Final Project Individual Report

McLain Wilkinson
Group 7

Introduction…….

Individual work……

The portion I did on the project in detail…….

Result ….……..……

Summary and conclusions……..

The percentage of the code I found from the internet…….

Reference……...

Introduction

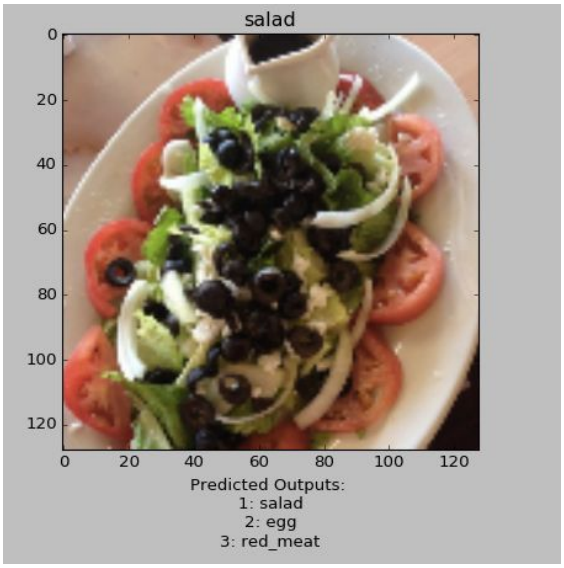
For this project, we attempt to design and train a convolution neural network to classify images of food into twelve broad categories. We found the data on Kaggle (https://kaggle.com/kmader/food41) in the form of .jpg images labelled as one of 101 classes. For preprocessing, we attempt to reduce the total number of classes and increase the number of examples for each class by combining similar labels into larger categories. For instance, images classified as “chocolate cake” and images labeled as “carrot cake” are labelled as “cake” in our dataset. We reduced the number of images by ~½ and reduced the number of categories by ~⅛, resulting in roughly four times the number of images per class for each category in the new dataset. The color images of our new dataset were resized to be 128x128 pixels. Using Amir’s Conv_Mnist_gpu.py code as a base idea, we manipulated the file to:

1) accommodate our new training and testing sets for input
2) changed the network architecture to accommodate our input images (increased size and number of channels), number of layers, number of kernels for each layer, and dropout layer
3) Added code to show plot of average loss value for each epoch
4) Added code to provide more detailed test results including:
    a) Confusion matrix
    b) Accuracy for individual categories
    c) Plot 10 test set images with class label along with top 3 predicted outputs

Description of your individual work. Provide some background information on the development of the algorithm and include necessary equations and figures.

I am responsible for writing much of the code and adding testing functionality and plots for the project, while my teammates were focused more on determining optimal values for parameters and tweaking the design of the network architecture to increase the prediction accuracy of the network. The preprocessing code file (create_dataset.py) to create the training and testing sets from the downloaded Kaggle folder of images was written by me. Working with Joanne, I was able to alter the Conv_Mnist_gpu.py code to successfully read in our h5 dataset files through PyTorch’s DataLoader utility and alter network parameters (input channels, output channels, kernel_size, padding, etc.) to produce outputs used to get an accuracy score on the images of the test set. I added code to the training section to create a plot showing the average loss value for each epoch. I then focused on producing more insightful test results, such as printing out accuracy values for the network’s classification of images of individual categories, printing a confusion matrix, as well as plotting 10 labelled test set images with the top 3 most likely predicted outputs determined by the network. These additional results were useful for me and my teammates to determine how to increase the overall accuracy by changing the network architecture as well as changing or removing labels of certain images when it appeared the network was regularly misclassifying a specific category of image. Examples of figures and more detailed description of code I wrote is included below.

Test image and top 3 predicted network outputs:



Individual category accuracy, confusion matrix:

Documents — ubuntu@ip-172-31-21-224: ~/ML2project — ssh — 129×40

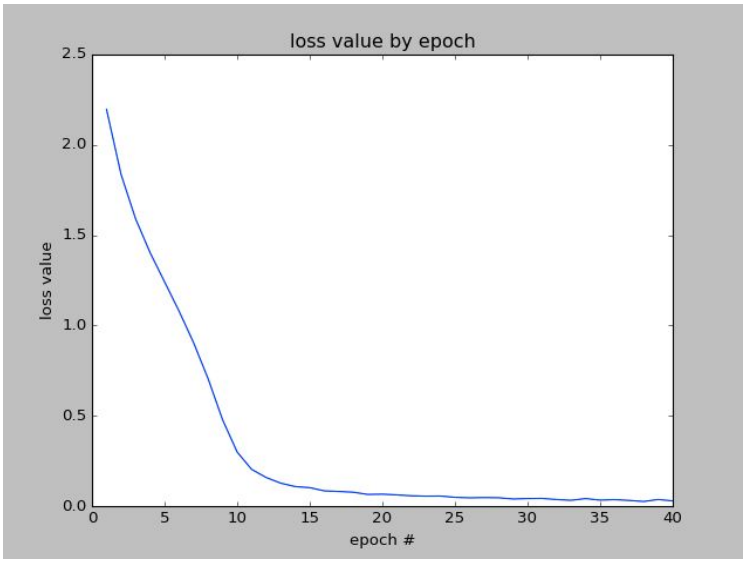
```
Epoch [5/5], Iter [900/960] Loss: 1.2279
training time: 1580.43 seconds
-----------------------------------------------------------
Testing...
-----------------------------------------------------------
Results
Accuracy of the model on the test images: 55 %

Individual class accuracy:
smooth_des       171 correct / 600 total        =>       28.50 % accuracy
red_meat         368 correct / 800 total        =>       46.00 % accuracy
egg      248 correct / 800 total        =>       31.00 % accuracy
pasta    723 correct / 1200 total       =>       60.25 % accuracy
soup     561 correct / 800 total        =>       70.12 % accuracy
salad    266 correct / 400 total        =>       66.50 % accuracy
fried_food       630 correct / 1000 total       =>       63.00 % accuracy
sandwich         690 correct / 1200 total       =>       57.50 % accuracy
noodles  231 correct / 600 total        =>       38.50 % accuracy
cake     833 correct / 1000 total       =>       83.30 % accuracy
sweet_brea       315 correct / 600 total        =>       52.50 % accuracy
shell    295 correct / 600 total        =>       49.17 % accuracy

class with highest accuracy: cake 83.30 %
class with lowest accuracy: smooth_des 28.50 %

Confusion Matrix
            smooth_des  red_meat  egg  pasta  soup  salad  fried_food  sandwich  noodles  cake  sweet_brea  shell
smooth_des         171         5   10      9    16      4          14        10       10    28           4     12
red_meat            14       368   23     30     7      6          13        30        8    12           4     43
egg                  4         6  248     12     6      4           8         9       12     1           6      6
pasta               13        43   76    723    80     15          51        36      113    15          38     33
soup                14        13   21     52   561      2           8        12       47     8          24     14
salad                5         9   40     28     4    266           5        14       16     1           0     17
fried_food          15        25   30     98    25      3         630       149       38    13          41     24
sandwich            36        83  131     77    21     46         103       690       81    41          64     25
noodles              1         2   16     12     8      6           3         5      231     0           1      5
cake               281       192  105     72    44     32          63       142       19   833          94     84
sweet_brea          30        46   80     78    22      5          93        97       15    41         315     42
shell               16         8   20      9     6     11           9         6       10     7           9    295
ubuntu@ip-172-31-21-224:~/ML2project$
```

Loss plot:



Describe the portion of the work that you did on the project in detail. It can be figures, codes, explanation, pre-processing, training, etc.

Code for create_dataset.py:

```
import os
import numpy as np
import h5py
from PIL import Image
import matplotlib.pyplot as plt
from skimage.transform import resize
from sklearn.model_selection import train_test_split
import random

'''
This file will create an HDF5 database of images, labels, and categories given that the following
directories and files exist in the current working directory:
  - folder titled 'images' containing subfolders titled with a type of food
  - these subfolders (i.e. 'tacos' or 'hot_dog') contain .jpg files of pictures of food from folder title
'''

# define 12 new categories in array
new_category_names = ['smooth_dessert', 'red_meat', 'egg', 'pasta', 'soup', 'salad', 'fried_food',
                      'sandwich', 'noodles', 'cake', 'sweet_breakfast', 'shell']

# create mapping from old categories --> new categories
category_name_mapping = {'bibimbap': 'egg',
                         'caesar_salad': 'salad',
                         'carrot_cake': 'cake',
                         'chocolate_cake': 'cake',
                         'chocolate_mousse': 'smooth_dessert',
                         'club_sandwich': 'sandwich',
                         'cup_cakes': 'cake',
```

```
                    'deviled_eggs': 'egg',
                    'eggs_benedict': 'egg',
                    'escargots': 'shell',
                    'filet_mignon': 'red_meat',
                    'fish_and_chips': 'fried_food',
                    'french_fries': 'fried_food',
                    'french_onion_soup': 'soup',
                    'french_toast': 'sweet_breakfast',
                    'fried_calamari': 'fried_food',
                    'frozen_yogurt': 'smooth_dessert',
                    'gnocchi': 'pasta',
                    'greek_salad': 'salad',
                    'grilled_cheese_sandwich': 'sandwich',
                    'hamburger': 'sandwich',
                    'hot_and_sour_soup': 'soup',
                    'hot_dog': 'sandwich',
                    'huevos_rancheros': 'egg',
                    'ice_cream': 'smooth_dessert',
                    'lasagna': 'pasta',
                    'lobster_bisque': 'soup',
                    'lobster_roll_sandwich': 'sandwich',
                    'macaroni_and_cheese': 'pasta',
                    'miso_soup': 'soup',
                    'mussels': 'shell',
                    'onion_rings': 'fried_food',
                    'oysters': 'shell',
                    'pad_thai': 'noodles',
                    'pancakes': 'sweet_breakfast',
                    'pho': 'noodles',
                    'pork_chop': 'red_meat',
                    'poutine': 'fried_food',
                    'prime_rib': 'red_meat',
                    'pulled_pork_sandwich': 'sandwich',
                    'ramen': 'noodles',
                    'ravioli': 'pasta',
                    'red_velvet_cake': 'cake',
                    'spaghetti_bolognese': 'pasta',
                    'spaghetti_carbonara': 'pasta',
                    'steak': 'red_meat',
                    'strawberry_shortcake': 'cake',
                    'waffles': 'sweet_breakfast'}

# lists to add images and labels
images = []
labels = []

# resize images to be 128x128x3, add images and labels to respective lists
for folder in os.listdir('images'):
  if folder in category_name_mapping:
    print('converting images from:', folder, '-->', category_name_mapping[folder])
```

```
        for f in os.listdir('images/' + folder):
            image = np.array(Image.open('images/' + folder + '/' + f))
            try:
                resized_image = resize(image, (128, 128, 3))
                images.append(resized_image)
                labels.append(new_category_names.index(category_name_mapping[folder]))
            except:
                continue
    else:
        print(folder, 'not used to create target classes')
'''
# shuffle images and labels together
image_labels = list(zip(images, labels))
random.shuffle(image_labels)
images, labels = zip(*image_labels)
images = list(images)
labels = list(labels)
'''
# convert lists to numpy arrays
images = np.array(images)
labels = np.array(labels)
categories = np.array(new_category_names, dtype='S10')

'''
# split into train and test sets 80/20 split
random.seed(0)
num_images = len(images)
split_index = 4 * num_images // 5
train_images = images[:split_index]
train_labels = labels[:split_index]
test_images = images[split_index:]
test_labels = labels[split_index:]
print('number of training images:', len(train_images))
print('number of testing images:', len(test_images))
'''

# use sklearn train_test_split to create training and testing sets
train_images, test_images, train_labels, test_labels = train_test_split(images, labels, test_size=0.2,
stratify=labels)

# create new dataset objects
train_filename = 'food_train.h5'
train_file = h5py.File(train_filename, 'w')
test_filename = 'food_test.h5'
test_file = h5py.File(test_filename, 'w')

# add datasets to dataset objects
train_categories = train_file.create_dataset('categories', data=categories)
train_labels = train_file.create_dataset('labels', data=train_labels)
train_images = train_file.create_dataset('images', data=train_images)
```

```
test_categories = test_file.create_dataset('categories', data=categories)
test_labels = test_file.create_dataset('labels', data=test_labels)
test_images = test_file.create_dataset('images', data=test_images)

# confirm that objects are in dataset
print('training dataset categories:', list(train_file.keys()))
print('label for training image 0:', train_file.get('labels')[0])
print('category for training image 0:', train_file.get('categories')[train_file.get('labels')[0]].decode())
image0 = train_file.get('images')[0]
print('size of training image 0:', image0.shape)
plt.imshow(image0)
plt.title('training image 0')
plt.show()
print('target classes for dataset:')
for cat in train_file.get('categories'):
  print(cat.decode())
u, c = np.unique(test_labels, return_counts=True)
for un, co in zip(u, c):
  print(un, co)
train_file.close()
test_file.close()
```

Food_CNN.py

```
# define the network model
class CNN(nn.Module):
  def __init__(self):
    super(CNN, self).__init__()
    self.layer1 = nn.Sequential(            # input size = 128 x 128
      nn.Conv2d(3, 64, kernel_size=10, padding=4),
      nn.BatchNorm2d(64),
      nn.ReLU(),
      nn.MaxPool2d(2, stride=2, padding=1))  # output size = 64 x 64
    self.layer2 = nn.Sequential(
      nn.Conv2d(64, 128, kernel_size=10, padding=4),
      nn.BatchNorm2d(128),
      nn.ReLU(),
      nn.MaxPool2d(2, stride=2, padding=1))  # output size = 32 x 32
    self.layer3 = nn.Sequential(
      nn.Conv2d(128, 256, kernel_size=10, padding=4),
      nn.BatchNorm2d(256),
      nn.ReLU(),
      nn.MaxPool2d(2, stride=2, padding=1))  # output size = 16 x 16
    self.layer4 = nn.Sequential(
      nn.Conv2d(256, 384, kernel_size=10, padding=4),
      nn.BatchNorm2d(384),
      nn.ReLU(),
      nn.MaxPool2d(2, stride=2, padding=1))  # output size = 8 x 8
    self.layer5 = nn.Sequential(
```

```
            nn.Conv2d(384, 256, kernel_size=4, padding=2),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(2, stride=2, padding=1))  # output size = 5 x 5
        self.layer6 = nn.Sequential(
            nn.Conv2d(256, 128, kernel_size=2),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2, stride=2))           # output size = 2 x 2
        self.dropout = nn.Dropout(0.5)
        self.fc = nn.Linear(2 * 2 * 128, 12)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = self.layer5(out)
        out = self.layer6(out)
        out = self.dropout(out)
        out = out.view(out.size(0), -1)
        out = self.fc(out)
        return out

# Test the Model
cnn.eval()  # Change model to 'eval' mode (BN uses moving mean/var).

print('Testing...')

correct = 0
total = 0
correct_array = np.zeros(len(target_classes))
total_array = np.zeros(len(target_classes))
confusion = {x: [0 for i in range(len(target_classes))] for x in range(len(target_classes))}

for images, labels in test_loader:
    input_images = Variable(images).cuda()
    outputs = cnn(input_images)
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted.cpu() == labels).sum()
    for pred, actual in zip(predicted.cpu(), labels):
        confusion[int(actual)][int(pred)] += 1
        total_array[actual] += 1
        if pred == actual:
            correct_array[actual] += 1
    images = images
# ------------------------------------------------------------------------
# display overall results
print('--------------------------------------------------')
```

```
print('Results')
print('Accuracy of the model on the test images: %d %%' % (100 * correct / total))
print('')
# display individual category results
print('Individual class accuracy:')
target_classes = tuple(target_class.decode() for target_class in train_dataset.classes)
for i, t in enumerate(target_classes):
    print(t + '\t', int(correct_array[i]), 'correct', '/', int(total_array[i]), 'total',
          '\t=>\t', '%.2f' % (100 * correct_array[i] / total_array[i]), '%', 'accuracy')

# show and print out class/accuracy for highest and lowest accuracy values
pct_array = correct_array / total_array
best = int(np.argmax(pct_array))
worst = int(np.argmin(pct_array))
print('')
print('class with highest accuracy:', target_classes[best],
      '%.2f' % (100 * correct_array[best] / total_array[best]), '%')
print('class with lowest accuracy:', target_classes[worst],
      '%.2f' % (100 * correct_array[worst] / total_array[worst]), '%')
print('')

# create and show confusion matrix
confusion = pd.DataFrame.from_dict(confusion)
confusion.columns = target_classes
confusion.index = target_classes
print('Confusion Matrix')
print(confusion)

# show 10 test images with target class & top 3 predicted outputs
for i, (output, label) in enumerate(zip(outputs.data.cpu(), labels)):
    if i < 10:
        output = np.array(output)
        ndx = output.argsort()[-3:][::-1]
        caption = 'Predicted Outputs: \n'
        for j, n in enumerate(ndx):
            caption = caption + (str(j + 1) + ': ' + target_classes[n] + '\n')
        image = np.array(images[i]).T
        plt.figure(i + 1)
        plt.imshow(image)
        plt.title(target_classes[label])
        plt.xlabel(caption)
        plt.tight_layout()

# show loss plot & 10 images w/ predicted classes
plt.show()
```

Research of Cross Entropy Loss

Results.
Describe the results of your experiments, using figures and tables wherever possible. Include all results (including all figures and tables) in the main body of the report, not in appendices. Provide an explanation of each figure and table that you include. Your discussions in this section will be the most important part of the report.

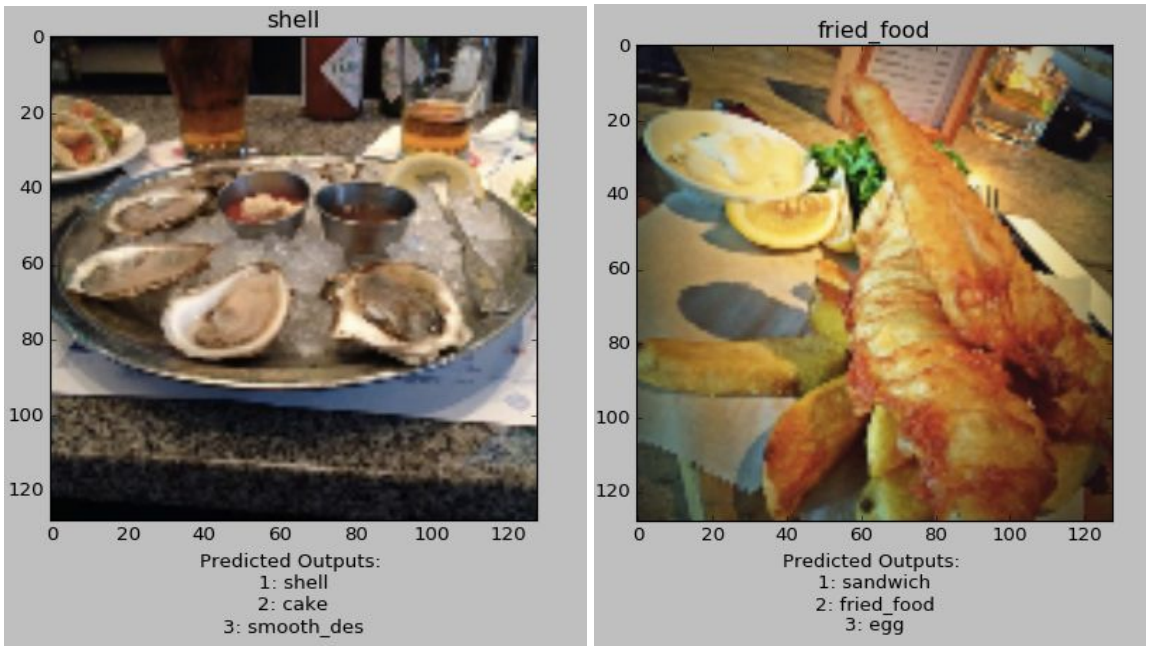
The usefulness of our CNN can be determined by its ability to accurately label previously unseen test images. Upon training the network, we obtained the results of the CNN's predicted outputs for the 9,600 images in our testing dataset. The model will print out overall results containing the following metrics: Overall accuracy (%) on the testing set, individual class accuracy, the two categories producing the maximum and minimum accuracy values, and a confusion matrix. The model output and confusion matrix are shown below.

Confusion Matrix

	smooth_des	red_meat	egg	pasta	soup	salad	fried_food	sandwich	noodles	cake	sweet_brea	shell
smooth_des	290	16	20	12	21	3	20	33	10	79	24	10
red_meat	51	533	45	42	11	10	24	71	19	67	31	59
egg	19	29	427	54	17	40	25	68	31	23	24	25
pasta	24	27	90	795	44	33	100	80	78	29	62	35
soup	15	13	19	67	618	1	9	21	37	12	25	12
salad	3	8	9	17	1	244	3	13	13	7	2	8
fried_food	20	27	29	52	14	4	660	135	19	15	46	12
sandwich	22	38	47	29	11	19	68	591	19	37	36	8
noodles	7	19	21	48	24	24	11	20	351	1	5	12
cake	108	48	30	21	12	9	29	68	6	655	52	32
sweet_brea	25	24	48	41	14	3	36	84	7	50	273	29
shell	16	18	15	22	13	10	15	16	10	25	20	358

The above outputs show a 60% overall accuracy value on the testing set, suggesting the model predicted the correct label for 60% of images contained in the test set. While this accuracy is not stellar, it is a significant improvement over the accuracy one would expect to achieve from a random guess, a 1/12 chance (~8-9%). The following breakdown of the CNN's predictions for images of individual classes provides more insight on the behavior of the network and helps to explain what could have lowered the overall accuracy. The network achieved the best accuracy labeling images belonging to the "soup" category, correctly identifying 77.25% of soup images, while only correctly labeling 45.5% of "sweet breakfast" items. This difference in accuracy percentage (31.75%) for these two categories of items is significant, and potentially suggests issues concerning the uniformity and similarity of images in the "sweet breakfast" category. The confusion matrix offers an even deeper breakdown of the CNN's test set predictions, showing the specific counts of predicted labels for each image category with columns representing the actual label and rows representing the predicted label. The diagonal values show correctly classified observations, and all other cells represent incorrectly classified images. Consequently, if non-diagonal count is particularly large, it may suggest that the network struggled to distinguish the two classes. For instance, one of the highest counts not occurring on the diagonal represents "smooth dessert" images misclassified as "cake". Interestingly, the most frequently mislabeled "cake" image was predicted to be in the "smooth dessert" class. So the CNN seems to have difficulty distinguishing images of these two classes. Additionally, the highest number of misclassifications occurs for "sandwich" images misclassified as "fried food".

Inspecting some of the images belonging to these few categories does provide intuitive explanations for the errors, however. The final results the model produces involve plotting 10 labeled images along with the top three most likely labels predicted by the CNN. Two sample images with predicted label plots are shown below.



The CNN correctly predicted the “shell” image on the left to be in the “shell” category. Unfortunately, the “fried food” image on the right was incorrectly predicted to be a “sandwich”, although “fried food” was the second most likely prediction. The food item in this image does appear to be sandwich-like upon first glance, so the network didn’t make a terrible prediction. Other images prove to be more problematic. For example, some “sandwich” images in the dataset contain a picture of a sandwich and french fries on a plate. What should the true class for this picture be, “sandwich” or “fried food”? Here is where the network as it currently exists may fall short at classifying images correctly. For future consideration, we may want to allow images to carry multiple labels, so the sandwich and fries image could be correctly identified as both “sandwich” and “fried food”.

Summary and conclusions.

Given the structure and accompanying difficulty of our problem, we successfully built a network that could achieve a reasonable accuracy classifying images into 12 broad categories of food. The biggest problems of our experiment are more associated with the nature of the images in our dataset and our decisions concerning their labels. While combining classes together may have provided the network with more training images, it introduced much more variance into the nature of the categories. For instance, our “pasta” category contains images consisting if 6 very diverse dishes in shape and color: gnocchi, lasagna, macaroni and cheese, ravioli, spaghetti bolognese, and spaghetti carbonara. Even though we had more training images, the network

was not given very uniform examples of what a pasta dish entails. Maybe this combining did more harm than good.

To prevent this problem, we could introduce using multiple labels for each image. For instance, a picture of a slice of cake topped with ice cream could be labeled by a tuple (0, 9) representing both “smooth dessert” and “cake”. This way, the network would not be penalized for classifying the network as only one or the other. Additionally, we could allow the network to classify an image with 2 classes, by selecting both classes if the predicted probability for each is greater than a certain threshold. This could allow the network to predict “sandwich” AND “fried food” if the image contains pictures of both.

Either way, the problem could be worked through with the presence of cleaner and more uniform data. Maybe our network would have been more successful had we not chosen to combine classes of images.

Calculate the percentage of the code that you found or copied from the internet. (For example, if you used 50 lines of code from the internet and then you modified 10 of lines and added another 15 lines of your own code, the percentage will be (50−10) / (50+15) ×100.)

Create_dataset.py
(120-110)/(120+30)= 7%

References

Cross-entropy loss:
https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html#cross-entropy