Session 3: Introduction to Deep Learning and CNNs

Neural Networks and Convolutional Architectures for Earth Observation

Stylianos Kotsopoulos
EU-Philippines CoPhil Programme

Session Overview

Duration: 2.5 hours
Type: Theory + Interactive Demos
Goal: Bridge traditional ML → deep learning for EO

You will learn: - ML → DL transition and when to use each - Neural network fundamentals (perceptron, activations) - CNN building blocks and intuition - Popular architectures (LeNet, VGG, ResNet, U-Net) - Practicalities: data, compute, transfer learning - Philippine EO applications (PhilSA, DENR, LGUs)

Prerequisites: - Sessions 1–2 completed (Random Forest) - Basics of Python/NumPy - Colab GPU runtime enabled

Resources: - Theory notebook: `session3_theory_STUDENT.ipynb` - CNN ops notebook: `session3_cnn_operations_STUDENT.ipynb`

ML $\rightarrow$ DL Transition

From feature engineering to feature learning

Traditional ML (Sessions 1–2) - Manual features: NDVI, NDWI, NDBI - Texture (GLCM), temporal, topographic - Pros: Interpretable, data-efficient - Cons: Limited by manual design

Deep Learning (Sessions 3–4) - Learns features from raw pixels - Hierarchical representations - Pros: SOTA accuracy, rich spatial context - Cons: Needs more data/compute

When to use which?

- **Random Forest:** small labeled sets, interpretability needed, fast prototype
- **CNNs:** complex spatial patterns, larger datasets, highest accuracy

Neural Network Fundamentals

Perceptron and activations

Perceptron:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Activation functions: - Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$ (probabilities) - ReLU: $\max(0, z)$ (hidden layers) - Softmax: $\mathrm{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$ (multi-class)

```python
# Perceptron skeleton for intuition (NumPy)
class Perceptron:
    def __init__(self, d):
        self.w = np.random.randn(d)
        self.b = 0.0
    def predict(self, X):
        z = X @ self.w + self.b
        return (z > 0).astype(int)
```

Training: gradient descent and backprop

Training loop: 1. Forward pass → predictions
2. Compute loss (e.g., cross-entropy)
3. Backprop gradients
4. Update weights

Key hyperparameters: learning rate, batch size, epochs

Convolutional Neural Networks

Why CNNs for images?

- Local connectivity (spatial awareness)
- Parameter sharing (few weights)
- Translation invariance (features anywhere)

Convolution:

$$(I * K)(i, j) = \sum_{m} \sum_{n} I(i + m, j + n)\, K(m, n)$$

Pooling (MaxPool 2×2): reduces spatial size, adds invariance

CNN building blocks

- Convolution (filters, stride, padding)
- Pooling (max/avg)
- Non-linearities (ReLU)
- Fully-connected head
- Regularization (dropout, weight decay)

```
Input (256×256×C)
  → [Conv + ReLU] × N → Pool → …
  → Flatten → Dense → Softmax
```

Architectures to know

LeNet-5: classic, small; education & prototypes
VGG-16: many 3×3 convs; simple but heavy
ResNet-50: residual blocks; deep & efficient

U-Net: encoder-decoder + skip connections
- Semantic segmentation (flood, buildings)
- Preserves detail via skips

EO Tasks and CNNs

Matching methods to problems

Task	Output	Typical CNN
Scene classification	One label per chip	ResNet, EfficientNet
Semantic segmentation	Pixel-wise labels	U-Net, DeepLabv3+
Object detection	Boxes + classes	YOLO, Faster R-CNN
Change detection	Change mask	Siamese/U-Net variants

Philippine use cases: - Land cover, cloud detection, floods, buildings, mining, DRM

Practical Considerations

Data requirements (rule-of-thumb)

- Simple CNN: 5k–10k samples
- ResNet (fine-tune): 1k–5k samples
- U-Net (segmentation): 100–500 labeled images

Data-centric AI

Quality > quantity; representative sampling; balanced classes; solid validation split

Transfer learning (Keras)

```python
from tensorflow.keras.applications import ResNet50
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Dropout

base = ResNet50(include_top=False, weights='imagenet', pooling='avg', input_shape=(64,64,3))
base.trainable = False  # feature extractor

model = Sequential([
    base,
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])
```

When: limited labels, need quick/strong baseline

Augmentation (EO-aware)

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
aug = ImageDataGenerator(rotation_range=90, horizontal_flip=True,
                         vertical_flip=True, brightness_range=[0.8,1.2],
                         zoom_range=0.1)
```

- Rotations/flips OK for overhead imagery
- Brightness/contrast for atmospherics
- Caution with orientation-sensitive features (roads)

Compute planning (Colab)

Model	Time (GPU)	Memory
Simple CNN	~30 min	4 GB
ResNet50 (fine-tune)	2–4 h	8 GB
U-Net	4–8 h	12 GB

Tips: mixed precision, batch size tuning, smaller chips

Philippine EO Applications

PhilSA & partners

- Cloud masking U-Net (S2): ~95% acc
- National land cover (ResNet fine-tuned)
- Flood mapping (S1 + U-Net)
- Damage assessment (object detection)

Agencies: PhilSA, DENR, DA, NDRRMC, LGUs

Summary & Resources

Key takeaways

1. CNNs learn features automatically and excel on spatial tasks
2. Architectures: ResNet (classification), U-Net (segmentation)
3. Transfer learning is the pragmatic starting point
4. Data & compute planning are essential
5. Strong fit for Philippine EO applications

Notebooks: - `session3_theory_STUDENT.ipynb` - `session3_cnn_operations_STUDENT.ipynb`

Docs: TensorFlow/Keras, CNN architectures, EO applications