Applications of Machine Learning in Remote Sensing
Homework 5

John Smith – `johnsmith@rit.edu`

`https://github.com/johnsmith/repo.git`

- In your submission, include **explanation**, **results**, and **the code** for the problem in the same PDF file in form of a Jupyter Notebook results. Also *separately*, attach solution's codes so I can replicate your results; **One PDF file for the entire homework + Jupyter Notebook File**.
- Show your understanding of the problem by providing **explanation**.
- Provide sufficient commenting in your code.
- Ensure all text/images are legible and organized.
- Ensure that your code can reproduce the submitted results.

Create a directory in your repository and name it `dl`, if you already do not have. The workflows and the scripts created in this homework would go under `dl`.

Problem 1 (30 points): CNN for Chlorophyll Estimation

This problem is provided as a lab exercise. Visit the class github repository for this problem (Here) and familiarize yourself with the files, especially the notebook $hw5\_p1.ipynb$. You will be working with the leaf spectra dataset originally generated in Homework 3. The data have been forward-modeled to the top of atmosphere, and system noise has been added to simulate realistic conditions.

The system is designed to replicate 425 spectral bands of the AVIRIS-NG, spanning the **0.4 to 2.5 micron wavelength range**. Your task is to perform chlorophyll content regression using deep convolutional neural networks (CNNs). Be sure to read through all parts of the notebook carefully and complete each section as instructed. Make sure to perform any necessary **EDA** on the data prior to modelling.

Problem 2 (70 Points): Transfer Learning

This assignment focuses on **remote sensing image classification** using **PyTorch**, leveraging the power of **pre-trained convolutional neural networks** for feature extraction and downstream classification.

You will be working with the **UCMerced dataset**, which contains 21 balanced classes and 100 RGB images per class in 8 bit format. Each image is 256 $\times$ 256 pixels, with 3 channels (Red, Green, Blue). The dataset is available to download from here.

Problem 2.a (30 points): Using a Pretrained Model as a Feature Extractor

In this part, you will use a **pre-trained ResNet model** (e.g., ResNet18, ResNet50, etc.) to extract features from the UCMerced dataset and perform classification using a traditional machine learning classifier (e.g., XGBoost, SVM, etc.). You will use Google Drive to store the dataset and Google Colab as your coding environment.

Step 1: Data Preparation

Upload the dataset to your Google Drive and mount the drive in Colab. Be sure to select a GPU runtime in Colab. Normalize all images using the **standard ImageNet statistics** to ensure compatibility with the pretrained model:

$$\text{mean} = [0.485, 0.456, 0.406], \quad \text{std} = [0.229, 0.224, 0.225]$$

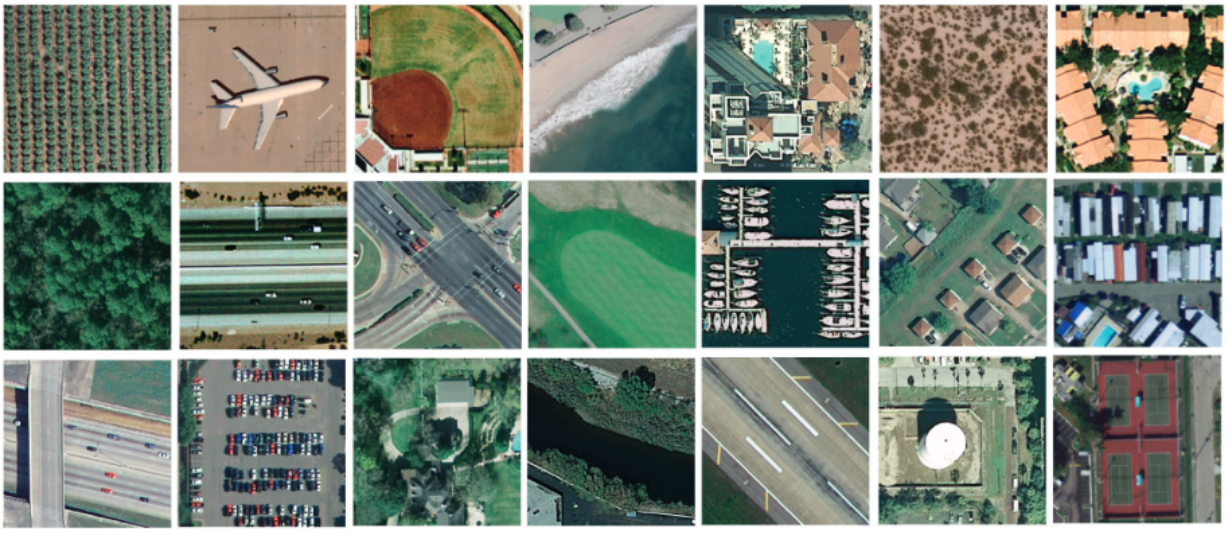

FIGURE 10: Picture gallery taken from all classes of UCM dataset.

To learn more about how to apply transforms in PyTorch, see the documentation on PyTorch Transforms.

There are **no predefined splits** in the dataset, so you must **manually create training, validation, and test sets**. Ensure that your splits are: **Deterministic** – use a fixed random seed or filename sorting, and **Class-balanced**. You will also need to create a mapping from `sample` to `class index`, where the class is a numeric value instead of a string (folder name).

Step 2: Using a Pretrained Model for Feature Extraction

Choose a ResNet architecture of your choice and load it with **ImageNet pretrained weights**. You can explore available options at PyTorch Models.

Next, **remove the final classification head** of the model (the fully connected layer that maps to 1000 ImageNet classes). This is because we are only interested in the feature representations learned by the model—not its predictions for ImageNet classes.

Once removed, the model will output a matrix of shape $B \times N$, where B is the batch size and N is the size of the feature vector. Make sure to:

- Send both your model and image batches to the GPU
- Bring the output features back to CPU before saving them

Repeat this process for both your training and test sets. You should store and reuse the features instead of recomputing them every time.

Tip: You can experiment with `prefetch_factor`, `num_workers`, *and* `pin_memory` *in your DataLoader to speed up feature extraction. See the PyTorch DataLoader documentation for more.*

Step 3: Classification Using Off-the-Shelf Models

Once you have extracted and saved your features, you can train any off-the-shelf classifier (e.g., `XGBoost`, `SVM`). Train your classifier on the training features and evaluate it on the test set.

You must report:

- Accuracy
- Precision, Recall, and F1-score
- Confusion matrix and normalized confusion matrix

Include a brief analysis of your classifier's performance.

Problem 2.b (20 points): Fine-tuning a Pretrained Model – Training Only the Head

In this part, you will continue to use a **pre-trained ResNet model**, but instead of extracting features, you will perform **transfer learning** by modifying and training the model directly. We adopt transfer learning because the deep model's large number of parameters cannot be supported by the small dataset provided, which can lead to overfitting (i.e., a high variance problem).

Step 1: Modify the Network

You will modify the final layer of the ResNet model to suit the 21-class classification task:

```
# Grab the number of input features to the classification head
num_ftrs = model.fc.in_features

# Replace the original head with a new one suited to your dataset
model.fc = nn.Linear(num_ftrs, num_classes)
```

Step 2: Freeze the Base Model

Next, freeze all layers in the base model to retain pre-trained ImageNet features, and only allow the new head to train:

```
# Freeze all layers
for param in model.parameters():
    param.requires_grad = False

# Unfreeze only the classification head
for param in model.fc.parameters():
    param.requires_grad = True
```

Step 3: Training Setup

Use the **ImageNet normalization statistics** for your dataset during both training and testing:

$$\text{mean} = [0.485, 0.456, 0.406], \quad \text{std} = [0.229, 0.224, 0.225]$$

Set up your training using the following guidelines:

- Use a **learning rate of** 10^{-3}.
- Use the `Adam` optimizer.
- Use a `CrossEntropyLoss` as your classification loss function.
- Optionally, use an `ExponentialLR` scheduler to decay the learning rate.
- Make sure your model and batches are sent to `cuda()` if available.

Train your model until convergence. Save the model parameters when the **validation loss is at its minimum**. Keep track of the training and validation **loss and accuracy per epoch**, and **plot both training and validation losses** across epochs.

Step 4: Evaluation

After training, evaluate your model's performance on the test set. Report the same metrics as in Problem 2.a:

- Accuracy
- Precision, Recall, F1-score
- Confusion matrix and normalized confusion matrix

Include a short discussion comparing this approach to the feature extraction method in the previous part.

Problem 2.c (20 points): Fine-tuning a Pretrained Model – Retraining the Entire Network

In this section, you will perform full **fine-tuning** of the pretrained model. Unlike Part 2.b, where only the classification head was trained, you will now **unfreeze the entire model** and retrain all of its parameters. This allows the model to adjust its feature representations to better suit the UCMerced dataset.

Step 1: Modify and Unfreeze the Network

Just like before, replace the final fully connected (FC) layer to match your number of output classes:

```
# Replace final classification head
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, num_classes)
```

This time, **do not freeze** the base model. Leave all parameters trainable:

```
# Make all layers trainable
for param in model.parameters():
    param.requires_grad = True
```

Step 2: Training Setup

As before, apply **ImageNet normalization statistics** to your dataset:

$$\text{mean} = [0.485, 0.456, 0.406], \quad \text{std} = [0.229, 0.224, 0.225]$$

Use the following setup for training:

- Use a **smaller learning rate of** 10^{-4}.
- Use the `Adam` optimizer and an appropriate loss function like `CrossEntropyLoss`.
- Optionally, use a learning rate scheduler (e.g., `ExponentialLR`).
- Train until convergence and monitor for signs of overfitting (e.g., validation loss increasing).
- Save model weights at the epoch with the lowest validation loss.
- Plot the **training and validation loss per epoch**.

Step 3: Evaluation and Comparison

After training, evaluate your model on the test set. Report:

- Accuracy
- Precision, Recall, and F1-score
- Confusion matrix and normalized confusion matrix

Finally, **compare all three approaches** from this assignment—(2.a) feature extraction + classifier, (2.b) head-only fine-tuning, and (2.c) full model fine-tuning. Explain on the performance differences across these methods in terms of the reported metrics. Explain your ideas on how you can improve the performance with respect to using different models and weights, generalization, etc.