การ Finetune Model ResNet เพื่อจำแนกสภาพอากาศจากภาพ

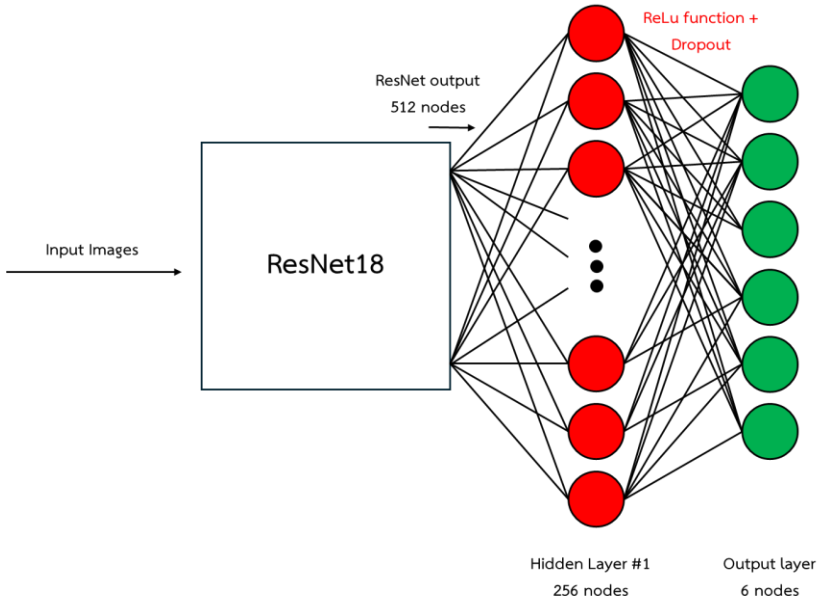
Repository Link : https://github.com/ValorXIV/ResNet_WeatherImageClassification

- หัวข้อนี้น่าสนใจอย่างไร ทำไมถึงเลือกหัวข้อนี้มาทำเป็น final project
  จากสภาวะแวดล้อมในปัจจุบัน ที่โลกของเราร้อนขึ้นเรื่อย ๆ ทำให้สภาพอากาศแปรปรวน
  และมีการเปลี่ยนแปลงอยู่บ่อยครั้ง ประกอบกับการที่ในปัจจุบัน social media เป็นที่นิยมในวงกว้าง
  และผู้คนต่างอัพโหลดข้อมูลรูปภาพจำนวนมหาศาลลงในอินเทอร์เน็ต ทำให้กลุ่มของเราต้องการจะสร้าง
  model ที่ช่วยจำแนกสภาพอากาศจากภาพถ่ายต่าง ๆ บนอินเทอร์เน็ตขึ้นมา
  เพื่อศึกษาและทำนายสภาพอากาศในแต่ละพื้นที่ และสามารถนำ model
  ไปพัฒนาต่อยอดเพื่อการช่วยรายงานสภาพอากาศแบบ real time
  จากภาพถ่ายในแต่ละบริเวณได้อีกด้วย

- ทำไมหัวข้อนี้จึงต้องใช้ deep learning ในการแก้ปัญหา เปรียบเทียบกับการแก้ปัญหานี้ด้วยวิธีอื่นๆ วิธี deep learning มีข้อเด่น ข้อด้อยอย่างไร
  เนื่องจากภาพสภาพอากาศในแต่ละวันมีความต่อเนื่อง เปลี่ยนแปลงบ่อยและมีจำนวนมหาศาลจากหลาย ๆ พื้นที่ ทำให้การจำแนกภาพด้วยวิธีอื่น ๆ เช่น การจำแนกด้วยฝีมือของมนุษย์
  นั้นทำได้ช้าและสิ้นเปลืองทรัพยากรเป็นอย่างมาก แต่หากเราใช้ model ที่ถูกสร้างขึ้นมาจาก deep learning ในการจำแนกรูปภาพ ก็จะทำให้สามารถจำแนกรูปภาพได้อย่างรวดเร็วและต่อเนื่องตลอดเวลา
  นอกจากนี้ข้อมูลยังมีจำนวนมากพอที่จะทำให้โมเดลสามารถเรียนรู้ลักษณะของภาพแต่ละสภาพอากาศได้อย่างถูกต้อง

- อธิบายสถาปัตยกรรม deep learning ที่ใช้ (feedforward NN CNN RNN GAN หรือ VAE)
  วาดรูปแสดงจำนวนโหนด weight bias รวมถึงการเชื่อมต่อ และ activation function ต่างๆให้ชัดเจน

นำ Model ResNet18 ซึ่งเป็น Model ประเภท CNN ที่ถูก pre-train เอาไว้ มาเชื่อมต่อกับ Feedforward NN ที่ใช้เพื่อ Finetune อีกหนึ่งชั้น ทำให้ Model ResNet สามารถทำงานกับ dataset ใหม่ที่ยังไม่เคยเห็นได้ดียิ่งขึ้น โดย ResNet จะมี output dimension อยู่ที่ 512 จะนำมาเชื่อมต่อกับ fully connected layer อีก 2 ชั้น โดยชั้นแรกจะมีจำนวน 256 node และใช้ ReLu เป็น activation function พร้อมกับทำการ dropout ด้วยอัตรา 0.5 หรือครึ่งหนึ่ง เพื่อไม่ให้ model เกิดการ overfitting ก่อนส่งต่อไปให้ชั้นที่สองซึ่งเป็น output layer ที่มีทั้งหมด 6 node ซึ่งแทน 6 output class



- อธิบายโค้ด PyTorch หรือ TensorFlow รวมไปถึงโค้ดส่วนอื่นๆที่ใช้ในการเชื่อมต่อกับโมเดลอื่นๆ อย่างชัดเจน

Prepare dataset

```python
import os
import torch
from torch.utils.data import DataLoader, random_split
from torchvision import datasets, transforms

# --- 1. Define Image Transforms ---
# Pre-trained models expect 224x224 images
IMG_SIZE = 224
IMAGENET_MEAN = [0.485, 0.456, 0.406]
IMAGENET_STD = [0.229, 0.224, 0.225]

# Transforms for the TRAINING set (with data augmentation)
train_transforms = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.1),
    transforms.ToTensor(),
    transforms.Normalize(IMAGENET_MEAN, IMAGENET_STD)
])

# Transforms for the VALIDATION set (NO augmentation)
val_transforms = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize(IMAGENET_MEAN, IMAGENET_STD)
])

# --- 2. Load Dataset and Print Counts ---
data_dir = './multiclass-weather-dataset'
full_data_path = os.path.join(data_dir, 'dataset')

# Load the entire dataset using ImageFolder
try:
    full_dataset = datasets.ImageFolder(full_data_path, transform=train_transforms)

    class_names = full_dataset.classes
    NUM_CLASSES = len(class_names)
    print(f"--- Dataset Found! ---")
    print(f"Found {NUM_CLASSES} classes: {class_names}")
    print(f"Total images found: {len(full_dataset)}")

```

- โค้ดส่วนการเตรียมข้อมูลกำหนดขนาดรูปภาพเป็น 224x224 พิกเซล ทำ Data Augmentation เพื่อเพิ่มความหลากหลายของข้อมูล และแบ่งชุดข้อมูลเป็น 2 ชุดคือ Training set (80%) กับ Validation set (20%)

```python
import torch.nn as nn
from torchvision import models

# --- 1. Load Pre-trained Base Model ---
# We load ResNet18, which is pre-trained on ImageNet
model = models.resnet18(weights=models.ResNet18_Weights.IMAGENET1K_V1)

# --- 2. Freeze the Base Model Layers ---
# We tell PyTorch *not* to update the weights of these layers
for param in model.parameters():
    param.requires_grad = False

# --- 3. Create Your "Own Model" (The Head) ---
# Get the number of input features from the ResNet's final layer (it's 512)
num_features = model.fc.in_features

# Replace the original 'fc' layer with your new classifier
# This is your "original model" part
model.fc = nn.Sequential(
    nn.Linear(num_features, 256),   # Input (512) -> Hidden Layer (256)
    nn.ReLU(),                      # Activation function
    nn.Dropout(0.5),                # Dropout to prevent overfitting
    nn.Linear(256, NUM_CLASSES)     # Hidden Layer (256) -> Output (4 classes)
)

# --- 4. Move Model to Device (GPU) ---
model = model.to(device)

# --- 5. Verify ---
# Print which parameters will be trained.
# It should ONLY show 'fc.0.weight', 'fc.0.bias', 'fc.3.weight', 'fc.3.bias'
print("--- Model Created ---")
print("Parameters to be trained:")
for name, param in model.named_parameters():
    if param.requires_grad:
        print(f"  {name}")
```

- โค้ดส่วนการสร้างโมเดล เริ่มจากการดาวน์โหลด model Resnet18 มา

จากนั้นทำการ freeze parameter เดิมเพื่อไม่ให้เกิดการอัพเดต parameter ระหว่างการ train จากนั้นนำ feature จาก layer สุดท้ายจาก Resnet18 มาต่อกับโมเดล FNN ที่สร้างขึ้น มี 2 layer คือ Hidden layer มี 256 nodes, drop out เพื่อป้องกันการ overfitting และ output layer ที่มี 6 nodes ตามประเภทที่เราต้องการจำแนก

```python
import torch.optim as optim

# 1. Define Loss Function
criterion = nn.CrossEntropyLoss()

# 2. Define Optimizer
optimizer = optim.Adam(model.fc.parameters(), lr=0.0002)

# 3. (Optional) Learning Rate Scheduler
from torch.optim.lr_scheduler import StepLR
scheduler = StepLR(optimizer, step_size=7, gamma=0.1)

print("--- Loss Function and Optimizer Defined ---")
print(f"Optimizer will train {sum(p.numel() for p in model.fc.parameters())} parameters.")
```

- โค้ดต่อมาเป็นการกำหนดตัว Loss function ที่ใช้แบบ CrossEntropyLoss และกำหนด Optimizer

Start Training loop

```python
import time

# --- 1. Create lists to store history ---
train_loss_history = []
train_acc_history = []
val_loss_history = []
val_acc_history = []

NUM_EPOCHS = 7

print(f"--- Starting Training for {NUM_EPOCHS} Epochs ---")

for epoch in range(NUM_EPOCHS):
    epoch_start_time = time.time()

    # --- Training Phase ---
    model.train()
    running_loss = 0.0
    running_corrects = 0

    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        _, preds = torch.max(outputs, 1)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * inputs.size(0)
        running_corrects += torch.sum(preds == labels.data)

    scheduler.step()

    epoch_loss = running_loss / len(train_dataset)
    epoch_acc = running_corrects.double() / len(train_dataset)

    epoch_loss = running_loss / len(train_dataset)
    epoch_acc = running_corrects.double() / len(train_dataset)

    # --- 2. Save training history ---
    train_loss_history.append(epoch_loss)
    train_acc_history.append(epoch_acc.cpu().item())

    # --- Validation Phase ---
    model.eval()
    val_loss = 0.0
    val_corrects = 0

    with torch.no_grad():
        for inputs, labels in val_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            _, preds = torch.max(outputs, 1)
            val_loss += loss.item() * inputs.size(0)
            val_corrects += torch.sum(preds == labels.data)

    epoch_val_loss = val_loss / len(val_dataset)
    epoch_val_acc = val_corrects.double() / len(val_dataset)

    # --- 3. Save validation history ---
    val_loss_history.append(epoch_val_loss)
    val_acc_history.append(epoch_val_acc.cpu().item())

    # Print results for this epoch
    epoch_time = time.time() - epoch_start_time
    print(f"Epoch {epoch+1}/{NUM_EPOCHS} | Train Loss: {epoch_loss:.4f} Acc: {ep

print("--- Finished Training ---")
```

- โค้ดต่อมาเป็นส่วนของการทำ Training ที่ให้โมเดลคำนวณค่า Loss คำนวณค่า gradient และทำการ update weight ของตัวโมเดลเองและทำซ้ำเรื่อยๆตามจำนวน epoch ที่เรากำหนดไว้คือ 7 epoch

Show loss function graph

```python
import matplotlib.pyplot as plt

# --- Create a 1x2 grid of plots ---
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

# --- Plot 1: Loss ---
ax1.plot(train_loss_history, label='Train Loss')
ax1.plot(val_loss_history, label='Validation Loss')
ax1.set_title('Training & Validation Loss')
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Loss')
ax1.legend()

# --- Plot 2: Accuracy ---
ax2.plot(train_acc_history, label='Train Accuracy')
ax2.plot(val_acc_history, label='Validation Accuracy')
ax2.set_title('Training & Validation Accuracy')
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Accuracy')
ax2.legend()

# --- Show the plots ---
plt.show()
```

```python
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import pandas as pd

print("--- Evaluating Model on Validation Set ---")
all_preds = []
all_labels = []

model.eval()
with torch.no_grad():
    for inputs, labels in val_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        outputs = model(inputs)
        _, preds = torch.max(outputs, 1)

        # Append batch predictions and labels to the lists
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# --- 1. Print Classification Report ---
print("\n--- Classification Report ---")
report = classification_report(all_labels, all_preds, target_names=class_names)
print(report)


# --- 2. Plot Confusion Matrix ---
print("\n--- Generating Confusion Matrix ---")
cm = confusion_matrix(all_labels, all_preds)

# Plot the confusion matrix using seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

- โค้ดส่วนของการสรุปผลทั้ง Training&Validation loss และ Accuracy ของโมเดล เพื่อนำมาใช้ในการพิจารณาปรับค่า parameter ต่างๆของโมเดลและสรุปผล

- อธิบายวิธีในการ train ตัว deep learning network ที่เลือกมาใช้ รวมไปถึงอธิบาย dataset ที่เกี่ยวข้องและแหล่งที่มา

Dataset: https://www.kaggle.com/datasets/vijaygiitk/multiclass-weather-dataset

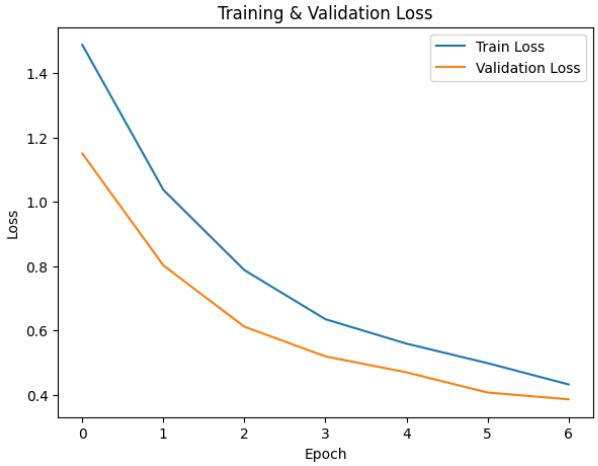
Dataset ที่นำมาใช้เกิดจากการรวบรวมภาพของสภาพอากาศต่าง ๆ ที่อยู่ภายใต้สัญญา Creative Commons จากแหล่งต่าง ๆ ในอินเทอร์เน็ต ทำให้สามารถนำมาใช้ได้อย่างอิสระ โดยประกอบไปด้วยภาพถ่ายของสภาพอากาศจริง ๆ จากทั่วทุกมุมโลก โดยแต่ละภาพมีขนาดต่างกันทั้งหมด และแต่ละภาพจะแทน 1 ใน 5 สภาพอากาศเหล่านี้ ได้แก่ มีเมฆมาก (cloudy) มีหมอกหมาก (foggy) มีฝน (rainy) มีแสงแดด (shine) และพระอาทิตย์ขึ้น/ตก (sunrise) โดยการใช้ dataset

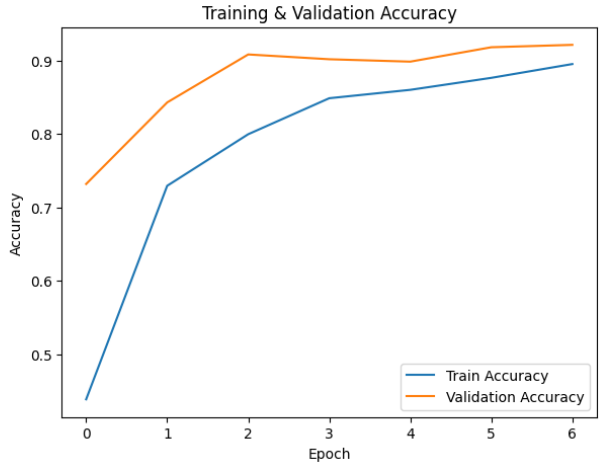
นี้มีจุดประสงค์เพื่อที่จะสร้าง model ที่ช่วยจำแนกภาพสภาพอากาศได้ว่าภาพแบบนี้มีสภาพอากาศแบบใด

วิธีการ train model

- ปรับขนาดข้อมูลนำเข้าให้เป็น 224x224 pixels ทำ Data Augmentation เพื่อเพิ่มความหลากหลายของข้อมูล และทำการแบ่ง Training set และ Validation set
- สร้างโมเดล ด้วยการ Fine-tuning model Resnet18 และเพิ่มโมเดลของตัวเองต่อท้ายเข้าไปเพื่อปรับให้โมเดลทำงานได้ดีกับ dataset ชุดนี้
- Train model ด้วยการกำหนด Learning rate, Loss function, Optimizer และทำ loop training โดยให้โมเดลคำนวณค่า Loss, Gradient เพื่อทำการปรับ weight ของตัวโมเดลเอง แล้ววัดผลด้วย validation set

- อธิบายการประเมิน (evaluate) model แสดงค่า loss จากการ train และ metric ที่เหมาะสมในการประเมิน

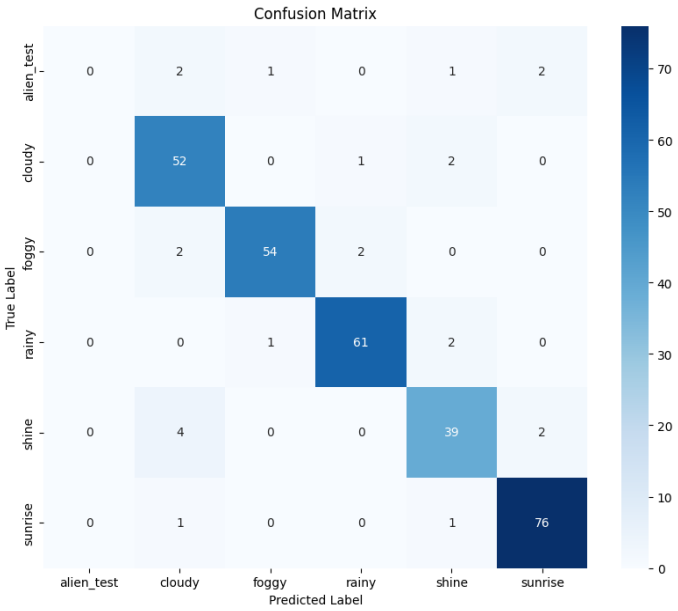




จากค่า validation loss และ accuracy ที่ได้จากกราฟ จะพบว่าค่า loss ลดลงจากประมาณ 1.15 เหลือเพียงประมาณ 0.4 และ accuracy เพิ่มจาก 0.73 ไปเป็น 0.92 ซึ่งแสดงให้เห็นว่า model สามารถทำงานได้ดีกับ unseen dataset และไม่เกิดการ overfitting

--- Classification Report ---

	precision	recall	f1-score	support
alien_test	0.00	0.00	0.00	6
cloudy	0.85	0.95	0.90	55
foggy	0.96	0.93	0.95	58
rainy	0.95	0.95	0.95	64
shine	0.87	0.87	0.87	45
sunrise	0.95	0.97	0.96	78
accuracy			0.92	306
macro avg	0.76	0.78	0.77	306
weighted avg	0.90	0.92	0.91	306



จาก classification report และ confusion matrix จะเห็นได้ว่า model มี accuracy อยู่ที่ 92% และมีค่า f1-score ของทุก class (ยกเว้น alien_test ซึ่งเป็นคลาสของข้อมูลเสีย) อยู่ระหว่าง 0.87-0.96 ซึ่งถือว่าเป็นค่าที่สูงและค่อนข้างแม่นยำมาก จึงสรุปได้ว่า การ finetune model ResNet ให้ทำนายสภาพอากาศจากภาพ เป็นไปได้ตามเป้าหมายที่ต้องการ

สมาชิกในกลุ่ม

นายชญานนท์ มานะกิจจานนท์ 6510503298

สัดส่วนงานที่รับผิดชอบ ค้นหาข้อมูลและ dataset ที่ใช้ ออกแบบโครงสร้างของ model และจัดทำรายงาน (50%)

นายศุภกิตต์ วงศ์โต 6510503816

สัดส่วนงานที่รับผิดชอบ สร้าง model ตามแผนที่วางไว้ และ train model ให้ได้ผลลัพธ์ตามเป้าหมาย และจัดทำรายงาน (50%)