

resnetcnn1

December 3, 2025

```
[59]: import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pickle
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader, random_split, Dataset
import os
```

```
[60]: np.random.seed(0)
```

```
[61]: if torch.backends.mps.is_available():
    device = torch.device("mps")
    use_mps = True
elif torch.cuda.is_available():
    device = torch.device("cpu")
    use_mps = False

print(device)
```

mps

```
[62]: from PIL import Image
import cv2
import numpy

class PLKDataset(Dataset):
    def __init__(self, file_path, transform=None, apply_clahe=False):
        with open(file_path, 'rb') as f:
            data = pickle.load(f)
        self.images = data['images']
        self.labels = data['labels'].reshape(-1)
        self.transform = transform
        self.apply_clahe = apply_clahe
```

```

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

def __getitem__(self, idx):
    image = self.images[idx]
    label = int(self.labels[idx])

    if self.apply_clahe:
        image = self.clahe_preprocess(image)

    image = Image.fromarray(image.astype('uint8'))

    if self.transform:
        image = self.transform(image)

    return image, label

@staticmethod
def clahe_preprocess(img):
    lab = cv2.cvtColor(img, cv2.COLOR_RGB2LAB)
    l, a, b = cv2.split(lab)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    l2 = clahe.apply(l)
    lab2 = cv2.merge((l2, a, b))
    return cv2.cvtColor(lab2, cv2.COLOR_LAB2RGB)

```

[63]: dataset = PLKDataset('ift-3395-6390-kaggle-2-competition-fall-2025/train_data.
pkl')

[64]: raw_images, raw_labels = dataset.images, dataset.labels

[65]: from skimage import exposure
import cv2

```

bad_images= []
for i, img in enumerate(raw_images):

    img_norm = img / 255.0

    if img_norm.std() < 0.08:
        bad_images.append(i)

    if img_norm.mean() > 0.3:
        bad_images.append(i)

```

```

r, g, b = img_norm[:, :, 0], img_norm[:, :, 1], img_norm[:, :, 2]
if r.mean() > g.mean() * 1.8 and r.mean() > b.mean() * 1.8:
    bad_images.append(i)

"""
for i, img in enumerate(raw_labels):
    if img == 4:
        bad_images.append(i)
"""

print(f"Found {len(bad_images)} bad images out of {len(raw_images)}")

mask = np.ones(len(raw_images), dtype=bool)
mask[bad_images] = False

n_show = min(40, len(bad_images))

if n_show == 0:
    print("aucune image")

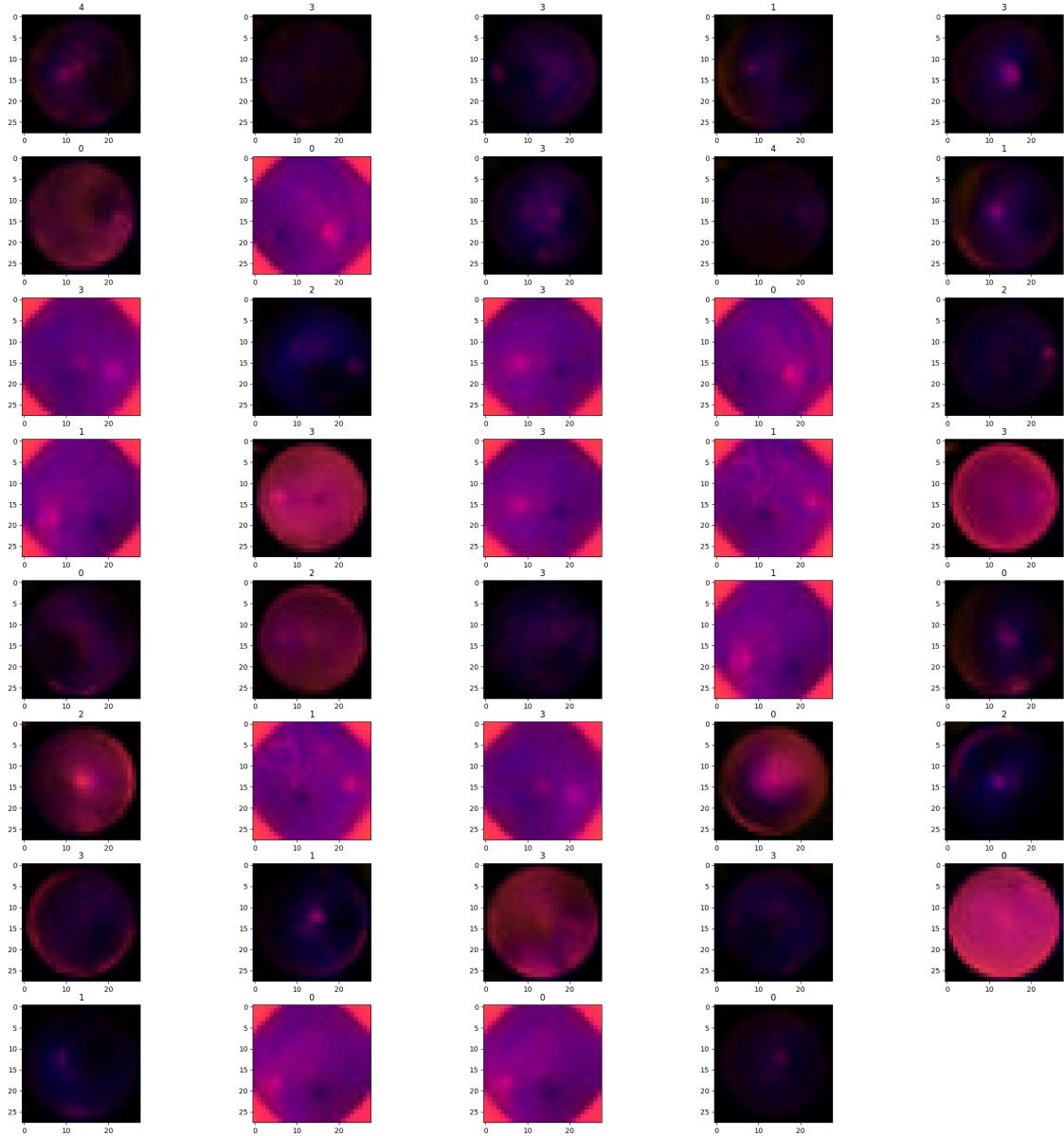
else:
    plt.figure(figsize=(28,28))
    for i, idx in enumerate(bad_images[:n_show]):
        plt.subplot(8, 5, i+1)
        plt.title(raw_labels[idx])
        plt.imshow(raw_images[idx])
    plt.show()

cleaned_images = raw_images[mask]
cleaned_labels = raw_labels[mask]

dataset.images = cleaned_images
dataset.labels = cleaned_labels

```

Found 39 bad images out of 1080



```
[66]: loader = DataLoader(dataset, batch_size=32, shuffle=True)
```

```
[67]: train_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(25),
    #transforms.RandomResizedCrop(224, scale=(0.9, 1.0)),
```

```

        transforms.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3, hue=0.
˓→05),

        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                            [0.229, 0.224, 0.225])
    )

    val_transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

```

```
[68]: from sklearn.model_selection import train_test_split

labels = dataset.labels
idx = np.arange(len(dataset))
train_idx, valid_idx = train_test_split(idx, test_size=0.2, stratify=labels,
˓→random_state=42)
```

```
[69]: class TransformSubset(Dataset):
    def __init__(self, subset, transform=None):
        self.subset = subset
        self.transform = transform

    def __getitem__(self, idx):
        image, label = self.subset[idx]
        if self.transform:
            image = self.transform(image)
        return image, label

    def __len__(self):
        return len(self.subset)
```

```
[70]: from torch.utils.data import Subset

train_dataset = Subset(dataset, train_idx)
train_data = TransformSubset(train_dataset, train_transform)

val_dataset = Subset(dataset, valid_idx)
val_data = TransformSubset(val_dataset, val_transform)

train_loader = DataLoader(train_data, batch_size=64, shuffle=True,
˓→pin_memory=True)
val_loader = DataLoader(val_data, batch_size=64, shuffle=False, pin_memory=True)
```

```
[71]: model = models.resnet50(weights="IMAGENET1K_V2")

[72]: num_classes = 5

model.fc = nn.Linear(model.fc.in_features, num_classes)

[73]: for param in model.parameters():
    param.requires_grad = False

for param in model.fc.parameters():
    param.requires_grad = True

for param in model.layer4.parameters():
    param.requires_grad = True

for param in model.layer3.parameters():
    param.requires_grad = True

"""

for param in model.layer2.parameters():
    param.requires_grad = True"""

model = model.to(device)

[74]: classes = np.unique(dataset.labels)
from sklearn.utils.class_weight import compute_class_weight
class_weights = compute_class_weight('balanced', classes=classes, y=dataset.
                                     labels)
weights_tensor = torch.tensor(class_weights, dtype=torch.float32).to(device)

[75]: criterion = nn.CrossEntropyLoss(weight=weights_tensor)
optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()), lr=1e-4)

[76]: for epoch in range(20):
    model.train()

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

    print(f"Epoch {epoch+1}, Loss: {loss.item():.4f}")
```

/Users/yamira.poldosilva/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.

```
13/site-packages/torch/utils/data/dataloader.py:692: UserWarning: 'pin_memory'  
argument is set as true but not supported on MPS now, device pinned memory won't  
be used.  
    warnings.warn(warn_msg)  
  
Epoch 1, Loss: 1.5889  
Epoch 2, Loss: 1.6083  
Epoch 3, Loss: 1.4879  
Epoch 4, Loss: 1.4818  
Epoch 5, Loss: 1.4806  
Epoch 6, Loss: 1.3771  
Epoch 7, Loss: 1.3400  
Epoch 8, Loss: 1.4485  
Epoch 9, Loss: 1.2533  
Epoch 10, Loss: 1.2490  
Epoch 11, Loss: 1.4477  
Epoch 12, Loss: 1.1992  
Epoch 13, Loss: 0.9473  
Epoch 14, Loss: 1.1280  
Epoch 15, Loss: 0.9361  
Epoch 16, Loss: 1.1178  
Epoch 17, Loss: 0.9869  
Epoch 18, Loss: 1.2299  
Epoch 19, Loss: 1.1430  
Epoch 20, Loss: 0.6438
```

```
[77]: model.eval()  
  
all_preds = []  
all_labels = []  
  
from sklearn.metrics import accuracy_score, recall_score,  
    balanced_accuracy_score, classification_report, confusion_matrix  
  
with torch.no_grad():  
    for images, labels in val_loader:  
        images = images.to(device)  
        labels = labels.to(device)  
  
        outputs = model(images)  
        _, predicted = torch.max(outputs, 1)  
  
        all_preds.extend(predicted.cpu().numpy())  
        all_labels.extend(labels.cpu().numpy())  
  
all_preds = np.array(all_preds)  
all_labels = np.array(all_labels)
```

```
[78]: bal_acc = balanced_accuracy_score(all_labels, all_preds)
recall = recall_score(all_labels, all_preds, average='macro')
acc = accuracy_score(all_labels, all_preds)

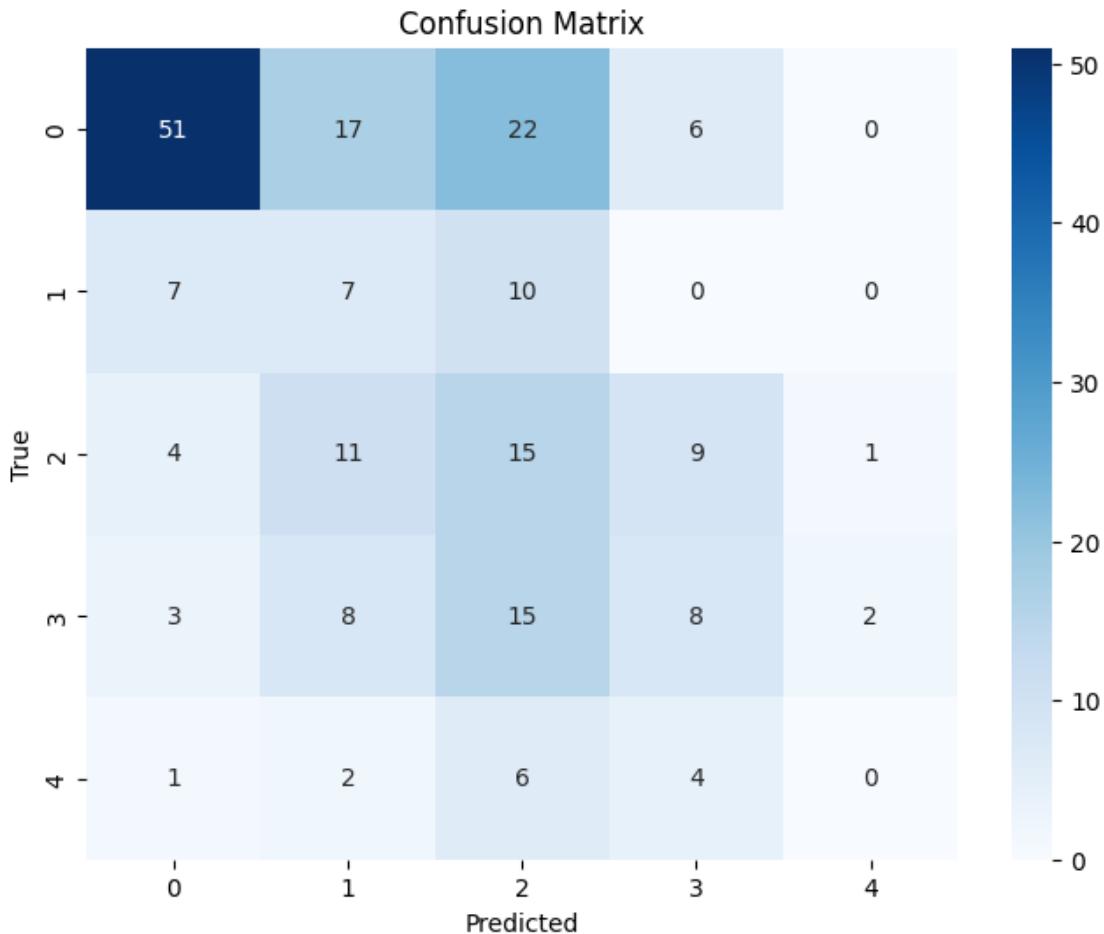
print(f"Validation Balanced Accuracy: {bal_acc:.4f}")
print(f"Validation Recall: {recall:.4f}")
print(f"Validation Accuracy: {acc:.4f}")
print(classification_report(all_labels, all_preds, digits=4))
```

Validation Balanced Accuracy: 0.2840
 Validation Recall: 0.2840
 Validation Accuracy: 0.3876

	precision	recall	f1-score	support
0	0.7727	0.5312	0.6296	96
1	0.1556	0.2917	0.2029	24
2	0.2206	0.3750	0.2778	40
3	0.2963	0.2222	0.2540	36
4	0.0000	0.0000	0.0000	13
accuracy			0.3876	209
macro avg	0.2890	0.2840	0.2729	209
weighted avg	0.4661	0.3876	0.4094	209

```
[79]: import seaborn as sns

cm = confusion_matrix(all_labels, all_preds)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



```
[80]: torch.save({
    "model_state_dict": model.state_dict(),
    "num_classes": num_classes,
}, "resnet50_finetuned.pth")
```

```
[81]: checkpoint = torch.load("resnet50_finetuned.pth", map_location=device, ↴
    weights_only=False)

model = models.resnet50(weights=None)
model.fc = nn.Linear(model.fc.in_features, checkpoint["num_classes"])

model.load_state_dict(checkpoint["model_state_dict"])
model.to(device)
model.eval()
```

```
[81]: ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
```

```

bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
    (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
    (layer1): Sequential(
        (0): Bottleneck(
            (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
            (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (relu): ReLU(inplace=True)
            (downsample): Sequential(
                (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
                (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            )
        )
        (1): Bottleneck(
            (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
            (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (relu): ReLU(inplace=True)
        )
        (2): Bottleneck(
            (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
            (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)

```

```

        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
    )
)
)
(layer2): Sequential(
    (0): Bottleneck(
        (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
            (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
            (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
    )
)
(1): Bottleneck(
    (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
)
)
(2): Bottleneck(
    (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,

```

```

track_running_stats=True)
    (relu): ReLU(inplace=True)
)
(3): Bottleneck(
    (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
)
)
(layer3): Sequential(
    (0): Bottleneck(
        (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
            (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
            (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
    )
    (1): Bottleneck(
        (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
)

```

```

        (relu): ReLU(inplace=True)
    )
(2): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
)
(3): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
)
(4): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
    (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu): ReLU(inplace=True)
)
(5): Bottleneck(
    (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
    (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)

```

```

        (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
    )
)
)
(layer4): Sequential(
    (0): Bottleneck(
        (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
            (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
            (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        )
    )
    (1): Bottleneck(
        (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
    )
    (2): Bottleneck(
        (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
        (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,

```

```

        track_running_stats=True)
        (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu): ReLU(inplace=True)
    )
)
)
(avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
(fc): Linear(in_features=2048, out_features=5, bias=True)
)

```

```
[82]: test_dataset = pickle.load(open('ift-3395-6390-kaggle-2-competition-fall-2025/
˓→test_data.pkl', 'rb'))
test_images = test_dataset['images']

test_transform = transforms.Compose([
    #transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

```
[83]: class TestPKLDataset(Dataset):
    def __init__(self, images, transform=None):
        self.images = images
        self.transform = transform

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

    def __getitem__(self, idx):
        image = self.images[idx]
        image = Image.fromarray(image.astype('uint8'))
        if self.transform:
            image = self.transform(image)
        return image
```

```
[84]: test_ds = TestPKLDataset(test_images, transform=test_transform)
test_loader = DataLoader(test_ds, batch_size=64, shuffle=False, pin_memory=True)
preds = []

with torch.no_grad():
    for images in test_loader:
        images = images.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        preds.extend(predicted.cpu().numpy())
```

```
df = pd.DataFrame({  
    "ID": np.arange(1, len(preds) + 1),  
    "Label": preds  
})  
  
df.to_csv("IFT3395_YAPS_MCSV51.csv", index=False)
```

```
/Users/yamira.poldosilva/Documents/UDEM/A25/IFT3395/kaggle2/kaggle2/lib/python3.  
13/site-packages/torch/utils/data/dataloader.py:692: UserWarning: 'pin_memory'  
argument is set as true but not supported on MPS now, device pinned memory won't  
be used.  
    warnings.warn(warn_msg)
```