Assignment 2 - Image Classification

2020310083 Hyungjun Shon

Dept. of System Management Engineering
Dept. of Software
Sungkyunkwan University

2024년 11월 27일

차 례

1 Introduction

The purpose of this project is to classify satellite images from the EuroSAT dataset into distinct land cover categories. The EuroSAT dataset is a publicly available collection of images representing different land cover types, including agricultural fields, forests, residential areas, and more. This project aims to classify the images into 10 different classes as accurately as possible.

2 Dataset Overview

The EuroSAT dataset contains 10 different classes of land cover images. It is widely used for benchmarking image classification models. (Helber et al. 2024)

- **Classes:** 10 (e.g., Residential, Industrial, Forest, etc.)
- **Image Size:** Variable, resized to 64x64 pixels for this implementation.
- **Splits:** Training set: 70%, Validation set: 20%, Test set: 10%

3 Implementation Details

3.1 Libraries Used

```
import matplotlib.pyplot as plt # For visualizations
import numpy as np # For numerical computations
import tensorflow as tf # For machine learning models
import tensorflow_datasets as tfds # For loading datasets
import datetime # For date and time manipulation
from tensorflow.keras import layers, Model, models # For building Keras models
from tensorflow.keras.applications import EfficientNetB0 # Pre-trained model for image classification
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

tf.random.set_seed(42) # For reproducibility
```

- **TensorFlow**: For deep learning model development
- **TensorFlow Datasets (TFDS)**: To load and split the dataset
- **NumPy**: For numerical operations
- **Matplotlib**: For data visualization

set the `random seed` to `42` for reproducibility

3.2 Dataset Preprocessing

```
# Set parameters
batch_size = 32  # Samples per update
epochs = 10  # Training epochs
img_height = 64  # Image height
img_width = 64  # Image width
num_classes = metadata.features["label"].num_classes  # Unique class count

# Function to resize and normalize images
def resize_and_rescale(image, label):
    image = tf.cast(image, tf.float32)  # Convert to float32
    image = tf.image.resize(image, [img_height, img_width])  # Resize image
    image = image / 255.0  # Normalize to [0, 1]
```

```python
    return image, label  # Return image and label

# Function to augment training images
def augment(image, label):
    image, label = resize_and_rescale(image, label)  # Resize and normalize
    image = tf.image.rot90(image)  # Rotate image 90 degrees
    image = tf.image.random_crop(image, size=[img_height, img_width, 3])  # Random crop
    return image, label  # Return augmented image and label

# Set auto-tuning for data pipeline
AUTOTUNE = tf.data.AUTOTUNE

# Shuffle, augment, batch, and prefetch the training dataset
train = (
    train.shuffle(1000)  # Shuffle the training data
    .map(augment, num_parallel_calls=AUTOTUNE)  # Apply augmentation
    .batch(batch_size)  # Batch the data
    .prefetch(AUTOTUNE)  # Prefetch for performance optimization
)

# Resize and batch the validation dataset
val = (
    val.map(resize_and_rescale, num_parallel_calls=AUTOTUNE)  # Resize and normalize
    .batch(batch_size)  # Batch the data
    .prefetch(AUTOTUNE)  # Prefetch for performance optimization
)

# Resize and batch the test dataset
test = (
    test.map(resize_and_rescale, num_parallel_calls=AUTOTUNE)  # Resize and normalize
    .batch(batch_size)  # Batch the data
    .prefetch(AUTOTUNE)  # Prefetch for performance optimization
)
```

3.2.1 Resizing and Normalization

- All images are resized to `64x64` pixels (same as the original eurosat dataset)
- Pixel values are normalized to the range `[0, 1]` (this is done to stabilize the training process of neural networks)

3.2.2 Data Augmentation (for Training Data)

For training data, the best accuracy was achieved by applying the following transformations, so same transformations are applied to training data. (Neumann et al. 2019)

- **Rotation**: Random rotations by 90 degrees.
- **Cropping**: Random cropping to introduce variability.

3.2.3 Shuffling and Batching

- Training data is shuffled with a buffer size of `1000` for better generalization.
- All datasets (train, validation, test) are batched into `32` samples per batch for efficient processing.

> The data preprocessing pipeline ensures the images are resized, normalized, and augmented to enhance generalization.

3.3 Model Architecture

```python
# Load the pre-trained EfficientNetB0 model without the top layer
base_model = EfficientNetB0(
    input_shape=(img_height, img_width, 3),
    include_top=False,
    weights="imagenet"
)

# Build the complete model with additional classification layers
model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dropout(0.5),
    layers.Dense(metadata.features['label'].num_classes, activation='softmax')
])

# Compile the model and show summary
model.summary()
```

3.3.1 EfficientNetB0

Below is the baseline network of EfficientNetB0 (input size is modified to `64x64` in this project). (Tan and Le 2019)

Stage	Operator	Resolution	#Channels	#Layers
1	Conv3x3	224 × 224	32	1
2	MBConv1, k3x3	112 × 112	16	1
3	MBConv6, k3x3	112 × 112	24	2
4	MBConv6, k5x5	56 × 56	40	2
5	MBConv6, k3x3	28 × 28	80	3
6	MBConv6, k5x5	14 × 14	112	3
7	MBConv6, k5x5	14 × 14	192	4
8	MBConv6, k3x3	7 × 7	320	1
9	Conv1x1 & Pooling & FC	7 × 7	1280	1

Pre-trained CNN model **EfficientNetB0, was selected** due to: (Keras 2024)

- 93.3% top-5 accuracy, demonstrating its capability to capture relevant features and make accurate predictions across multiple classes
- size of 29 MB, making it much smaller than VGG19 (549 MB), ResNet50 (98 MB), and even Xception (88 MB) → colab has restricted memory (reduced training time, faster inference, and lower memory usage needed)
- significantly higher accuracy compared to MobileNet

3.3.2 Base Model

- Input shape: `(64, 64, 3)`
- Pre-trained on ImageNet.
- The top (classification) layers were removed (`include_top=False`) for custom classification.

3.3.3 Added Layers for classification

- **Global Average Pooling layer**: Reduces spatial dimensions to a single vector per image.
  - preserving spatial information while reducing the number of parameters.
  - prevents overfitting and serves as a bridge between the convolutional base and the dense layers

- **Dropout layer**: Regularization with a dropout rate of `0.5`.
    - forcing the network to learn more robust features
- **Dense layer**: Outputs class probabilities using a `softmax` activation.
    - mapping the features extracted by the convolutional base to the class probabilities

3.3.4 Parameters

- Total params: 4,062,381 (15.50 MB)
- **Trainable params: 4,020,358 (15.34 MB)**
- Non-trainable params: 42,023 (164.16 KB)

3.4 Fine-Tuning

```python
# Make the base model trainable for fine-tuning
base_model.trainable = True

# Compile the model with Adam optimizer and set loss and metrics
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"],
)

log_dir = "logs" + datetime.datetime.now().strftime("/%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)


# Train the model with training and validation data
history_fine = model.fit(
    train,  # Training data
    validation_data=val,  # Validation data
    epochs=20,  # Total epochs for training
    callbacks=[tensorboard_callback],  # Save best weights during training
)

# Load the best weights after training is complete
model.load_weights("best_model_weights.h5")
```

The EfficientNetB0 model is not suitable for catching the spatial information of the images for eurosat with pre-trained weights. Therefore, **rather than transfer learning, fine-tuning for all layers is applied.**

- **Loss Function:** Sparse Categorical Crossentropy
    - Chosen for integer-based multi-class classification
- **Optimizer:** Adam
    - Learning rate: `1e-4`
- **Metrics:** Accuracy
    - Evaluates the percentage of correctly classified images.
- **Epochs:** `20`
    - Total epochs for training is `20` for small model size

4 Results

4.1 Training and Validation Loss and Accuracy

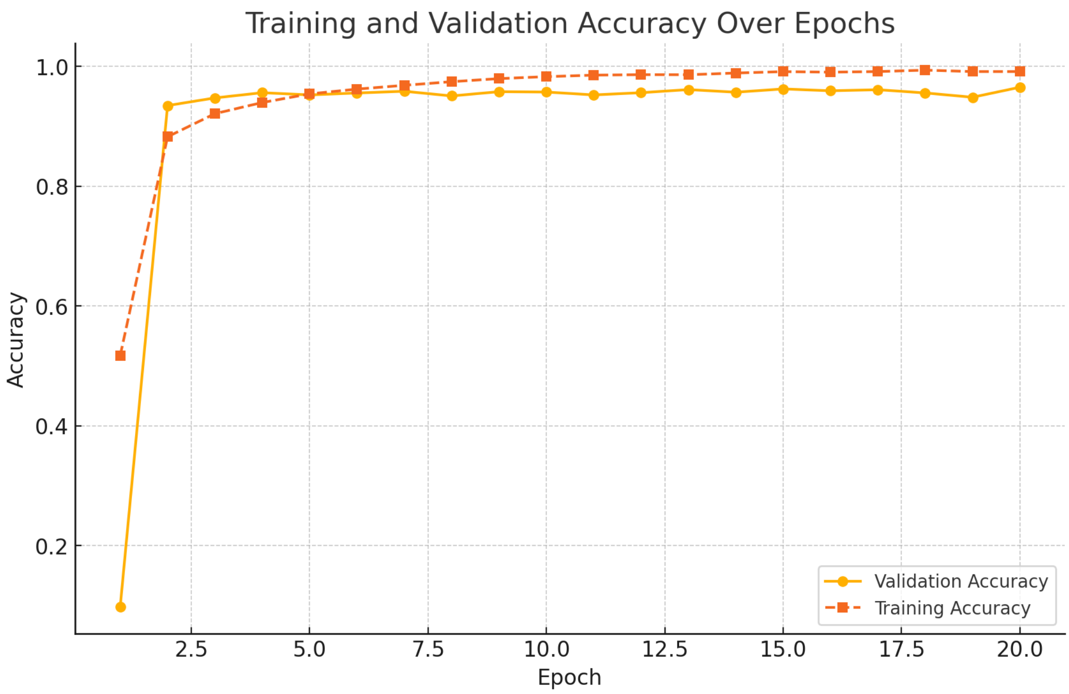

Figure 1: training and validation loss and accuracy

Epoch	Train Accuracy	Validation Accuracy	Train Loss	Validation Loss
1	0.5172	0.0978	1.4705	18.8148
2	0.8825	0.9346	0.3556	0.2003
3	0.9211	0.9474	0.2475	0.1542
4	0.9397	0.9561	0.1878	0.1345
5	0.9541	0.9524	0.1376	0.1401
6	0.9621	0.9556	0.1145	0.1292
7	0.9684	0.9585	0.1000	0.1226
8	0.9745	0.9507	0.0769	0.1507
9	0.9796	0.9578	0.0628	0.1278
10	0.9829	0.9572	0.0582	0.1417
11	0.9854	0.9524	0.0443	0.1501
12	0.9862	0.9561	0.0402	0.1440
13	0.9861	0.9613	0.0449	0.1410
14	0.9889	0.9569	0.0349	0.1426
15	0.9913	0.9624	0.0270	0.1298
16	0.9904	0.9593	0.0306	0.1400
17	0.9914	0.9611	0.0270	0.1358
18	0.9937	0.9557	0.0219	0.1648
19	0.9914	0.9485	0.0257	0.1899
20	0.9914	0.9652	0.0240	0.1380

During training, the last epoch has the highest validation accuracy with `96.52%` (but not for the validation loss). So compare with lowest validation loss weight, the best validation accuracy model is perform better when it is tested. Therefore, the best model check pointing is not used. (just used the last epoch weight)

4.2 Test Accuracy

Rank	Model	Test Accuracy (%)	Test Loss
1	IMP+MTP (IntenImage-XL)	99.24	-
2	µ2Net+ (ViT-L/16)	99.22	-
3	µ2Net (ViT-L/16)	99.20	-
4	ResNet50	99.20	-
5	WaveMix	98.96	-
6	MoCo-v2 (ResNet18, fine-tune)	98.90	-
7	DINO-MC (Wide ResNet)	98.78	-
8	MAE+MTP (ViT-L+RVSA)	98.78	-
9	MAE+MTP (ViT-B+RVSA)	98.76	-
10	MSMatch Multispectral	98.65	-
11	MSMatch RGB	98.14	-
12	SEER (RegNet10B - linear eval)	97.50	-
13	**EfficientNetB0 (Our Result)**	**96.81**	**0.1281**
14	DINO-MC (WRN linear eval)	95.70	-
15	MoCo-v2 (ResNet18, linear eval)	94.40	-

The test accuracy is `96.81%` with `0.1281` test loss. This can be competitive with the state-of-the-art models results with much smaller model size. So the result is quite good for balance between accuracy and computational cost. ("Ranking of the Best Models on EuroSAT | Papers with Code" 2024)

4.3 Confusion Matrix

```python
# Get predictions for the entire test dataset
y_pred = []
y_true = []

# Iterate through the test dataset to collect true labels and predictions
for images, labels in test:
    predictions = model.predict(images)  # Get predictions for each batch
    y_pred.extend(np.argmax(predictions, axis=1))  # Convert predictions to class indices
    y_true.extend(labels.numpy())  # Convert true labels to numpy array

# Convert predictions and true labels to numpy arrays
y_pred = np.array(y_pred)
y_true = np.array(y_true)

# Generate confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Display confusion matrix with rotated x-axis labels
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=metadata.features['label'].names)
fig, ax = plt.subplots(figsize=(10, 8))  # Adjust figure size for better readability
disp.plot(cmap=plt.cm.Blues, ax=ax)  # Plot confusion matrix with custom axes

# Rotate x-axis labels for better visibility
plt.xticks(rotation=45, ha="right")
plt.title('Confusion Matrix')
plt.show()
```

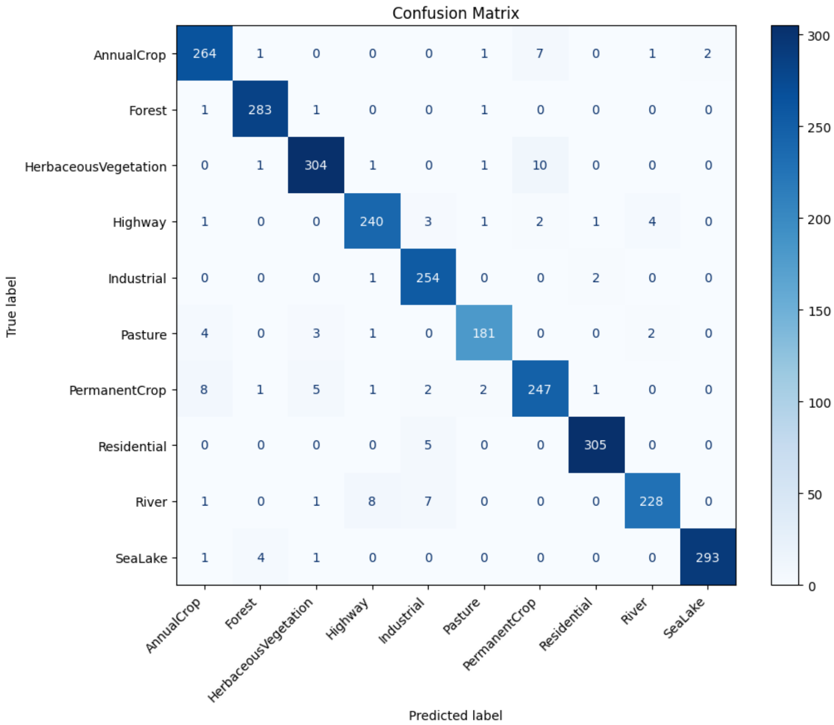

Figure 2: confusion matrix

The majority of the predictions lie on the diagonal, showing a high level of accuracy. This indicates that the model has learned to identify these classes very well. Classes with unique or easily distinguishable features (e.g., SeaLake, Forest) tend to have higher accuracy, but classes with visual overlap, such as AnnualCrop vs. PermanentCrop or River vs. Industrial, have more misclassifications

4.3.1 Misclassifications

Some misclassifications are observed in specific classes.

1. **AnnualCrop:**
   - Misclassified as **SeaLake** (2 samples) and **Herbaceous Vegetation** (1 sample).
   - Visual similarities in textures or patterns between these land types in satellite imagery may cause confusion.
2. **PermanentCrop:**
   - Misclassified as **Herbaceous Vegetation** (5 samples) and **AnnualCrop** (8 samples).
   - Permanent and Annual Crops can have overlapping features like vegetation patterns, making them challenging to distinguish.
3. **River:**
   - Misclassified as **Highway** (8 samples) and **Industrial** (7 samples).
   - Water bodies may appear near urban areas, leading to confusion with industrial or highway-related regions.
4. **Pasture:**
   - Misclassified as **Herbaceous Vegetation** (3 samples).
   - Similar grassland or vegetation textures between these two classes.

4.3.2 Possible Reasons for Errors

1. **Visual Similarities:**
   - Classes like crops (AnnualCrop, PermanentCrop) and vegetation (Pasture, Herbaceous Vegetation) may share textures and patterns, leading to confusion.

2. **Low Resolution:**
    - Image size of 64x64 pixels might have caused the loss of finer details critical for distinguishing certain classes.
3. **Model Limitations:**
    - While EfficientNetB0 is lightweight and efficient, its feature extraction capabilities might be slightly limited compared to larger architectures.
    - Also the base model input size is `224x224`, so the performance might be limited by the modified input size (model show best performance when input size is `224x224`)

The model performs exceptionally well, achieving high accuracy across most classes. While certain classes with overlapping features (e.g., crops and vegetation) present challenges, the results demonstrate that **EfficientNetB0 is highly effective** for the EuroSAT classification task. With minor refinements, the model could approach state-of-the-art performance on this dataset.

5 References

Helber, Patrick, Benjamin Bischke, Andreas Dengel, and Damian Borth. 2024. "EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification." https://github.com/phelber/EuroSAT.

Keras. 2024. "Keras Applications Documentation." https://keras.io/api/applications/.

Neumann, Maxim, Andre Susano Pinto, Xiaohua Zhai, and Neil Houlsby. 2019. "In-Domain Representation Learning for Remote Sensing."

"Ranking of the Best Models on EuroSAT | Papers with Code." 2024. https://paperswithcode.com/sota/image-classification-on-eurosat.

Tan, Mingxing, and Quoc V. Le. 2019. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks."