Cat vs. Dog Image Classification Using Deep Learning in Python

Carter Susi

Dec 02, 2025

CAP4613

Repo: https://github.com/cartersusi/susi-ml-bonus

```
!pip install kagglehub matplotlib torch torchinfo torchvision tqdm scik
```

```
Requirement already satisfied: kagglehub in /usr/local/lib/python3.12/
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12
Requirement already satisfied: torch in /usr/local/lib/python3.12/dist
Collecting torchinfo
  Downloading torchinfo-1.8.0-py3-none-any.whl.metadata (21 kB)
Requirement already satisfied: torchvision in /usr/local/lib/python3.1
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.
Requirement already satisfied: seaborn in /usr/local/lib/python3.12/di
Requirement already satisfied: packaging in /usr/local/lib/python3.12/
Requirement already satisfied: pyyaml in /usr/local/lib/python3.12/dis
Requirement already satisfied: requests in /usr/local/lib/python3.12/d
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/pyth
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/pyt
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/pyt
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3.1
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/pyth
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/
Requirement already satisfied: filelock in /usr/local/lib/python3.12/d
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local
Requirement already satisfied: setuptools in /usr/local/lib/python3.12
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3
Requirement already satisfied: networkx>=2.5.1 in /usr/local/lib/pytho
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dis
Requirement already satisfied: fsspec>=0.8.5 in /usr/local/lib/python3
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /u
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/lo
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/lo
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/loc
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/l
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/
```

```
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/l
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in /usr/local/
Requirement already satisfied: nvidia-nvshmem-cu12==3.3.20 in /usr/loc
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/lo
Requirement already satisfied: triton==3.5.0 in /usr/local/lib/python3
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/
Requirement already satisfied: pandas>=1.2 in /usr/local/lib/python3.1
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/d
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/py
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/pytho
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/py
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/py
Downloading torchinfo-1.8.0-py3-none-any.whl (23 kB)
Installing collected packages: torchinfo
```

Set Config

```
import os
import shutil

import torch
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

import warnings
from PIL import Image, ImageFile

warnings.filterwarnings("ignore", category=UserWarning, module="PIL")
warnings.filterwarnings("ignore", ".*Truncated File Read.*")
ImageFile.LOAD_TRUNCATED_IMAGES = True

cwd = os.getcwd()
home = os.path.expanduser("~")

conf = {
  "models_dir": f"{cwd}/models",
  "kaggle_creds": f"{home}/.kaggle/kaggle.json",
  "model_path": f"/{cwd}/models/cat_dog_cnn_25.pth"
}

def copy_creds(from_path: str, to_path: str) -> bool:
  try:
    os.makedirs(os.path.dirname(conf["kaggle_creds"]), exist_ok=True)
```

```
    shutil.copy(from_path, to_path)
    os.chmod(to_path, 0o600)
  except Exception as e:
    print("error copying kaggle credential file", e)

if not os.path.exists(conf["kaggle_creds"]):
  try:
    from google.colab import files
    print("Please upload your 'kaggle.json' credential file\n")
    files.upload()
    copy_creds("kaggle.json", conf["kaggle_creds"])
  except ImportError:
    if not os.path.exists("kaggle.json"):
      raise Exception(f"kaggle.json is not in current directory or {c
    copy_creds("kaggle.json", conf["kaggle_creds"])


try:
  import google.colab

  image_size = (64, 64)
  print(f"In Colab, using image size: '{image_size[0]}x{image_size[1]
except ImportError:
  vram = torch.cuda.get_device_properties(0).total_memory / (1024**3)
  if vram < 10.0:  # Needs ~8.5 Gb for 256x256
    image_size = (128, 128)
  else:
    image_size = (256, 256)

  print(
    f"Not in Colab with {vram} GB VRAM, default image size: '{image_s
  )
```

```
Please upload your 'kaggle.json' credential file
Browse... kaggle.json
kaggle.json(application/json) - 66 bytes, last modified: n/a - 100% done
Saving kaggle.json to kaggle.json
In Colab, using image size: '64x64px'
```

Download Cat Vs Dog Dataset

```
import kagglehub

def _find_image_folder(root_path: str) -> str:
  for dirpath, dirnames, _ in os.walk(root_path):
    if "Cat" in dirnames and "Dog" in dirnames:
      return dirpath
  return ""
```

```
dataset_path = _find_image_folder(
  kagglehub.dataset_download("karakaggle/kaggle-cat-vs-dog-dataset")
)

print(f"Kaggle Dataset at: {dataset_path}")
```

```
Using Colab cache for faster access to the 'kaggle-cat-vs-dog-dataset'
Kaggle Dataset at: /kaggle/input/kaggle-cat-vs-dog-dataset/kagglecatsa
```

Load Dataset

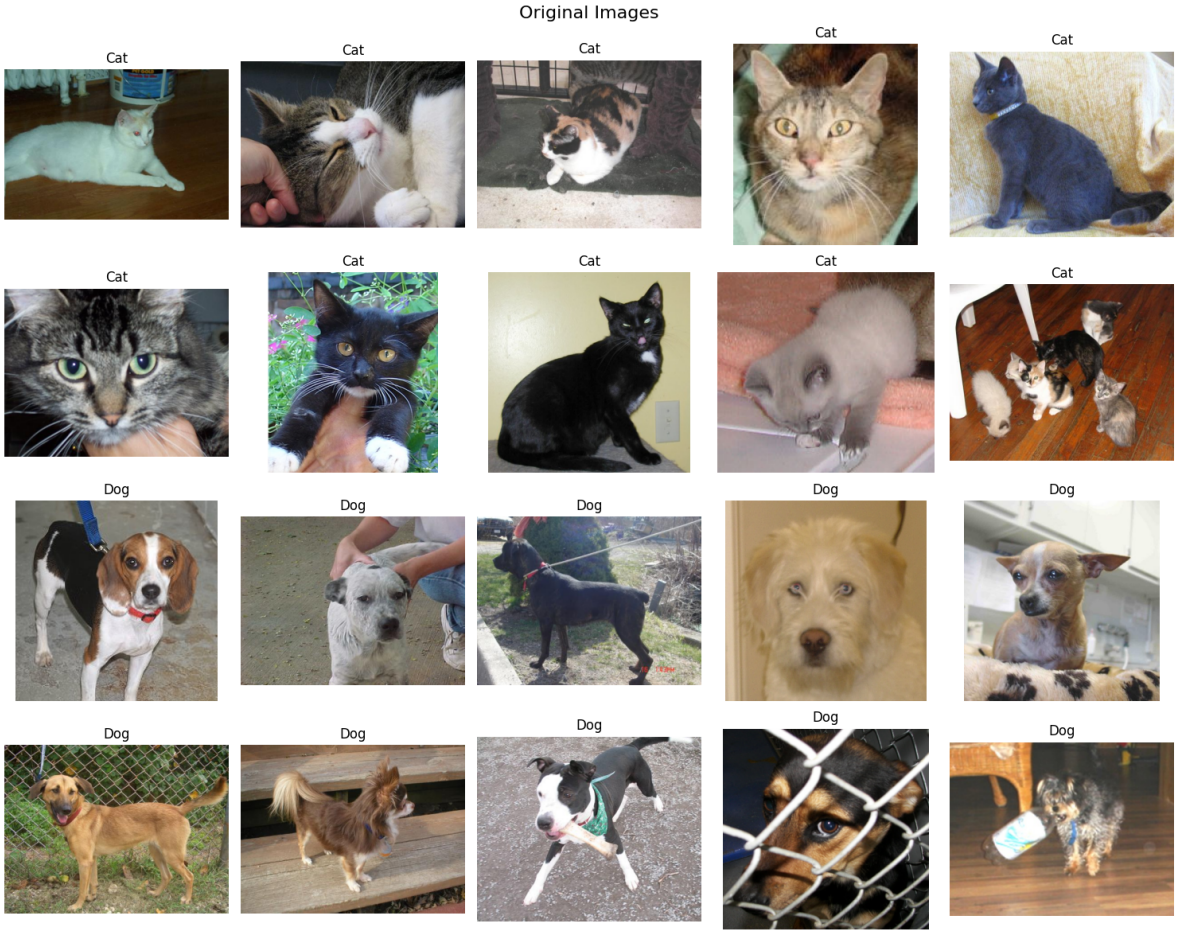
```
import random

import matplotlib.pyplot as plt
import torch
from torch.utils.data import DataLoader, Subset, random_split
from torchvision import datasets, transforms
from torchvision.transforms import v2 as T


class Dataset:
  def __init__(self, fpath):
    self.data_path = fpath
    self.dataset = datasets.ImageFolder(root=fpath)

    # Training image transforms (70% training)
    self.train_transform = transforms.Compose(
      [
        transforms.Resize(image_size),  # Resize all images to 150×15
        transforms.RandomHorizontalFlip(),  # Apply basic data augmen
        transforms.RandomRotation(
          15
        ),  # Apply basic data augmentation (rotation)
        transforms.ToTensor(),  # Normalize pixel values to [0, 1] ra
        transforms.RandomApply(
          [T.GaussianNoise(mean=0.0, sigma=0.20, clip=True)],
          p=0.5,  # Apply max-noise of 0.20 with 50% probability - fr
        ),
      ]
    )

    # Validation image transforms (15% validation)
    self.validation_transform = transforms.Compose(
      [
        transforms.Resize(image_size),
        transforms.ToTensor(),
      ]
    )
```

```
    # Test image transforms (15% val)
    self.test_transform = self.validation_transform

  @property
  def data_size(self) -> int:
    return len(self.dataset)

  @property
  def train_size(self) -> int:
    return int(0.7 * self.data_size)

  @property
  def validation_size(self) -> int:
    return int(0.15 * self.data_size)

  @property
  def test_size(self) -> int:
    return int(self.data_size - self.train_size - self.validation_siz

  def preprocess(self):
    if not self.dataset:
      raise Exception("Dataset must be loaded before preprocessing.")

    train_start, validation_start, test_start = random_split(
      self.dataset, [self.train_size, self.validation_size, self.test_
    )

    self.train_data = Subset(
      datasets.ImageFolder(self.data_path, transform=self.train_trans
      train_start.indices,
    )
    self.val_data = Subset(
      datasets.ImageFolder(self.data_path, transform=self.validation_
      validation_start.indices,
    )
    self.test_data = Subset(
      datasets.ImageFolder(self.data_path, transform=self.test_transf
      test_start.indices,
    )

  def show_samples(self, train_data=False):
    cats = [
      i for i, (path, _) in enumerate(self.dataset.samples) if "Cat" :
    ]
    dogs = [
      i for i, (path, _) in enumerate(self.dataset.samples) if "Dog" :
    ]

    sample_cats = random.sample(cats, min(10, len(cats)))
    sample_dogs = random.sample(dogs, min(10, len(dogs)))
```

```
    sample_dogs = random.sample(dogs, min(10, len(dogs)))

    title = "Augmented Images" if train_data else "Original Images"
    fig, axes = plt.subplots(4, 5, figsize=(15, 12))
    fig.suptitle(title, fontsize=16)

    for idx, ax in enumerate(axes.flat):
      img_idx = sample_cats[idx % 10] if idx < 10 else sample_dogs[id
      img, _ = self.dataset[img_idx]

      if train_data:
        img = self.train_transform(img).permute(1, 2, 0).numpy()

      ax.imshow(img)
      ax.set_title("Cat" if idx < 10 else "Dog")
      ax.axis("off")

    plt.tight_layout()
    plt.show()

  def augment(self, datatype: str):
    torch.manual_seed(42)

    if datatype == "train":
      return DataLoader(
        self.train_data, batch_size=32, shuffle=True, num_workers=os.
      )
    else:
      return DataLoader(
        self.val_data, batch_size=32, shuffle=True, num_workers=os.cp
      )

dataset = Dataset(dataset_path)

print("Dataset samples before processing.\n")
dataset.show_samples()

dataset.preprocess()

print("\nDataset samples after processing.\n")
dataset.show_samples()
```

```
Dataset samples before processing.
```

Dataset samples after processing.



Create Model

```
import torch.nn as nn
import torchinfo

class CatDogCNN(nn.Module):
  def __init__(self):
    super().__init__()

    self.conv_layer_1 = nn.Sequential(
      nn.Conv2d(3, 64, 3, padding=1),
      nn.ReLU(),
      nn.BatchNorm2d(64),
      nn.MaxPool2d(2),
    )

    self.conv_layer_2 = nn.Sequential(
      nn.Conv2d(64, 512, 3, padding=1),
      nn.ReLU(),
      nn.BatchNorm2d(512),
      nn.MaxPool2d(2),
    )

    self.conv_layer_3 = nn.Sequential(
      nn.Conv2d(512, 512, kernel_size=3, padding=1),
      nn.ReLU(),
      nn.BatchNorm2d(512),
      nn.MaxPool2d(2)
```

```
      nn.MaxPool2d(2),
    )

    self.adaptive_pool = nn.AdaptiveAvgPool2d((2, 2))

    self.classifier = nn.Sequential(
      nn.Flatten(), nn.Linear(in_features=512 * 2 * 2, out_features=2
    )

  def forward(self, x: torch.Tensor) -> torch.Tensor:
    x = self.conv_layer_1(x)
    x = self.conv_layer_2(x)
    x = self.conv_layer_3(x)
    x = self.conv_layer_3(x)
    x = self.conv_layer_3(x)
    x = self.conv_layer_3(x)
    x = self.adaptive_pool(x)
    x = self.classifier(x)
    return x

  def summary(self) -> None:
    torchinfo.summary(self, input_size=[1, 3, image_size[0], image_si

model = CatDogCNN()
model.summary()
model.to(device)
```

```
CatDogCNN(
  (conv_layer_1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): ReLU()
    (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (conv_layer_2): Sequential(
    (0): Conv2d(64, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (conv_layer_3): Sequential(
    (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (adaptive_pool): AdaptiveAvgPool2d(output_size=(2, 2))
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=2048, out_features=2, bias=True)
  )
)
```

Train Model

```
from timeit import default_timer as timer
from typing import Any, Literal, overload

from tqdm import tqdm

train_dataloader_augmented = dataset.augment("train")
val_dataloader_augmented = dataset.augment("val")

loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model.parameters(), lr=1e-3)

def train() -> dict[str, list[Any]]:
  torch.manual_seed(42)
  torch.cuda.manual_seed(42)
  os.makedirs(conf["models_dir"], exist_ok=True)

  start_time = timer()

  model_results = _train(epochs=25)

  end_time = timer()
  print(f"Total training time: {end_time - start_time:.3f} seconds")

  return model_results


def _train(epochs: int = 25) -> dict[str, list[Any]]:
  results = {"train_loss": [], "train_acc": [], "val_loss": [], "val_

  for epoch in tqdm(range(epochs)):
    train_loss, train_acc = _train_step()

    val_loss, val_acc = _val_step(return_preds=False)

    tqdm.write(
      f"Epoch: {epoch + 1} | "
      f"train_loss: {train_loss:.4f} | "
```

```
      f"train_acc: {train_acc:.4f} | "
      f"val_loss: {val_loss:.4f} | "
      f"val_acc: {val_acc:.4f}"
    )

    results["train_loss"].append(train_loss)
    results["train_acc"].append(train_acc)
    results["val_loss"].append(val_loss)
    results["val_acc"].append(val_acc)

    torch.save(
      model.state_dict(), os.path.join(conf["models_dir"], f"cat_dog_
    )

  return results


def _train_step() -> tuple[float, float]:
  model.train()

  train_loss, train_acc = 0, 0
  for _, (X, y) in enumerate(train_dataloader_augmented):
    X, y = X.to(device), y.to(device)

    # 1. Forward pass
    y_pred = model(X)

    # 2. Calculate  and accumulate loss
    loss = loss_fn(y_pred, y)
    train_loss += loss.item()

    # 3. Optimizer zero grad
    optimizer.zero_grad()

    # 4. Loss backward
    loss.backward()

    # 5. Optimizer step
    optimizer.step()

    y_pred_class = torch.argmax(torch.softmax(y_pred, dim=1), dim=1)
    train_acc += (y_pred_class == y).sum().item() / len(y_pred)

  train_loss = train_loss / len(train_dataloader_augmented)
  train_acc = train_acc / len(train_dataloader_augmented)
  return train_loss, train_acc


@overload
def _val_step(return_preds: Literal[False] = False) -> tuple[float, f
@overload
```

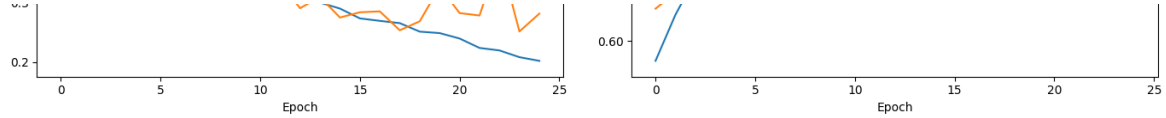
```
@overload
def _val_step(return_preds: Literal[True]) -> tuple[float, float, lis
def _val_step(return_preds: bool = False) -> tuple[float, float] | tu
  model.eval()
  val_loss, val_acc = 0, 0
  all_preds, all_labels = [], []

  with torch.inference_mode():
    for _, (X, y) in enumerate(val_dataloader_augmented):
      X, y = X.to(device), y.to(device)
      val_pred_logits = model(X)
      loss = loss_fn(val_pred_logits, y)
      val_loss += loss.item()
      val_pred_labels = val_pred_logits.argmax(dim=1)
      val_acc += (val_pred_labels == y).sum().item() / len(val_pred_l

      if return_preds:
        all_preds.extend(val_pred_labels.cpu().numpy())
        all_labels.extend(y.cpu().numpy())

  val_loss = val_loss / len(val_dataloader_augmented)
  val_acc = val_acc / len(val_dataloader_augmented)

  if return_preds:
    return val_loss, val_acc, all_preds, all_labels
  return val_loss, val_acc


res = train()
```

```
  4%|▍         | 1/25 [01:25<34:08, 85.35s/it]Epoch: 1 | train_loss: 0
  8%|▊         | 2/25 [02:19<25:42, 67.05s/it]Epoch: 2 | train_loss: 0
 12%|█▏        | 3/25 [03:11<22:03, 60.16s/it]Epoch: 3 | train_loss: 0
 16%|█▌        | 4/25 [04:04<20:04, 57.34s/it]Epoch: 4 | train_loss: 0
 20%|██        | 5/25 [04:56<18:28, 55.44s/it]Epoch: 5 | train_loss: 0
 24%|██▍       | 6/25 [05:50<17:22, 54.84s/it]Epoch: 6 | train_loss: 0
 28%|██▊       | 7/25 [06:42<16:13, 54.10s/it]Epoch: 7 | train_loss: 0
 32%|███▏      | 8/25 [07:34<15:08, 53.43s/it]Epoch: 8 | train_loss: 0
 36%|███▌      | 9/25 [08:27<14:09, 53.10s/it]Epoch: 9 | train_loss: 0
 40%|████      | 10/25 [09:21<13:21, 53.43s/it]Epoch: 10 | train_loss:
 44%|████▍     | 11/25 [10:14<12:27, 53.42s/it]Epoch: 11 | train_loss:
 48%|████▊     | 12/25 [11:07<11:29, 53.06s/it]Epoch: 12 | train_loss:
 52%|█████▏    | 13/25 [11:59<10:33, 52.79s/it]Epoch: 13 | train_loss:
 56%|█████▌    | 14/25 [12:52<09:41, 52.84s/it]Epoch: 14 | train_loss:
 60%|██████    | 15/25 [13:51<09:07, 54.75s/it]Epoch: 15 | train_loss:
 64%|██████▍   | 16/25 [14:44<08:09, 54.39s/it]Epoch: 16 | train_loss:
 68%|██████▊   | 17/25 [15:37<07:11, 53.94s/it]Epoch: 17 | train_loss:
 72%|███████▏  | 18/25 [16:30<06:14, 53.51s/it]Epoch: 18 | train_loss:
 76%|███████▌  | 19/25 [17:36<05:43, 57.24s/it]Epoch: 19 | train_loss:
 80%|████████  | 20/25 [18:46<05:05, 61.04s/it]Epoch: 20 | train_loss:
 84%|████████▍ | 21/25 [19:54<04:13, 63.35s/it]Epoch: 21 | train_loss:
 88%|████████▊ | 22/25 [20:56<03:08, 62.96s/it]Epoch: 22 | train_loss:
 92%|█████████▏| 23/25 [21:51<02:00, 60.36s/it]Epoch: 23 | train_loss:
```

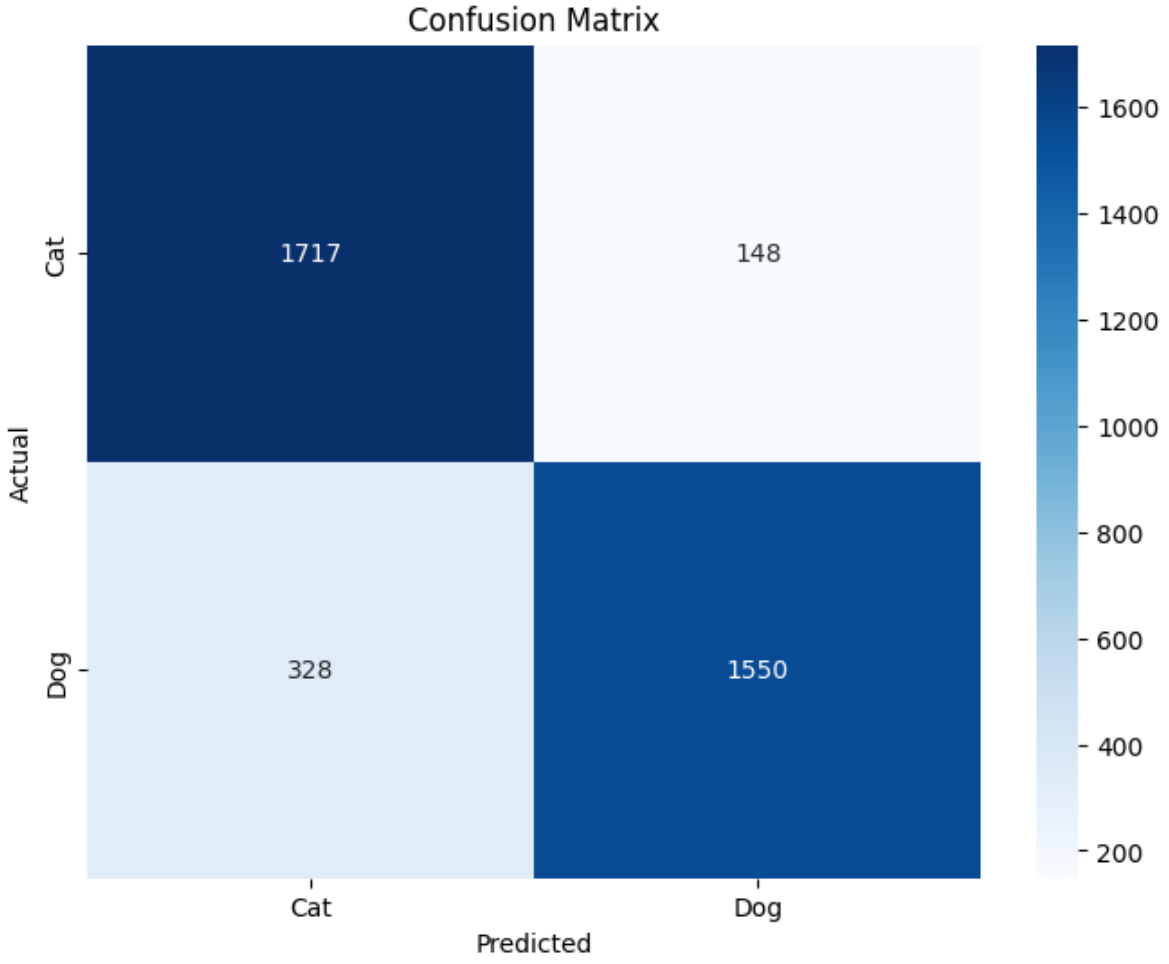
```
 96%|[redacted]| 24/25 [22:43<00:57, 57.84s/it]Epoch: 24 | train_loss:
100%|[redacted]| 25/25 [23:35<00:00, 56.61s/it]Epoch: 25 | train_loss:
Total training time: 1415.255 seconds
```

Visualizations

```
from sklearn.metrics import confusion_matrix
import seaborn as sns

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))
ax1.plot(res['train_loss'], label='Train Loss')
ax1.plot(res['val_loss'], label='Val Loss')
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Loss')
ax1.set_title('Training and Validation Loss')
ax1.legend()

ax2.plot(res['train_acc'], label='Train Accuracy')
ax2.plot(res['val_acc'], label='Val Accuracy')
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Accuracy')
ax2.set_title('Training and Validation Accuracy')
ax2.legend()
plt.tight_layout()
plt.show()

model.eval()
_, val_acc, val_preds, val_labels = _val_step(return_preds=True)

print(f'\nFinal Validation Accuracy: {val_acc:.4f}')


cm = confusion_matrix(val_labels, val_preds)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Cat'
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```




```
Final Validation Accuracy: 0.8728
```



Model Inference - Upload Image

```
#@title Model Inference - Upload Image

model.eval()

transform = transforms.Compose(
  [
    transforms.Resize(image_size),
    transforms.ToTensor(),
  ]
)

uploaded = files.upload()

image_path = list(uploaded.keys())[0]
image = Image.open(image_path).convert("RGB")
image_tensor = transform(image)
image_tensor = image_tensor.unsqueeze(0).to(device)

with torch.inference_mode():
  logits = model(image_tensor)
  probs = torch.softmax(logits, dim=1)
  predicted_class = torch.argmax(probs, dim=1).item()

class_names = ["Cat", "Dog"]

label = class_names[predicted_class]

plt.figure(figsize=(8, 6))
plt.imshow(image)
plt.axis('off')
plt.title(f'Prediction: {class_names[predicted_class]} ({probs[0, predi
          fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()
```

Browse... b54472aacb5d6ee99f29c6170fa6725e.JPG

b54472aacb5d6ee99f29c6170fa6725e.JPG(image/jpeg) - 279456 bytes, last modified: n/a - 100% done

Saving b54472aacb5d6ee99f29c6170fa6725e.JPG to b54472aacb5d6ee99f29c61