resnetcnn_to_debug

December 3, 2025

[1]:

```
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
```

[2]:

```
np.random.seed(0)
```

[3]:

```
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
```

[4]:

```
from PIL import Image
import cv2
import numpy

class PLKDataset(Dataset):
    def __init__(self, file_path, transform=None, apply_clahe=True):
        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)
```

[5]:

```
dataset = PLKDataset('ift-3395-6390-kaggle-2-competition-fall-2025/train_data.
 ↪pkl')
```

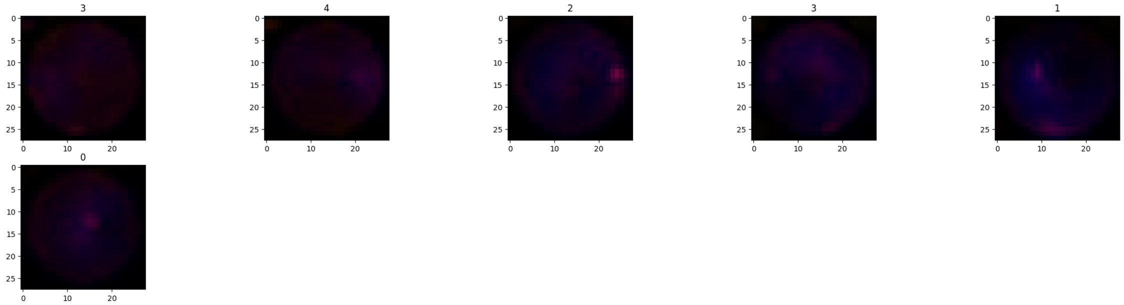
[6]:

```
raw_images, raw_labels = dataset.images, dataset.labels
```

[7]:

```
from skimage import exposure
import cv2

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

    img_norm = img / 255.0

    if img_norm.std() < 0.06:
        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 6 bad images out of 1080
```



[8]:

```
loader = DataLoader(dataset, batch_size=32, shuffle=True)
```

[9]:

```
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])
])
```

[10]:

```
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)
```

[11]:

```
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)
```

[12]:

```
from torch.utils.data import WeightedRandomSampler
from sklearn.utils.class_weight import compute_class_weight

train_labels = dataset.labels[train_idx]

classes = np.unique(train_labels)

class_weights = compute_class_weight('balanced', classes=classes,␣
 ↪y=train_labels)

sample_weights = class_weights[train_labels]

sampler = WeightedRandomSampler(
    weights=sample_weights,
    num_samples=len(sample_weights),
    replacement=True
)
```

[13]:

```
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,sampler=sampler,␣
 ↪pin_memory=True)
val_loader = DataLoader(val_data, batch_size=64, shuffle=False, pin_memory=True)
```

[14]:

```
from torchvision.models import efficientnet_b0, EfficientNet_B0_Weights

model = efficientnet_b0(weights=EfficientNet_B0_Weights.IMAGENET1K_V1)
```

[15]:

```
num_classes = 5
```

```
model.classifier[1] = nn.Linear(model.classifier[1].in_features, num_classes)
```

[16]:

```
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


"""

# Débloquer les derniers blocs (plus similaires à ResNet layer3/4)
for param in model.features[-1].parameters():
    param.requires_grad = True

for param in model.features[5].parameters():
    param.requires_grad = True

for param in model.features[6].parameters():
    param.requires_grad = True

# 3. Débloquer le classifier (obligatoire pour apprendre)
for param in model.classifier.parameters():
    param.requires_grad = True

model = model.to(device)
```

[17]:

```
weights_tensor = torch.tensor(class_weights, dtype=torch.float32).to(device)
```

[18]:

```
criterion = nn.CrossEntropyLoss(label_smoothing=0.1)
optimizer = optim.Adam(filter(lambda p: p.requires_grad, model.parameters()),␣
 ↪lr=1e-4)
```

[19]:

```
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: 1.5863
Epoch 2, Loss: 1.5955
Epoch 3, Loss: 1.5369
Epoch 4, Loss: 1.5433
Epoch 5, Loss: 1.4824
Epoch 6, Loss: 1.5436
Epoch 7, Loss: 1.4616
Epoch 8, Loss: 1.5802
Epoch 9, Loss: 1.5189
Epoch 10, Loss: 1.3693
Epoch 11, Loss: 1.5216
Epoch 12, Loss: 1.3240
Epoch 13, Loss: 1.3609
Epoch 14, Loss: 1.3715
Epoch 15, Loss: 1.2643
Epoch 16, Loss: 1.3064
Epoch 17, Loss: 1.4265
Epoch 18, Loss: 1.3502
Epoch 19, Loss: 1.3322
Epoch 20, Loss: 1.1292
```

[20]:

```
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)
```

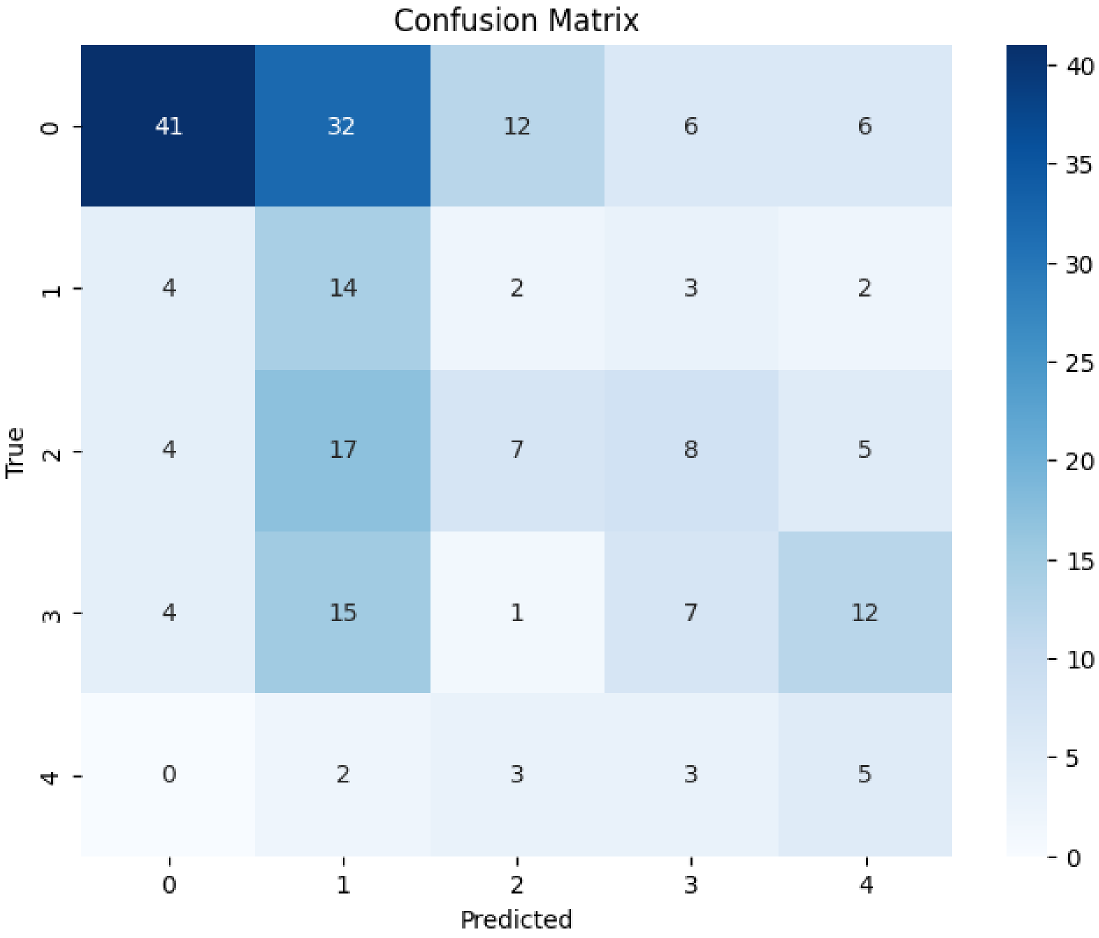
[21]:

```
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: 0.3435
Validation Recall: 0.3435
Validation Accuracy: 0.3442
              precision    recall  f1-score   support

           0     0.7736    0.4227    0.5467        97
           1     0.1750    0.5600    0.2667        25
           2     0.2800    0.1707    0.2121        41
           3     0.2593    0.1795    0.2121        39
           4     0.1667    0.3846    0.2326        13

    accuracy                         0.3442       215
   macro avg     0.3309    0.3435    0.2940       215
weighted avg     0.4799    0.3442    0.3706       215
```

[22]:

```
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()
```



	0	1	2	3	4
0	41	32	12	6	6
1	4	14	2	3	2
2	4	17	7	8	5
3	4	15	1	7	12
4	0	2	3	3	5

[23]:

```
torch.save({
    "model_state_dict": model.state_dict(),
    "num_classes": num_classes,
}, "efficientnet_finetuned.pth")
```

[24]:

```
checkpoint = torch.load("efficientnet_finetuned.pth", map_location=device, weights_only=False)

model = efficientnet_b0(weights=None)

model.classifier[1] = nn.Linear(model.classifier[1].in_features, checkpoint["num_classes"])

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

```
[24]: EfficientNet(
        (features): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
      bias=False)
            (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
      track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Sequential(
            (0): MBConv(
              (block): Sequential(
                (0): Conv2dNormActivation(
                  (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
      1), groups=32, bias=False)
                  (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
      track_running_stats=True)
                  (2): SiLU(inplace=True)
                )
                (1): SqueezeExcitation(
                  (avgpool): AdaptiveAvgPool2d(output_size=1)
                  (fc1): Conv2d(32, 8, kernel_size=(1, 1), stride=(1, 1))
                  (fc2): Conv2d(8, 32, kernel_size=(1, 1), stride=(1, 1))
                  (activation): SiLU(inplace=True)
                  (scale_activation): Sigmoid()
                )
                (2): Conv2dNormActivation(
                  (0): Conv2d(32, 16, kernel_size=(1, 1), stride=(1, 1), bias=False)
                  (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
      track_running_stats=True)
                )
              )
              (stochastic_depth): StochasticDepth(p=0.0, mode=row)
            )
          )
          (2): Sequential(
            (0): MBConv(
              (block): Sequential(
                (0): Conv2dNormActivation(
                  (0): Conv2d(16, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
                  (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
      track_running_stats=True)
                  (2): SiLU(inplace=True)
                )
                (1): Conv2dNormActivation(
                  (0): Conv2d(96, 96, kernel_size=(3, 3), stride=(2, 2), padding=(1,
      1), groups=96, bias=False)
```

```
            (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(96, 4, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(4, 96, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(96, 24, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.0125, mode=row)
      )
      (1): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(24, 144, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(144, 144, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), groups=144, bias=False)
            (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(144, 6, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(6, 144, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(144, 24, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.025, mode=row)
      )
    )
    (3): Sequential(
      (0): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(24, 144, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(144, 144, kernel_size=(5, 5), stride=(2, 2), padding=(2,
2), groups=144, bias=False)
            (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(144, 6, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(6, 144, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(144, 40, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(40, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.037500000000000006, mode=row)
      )
      (1): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(40, 240, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(240, 240, kernel_size=(5, 5), stride=(1, 1), padding=(2,
2), groups=240, bias=False)
            (1): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(240, 10, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(10, 240, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(240, 40, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(40, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.05, mode=row)
      )
    )
    (4): Sequential(
      (0): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(40, 240, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(240, 240, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), groups=240, bias=False)
            (1): BatchNorm2d(240, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(240, 10, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(10, 240, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(240, 80, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(80, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.0625, mode=row)
      )
      (1): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(80, 480, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(480, 480, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), groups=480, bias=False)
            (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(480, 20, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(20, 480, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(480, 80, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(80, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.07500000000000001, mode=row)
      )
      (2): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(80, 480, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(480, 480, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), groups=480, bias=False)
            (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(480, 20, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(20, 480, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(480, 80, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(80, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.08750000000000001, mode=row)
      )
    )
    (5): Sequential(
      (0): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(80, 480, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(480, 480, kernel_size=(5, 5), stride=(1, 1), padding=(2,
2), groups=480, bias=False)
            (1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(480, 20, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(20, 480, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(480, 112, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(112, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.1, mode=row)
      )
      (1): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(112, 672, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(672, 672, kernel_size=(5, 5), stride=(1, 1), padding=(2,
2), groups=672, bias=False)
            (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(672, 28, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(28, 672, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(672, 112, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(112, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.1125, mode=row)
      )
      (2): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(112, 672, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(672, 672, kernel_size=(5, 5), stride=(1, 1), padding=(2,
2), groups=672, bias=False)
            (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(672, 28, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(28, 672, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(672, 112, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(112, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.125, mode=row)
      )
    )
    (6): Sequential(
      (0): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(112, 672, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(672, 672, kernel_size=(5, 5), stride=(2, 2), padding=(2,
2), groups=672, bias=False)
            (1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(672, 28, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(28, 672, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(672, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.1375, mode=row)
      )
      (1): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(192, 1152, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(1152, 1152, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=1152, bias=False)
            (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(1152, 48, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(48, 1152, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(1152, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.15000000000000002, mode=row)
      )
      (2): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(192, 1152, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(1152, 1152, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=1152, bias=False)
            (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(1152, 48, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(48, 1152, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(1152, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.1625, mode=row)
      )
      (3): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(192, 1152, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(1152, 1152, kernel_size=(5, 5), stride=(1, 1),
padding=(2, 2), groups=1152, bias=False)
            (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(1152, 48, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(48, 1152, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(1152, 192, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.17500000000000002, mode=row)
      )
```

```
    )
    (7): Sequential(
      (0): MBConv(
        (block): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(192, 1152, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(1152, 1152, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), groups=1152, bias=False)
            (1): BatchNorm2d(1152, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
            (2): SiLU(inplace=True)
          )
          (2): SqueezeExcitation(
            (avgpool): AdaptiveAvgPool2d(output_size=1)
            (fc1): Conv2d(1152, 48, kernel_size=(1, 1), stride=(1, 1))
            (fc2): Conv2d(48, 1152, kernel_size=(1, 1), stride=(1, 1))
            (activation): SiLU(inplace=True)
            (scale_activation): Sigmoid()
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(1152, 320, kernel_size=(1, 1), stride=(1, 1),
bias=False)
            (1): BatchNorm2d(320, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
          )
        )
        (stochastic_depth): StochasticDepth(p=0.1875, mode=row)
      )
    )
    (8): Conv2dNormActivation(
      (0): Conv2d(320, 1280, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (1): BatchNorm2d(1280, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): SiLU(inplace=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=1)
  (classifier): Sequential(
    (0): Dropout(p=0.2, inplace=True)
    (1): Linear(in_features=1280, out_features=5, bias=True)
  )
```

```
)
```

[25]:

```
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])
])
```

[26]:

```
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
```

[27]:

```
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)
```