bread = 2.685731076346051%
The predicted image is : Challah
```

Model

BreadNetAlex:

Input data that has been loaded with AlexNet features (256 layers), outputs data as 10 layers corresponding to 10 classes.

BreadNetCNN:

Input data that has been normalized and intensity channel removed (3 layers, RGB), outputs data as 10 layers corresponding to 10 classes.

In [ ]:
```
import os
import torchvision.models
alexnet = torchvision.models.alexnet(pretrained=True)

data_dir = '/content/gdrive/MyDrive/APS360 Project Group/Data/Alexnet features'
data_directory = '/content/gdrive/MyDrive/APS360 Project Group/Data/Images'
transform = transforms.Compose([transforms.ToTensor(), transforms.Resize((224,224))])
full_dataset = torchvision.datasets.ImageFolder(data_directory, transform=transform)

trainset, val_and_test = torch.utils.data.random_split(full_dataset, [int(0.8 * len(full_dataset)), int(0.2 * len(full_dataset))])
valset, testset = torch.utils.data.random_split(val_and_test, [int(0.5 * len(val_and_test)), int(0.5 * len(val_and_test))])

classes = ('Bagel', 'Baguette', 'Brioche', 'Challah', 'Cheese Bread', 'Ciabatta', 'Cinnamon Raisin', 'Focaccia', 'Wheat Grain Bread', 'White Bread')

from PIL import PngImagePlugin
LARGE_ENOUGH_NUMBER = 100
PngImagePlugin.MAX_TEXT_CHUNK = LARGE_ENOUGH_NUMBER * (1024**2)

n = 0
print('Training Data Converting...')
for i, label in iter(trainset):
  features = alexnet.features(i)
  features_tensor = torch.from_numpy(features.detach().numpy())

  folder_name = data_dir + '/train/' + str(classes[label])
  torch.save(features_tensor.squeeze(0), folder_name + '/' + str(n) + '.tensor')
  n += 1
print('Training Data Completed')

n = 0
print('Validation Data Converting...')
for i, label in iter(valset):
  features = alexnet.features(i)
  features_tensor = torch.from_numpy(features.detach().numpy())

  folder_name = data_dir + '/val/' + str(classes[label])
  torch.save(features_tensor.squeeze(0), folder_name + '/' + str(n) + '.tensor')
  n += 1
print('Validation Data Completed')

n = 0
print('Test Data Converting...')
for i, label in iter(testset):
  features = alexnet.features(i)
  features_tensor = torch.from_numpy(features.detach().numpy())

  folder_name = data_dir + '/test/' + str(classes[label])
  torch.save(features_tensor.squeeze(0), folder_name + '/' + str(n) + '.tensor')
  n += 1
print('Test Data Completed')
```

```
/usr/local/lib/python3.10/dist-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(
/usr/local/lib/python3.10/dist-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=AlexNet_Weights.IMAGENET1K_V1`. You can
also use `weights=AlexNet_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/alexnet-owt-7be5be79.pth" to /root/.cache/torch/hub/checkpoints/alexnet-owt-7be5be79.pth
100%|██████████| 233M/233M [00:04<00:00, 55.5MB/s]
Training Data Converting...
/usr/local/lib/python3.10/dist-packages/torchvision/transforms/functional.py:1603: UserWarning: The default value of the antialias parameter of all th
e resizing transforms (Resize(), RandomResizedCrop(), etc.) will change from None to True in v0.17, in order to be consistent across the PIL and Tenso
r backends. To suppress this warning, directly pass antialias=True (recommended, future default), antialias=None (current default, which means False f
or Tensors and True for PIL), or antialias=False (only works on Tensors - PIL will still use antialiasing). This also applies if you are using the inf
erence transforms from the models weights: update the call to weights.transforms(antialias=True).
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/PIL/Image.py:996: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA
images
  warnings.warn(
Training Data Completed
Validation Data Converting...
Validation Data Completed
Test Data Converting...
Test Data Completed
```

In [ ]:
```
#saved pretrained features to separate folders with .tensor extensions
train_features = torchvision.datasets.DatasetFolder(data_dir + '/train/', loader = torch.load, extensions = ('.tensor'))
val_features = torchvision.datasets.DatasetFolder(data_dir + '/val/', loader = torch.load, extensions = ('.tensor'))
test_features = torchvision.datasets.DatasetFolder(data_dir + '/test/', loader = torch.load, extensions = ('.tensor'))

batch_size  = 256
num_workers = 0

#loading pretrained data
train_loader_pre = torch.utils.data.DataLoader(train_features, batch_size = batch_size, num_workers = num_workers, shuffle = True)
val_loader_pre = torch.utils.data.DataLoader(val_features, batch_size = batch_size, num_workers = num_workers, shuffle = True)
test_loader_pre = torch.utils.data.DataLoader(test_features, batch_size = batch_size, num_workers = num_workers, shuffle = True)
```

In [ ]:
```
class BreadNetAlex(nn.Module):
  def __init__(self):
    super(BreadNetAlex, self).__init__()
    self.name = "Alex"
    self.conv1 = nn.Conv2d(256, 256, 3, padding=1)  #AlexNet 256 input
    self.pool = nn.MaxPool2d(2,2)
    self.fc1 = nn.Linear(256*3*3, 64)
    self.fc2 = nn.Linear(64, 10)   #10 classes of bread
```

```
  def forward(self, x):
    x = F.relu(self.conv1(x))
    x = self.pool(x)
    x = x.view(-1, 256*3*3)
    x = F.relu(self.fc1(x))
    x = self.fc2(x)
    x.squeeze(1)
    return x


#This model gives the option to not use pre-trained features in training, can compare more hyperparameter modifications with the features in AlexNet
class BreadNetCNN(nn.Module):
  def __init__(self):
    super(BreadNetCNN, self).__init__()
    self.name = "CNN"
    self.conv1 = nn.Conv2d(3, 5, 3, stride=2, padding=1)
    self.pool = nn.MaxPool2d(2,2)
    self.conv2 = nn.Conv2d(5, 5, 7, stride=2, padding=1)
    self.fc1 = nn.Linear(5*13*13, 64)
    self.fc2 = nn.Linear(64, 10)

  def forward(self,x):
    x = self.pool(F.relu(self.conv1(x)))
    x = self.pool(F.relu(self.conv2(x)))
    x = x.view(-1, 5*13*13)
    x = F.relu(self.fc1(x))
    x = self.fc2(x)
    x.squeeze(1)
    return x
```

In [ ]:

Possibly Helpful Functions - alice

get_accuracy(model, data_loader):

Depending on if model is pretrained with AlexNet or CNN, data_loader needs to be loaded according to the model selected. Returns decimal for accuracy.

get_loss(model, data_loader, criterion):

data_loader needs to correspond with model selected. Returns loss as a float.

In [ ]:
```
def get_accuracy(model, train_loader, valid_loader, train=False):
    if train:
        data = train_loader
    else:
        data = valid_loader

    correct = 0
    total = 0
    for imgs, labels in data:

        output = model(imgs)

        #select index with maximum prediction score
        pred = output.max(1, keepdim=True)[1]
        correct += pred.eq(labels.view_as(pred)).sum().item()
        total += imgs.shape[0]
    return correct / total

def get_loss(model, data_loader, criterion):
  total_loss = 0.0
  loss = 0.0
  eval_mod = model.eval()

  for i, labels in data_loader:
    output = eval_mod(i)
    loss = criterion(output, labels)
    total_loss += loss.item()

  return (float(total_loss)/len(data_loader))

def get_model_name(name, batch_size, learning_rate, epoch):
  path = 'model_{0}_bs{1}_lr{2}_epoch{3}'.format(name, batch_size, learning_rate, epoch)

  return path
```

In [ ]:

In [ ]:
```
#training (name, batch_size, learning_rate, epoch)

#Alice's training code, input is dataset NOT loader
#can input CNN or AlexNet dataset

def train(model, train_dataset, valid_dataset, batch_size=64, learning_rate=0.001, num_epochs=1):
    train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    valid_loader = torch.utils.data.DataLoader(valid_dataset, batch_size=batch_size, shuffle=True)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=learning_rate, momentum=0.9)
    print(len(train_loader))
    print(len(valid_loader))
    iters, losses, train_acc, val_acc = [], [], [], []

    # training
    epoch = 0 # the number of iterations
    for epoch in range(num_epochs):
        for imgs, labels in iter(train_loader):

            out = model(imgs)             # forward pass
            loss = criterion(out, labels) # compute the total loss
```

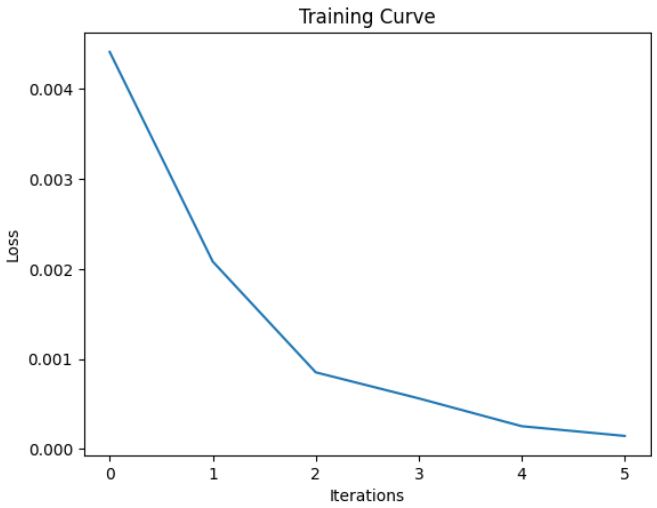
```
            loss.backward()               # backward pass (compute parameter updates)
            optimizer.step()              # make the updates for each parameter
            optimizer.zero_grad()         # a clean up step for PyTorch


        # save the current training information
        iters.append(epoch)
        losses.append(float(loss)/batch_size)             # compute *average* loss
        train_acc.append(get_accuracy(model, train_loader, valid_loader, train=True)) # compute training accuracy
        print("epoch number ", epoch+1, "accuracy: ",train_acc[epoch])
        val_acc.append(get_accuracy(model, train_loader, valid_loader, train=False))  # compute validation accuracy
        model_path = "model_{0}_bs{1}_lr{2}_epoch{3}".format(model.name, batch_size, learning_rate, epoch)
        torch.save(model.state_dict(), model_path)


    # plotting
    plt.title("Training Curve")
    plt.plot(iters, losses, label="Train")
    plt.xlabel("Iterations")
    plt.ylabel("Loss")
    plt.show()

    plt.title("Training Curve")
    plt.plot(iters, train_acc, label="Train")
    plt.plot(iters, val_acc, label="Validation")
    plt.xlabel("Iterations")
    plt.ylabel("Training Accuracy")
    plt.legend(loc='best')
    plt.show()

    print("Final Training Accuracy: {}".format(train_acc[-1]))
    print("Final Validation Accuracy: {}".format(val_acc[-1]))
```

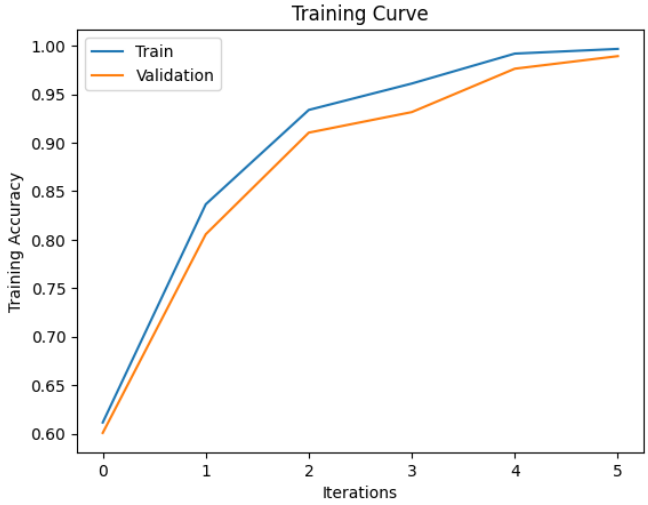
Training for Hyperparameters

adjusting the batch size, learning rate and number of epochs in order to select the best model in terms of accuracy.

In [ ]:
```
#training alexnet model
model = BreadNetAlex()

train(model, train_features, val_features, batch_size = 256, learning_rate = 0.01, num_epochs = 6)
```

```
33
4
epoch number  1 accuracy:  0.6114271879917685
epoch number  2 accuracy:  0.8365815276600895
epoch number  3 accuracy:  0.9339063067425252
epoch number  4 accuracy:  0.9611427187991769
epoch number  5 accuracy:  0.9920106524633822
epoch number  6 accuracy:  0.9968526812734536
```

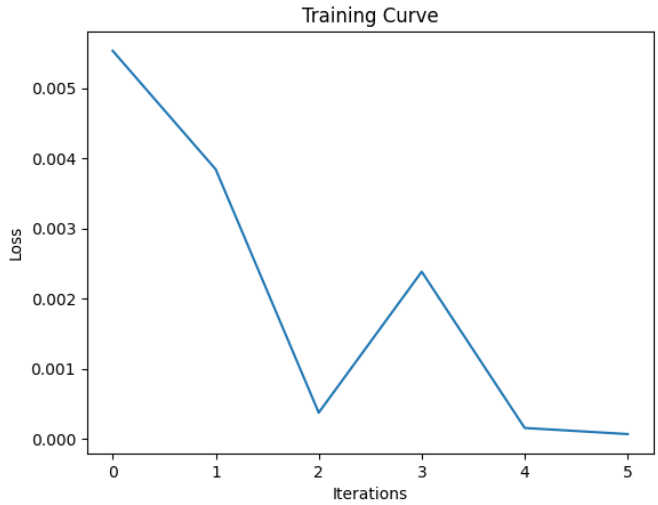


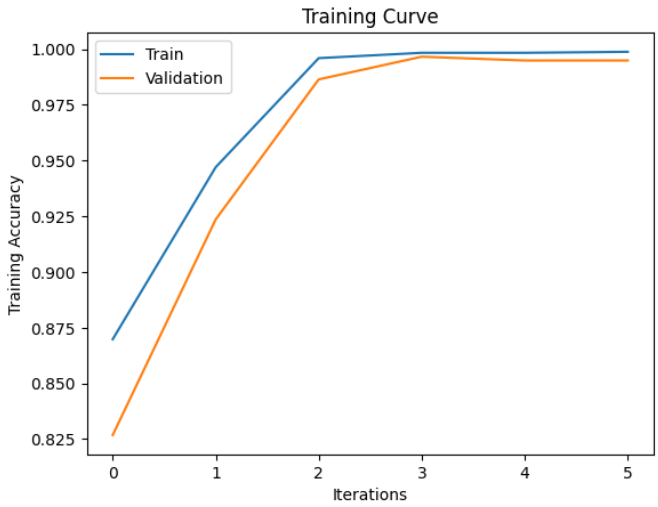



```
Final Training Accuracy: 0.9968526812734536
Final Validation Accuracy: 0.9893992932862191
```

In [ ]:
```
#training alexnet model
model = BreadNetAlex()

train(model, train_features, val_features, batch_size = 64, learning_rate = 0.01, num_epochs = 6)
```

```
105
10
epoch number  1 accuracy:  0.8698189435882089
epoch number  2 accuracy:  0.9470297770462367
epoch number  3 accuracy:  0.9959598982492892
epoch number  4 accuracy:  0.9983540326200808
epoch number  5 accuracy:  0.9983540326200808
epoch number  6 accuracy:  0.9988029328146042
```

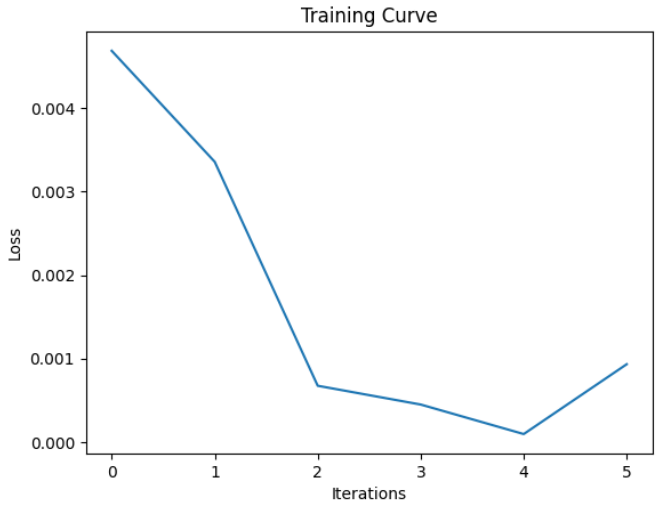


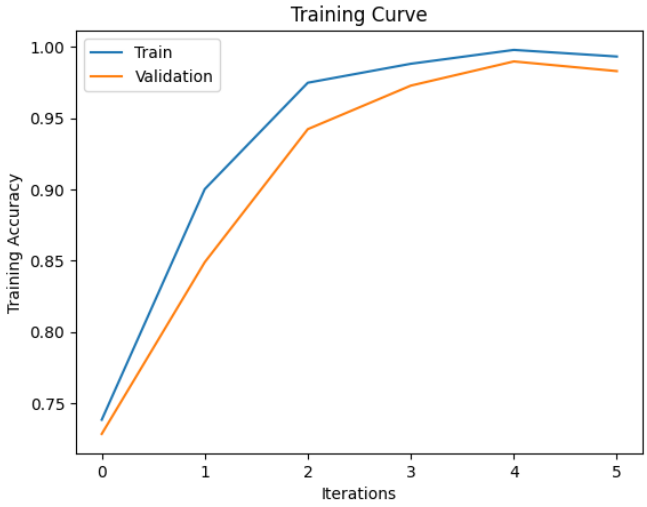



```
Final Training Accuracy: 0.9988029328146042
Final Validation Accuracy: 0.9949066213921901
```

In [ ]:
```
model = BreadNetAlex()

train(model, train_features, val_features, batch_size = 128, learning_rate = 0.01, num_epochs = 6)
```

```
53
5
epoch number  1 accuracy:  0.7382911865928475
epoch number  2 accuracy:  0.9001945234176268
epoch number  3 accuracy:  0.9748615891066886
epoch number  4 accuracy:  0.9881789615442167
epoch number  5 accuracy:  0.9979051324255573
epoch number  6 accuracy:  0.9932664970821488
```

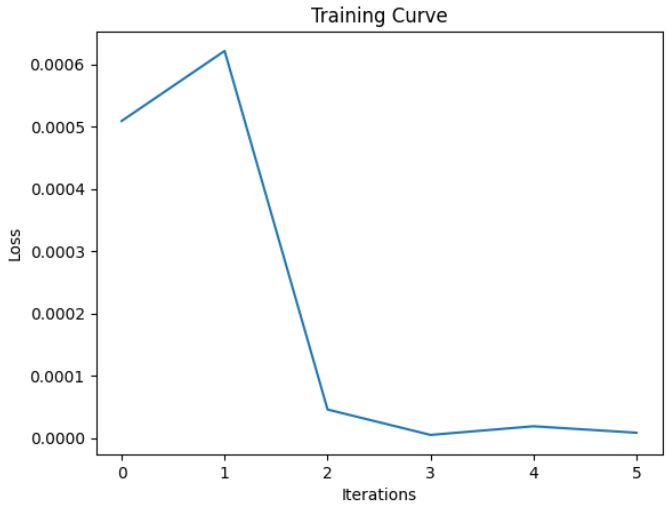


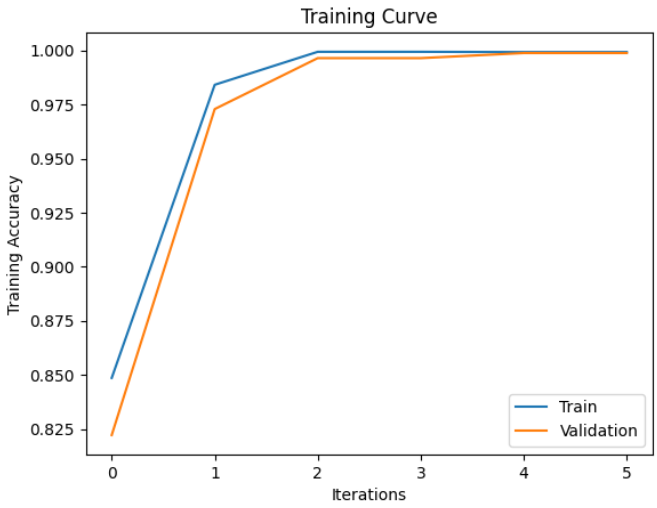



```
Final Training Accuracy: 0.9932664970821488
Final Validation Accuracy: 0.9830220713073005
```

In [ ]:
```
model = BreadNetAlex()

train(model, train_features, val_features, batch_size = 32, learning_rate = 0.01, num_epochs = 6)
#final training curve
```

```
259
27
epoch number  1 accuracy:  0.8485655489650163
epoch number  2 accuracy:  0.9841423556470161
epoch number  3 accuracy:  0.999394746398741
epoch number  4 accuracy:  0.999394746398741
epoch number  5 accuracy:  0.9992736956784893
epoch number  6 accuracy:  0.9992736956784893
```

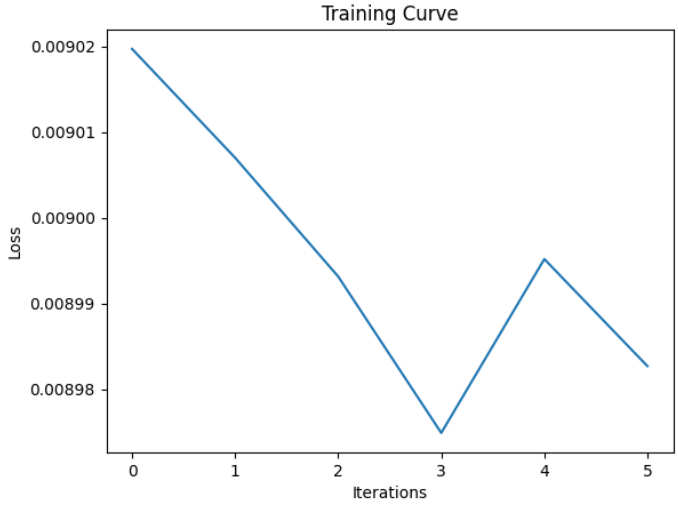


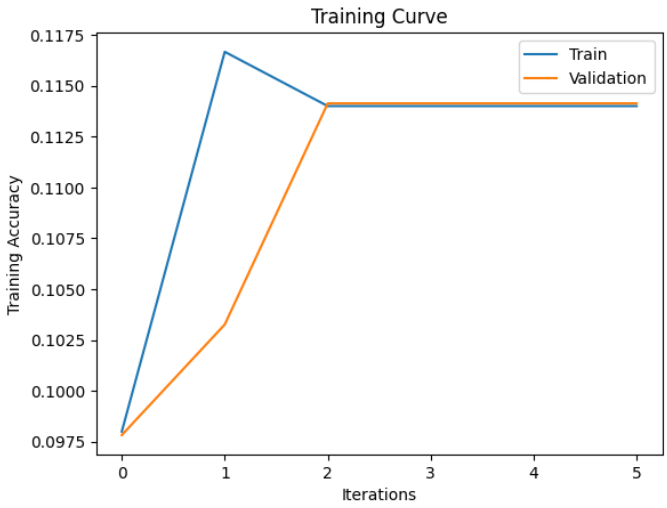



```
Final Training Accuracy: 0.9992736956784893
Final Validation Accuracy: 0.9988221436984688
```

In [ ]:
```
#training for batch_size = 256, learning_rate = 0.01, num_epochs = 6
use_cuda = True
model = BreadNetCNN()
if use_cuda and torch.cuda.is_available():
  model.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')

#proper model
train(model, train_data, val_data, batch_size = 256, learning_rate = 0.01, num_epochs=6)
```

```
CUDA is not available.  Training on CPU ...
6
1
epoch number  1 accuracy:  0.098
epoch number  2 accuracy:  0.11666666666666667
epoch number  3 accuracy:  0.114
epoch number  4 accuracy:  0.114
epoch number  5 accuracy:  0.114
epoch number  6 accuracy:  0.114
```






```
Final Training Accuracy: 0.114
Final Validation Accuracy: 0.11413043478260869
```

In [ ]:
```
#training for batch_size = 128, learning_rate = 0.008, num_epochs = 6
use_cuda = True
model = BreadNetCNN()
if use_cuda and torch.cuda.is_available():
  model.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')

#proper model
train(model, train_data, val_data, batch_size = 128, learning_rate = 0.008, num_epochs=6)
```

In [ ]:
```
#training for batch_size = 64, learning_rate = 0.005, num_epochs = 6
use_cuda = True
model = BreadNetCNN()
if use_cuda and torch.cuda.is_available():
  model.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')

#proper model
train(model, train_data, val_data, batch_size = 64, learning_rate = 0.005, num_epochs=6)
```

In [ ]:
```
#training for batch_size = 128, learning_rate = 0.005, num_epochs = 10
use_cuda = True
model = BreadNetCNN()
if use_cuda and torch.cuda.is_available():
  model.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')

#proper model
train(model, train_data, val_data, batch_size = 128, learning_rate = 0.005, num_epochs=10)
```

In [ ]:
```
#training for batch_size = 64, learning_rate = 0.01, num_epochs = 10
use_cuda = True
model = BreadNetCNN()
if use_cuda and torch.cuda.is_available():
  model.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')
```

```
#proper model
train(model, train_data, val_data, batch_size = 64, learning_rate = 0.01, num_epochs=10)
```

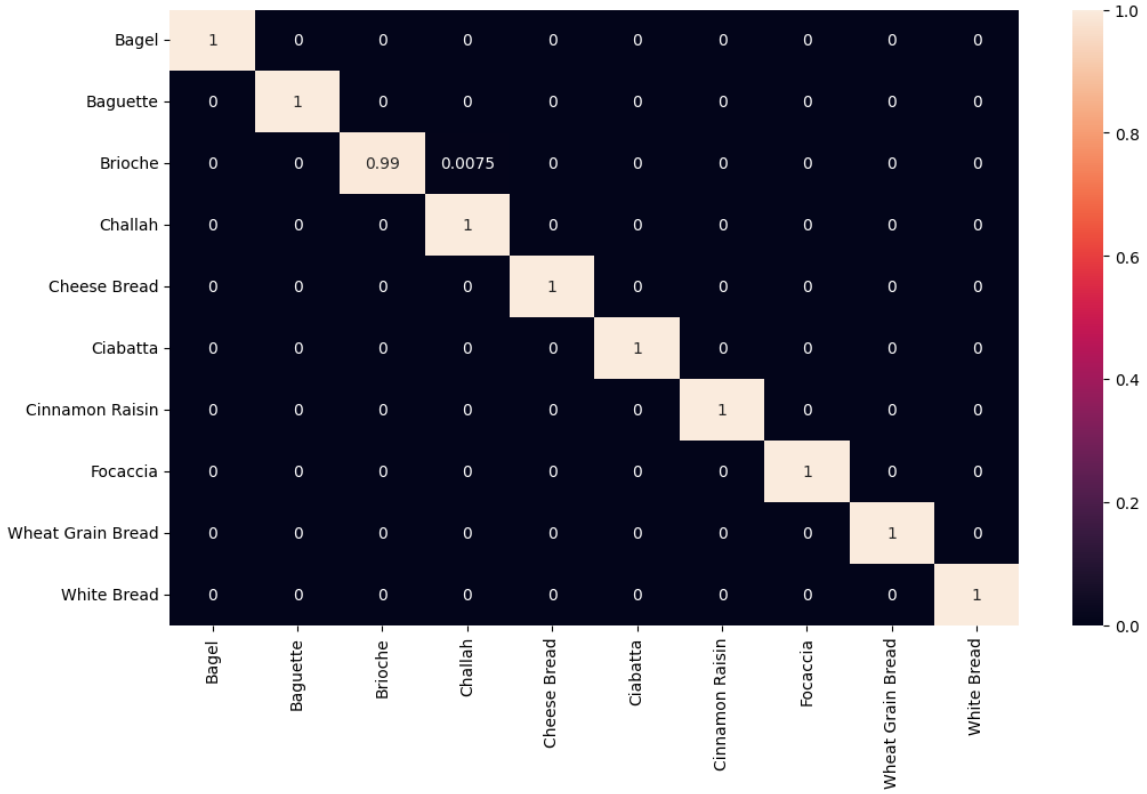
In [ ]:
```
#training for batch_size = 256, learning_rate = 0.01, num_epochs = 20
use_cuda = True
model = BreadNetCNN()
if use_cuda and torch.cuda.is_available():
  model.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')

#proper model
train(model, train_data, val_data, batch_size = 256, learning_rate = 0.01, num_epochs=20)
```

In [ ]:
```
#training for batch_size = 32, learning_rate = 0.01, num_epochs = 6
use_cuda = True
model = BreadNetCNN()
if use_cuda and torch.cuda.is_available():
  model.cuda()
  print('CUDA is available!  Training on GPU ...')
else:
  print('CUDA is not available.  Training on CPU ...')

#proper model
train(model, train_data, val_data, batch_size = 32, learning_rate = 0.01, num_epochs=6)
```

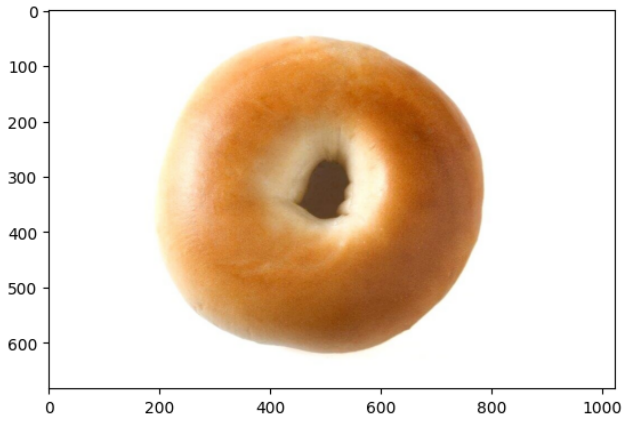
Getting Qualitative Results

In [ ]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sn
import pandas as pd

model = BreadNetAlex()
model_path = "model_{0}_bs{1}_lr{2}_epoch{3}".format("Alex", 32, 0.01, 5)
state = torch.load(model_path)
model.load_state_dict(state)

y_pred = []
y_true = []

train_alex_loader = torch.utils.data.DataLoader(train_features, batch_size=batch_size,num_workers=num_workers, shuffle=True) #or 16

# iterate over train data
for inputs, labels in train_alex_loader:
        output = model(inputs) # Feed Network

        output = (torch.max(torch.exp(output), 1)[1]).data.cpu().numpy()
        y_pred.extend(output) # Save Prediction

        labels = labels.data.cpu().numpy()
        y_true.extend(labels) # Save Truth


# Build confusion matrix
cf_matrix = confusion_matrix(y_true, y_pred)
df_cm = pd.DataFrame(cf_matrix / np.sum(cf_matrix, axis=1)[:, None], index = [i for i in classes],
                     columns = [i for i in classes])
plt.figure(figsize = (12,7))
sn.heatmap(df_cm, annot=True)
```

Out[ ]:
```
<Axes: >
```



In [ ]:
```
# sample prediction of a random image
from matplotlib.pyplot import imread
import PIL, torch, torchvision

test_image_path = '/content/gdrive/MyDrive/APS360 Project Group/Data/Images/Bagel/plain+bagel.jpg'
img_show=imread(test_image_path)
plt.imshow(img_show)
plt.show()

transform = transforms.Compose([transforms.ToTensor(), transforms.Resize((224,224))])
from PIL import Image
img = Image.open(test_image_path)

img_t = transform(img)

features = alexnet.features(img_t.float())
new_img = torch.from_numpy(features.detach().numpy())
alex_img = new_img.squeeze(0)

out = model(alex_img)
prob = F.softmax(out, dim=1)

pred_label = prob.max(1, keepdim=True)[1].item()
#pred_label = out.max(1, keepdim=True)[1].item()
print('predicted label: ', classes[pred_label], 'with a probability of', prob.max(1, keepdim=True)[0].item()) #change prob-->out

for ind,val in enumerate(classes):
    print(f'{val} = {prob[0][ind]*100}%') #predicted probability of classes the image is
```



```
predicted label:  Bagel with a probability of 0.9999986886978149
Bagel = 99.99987030029297%
Baguette = 2.3568477445223834e-06%
Brioche = 0.0001291875960305333 1%
Challah = 1.196719807694535e-07%
Cheese Bread = 4.420855148623559e-08%
Ciabatta = 8.836963799607744e-11%
Cinnamon Raisin = 1.180573917736183e-06%
Focaccia = 7.26697777508889e-08%
Wheat Grain Bread = 8.082902347439358e-09%
White Bread = 4.173382528449565e-09%
```

Evaluating the model

In [ ]:
```
def get_test_accuracy(model, test_loader):
    data = test_loader

    correct = 0
    total = 0
    for imgs, labels in data:

        output = model(imgs)

        #select index with maximum prediction score
        pred = output.max(1, keepdim=True)[1]
        correct += pred.eq(labels.view_as(pred)).sum().item()
        total += imgs.shape[0]
    return correct / total
```

In [ ]:
```
best_model = BreadNetAlex()
best_model_path = "model_{0}_bs{1}_lr{2}_epoch{3}".format("Alex", 32, 0.01, 5)
state = torch.load(best_model_path)
best_model.load_state_dict(state)
```

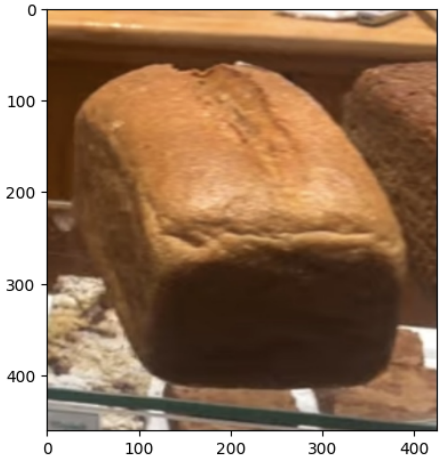
Out[ ]:
```
<All keys matched successfully>
```

In [ ]:
```
test_dataloader = torch.utils.data.DataLoader(test_features, batch_size=16, shuffle=True)
test_accuracy = get_test_accuracy(best_model, test_dataloader)
print("test accuracy:", test_accuracy)
```

```
test accuracy: 0.996551724137931
```

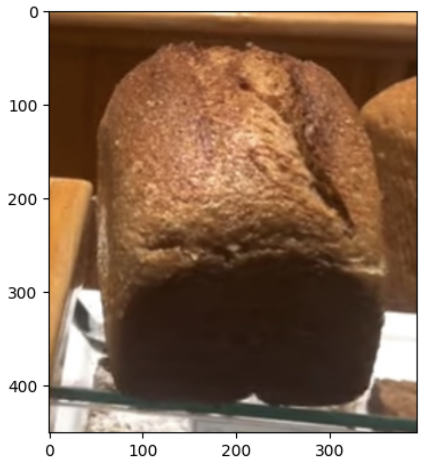
Demonstration on new data

In [ ]:
```
test_image_path = '/content/gdrive/MyDrive/APS360 Project Group/Data/demo_images/demo_bread(2).jpg'
img_show=imread(test_image_path)
plt.imshow(img_show)
plt.show()

transform = transforms.Compose([transforms.ToTensor(), transforms.Resize((224,224))])
from PIL import Image
img = Image.open(test_image_path)

img_t = transform(img)

features = alexnet.features(img_t.float())
new_img = torch.from_numpy(features.detach().numpy())
alex_img = new_img.squeeze(0)

out = model(alex_img)
prob = F.softmax(out, dim=1)

pred_label = prob.max(1, keepdim=True)[1].item()
print('predicted label: ', classes[pred_label], 'with a probability of', prob.max(1, keepdim=True)[0].item()) #change prob-->out

for ind,val in enumerate(classes):
    print(f'{val} = {prob[0][ind]*100}%') #predicted probability of classes the image is
```



```
predicted label:  Wheat Grain Bread with a probability of 0.6434589624404907
Bagel = 0.010713954456150532%
Baguette = 6.0448001022450626e-05%
Brioche = 1.2319226264953613%
Challah = 0.0013820258900523186%
Cheese Bread = 0.23848067224025726%
Ciabatta = 6.584043025970459%
Cinnamon Raisin = 14.003668785095215%
Focaccia = 0.0013894105795770884%
Wheat Grain Bread = 64.34589385986328%
White Bread = 13.582435607910156%
```

In [ ]:
```
test_image_path = '/content/gdrive/MyDrive/APS360 Project Group/Data/demo_images/demo_bread(3).jpg'
img_show=imread(test_image_path)
plt.imshow(img_show)
plt.show()

transform = transforms.Compose([transforms.ToTensor(), transforms.Resize((224,224))])
from PIL import Image
img = Image.open(test_image_path)

img_t = transform(img)

features = alexnet.features(img_t.float())
new_img = torch.from_numpy(features.detach().numpy())
alex_img = new_img.squeeze(0)

out = model(alex_img)
prob = F.softmax(out, dim=1)

pred_label = prob.max(1, keepdim=True)[1].item()
print('predicted label: ', classes[pred_label], 'with a probability of', prob.max(1, keepdim=True)[0].item()) #change prob-->out

for ind,val in enumerate(classes):
    print(f'{val} = {prob[0][ind]*100}%') #predicted probability of classes the image is
```




```
predicted label:  Wheat Grain Bread with a probability of 0.8391998410224915
Bagel = 6.905612735863542e-06%
Baguette = 1.2455145679268753e-06%
Brioche = 0.04409118741750717%
Challah = 1.8364974039286608e-06%
Cheese Bread = 2.2280519260675646e-05%
Ciabatta = 0.001297139679081738%
Cinnamon Raisin = 0.009951649233698845%
Focaccia = 2.6308391909424245e-08%
Wheat Grain Bread = 83.91998291015625%
White Bread = 16.02464485168457%
```

In [2]:
```
%%shell
jupyter nbconvert --to html "/content/Copy_of_Final_Project_Bread.ipynb"
```

```
[NbConvertApp] Converting notebook /content/Copy_of_Final_Project_Bread.ipynb to html
[NbConvertApp] Writing 2675631 bytes to /content/Copy_of_Final_Project_Bread.html
```

Out[2]: