

# Project I | Deep Learning: Image Classification with CNN

## Introduction

This project focuses on implementing a deep learning model using PyTorch for image classification. The model is based on a ResNet-18 architecture, fine-tuned for a custom dataset.

## Tools and Technologies Used

**Programming Language:** Python

**Machine Learning Libraries:** PyTorch, Torchvision


**Data Processing:** PIL, NumPy, Matplotlib, Seaborn

**Performance Evaluation:** Confusion Matrix, Classification Report

**Execution Environment:** Google Colab (with GPU support)

## Methodology

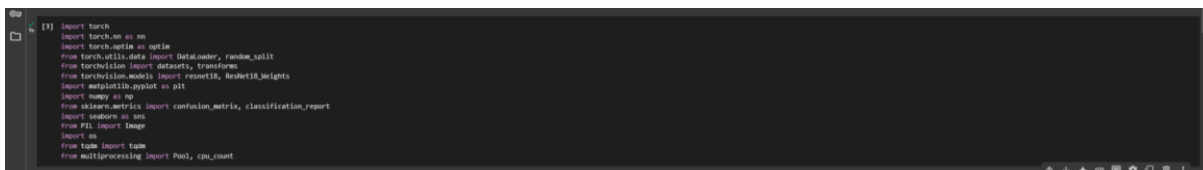
### Mounting Google Drive



```
[2]: from google.colab import drive
drive.mount('/content/drive')

[4]: Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

### importing Required Libraries



```
[1]: import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torchvision import datasets, transforms
from torchvision.models import resnet18, ResNet18_Weights
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
from PIL import Image
import os
from tqdm import tqdm
from multiprocessing import Pool, cpu_count
```

Iron hack

Triple pixels

## Configuring Device for Computation

```
torch.backends.cudnn.allow_tf32 = True
torch.backends.cudnn.allow_tf32 = True
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

## Filtering Valid Images

```
def is_valid_image(file_path):
    """Check if a file is a valid image."""
    try:
        with Image.open(file_path) as img:
            img.verify()
        return True
    except Exception:
        return False

def filter_valid_images(file_paths):
    """Filter valid images using multiprocessing."""
    with Pool(cpu_count()) as pool:
        results = pool.map(is_valid_image, file_paths)
    return [file_path for file_path, is_valid in zip(file_paths, results) if is_valid]
```

## Data Preprocessing and Augmentation

```
def load_dataset(data_path, batch_size=64, num_workers=4):
    train_transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(10),
        transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.405, 0.406, 0.406], std=[0.229, 0.224, 0.229])
    ])
    val_transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.405, 0.406, 0.406], std=[0.229, 0.224, 0.229])
    ])
```

## Loading and Splitting the Dataset

```
full_dataset = datasets.ImageFolder(root_data_path, transform=train_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])
val_dataset.dataset.transform = val_transform
```

## Initializing Data Loaders

```
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=num_workers, pin_memory=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False, num_workers=num_workers, pin_memory=True)

return train_loader, val_loader
```

## data\_path

```
[ ] data_path = "/content/drive/My Drive/animal18/raw-img"
train_loader, val_loader = load_dataset(data_path)
class_names = train_loader.dataset.classes
visualize_images(train_loader, class_names)
```


WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-2.1007791..2.622571].

WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-2.0836544..2.622571].

WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-2.0836544..2.622571].

WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-2.0865286..2.64].

WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-2.8357141..2.2146587].



## Model Initialization and Modifications

```
transfer_model = resnet18(weights=ResNet18_Weights.DEFAULT)
transfer_model.fc = nn.Linear(transfer_model.fc.in_features, len(class_names))
transfer_model = transfer_model.to(device)
```

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, 152MB/s]

## Setting Loss Function and Optimizer

```
[ ] criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001, weight_decay=1e-5)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience=5, verbose=True)

<> /usr/local/lib/python3.11/dist-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to access the learning rate.
warnings.warn(
```

## Training the Model

```
best_val_loss = float('inf')
patience = 5
no_improvement = 0

def train(model, train_loader, val_loader, criterion, optimizer, epochs=15):
    global best_val_loss, no_improvement
    model.train()
    for epoch in range(epochs):
        running_loss, correct, total = 0.0, 0, 0
        with tqdm(train_loader, total=len(train_loader), desc=f'Epoch {epoch+1}/{epochs}', ncols=100) as pbar:
            for images, labels in pbar:
                images, labels = images.to(device), labels.to(device)
                optimizer.zero_grad()

                outputs = model(images)
                loss = criterion(outputs, labels)
                loss.backward()
                torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
                optimizer.step()

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

            pbar.set_postfix(loss=running_loss/len(train_loader), accuracy=100 * correct/total)

        val_loss, val_accuracy = validate(model, val_loader, criterion)
        print(f'Epoch {epoch+1}/{epochs} | Train Loss: {running_loss/len(train_loader):.4f} | Train Accuracy: {100 * correct/total:.2f}% | Val Loss: {val_loss:.4f} | Val Accuracy: {val_accuracy:.2f}%')

        if val_loss < best_val_loss:
            best_val_loss = val_loss
            no_improvement = 0
            torch.save(model.state_dict(), 'best_model.pth')
        else:
            no_improvement += 1
            if no_improvement == patience:
                print(f'Early stopping at epoch {epoch+1}')
                break

    scheduler.step(val_loss)
```

## Evaluating the Model

```
[ ] def validate(model, val_loader, criterion):
    model.eval()
    val_loss, correct, total = 0.0, 0, 0
    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    model.train()
    return val_loss / len(val_loader), 100 * correct / total

[ ] train(train_loader, val_loader, criterion, optimizer, epochs=15)

Epoch 1/15: 100% | 328/328 [19:17:00:00, 3.53s/it, accuracy=92.5, loss=0.252]
Epoch 1/15 | Train Loss: 0.2517 | Train Accuracy: 92.47% | Val Loss: 0.1444 | Val Accuracy: 96.63%
Epoch 2/15: 100% | 328/328 [00:44:00:00, 7.321t/s, accuracy=98.7, loss=0.0403]
Epoch 2/15 | Train Loss: 0.0403 | Train Accuracy: 98.71% | Val Loss: 0.1530 | Val Accuracy: 95.67%
Epoch 3/15: 100% | 328/328 [00:44:00:00, 7.331t/s, accuracy=99.6, loss=0.0147]
Epoch 3/15 | Train Loss: 0.0147 | Train Accuracy: 99.62% | Val Loss: 0.1500 | Val Accuracy: 96.30%
Epoch 4/15: 100% | 328/328 [00:44:00:00, 7.351t/s, accuracy=99.8, loss=0.00334]
Epoch 4/15 | Train Loss: 0.0083 | Train Accuracy: 99.79% | Val Loss: 0.1620 | Val Accuracy: 96.26%
Epoch 5/15: 100% | 328/328 [00:45:00:00, 7.281t/s, accuracy=99.8, loss=0.00014]
Epoch 5/15 | Train Loss: 0.0001 | Train Accuracy: 99.84% | Val Loss: 0.1760 | Val Accuracy: 95.90%
Epoch 6/15: 100% | 328/328 [00:45:00:00, 7.271t/s, accuracy=99.8, loss=0.00004]
Epoch 6/15 | Train Loss: 0.0000 | Train Accuracy: 99.83% | Val Loss: 0.2152 | Val Accuracy: 95.10%
Early stopping at epoch 6
```

This code evaluates the model's performance on vconfusion matrix and classification report (Precisianalyze classification accuracy, identify errors, and improve the model ba

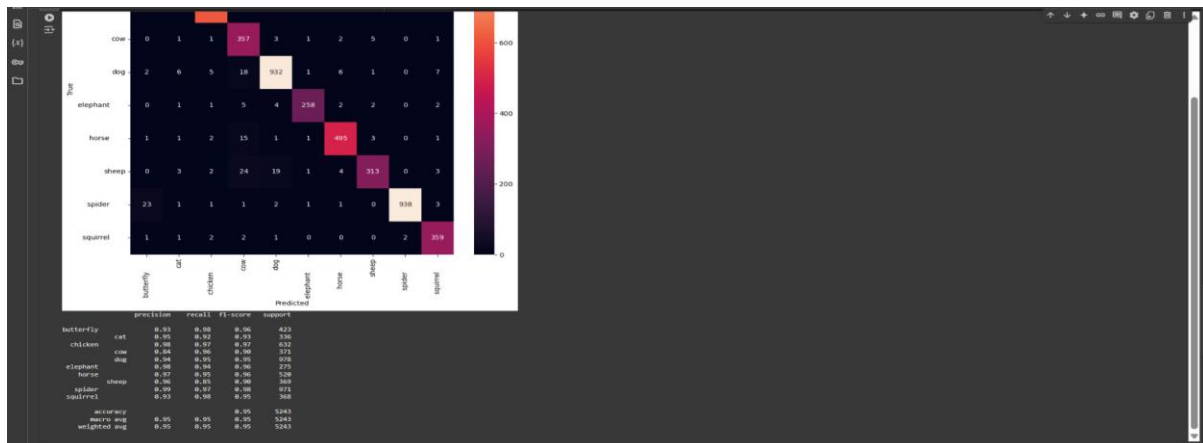
```
[ ] def evaluate(model, val_loader):
    model.eval()
    all_preds = []
    all_labels = []
    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, preds = torch.max(outputs, 1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    cm = confusion_matrix(all_labels, all_preds)
    plt.figure(figsize=(10, 10))
    sns.heatmap(cm, annot=True, fmt="d", xticklabels=class_names, yticklabels=class_names)
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.show()

    print(classification_report(all_labels, all_preds, target_names=class_names))

evaluate(transfer_model, val_loader)
```

## Results



	precision	recall	f1-score	support
butterfly	0.93	0.98	0.96	423
cat	0.95	0.92	0.93	336
chicken	0.98	0.97	0.97	632
cow	0.84	0.96	0.90	371
dog	0.94	0.95	0.95	978
elephant	0.98	0.94	0.96	275
horse	0.97	0.95	0.96	520
sheep	0.96	0.85	0.90	369
spider	0.99	0.97	0.98	971
squirrel	0.93	0.98	0.95	368
accuracy			0.95	5243
macro avg	0.95	0.95	0.95	5243
weighted avg	0.95	0.95	0.95	5243

This code classifies images into 10 animal categories using a pre-trained ResNet-18 model. It uploads an image, preprocesses it, passes it through the model, and displays predictions with probabilities for each category

Iron hack

Triple pixels

```
import torch
import torch.nn as nn
from torchvision import transforms
from PIL import Image
from pathlib import Path

model = torchvision.models.resnet18(pretrained=True)
model.load_state_dict(torch.load('best_model.pth'))
model.eval()

class_names = [
    "airplane", "bird", "cat", "deer", "dog", "elephant", "fish", "horse", "sheep",
    "truck", "train", "aircraft carrier", "motorcycle", "zebra", "giraffe"
]

def preprocess_image(image):
    transform = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
    return transform(image)

def predict_image_path(image_path):
    image = Image.open(image_path).convert('RGB')
    image_tensor = preprocess_image(image)

    with torch.no_grad():
        output = model(image_tensor)
        _, predicted = torch.max(output, dim=1)
        prediction = class_names[predicted.item()]

    return prediction

def main():
    # Load the model
    model.load_state_dict(torch.load('best_model.pth'))
    model.eval()

    # Load the image
    image_path = 'sample_image.jpg'
    image = Image.open(image_path)

    # Preprocess the image
    image_tensor = preprocess_image(image)

    # Predict the class
    with torch.no_grad():
        output = model(image_tensor)
        _, predicted = torch.max(output, dim=1)
        prediction = class_names[predicted.item()]

    # Print the prediction
    print(f'Predicted class: {prediction}')

if __name__ == '__main__':
    main()
```

## Results

Elephant:0.999

```
if __name__ == "__main__":
    main()

<ipython-input-14-4a3b1c55db77>:18: FutureWarning: You are using 'torch.load' with 'weights_only=False' (the current default value), which uses the default pickle module implicitly. It is possible to construct
malicious pickle payloads which could allow arbitrary code execution when using 'torch.load' with 'weights_only=False'. To avoid this warning, you should explicitly pass 'weights_only=True' to 'torch.load'. For more details, see
https://pytorch.org/docs/stable/generated/torch.load.html

e83cb00a2ef1053ed1584d05fb1d4e9fe777ead218ac104497f5c978a4efbcb0_640.jpg (image/jpeg) - 125352 bytes, last modified 6100% - 2025/3/ done
Saving e83cb00a2ef1053ed1584d05fb1d4e9fe777ead218ac104497f5c978a4efbcb0_640.jpg to e83cb00a2ef1053ed1584d05fb1d4e9fe777ead218ac104497f5c978a4efbcb0_640.jpg

Processing image: e83cb00a2ef1053ed1584d05fb1d4e9fe777ead218ac104497f5c978a4efbcb0_640.jpg
Predictions with probabilities:
butterfly: 0.0000
cat: 0.0000
chicken: 0.0000
cow: 0.0000
dog: 0.0000
elephant: 0.9999
horse: 0.0000
sheep: 0.0000
spider: 0.0000
squirrel: 0.0000
```

Sample used

Iron hack

Triple pixels



Horse:0.993

```
Untitled4.ipynb
File Edit View Insert Runtime Tools Help

for image_name, image_content in uploaded_files.items():
    print(f"\nProcessing image: {image_name}")

    with open(image_name, "wb") as f:
        f.write(image_content)

    predictions = predict_image(image_name)

    if "error" in predictions:
        print(f"Error processing {image_name}: {predictions['error']}")
    else:
        print("Predictions with probabilities:")
        for class_name, prob in predictions.items():
            print(f"({class_name}): {prob:.4f}")

if __name__ == "__main__":
    main()
```

Processing image: (3) قرد.jpg  
Predictions with probabilities:  
butterfly: 0.0001  
cat: 0.0000  
chicken: 0.0001  
cow: 0.0002  
dog: 0.0001  
elephant: 0.0000  
horse: 0.9993  
sheep: 0.0000  
spider: 0.0002  
squirrel: 0.0000

Sample used

