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

importing Required Libraries

Configuring Device for Computation

```
torch.backends.cuda.matmul.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_transforms = 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=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])
    val_transforms = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
    ])
```

Loading and Splitting the Dataset

```
    full_dataset = datasets.ImageFolder(root=data_path, transform=train_transforms)

    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_transforms
```

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/animals10/raw-img"
    train_loader, val_loader = load_dataset(data_path)

    class_names = train_loader.dataset.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 [-2.1007793..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 [-2.0665298..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 [-2.0357141..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, 162MB/s]
```

Setting Loss Function and Optimizer

```
[ ] criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(transfer_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, 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(transfer_model, 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: 96.03%
Epoch 2/15: 100%|██████████| 328/328 [00:44<00:00,  7.32it/s, accuracy=98.7, loss=0.0483]
Epoch 2/15 | Train Loss: 0.0483 | Train Accuracy: 98.71% | Val Loss: 0.1530 | Val Accuracy: 95.67%
Epoch 3/15: 100%|██████████| 328/328 [00:44<00:00,  7.33it/s, accuracy=99.6, loss=0.0147]
Epoch 3/15 | Train Loss: 0.0147 | Train Accuracy: 99.62% | Val Loss: 0.1500 | Val Accuracy: 96.36%
Epoch 4/15: 100%|██████████| 328/328 [00:44<00:00,  7.36it/s, accuracy=99.8, loss=0.00834]
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,  7.28it/s, accuracy=99.8, loss=0.00614]
Epoch 5/15 | Train Loss: 0.0061 | Train Accuracy: 99.84% | Val Loss: 0.1766 | Val Accuracy: 95.90%
Epoch 6/15: 100%|██████████| 328/328 [00:45<00:00,  7.27it/s, accuracy=99.8, loss=0.00604]
Epoch 6/15 | Train Loss: 0.0060 | Train Accuracy: 99.85% | 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

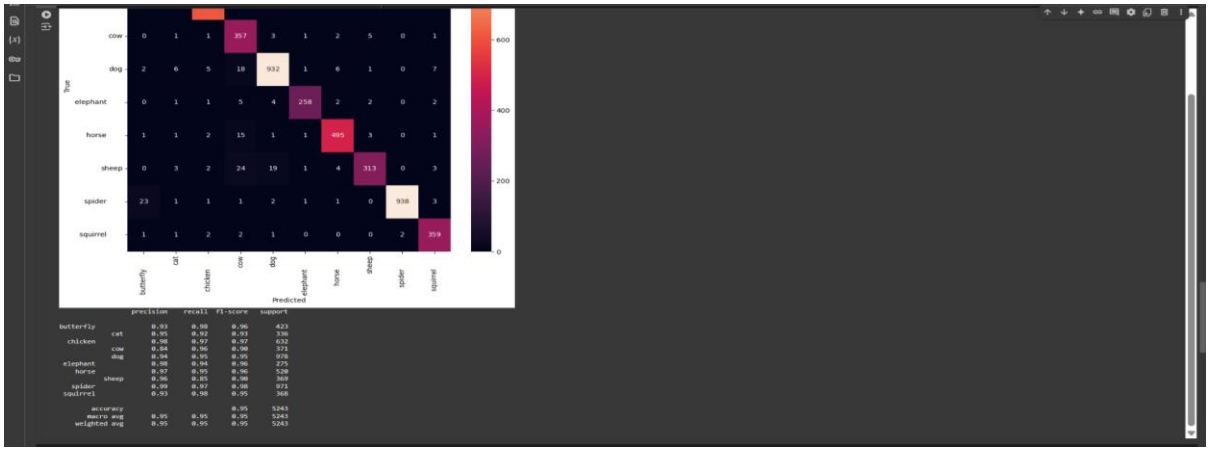
```python
[ ] def evaluate(model, val_loader):
        model.eval()
        all_preds = []
        all_labels = []
        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                _, 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)
```

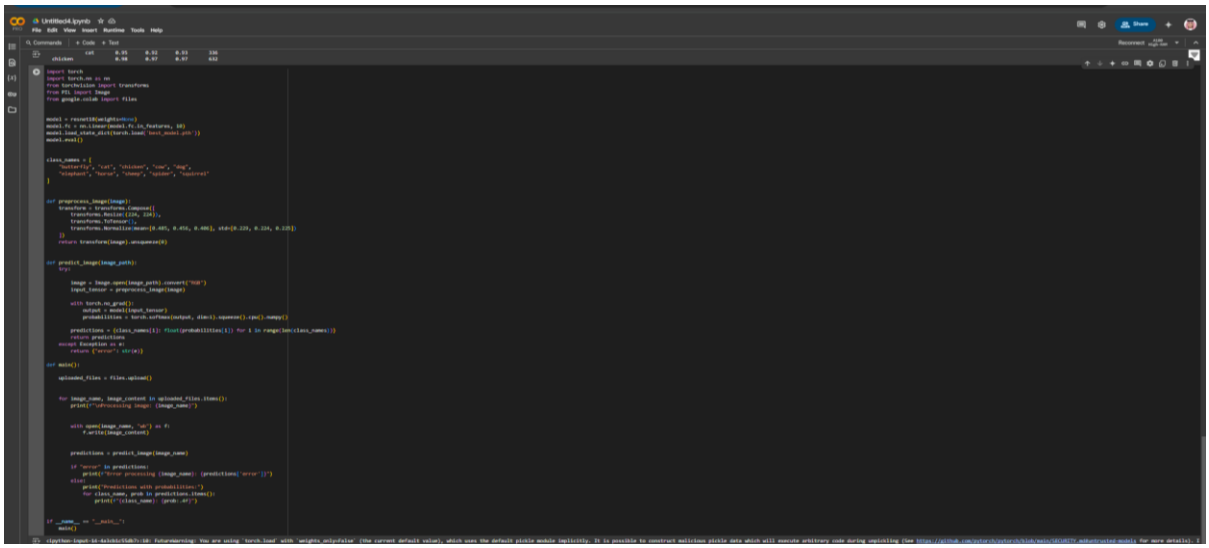
Results



	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

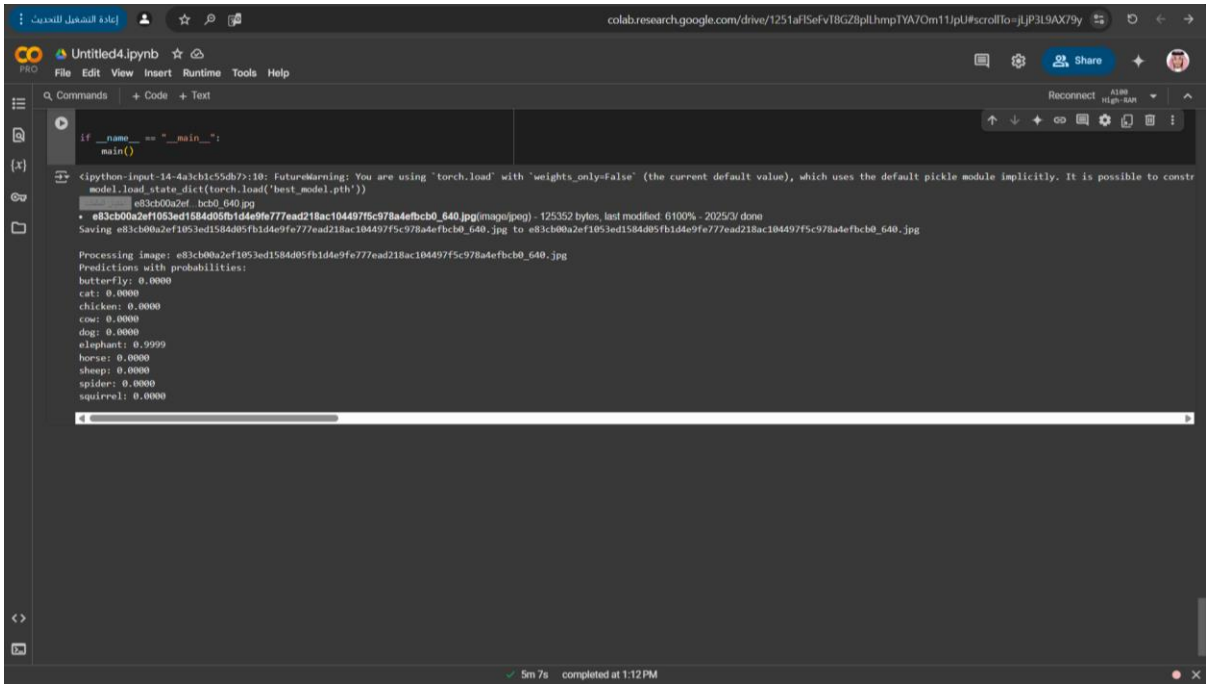
This code classifies images into 10 animal categories using a pre-trained ResNet-18 model. It uploads an image, preprocesses it, passes it through the model, and displays predictions with probabilities for each category

Iron hack Triple pixels



Results

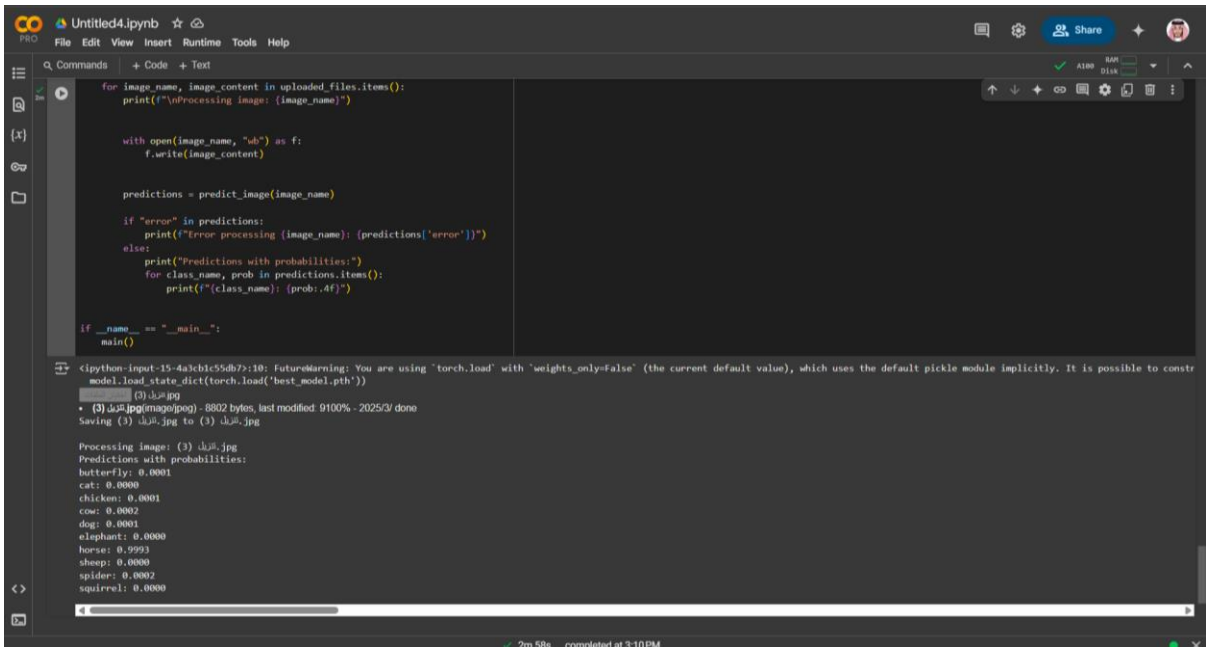
Elephant:0.999



Sample used

Iron hack Triple pixels

Horse:0.993



Sample used