🎓 Foundations of Deep Learning

Focus:

Deepening Skills in Advanced Computer Vision with Deep Learning where we dive into: more sophisticated architectures, evaluation, model tuning and optimization Topics, real-world applications of Computer Vision with Deep Learning. Explore advanced neural networks, tackle real-world problems, and deploy intelligent image systems.

🎯 Learning Objectives | Outcomes

- Understand Advanced CNN architectures and their trade-offs: ResNet, MobileNet, etc.
- Apply Transfer Learning and Fine-Tuning on custom domain-specific datasets
- Build models for Object Detection: YOLO, SSD, Faster R-CNN
- Build models for Semantic Segmentation with U-Net
- Model Evaluation and Optimization with appropriate Metrics
- Loss functions, Metrics, and Hyperparameter Tuning for better Performance

🧪 Capstone Projects

- Train a Pet Breed Classifier from scratch, Train multi-class CNN model by PyTorch in ResNet
- Lane Segmentation in Dashcam Videos
- Custom Object Detector, Detect objects in videos by YOLO & Ultralytics
- Fine-Tune a CNN to Identify Medical Conditions
  - Use pretrained ResNet on X-ray images or skin scans
  - Visualize activation maps and feature filters
- Segment Road Lanes with U-Net and OpenCV for Autonomous Driving
  - Implement lane detection pipeline from dashcam footage data
  - Post-process segmentation output with morphology filters
- Dataset Pipeline Builder, Custom data loader & Augmenter by Albumentations & TorchVision

⚙️ Tools & Frameworks & Libraries Required:

✓ Python $\geq$ 3.9
✓ PyTorch + TorchVision
✓ OpenCV
✓ Ultralytics YOLO
✓ Albumentations
✓ Matplotlib & Seaborn
✓ Weights & Biases or TensorBoard
✓ TensorFlow & TensorBoard

🔍 Core Topics & Tutorials

Model architecture, evaluation, optimization, and deployment using open-source deep learning tools

1. Advanced CNN Architectures: ResNet, MobileNet, VGG

Explore how modern convolutional neural networks like ResNet and MobileNet improve accuracy and efficiency.

Topics Covered:

- Concept of residual connections (ResNet)
- Lightweight vs Heavy models
- Code comparison on CIFAR-10 or ImageNet subsets
- ResNet, Inception, MobileNet
- Skip connections & bottlenecks

🧠 What are CNN Architectures?

Definition:

Convolutional Neural Networks (CNNs) use layers of filters to extract visual features.

Advanced architectures improve accuracy, reduce computational cost,

or optimize training by restructuring the model.

🧱 Concepts:

🔹 ResNet (Residual Networks): ResNet introduces **skip connections** that bypass one or more layers

to reduce vanishing gradient problems in deep networks.

🔹 MobileNet: uses **depthwise separable convolutions**, making it lightweight and efficient for speed

on mobile devices.

Libraries Used: PyTorch **and** TorchVision

🔧 Code Example in Python:

```
from torchvision import models
                            # pretrained = True Depricated
resnet = models.resnet18(weights = "ResNet18_Weights.DEFAULT")
mobilenet = models.mobilenet_v2(weights="MobileNet_V2_Weights.DEFAULT")
print(resnet)
print(mobilenet)
```

📌 Try replacing the output layer to classify your own dataset.

📌 Skip connections let layers learn identity mappings to stabilize gradients.

📌 Perfect for edge devices where performance meets low power usage.

2. Transfer Learning & Fine-Tuning via ResNet

🔄 What is Transfer Learning?

Definition:
Transfer learning reuses a pretrained model/ weights for faster training.

Concept:
Adjust specific layers (usually head of the model) to fit new classes while preserving learned feature extraction.

🌱 Core Idea: Reuse pretrained models and tailor them to new tasks/ dataset when data is limited.

🔹 Fine-Tuning 🔹 Adapt model to niche dataset

Topics Covered:

- Load pre-trained ResNet or MobileNet models (TorchVision) and visualize custom image dataset
- Feature extraction vs full fine-tuning
- Freeze layers and retrain classifier head
- Replace final Layer
- Apply Transfer Learning to custom datasets (pets, fashion, medical images)
- Evaluate model performance with accuracy and confusion matrix
- Fine-tune network layers
- Evaluate predictions and visualize performance
- Bonus: Export predictions with bounding boxes

📌 Visual Explanation: [Image Input] → [ResNet Backbone] → [Replaced FC Layer] → [Predictions]

✨ Setup: pip install torch torchvision

🔧 Code Example in Python: (Replace the classifier)

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

resnet50 = models.resnet50(pretrained=True)
resnet50.fc = nn.Linear(resnet50.fc.in_features, 37)  # for Example: 37 pet breeds
resnet18 = models.resnet18(pretrained=True)
# Freeze backbone
for param in resnet18.parameters():
    param.requires_grad = False
for param in resnet50.parameters():
    param.requires_grad = False
```

📣 *Useful for tasks like medical imaging or niche classification.*

📌 *Freeze early layers to keep base knowledge and speed up training.*

👉 *Fine-tuning only the last layers is a great strategy when data is scarce.*

3. **Object Detection with YOLO and SSD**

👁 What is Object Detection?

Definition:
Object Detection Identify and localize objects using **bounding boxes** and labels within images.

Topics Covered:

- Bounding box annotation formats
- YOLO (You Only Look Once), SSD (Single Shot Detector)
- Visualizing results and false positives
- Region Proposal Networks (RPN), Faster R-CNN

🔹 YOLO (You Only Look Once): Real-time single-pass detector

Concept:
YOLO predicts bounding boxes and classes in **one forward pass**, enabling real-time detection.

🔹 SSD (Single Shot Detector): Multi-scale feature maps

Concept:
SSD generates multiple feature maps and makes **class + box predictions simultaneously**.

📌 *Fast and modular, SSD works well with small datasets.*

🔹 Faster R-CNN: : Region proposals + CNN

Concept:
Combines a **Region Proposal Network (RPN)** with a CNN to refine object proposals.

✨ Setup: pip install YOLO   pip install ultralytics

🔧 Code Example in Python:

```python
from ultralytics import YOLO

model = YOLO("yolov8s.pt")  # Load small YOLO model
# Train on custom dataset
model.train(data="pet.yaml", epochs=50, imgsz=640)
# Run inference
results = model("test_pet.jpg", show=True)
model.predict("street.jpg", show=True)

from torchvision.models.detection import fasterrcnn_resnet50_fpn

model = fasterrcnn_resnet50_fpn(pretrained=True)
```

📌 *Great for high-accuracy applications like medical or traffic imaging.*

📌 *Use different anchor boxes for small vs. large objects.*

4. **Semantic Segmentation with U-Net**

🧬 What is Semantic Segmentation?

Definition:
Pixel-level classification of an image, assigning a label to every pixel. Label every pixel in an image. ideal for medical scans or self-driving cars.

🔹 U-Net

Concept:
U-Net uses **encoder-decoder** architecture with skip connections, ideal for medical and road segmentation.

Topics Covered:

- Fully Convolutional Networks (FCNs), U-Net
- Pixel-level classification tasks, Pixel-wise labeling and mask creation
- Implementing encoder–decoder architecture
- Dataset: Cell segmentation or Cityscapes

🔧 Code Example in Python: **Basic U-Net Block -** Prediction Preview (Static Mask Overlay)

```python
import torch
import torch.nn as nn

class UNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.enc = nn.Conv2d(3, 64, 3, padding=1)
        self.dec = nn.ConvTranspose2d(64, 1, 2, stride=2)
    def forward(self, x):
        x1 = F.relu(self.enc(x))
        x2 = self.dec(x1)
        return torch.sigmoid(x2)

import numpy as np
import matplotlib.pyplot as plt
image = plt.imread("dashcam.jpg")
mask = np.load("predicted_mask.npy")  # Binary mask from U-Net
plt.imshow(image)
plt.imshow(mask, cmap='Reds', alpha=0.4)
plt.title("Segmented Lane Overlay")
plt.axis("off")
plt.show()
```

📌 *U-Net shines in tasks with limited data and well-defined boundaries.*

📣 Try with road lane masks in dashcam footage.

5. **Model Evaluation, Loss Functions & Tuning (Optimization)**

Topics Covered:

- Loss functions (CrossEntropy, IoU Loss)
- Metrics (Precision, Recall, F1, mAP)
- Hyperparameter search (grid/random) and tuning strategies
- Batch size, learning rate, dropout
- Use of TensorBoard or Optuna

📉 What are Loss Functions?

Definition:
Loss functions measure the error between predicted and actual labels.

🔹 CrossEntropy Loss for classification

Use: Classification

```
loss_fn = nn.CrossEntropyLoss()
```

🔹 IoU Loss for segmentation (Intersection over Union)

Use: Object detection/segmentation

📌 *Higher IoU = better overlap between prediction and ground truth.*

🧪 Metrics

🔹 Precision/Recall/F1

🔹 Mean Average Precision (mAP)

Precision, Recall, F1 Score, mAP
These evaluate performance beyond accuracy.

🔧 Hyperparameter Tuning

```
optimizer = torch.optim.Adam(model.parameters(), lr=0.0003)
```

Definition:
Systematically adjust training variables (learning rate, batch size) to maximize performance.

🧪 Sample Projects

🐶 Pet Breed Classifier

- **Goal:** Train model to classify 37 cat/dog breeds
- **Tools:** PyTorch, torchvision, ResNet18

🚗 Lane Segmentation

- **Goal:** Detect lane boundaries in dashcam videos
- **Tools:** U-Net, OpenCV, image masks

🔧 Code Example in Python:

```python
import torch
import torch.nn as nn
from sklearn.metrics import f1_score

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
# Training loop (simplified)
for epoch in range(10):
    outputs = model(inputs)
    loss = criterion(outputs, targets)
    loss.backward()
    optimizer.step()
```

📌 *Use libraries like Ray Tune or Optuna to automate search.*

📌 *Use TensorBoard to visualize performance across epochs.*