

```
In [4]: #Imports and Setup
import os
import numpy as np
import torch
from torchvision import datasets, transforms as T, models
from torch import nn, optim
from torch.utils.data import DataLoader
import warnings
warnings.filterwarnings ("ignore")
from tqdm import tqdm
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, classification_report
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
from sklearn.metrics import auc
import time
```

```
In [5]: #Image Resizing and Preprocessing
from PIL import Image
import os

input_root = 'dataset/'
output_root = 'output/'

resize_size = (256, 256) # Change size as needed

# Create output root if it doesn't exist
os.makedirs (output_root, exist_ok=True)

# Iterate through each class folder
for class_name in os.listdir(input_root):
    class_path = os.path.join(input_root, class_name)

    # Skip if it's not a directory or if it's the 'masks' folder
    if not os.path.isdir(class_path) or class_name == 'masks':
        continue

    # Create output path for each class
    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', '.tiff', '.bmp')):
            input_image_path = os.path.join(class_path, image_name)
            # Change the output extension to .jpg
            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')
                    print(f"Processed: {input_image_path} -> {output_image_path}")
            except Exception as e:
                print(f"Failed to process {input_image_path}: {e}")
```

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```

Processed: dataset/Moon_jellyfish\aug-42-55.jpg -> output/Moon_jellyfish\aug-42-55.jpg
Processed: dataset/Moon_jellyfish\aug-43-56.jpg -> output/Moon_jellyfish\aug-43-56.jpg
Processed: dataset/Moon_jellyfish\aug-44-57.jpg -> output/Moon_jellyfish\aug-44-57.jpg
Processed: dataset/Moon_jellyfish\aug-45-58.jpg -> output/Moon_jellyfish\aug-45-58.jpg
Processed: dataset/Moon_jellyfish\aug-46-61.jpg -> output/Moon_jellyfish\aug-46-61.jpg
Processed: dataset/Moon_jellyfish\aug-47-01.jpg -> output/Moon_jellyfish\aug-47-01.jpg
Processed: dataset/Moon_jellyfish\aug-48-02.jpg -> output/Moon_jellyfish\aug-48-02.jpg
Processed: dataset/Moon_jellyfish\aug-49-03.jpg -> output/Moon_jellyfish\aug-49-03.jpg
Processed: dataset/Moon_jellyfish\aug-5-06.jpg -> output/Moon_jellyfish\aug-5-06.jpg
Processed: dataset/Moon_jellyfish\aug-50-04.jpg -> output/Moon_jellyfish\aug-50-04.jpg
Processed: dataset/Moon_jellyfish\aug-51-05.jpg -> output/Moon_jellyfish\aug-51-05.jpg
Processed: dataset/Moon_jellyfish\aug-52-06.jpg -> output/Moon_jellyfish\aug-52-06.jpg
Processed: dataset/Moon_jellyfish\aug-53-08.jpg -> output/Moon_jellyfish\aug-53-08.jpg
Processed: dataset/Moon_jellyfish\aug-54-10.jpg -> output/Moon_jellyfish\aug-54-10.jpg
Processed: dataset/Moon_jellyfish\aug-55-12.jpg -> output/Moon_jellyfish\aug-55-12.jpg
Processed: dataset/Moon_jellyfish\aug-56-13.jpg -> output/Moon_jellyfish\aug-56-13.jpg
Processed: dataset/Moon_jellyfish\aug-57-14.jpg -> output/Moon_jellyfish\aug-57-14.jpg
Processed: dataset/Moon_jellyfish\aug-58-16.jpg -> output/Moon_jellyfish\aug-58-16.jpg
Processed: dataset/Moon_jellyfish\aug-59-17.jpg -> output/Moon_jellyfish\aug-59-17.jpg
Processed: dataset/Moon_jellyfish\aug-6-08.jpg -> output/Moon_jellyfish\aug-6-08.jpg
Processed: dataset/Moon_jellyfish\aug-60-18.jpg -> output/Moon_jellyfish\aug-60-18.jpg
Processed: dataset/Moon_jellyfish\aug-61-21.jpg -> output/Moon_jellyfish\aug-61-21.jpg
Processed: dataset/Moon_jellyfish\aug-62-22.jpg -> output/Moon_jellyfish\aug-62-22.jpg
Processed: dataset/Moon_jellyfish\aug-63-24.jpg -> output/Moon_jellyfish\aug-63-24.jpg
Processed: dataset/Moon_jellyfish\aug-64-25.jpg -> output/Moon_jellyfish\aug-64-25.jpg
Processed: dataset/Moon_jellyfish\aug-65-26.jpg -> output/Moon_jellyfish\aug-65-26.jpg
Processed: dataset/Moon_jellyfish\aug-66-28.jpg -> output/Moon_jellyfish\aug-66-28.jpg
Processed: dataset/Moon_jellyfish\aug-67-29.JPG -> output/Moon_jellyfish\aug-67-29.jpg
Processed: dataset/Moon_jellyfish\aug-68-30.jpg -> output/Moon_jellyfish\aug-68-30.jpg
Processed: dataset/Moon_jellyfish\aug-69-31.jpg -> output/Moon_jellyfish\aug-69-31.jpg
Processed: dataset/Moon_jellyfish\aug-7-10.jpg -> output/Moon_jellyfish\aug-7-10.jpg
Processed: dataset/Moon_jellyfish\aug-70-32.jpg -> output/Moon_jellyfish\aug-70-32.jpg
Processed: dataset/Moon_jellyfish\aug-71-34.jpg -> output/Moon_jellyfish\aug-71-34.jpg
Processed: dataset/Moon_jellyfish\aug-72-35.jpg -> output/Moon_jellyfish\aug-72-35.jpg
Processed: dataset/Moon_jellyfish\aug-73-36.jpg -> output/Moon_jellyfish\aug-73-36.jpg
Processed: dataset/Moon_jellyfish\aug-74-37.jpg -> output/Moon_jellyfish\aug-74-37.jpg
Processed: dataset/Moon_jellyfish\aug-75-38.jpg -> output/Moon_jellyfish\aug-75-38.jpg
Processed: dataset/Moon_jellyfish\aug-76-40.jpg -> output/Moon_jellyfish\aug-76-40.jpg
Processed: dataset/Moon_jellyfish\aug-77-41.jpg -> output/Moon_jellyfish\aug-77-41.jpg
Processed: dataset/Moon_jellyfish\aug-78-42.jpg -> output/Moon_jellyfish\aug-78-42.jpg
Processed: dataset/Moon_jellyfish\aug-79-43.jpg -> output/Moon_jellyfish\aug-79-43.jpg
Processed: dataset/Moon_jellyfish\aug-8-12.jpg -> output/Moon_jellyfish\aug-8-12.jpg
Processed: dataset/Moon_jellyfish\aug-80-45.jpg -> output/Moon_jellyfish\aug-80-45.jpg
Processed: dataset/Moon_jellyfish\aug-81-46.jpg -> output/Moon_jellyfish\aug-81-46.jpg
Processed: dataset/Moon_jellyfish\aug-82-47.jpg -> output/Moon_jellyfish\aug-82-47.jpg
Processed: dataset/Moon_jellyfish\aug-83-49.jpg -> output/Moon_jellyfish\aug-83-49.jpg
Processed: dataset/Moon_jellyfish\aug-84-50.jpg -> output/Moon_jellyfish\aug-84-50.jpg
Processed: dataset/Moon_jellyfish\aug-85-51.jpg -> output/Moon_jellyfish\aug-85-51.jpg
Processed: dataset/Moon_jellyfish\aug-86-52.JPG -> output/Moon_jellyfish\aug-86-52.jpg
Processed: dataset/Moon_jellyfish\aug-87-53.jpg -> output/Moon_jellyfish\aug-87-53.jpg
Processed: dataset/Moon_jellyfish\aug-88-54.jpg -> output/Moon_jellyfish\aug-88-54.jpg
Processed: dataset/Moon_jellyfish\aug-89-55.jpg -> output/Moon_jellyfish\aug-89-55.jpg
Processed: dataset/Moon_jellyfish\aug-9-13.jpg -> output/Moon_jellyfish\aug-9-13.jpg
Processed: dataset/Moon_jellyfish\aug-90-56.jpg -> output/Moon_jellyfish\aug-90-56.jpg
Processed: dataset/Moon_jellyfish\aug-91-57.jpg -> output/Moon_jellyfish\aug-91-57.jpg
Processed: dataset/Moon_jellyfish\aug-92-58.jpg -> output/Moon_jellyfish\aug-92-58.jpg
Processed: dataset/Moon_jellyfish\aug-93-61.jpg -> output/Moon_jellyfish\aug-93-61.jpg
Processed: dataset/Moon_jellyfish\aug-94-01.jpg -> output/Moon_jellyfish\aug-94-01.jpg
Processed: dataset/Moon_jellyfish\aug-95-02.jpg -> output/Moon_jellyfish\aug-95-02.jpg
Processed: dataset/Moon_jellyfish\aug-96-03.jpg -> output/Moon_jellyfish\aug-96-03.jpg
Processed: dataset/Moon_jellyfish\aug-97-04.jpg -> output/Moon_jellyfish\aug-97-04.jpg
Processed: dataset/Moon_jellyfish\aug-98-05.jpg -> output/Moon_jellyfish\aug-98-05.jpg
Processed: dataset/Moon_jellyfish\aug-99-06.jpg -> output/Moon_jellyfish\aug-99-06.jpg

```

```

In [6]: #Dataset Splitting
import os, shutil
from sklearn.model_selection import train_test_split

original_dataset_dir = 'output'
base_dir = '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 dir in [train_dir, validation_dir, test_dir]:
    os.makedirs (dir, exist_ok=True)

train_ratio, validation_ratio, test_ratio = 0.7, 0.1, 0.2

def split_and_copy_files(class_dir, dest_base_dir):
    all_images = [os.path.join(class_dir, f) for f in os.listdir(class_dir) if os.path.isfile(os.path.join(class_dir, f))]

    # Split into train and temp (validation + test)
    train_images, temp_images = train_test_split(all_images,
        test_size=(1 - train_ratio), random_state=42)

    # Split temp into validation and test
    validation_images, test_images = train_test_split(temp_images,
        test_size=test_ratio/(validation_ratio + test_ratio), random_state=42)

    for images, folder_name in zip([train_images, validation_images, test_images], ['train', 'validation', 'test']):
        dest_dir = os.path.join(dest_base_dir, folder_name, os.path.basename(class_dir))
        os.makedirs (dest_dir, exist_ok=True)

```

```

        for file in images:
            shutil.copy(file, dest_dir)

# Apply to all class subfolders
for class_dir in os.listdir(original_dataset_dir):
    full_class_dir = os.path.join(original_dataset_dir, class_dir)
    if os.path.isdir(full_class_dir):
        split_and_copy_files(full_class_dir, base_dir)

```

```

In [7]: #Data Utility Functions and Loaders
#Get Class Function
def get_classes (data_dir):
    all_data = datasets.ImageFolder (data_dir)
    return all_data.classes
classes = get_classes("new dataset/test")
print(classes, len(classes))

```

```
['Moon_jellyfish', 'barrel_jellyfish', 'blue_jellyfish', 'compass_jellyfish', 'lions_mane_jellyfish', 'mauve_stinger_jellyfish'] 6
```

```

In [8]: #Data Agumentation
#Get Data Loaders
def get_data_loaders (data_dir, batch_size, train=False):
    """Prepare data loaders for training, validation, and testing."""
    if train:
        # Data augmentation for training
        transform = T.Compose ([
            T.RandomHorizontalFlip(),
            T.RandomVerticalFlip(),
            T.RandomRotation(10),
            T.Resize ((299, 299)), # Note: Image size for InceptionV3
            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=4)
        return train_loader, len (train_data)
    else:
        # Validation and test data transformations
        transform = T.Compose ([
            T.Resize((299, 299)), # Note: Image size for InceptionV3
            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=4)
        test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=False, num_workers=4)
        return val_loader, test_loader, len (val_data), len(test_data)

```

```

In [9]: #Data Loader and Device Setup
dataset_path = "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("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 setup
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

```

```

In [10]: print (torch.cuda.is_available() )
print (device)

```

```
True
cuda
```

```

In [20]: # Load the pre-trained efficientnet_b0 model
model = models.efficientnet_b0(pretrained=True)
# Update the fully connected layer for the number of classes
model.classifier[1] = nn.Linear(model.classifier[1].in_features, num_classes)
model = model.to(device)

# Loss function and optimizer
criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)

# Learning rate scheduler
exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.97)

# Train Model
def train_model(model, criterion, optimizer, scheduler, num_epochs=20):
    """Train and validate the model."""
    for epoch in range(num_epochs):
        # Start timer for this epoch
        start_time = time.time()
        print("Epoch {} started at: {}".format(
            epoch + 1, time.strftime("%Y-%m-%d %H:%M:%S", time.localtime(start_time))
        ))
        print(f"Epoch {epoch+1}/{num_epochs}")
        print('-' * 10)

```

```

# Each epoch has a training and validation phase
for phase in ['train', 'val']:
    if phase == 'train':
        model.train() # Set model to training mode
    else:
        model.eval() # Set model to evaluation mode

    running_loss = 0.0
    running_corrects = 0

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

        # Zero the parameter gradients
        optimizer.zero_grad()

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

            # Handle auxiliary logits for models like Inception-v3
            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: # Add auxiliary loss if present
                    loss += 0.4 * criterion(aux_outputs, labels)
            else:
                main_outputs = outputs
                loss = criterion(main_outputs, labels)

        # Predictions from logits
        _, preds = torch.max(main_outputs, 1)

        # Backward pass and optimization in training phase
        if phase == 'train':
            loss.backward()
            optimizer.step()

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

    # Step the scheduler in the training phase
    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}")

    # End timer for this epoch
    end_time = time.time()
    print("Epoch {} finished at: {}".format(
        epoch + 1, time.strftime("%Y-%m-%d %H:%M:%S", time.localtime(end_time))
    ))
    print("Epoch {} duration: {:.2f} minutes".format(
        epoch + 1, (end_time - start_time) / 60
    ))
    print("=" * 30)

return model

```

In [21]: `model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=2)`

```

Epoch 1 started at: 2025-11-14 21:34:39
Epoch 1/2
-----
train Loss: 0.8563 Acc: 0.7179
val Loss: 0.6292 Acc: 0.8000
Epoch 1 finished at: 2025-11-14 21:34:58
Epoch 1 duration: 0.31 minutes
=====
Epoch 2 started at: 2025-11-14 21:34:58
Epoch 2/2
-----
train Loss: 0.3538 Acc: 0.8718
val Loss: 0.2752 Acc: 0.9333
Epoch 2 finished at: 2025-11-14 21:35:15
Epoch 2 duration: 0.29 minutes
=====

```

In [22]: `#General Training and Evaluation Functions`
`#Evaluation 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("Accuracy:", accuracy)
print("Precision", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("Classification Report:")
print(classification_report(y_true, y_pred))

# Plot confusion matrix
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=classes, yticklabels=classes)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
#Test/Inference Loop
y_true = []
y_proba = []
y_pred = []

# Disable gradient calculation
with torch.no_grad():
    for x, y in tqdm(test_loader):
        # Forward pass
        output = model(x.to(device))

        # Store true labels
        y_true.extend(y.numpy())

        # Get predicted probabilities (softmax output)
        probabilities = torch.softmax(output, dim=1) # Assuming it's a multi-class classification
        y_proba.extend(probabilities.cpu().numpy()) # Store probabilities

        # Get predicted classes (argmax for final class prediction)
        pred = torch.argmax(probabilities, axis=1).cpu().numpy()
        y_pred.extend(pred)

# Convert to numpy arrays for further processing
y_true = np.array(y_true)
y_proba = np.array(y_proba)
y_pred = np.array(y_pred)

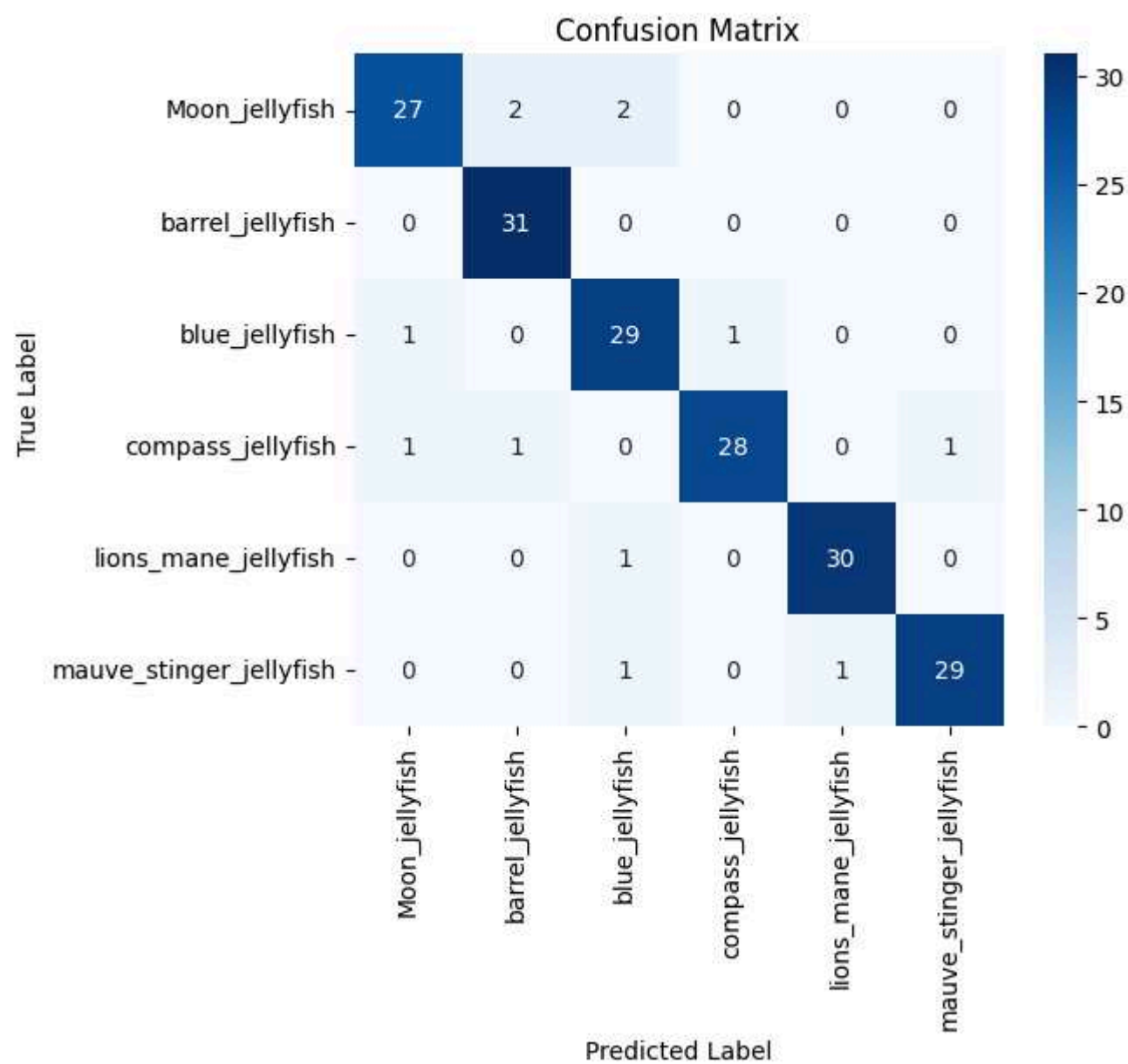
# Evaluation call
evaluate_model("efficientnet_b0", y_true, y_pred, classes)
# Note: The evaluation call changes based on the model being tested.

```

100%|██████████| 6/6 [00:06<00:00, 1.14s/it]

Accuracy: 0.9354838709677419
Precision 0.9369188184931168
Recall: 0.9354838709677419
F1-score: 0.9353318491324176
Classification Report:

	precision	recall	f1-score	support
0	0.93	0.87	0.90	31
1	0.91	1.00	0.95	31
2	0.88	0.94	0.91	31
3	0.97	0.90	0.93	31
4	0.97	0.97	0.97	31
5	0.97	0.94	0.95	31
accuracy			0.94	186
macro avg	0.94	0.94	0.94	186
weighted avg	0.94	0.94	0.94	186



```
In [23]: from torchvision import models
import torch.nn as nn
import torch.optim as optim

# Initialize ResNet-18 from scratch (no pretrained weights)
model = models.resnet18(pretrained=False)
# Replace the final fully connected layer to match the number of output classes
model.fc = nn.Linear(model.fc.in_features, num_classes)
model.to(device)

# Loss function and optimizer
criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)

# Learning rate scheduler
exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.97)

# Train the model (Note: num_epochs is set to 2 in the lecture)
model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=2)
y_true = []
y_proba = []
y_pred = []
# Disable gradient calculation
with torch.no_grad():
    for x, y in tqdm(test_loader):
        # Forward pass
        output = model(x.to(device))

        # Store true labels
        y_true.extend(y.numpy())

        # Get predicted probabilities (softmax output)
        probabilities = torch.softmax(output, dim=1)
        # Assuming it's a multi-class classification
        y_proba.extend(probabilities.cpu().numpy()) # Store probabilities

        # Get predicted classes (argmax for final class prediction)
        pred = torch.argmax(probabilities, axis=1).cpu().numpy()
        y_pred.extend(pred)
# Convert to numpy arrays for further processing
y_true = np.array(y_true)
y_proba = np.array(y_proba)
y_pred = np.array(y_pred)
evaluate_model("Resnet18", y_true, y_pred, classes)
```

Epoch 1 started at: 2025-11-14 21:35:53
Epoch 1/2

train Loss: 1.7475 Acc: 0.3221
val Loss: 18.4013 Acc: 0.2222
Epoch 1 finished at: 2025-11-14 21:36:09
Epoch 1 duration: 0.27 minutes
=====

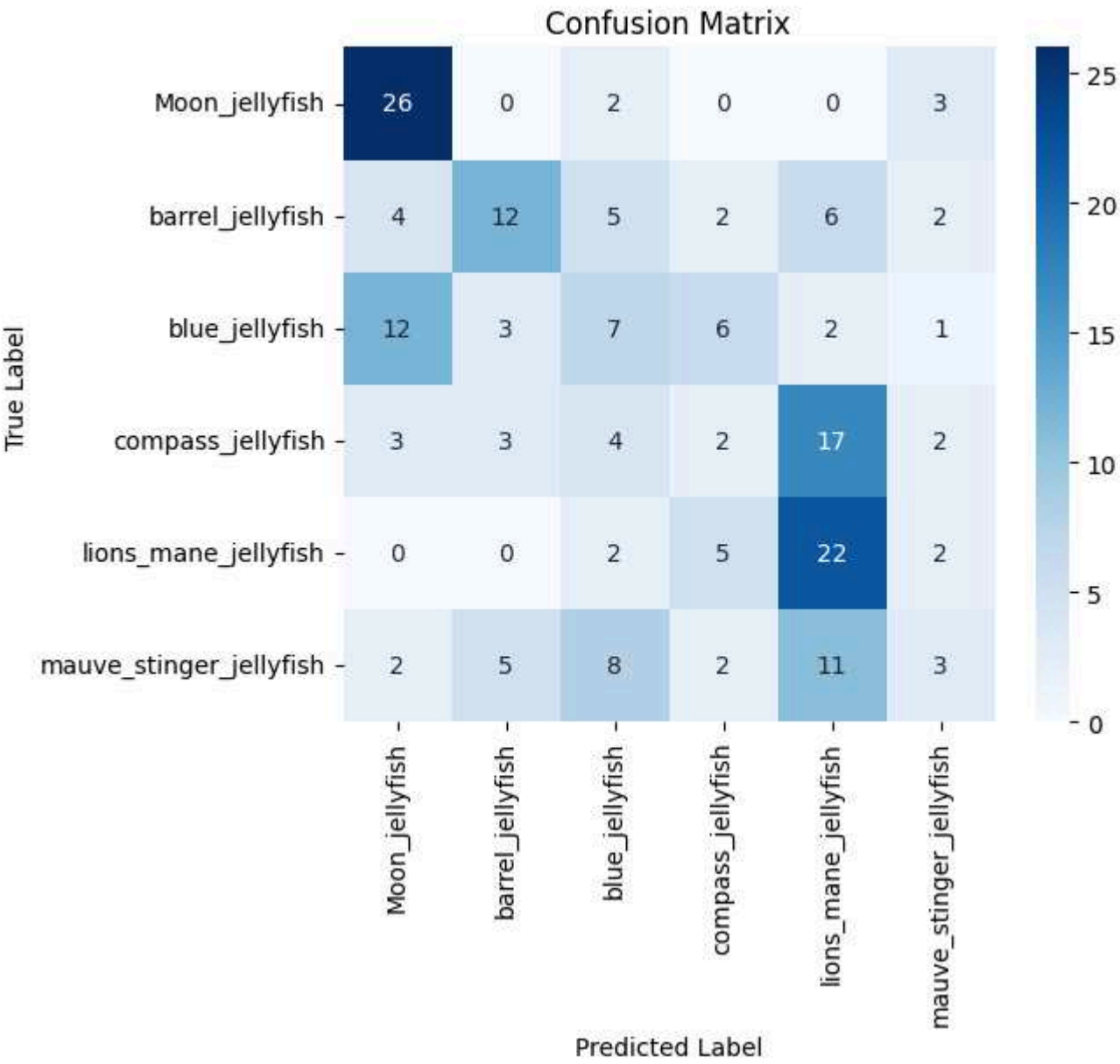
Epoch 2 started at: 2025-11-14 21:36:09
Epoch 2/2

train Loss: 1.4779 Acc: 0.4087
val Loss: 1.3832 Acc: 0.4556
Epoch 2 finished at: 2025-11-14 21:36:25
Epoch 2 duration: 0.27 minutes
=====

100%|██████████| 6/6 [00:06<00:00, 1.05s/it]

Accuracy: 0.3870967741935484
Precision 0.3421095423694719
Recall: 0.3870967741935484
F1-score: 0.3437463731455352
Classification Report:

	precision	recall	f1-score	support
0	0.55	0.84	0.67	31
1	0.52	0.39	0.44	31
2	0.25	0.23	0.24	31
3	0.12	0.06	0.08	31
4	0.38	0.71	0.49	31
5	0.23	0.10	0.14	31
accuracy			0.39	186
macro avg	0.34	0.39	0.34	186
weighted avg	0.34	0.39	0.34	186



```
In [25]: #Models Setup and Training DenseNet121 Setup
model = models.densenet121(pretrained=False)
# Replace the classifier to match the number of output classes
model.classifier = nn.Linear(model.classifier.in_features, num_classes)
model = model.to(device)

# Loss function and optimizer
criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)

# Learning rate scheduler
exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.97)

# Train the model (Note: num_epochs is set to 1 in the Lecture)
model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=2)
y_true = []
y_proba = []
y_pred = []
# Disable gradient calculation
```

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

        # Store true labels
        y_true.extend(y.numpy())

        # Get predicted probabilities (softmax output)
        probabilities = torch.softmax(output, dim=1)
        # Assuming it's a multi-class classification
        y_proba.extend(probabilities.cpu().numpy()) # Store probabilities

        # Get predicted classes (argmax for final class prediction)
        pred = torch.argmax(probabilities, axis=1).cpu().numpy()
        y_pred.extend(pred)
# Convert to numpy arrays for further processing
y_true = np.array(y_true)
y_proba = np.array(y_proba)
y_pred = np.array(y_pred)
evaluate_model("DenseNet121", y_true, y_pred, classes)
```

Epoch 1 started at: 2025-11-14 21:42:36

Epoch 1/2

train Loss: 1.6422 Acc: 0.3397

val Loss: 5.1127 Acc: 0.2444

Epoch 1 finished at: 2025-11-14 21:45:36

Epoch 1 duration: 3.00 minutes

=====

Epoch 2 started at: 2025-11-14 21:45:36

Epoch 2/2

train Loss: 1.4293 Acc: 0.4231

val Loss: 1.5156 Acc: 0.3556

Epoch 2 finished at: 2025-11-14 21:48:36

Epoch 2 duration: 3.00 minutes

=====

100%|██████████| 6/6 [00:28<00:00, 4.73s/it]

Accuracy: 0.43010752688172044

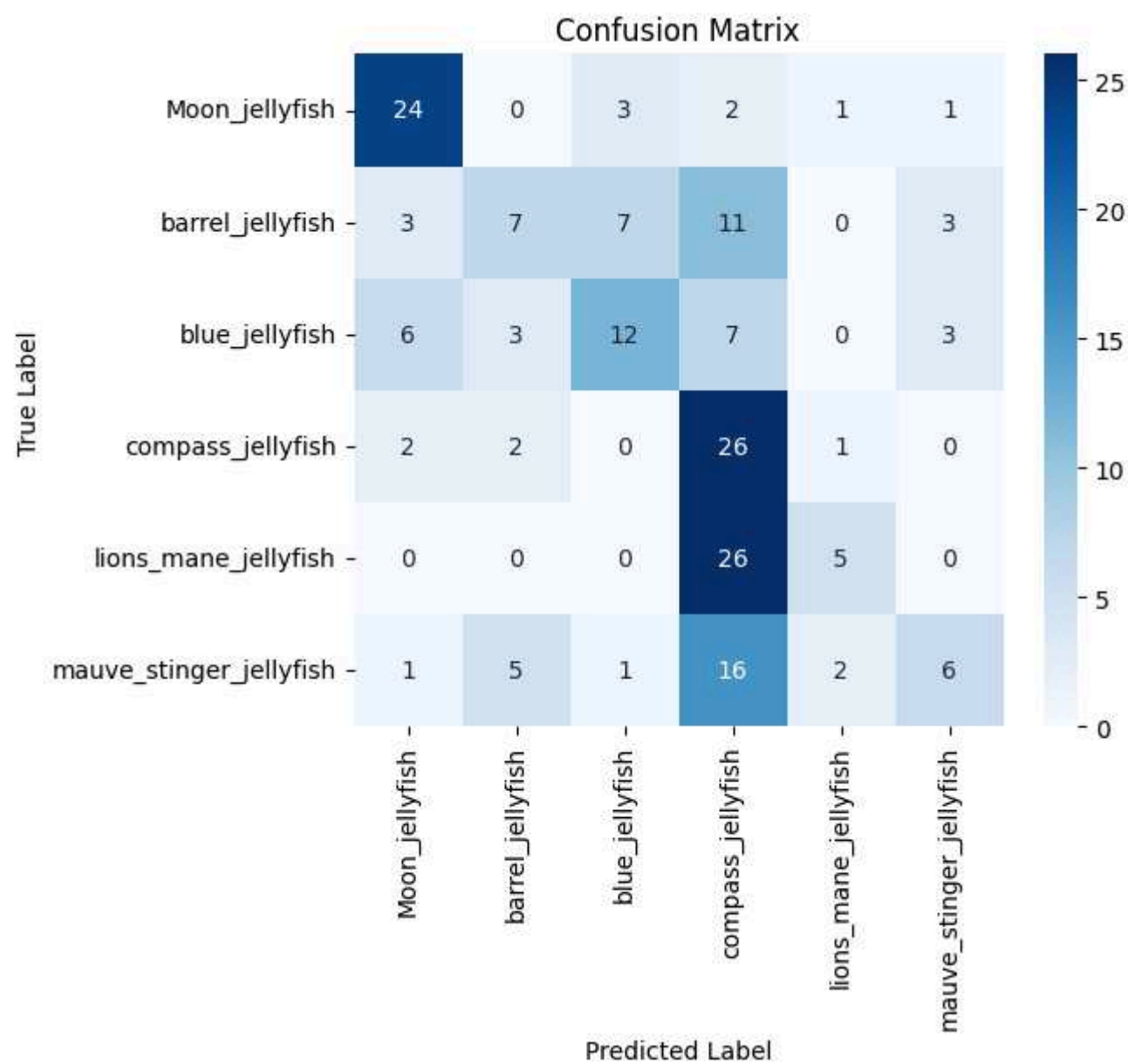
Precision 0.48545317758872747

Recall: 0.43010752688172044

F1-score: 0.40203851403368523

Classification Report:

	precision	recall	f1-score	support
0	0.67	0.77	0.72	31
1	0.41	0.23	0.29	31
2	0.52	0.39	0.44	31
3	0.30	0.84	0.44	31
4	0.56	0.16	0.25	31
5	0.46	0.19	0.27	31
accuracy			0.43	186
macro avg	0.49	0.43	0.40	186
weighted avg	0.49	0.43	0.40	186



```
In [26]: # Initialize VGG16 from scratch (no pretrained weights)
model = models.vgg16(pretrained=False)
# Replace the classifier to match the number of output classes
model.classifier[6] = nn.Linear(model.classifier[6].in_features, num_classes)
model = model.to(device)

# Loss function and optimizer
criterion = nn.CrossEntropyLoss()
criterion = criterion.to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)

# Learning rate scheduler
exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.97)

# Train the model (Note: num_epochs is set to 1 in the Lecture)
model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=2)
y_true = []
y_proba = []
y_pred = []
# Disable gradient calculation
with torch.no_grad():
    for x, y in tqdm(test_loader):
        # Forward pass
        output = model(x.to(device))

        # Store true labels
        y_true.extend(y.numpy())

        # Get predicted probabilities (softmax output)
        probabilities = torch.softmax(output, dim=1)
        # Assuming it's a multi-class classification
        y_proba.extend(probabilities.cpu().numpy()) # Store probabilities

        # Get predicted classes (argmax for final class prediction)
        pred = torch.argmax(probabilities, axis=1).cpu().numpy()
        y_pred.extend(pred)
# Convert to numpy arrays for further processing
y_true = np.array(y_true)
y_proba = np.array(y_proba)
y_pred = np.array(y_pred)
evaluate_model("VGG16", y_true, y_pred, classes)
```

Epoch 1 started at: 2025-11-14 21:49:59
Epoch 1/2

train Loss: 29.7542 Acc: 0.1603
val Loss: 1.7931 Acc: 0.1667
Epoch 1 finished at: 2025-11-14 21:58:46
Epoch 1 duration: 8.80 minutes
=====

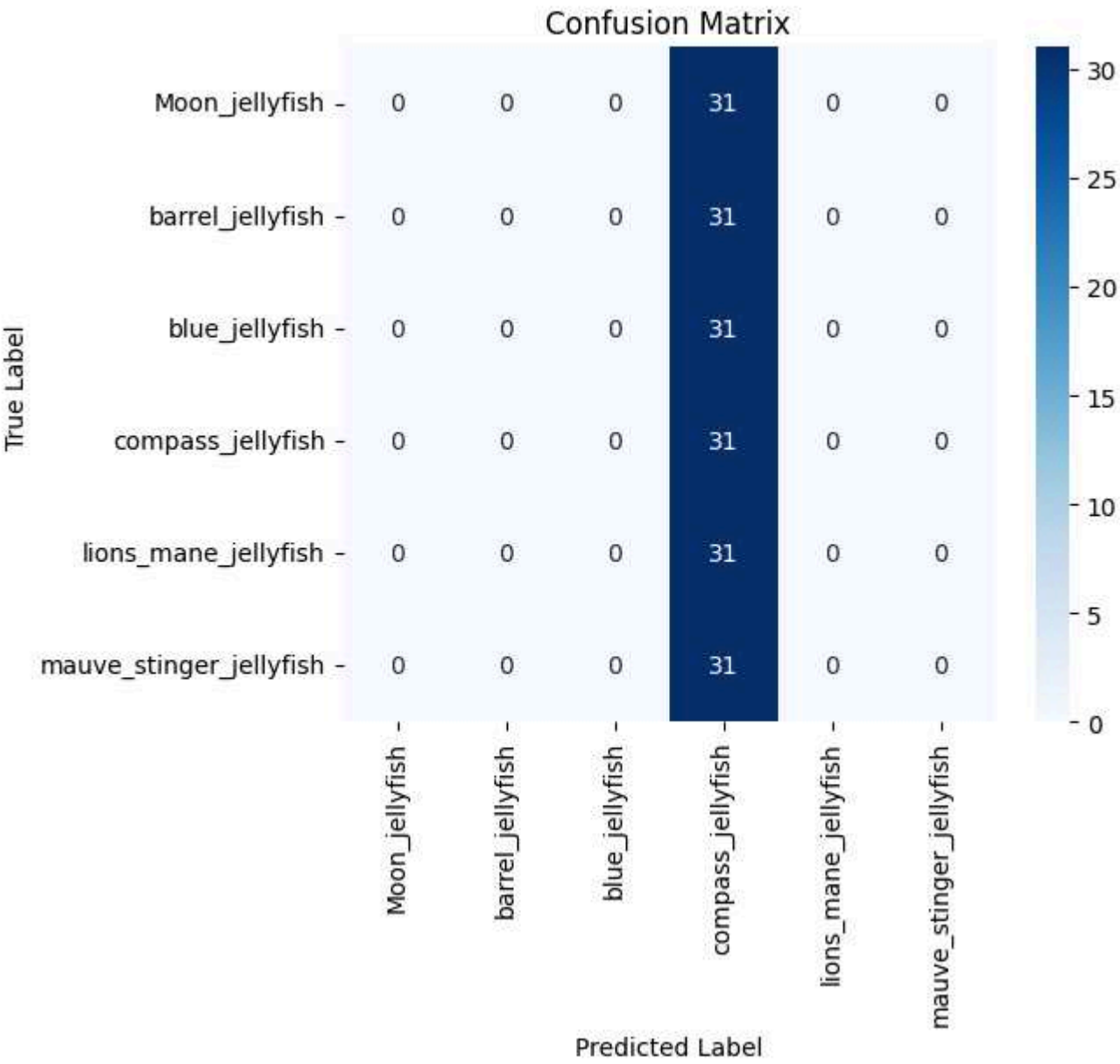
Epoch 2 started at: 2025-11-14 21:58:46
Epoch 2/2

train Loss: 1.7938 Acc: 0.1603
val Loss: 1.7921 Acc: 0.1667
Epoch 2 finished at: 2025-11-14 22:07:30
Epoch 2 duration: 8.73 minutes
=====

100%|██████████| 6/6 [00:55<00:00, 9.25s/it]

Accuracy: 0.16666666666666666
Precision 0.027777777777777776
Recall: 0.16666666666666666
F1-score: 0.04761904761904761
Classification Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	31
1	0.00	0.00	0.00	31
2	0.00	0.00	0.00	31
3	0.17	1.00	0.29	31
4	0.00	0.00	0.00	31
5	0.00	0.00	0.00	31
accuracy			0.17	186
macro avg	0.03	0.17	0.05	186
weighted avg	0.03	0.17	0.05	186



```
In [27]: # Load InceptionV3 model without pretraining
model = models.inception_v3(pretrained=False, aux_logits=True)
# Replace the final fully connected layer to match the number of output classes
model.fc = nn.Linear(model.fc.in_features, num_classes)
# Also replace the auxiliary classifier's fully connected layer (if used)
if model.aux_logits:
    model.AuxLogits.fc = nn.Linear(model.AuxLogits.fc.in_features, num_classes)
model = model.to(device)

# Loss function and optimizer
criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.AdamW(model.parameters(), lr=0.001)

# Learning rate scheduler
exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.97)

# Train the model (Note: num_epochs is set to 1 in the Lecture)
model = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=2)
y_true = []
y_proba = []
```

```
y_pred = []
# Disable gradient calculation
with torch.no_grad():
    for x, y in tqdm(test_loader):
        # Forward pass
        output = model(x.to(device))

        # Store true labels
        y_true.extend(y.numpy())

        # Get predicted probabilities (softmax output)
        probabilities = torch.softmax(output, dim=1)
        # Assuming it's a multi-class classification
        y_proba.extend(probabilities.cpu().numpy()) # Store probabilities

        # Get predicted classes (argmax for final class prediction)
        pred = torch.argmax(probabilities, axis=1).cpu().numpy()
        y_pred.extend(pred)
# Convert to numpy arrays for further processing
y_true = np.array(y_true)
y_proba = np.array(y_proba)
y_pred = np.array(y_pred)
evaluate_model("InceptionV3", y_true, y_pred, classes)
```

Epoch 1 started at: 2025-11-14 22:09:04

Epoch 1/2

train Loss: 2.3811 Acc: 0.2804

val Loss: 6.1176 Acc: 0.2333

Epoch 1 finished at: 2025-11-14 22:10:07

Epoch 1 duration: 1.04 minutes

=====

Epoch 2 started at: 2025-11-14 22:10:07

Epoch 2/2

train Loss: 2.0500 Acc: 0.4151

val Loss: 1.6430 Acc: 0.4333

Epoch 2 finished at: 2025-11-14 22:11:08

Epoch 2 duration: 1.02 minutes

=====

100%|██████████| 6/6 [00:11<00:00, 1.88s/it]

Accuracy: 0.41397849462365593

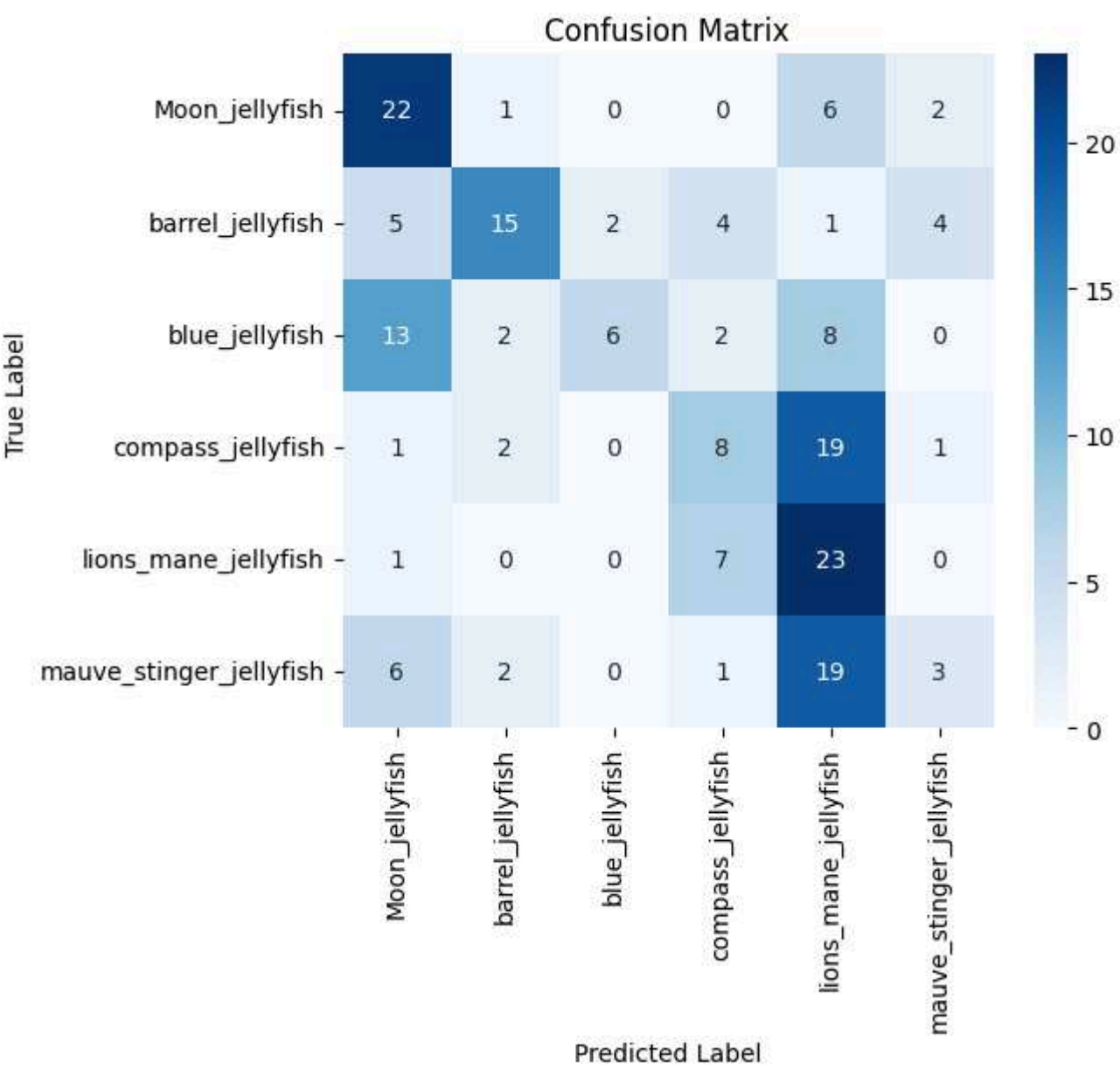
Precision 0.47606990962254125

Recall: 0.41397849462365593

F1-score: 0.38480447779689314

Classification Report:

	precision	recall	f1-score	support
0	0.46	0.71	0.56	31
1	0.68	0.48	0.57	31
2	0.75	0.19	0.31	31
3	0.36	0.26	0.30	31
4	0.30	0.74	0.43	31
5	0.30	0.10	0.15	31
accuracy			0.41	186
macro avg	0.48	0.41	0.38	186
weighted avg	0.48	0.41	0.38	186



```
In [ ]:
```