khlmwgbbd

November 18, 2025

[1]:
```
from google.colab import files
files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving kaggle.json to kaggle.json
```

[1]:
```
{'kaggle.json':
b'{"username":"sanatkulkarni","key":"fb3677e7e952b220946406f6e9ac346b"}'}
```

[2]:
```
!mkdir ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

[3]:
```
!pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.12/dist-packages
(1.7.4.5)
Requirement already satisfied: bleach in /usr/local/lib/python3.12/dist-packages
(from kaggle) (6.3.0)
Requirement already satisfied: certifi>=14.05.14 in
/usr/local/lib/python3.12/dist-packages (from kaggle) (2025.10.5)
Requirement already satisfied: charset-normalizer in
/usr/local/lib/python3.12/dist-packages (from kaggle) (3.4.4)
Requirement already satisfied: idna in /usr/local/lib/python3.12/dist-packages
(from kaggle) (3.11)
Requirement already satisfied: protobuf in /usr/local/lib/python3.12/dist-
packages (from kaggle) (5.29.5)
Requirement already satisfied: python-dateutil>=2.5.3 in
/usr/local/lib/python3.12/dist-packages (from kaggle) (2.9.0.post0)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.12/dist-
packages (from kaggle) (8.0.4)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-
packages (from kaggle) (2.32.4)
Requirement already satisfied: setuptools>=21.0.0 in
/usr/local/lib/python3.12/dist-packages (from kaggle) (75.2.0)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.12/dist-
packages (from kaggle) (1.17.0)
Requirement already satisfied: text-unidecode in /usr/local/lib/python3.12/dist-
```

```
packages (from kaggle) (1.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages
(from kaggle) (4.67.1)
Requirement already satisfied: urllib3>=1.15.1 in
/usr/local/lib/python3.12/dist-packages (from kaggle) (2.5.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.12/dist-
packages (from kaggle) (0.5.1)
```

[4]:
```
!kaggle datasets download -d banuprasadb/wildlife-dataset
```

```
Dataset URL: https://www.kaggle.com/datasets/banuprasadb/wildlife-dataset
License(s): CC-BY-SA-4.0
Downloading wildlife-dataset.zip to /content
100% 5.31G/5.31G [01:31<00:00, 179MB/s]
100% 5.31G/5.31G [01:31<00:00, 62.5MB/s]
```

[5]:
```
import zipfile

zip_path = "/content/wildlife-dataset.zip"

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall("wildlife-dataset")
```

[6]:
```
DATASET_ROOT = "/content/wildlife-dataset"
```

[7]:
```
import os
import cv2
import glob
from tqdm import tqdm

input_images = os.path.join(DATASET_ROOT, 'train/images')
input_labels = os.path.join(DATASET_ROOT, 'train/labels')

output_dir = '/content/processed_data_resnet/train'

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

print(f"Processing images from: {input_images}")
print(f"Saving crops to: {output_dir}")

image_files = glob.glob(os.path.join(input_images, '*.jpg'))

if not image_files:
    print("ERROR: No images found! Check your DATASET_ROOT path.")
else:
    count = 0
```

```
    for img_path in tqdm(image_files):

        filename = os.path.basename(img_path)
        file_id = os.path.splitext(filename)[0]
        label_path = os.path.join(input_labels, file_id + '.txt')

        if not os.path.exists(label_path): continue

        img = cv2.imread(img_path)
        if img is None: continue
        h_img, w_img, _ = img.shape

        with open(label_path, 'r') as f:
            lines = f.readlines()
            for i, line in enumerate(lines):
                parts = line.strip().split()
                class_id = parts[0]

                class_folder = os.path.join(output_dir, f"Animal_{class_id}")
                os.makedirs(class_folder, exist_ok=True)

                x_c, y_c, bw, bh = map(float, parts[1:])

                x1 = int((x_c - bw/2) * w_img)
                y1 = int((y_c - bh/2) * h_img)
                x2 = int((x_c + bw/2) * w_img)
                y2 = int((y_c + bh/2) * h_img)

                x1, y1 = max(0, x1), max(0, y1)
                x2, y2 = min(w_img, x2), min(h_img, y2)

                crop = img[y1:y2, x1:x2]

                if crop.size > 0:
                    save_name = f"{file_id}_crop_{i}.jpg"
                    cv2.imwrite(os.path.join(class_folder, save_name), crop)
                    count += 1

    print(f"Preprocessing Complete. {count} animal images cropped and saved.")
```

```
Processing images from: /content/wildlife-dataset/train/images
Saving crops to: /content/processed_data_resnet/train
100%|      | 13605/13605 [02:28<00:00, 91.70it/s]
Preprocessing Complete. 15791 animal images cropped and saved.
```

[8]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, models, transforms
from torch.utils.data import DataLoader, random_split

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

data_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

data_dir = '/content/processed_data_resnet/train'
full_dataset = datasets.ImageFolder(data_dir, transform=data_transform)

train_size = int(0.8 * len(full_dataset))
val_size = len(full_dataset) - train_size
train_dataset, val_dataset = random_split(full_dataset, [train_size, val_size])

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)

class_names = full_dataset.classes
print(f"Classes Detected: {class_names}")

model = models.resnet18(weights='IMAGENET1K_V1')

for param in model.parameters():
    param.requires_grad = False

num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, len(class_names))
model = model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=0.001)

epochs = 10
print("Starting Training...")

for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    correct = 0
```

```
    total = 0

    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    epoch_acc = 100 * correct / total
    print(f"Epoch {epoch+1}/{epochs} | Loss: {running_loss/len(train_loader):.
 ↪4f} | Acc: {epoch_acc:.2f}%")

print("Training Finished.")

torch.save(model.state_dict(), 'resnet18_wildlife.pth')
```

```
Using device: cuda
Classes Detected: ['Animal_0', 'Animal_1', 'Animal_10', 'Animal_11',
'Animal_12', 'Animal_13', 'Animal_14', 'Animal_15', 'Animal_16', 'Animal_17',
'Animal_18', 'Animal_19', 'Animal_2', 'Animal_20', 'Animal_21', 'Animal_22',
'Animal_23', 'Animal_24', 'Animal_25', 'Animal_26', 'Animal_27', 'Animal_28',
'Animal_29', 'Animal_3', 'Animal_30', 'Animal_31', 'Animal_32', 'Animal_33',
'Animal_34', 'Animal_35', 'Animal_36', 'Animal_37', 'Animal_38', 'Animal_39',
'Animal_4', 'Animal_40', 'Animal_41', 'Animal_42', 'Animal_43', 'Animal_44',
'Animal_45', 'Animal_46', 'Animal_47', 'Animal_48', 'Animal_49', 'Animal_5',
'Animal_50', 'Animal_51', 'Animal_52', 'Animal_53', 'Animal_6', 'Animal_7',
'Animal_8', 'Animal_9']
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to
/root/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth

100%|      | 44.7M/44.7M [00:00<00:00, 173MB/s]

Starting Training...
Epoch 1/10 | Loss: 1.1326 | Acc: 74.26%
Epoch 2/10 | Loss: 0.4489 | Acc: 88.08%
Epoch 3/10 | Loss: 0.3491 | Acc: 90.39%
Epoch 4/10 | Loss: 0.2935 | Acc: 91.59%
Epoch 5/10 | Loss: 0.2708 | Acc: 91.99%
Epoch 6/10 | Loss: 0.2416 | Acc: 92.88%
Epoch 7/10 | Loss: 0.2244 | Acc: 93.11%
```

```
Epoch 8/10 | Loss: 0.2089 | Acc: 93.73%
Epoch 9/10 | Loss: 0.1935 | Acc: 94.23%
Epoch 10/10 | Loss: 0.1801 | Acc: 94.67%
Training Finished.
```

1 Testing

[9]:
```python
import os
import cv2
import glob
from tqdm import tqdm

test_images_path = os.path.join(DATASET_ROOT, 'test/images')
test_labels_path = os.path.join(DATASET_ROOT, 'test/labels')
output_test_dir = '/content/processed_data_resnet/test'

if not os.path.exists(output_test_dir):
    os.makedirs(output_test_dir)

print(f"Preprocessing TEST data from: {test_images_path}")

image_files = glob.glob(os.path.join(test_images_path, '*.jpg'))

count = 0
for img_path in tqdm(image_files):
    filename = os.path.basename(img_path)
    file_id = os.path.splitext(filename)[0]
    label_path = os.path.join(test_labels_path, file_id + '.txt')

    if not os.path.exists(label_path): continue

    img = cv2.imread(img_path)
    if img is None: continue
    h_img, w_img, _ = img.shape

    with open(label_path, 'r') as f:
        lines = f.readlines()
        for i, line in enumerate(lines):
            parts = line.strip().split()
            class_id = parts[0]

            class_folder = os.path.join(output_test_dir, f"Animal_{class_id}")
            os.makedirs(class_folder, exist_ok=True)

            x_c, y_c, bw, bh = map(float, parts[1:])
            x1 = int((x_c - bw/2) * w_img)
            y1 = int((y_c - bh/2) * h_img)
```

```
            x2 = int((x_c + bw/2) * w_img)
            y2 = int((y_c + bh/2) * h_img)

            x1, y1 = max(0, x1), max(0, y1)
            x2, y2 = min(w_img, x2), min(h_img, y2)

            crop = img[y1:y2, x1:x2]
            if crop.size > 0:
                save_name = f"{file_id}_crop_{i}.jpg"
                cv2.imwrite(os.path.join(class_folder, save_name), crop)
                count += 1

print(f"Test Set Ready: {count} images cropped.")
```

```
Preprocessing TEST data from: /content/wildlife-dataset/test/images
100%|      | 1717/1717 [00:19<00:00, 88.28it/s]
Test Set Ready: 1992 images cropped.
```

[10]:
```
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

test_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

test_dataset = datasets.ImageFolder(output_test_dir, transform=test_transform)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)

test_class_names = test_dataset.classes
print(f"Test Classes: {test_class_names}")

all_preds = []
all_labels = []
all_probs = []

model.eval()
print("Running Inference on Test Set...")

with torch.no_grad():
    for inputs, labels in test_loader:
        inputs = inputs.to(device)
        labels = labels.to(device)
```

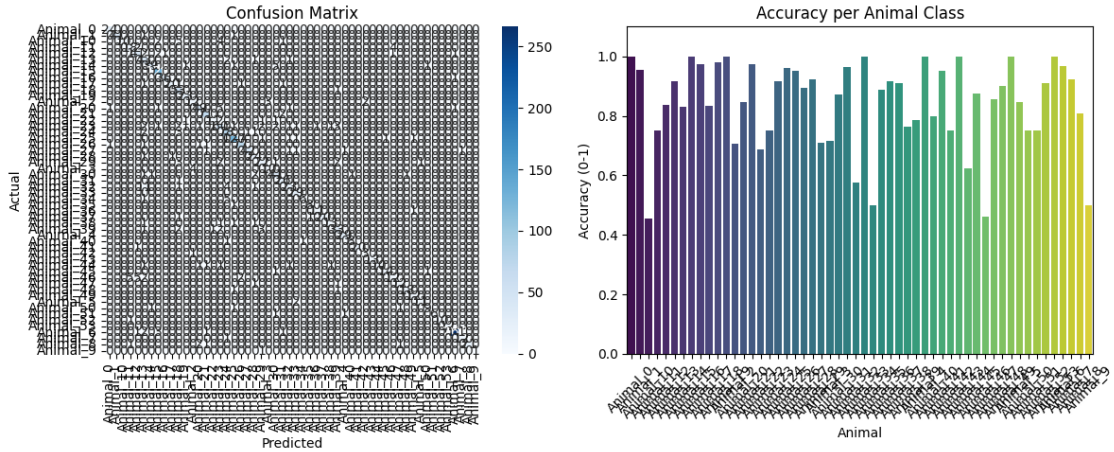
```
        outputs = model(inputs)
        probs = torch.nn.functional.softmax(outputs, dim=1)
        _, preds = torch.max(outputs, 1)

        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())
        all_probs.extend(probs.cpu().numpy())

print("Inference Complete.")
```

```
Test Classes: ['Animal_0', 'Animal_1', 'Animal_10', 'Animal_11', 'Animal_12',
'Animal_13', 'Animal_14', 'Animal_15', 'Animal_16', 'Animal_17', 'Animal_18',
'Animal_19', 'Animal_2', 'Animal_20', 'Animal_21', 'Animal_22', 'Animal_23',
'Animal_24', 'Animal_25', 'Animal_26', 'Animal_27', 'Animal_28', 'Animal_29',
'Animal_3', 'Animal_30', 'Animal_31', 'Animal_32', 'Animal_33', 'Animal_34',
'Animal_35', 'Animal_36', 'Animal_37', 'Animal_38', 'Animal_39', 'Animal_4',
'Animal_40', 'Animal_41', 'Animal_42', 'Animal_43', 'Animal_44', 'Animal_45',
'Animal_46', 'Animal_47', 'Animal_48', 'Animal_49', 'Animal_5', 'Animal_50',
'Animal_51', 'Animal_52', 'Animal_53', 'Animal_6', 'Animal_7', 'Animal_8',
'Animal_9']
Running Inference on Test Set…
Inference Complete.
```

[11]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
import pandas as pd

cm = confusion_matrix(all_labels, all_preds)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=test_class_names, yticklabels=test_class_names)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')

class_accuracies = cm.diagonal() / cm.sum(axis=1)
class_acc_df = pd.DataFrame({
    'Animal': test_class_names,
    'Accuracy': class_accuracies
})

plt.subplot(1, 2, 2)
```

```
sns.barplot(x='Animal', y='Accuracy', data=class_acc_df, palette='viridis')
plt.ylim(0, 1.1)
plt.title('Accuracy per Animal Class')
plt.ylabel('Accuracy (0-1)')
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()

print("\n--- Detailed Classification Report ---")
print(classification_report(all_labels, all_preds,␣
 ↪target_names=test_class_names))
```

```
/tmp/ipython-input-1151455535.py:25: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='Animal', y='Accuracy', data=class_acc_df, palette='viridis')
```

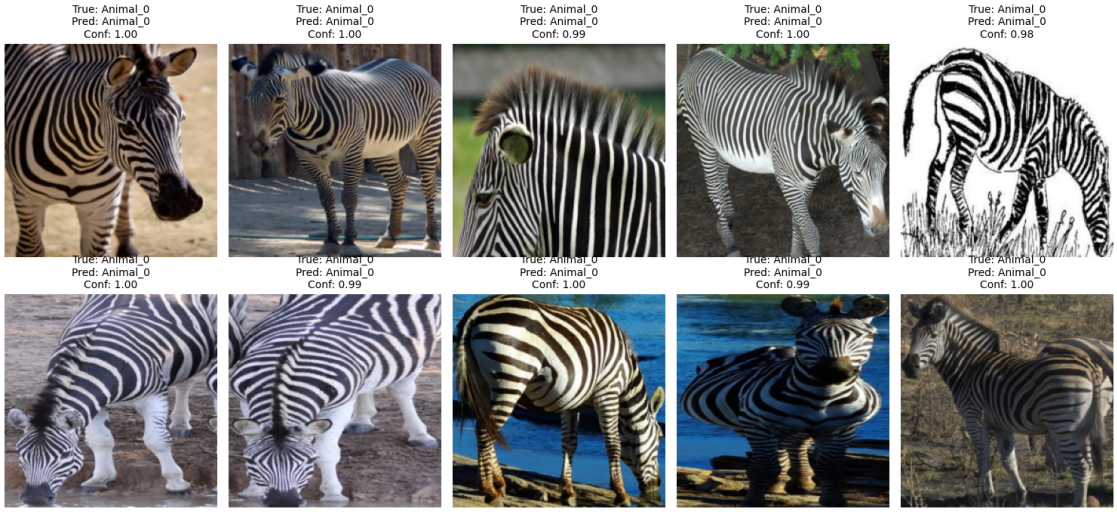



```
--- Detailed Classification Report ---
              precision    recall  f1-score   support

    Animal_0       0.92      1.00      0.96        24
    Animal_1       1.00      0.95      0.98        43
   Animal_10       1.00      0.45      0.62        22
   Animal_11       0.69      0.75      0.72        24
   Animal_12       0.82      0.84      0.83        49
   Animal_13       0.59      0.91      0.72        47
   Animal_14       0.95      0.83      0.89        47
```

Animal_15	0.93	1.00	0.97	128
Animal_16	0.97	0.97	0.97	37
Animal_17	0.95	0.83	0.89	24
Animal_18	0.74	0.98	0.84	54
Animal_19	0.88	1.00	0.94	23
Animal_2	0.71	0.71	0.71	17
Animal_20	0.91	0.84	0.88	58
Animal_21	0.93	0.97	0.95	73
Animal_22	0.85	0.69	0.76	16
Animal_23	0.78	0.75	0.76	80
Animal_24	0.86	0.92	0.89	85
Animal_25	0.95	0.96	0.95	132
Animal_26	0.96	0.95	0.96	81
Animal_27	0.98	0.89	0.93	47
Animal_28	0.77	0.92	0.84	26
Animal_29	0.73	0.71	0.72	31
Animal_3	0.77	0.71	0.74	14
Animal_30	0.79	0.87	0.83	47
Animal_31	0.84	0.96	0.90	27
Animal_32	0.79	0.57	0.67	40
Animal_33	0.85	1.00	0.92	29
Animal_34	0.86	0.50	0.63	12
Animal_35	0.89	0.89	0.89	9
Animal_36	0.92	0.92	0.92	12
Animal_37	1.00	0.91	0.95	22
Animal_38	0.87	0.76	0.81	17
Animal_39	0.80	0.79	0.80	42
Animal_4	0.94	1.00	0.97	29
Animal_40	0.89	0.80	0.84	10
Animal_41	1.00	0.95	0.98	21
Animal_42	0.60	0.75	0.67	4
Animal_43	1.00	1.00	1.00	13
Animal_44	1.00	0.62	0.77	16
Animal_45	0.93	0.88	0.90	16
Animal_46	0.71	0.46	0.56	26
Animal_47	0.86	0.86	0.86	14
Animal_48	0.95	0.90	0.92	20
Animal_49	0.86	1.00	0.92	12
Animal_5	0.92	0.85	0.88	13
Animal_50	0.82	0.75	0.78	12
Animal_51	1.00	0.75	0.86	8
Animal_52	1.00	0.91	0.95	11
Animal_53	1.00	1.00	1.00	12
Animal_6	0.99	0.97	0.98	275
Animal_7	0.86	0.92	0.89	13
Animal_8	1.00	0.81	0.89	26
Animal_9	1.00	0.50	0.67	2

```
    accuracy                           0.89      1992
   macro avg       0.88      0.84      0.85      1992
weighted avg       0.89      0.89      0.89      1992
```

[12]:
```python
def imshow_tensor(inp, title=None, ax=None):
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    if ax:
        ax.imshow(inp)
        if title: ax.set_title(title, fontsize=10)
        ax.axis('off')

inputs, labels = next(iter(test_loader))
inputs = inputs.to(device)
outputs = model(inputs)
_, preds = torch.max(outputs, 1)
probs = torch.nn.functional.softmax(outputs, dim=1)
max_probs, _ = torch.max(probs, 1)

fig, axes = plt.subplots(2, 5, figsize=(15, 7))
axes = axes.flatten()

for i in range(10):
    if i >= len(inputs): break

    img_t = inputs[i].cpu()
    true_label = test_class_names[labels[i]]
    pred_label = test_class_names[preds[i]]
    conf = max_probs[i].item()

    color = 'green' if true_label == pred_label else 'red'
    title = f"True: {true_label}\nPred: {pred_label}\nConf: {conf:.2f}"

    imshow_tensor(img_t, title=title, ax=axes[i])

    for spine in axes[i].spines.values():
        spine.set_edgecolor(color)
        spine.set_linewidth(2)

plt.tight_layout()
plt.show()
```


True: Animal_0
Pred: Animal_0
Conf: 1.00
True: Animal_0
Pred: Animal_0
Conf: 1.00
True: Animal_0
Pred: Animal_0
Conf: 0.99
True: Animal_0
Pred: Animal_0
Conf: 1.00
True: Animal_0
Pred: Animal_0
Conf: 0.98
True: Animal_0
Pred: Animal_0
Conf: 1.00
True: Animal_0
Pred: Animal_0
Conf: 0.99
True: Animal_0
Pred: Animal_0
Conf: 1.00
True: Animal_0
Pred: Animal_0
Conf: 0.99
True: Animal_0
Pred: Animal_0
Conf: 1.00