Front Matter

I want the Notebook to be as informative as possible, but model creating and training process follows some standard procedure that I do not want to repeat. Therefore, if you can, spend time reading the `PROLOGUE/Routine.ipynb` Notebook first.

Paper Implementation - VGG16

Hello, this is my first milestone project - implementation of the VGG16 architecture from the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition". The paper explored the effect of increasing layers on a model based on the filter size of $3 * 3$ that we previously explored in the `TinyCNN` notebook (that is also the reason why that architecture is called TinyVGG). The architecture was the runner-up in the ImageNet 2014 Challenge for classification.

16 in VGG16 stands for 16 layers, all based on two basic units: convolution with filter size $3 * 3$, stride 1, padding 1 and max-pooling with window size $2 * 2$, stride 2. The table shown below, taken from the paper abovem, is the architecture for each of the VGG configuration. In this project, I will implement the VGG16-D one.

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

I divide the project into two notebooks. This is the first one, where I will focus on acquiring

and transforming the data. The next one will be about building and training the model.

Downloading and extracting data

The task for our model will be classification, using a bigger dataset called Food101. This is a built-in PyTorch dataset, so the processing can be fairly straightforward. However, it is not fun, so let's take the Kaggle version and process it to what we want.

First, downloading data from Kaggle. The easy way: you can download the zip file (~6 GB), upload it to Google Drive, and then mount Google Drive to Colab. The slightly harder approach: you will need to sign up and obtain a Kaggle token, and then use the `kaggle` module to download the data. Let's do that.

First, we need a variable to keep track of the environment we are in as it is different to run the notebook right on Kaggle and run it anywhere else (from the teaching of a Kaggle Grandmaster)

In [ ]:
```
# Import modules
import os
from pathlib import Path

# Keep an environment variable
iskaggle = os.environ.get('KAGGLE_KERNEL_RUN_TYPE', '')
```

Next, based on the docs, we will need to create a `/.kaggle/kaggle.json` . You can go to File Explorer and create a folder in your machine, or we can create that file right from the notebook. Let's do that.

In [ ]:
```
creds = '{"username":"hangenyuu","key":"a34fdad4d3f3d0ee2b1b7a010b3ffc4c"}'
```

In [ ]:
```
# Paste your API here (I have run and then deleted mine)
# creds = ''
```

In [ ]:
```
cred_path = Path('~/.kaggle/kaggle.json').expanduser()
if not cred_path.exists():
    cred_path.parent.mkdir(exist_ok=True)
    cred_path.write_text(creds)
    cred_path.chmod(0o600)
```

Next, let's use the method `dataset_download_cli` to download and unzip data files.

In [ ]:
```
# Sanity check
!kaggle datasets list
```

```
ref                                                            title
size  lastUpdated          downloadCount  voteCount  usabilityRating
-------------------------------------------------------------
-----------------------------------------------  -----  -------------------
------------  ---------  --------------
meirnizri/covid19-dataset                                      COVID-19 Dataset
5MB  2022-11-13 15:47:17          13208        373  1.0
thedevastator/analyzing-credit-card-spending-habits-in-india    Credit Card Spendi
```

```
ng Habits in India           319KB  2022-12-14 07:30:37            761         32
1.0
michals22/coffee-dataset                                          Coffee dataset
24KB  2022-12-15 20:02:12           2865         71  1.0
thedevastator/unlock-profits-with-e-commerce-sales-data           E-Commerce Sales D
ataset                         6MB  2022-12-03 09:27:17           2157         55
1.0
thedevastator/jobs-dataset-from-glassdoor                         Salary Prediction
3MB  2022-11-16 13:52:31          7696        168  1.0
die9origephit/fifa-world-cup-2022-complete-dataset                Fifa World Cup 202
2: Complete Dataset            7KB  2022-12-18 22:51:11           2380         98
1.0
mattop/highest-grossing-mobile-games                              Highest Grossing M
obile Games                    3KB  2022-12-19 15:20:22            450         23
1.0
thedevastator/uncover-global-trends-in-mental-health-disorder     Global Trends in M
ental Health Disorder          1MB  2022-12-14 05:30:38            699         24
1.0
rajkumarpandey02/fifa-world-cup-attendance-19302022               FIFA World Cup Att
endance 1930-2022              5KB  2022-12-19 10:04:26            746         21
1.0
thedevastator/revealing-insights-from-youtube-video-and-channe    YouTube Videos and
Channels Metadata             82MB  2022-12-14 02:48:24            441         23
0.9411765
thedevastator/uncovering-insights-to-college-majors-and-their     College Majors and
their Graduates               39KB  2022-12-06 16:06:52           1101         32
1.0
mvieira101/global-cost-of-living                                  Global Cost of Liv
ing                            1MB  2022-12-03 16:37:53           3547         73
0.9705882
anashamoutni/students-employability-dataset                       Students' Employab
ility Dataset - Philippines   97KB  2022-12-18 15:51:39            652         27
0.88235295
swaptr/fifa-world-cup-2022-statistics                             FIFA World Cup 202
2 Team Data                   15KB  2022-12-19 00:29:15           2708         61
0.9705882
thedevastator/the-ultimate-netflix-tv-shows-and-movies-dataset    Netflix TV Shows a
nd Movies (2022 Updated)       2MB  2022-11-27 20:41:41           2525         46
1.0
whenamancodes/predict-diabities                                   Predict Diabetes
9KB  2022-11-09 12:18:49          7999        122  1.0
kulturehire/understanding-career-aspirations-of-genz              Understanding Care
er Aspirations of GenZ         8KB  2022-12-21 13:44:32            242         23
0.9117647
thedevastator/uncovering-wage-disparities-in-pennsylvania-s-hi    Higher Education W
ages                         223KB  2022-12-04 15:42:36           1285         40
1.0
laibaanwer/superstore-sales-dataset                               SuperStore Sales D
ataset                         2MB  2022-12-07 08:53:32           1505         37
1.0
catherinerasgaitis/mxmh-survey-results                            Music & Mental Hea
lth Survey Results            22KB  2022-11-21 10:03:12           3201         75
```

In [ ]:
```
path = Path('kmader/food41')
```

In [ ]:
```
if not iskaggle and not path.exists():
    import kaggle
    kaggle.api.dataset_download_cli(str(path))
```

```
Downloading food41.zip to /content
```

```
100%|██████████| 5.30G/5.30G [00:41<00:00, 137MB/s]
```

We have the data in the zip file. Now all we need to do is to extract them out.

In [ ]:
```
folder_path = Path('food41')
os.mkdir(folder_path)
```

In [ ]:
```
import zipfile
zipfile.ZipFile(f'{folder_path}.zip').extractall(folder_path)
```

Food-101

The dataset is introduced in this paper, consisting of 101 classes, each with 750 training and 250 testing examples, totalling 1000 images each. The dataset comes with a metadata folder, giving information about which image should go into which subset, which is great! The data was already split into traing and testing examples, but we also need a *validation set*. The testing examples were manually selected to contain noise and challenge the model, so we will not touch that, but we will split the training set further to create a validation set. Now, creating a good validation set is an art, but here we will jsut use good ol' random splitting.

First, we will need to format the in the `images` folder into `train`, `valid`, and `test` folders. Next, we will load the data. The process is quite the same, what's new this time is we will apply more transformation.

In [ ]:
```
# Generic torch process
from torch import nn
import torch
from torch.utils.data import DataLoader

# Specifically for computer vision
import torchvision
from torchvision import datasets, transforms

# Other module(s)
import matplotlib.pyplot as plt
import gc
import json
import shutil
from sklearn.model_selection import train_test_split
import numpy as np
```

In [ ]:
```
np.random.seed(17)
torch.manual_seed(17)
```

Out[ ]:
```
<torch._C.Generator at 0x7fadaaccae90>
```

In [ ]:
```
with open('/content/food41/meta/meta/train.json', 'r') as fp:
    train_dict = json.load(fp)
with open('/content/food41/meta/meta/test.json', 'r') as fp:
    test_dict = json.load(fp)
print(len(train_dict['apple_pie']), train_dict['apple_pie'][-10:])
print(len(test_dict['apple_pie']), test_dict['apple_pie'][-10:])
```

```
750 ['apple_pie/960233', 'apple_pie/960669', 'apple_pie/962315', 'apple_pie/966595
', 'apple_pie/973088', 'apple_pie/973428', 'apple_pie/98352', 'apple_pie/98449', '
apple_pie/987860', 'apple_pie/997124']
250 ['apple_pie/885848', 'apple_pie/886793', 'apple_pie/904832', 'apple_pie/908367
', 'apple_pie/963140', 'apple_pie/981895', 'apple_pie/984571', 'apple_pie/986844',
'apple_pie/99556', 'apple_pie/997950']
```

As mentioned, there are 1000 images for 101 categories of food. Each .json file is essentially a dictionary containing 101 keys and 101 values, each value is a list of paths of 750 training images or 250 testing images.

The list value of a dictionary key contains the strings that are the file paths of the images without the extension. We will note this while using it to copy the images to proper folders.

In [ ]:
```
if not os.path.exists('data'):
    os.mkdir('data')
new_data_path = Path('data')
original_data_path = Path('food41/images')
new_folders = ['train', 'test']
for folder in new_folders:
    if not os.path.exists(new_data_path/folder):
        os.mkdir(new_data_path/folder)
    if folder == 'train':
        if not os.path.exists(new_data_path/'valid'):
            os.mkdir(new_data_path/'valid')
        for key, value in train_dict.items():
            train_value, valid_value = train_test_split(value, train_size=0.75)
            train_set, valid_set = set(train_value), set(valid_value)
            if not os.path.exists(new_data_path/folder/key):
                os.mkdir(new_data_path/folder/key)
            if not os.path.exists(new_data_path/'valid'/key):
                os.mkdir(new_data_path/'valid'/key)
            for image in os.listdir(original_data_path/key):
                image_path = key + '/' + image
                image_path = image_path.split('.')[0]
                if image_path in train_set:
                    shutil.copy(original_data_path/key/image, new_data_path/'trai
                if image_path in valid_set:
                    shutil.copy(original_data_path/key/image, new_data_path/'vali
    else:
        for key, value in test_dict.items():
            value_set = set(value)
            if not os.path.exists(new_data_path/folder/key):
                os.mkdir(new_data_path/folder/key)
            for image in os.listdir(original_data_path/key):
                image_path = key + '/' + image
                image_path = image_path.split('.')[0]
                if image_path in value_set:
                    shutil.copy(original_data_path/key/image, new_data_path/folde
```

And we are done! Now we can load data as we like!

But first, let's write some transformations for the images to perform data augmentation.

Data Augmentation

This is a technique to generate more training data by performing operations on the original data (such as flipping, shearing, rotating for images). The artificial data should generate the same output as the original one, but they are different, so hopefully the model is encouraged to learn the general pattern of the data instead of overfitting. Data augmentation is usually seen in training, but there is a technique called "test-time augmentation" that has been passed down among the Kaggle Grandmaster and implemented in the library fastai.

For the transformations, first we have the staples: `ToTensor()`, which turns images to `torch.tensor` objects. You may notice the `Normalize()` with some arbitrary parameters (the first is a list of means for each color channel and the second is a list of standard deviations for each color channel). These are parameters for the normalization of ImageNet dataset and are required by all PyTorch pre-trained models. This may not necessarily be true for our data, but it can be used. PyTorch also recommends having images of size $224 * 224$ pixels, so we use resize to that. The other transformations do what it is called for, with parameters for angle, probability, etc. (Explore more transformations on PyTorch docs.) Finally, we call `Compose()` to stack these transformations together.

In [ ]:
```
train_transforms = transforms.Compose([transforms.RandomResizedCrop(224),
                                       transforms.RandomRotation(35),
                                       transforms.RandomVerticalFlip(0.27),
                                       transforms.RandomHorizontalFlip(0.27),
                                       transforms.ToTensor(),
                                       transforms.Normalize([0.485, 0.456, 0.406],

valid_n_test_transforms = transforms.Compose([transforms.Resize((224,224)),
                                        transforms.ToTensor(),
                                        transforms.Normalize([0.485, 0.456, 0.406],
```

To load the data, the generally workflow is: data → `datasets` object to transform → `DataLoader` object to feed to model. Here we will use the class `ImageFolder` to load the dataset from folder. The docs species file that the file should be arranged in the sequence

```
root/dog/xxx.png
root/dog/xxy.png
root/dog/[...]/xxz.png

root/cat/123.png
root/cat/nsdf3.png
root/cat/[...]/asd932_.png
```

which exactly what I have done above.

In [ ]:
```
data_dir = Path('data')
train_dir = data_dir/'train'
valid_dir = data_dir/'valid'
test_dir = data_dir/'test'
```

In [ ]:

```
train_dataset = datasets.ImageFolder(train_dir, transform = train_transforms)
valid_dataset = datasets.ImageFolder(valid_dir, transform = valid_n_test_transforr
test_dataset = datasets.ImageFolder(test_dir, transform = valid_n_test_transforms)

print(train_dataset)
print(valid_dataset)
print(test_dataset)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=128, shuffle=
valid_loader = torch.utils.data.DataLoader(valid_dataset, batch_size=128)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=128)
```

```
Dataset ImageFolder
    Number of datapoints: 56762
    Root location: data/train
    StandardTransform
Transform: Compose(
               RandomResizedCrop(size=(224, 224), scale=(0.08, 1.0), ratio=(0.75,
1.3333), interpolation=bilinear), antialias=None)
               RandomRotation(degrees=[-35.0, 35.0], interpolation=nearest, expand
=False, fill=0)
               RandomVerticalFlip(p=0.27)
               RandomHorizontalFlip(p=0.27)
               ToTensor()
               Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
           )
Dataset ImageFolder
    Number of datapoints: 18988
    Root location: data/valid
    StandardTransform
Transform: Compose(
               Resize(size=(224, 224), interpolation=bilinear, max_size=None, anti
alias=None)
               ToTensor()
               Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
           )
Dataset ImageFolder
    Number of datapoints: 25250
    Root location: data/test
    StandardTransform
Transform: Compose(
               Resize(size=(224, 224), interpolation=bilinear, max_size=None, anti
alias=None)
               ToTensor()
               Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
           )
```

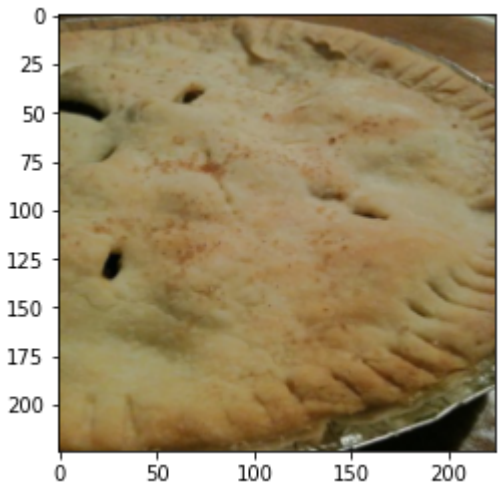
Now let's define a function to visualize the image from the DataLoader to make sure that we do everything right. One quirk of PyTorch is that the pixel in an image are arranged (color channels, width, height) while matplotlib expects it to be (width, height, color channels) so we will need to move data around with `permute((1,2,0))` . We also need to undo the normalization we have done above, and clipping the pixel values to remove noise.

In [ ]:

```
def imshow(image, ax=None, title=None):
    """Imshow for Tensor."""
    if ax is None:
        fig, ax = plt.subplots()

    # PyTorch tensors assume the color channel is the first dimension
    # but matplotlib assumes is the third dimension
    image = image.permute((1, 2, 0))

    # Undo preprocessing
    mean = torch.tensor([0.485, 0.456, 0.406])
    std = torch.tensor([0.229, 0.224, 0.225])
    image = (std * image + mean)
    # Image needs to be clipped between 0 and 1
    # or it looks like noise when displayed
    # (Something I learnt from Udacity)
    image = torch.clip(image, 0, 1)

    ax.imshow(image)

    return ax
```

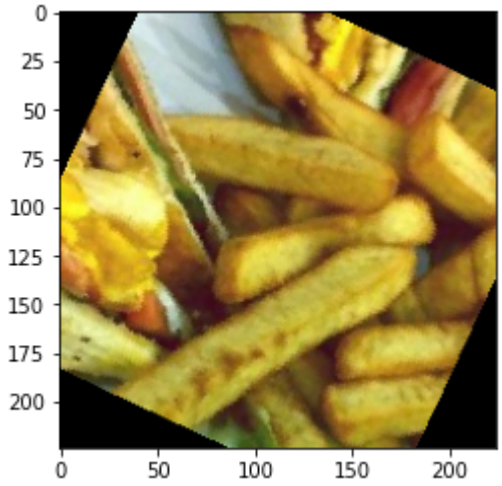
In [ ]:

```
# A test image
imshow(next(iter(test_loader))[0][1]);
```



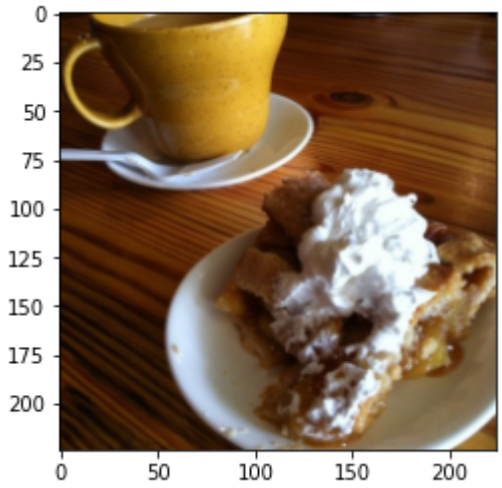
In [ ]:

```
# A training image
imshow(next(iter(train_loader))[0][1]);
# Look at the rotation from data augmentation!
```



In [ ]:

```
# A validation image
imshow(next(iter(valid_loader))[0][1]);
```



That's basically it for this notebook! Now let's move on to the next notebook, where we will create a model and start training!