Session 4: CNN Hands-on Lab

Building and Training CNNs for EO Image Classification

Stylianos Kotsopoulos
EU-Philippines CoPhil Programme

Session Overview

Duration: 2.5 hours
Type: Intensive hands-on lab
Goal: Turn CNN theory into a working model

You will: - Prepare the EuroSAT dataset - Build a CNN from scratch in TensorFlow/Keras - Train with GPU acceleration in Colab - Evaluate with confusion matrix and F1 - Apply transfer learning (ResNet50)

Prerequisites: - Session 3 complete (CNN basics) - Colab account with GPU enabled - Python & NumPy fundamentals

Notebook: `session4_cnn_classification_STUDENT.ipynb`

Setup & Data Preparation

Colab GPU + Environment

Steps: 1. Runtime → Change runtime type → Hardware accelerator → GPU
2. Verify with `!nvidia-smi`
3. `pip install tensorflow` (if needed)
4. Set seeds for reproducibility

```python
import tensorflow as tf, numpy as np, random, os
SEED = 42
random.seed(SEED); np.random.seed(SEED); tf.random.set_seed(SEED)
print(tf.__version__, tf.config.list_physical_devices('GPU'))
```

EuroSAT Dataset (Sentinel-2, 10 classes)

- ~27k RGB chips (64×64) derived from S2
- Classes: AnnualCrop, Forest, Herbaceous, Highway, Industrial, Pasture, PermanentCrop, Residential, River, SeaLake

```python
# Example TFDS approach
import tensorflow_datasets as tfds
(ds_train, ds_val, ds_test), meta = tfds.load(
  'eurosat/rgb', split=['train[:70%]','train[70%:85%]','train[85%:]'],
  as_supervised=True, with_info=True)
```

Preprocessing: Normalize to [0,1], one-hot labels, split 70/15/15

Building a CNN from Scratch

Architecture (reference)

```
Input (64×64×3)
 → [Conv(32, 3×3) + ReLU] → MaxPool
 → [Conv(64, 3×3) + ReLU] → MaxPool
 → [Conv(128,3×3) + ReLU] → MaxPool
 → Flatten → Dropout(0.5)
 → Dense(128) + ReLU → Dropout(0.5)
 → Dense(10) + Softmax
```

```python
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Flatten

model = Sequential([
  Conv2D(32, (3,3), activation='relu', input_shape=(64,64,3)),
  MaxPooling2D(),
  Conv2D(64, (3,3), activation='relu'),
  MaxPooling2D(),
  Conv2D(128,(3,3), activation='relu'),
  MaxPooling2D(),
  Flatten(), Dropout(0.5),
  Dense(128, activation='relu'), Dropout(0.5),
  Dense(10, activation='softmax')
])
```

Compile & Train

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy','top_k_categorical_accuracy'])

cb = [
  tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True),
  tf.keras.callbacks.ModelCheckpoint('best_model.h5', save_best_only=True),
  tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3)
]

history = model.fit(ds_train.batch(64).prefetch(2),
                    validation_data=ds_val.batch(64).prefetch(2),
                    epochs=30, callbacks=cb)
```

Healthy curves: train↓, val↓ then plateau; small gap

Overfitting: large gap → add dropout/augmentation

Evaluation & Error Analysis

Accuracy + Confusion Matrix

```
import numpy as np, matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report

y_true, y_pred = [], []
for x, y in ds_test.batch(64):
  p = model.predict(x, verbose=0).argmax(axis=1)
  y_pred.extend(p); y_true.extend(y.numpy())

cm = confusion_matrix(y_true, y_pred)
print(classification_report(y_true, y_pred))
```

Look for: - Which pairs are confused? (e.g., AnnualCrop vs Herbaceous)
- Per-class precision/recall balance

Visualize Misclassifications

```
# Show a grid of wrong predictions for qualitative review
```

- Investigate systematic errors
- Adjust augmentation / architecture accordingly

Data Augmentation

EO-aware augmentations

```python
import tensorflow as tf
from tensorflow.keras import layers

data_aug = tf.keras.Sequential([
  layers.RandomRotation(0.25),
  layers.RandomFlip('horizontal_and_vertical'),
  layers.RandomZoom(0.1),
  layers.RandomContrast(0.1)
])
```

- Rotations/flips OK for overhead imagery
- Brightness/contrast for atmospherics
- Avoid orientation-critical tasks if sensitive (roads)

Transfer Learning (ResNet50)

Feature extraction → fine-tuning

```python
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout, Input
from tensorflow.keras.models import Model

base = ResNet50(include_top=False, weights='imagenet', input_shape=(64,64,3))
for l in base.layers[:int(0.8*len(base.layers))]:
    l.trainable = False

x = GlobalAveragePooling2D()(base.output)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
out = Dense(10, activation='softmax')(x)
model = Model(base.input, out)
```

Strategy: - Start with frozen base → quick convergence
- Unfreeze top blocks for small accuracy gains

Compare Approaches

Approach	Train Time	Accuracy
From scratch	15–25 min	92–95%
Feature extraction	5–10 min	94–96%
Partial fine-tune	10–20 min	95–97%

Tip: Use the fastest path during live sessions, fine-tune offline later

Troubleshooting

Common issues & fixes

- **GPU not detected:** set runtime → GPU; restart runtime
- **OOM error:** lower batch size; fewer filters; mixed precision
- **Accuracy stuck ~10%:** check labels; learning rate; normalization
- **Overfitting:** stronger augmentation; more dropout; L2; early stop
- **Slow training:** reduce model depth; use caching/prefetch

Philippine Context

Why CNNs matter operationally

- National land cover refresh (PhilSA)
- Cloud masking for S2 mosaics
- Disaster mapping (flood/damage)
- Urban growth monitoring
- Supports DENR, DA, NDRRMC, LGUs

Next steps: move to Day 3 (U-Net segmentation, flood mapping)

Summary & Notebook

What you achieved today

1. Built and trained a CNN classifier (EuroSAT)
2. Evaluated with robust metrics and confusion matrix
3. Applied transfer learning for higher accuracy
4. Learned practical debugging strategies

Notebook:
`session4_cnn_classification_STUDENT.ipynb`

Render slides (local):

```
cd course_site/day2/presentations
quarto render session4_cnn_lab.qmd
```