Intro to Deep Learning - 0571418201
Course Project 2025/2026

Dor Bar - dorbar1@mail.tau.ac.il

Introduction to Deep Learning – Final Project 2025/2026

🎨 Bring Van Gogh Back to Life: Style Transfer & Classification 🎨

Submissions due: 25.1.26.

The project must be submitted in triplicate.

(Instructions for accessing data and Lab 424 are on the last page.)

***Pay attention:** this project contains 3 parts and bonus question - make sure to go over all of the instructions below!*

Overview

Over the past few weeks, you've delved into the exciting world of deep learning. Now, it's time to put your knowledge into action with a creative twist!

Your mission is two-fold:

1. First, you will train a classifier to become an "art expert" that can distinguish a real Van Gogh painting from other Post-Impressionist works.
2. Second, you will build a neural style transfer function to "paint" new images in Van Gogh's iconic style and then use your classifier to evaluate your own creations.

This project will challenge you to connect classification, transfer learning, and generative techniques. Your analysis and explanations are just as important as your code.

Focus Models: VGG-19 and AlexNet

```
Python

# Importing the pre-trained models (as shown in exercise 6)
from torchvision import models
vgg19 = models.vgg19(pretrained=True)
alexnet = models.alexnet(pretrained=True)
```

Part 1: The Van Gogh Classifier

Your first goal is to fine-tune pre-trained models to perform **binary classification**: determining whether a painting was painted by Van Gogh (label: `is_van_gogh: yes/no`).

1.1. Goal

Fine-tune **both** VGG-19 and AlexNet on the provided Post-Impressionism dataset to act as "Van Gogh detectors."

1.2. Data Preparation

- **Dataset:** Use the provided Post-Impressionism dataset. (See "Accessing the Data" on the last page).
- **Labels:** The `classes.csv` file contains the metadata. Use the artist's name to create your binary labels (`"Vincent van Gogh"` vs. all others).
- **Transformations:** Apply all necessary transformations (e.g., resizing, normalization) to prepare the images for your pre-trained models. Remember, VGG and AlexNet may have different input requirements!

1.3. Model Training & Tuning

- **Data Augmentation:** Apply a variety of data augmentation techniques (e.g., rotations, flips, color jitter, cropping) during training to create a more robust model.
- **Hyperparameter Tuning :**
    - Use Optuna to perform a systematic hyperparameter search. Tune key parameters such as learning rate, batch size, and optimizer type.
    - To keep the search computationally reasonable and comparable across students, use the following constraints:
        - Design your search so that, when running on a single Lab 424 GPU, the total hyperparameter search takes at least **half an hour of searching and does not exceed 1 hour**.
    - In your report, you must clearly state:
        - How many trials you actually ran.
        - How many epochs were used per trial.
        - The approximate total wall-clock time of your Optuna search.
    - Make sure to log all trials and results in Weights & Biases (W&B) so that you can later analyze the learning curves and validation metrics.
- **Experiment Tracking :**
    - You **must** use **Weights & Biases (W&B)** to track all your experiments.
    - Log your hyperparameters, augmentation pipelines, and performance metrics for each run. Your final report must include explanations and analysis of your W&B graphs.
- **Cross-Validation (Optional):**
    - To improve generalization, you can implement K-Fold Cross-Validation. This is **optional** as it is **computationally intensive**, but it is good practice.
    - A standard train/validation/test (train/test is fine also) split is the minimum requirement.

1.4. Evaluation

- **Metrics:** Evaluate your final models using a comprehensive set of metrics:
    - **Accuracy:** Overall performance.
    - **AUC-ROC:** Ability to distinguish between classes.
    - **F1-Score:** Balance between precision and recall.
    - **Confusion Matrix:** A full breakdown of your model's predictions.

Part 2: Recreating Van Gogh's Style

In this section, you will create a style transfer function and then automate the search for the "best" style-transfer parameters.

Part 2A: The Style Transfer Function

You will create a single, generic function capable of performing neural style transfer.

- **Required Inputs:**
  - `model`: The pre-trained network (VGG-19 or AlexNet).
  - `content_image`: The image whose structure you want to preserve.
  - `style_image`: The image whose artistic style you want to apply.
  - `content_layers`: A list of layer names to use for content features.
  - `style_layers`: A list of layer names to use for style features.
  - `content_weight`: A scalar value (α) to control the emphasis on content.
  - `style_weight`: A scalar value (β) to control the overall intensity of the style.
  - `style_layer_weights`: A list of weights to balance the contribution of each style layer.
- **Output:**
  - The final, stylized image.

Part 2B: Hyperparameter Search for Style Transfer

Your style transfer function has its own hyperparameters (the weights α, β, etc.). You will **find optimal parameters** using a systematic search.

- **Goal:** Find a "global" set of style transfer hyperparameters (e.g., `content_weight`, `style_weight`) that consistently produces images that your **Part 1 classifier** believes are *most* like Van Gogh.

  This is a new optimization task, with a new objective function that you need to define. Use the classifier from Part 1 to evaluate the style transfer and use this evaluation as the objective for a new Optuna search.

- **Method:**
  1. Choose **one** of your classifiers from Part 1 (e.g., your best-performing VGG-19). This will serve as your "judge."
  2. Choose a fixed set of content and style images to use for this search.
  3. Define an **objective function** for an optimizer like **Optuna**.
  4. This function should:
     - Take in hyperparameter suggestions from Optuna (e.g., `content_weight`, `style_weight`).
     - Run your style transfer function with these parameters.
     - Pass the *output* (stylized) image through your Part 1 classifier.
     - Return a score to be **maximized** - for example, the classifier's output probability for the "is_van_gogh" class.
- **Deliverable:** Report the "optimal" set of hyperparameters you found and discuss the process.

Part 2C: Application & Evaluation

Now, use your function and newfound knowledge.

1. **Apply the Function:**
    - For **both** VGG-19 and AlexNet, use your style transfer function on **20 images of your choice**. (Personal photos are highly encouraged!)
    - Use at least **5 different Van Gogh paintings** as your style images.
2. **Set Epochs:** Decide on a consistent number of optimization epochs (steps) for the style transfer process. This number must be the same for all images and models.
3. **Evaluate Quality:**
    - Use **both** of your Part 1 classification models to assess the "Van Gogh-ness" of your 20 generated images.
    - Include a detailed discussion of these results in your report.
4. **Compare Models:** Evaluate and compare the style transfer performance of VGG-19 vs. AlexNet.

Part 3: Discussion & Analysis

Your explanations and discussions will significantly impact your grade. We are particularly interested in your ability to analyze your results, understand the "why" behind the code, and articulate the core concepts.

Part 1 Discussion:

1. **Dataset:** Provide an overview of the dataset. What proportion consists of Van Gogh's paintings?
2. **Models & Transformations:** Explain the VGG-19 and AlexNet models. Highlight their key differences (architecture, original training data, etc.). Describe the specific transformations and augmentations you applied and *why*.
3. **Training & Tuning:** Outline your training approach. Discuss your final choices for the optimizer, learning rate, etc. **Include screenshots of your W&B and Optuna graphs** and explain what they show.
4. **Model Comparison:** Analyze and compare your two classifiers. Which performed better? Why?
5. **Prediction Analysis:** Present the **full confusion matrix** for both models. Show and analyze examples of True Positives, True Negatives, False Positives (What non-Van Gogh paintings fool your model?), and False Negatives.

Part 2 Discussion:

1. **Style Transfer Process:** Explain the mechanics of neural style transfer. Attach a screenshot of your style transfer function's code.
2. **Style Loss Normalization:** Discuss why it is (or isn't) important to normalize the loss from each style layer.
3. **Hyperparameter Search:** Describe the setup for your **Part 2B search**. What were the "optimal" `content_weight` and `style_weight` values found by Optuna?
4. **Classifier Evaluation:** Analyze the results from Part 2C. Show examples where:
    - Both models classified the result as "Van Gogh."
    - One model did, but the other did not.
    - Neither model classified the result as "Van Gogh."
5. **Layer Selection:** Which layers did you choose for content and style extraction? **Justify your choices.**
6. **Overall Reflection:** How do the results from Part 2 align with your findings from Part 1?

Part 3 Discussion:

To demonstrate that you actually used the GPU, you must include a short benchmark comparing CPU vs GPU performance for your model.

You may choose one of the following options:

1. Measure the time required for a single training epoch on CPU vs GPU, or

2. Measure the time required for 20 forward+backward iterations (mini-batches) on CPU vs GPU.

For this benchmark, you must print and report:

- The machine name (from `platform.node()`).
- The GPU model (from `torch.cuda.get_device_name(0)`).
- Print the measured time on CPU and on GPU.

Include the results and a short discussion in your report:

- Was the speedup as you expected?
- Did you notice any bottlenecks (e.g., data loading, small batch sizes, I/O)?

This benchmark must appear both in your notebook (with printed outputs) and in the PDF report (in this Discussion section). In the first code cell of your main notebook, include and run:

```
import torch

import os

import platform

print("cuda available:", torch.cuda.is_available())

if torch.cuda.is_available():

    print("GPU Device:", torch.cuda.get_device_name(0))

print("Python version:", platform.python_version())

print("Machine name:", platform.node())   # This is unique to the machine

print("Working directory:", os.getcwd())

if torch.cuda.is_available():

    print("Initial GPU memory usage:")

    print(torch.cuda.memory_allocated() / 1024**2, "MB allocated")

    print(torch.cuda.memory_reserved()  / 1024**2, "MB reserved")
```

Showing GPU Activity with nvidia-smi

While your model is training on the GPU, run the following command in a notebook cell:

!nvidia-smi

Take a **screenshot** (or save the notebook output as an image) showing:

- GPU utilization.
- GPU memory usage during training.

Include this screenshot in your PDF report (for example, in Part 1 Discussion or in an Appendix) with a short caption explaining what we see.

🌟 Bonus: "What is My Classifier Thinking?" (Model Interpretability) 🌟

Conceptual Background:

In Part 1, you built a classifier that acts like a "black box." It takes an image in and outputs a probability ("Van Gogh" or "Not Van Gogh"). But why did it make that decision? What parts of the image was it "looking" at?

A technique called **Gradient-weighted Class Activation Mapping (Grad-CAM)** lets us create a "heatmap" that highlights the regions of an image that were most important for a specific classification. This bonus asks you to implement a Grad-CAM function to understand *what* your classifier learned.

Part A: The Grad-CAM Function

1. **Build the Function:** Create a function that can perform Grad-CAM.
2. **Inputs:** The function should take your fine-tuned classifier from Part 1, a target image, and the *final convolutional layer* of the model.
3. **Process (Hint):** You will need to use **PyTorch hooks** to get the feature maps from your target layer and the gradients flowing back into it. You'll then compute the weighted sum of the feature maps (weighted by the global average of the gradients) and pass it through a ReLU to get the heatmap.

Part B: Analysis of the Classifier

Now, use your Grad-CAM function to analyze your classifier's "thought process."

1. **True Positives:** Take 3-5 images your model *correctly* identified as Van Gogh. Generate their heatmaps. **Analyze:** Where did the model "look"? Did it focus on swirls, thick brush strokes, or faces?
2. **False Positives:** Take 3-5 images your model *incorrectly* labeled as Van Gogh. Generate their heatmaps. **Analyze:** What *fooled* your model? Did it find a swirly cloud in another artist's landscape?
3. **Conclusion:** Based on your heatmaps, what "visual signature" did your classifier *actually* learn for Van Gogh?

Part C: Improving Part 2 (Conceptual Question)

This is a conceptual question - no implementation is required.

In Part 2, your style loss was applied *uniformly* across the entire image. This can sometimes "ruin" important content (like faces).

The Question: How could you use your Grad-CAM function to *improve* the style transfer process from Part 2?

- **Hint:** Think about using the heatmap as a **spatial weight mask**.
- Describe a method to modify your `content_loss` or `style_loss` calculation.
- What would be the benefit? (e.g., could you use it to *protect* certain areas from style transfer or *amplify* it in others?) Explain your proposed modification.

Grading Criteria

The project grade will be based on the following breakdown:

- Part 1 – Van Gogh Classifier: 35%

  Data preparation, model training, hyperparameter tuning (including Optuna constraints), evaluation metrics, and the use of W&B.

- Part 2 – Style Transfer & Hyperparameter Search: 40%

  Correct implementation of the style transfer function, design and execution of the Optuna search for global style hyperparameters, quality and diversity of generated images, and evaluation using the classifiers.

- Part 3 – Discussion & Analysis (including CPU vs GPU benchmark): 25%

  Depth and clarity of the written discussion, quality of the analysis of results and graphs, interpretation of successes/failures, and the hardware benchmark and reflection.

- Bonus – Interpretability (e.g., Grad-CAM): 10 pt

Intro to Deep Learning - 0571418201
Course Project 2025/2026

Submission Instructions

- **Report (PDF):**
  - Write your report in a Word document and submit it as a **PDF**.
  - Include all relevant results, images, and graphs. Use appendices for large collections of images.
  - Clearly separate your discussion answers to align with the questions.
  - Specify any non-standard Python packages used and explain why you chose them.
- **Code (.ipynb):**
  - Submit your outputs **in .ipynb format** (Jupyter Notebook).
  - If you used regular scripts (pycharm, vs code etc. upload those scripts as well).
  - Ensure all cells are executed and all outputs are saved and visible.
  - A link to a **GitHub repository** containing the notebooks is also acceptable.
- **Compression:**
  - Submit both the PDF report and the .ipynb file(s) in a **single compressed file (.zip or .rar)**.
  - Name this file: `Group_XX_DL_Project.zip` (where XX is your group number).

Accessing the Data & Lab Instructions

You have two main options for accessing the data.

Option 1: Lab 424 (Recommended)

The curated dataset (Post-Impressionism subset) is available for you.

- **Location:** `D:\course_data`
- Inside this directory, you will find:
  1. The `classes.csv` metadata file.
  2. The directory of all the corresponding images.

Option 2: Home / Kaggle

If you prefer to work on your own machine, you can.

1. **Kaggle:** The larger, full dataset is available on the Kaggle "WIKIART" website.
2. **Warning:** This is the complete dataset, which is very large (approx. 32 GB).
3. **Filtering:** You will need to download this file and then filter it to contain only the "Post-Impressionism" data (each genre is a directory so it won't be hard to filter it).

Fine-tuning Hint:

To efficiently fine-tune the model, it is recommended to do so using PyCharm (it can be 3-4 times faster than in a notebook). You can save the trained model (.pth file) and then load it into your notebook to continue your analysis and display outputs.

Intro to Deep Learning - 0571418201
Course Project 2025/2026