Image Classification

Graduate Quantitative Economics and Datascience

Jesse Perla

jesse.perla@ubc.ca

University of British Columbia



Table of contents

- Overview
- EuroSAT
- A Vision-Language Model
- A Downloaded Classifier
- Comparing the Models

Overview

Images as Economic Data

- A satellite chip is **unstructured data**, $64 \times 64 \times 3$ numbers. A classifier turns it into a variable: built-up or agricultural land, roads, forest, water (Donaldson and Storeygard 2016)
- The classifier is a **measuring instrument**: its mistakes are measurement error in any regression the labels enter, and here they vary with the landscape and the prompt. The classification lecture covers what that does and how to correct it
- Two routes, neither of which trains anything: a **vision-language model** through Ollama (the local LLM lecture, with `qwen3.5:4b` pulled, or `qwen3.5:2b` on 8 GB machines) and a **downloaded classifier** trained for exactly this task (`uv sync --extra extras` for `torch` and `transformers`)

EuroSAT

Vibe-coding Warning!

This material has not yet been fully vetted and checked.

The Data

- EuroSAT (Helber, Bischke, Dengel, and Borth 2019): $27{,}000$ Sentinel-2 chips, 64×64 pixels at 10 m, ten land-use classes; 92 MB on the Hub as `cm93/eurosat`
- `load_dataset` downloads it once. The test rows are in random order, so the first 100 are a sample

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from datasets import load_dataset

chips = load_dataset("cm93/eurosat", split="test[:100]")
labels = chips.features["label"].names
truth = [labels[i] for i in chips["label"]]
print(chips[0]["image"].size, truth[0])
```

README.md: 100% 1.69k/1.69k [00:00<00:00, 274kB/s]

data/train-00000-of-00001.parquet: reconstructing file: 100% 73.6MB / 73.6MB, 7.11MB/s

data/train-00000-of-00001.parquet: downloading bytes: 71.6MB, 6.81MB/s



data/validation-00000-of-00001.parquet: reconstructing file: 100% 9.13MB / 9.13MB, 902kB/s

data/validation-00000-of-00001.parquet: downloading bytes: 8.91MB, 879kB/s

data/test-00000-of-00001.parquet: reconstructing file: 100% 9.20MB / 9.20MB, 907kB/s

data/test-00000-of-00001.parquet: downloading bytes: 8.96MB, 882kB/s

Generating train split: 100% 21600/21600 [00:00<00:00, 187807.63 examples/s]

Generating validation split: 100% 2700/2700 [00:00<00:00, 123174.04 examples/s]

Generating test split: 100% 2700/2700 [00:00<00:00, 134100.12 examples/s]

```
(64, 64) SeaLake
```

- Each observation is (x_n, y_n), an image and a class. No hand-built features



Ten Chips

```
fig, axes = plt.subplots(2, 5, figsize=(9, 3.9))
for ax, label in zip(axes.flat, labels):
    ax.imshow(chips[truth.index(label)]["image"])
    ax.set_title(label, fontsize=10)
    ax.axis("off")
plt.show()
```



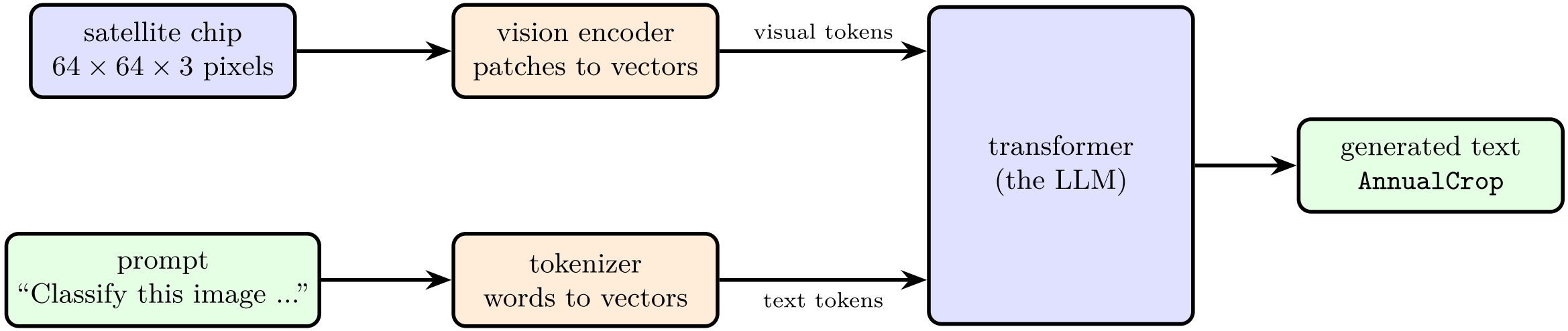
- Each chip covers 640 m on a side. Some classes are obvious at this resolution; `Pasture`, `HerbaceousVegetation`, and `PermanentCrop` are not, even for a person

A Vision-Language Model



Can a Language Model See?

- A **vision-language model** adds a vision encoder that cuts the image into patches and maps each to a vector in the token space. The transformer reads image and text together and generates text
- Qwen 3.5, the local LLM lecture's model, is built this way, so the chat call already accepts images



Two Ways to Connect Pixels and Words

- **Embedding models** (CLIP, and RemoteCLIP for satellite imagery) put images and text in one vector space. Embed the chip and each label, pick the closest: a **zero-shot classifier** in the spirit of the embeddings lecture
- **Generative models** (Qwen 3.5, Qwen3-VL, GPT-4o, Gemini) put the visual tokens in front of the LLM and generate the label as text, given the image, a question, and the allowed labels
- We use the generative route: the local LLM lecture's chat call with one more content part

Encoding the Image

- The chip travels inside the JSON request as a base64 **data URL**. The question lists the allowed labels, so the class list lives in the prompt, not in the weights

```
import base64
import io
from openai import OpenAI

client = OpenAI(base_url="http://localhost:11434/v1", api_key="ollama")

def image_url(image):
    buffer = io.BytesIO()
    image.save(buffer, format="PNG")
    data = base64.b64encode(buffer.getvalue()).decode()
    return "" + data

question = ("Classify this satellite image as exactly one of: "
            + ", ".join(labels) + ". Answer with the label only.")
```



Same Client, Same Call, Plus an Image

- `content` is a list of parts, text and image, instead of a string; the rest is the local LLM lecture's call. `temperature=0` makes reruns repeatable, and `reasoning_effort="none"` skips Qwen's thinking, which took 30 to 50 seconds per chip for a few points

```
def classify(image, model="qwen3.5:4b", examples=()):
    response = client.chat.completions.create(
        model=model, temperature=0, reasoning_effort="none",
        messages=[{"role": "user", "content": [
            *examples,
            {"type": "text", "text": question},
            {"type": "image_url", "image_url": {"url": image_url(image)}},
        ]}],
    )
    return response.choices[0].message.content.strip()

predicted = [classify(row["image"]) for row in chips]
print(np.mean([p == t for p, t in zip(predicted, truth)]))
```

```
0.51
```

- No training step, well under a minute for 100 chips; `examples` is for the next slide. Runnable version: `eurosat_vlm.py`

One Labelled Chip per Class in the Prompt

- The text lecture's one-shot prompt, with images: labelled chips from outside the scored 100 go in front of the question, and the weights still never change

```
pool = load_dataset("cm93/eurosat", split="test[100:300]")
pool_truth = [labels[i] for i in pool["label"]]
examples = [{"type": "text", "text": "One labelled example of each class:"}]
for label in labels:
    url = image_url(pool[pool_truth.index(label)]["image"])
    examples += [{"type": "image_url", "image_url": {"url": url}},
                 {"type": "text", "text": label}]
predicted = [classify(row["image"], examples=examples) for row in chips]
```

Model	Zero-shot	Ten examples
Qwen 3.5 2B	27%	33%
Qwen 3.5 4B	51%	62%
Qwen 3.8 27B	60%	71%
Qwen3-VL 4B	41%	54%

- About ten points for every model, at a few hundred tokens per request. The ViT saw 21,600 labelled chips. Runnable version: `eurosat_vlm_examples.py`

A Downloaded Classifier



Weights Somebody Else Trained

- The Hub also hosts task-specific classifiers. `Haifald/vit-eurosat` is a Vision Transformer (86 million parameters) fine-tuned on the $21{,}600$ training chips; its card reports 98.7% test accuracy
- Three lines with `transformers`, which the NLP lecture skipped because most of it is about training. `pipeline` downloads 340 MB once, resizes each chip to 224×224, and runs on the CPU

```
from transformers import pipeline

vit = pipeline("image-classification", model="Haifald/vit-eurosat")
print(vit(chips[0]["image"], top_k=3))
```

```
[{'label': 'SeaLake', 'score': 0.99993},
 {'label': 'AnnualCrop', 'score': 0.00001},
 {'label': 'Residential', 'score': 0.00001}]
```

- Runnable version: `eurosat_vit.py`



What the Classifier Computes

- One score per class, $z = f_\theta(x) \in \mathbb{R}^{10}$, and the softmax of the classification lecture

$$\mathbb{P}(y = k \mid x) = \frac{e^{z_k}}{\sum_{j=1}^{10} e^{z_j}}, \qquad \hat{y} = \operatorname{argmax}_k \mathbb{P}(y = k \mid x)$$

- Multinomial logit had $z_k = x^\top \beta_k$ with hand-built features. Here f_θ is a transformer over 16×16 pixel patches, and θ has 86 million entries fit by cross-entropy on labelled chips, on someone else's GPU
- The VLM has no such head. The classes enter the prompt, and the label comes out as text

Comparing the Models



Accuracy on the Same 100 Chips

Class	n	Qwen 3.5 2B	Qwen 3.5 4B	Qwen 3.8 27B	Qwen3-VL 4B	GPT-4o mini	ViT
`Forest`	8	12%	75%	88%	38%	25%	100%
`River`	13	0%	54%	15%	69%	69%	100%
`Highway`	7	14%	100%	86%	86%	100%	100%
`AnnualCrop`	11	0%	0%	18%	55%	36%	100%
`SeaLake`	17	24%	100%	100%	82%	94%	94%
`HerbaceousVegetation`	10	0%	0%	0%	0%	0%	100%
`Industrial`	9	89%	0%	78%	0%	100%	100%
`Residential`	9	44%	78%	89%	33%	89%	100%
`PermanentCrop`	9	100%	78%	56%	0%	0%	100%
`Pasture`	7	0%	0%	86%	0%	43%	100%
All	100	**27%**	**51%**	**60%**	**41%**	**58%**	**99%**

- The same chips, the same zero-shot question, `temperature=0`, September 2026. Qwen 3.8 27B is the 48 GB Mac row of the local LLM lecture's table; GPT-4o mini is the NLP lecture's hosted model, `OpenAI()` with the defaults and no `reasoning_effort`, about 15 cents for the 100 chips



Where They Disagree



- The first five chips the VLM gets wrong, with the ViT's answer. Read the errors, not only the rate: `Pasture` for `HerbaceousVegetation` is harmless for many uses, a lake for a forest is not

Specialized or General?

- The ViT was trained for these ten classes and nothing else: 100 chips in seconds, a probability per class, fixed outputs. A new question means labelling data and training again
- The VLM had no training from us. The classes are words in the prompt, and the same model answers other questions about the same chip; the table is the price
- Specialization is not automatic: Qwen3-VL loses to Qwen 3.5 4B, seven times the parameters buy nine points, ten labelled chips about as many, and only the fine-tuned ViT is reliable here

Things to Try and Extra Materials

- Ask yes or no about roads, or for JSON attributes with `response_format={"type": "json_object"}`, and check which chips flip; try `qwen3.5:2b`, `qwen3-vl:4b-instruct`, or `qwen3.8:27b-mlx`, and thinking by dropping `reasoning_effort`
- Aggregate predictions to a region, regress an outcome on the share of built-up land, then estimate p_0 and p_1 from labelled chips and correct the slope (classification lecture)
- The tables come from the committed `data/eurosat_predictions.csv`. Rebuild it with `uv run python utilities/build_eurosat_predictions.py`, then render with `--no-cache`
- References
  - → Helber, Bischke, Dengel, and Borth (2019), "EuroSAT", *IEEE JSTARS*, and the dataset on the Hub
  - → Donaldson and Storeygard (2016), "The View from Above", *JEP*; Jean et al. (2016), *Science*, on predicting poverty from imagery
  - → Dell (2025), "Deep Learning for Economists", *JEL*, and EconDL
  - → Ollama vision and OpenAI compatibility documentation