Applying a Pretrained Image Classifier in PyTorch: Steps & Minimal Script

Nareli

Overview

This note shows the practical steps to run inference with a pretrained image classification model using **PyTorch** and `torchvision`. It ends with a compact, copy-pasteable script that prints the Top-5 predictions for a single image.

Prerequisites

- Python 3.9+ recommended.
- Packages: `torch`, `torchvision`, `Pillow`.

Install:

```
pip install torch torchvision pillow
```

Steps (PyTorch)

1. **Choose a pretrained backbone.** e.g., ResNet-50, EfficientNet, ViT from `torchvision.models`.
2. **Load pretrained weights.** Use the *weights enum* so preprocessing and categories match the model.
3. **Switch to eval mode.** `model.eval()` to disable dropout and BN updates.
4. **Build the preprocessing pipeline.** Use `weights.transforms()` (resize, crop, normalize).
5. **Load an image.** Open with PIL (`convert("RGB")`).
6. **Preprocess and add batch dim.** Transform to tensor, then `unsqueeze(0)`.
7. **Pick device.** `cuda` if available, else `cpu`; move model and batch there.

8. **Forward pass without gradients.** Wrap in `torch.no_grad()` and obtain logits.

9. **Postprocess.** Softmax to probabilities; `topk` for top-N; map indices to class names.

10. **(Optional) Batch inference.** Use a `Dataset` + `DataLoader` for throughput.

11. **(Optional) Custom labels.** If you fine-tuned a head, replace the default ImageNet label list.

12. **(Optional) Export/serve.** TorchScript/ONNX; wrap in CLI or an API (e.g., FastAPI) for deployment.

Minimal Working Example (Single Image, Top-5)

Save as `classify.py` and run `python classify.py path/to/image.jpg`.

```python
import sys
import torch
from torchvision import models
from PIL import Image

def main(image_path: str):
    # 1) Pick model + weights
    weights = models.ResNet50_Weights.IMAGENET1K_V2
    model = models.resnet50(weights=weights)

    # 2) Eval mode + device
    model.eval()
    device = "cuda" if torch.cuda.is_available() else "cpu"
    model.to(device)

    # 3) Preprocessing bound to the chosen weights
    preprocess = weights.transforms()

    # 4) Load & preprocess image
    img = Image.open(image_path).convert("RGB")
    batch = preprocess(img).unsqueeze(0).to(device)

    # 5) Inference (no gradients)
    with torch.no_grad():
        logits = model(batch)

    # 6) Softmax + top-k
    probs = torch.softmax(logits[0], dim=0)
    top_probs, top_idxs = probs.topk(5)
    # 7) Human-readable labels (ImageNet)
    categories = weights.meta["categories"]

    print("Top-5 predictions:")
    for p, i in zip(top_probs.tolist(), top_idxs.tolist()):
        print(f"{categories[i]:<30}  {p:.4f}")

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python classify.py <image_path>")
        sys.exit(1)
    main(sys.argv[1])
```

Notes & Tweaks

- **Different model?** Swap `resnet50` for `efficientnet_b3`, `vit_b_16`, etc., and adjust the weights enum accordingly.
- **Speed.** For many images, use batches (e.g., 32) and `DataLoader` with `num_workers`.
- **Determinism.** Set seeds and `torch.backends.cudnn.deterministic = True` if needed.
- **Export.** Consider `torch.jit.script(model)` or `torch.onnx.export(...)` for deployment to other runtimes.