

Student Name : Rahaf Barakat - id : 253002370

Sports balls - multiclass image classification

Dataset Link: <https://www.kaggle.com/datasets/samuelcortinhas/sports-balls-multiclass-image-classification>

In [1]: *# 1 connects Google Colab to Google Drive*

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

In [4]: *# 2 extracts the downloaded dataset*

```
!unzip -qq "/content/drive/MyDrive/My Project 1/archive (1).zip" -d "/content/dataset"
```

In []: *# 3 Import Required Libraries*

```
from IPython.display import Image, display

import os
import shutil
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from PIL import Image

import torch
from torch import nn, optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms as T, models

from sklearn.preprocessing import label_binarize
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    roc_curve,
    auc
)

from tqdm import tqdm
```

In []: *import os*

```
print(os.listdir("/content/dataset"))
```

['train', 'test']

In []: *# 4 Reduce Image Size (Resize)*

```
from PIL import Image
import os

resize_size = (256, 256)

def resize_folder(input_root, output_root):
    os.makedirs(output_root, exist_ok=True)
    for class_name in os.listdir(input_root):
        class_path = os.path.join(input_root, class_name)
        if not os.path.isdir(class_path):
            continue

        output_class_path = os.path.join(output_root, class_name)
        os.makedirs(output_class_path, exist_ok=True)

        for image_name in os.listdir(class_path):
            if image_name.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.tiff')):
                input_image_path = os.path.join(class_path, image_name)
                output_image_name = os.path.splitext(image_name)[0] + '.jpg'
                output_image_path = os.path.join(output_class_path, output_image_name)

                try:
                    with Image.open(input_image_path) as img:
```

```

        img = img.convert('RGB')
        img = img.resize(resize_size)
        img.save(output_image_path, 'JPEG')
    except Exception as e:
        print(f"Failed to process {input_image_path}: {e}")

resize_folder("/content/dataset/train", "/content/resized_dataset/train")
print("Train images resized successfully.")

resize_folder("/content/dataset/test", "/content/resized_dataset/test")
print("Test images resized successfully.")

```

Train images resized successfully.

Test images resized successfully.

In []: *# 5 Split Resized Dataset into Train and Validation*

```

import os
import shutil
from sklearn.model_selection import train_test_split

original_train_dir = "/content/resized_dataset/train"
base_dir = "/content/new_dataset"

train_dir = os.path.join(base_dir, "train")
validation_dir = os.path.join(base_dir, "validation")
test_dir = os.path.join(base_dir, "test")

for folder in [train_dir, validation_dir, test_dir]:
    os.makedirs(folder, exist_ok=True)

validation_ratio = 0.1

for class_name in os.listdir(original_train_dir):
    class_path = os.path.join(original_train_dir, class_name)

    if not os.path.isdir(class_path):
        continue

    images = [
        img for img in os.listdir(class_path)
        if img.lower().endswith((".jpg", ".jpeg", ".png"))
    ]

    train_images, val_images = train_test_split(
        images,
        test_size=validation_ratio,
        random_state=42
    )

    os.makedirs(os.path.join(train_dir, class_name), exist_ok=True)
    os.makedirs(os.path.join(validation_dir, class_name), exist_ok=True)

    for img in train_images:
        shutil.copy(
            os.path.join(class_path, img),
            os.path.join(train_dir, class_name, img)
        )

    for img in val_images:
        shutil.copy(
            os.path.join(class_path, img),
            os.path.join(validation_dir, class_name, img)
        )

original_test_dir = "/content/resized_dataset/test"

for class_name in os.listdir(original_test_dir):
    src = os.path.join(original_test_dir, class_name)
    dst = os.path.join(test_dir, class_name)
    if os.path.isdir(src):
        shutil.copytree(src, dst, dirs_exist_ok=True)

print("Dataset splitting completed successfully.")

```

Dataset splitting completed successfully.

In []: *# 6 Getting Name of Classes*

```

def get_classes(data_dir):
    all_data = datasets.ImageFolder(data_dir)
    return all_data.classes

classes = get_classes("/content/new_dataset/train")

print(classes)
print("Number of Classes:", len(classes))

```

```
['american_football', 'baseball', 'basketball', 'billiard_ball', 'bowling_ball', 'cricket_ball', 'football', 'golf_ball', 'hockey_ball', 'hockey_puck', 'rugby_ball', 'shuttlecock', 'table_tennis_ball', 'tennis_ball', 'volleyball']
Number of Classes: 15
```

In []: *# 7 Image Augmentation and Data Preprocessing*

```
def get_data_loaders(data_dir, batch_size, train=False):
    if train:
        transform = T.Compose([
            T.RandomHorizontalFlip(),
            T.RandomVerticalFlip(),
            T.RandomRotation(10),
            T.Resize((299, 299)),
            T.ToTensor(),
            T.Normalize(
                mean=[0.485, 0.456, 0.406],
                std=[0.229, 0.224, 0.225]
            )
        ])

        train_data = datasets.ImageFolder(
            os.path.join(data_dir, "train"),
            transform=transform
        )

        train_loader = DataLoader(
            train_data,
            batch_size=batch_size,
            shuffle=True,
            num_workers=2
        )

        return train_loader, len(train_data)

    else:
        transform = T.Compose([
            T.Resize((299, 299)),
            T.ToTensor(),
            T.Normalize(
                mean=[0.485, 0.456, 0.406],
                std=[0.229, 0.224, 0.225]
            )
        ])

        val_data = datasets.ImageFolder(
            os.path.join(data_dir, "validation"),
            transform=transform
        )

        test_data = datasets.ImageFolder(
            os.path.join(data_dir, "test"),
            transform=transform
        )

        val_loader = DataLoader(
            val_data,
            batch_size=batch_size,
            shuffle=False,
            num_workers=2
        )

        test_loader = DataLoader(
            test_data,
            batch_size=batch_size,
            shuffle=False,
            num_workers=2
        )

        return val_loader, test_loader, len(val_data), len(test_data)
```

In []: *# 8 Create Data Loaders*

```
dataset_path = "/content/new_dataset"
batch_size = 32

train_loader, train_data_len = get_data_loaders(
    dataset_path,
    batch_size,
    train=True
)

val_loader, test_loader, valid_data_len, test_data_len = get_data_loaders(
    dataset_path,
    batch_size,
    train=False
)
```

```

classes = get_classes("/content/new_dataset/train")
num_classes = len(classes)

dataloaders = {
    "train": train_loader,
    "val": val_loader
}

dataset_sizes = {
    "train": train_data_len,
    "val": valid_data_len
}

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

```

In []: # 9 Device Setup

```

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

print(device)

```

cuda

In []: # 10 Train Model Function

```

def train_model(model, criterion, optimizer, scheduler, num_epochs=10):
    """Train and validate the model."""
    for epoch in range(num_epochs):
        print(f"Epoch {epoch+1}/{num_epochs}")
        print('-' * 10)

        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0

            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)

                    if isinstance(outputs, tuple) or hasattr(outputs, "logits"):
                        main_outputs = outputs.logits if hasattr(outputs, "logits") else outputs[0]
                        aux_outputs = outputs.aux_logits if hasattr(outputs, "aux_logits") else None

                        loss = criterion(main_outputs, labels)

                        if aux_outputs is not None:
                            loss += 0.4 * criterion(aux_outputs, labels)
                    else:
                        main_outputs = outputs
                        loss = criterion(main_outputs, labels)

                    _, preds = torch.max(main_outputs, 1)

                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            if phase == 'train':
                scheduler.step()

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc = running_corrects.double() / dataset_sizes[phase]

            print(f"{phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}")

        return model

```

In []: # 11 Evaluate Model

```

def evaluate_model(model_name, y_true, y_pred, classes):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')

```

```

cm = confusion_matrix(y_true, y_pred)

print("=" * 50)
print(model_name)
print("=" * 50)
print("Accuracy :", accuracy)
print("Precision:", precision)
print("Recall   :", recall)
print("F1 Score :", f1)

print("\nClassification Report")
print(classification_report(y_true, y_pred, target_names=classes))

plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=classes, yticklabels=classes)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title(f"Confusion Matrix - {model_name}")
plt.show()

```

In []: # 12 EfficientNet_B0

```

model = models.efficientnet_b0(pretrained=True)

model.classifier[1] = nn.Linear(
    model.classifier[1].in_features,
    num_classes
)

model = model.to(device)

criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)

optimizer = optim.AdamW(model.parameters(), lr=0.001)

exp_lr_scheduler = optim.lr_scheduler.StepLR(
    optimizer, step_size=3, gamma=0.97
)

```

In []: # 13 Train EfficientNet_B0

```

model = train_model(
    model, criterion, optimizer, exp_lr_scheduler, num_epochs=10
)

```

Epoch 1/10

train Loss: 1.0310 Acc: 0.6955

val Loss: 0.6728 Acc: 0.8033

Epoch 2/10

train Loss: 0.5667 Acc: 0.8272

val Loss: 0.5374 Acc: 0.8494

Epoch 3/10

train Loss: 0.4535 Acc: 0.8645

val Loss: 0.5300 Acc: 0.8521

Epoch 4/10

train Loss: 0.3390 Acc: 0.8982

val Loss: 0.4842 Acc: 0.8711

Epoch 5/10

train Loss: 0.3021 Acc: 0.9062

val Loss: 0.4772 Acc: 0.8494

Epoch 6/10

train Loss: 0.2660 Acc: 0.9190

val Loss: 0.4925 Acc: 0.8562

Epoch 7/10

train Loss: 0.2157 Acc: 0.9351

val Loss: 0.4900 Acc: 0.8738

Epoch 8/10

train Loss: 0.1853 Acc: 0.9408

val Loss: 0.5325 Acc: 0.8630

Epoch 9/10

train Loss: 0.1812 Acc: 0.9416

val Loss: 0.4732 Acc: 0.8833

Epoch 10/10

train Loss: 0.1548 Acc: 0.9539

val Loss: 0.4498 Acc: 0.8779

```
In [ ]: # 14 Test and Evaluate EfficientNet_B0

model.eval()
y_true, y_pred = [], []

with torch.no_grad():
    for x, y in tqdm(test_loader):
        x = x.to(device)
        output = model(x)

        y_true.extend(y.numpy())
        pred = torch.argmax(output, axis=1).cpu().numpy()
        y_pred.extend(pred)

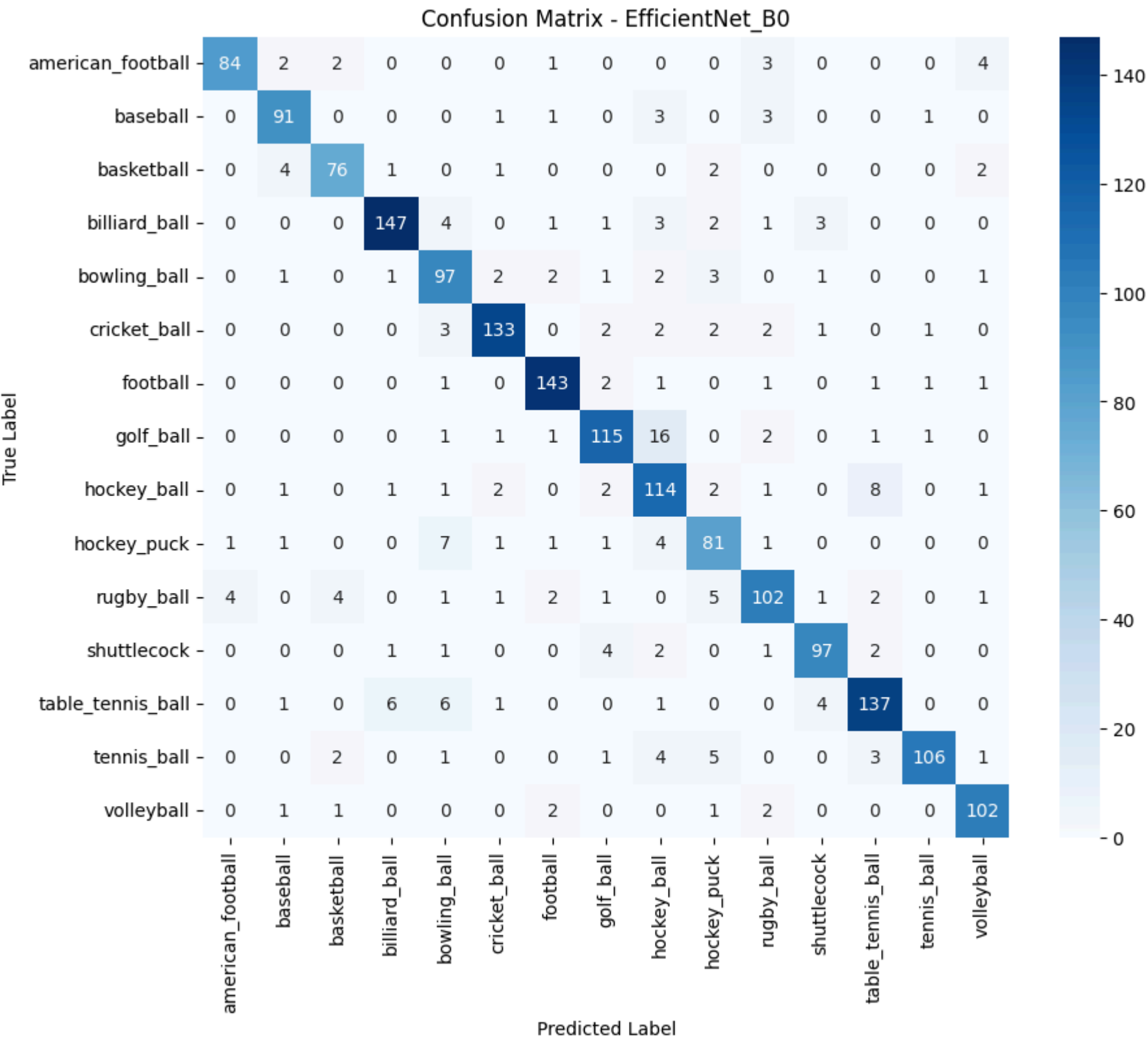
y_true = np.array(y_true)
y_pred = np.array(y_pred)

evaluate_model("EfficientNet_B0", y_true, y_pred, classes)
```

100%|██████████| 58/58 [00:08<00:00, 7.01it/s]

=====
EfficientNet_B0
=====
Accuracy : 0.8826724606192287
Precision: 0.885780646196958
Recall : 0.8826724606192287
F1 Score : 0.8834077399975422

Classification Report				
	precision	recall	f1-score	support
american_football	0.94	0.88	0.91	96
baseball	0.89	0.91	0.90	100
basketball	0.89	0.88	0.89	86
billiard_ball	0.94	0.91	0.92	162
bowling_ball	0.79	0.87	0.83	111
cricket_ball	0.93	0.91	0.92	146
football	0.93	0.95	0.94	151
golf_ball	0.88	0.83	0.86	138
hockey_ball	0.75	0.86	0.80	133
hockey_puck	0.79	0.83	0.81	98
rugby_ball	0.86	0.82	0.84	124
shuttlecock	0.91	0.90	0.90	108
table_tennis_ball	0.89	0.88	0.88	156
tennis_ball	0.96	0.86	0.91	123
volleyball	0.90	0.94	0.92	109
accuracy			0.88	1841
macro avg	0.88	0.88	0.88	1841
weighted avg	0.89	0.88	0.88	1841



```
In [ ]: # 15 ResNet18

model = models.resnet18(pretrained=True)

model.fc = nn.Linear(model.fc.in_features, num_classes)
model = model.to(device)

criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)

optimizer = optim.AdamW(model.parameters(), lr=0.001)

exp_lr_scheduler = optim.lr_scheduler.StepLR(
    optimizer, step_size=3, gamma=0.97
)

In [ ]: # 16 Train ResNet18

model = train_model(
    model, criterion, optimizer, exp_lr_scheduler, num_epochs=10
)
```

Epoch 1/10

train Loss: 1.6027 Acc: 0.5074
val Loss: 1.4073 Acc: 0.5563
Epoch 2/10

train Loss: 1.1331 Acc: 0.6526
val Loss: 1.5254 Acc: 0.5916
Epoch 3/10

train Loss: 0.9566 Acc: 0.7107
val Loss: 1.3003 Acc: 0.6282
Epoch 4/10

train Loss: 0.7908 Acc: 0.7607
val Loss: 0.9596 Acc: 0.6934
Epoch 5/10

train Loss: 0.6709 Acc: 0.7911
val Loss: 0.9112 Acc: 0.7327
Epoch 6/10

train Loss: 0.6303 Acc: 0.8097
val Loss: 0.8569 Acc: 0.7341
Epoch 7/10

train Loss: 0.5545 Acc: 0.8281
val Loss: 0.8942 Acc: 0.7178
Epoch 8/10

train Loss: 0.5118 Acc: 0.8352
val Loss: 0.6988 Acc: 0.7870
Epoch 9/10

train Loss: 0.4654 Acc: 0.8530
val Loss: 0.9512 Acc: 0.7436
Epoch 10/10

train Loss: 0.4085 Acc: 0.8701
val Loss: 0.6476 Acc: 0.8358

In []: *# 17 Test and Evaluate ResNet18*

```
model.eval()
y_true, y_pred = [], []

with torch.no_grad():
    for x, y in tqdm(test_loader):
        x = x.to(device)
        output = model(x)

        y_true.extend(y.numpy())
        pred = torch.argmax(output, axis=1).cpu().numpy()
        y_pred.extend(pred)

y_true = np.array(y_true)
y_pred = np.array(y_pred)

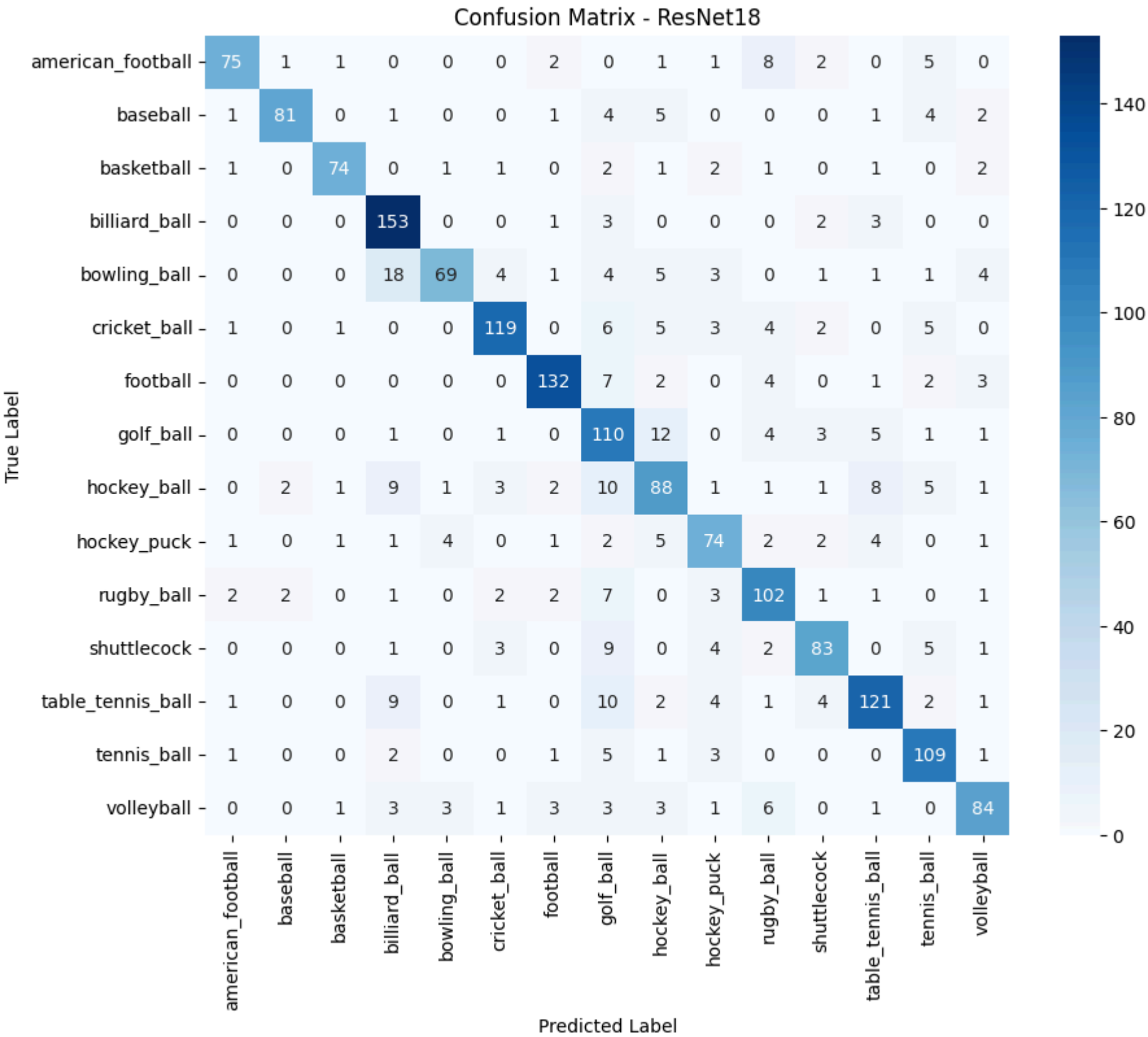
evaluate_model("ResNet18", y_true, y_pred, classes)
```

100%|██████████| 58/58 [00:07<00:00, 8.19it/s]

```
=====
ResNet18
=====
Accuracy : 0.8006518196632265
Precision: 0.8110859700302494
Recall   : 0.8006518196632265
F1 Score : 0.8015237293726855
```

Classification Report

	precision	recall	f1-score	support
american_football	0.90	0.78	0.84	96
baseball	0.94	0.81	0.87	100
basketball	0.94	0.86	0.90	86
billiard_ball	0.77	0.94	0.85	162
bowling_ball	0.88	0.62	0.73	111
cricket_ball	0.88	0.82	0.85	146
football	0.90	0.87	0.89	151
golf_ball	0.60	0.80	0.69	138
hockey_ball	0.68	0.66	0.67	133
hockey_puck	0.75	0.76	0.75	98
rugby_ball	0.76	0.82	0.79	124
shuttlecock	0.82	0.77	0.79	108
table_tennis_ball	0.82	0.78	0.80	156
tennis_ball	0.78	0.89	0.83	123
volleyball	0.82	0.77	0.80	109
accuracy			0.80	1841
macro avg	0.82	0.80	0.80	1841
weighted avg	0.81	0.80	0.80	1841



```
In [ ]: # 18 DenseNet121

model = models.densenet121(pretrained=True)

model.classifier = nn.Linear(model.classifier.in_features, num_classes)
model = model.to(device)
```

```

criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)

optimizer = optim.AdamW(model.parameters(), lr=0.001)

exp_lr_scheduler = optim.lr_scheduler.StepLR(
    optimizer, step_size=3, gamma=0.97
)
```

```

In [ ]: # 19 Train DenseNet121

model = train_model(
    model, criterion, optimizer, exp_lr_scheduler, num_epochs=10
)
```

Epoch 1/10

train Loss: 1.3508 Acc: 0.5876
val Loss: 1.0152 Acc: 0.6811
Epoch 2/10

train Loss: 0.9273 Acc: 0.7233
val Loss: 1.0518 Acc: 0.6784
Epoch 3/10

train Loss: 0.7664 Acc: 0.7663
val Loss: 0.6806 Acc: 0.7951
Epoch 4/10

train Loss: 0.6139 Acc: 0.8079
val Loss: 0.7296 Acc: 0.7626
Epoch 5/10

train Loss: 0.5863 Acc: 0.8178
val Loss: 0.6831 Acc: 0.7870
Epoch 6/10

train Loss: 0.4879 Acc: 0.8457
val Loss: 0.6265 Acc: 0.8223
Epoch 7/10

train Loss: 0.4309 Acc: 0.8604
val Loss: 0.6173 Acc: 0.8073
Epoch 8/10

train Loss: 0.3958 Acc: 0.8765
val Loss: 0.6259 Acc: 0.8209
Epoch 9/10

train Loss: 0.3917 Acc: 0.8797
val Loss: 0.6455 Acc: 0.8290
Epoch 10/10

train Loss: 0.3103 Acc: 0.9021
val Loss: 0.6879 Acc: 0.8195

```

In [ ]: # 20 Test and Evaluate DenseNet121

model.eval()
y_true, y_pred = [], []

with torch.no_grad():
    for x, y in tqdm(test_loader):
        x = x.to(device)
        output = model(x)

        y_true.extend(y.numpy())
        pred = torch.argmax(output, axis=1).cpu().numpy()
        y_pred.extend(pred)

y_true = np.array(y_true)
y_pred = np.array(y_pred)

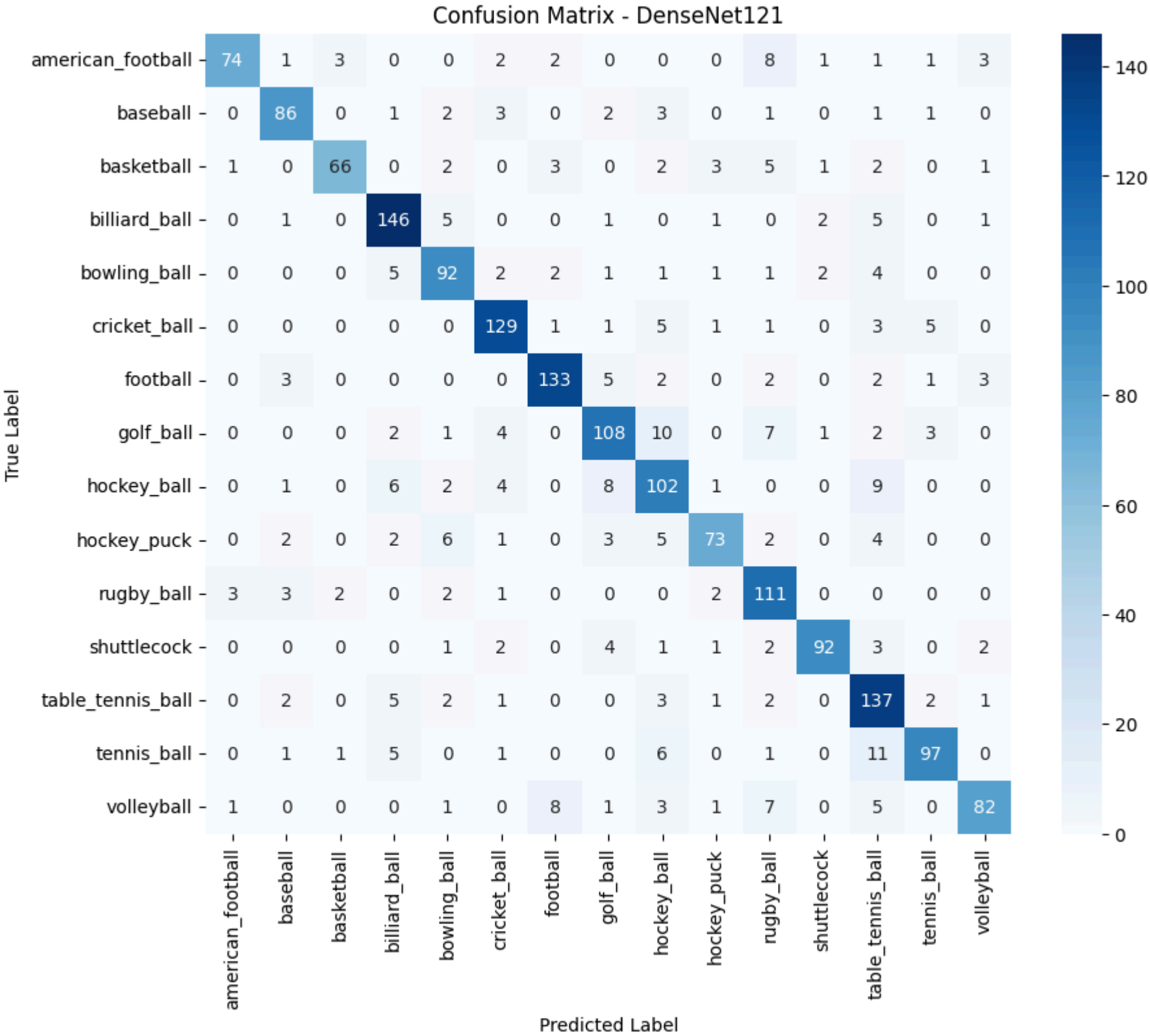
evaluate_model("DenseNet121", y_true, y_pred, classes)
```

100%|██████████| 58/58 [00:12<00:00, 4.70it/s]

=====
DenseNet121
=====
Accuracy : 0.8299837045084193
Precision: 0.8368705125290764
Recall : 0.8299837045084193
F1 Score : 0.8305978568797253

Classification Report

	precision	recall	f1-score	support
american_football	0.94	0.77	0.85	96
baseball	0.86	0.86	0.86	100
basketball	0.92	0.77	0.84	86
billiard_ball	0.85	0.90	0.87	162
bowling_ball	0.79	0.83	0.81	111
cricket_ball	0.86	0.88	0.87	146
football	0.89	0.88	0.89	151
golf_ball	0.81	0.78	0.79	138
hockey_ball	0.71	0.77	0.74	133
hockey_puck	0.86	0.74	0.80	98
rugby_ball	0.74	0.90	0.81	124
shuttlecock	0.93	0.85	0.89	108
table_tennis_ball	0.72	0.88	0.79	156
tennis_ball	0.88	0.79	0.83	123
volleyball	0.88	0.75	0.81	109
accuracy			0.83	1841
macro avg	0.84	0.82	0.83	1841
weighted avg	0.84	0.83	0.83	1841



```
In [ ]: # 21 VGG16

model = models.vgg16(pretrained=True)

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

```
model.classifier[6] = nn.Linear(
    model.classifier[6].in_features, num_classes
)

model = model.to(device)

criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)

optimizer = optim.AdamW(model.classifier.parameters(), lr=0.0001)

exp_lr_scheduler = optim.lr_scheduler.StepLR(
    optimizer, step_size=3, gamma=0.97
)
```

In []: *# 22 Train VGG16*

```
model = train_model(
    model, criterion, optimizer, exp_lr_scheduler, num_epochs=10
)
```

```
Epoch 1/10
-----
train Loss: 1.2387 Acc: 0.6216
val Loss: 0.7344 Acc: 0.7748
Epoch 2/10
-----
train Loss: 0.7329 Acc: 0.7647
val Loss: 0.6425 Acc: 0.8046
Epoch 3/10
-----
train Loss: 0.5854 Acc: 0.8172
val Loss: 0.6438 Acc: 0.8087
Epoch 4/10
-----
train Loss: 0.4505 Acc: 0.8556
val Loss: 0.5141 Acc: 0.8372
Epoch 5/10
-----
train Loss: 0.3779 Acc: 0.8803
val Loss: 0.5525 Acc: 0.8480
Epoch 6/10
-----
train Loss: 0.3221 Acc: 0.9005
val Loss: 0.5408 Acc: 0.8412
Epoch 7/10
-----
train Loss: 0.2525 Acc: 0.9184
val Loss: 0.5268 Acc: 0.8575
Epoch 8/10
-----
train Loss: 0.2323 Acc: 0.9252
val Loss: 0.5082 Acc: 0.8616
Epoch 9/10
-----
train Loss: 0.2204 Acc: 0.9263
val Loss: 0.4993 Acc: 0.8548
Epoch 10/10
-----
train Loss: 0.1708 Acc: 0.9460
val Loss: 0.5411 Acc: 0.8562
```

In []: *# 23 Test and Evaluate VGG16*

```
model.eval()
y_true, y_pred = [], []

with torch.no_grad():
    for x, y in tqdm(test_loader):
        x = x.to(device)
        output = model(x)

        y_true.extend(y.numpy())
        pred = torch.argmax(output, axis=1).cpu().numpy()
        y_pred.extend(pred)

y_true = np.array(y_true)
y_pred = np.array(y_pred)

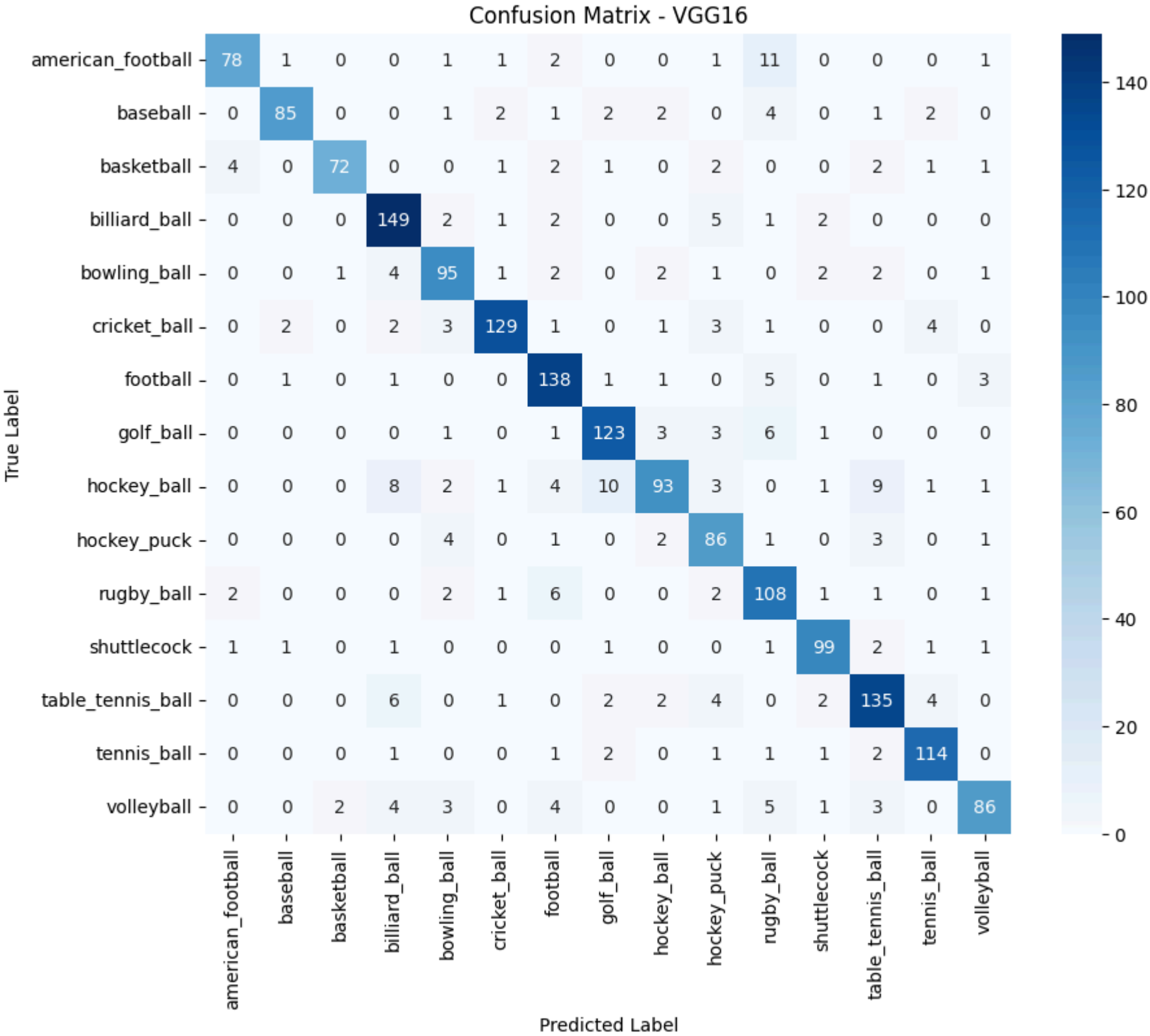
evaluate_model("VGG16", y_true, y_pred, classes)
```

100%|██████████| 58/58 [00:19<00:00, 2.91it/s]

=====
VGG16
=====
Accuracy : 0.8636610537751223
Precision: 0.8680348101064493
Recall : 0.8636610537751223
F1 Score : 0.8635323451445537

Classification Report

	precision	recall	f1-score	support
american_football	0.92	0.81	0.86	96
baseball	0.94	0.85	0.89	100
basketball	0.96	0.84	0.89	86
billiard_ball	0.85	0.92	0.88	162
bowling_ball	0.83	0.86	0.84	111
cricket_ball	0.93	0.88	0.91	146
football	0.84	0.91	0.87	151
golf_ball	0.87	0.89	0.88	138
hockey_ball	0.88	0.70	0.78	133
hockey_puck	0.77	0.88	0.82	98
rugby_ball	0.75	0.87	0.81	124
shuttlecock	0.90	0.92	0.91	108
table_tennis_ball	0.84	0.87	0.85	156
tennis_ball	0.90	0.93	0.91	123
volleyball	0.90	0.79	0.84	109
accuracy			0.86	1841
macro avg	0.87	0.86	0.86	1841
weighted avg	0.87	0.86	0.86	1841



```
In [ ]: # 24 InceptionV3

model = models.inception_v3(pretrained=True, aux_logits=True)

model.fc = nn.Linear(model.fc.in_features, num_classes)

if model.aux_logits:
```

```

        model.AuxLogits.fc = nn.Linear(
            model.AuxLogits.fc.in_features, num_classes
        )

    model = model.to(device)

    criterion = nn.CrossEntropyLoss().to(device)

    optimizer = optim.AdamW(model.parameters(), lr=0.002)

    exp_lr_scheduler = optim.lr_scheduler.StepLR(
        optimizer, step_size=3, gamma=0.97
    )

```

In []: *# 25 Train InceptionV3*

```

model = train_model(
    model, criterion, optimizer, exp_lr_scheduler, num_epochs=10
)

```

```

Epoch 1/10
-----
train Loss: 3.2084 Acc: 0.2902
val Loss: 2.0579 Acc: 0.3501
Epoch 2/10
-----
train Loss: 2.5916 Acc: 0.4330
val Loss: 1.6824 Acc: 0.4518
Epoch 3/10
-----
train Loss: 2.3105 Acc: 0.4882
val Loss: 1.8035 Acc: 0.4613
Epoch 4/10
-----
train Loss: 1.9983 Acc: 0.5594
val Loss: 2.0533 Acc: 0.4396
Epoch 5/10
-----
train Loss: 1.7788 Acc: 0.6049
val Loss: 1.2719 Acc: 0.6174
Epoch 6/10
-----
train Loss: 1.6230 Acc: 0.6471
val Loss: 1.2362 Acc: 0.6255
Epoch 7/10
-----
train Loss: 1.4796 Acc: 0.6755
val Loss: 1.0983 Acc: 0.6649
Epoch 8/10
-----
train Loss: 1.3525 Acc: 0.7007
val Loss: 1.0790 Acc: 0.6906
Epoch 9/10
-----
train Loss: 1.2533 Acc: 0.7255
val Loss: 0.8686 Acc: 0.7341
Epoch 10/10
-----
train Loss: 1.1608 Acc: 0.7437
val Loss: 0.9062 Acc: 0.7083

```

In []: *# 26 Test and Evaluate InceptionV3*

```

model.eval()
y_true, y_pred = [], []

with torch.no_grad():
    for x, y in tqdm(test_loader):
        x = x.to(device)
        output = model(x)

        if isinstance(output, tuple):
            output = output[0]

        y_true.extend(y.numpy())
        pred = torch.argmax(output, axis=1).cpu().numpy()
        y_pred.extend(pred)

y_true = np.array(y_true)
y_pred = np.array(y_pred)

evaluate_model("InceptionV3", y_true, y_pred, classes)

```

100%|██████████| 58/58 [00:10<00:00, 5.62it/s]

=====
InceptionV3
=====
Accuracy : 0.7077675176534493
Precision: 0.7266957447474496
Recall : 0.7077675176534493
F1 Score : 0.7059616321545075

Classification Report				
	precision	recall	f1-score	support
american_football	0.73	0.75	0.74	96
baseball	1.00	0.59	0.74	100
basketball	0.68	0.80	0.73	86
billiard_ball	0.56	0.94	0.70	162
bowling_ball	0.61	0.53	0.57	111
cricket_ball	0.87	0.72	0.79	146
football	0.82	0.76	0.79	151
golf_ball	0.78	0.59	0.67	138
hockey_ball	0.57	0.42	0.48	133
hockey_puck	0.66	0.60	0.63	98
rugby_ball	0.67	0.83	0.74	124
shuttlecock	0.75	0.79	0.77	108
table_tennis_ball	0.65	0.76	0.70	156
tennis_ball	0.90	0.76	0.82	123
volleyball	0.71	0.69	0.70	109
accuracy			0.71	1841
macro avg	0.73	0.70	0.71	1841
weighted avg	0.73	0.71	0.71	1841

