In [1]:
```
!cp -r ../input/vittutorialillustrations/* ./

!pip install nb_black
%load_ext nb_black
```

```
Collecting nb_black
  Downloading nb_black-1.0.7.tar.gz (4.8 kB)
Requirement already satisfied: ipython in /opt/conda/lib/python3.7/site-packages (from nb_black) (7.1
3.0)
Requirement already satisfied: black>='19.3' in /opt/conda/lib/python3.7/site-packages (from nb_blac
k) (19.10b0)
Requirement already satisfied: jedi>=0.10 in /opt/conda/lib/python3.7/site-packages (from ipython->nb
_black) (0.15.2)
Requirement already satisfied: traitlets>=4.2 in /opt/conda/lib/python3.7/site-packages (from ipython
->nb_black) (4.3.3)
Requirement already satisfied: pickleshare in /opt/conda/lib/python3.7/site-packages (from ipython->n
b_black) (0.7.5)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /opt/conda/lib/python
3.7/site-packages (from ipython->nb_black) (3.0.5)
Requirement already satisfied: pexpect; sys_platform != "win32" in /opt/conda/lib/python3.7/site-pack
ages (from ipython->nb_black) (4.8.0)
Requirement already satisfied: pygments in /opt/conda/lib/python3.7/site-packages (from ipython->nb_b
lack) (2.6.1)
Requirement already satisfied: backcall in /opt/conda/lib/python3.7/site-packages (from ipython->nb_b
lack) (0.1.0)
Requirement already satisfied: setuptools>=18.5 in /opt/conda/lib/python3.7/site-packages (from ipyth
on->nb_black) (46.1.3.post20200325)
Requirement already satisfied: decorator in /opt/conda/lib/python3.7/site-packages (from ipython->nb_
black) (4.4.2)
Requirement already satisfied: appdirs in /opt/conda/lib/python3.7/site-packages (from black>='19.3'-
>nb_black) (1.4.3)
Requirement already satisfied: attrs>=18.1.0 in /opt/conda/lib/python3.7/site-packages (from black>
='19.3'->nb_black) (19.3.0)
Requirement already satisfied: click>=6.5 in /opt/conda/lib/python3.7/site-packages (from black>='19.
3'->nb_black) (7.1.1)
Requirement already satisfied: toml>=0.9.4 in /opt/conda/lib/python3.7/site-packages (from black>='1
9.3'->nb_black) (0.10.0)
Requirement already satisfied: typed-ast>=1.4.0 in /opt/conda/lib/python3.7/site-packages (from black
>='19.3'->nb_black) (1.4.1)
Requirement already satisfied: regex in /opt/conda/lib/python3.7/site-packages (from black>='19.3'->n
b_black) (2020.4.4)
Requirement already satisfied: pathspec<1,>=0.6 in /opt/conda/lib/python3.7/site-packages (from black
>='19.3'->nb_black) (0.8.0)
Requirement already satisfied: parso>=0.5.2 in /opt/conda/lib/python3.7/site-packages (from jedi>=0.1
0->ipython->nb_black) (0.5.2)
Requirement already satisfied: ipython-genutils in /opt/conda/lib/python3.7/site-packages (from trait
lets>=4.2->ipython->nb_black) (0.2.0)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages (from traitlets>=4.2->ip
ython->nb_black) (1.14.0)
Requirement already satisfied: wcwidth in /opt/conda/lib/python3.7/site-packages (from prompt-toolki
t!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython->nb_black) (0.1.9)
Requirement already satisfied: ptyprocess>=0.5 in /opt/conda/lib/python3.7/site-packages (from pexpec
t; sys_platform != "win32"->ipython->nb_black) (0.6.0)
Building wheels for collected packages: nb-black
  Building wheel for nb-black (setup.py) ... - \ done
  Created wheel for nb-black: filename=nb_black-1.0.7-py3-none-any.whl size=5280 sha256=df9d43f8b50bb
748b70902d77c44c28c89842ee1c713afc0c6e4bad04efa5cc9
  Stored in directory: /root/.cache/pip/wheels/1e/b2/88/51c66d23ea5fd0d40ed50997555e15d981d92671376a9
a412a
Successfully built nb-black
Installing collected packages: nb-black
Successfully installed nb-black-1.0.7
WARNING: You are using pip version 20.1.1; however, version 20.3 is available.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip install --upgrade pip' comman
d.
```

Introduction

This notebook essentially has two parts as seen from the title.

1. Vision Transformer: A gentle introduction
2. Implementation in PyTorch

I'll briefly try to explain the fundamental ideas behind Vision Transformers and how it works before getting into the implementation of ViT in PyTorch for this competition.

So if you are only interested in code, you can feel free to skip straight ahead to the second section of this notebook. The implementation isn't a whole lot different, thanks to rwightman/pytorch-image-models library which contains all the model implementations including the pretrained weights for us to use.

If you find this notebook useful, leave an upvote, that motivates me to write more such notebooks.

Vision Transformers: A gentle introduction

Vision Transformers were first introduced in the paper AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE by the Google Brain team in late October 2020.

To understand how ViT works, obviously you must have prior knowledge about how Transformers work and what problems it solved. I'll briefly introduce you to how transformers work before getting into the details of the topic at hand - ViT.

ViT-Illustration

If you are new to NLP and interested in learning more about the transformer models and get a fair intuition of how they actually work, I recomment checking out the fantastic blog posts of Jay Allamar. (The image above was also inspired from one of his blog posts)

Transformers: A brief overview

> **If you already understand Transformers, feel free to skip ahead to the next section.**

Transformer models well and truly revolutionized Natural Language Processing as we know. When they were first introduced they broke multiple NLP records and were pushing the then State of the Art. Now, they have become a de-facto standard for modern NLP tasks and they bring spectacular performance gains when compared to the previous generation of models like LSTMs and GRUs.

By far the most important paper that transformed the NLP landscape is the "Attention is all you need" paper. The transformer architecture was introduced in this paper.

Motivations:

The existing models at that time for sequence and NLP tasks mostly involved RNNs. **The problem with these networks were that they couldn't capture long term dependencies.**

LSTMs and GRUs - variants of RNNs were capable of capturing the dependencies but it is also limited.

So, the main inspiration behind the transformer was to get rid of this recurrence and still end up capturing almost all the dependencies, to be precise global dependencies, yes the reference window of tranformers is full-range. This was achieved using a variant of attention mechanism called *self-attention* (multi-headed) which is very important for their success. One other advantage of Tranformer models are that they are highly parallelizable.

Transformer Architecture

Note: The architecture diagrams are annotated with the corresponding step in the explanation.

TranformerArchitecture

- Transformer has two parts, the decoder which is on the left side on the above diagram and the encoder which is on the right.
- Imagine we are doing machine translation for now.
- The encoder takes the input data (sentence), and produces an intermediate representation of the input.
- The decoder decodes this intermediate representation step by step and generates the output. The difference however is in how it is doing this.
- Understanding the Encoder section is enough for ViT.

> **Note: The explanations here are more about the intuition behind the architectures. For more mathematical details check out the respective research papers instead.**

Tranformers: Step by step overview

(1) The input data first gets embedded into a vector. The embedding layer helps us grab a learned vector representation for each word.

(2) In the next stage a positional encoding is injected into the input embeddings. This is because a transformer has no idea about the order of the sequence that is being passed as input - for example a sentence.

(3) Now the multi-headed attention is where things get a little different.

Multi-headed-attention architecture:

multi-headed-attn

(4) Multi-Headed Attention consists of three learnable vectors. Query, Key and Value vectors. The motivation of this reportedly comes from information retrival where you search (query) and the search engine compares your query with a key and responds with a value.

(5) The Q and K representations undergo a dot product matrix multiplication to produce a score matrix which represents how much a word has to attend to every other word. Higher score means more attention and vice-versa.

(6) Then the Score matrix is scaled down according to the dimensions of the Q and K vectors. This is to ensure more stable gradients as multiplication can have exploding effects.

(We'll discuss the mask part when we reach the decoder section)

(7) Next the Score matrix is softmaxed to turn attention scores into probabilities. Obviously higher scores are heightened and lower scores are depressed. This ensures the model to be confident on which words to attend to.

(8) Then the resultant matrix with probabilites is multiplied with the value vector. This will make the higher probaility scores the model has learned to be more important. The low scoring words will effectively drown out to become irrelevant.

(9) Then, the concatenated output of QK and V vectors are fed into the Linear layer to process further.

(10) Self-Attention is performed for each word in the sequence. Since one doesn't depend on the other a copy of the self attention module can be used to process everything simultaneously making this **multi-headed**.

(11) Then the output value vectors are concatenated and added to the residual connection coming from the input layer and then the resultant respresentation is passed into a *LayerNorm* for normalization. (Residual connection help gradients flow through the network and LayernNorm helps reduce the training time by a small fraction and stabilize the network)

(12) Further, the output is passed into a point-wise feed forward network to obtain an even richer representation.

(13) The outputs are again Layer-normed and residuals are added from the previous layer.

Note: This wraps up the encoder section and trust me this is enough to fully understand Vision Transformer. I'll be largely leaving it up to you to understand the decoder part as it is very similar to the encoding layer.

(14) The output from the encoder along with the inputs (if any) from the previous time steps/words are fed into the decoder where the outputs undergo masked-multi headed attention before being fed into the next attention layer along with the output from encoder.

(15) Masked multi headed attention is necessary because the network shouldn't have any visibility into the words that are to come later in the sequence while decoding, to ensure there is no leak. This is done by masking the entries of words that come later in the series in the Score matrix. Current and previous words in the sequence are added with 1 and the future word scores are added with `-inf`. This ensures the future words in the series get drowned out into 0 when performing softmax to obtain the probabilities, while the rest are retained.

(16) There are residual connections here as well, to improve the flow of gradients. Finally the output is sent to a Linear layer and softmaxed to obtain the outputs in probabilities.

How Vision Tranformers works?

Now that we have covered transformers' internal working at a high level, we are finally ready to tackle Vision Tranformers.

Applying Transformers on images was always going to be a challenge for the following reasons,

- Unlike words/sentences/paragraphs, images contain much much more information in them basically in form of pixels.
- It would be very hard, even with current hardware to attend to every other pixel in the image.
- Instead, a popular alternative was to use localized attention.
- In fact CNNs do something very similar through convolutions and the receptive field essentially grows bigger as we go deeper into the model's layers, but Tranformers were always going to be computationally more expensive than CNNs because of the' nature of Transformers. And of course, we know how incredibly much CNNs have contributed to the current advancements in Computer Vision.

Google researchers have proposed something different in their paper than can possibly be the next big step in Computer Vision. They show that the reliance on CNNs may not be necessary anymore. So, let's dive right in and explore more about Vision Transformers.

Vision Transformer Architecture

vit-architecture

(1) They are only using the Encoder part of the transformer but the difference is in how they are feeding the images into the network.

(2) They are breaking down the image into fixed size patches. So one of these patches can be of dimension 16x16 or 32x32 as proposed in the paper. More patches means more simpler it is to train these networks as the patches themselves get smaller. Hence we have that in the title - "An Image is worth 16x16 words".

(3) The patches are then unrolled (flattened) and sent for further processing into the network.

(4) Unlike NNs here the model has no idea whatsoever about the position of the samples in the sequence, here each sample is a patch from the input image. So the image is fed **along with a positional embedding vector** and into the encoder. One thing to note here is the positional embeddings are also learnable so you don't actually feed hard-coded vectors w.r.t to their positions.

(5) There is also a special token at the start just like BERT.

(6) So each image patch is first unrolled (flattened) into a big vector and gets multiplied with an embedding matrix which is also learnable, creating embedded patches. And these embedded patches are combined with the positional embedding vector and that gets fed into the Tranformer.

> **Note: From here everything is just the same as a standard transformer**

(7) With the only difference being, instead of a decoder the output from the encoder is passed directly into a Feed Forward Neural Network to obtain the classification output.

Things to note:

- The paper ALMOST completely neglects Convolutions.
- They are however using a couple of variants of ViT in which Convolutional embeddings of image patches are used. But that doesn't seem to impact performance much.
- Vision Transformers, at the time of writing this are topping Image Classification benchmarks on ImageNet.

  There are a lot more interesting things in this paper but the one thing that stands out for me and potentially shows the power of transformers over CNNs is illustrated in the image below which shows the attention distance with respect to the layers.

- The graph above suggests that the Transformers are already capable of paying attention to regions that are far apart right from the starting layers of the network which is a pretty significant gain the Transformers bring over CNNs which has a finite receptive field at the start.

If you find this notebook useful, leave an upvote, that motivates me to write more such notebooks.

Vision Transformer Implementation in PyTorch

Now that you understand vision transformers, let's build a baseline model for this competition

First lets install torch-xla to be able to use the TPU and torch-image-models (timm).

In [2]:
```
!curl https://raw.githubusercontent.com/pytorch/xla/master/contrib/scripts/env-setup.py -o pytorch-xla-
env-setup.py
!python pytorch-xla-env-setup.py --version 1.7
!pip install timm
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  5116  100  5116    0     0  16088      0 --:--:-- --:--:-- --:--:-- 16037
Updating... This may take around 2 minutes.
Updating TPU runtime to pytorch-1.7 ...
Found existing installation: torch 1.5.0
Uninstalling torch-1.5.0:
  Successfully uninstalled torch-1.5.0
Found existing installation: torchvision 0.6.0a0+35d732a
Uninstalling torchvision-0.6.0a0+35d732a:
Done updating TPU runtime
  Successfully uninstalled torchvision-0.6.0a0+35d732a
Copying gs://tpu-pytorch/wheels/torch-1.7-cp37-cp37m-linux_x86_64.whl...

Operation completed over 1 objects/114.2 MiB.
Copying gs://tpu-pytorch/wheels/torch_xla-1.7-cp37-cp37m-linux_x86_64.whl...

Operation completed over 1 objects/127.4 MiB.
Copying gs://tpu-pytorch/wheels/torchvision-1.7-cp37-cp37m-linux_x86_64.whl...

Operation completed over 1 objects/3.1 MiB.
Processing ./torch-1.7-cp37-cp37m-linux_x86_64.whl
Collecting dataclasses
  Downloading dataclasses-0.6-py3-none-any.whl (14 kB)
Requirement already satisfied: typing-extensions in /opt/conda/lib/python3.7/site-packages (from torc
h==1.7) (3.7.4.1)
Requirement already satisfied: numpy in /opt/conda/lib/python3.7/site-packages (from torch==1.7) (1.1
8.5)
Requirement already satisfied: future in /opt/conda/lib/python3.7/site-packages (from torch==1.7) (0.
18.2)
ERROR: fastai 1.0.61 requires torchvision, which is not installed.
ERROR: kornia 0.3.1 has requirement torch==1.5.0, but you'll have torch 1.7.0a0 which is incompatibl
e.
ERROR: allennlp 1.0.0 has requirement torch<1.6.0,>=1.5.0, but you'll have torch 1.7.0a0 which is inc
ompatible.
Installing collected packages: dataclasses, torch
Successfully installed dataclasses-0.6 torch-1.7.0a0
WARNING: You are using pip version 20.1.1; however, version 20.3 is available.
You should consider upgrading via the '/opt/conda/bin/python -m pip install --upgrade pip' command.
Processing ./torch_xla-1.7-cp37-cp37m-linux_x86_64.whl
Installing collected packages: torch-xla
Successfully installed torch-xla-1.7
WARNING: You are using pip version 20.1.1; however, version 20.3 is available.
You should consider upgrading via the '/opt/conda/bin/python -m pip install --upgrade pip' command.
Processing ./torchvision-1.7-cp37-cp37m-linux_x86_64.whl
Requirement already satisfied: torch in /opt/conda/lib/python3.7/site-packages (from torchvision==1.
7) (1.7.0a0)
Requirement already satisfied: numpy in /opt/conda/lib/python3.7/site-packages (from torchvision==1.
7) (1.18.5)
Requirement already satisfied: pillow>=4.1.1 in /opt/conda/lib/python3.7/site-packages (from torchvis
ion==1.7) (7.2.0)
Requirement already satisfied: typing-extensions in /opt/conda/lib/python3.7/site-packages (from torc
h->torchvision==1.7) (3.7.4.1)
Requirement already satisfied: future in /opt/conda/lib/python3.7/site-packages (from torch->torchvis
ion==1.7) (0.18.2)
Requirement already satisfied: dataclasses in /opt/conda/lib/python3.7/site-packages (from torch->tor
chvision==1.7) (0.6)
Installing collected packages: torchvision
Successfully installed torchvision-0.9.0a0+75e4a7d
WARNING: You are using pip version 20.1.1; however, version 20.3 is available.
You should consider upgrading via the '/opt/conda/bin/python -m pip install --upgrade pip' command.


The following NEW packages will be installed:
  libomp5
0 upgraded, 1 newly installed, 0 to remove and 59 not upgraded.
Need to get 234 kB of archives.
After this operation, 774 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu bionic/universe amd64 libomp5 amd64 5.0.1-1 [234 kB]
Fetched 234 kB in 1s (253 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package libomp5:amd64.
(Reading database ... 107745 files and directories currently installed.)
Preparing to unpack .../libomp5_5.0.1-1_amd64.deb ...
Unpacking libomp5:amd64 (5.0.1-1) ...
Setting up libomp5:amd64 (5.0.1-1) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Collecting timm
  Downloading timm-0.3.1-py3-none-any.whl (247 kB)
     |████████████████████████████████| 247 kB 3.0 MB/s
Requirement already satisfied: torch>=1.0 in /opt/conda/lib/python3.7/site-packages (from timm) (1.7.
0a0)
Requirement already satisfied: torchvision in /opt/conda/lib/python3.7/site-packages (from timm) (0.
9.0a0+75e4a7d)
Requirement already satisfied: typing-extensions in /opt/conda/lib/python3.7/site-packages (from torc
h>=1.0->timm) (3.7.4.1)
Requirement already satisfied: numpy in /opt/conda/lib/python3.7/site-packages (from torch>=1.0->tim
m) (1.18.5)
Requirement already satisfied: future in /opt/conda/lib/python3.7/site-packages (from torch>=1.0->tim
m) (0.18.2)
Requirement already satisfied: dataclasses in /opt/conda/lib/python3.7/site-packages (from torch>=1.0
->timm) (0.6)
Requirement already satisfied: pillow>=4.1.1 in /opt/conda/lib/python3.7/site-packages (from torchvis
ion->timm) (7.2.0)
Installing collected packages: timm
Successfully installed timm-0.3.1
WARNING: You are using pip version 20.1.1; however, version 20.3 is available.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip install --upgrade pip' comman
d.
```

In [3]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

plt.style.use("ggplot")

import torch
import torch.nn as nn
import torchvision.transforms as transforms

import torch_xla
import torch_xla.core.xla_model as xm
import torch_xla.distributed.xla_multiprocessing as xmp
import torch_xla.distributed.parallel_loader as pl

import timm

import gc
import os
import time
import random
from datetime import datetime

from PIL import Image
from tqdm.notebook import tqdm
from sklearn import model_selection, metrics
```

In [4]:
```python
# For parallelization in TPUs
os.environ["XLA_USE_BF16"] = "1"
os.environ["XLA_TENSOR_ALLOCATOR_MAXSIZE"] = "100000000"
```

In [5]:
```python
def seed_everything(seed):
    """
    Seeds basic parameters for reproductibility of results

    Arguments:
        seed {int} -- Number of the seed
    """
    random.seed(seed)
    os.environ["PYTHONHASHSEED"] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False


seed_everything(1001)
```

In [6]:
```python
# general global variables
DATA_PATH = "../input/cassava-leaf-disease-classification"
TRAIN_PATH = "../input/cassava-leaf-disease-classification/train_images/"
TEST_PATH = "../input/cassava-leaf-disease-classification/test_images/"
MODEL_PATH = (
    "../input/vit-base-models-pretrained-pytorch/jx_vit_base_p16_224-80ecf9dd.pth"
)

# model specific global variables
IMG_SIZE = 224
BATCH_SIZE = 16
LR = 2e-05
GAMMA = 0.7
N_EPOCHS = 10
```

In [7]:
```python
df = pd.read_csv(os.path.join(DATA_PATH, "train.csv"))
df.head()
```

Out[7]:

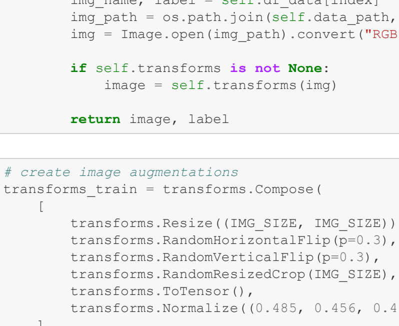
	image_id	label
0	1000015157.jpg	0
1	1000201771.jpg	3
2	100042118.jpg	1
3	1000723321.jpg	1
4	1000812911.jpg	3

In [8]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21397 entries, 0 to 21396
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   image_id  21397 non-null  object
 1   label     21397 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 334.5+ KB
```

In [9]:
```python
df.label.value_counts().plot(kind="bar")
```

Out[9]: `<matplotlib.axes._subplots.AxesSubplot at 0x7f921783b6d0>`



In [10]:
```python
train_df, valid_df = model_selection.train_test_split(
    df, test_size=0.1, random_state=42, stratify=df.label.values
)
```

In [11]:
```python
class CassavaDataset(torch.utils.data.Dataset):
    """
    Helper Class to create the pytorch dataset
    """

    def __init__(self, df, data_path=DATA_PATH, mode="train", transforms=None):
        super().__init__()
        self.df_data = df.values
        self.data_path = data_path
        self.transforms = transforms
        self.mode = mode
        self.data_dir = "train_images" if mode == "train" else "test_images"

    def __len__(self):
        return len(self.df_data)

    def __getitem__(self, index):
        img_name, label = self.df_data[index]
        img_path = os.path.join(self.data_path, self.data_dir, img_name)
        img = Image.open(img_path).convert("RGB")

        if self.transforms is not None:
            image = self.transforms(img)

        return image, label
```

In [12]:
```python
# create image augmentations
transforms_train = transforms.Compose(
    [
        transforms.Resize((IMG_SIZE, IMG_SIZE)),
        transforms.RandomHorizontalFlip(p=0.3),
        transforms.RandomVerticalFlip(p=0.3),
        transforms.RandomResizedCrop(IMG_SIZE),
        transforms.ToTensor(),
        transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),
    ]
)

transforms_valid = transforms.Compose(
    [
        transforms.Resize((IMG_SIZE, IMG_SIZE)),
        transforms.ToTensor(),
        transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),
    ]
)
```

In [13]:
```python
print("Available Vision Transformer Models: ")
timm.list_models("vit*")
```

```
Available Vision Transformer Models:
```

Out[13]:
```
['vit_base_patch16_224',
 'vit_base_patch16_384',
 'vit_base_patch32_384',
 'vit_base_resnet26d_224',
 'vit_base_resnet50d_224',
 'vit_huge_patch16_224',
 'vit_huge_patch32_384',
 'vit_large_patch16_224',
 'vit_large_patch16_384',
 'vit_large_patch32_384',
 'vit_small_patch16_224',
 'vit_small_resnet26d_224',
 'vit_small_resnet50d_s3_224']
```

In [14]:
```python
class ViTBase16(nn.Module):
    def __init__(self, n_classes, pretrained=False):

        super(ViTBase16, self).__init__()

        self.model = timm.create_model("vit_base_patch16_224", pretrained=False)
        if pretrained:
            self.model.load_state_dict(torch.load(MODEL_PATH))

        self.model.head = nn.Linear(self.model.head.in_features, n_classes)

    def forward(self, x):
        x = self.model(x)
        return x

    def train_one_epoch(self, train_loader, criterion, optimizer, device):
        # keep track of training loss
        epoch_loss = 0.0
        epoch_accuracy = 0.0

        ###################
        # train the model #
        ###################
        self.model.train()
        for i, (data, target) in enumerate(train_loader):
            # move tensors to GPU if CUDA is available
            if device.type == "cuda":
                data, target = data.cuda(), target.cuda()
            elif device.type == "xla":
                data = data.to(device, dtype=torch.float32)
                target = target.to(device, dtype=torch.int64)

            # clear the gradients of all optimized variables
            optimizer.zero_grad()
            # forward pass: compute predicted outputs by passing inputs to the model
            output = self.forward(data)
            # calculate the batch loss
            loss = criterion(output, target)
            # backward pass: compute gradient of the loss with respect to model parameters
            loss.backward()
            # Calculate Accuracy
            accuracy = (output.argmax(dim=1) == target).float().mean()
            # update training loss and accuracy
            epoch_loss += loss
            epoch_accuracy += accuracy

            # perform a single optimization step (parameter update)
            if device.type == "xla":
                xm.optimizer_step(optimizer)

                if i % 20 == 0:
                    xm.master_print(f"\tBATCH {i+1}/{len(train_loader)} - LOSS: {loss}")

            else:
                optimizer.step()

        return epoch_loss / len(train_loader), epoch_accuracy / len(train_loader)

    def validate_one_epoch(self, valid_loader, criterion, device):
        # keep track of validation loss
        valid_loss = 0.0
        valid_accuracy = 0.0

        ######################
        # validate the model #
        ######################
        self.model.eval()
        for data, target in valid_loader:
            # move tensors to GPU if CUDA is available
            if device.type == "cuda":
                data, target = data.cuda(), target.cuda()
            elif device.type == "xla":
                data = data.to(device, dtype=torch.float32)
                target = target.to(device, dtype=torch.int64)

            with torch.no_grad():
                # forward pass: compute predicted outputs by passing inputs to the model
                output = self.model(data)
                # calculate the batch loss
                loss = criterion(output, target)
                # Calculate Accuracy
                accuracy = (output.argmax(dim=1) == target).float().mean()
                # update average validation loss and accuracy
                valid_loss += loss
                valid_accuracy += accuracy

        return valid_loss / len(valid_loader), valid_accuracy / len(valid_loader)
```

In [15]:
```python
def fit_tpu(
    model, epochs, device, criterion, optimizer, train_loader, valid_loader=None
):

    valid_loss_min = np.Inf  # track change in validation loss

    # keeping track of losses as it happen
    train_losses = []
    valid_losses = []
    train_accs = []
    valid_accs = []

    for epoch in range(1, epochs + 1):
        gc.collect()
        para_train_loader = pl.ParallelLoader(train_loader, [device])

        xm.master_print(f"{'='*50}")
        xm.master_print(f"EPOCH {epoch} - TRAINING...")
        train_loss, train_acc = model.train_one_epoch(
            para_train_loader.per_device_loader(device), criterion, optimizer, device
        )
        xm.master_print(
            f"\n\t[TRAIN] EPOCH {epoch} - LOSS: {train_loss}, ACCURACY: {train_acc}\n"
        )
        train_losses.append(train_loss)
        train_accs.append(train_acc)
        gc.collect()

        if valid_loader is not None:
            gc.collect()
            para_valid_loader = pl.ParallelLoader(valid_loader, [device])
            xm.master_print(f"EPOCH {epoch} - VALIDATING...")
            valid_loss, valid_acc = model.validate_one_epoch(
                para_valid_loader.per_device_loader(device), criterion, device
            )
            xm.master_print(f"\t[VALID] LOSS: {valid_loss}, ACCURACY: {valid_acc}\n")
            valid_losses.append(valid_loss)
            valid_accs.append(valid_acc)
            gc.collect()

            # save model if validation loss has decreased
            if valid_loss <= valid_loss_min and epoch != 1:
                xm.master_print(
                    "Validation loss decreased ({:.4f} --> {:.4f}).  Saving model ...".format(
                        valid_loss_min, valid_loss
                    )
                )
            #                 xm.save(model.state_dict(), 'best_model.pth')

            valid_loss_min = valid_loss

    return {
        "train_loss": train_losses,
        "valid_losses": valid_losses,
        "train_acc": train_accs,
        "valid_acc": valid_accs,
    }
```

In [16]:
```python
model = ViTBase16(n_classes=5, pretrained=True)
```

In [17]:
```python
def _run():
    train_dataset = CassavaDataset(train_df, transforms=transforms_train)
    valid_dataset = CassavaDataset(valid_df, transforms=transforms_valid)

    train_sampler = torch.utils.data.distributed.DistributedSampler(
        train_dataset,
        num_replicas=xm.xrt_world_size(),
        rank=xm.get_ordinal(),
        shuffle=True,
    )

    valid_sampler = torch.utils.data.distributed.DistributedSampler(
        valid_dataset,
        num_replicas=xm.xrt_world_size(),
        rank=xm.get_ordinal(),
        shuffle=False,
    )

    train_loader = torch.utils.data.DataLoader(
        dataset=train_dataset,
        batch_size=BATCH_SIZE,
        sampler=train_sampler,
        drop_last=True,
        num_workers=8,
    )

    valid_loader = torch.utils.data.DataLoader(
        dataset=valid_dataset,
        batch_size=BATCH_SIZE,
        sampler=valid_sampler,
        drop_last=True,
        num_workers=8,
    )

    criterion = nn.CrossEntropyLoss()
    #     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    device = xm.xla_device()
    model.to(device)

    lr = LR * xm.xrt_world_size()
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)

    xm.master_print(f"INITIALIZING TRAINING ON {xm.xrt_world_size()} TPU CORES")
    start_time = datetime.now()
    xm.master_print(f"Start Time: {start_time}")

    logs = fit_tpu(
        model=model,
        epochs=N_EPOCHS,
        device=device,
        criterion=criterion,
        optimizer=optimizer,
        train_loader=train_loader,
        valid_loader=valid_loader,
    )

    xm.master_print(f"Execution time: {datetime.now() - start_time}")

    xm.master_print("Saving Model")
    xm.save(
        model.state_dict(), f'model_5e_{datetime.now().strftime("%Y%m%d-%H%M")}.pth'
    )
```

In [18]:
```python
# Start training processes
def _mp_fn(rank, flags):
    torch.set_default_tensor_type("torch.FloatTensor")
    a = _run()


# _run()
FLAGS = {}
xmp.spawn(_mp_fn, args=(FLAGS,), nprocs=8, start_method="fork")
```

```
INITIALIZING TRAINING ON 8 TPU CORES
Start Time: 2020-12-02 18:34:26.339168
==================================================
EPOCH 1 - TRAINING...
        BATCH 1/150 - LOSS: 1.609375
        BATCH 21/150 - LOSS: 0.7734375
        BATCH 41/150 - LOSS: 0.33984375
        BATCH 61/150 - LOSS: 0.333984375
        BATCH 81/150 - LOSS: 0.314453125
        BATCH 101/150 - LOSS: 0.765625
        BATCH 121/150 - LOSS: 0.439453125
        BATCH 141/150 - LOSS: 0.451171875

        [TRAIN] EPOCH 1 - LOSS: 0.59765625, ACCURACY: 0.83203125

EPOCH 1 - VALIDATING...
        [VALID] LOSS: 0.419921875, ACCURACY: 0.84375

==================================================
EPOCH 2 - TRAINING...
        BATCH 1/150 - LOSS: 0.458984375
        BATCH 21/150 - LOSS: 0.2734375
        BATCH 41/150 - LOSS: 0.1396484375
        BATCH 61/150 - LOSS: 0.177734375
        BATCH 81/150 - LOSS: 0.14453125
        BATCH 101/150 - LOSS: 0.5703125
        BATCH 121/150 - LOSS: 0.32421875
        BATCH 141/150 - LOSS: 0.60546875

        [TRAIN] EPOCH 2 - LOSS: 0.41796875, ACCURACY: 0.9140625

EPOCH 2 - VALIDATING...
        [VALID] LOSS: 0.4140625, ACCURACY: 0.83984375

Validation loss decreased (0.4199 --> 0.4141).  Saving model ...
==================================================
EPOCH 3 - TRAINING...
        BATCH 1/150 - LOSS: 0.20703125
        BATCH 21/150 - LOSS: 0.5625
        BATCH 41/150 - LOSS: 0.15625
        BATCH 61/150 - LOSS: 0.2373046875
        BATCH 81/150 - LOSS: 0.208984375
        BATCH 101/150 - LOSS: 0.435546875
        BATCH 121/150 - LOSS: 0.28515625
        BATCH 141/150 - LOSS: 0.337890625

        [TRAIN] EPOCH 3 - LOSS: 0.39453125, ACCURACY: 0.9140625

EPOCH 3 - VALIDATING...
        [VALID] LOSS: 0.404296875, ACCURACY: 0.83984375

Validation loss decreased (0.4141 --> 0.4043).  Saving model ...
==================================================
EPOCH 4 - TRAINING...
        BATCH 1/150 - LOSS: 0.17578125
        BATCH 21/150 - LOSS: 0.15625
        BATCH 41/150 - LOSS: 0.154296875
        BATCH 61/150 - LOSS: 0.11865234375
        BATCH 81/150 - LOSS: 0.1298828125
        BATCH 101/150 - LOSS: 0.36328125
        BATCH 121/150 - LOSS: 0.208984375
        BATCH 141/150 - LOSS: 0.333984375

        [TRAIN] EPOCH 4 - LOSS: 0.369140625, ACCURACY: 0.921875

EPOCH 4 - VALIDATING...
        [VALID] LOSS: 0.3984375, ACCURACY: 0.859375

Validation loss decreased (0.4043 --> 0.3984).  Saving model ...
==================================================
EPOCH 5 - TRAINING...
        BATCH 1/150 - LOSS: 0.5390625
        BATCH 21/150 - LOSS: 0.1865234375
        BATCH 41/150 - LOSS: 0.263671875
        BATCH 61/150 - LOSS: 0.1328125
        BATCH 81/150 - LOSS: 0.10498046875
        BATCH 101/150 - LOSS: 0.390625
        BATCH 121/150 - LOSS: 0.2890625
        BATCH 141/150 - LOSS: 0.318359375

        [TRAIN] EPOCH 5 - LOSS: 0.357421875, ACCURACY: 0.921875

EPOCH 5 - VALIDATING...
        [VALID] LOSS: 0.416015625, ACCURACY: 0.84375

==================================================
EPOCH 6 - TRAINING...
        BATCH 1/150 - LOSS: 0.51953125
        BATCH 21/150 - LOSS: 0.27734375
        BATCH 41/150 - LOSS: 0.2138671875
        BATCH 61/150 - LOSS: 0.1630859375
        BATCH 81/150 - LOSS: 0.19140625
        BATCH 101/150 - LOSS: 0.60546875
        BATCH 121/150 - LOSS: 0.271484375
        BATCH 141/150 - LOSS: 0.1943359375

        [TRAIN] EPOCH 6 - LOSS: 0.34375, ACCURACY: 0.92578125

EPOCH 6 - VALIDATING...
        [VALID] LOSS: 0.41796875, ACCURACY: 0.8515625

==================================================
EPOCH 7 - TRAINING...
        BATCH 1/150 - LOSS: 0.3125
        BATCH 21/150 - LOSS: 0.30078125
        BATCH 41/150 - LOSS: 0.2041015625
        BATCH 61/150 - LOSS: 0.3515625
        BATCH 81/150 - LOSS: 0.0849609375
        BATCH 101/150 - LOSS: 0.408203125
        BATCH 121/150 - LOSS: 0.146484375
        BATCH 141/150 - LOSS: 0.2001953125

        [TRAIN] EPOCH 7 - LOSS: 0.30078125, ACCURACY: 0.94140625

EPOCH 7 - VALIDATING...
        [VALID] LOSS: 0.40625, ACCURACY: 0.85546875

Validation loss decreased (0.4180 --> 0.4062).  Saving model ...
==================================================
EPOCH 8 - TRAINING...
        BATCH 1/150 - LOSS: 0.16015625
        BATCH 21/150 - LOSS: 0.181640625
        BATCH 41/150 - LOSS: 0.083984375
        BATCH 61/150 - LOSS: 0.142578125
        BATCH 81/150 - LOSS: 0.11962890625
        BATCH 101/150 - LOSS: 0.2197265625
        BATCH 121/150 - LOSS: 0.1875
        BATCH 141/150 - LOSS: 0.33984375

        [TRAIN] EPOCH 8 - LOSS: 0.28515625, ACCURACY: 0.94140625

EPOCH 8 - VALIDATING...
        [VALID] LOSS: 0.40625, ACCURACY: 0.87109375

Validation loss decreased (0.4062 --> 0.4062).  Saving model ...
==================================================
EPOCH 9 - TRAINING...
        BATCH 1/150 - LOSS: 0.435546875
        BATCH 21/150 - LOSS: 0.39453125
        BATCH 41/150 - LOSS: 0.212890625
        BATCH 61/150 - LOSS: 0.0830078125
        BATCH 81/150 - LOSS: 0.07958984375
        BATCH 101/150 - LOSS: 0.34765625
        BATCH 121/150 - LOSS: 0.279296875
        BATCH 141/150 - LOSS: 0.2470703125

        [TRAIN] EPOCH 9 - LOSS: 0.2890625, ACCURACY: 0.9453125

EPOCH 9 - VALIDATING...
        [VALID] LOSS: 0.41796875, ACCURACY: 0.87109375

==================================================
EPOCH 10 - TRAINING...
        BATCH 1/150 - LOSS: 0.38671875
        BATCH 21/150 - LOSS: 0.2119140625
        BATCH 41/150 - LOSS: 0.06884765625
        BATCH 61/150 - LOSS: 0.1806640625
        BATCH 81/150 - LOSS: 0.0546875
        BATCH 101/150 - LOSS: 0.19140625
        BATCH 121/150 - LOSS: 0.197265625
        BATCH 141/150 - LOSS: 0.328125

        [TRAIN] EPOCH 10 - LOSS: 0.28125, ACCURACY: 0.9453125

EPOCH 10 - VALIDATING...
        [VALID] LOSS: 0.453125, ACCURACY: 0.83984375

Execution time: 0:33:39.824880
Saving Model
```

Thanks a lot for reading all the way

If you find this notebook useful, leave an upvote, that motivates me to write more such notebooks.