Deep Learning Class Assigment

In this assignment, we will build a model for Pokemon Classification model using Kaggle Dataset, which contains 150 different pokemon classes.

Throughout the assignment, you can **freely copy and paste codes from Internet**. At section 2, I provided the first model_0, which will be used as the baseline throughout the assignment. Your actual assignments will start from Section 3.

- main file is **Pokemon.ipynb** (Jupyter notebook file).
- But the notebook file is not good for programming. It's just for distributing the assignment and your submission of the assignment to me when you do the programming, **I recommend to use PyCharm IDE for easy development and debugging.**
- Put comments as necessary to help me understand your code

What to submit

1. This .ipynb file (with all the code) (24 pts)
2. a printout version of this file (with all the code execution results shown)
3. A presentation file for explaing what you have done (may include purpose of each experiment, results, analysis, your opinion) (12 pts)

- please note that the (24pts, 12 pts) will be normalized to get the final 24 pts. Normalized_score= (your score)/(24+12)*24)

1. Data preparation

1. Unzip the PokemonData.zip into the ./PokemonData directory. In the PokemonData directory, it should contains folders starting "Abra", "Aerodactyl".
2. Install split-folders and seaborn package at linux command shell
3. Run the code through to before section 3.
4. You may comment out this 1. Data preparation to prevent if from running again next time

```
# installing the tools necessary to prepare the data into train, val, test dataset
# Alternatively, you may also manually install the split-folders utility at linux shell
!pip install split-folders seaborn tqdm
```

```
'pip'은(는) 내부 또는 외부 명령, 실행할 수 있는 프로그램, 또는
배치 파일이 아닙니다.
```

```
# this should show your assignemnt root directory
os.getcwd()
```

```
'D:\\UST\\01. 강의\\2023\\Deep Learning\\Assignment'
```

split-folders will be used to split dataset into training and validation dataset. Training dataset will be 70% of source dataset and validation will be 10%, test 20%. Although we prepare validation data here, in the code below we don't use it. You may make use of the validation data later through the assignment for hyper parameter tunning.

The splitted data will be in ./input/train, ./input/val, ./input/test

```
# this is for spliting data
import splitfolders

image_dir = './PokemonData'
splitfolders.ratio(image_dir, output="input", seed=1337, ratio=(.7, .1, .2), group_prefix=None, move=False) # default values
cmd="splitfolders.ratio('" + image_dir + "', output='output', seed=1337, ratio=(.8, .1, .1), group_prefix=None, move=False)"
cmd = '%s' % cmd
print(cmd)
```

```
Copying files: 6837 files [00:03, 1738.19 files/s]
```

```
splitfolders.ratio('./PokemonData', output='output', seed=1337,
ratio=(.8, .1, .1), group_prefix=None, move=False)
```

2. Programming Starts here

Let's check the GPU

```
# checking gpu
!nvidia-smi
```

```
Fri May 19 22:13:19 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 516.94       Driver Version: 516.94       CUDA Version: 11.7     |
|-------------------------------+----------------------+----------------------+
| GPU  Name            TCC/WDDM | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ... WDDM  | 00000000:07:00.0  On |                  N/A |
| 24%   66C    P2    85W / 250W |   3097MiB / 12288MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
```

```
|=============================================================================
=============|
|    0   N/A  N/A      6616    C+G   ...5n1h2txyewy\SearchApp.exe
   N/A      |
|    0   N/A  N/A     10804    C+G   C:\Windows\explorer.exe
    N/A      |
|    0   N/A  N/A    199944    C+G   ...2txyewy\TextInputHost.exe
   N/A      |
|    0   N/A  N/A    200432    C+G   ...me\Application\chrome.exe
   N/A      |
|    0   N/A  N/A    210588    C+G   ...gram Desktop\Telegram.exe
   N/A      |
|    0   N/A  N/A    212088    C+G   ...cw5n1h2txyewy\LockApp.exe
   N/A      |
|    0   N/A  N/A    214072    C+G   ...ge\Application\msedge.exe
   N/A      |
|    0   N/A  N/A    216656      C   ...nda3\envs\NNDL\python.exe
   N/A      |
|    0   N/A  N/A    231204    C+G   ...oot\Office16\POWERPNT.EXE
   N/A      |
|    0   N/A  N/A    254588    C+G   ...y\ShellExperienceHost.exe
   N/A      |
|    0   N/A  N/A    255944    C+G   ...e\root\Office16\EXCEL.EXE
   N/A      |
|    0   N/A  N/A    263920    C+G   ...774.42\msedgewebview2.exe
   N/A      |
|    0   N/A  N/A    265676    C+G   ...m Files\Typora\Typora.exe
   N/A      |
+-----------------------------------------------------------------------------
-------------+
```

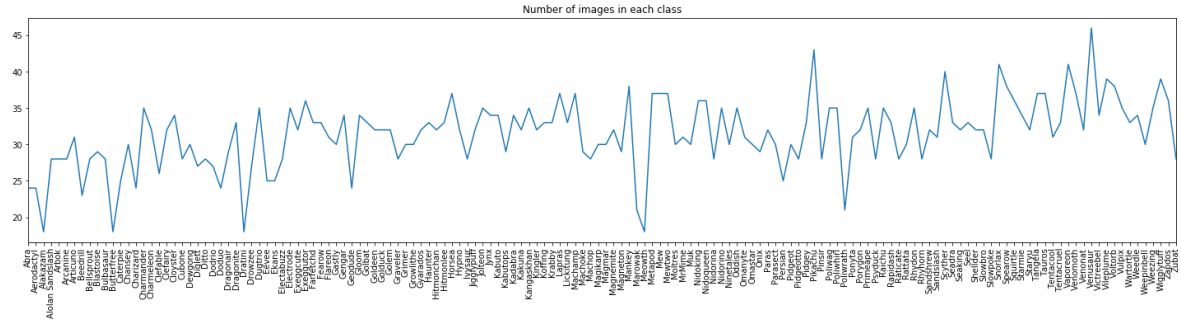
```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import datasets, transforms, models
from torchvision.utils import make_grid

import os
from PIL import Image
from IPython.display import display

import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
import time

%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

At this point your splitted data should be in ./input/train, ./input/test, ./input/val. Let's check the train data below

```
#Check image folder structure
#print(os.listdir(image_dir)[:10])

#Let's check the dataset for each classes

import seaborn as sns

data_dir= './input'
train_dir=data_dir+'/train'
classes = os.listdir(train_dir) # List of all classes
print(f'Total number of categories: {len(classes)}')


counts = {}
for c in classes:
    counts[c] = len(os.listdir(os.path.join(train_dir, c)))
print(f'Total number of images in training dataset:
{sum(list(counts.values()))}')
# Number of images in each clsss plot
fig = plt.figure(figsize = (25, 5))
sns.lineplot(x = list(counts.keys()), y =
list(counts.values())).set_title('Number of images in each
class')
plt.xticks(rotation = 90)
plt.margins(x=0)
plt.show()
```

```
Total number of categories: 150
Total number of images in training dataset: 4716
```




```
# constants
batch_size = 4
#stats = (0.5, 0.5, 0.5), (0.5, 0.5, 0.5)

#torch.manual_seed(42)
epochs = 1 # you better change this. 1 is just for my testing
```

Lets define data transformation

```
train_transform = transforms.Compose([
    transforms.RandomRotation(30),
    transforms.RandomHorizontalFlip(),
    transforms.Resize((240, 240)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

test_transform = transforms.Compose([
    transforms.Resize((240, 240)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])
```

Prepare Test and Train Sets, Loaders First, split the dataset into different train and test folders using split-folders.

```
train_data = datasets.ImageFolder(os.path.join(data_dir,
'train'), transform = train_transform)
val_data = datasets.ImageFolder(os.path.join(data_dir, 'val'),
transform = test_transform)
test_data = datasets.ImageFolder(os.path.join(data_dir, 'test'),
transform = test_transform)
```

```
train_loader = DataLoader(train_data, batch_size=batch_size,
shuffle=True)
val_loader = DataLoader(val_data, batch_size=batch_size,
shuffle=False)
test_loader = DataLoader(test_data, batch_size=batch_size,
shuffle=False)

class_names = train_data.classes
num_classes=len(class_names)

print(f'Num classes: {num_classes}')
print(class_names)
print(f'Training images available: {len(train_data)}')
print(f'Testing images available:  {len(test_data)}')
```

```
Num classes: 150
['Abra', 'Aerodactyl', 'Alakazam', 'Alolan Sandslash', 'Arbok',
'Arcanine', 'Articuno', 'Beedrill', 'Bellsprout', 'Blastoise',
'Bulbasaur', 'Butterfree', 'Caterpie', 'Chansey', 'Charizard',
'Charmander', 'Charmeleon', 'Clefable', 'Clefairy', 'Cloyster',
'Cubone', 'Dewgong', 'Diglett', 'Ditto', 'Dodrio', 'Doduo',
'Dragonair', 'Dragonite', 'Dratini', 'Drowzee', 'Dugtrio',
'Eevee', 'Ekans', 'Electabuzz', 'Electrode', 'Exeggcute',
'Exeggutor', 'Farfetchd', 'Fearow', 'Flareon', 'Gastly',
'Gengar', 'Geodude', 'Gloom', 'Golbat', 'Goldeen', 'Golduck',
'Golem', 'Graveler', 'Grimer', 'Growlithe', 'Gyarados',
'Haunter', 'Hitmonchan', 'Hitmonlee', 'Horsea', 'Hypno',
'Ivysaur', 'Jigglypuff', 'Jolteon', 'Jynx', 'Kabuto', 'Kabutops',
'Kadabra', 'Kakuna', 'Kangaskhan', 'Kingler', 'Koffing',
'Krabby', 'Lapras', 'Lickitung', 'Machamp', 'Machoke', 'Machop',
'Magikarp', 'Magmar', 'Magnemite', 'Magneton', 'Mankey',
'Marowak', 'Meowth', 'Metapod', 'Mew', 'Mewtwo', 'Moltres',
'MrMime', 'Muk', 'Nidoking', 'Nidoqueen', 'Nidorina', 'Nidorino',
'Ninetales', 'Oddish', 'Omanyte', 'Omastar', 'Onix', 'Paras',
'Parasect', 'Persian', 'Pidgeot', 'Pidgeotto', 'Pidgey',
'Pikachu', 'Pinsir', 'Poliwag', 'Poliwhirl', 'Poliwrath',
'Ponyta', 'Porygon', 'Primeape', 'Psyduck', 'Raichu', 'Rapidash',
'Raticate', 'Rattata', 'Rhydon', 'Rhyhorn', 'Sandshrew',
'Sandslash', 'Scyther', 'Seadra', 'Seaking', 'Seel', 'Shellder',
'Slowbro', 'Slowpoke', 'Snorlax', 'Spearow', 'Squirtle',
'Starmie', 'Staryu', 'Tangela', 'Tauros', 'Tentacool',
'Tentacruel', 'Vaporeon', 'Venomoth', 'Venonat', 'Venusaur',
'Victreebel', 'Vileplume', 'Voltorb', 'Vulpix', 'Wartortle',
'Weedle', 'Weepinbell', 'Weezing', 'Wigglytuff', 'Zapdos',
'Zubat']
Training images available: 4703
Testing images available:  1502
```

Check if GPU is available

```
device = torch.device("cuda:0" if torch.cuda.is_available() else
"cpu")
print(device)
print(torch.cuda.get_device_name(device))
```

```
cuda:0
NVIDIA GeForce GTX TITAN X
```

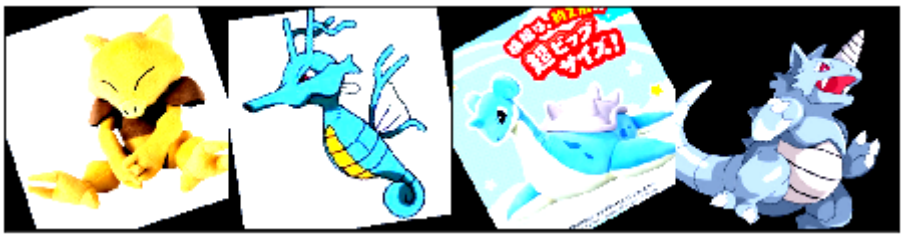
Utility functions

```
def denorm(img_tensors):
    return img_tensors * stats[1][0] + stats[0][0]

def show_images(images, nmax=64):
    fig, ax = plt.subplots(figsize=(8, 8))
    ax.set_xticks([]); ax.set_yticks([])
    ax.imshow(make_grid(denorm(images.detach()[:nmax]),
nrow=8).permute(1, 2, 0))

def show_batch(dl, nmax=64):
    for images, _ in dl:
        show_images(images, nmax)
        break
```

Let's display images of a batch

```
show_batch(train_loader)
```

```
Clipping input data to the valid range for imshow with RGB data
([0..1] for floats or [0..255] for integers).
```



Our first model: model_0

Let's define our own model. Here very simple model is defined. Through the assignment, you may design your own model.

```
# model_0
class ConvolutionalNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 3, 1)
        self.conv2 = nn.Conv2d(6, 12, 3, 1)
        self.conv3 = nn.Conv2d(12, 24, 3, 1)
        self.fc1 = nn.Linear(28*28*24, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, num_classes)
```

```
    def forward(self, X):
        X = F.relu(self.conv1(X)) # conv1
        X = F.max_pool2d(X, 2, 2)
        X = F.relu(self.conv2(X)) # conv2
        X = F.max_pool2d(X, 2, 2)
        X = F.relu(self.conv3(X)) # conv3
        X = F.max_pool2d(X, 2, 2)
        X = X.view(-1, 28*28*24)
        X = F.relu(self.fc1(X))
        X = F.relu(self.fc2(X))
        X = self.fc3(X)
        return F.log_softmax(X, dim=1)
```

```
#torch.manual_seed(101)
model = ConvolutionalNetwork()
model.to(device)
```

```
ConvolutionalNetwork(
  (conv1): Conv2d(3, 6, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(6, 12, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(12, 24, kernel_size=(3, 3), stride=(1, 1))
  (fc1): Linear(in_features=18816, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=150, bias=True)
)
```

Training

let's define loss

```
criterion = nn.CrossEntropyLoss().to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

Let's train the model

```
def accuracy(outputs, labels):
    _, preds = torch.max(outputs, dim=1)
    return torch.tensor(torch.sum(preds == labels).item() /
len(preds))
```

```
def train(model, epochs, train_loader, val_loader=None):
  model.train()
  start_time = time.time()

  # Trackers
  train_losses = []
  val_losses = []
  train_correct = []
  val_correct = []

  for i in range(epochs):
      trn_corr = 0
      val_corr=0
      tst_corr = 0
      train_loss=0

      # Training batches
      for b, (X_train, y_train) in enumerate(train_loader):
          b += 1

          X_train = X_train.to(device)
          y_train = y_train.to(device)

          y_pred = model(X_train)
          loss = criterion(y_pred, y_train)
          train_loss += loss.item()
          # Tracking correct predictions
          _, preds = torch.max(y_pred, dim=1)
          #batch_corr=torch.sum(preds == y_train).item() /
len(preds)
          batch_corr = torch.sum(preds == y_train)
          trn_corr += batch_corr

          optimizer.zero_grad()
          loss.backward()
          optimizer.step()

          # Print epoch and loss results
          if b%100==0:
              print(f'EPOCH: {i}  LOSS: {loss.item():10.8f}')

      train_losses.append(train_loss)
      train_correct.append(trn_corr)

      # Testing batches
```

```
      loss=0
      if val_loader:
          with torch.no_grad():
              for b, (X_val, y_val) in enumerate(val_loader):

                  X_val = X_val.to(device)
                  y_val = y_val.to(device)

                  y_pred = model(X_val)

                  # Tracking correct predictions
                  predicted = torch.max(y_pred.data, 1)[1]
                  val_corr += (predicted == y_val).sum()
                  loss += criterion(y_pred, y_val)
          print(f'\nValidation Accuracy:
{val_corr.item()/len(val_loader.dataset)} minutes')
          #loss = criterion(y_pred, y_val)
          val_losses.append(loss)
          val_correct.append(val_corr)
          print(
              f"EPOCH: {epochs} - VAL LOSS:
{loss.item()/len(val_loader.dataset):.4f}, VAL_ACCURACY:
{val_corr.item() / len(val_loader.dataset):.4f}"
          )

  print(f'\nDuration: {((time.time() - start_time)/60):.3f}
minutes')
```

```
train(model, epochs, train_loader, val_loader)
torch.save(model.state_dict(), 'PokemonModel.pt')   # save the
trained model. you can load the trained model later when
necessary
```

```
EPOCH: 0  LOSS: 5.03799057
EPOCH: 0  LOSS: 4.96565723
EPOCH: 0  LOSS: 5.04630184
EPOCH: 0  LOSS: 4.87055588
EPOCH: 0  LOSS: 4.81363916
EPOCH: 0  LOSS: 4.79415178
EPOCH: 0  LOSS: 4.24870777
EPOCH: 0  LOSS: 5.27701855
EPOCH: 0  LOSS: 4.69948101
EPOCH: 0  LOSS: 4.55316305
EPOCH: 0  LOSS: 4.80512762
```

```
Validation Accuracy: 0.03414634146341464 minutes
EPOCH: 1 - VAL LOSS: 1.1684, VAL_ACCURACY: 0.0341

Duration: 0.878 minutes
```

testing

Lets test our model with test data

```
  def test(model, data_loader):
    model.eval()
    test_loss = 0.0
    correct = 0

    with torch.no_grad():
        for images, labels in data_loader:
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)
            predicted = torch.max(outputs, 1)[1]
            loss = criterion(outputs, labels)

            test_loss += loss.item()
            correct += (labels == predicted).sum()
    test_acc=(correct.item()/len(data_loader.dataset))*100
    print(
        f"TEST LOSS: {test_loss / len(data_loader.dataset):.4f},
accuracy: {correct.item() / len(data_loader.dataset):.4f}"
    )
```

Let's reload the saved model

```
model = ConvolutionalNetwork()
model.load_state_dict(torch.load("PokemonModel.pt"))
model.to(device)
```

```
ConvolutionalNetwork(
  (conv1): Conv2d(3, 6, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(6, 12, kernel_size=(3, 3), stride=(1, 1))
  (conv3): Conv2d(12, 24, kernel_size=(3, 3), stride=(1, 1))
  (fc1): Linear(in_features=18816, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=150, bias=True)
)
```

```
#model.load_state_dict(torch.load('PokemonModel.pt'))
test(model, test_loader)
```

```
TEST LOSS: 1.1570, accuracy: 0.0313
```

3. Assignment

- Assignments 1 to 9 should start with model_0 .
- Assgnments 10-12 start with Resnet 34
- **Good presentation**: Through the assignments, you should present the results effectively by using the best visualization methods for the given assignments, for example, either text printout, graph, charts, or whatever that's most appropriate for the results.
- You can modify model_0 for each assignment **as necessary**
- Put your **opinion** on the results

1. Several Techniques (2pts)

Apply Xavier weight initialization, L2 regularization, Dropout, Batch Normalization. Each should start with model_0

```
# your code for Xavier weight initialization
```

```
# your code for L2 regularization
```

```
# your code for Dropout
```

```
# your code for Batch Normalization
```

2. Optimization variants (1pts)

Apply SGD, Momentum+SGD, maybe others optimizers can be tried as well

```
# your code for SGD
```

```
# your code for Momentum+SGD
```

3. Learning rate annealing (1pts)

Apply StepLR, MultistepLR, Exponential LR

```
# your code for StepLR
```

```
# your code for MultistepLR,
```

```
# your code for Exponential LR
```

4. Dilated Conv (1pts)

we can use dilated 3x3 conv to enlarge the receptive field. So this time, let's try dialted 3x3 conv to see what difference it makes.

```
# your code here
```

5. 1x1 Conv (2pts)

As we learned in the class 1x1 conv can be utilized to reduce the computation. To see the computation reduction, lets first change one or more of the three 3x3 convs to be 7x7 (you may have to adjust other layers as well in order to the NN work). Then apply 1x1 technique to make it faster

```
# your code here
```

6. Instable NN (1pts)

Let's modify the model so that the model has softmax output inside the model and then apply log and NLLLoss to test whether it really has numerical instability

```
# your code here
```

7. variable sized input (2pts)

The current model requires the input to be of 240x240 or more precisely 238x238 (both work with current model). Let's make the model takes any size input by using adaptive pooling. You may need to remove one or more pooling layers to make the minimum input size (you need to calculate this considering the operations in your model) less smaller than 240(or 238) so that you don't need to resize many of the images to make it bigger to fit the minimum input size. Still there maybe some images smaller than the minimum size, in that case you may have to reisze the image to the minimum size.

```
# your code here
```

8. Upsampling by transposed conv (2pts)

the model so far will look like (a) below. In order to get familiar with upsampling. Let's just apply the transposed conv here so that the model looks like (b).

```
# your code here
```

9. Two branch network (3pts)

Let's make a two-branch model, which looks like below.

```
# your code here
```

10. Pretrained Model (3pts)

Let's use pretrained model instead of the custom model we defined. Use a pre-trained ResNet-34 to train the model. Things to consider here is for example, up to which layers of the Resnet you are going to keep and what layers to drop. Upto which layers to freeze. Also need to adjust final classifier layers.

Assignment 10~12 should be based on the ResNet-34 pretrained model

```
# your code here
```

11. Visualization by CAM (3pts)

Since this is Resnet, meaning that it has global average pooling. Let's make use of it and visualize using Class Activation Map (although we skipped this part in the class but you can easily understand it and there are lots of code that you can refer.)

```
# your code here
```

12 Visualization by GradCAM (3pts)

While CAM only allows us to visualize the layer connected to the global average pooling, GradCAM allows to visualize any layers. So try to visualize other layers as well by using GradCAM. You can do some search for the GradCAM code and use it.

```
# your code here
```

13. Bonus (3pts)

Do whatever in order to get, for example

- the highest accuracy
- Any ideas
- Whatever