Sea Animals Classification

March 26, 2023

1 Sea Animals Classification

1.0.1 Team Members

- **Kshitij Yerande - 40194579**
- **Siddhartha Jha - 40201472**

1.0.2 Problem Statement

The goal of the project is to analyze the performance of the various machine learning approaches to solve image classification problem of classifying sea animals into 9 categories Corals, Crabs, Penguin, Sea Urchins, Seahorse, Seal, Sharks, Starfish,Turtle_Tortoise.

1.0.3 Dataset

The dataset is retrieved from kaggle.

Link: https://www.kaggle.com/datasets/vencerlanz09/sea-animals-image-dataste

1.0.4 Evaluation metrics

1. Accuracy: It is the most intuitive performance measure, and it is simply the ratio of correctly predicted observation to the total observations.
2. Precision: It is the ratio of the correctly predicted positive observations.
3. Recall: It is the ratio of correctly predicted positive observations to all observations in actual class.
4. F-score: It is the weighted average of precision and recall.

1.0.5 Solution approaches

- Custom-CNN model: CNN architecture desined from scratch
- ResNet50 (Pre-trained) : Pretrained weights of ImageNet dataset used to train last layer on our dataset.
- SVM with HOG features(histogram of gradients) : HOG feature extracted and trained using SVM
- SVM with SIFT features: SIFT features extracted and trained using SVM

[1]:
```
import numpy as np
import pandas as pd
from tqdm import tqdm
```

```python
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
import torch.nn.functional as F

from torchvision.utils import make_grid
from torchvision.datasets import ImageFolder
from torchvision.models import resnet50, vgg16_bn
import torchvision.transforms as transforms


from torch.utils.data import random_split, SubsetRandomSampler,␣
 ↪DataLoader,Subset, WeightedRandomSampler
from torch.optim import Adam

from collections import Counter
import itertools

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, precision_score,␣
 ↪recall_score, confusion_matrix
from torchviz import make_dot
import cv2
```

[2]:
```python
# Use GPU when possible
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)

# CONSTANTS
DATASET_DIR = 'data/archive'
NUM_CLASSES = 9 # no of classes in the garbage dataset
BATCH_SIZE = 500 # for dataloader
```

```
cpu
```

1.0.6 Data Preprocessing - Applying Transformations

[3]:
```python
# Resize images and normalize pixel values before passing it to the model
transformations = transforms.Compose([transforms.Resize((128,128)),
                                      transforms.ToTensor(),
                                      transforms.Normalize((0.5, 0.5, 0.5), (0.
 ↪5, 0.5, 0.5))
                                     ])

dataset = ImageFolder(DATASET_DIR, transform = transformations)
```

```
CLASSES = dataset.class_to_idx.keys()
TEST_CLASSES = dataset.class_to_idx.keys()
print('Total data',len(dataset))
print('Classes')
display(dataset.class_to_idx)
```

```
Total data 5944
Classes
```

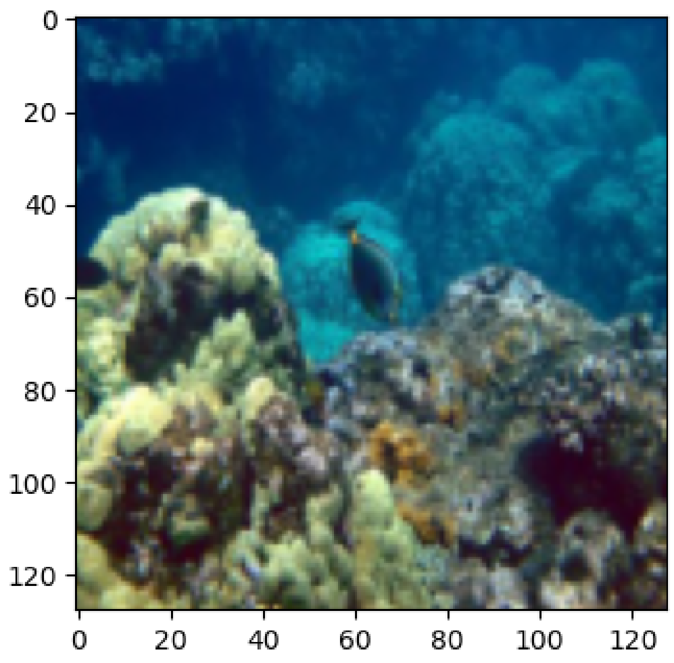
```
{'Corals': 0,
 'Crabs': 1,
 'Penguin': 2,
 'Sea Urchins': 3,
 'Seahorse': 4,
 'Seal': 5,
 'Sharks': 6,
 'Starfish': 7,
 'Turtle_Tortoise': 8}
```

1.0.7 Visualizing Data

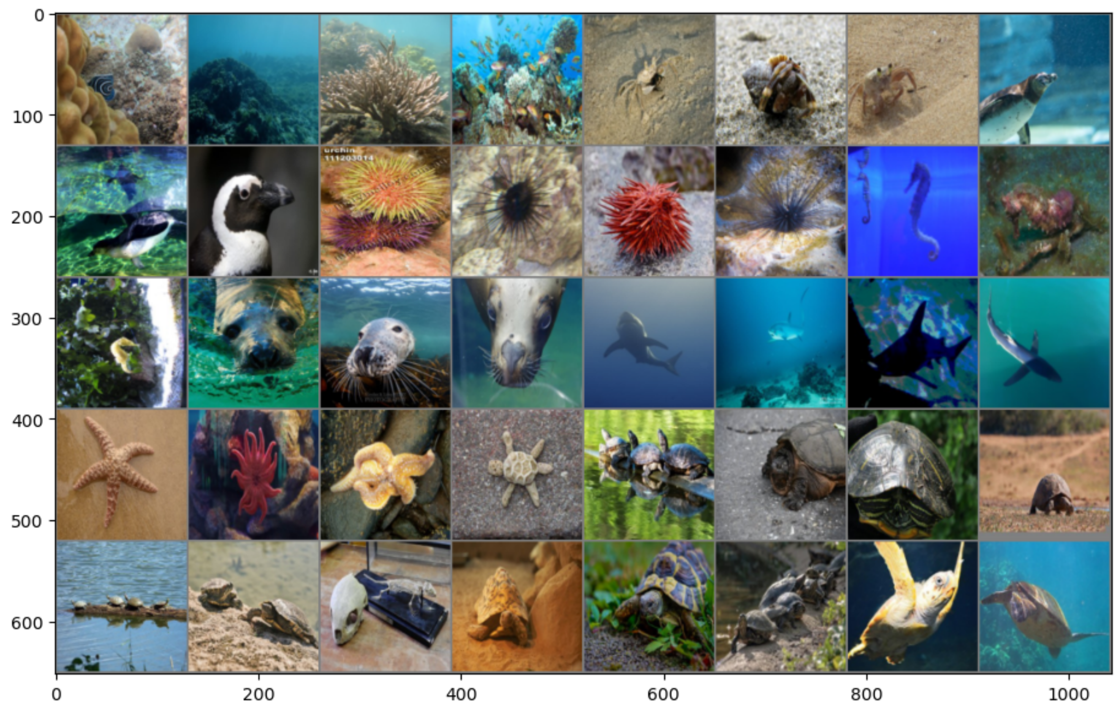
[4]:
```
# utility method to print class label(s) and show image(s) of a row in the␣
 ↪dataset
def imshow(image):
    plt.figure(figsize=(14,7))
    if not isinstance(image,torch.Tensor):
        plt.figure(figsize=(4,4))
        print(f'Image sampled from class {image[1]} - "{dataset.
 ↪classes[image[1]]}"')
        npimg = np.array(image[0])
    else:
        plt.figure(figsize=(14,7))
        npimg = image.numpy()
    npimg = npimg / 2 + 0.5
    plt.imshow(np.transpose(npimg, (1, 2, 0)))

imshow(dataset[10])
imshow(make_grid([dataset[idx][0] for idx in range(1,len(dataset),150)]))
```

```
Image sampled from class 0 - "Corals"
```

```
<Figure size 1400x700 with 0 Axes>
```


0
20
40
60
80
100
120
0
20
40
60
80
100
120

```
<Figure size 1400x700 with 0 Axes>
```


0
100
200
300
400
500
600
0
200
400
600
800
1000

[6]:
```
all_idxs = np.arange(len(dataset))

# perform a 70/30 train-test split
train_idxs, test_idxs = train_test_split(all_idxs, test_size=0.
 ↪3,stratify=dataset.targets, random_state=0)

# get the train and test set targets/labels
train_labels = np.array(dataset.targets)[train_idxs]
test_labels = np.array(dataset.targets)[test_idxs]

# set the test set's dataloader
test_set_loader = DataLoader(Subset(dataset, test_idxs), batch_size=BATCH_SIZE,␣
 ↪shuffle=True, drop_last=True)

print('train dataset',len(train_idxs))
print('test dataset',len(test_idxs))
```

```
train dataset 4160
test dataset 1784
```

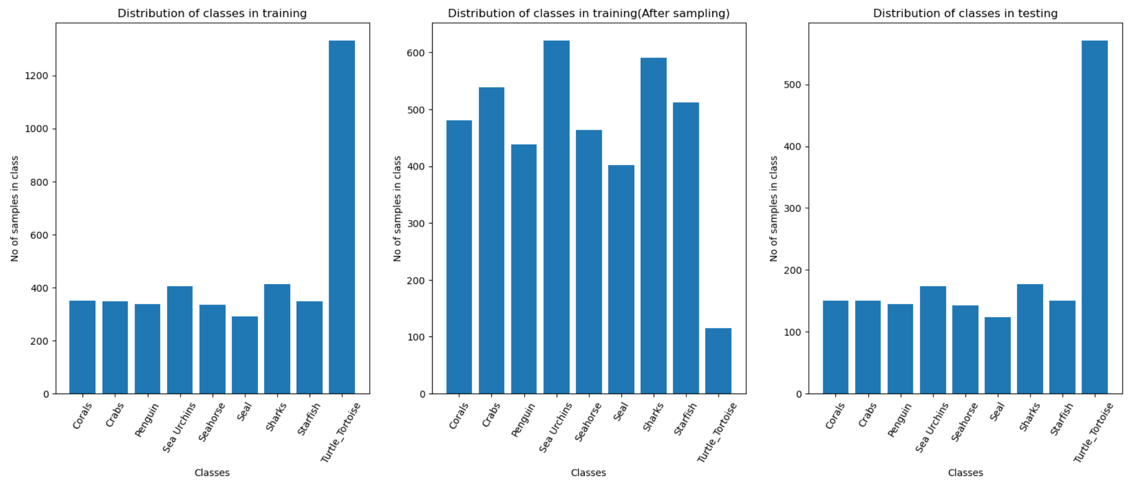
1.0.8 Balancing Training Data

[7]:
```
# Perform oversampling of training data to make the dataset balanced
unique, count = np.unique(train_labels,return_counts=True)
print("Unique Labels:",len(unique))

class_weights = [np.sum(count) / c for c in count]
print("Class Weights:",class_weights)

#Assign weight to each sample
example_weights = [class_weights[e] for e in train_labels]
sampler = WeightedRandomSampler(example_weights,len(train_labels))
train_sampled_dataloader = DataLoader(dataset, batch_size=BATCH_SIZE,␣
 ↪sampler=sampler)
```

```
Unique Labels: 9
Class Weights: [11.885714285714286, 11.919770773638968, 12.344213649851632,
10.271604938271604, 12.417910447761194, 14.344827586206897, 10.072639225181598,
11.919770773638968, 3.123123123123123]
```

1.0.9 Visualising Data Distribution - After OverSampling

[8]:
```
# Plot the distribution of classes in train and test sets
plt.figure(figsize=(20,7))
plt.subplot(1,3,1)
plt.title('Distribution of classes in training')
plt.xlabel('Classes')
plt.ylabel('No of samples in class')
plt.xticks(np.arange(NUM_CLASSES),CLASSES, rotation = 60)
train_labels_dist = Counter(train_labels)
plt.bar(train_labels_dist.keys(), train_labels_dist.values())

plt.subplot(1,3,2)
plt.title('Distribution of classes in training(After sampling)')
plt.xlabel('Classes')
plt.ylabel('No of samples in class')
temp_labels=[]
for data,labels in train_sampled_dataloader:
    for c in labels.numpy():
        temp_labels.append(c)

sampled_train_label = Counter(temp_labels)
plt.xticks(np.arange(NUM_CLASSES),CLASSES, rotation = 60)
plt.bar(sampled_train_label.keys(), sampled_train_label.values())


plt.subplot(1,3,3)
plt.title('Distribution of classes in testing')
plt.xlabel('Classes')
plt.ylabel('No of samples in class')
plt.xticks(np.arange(NUM_CLASSES),CLASSES, rotation = 60)
test_labels_dist = Counter(test_labels)
plt.bar(test_labels_dist.keys(), test_labels_dist.values());
```



1.0.10 CNN Model Architectures:

Custom-CNN Model Custom CNN model with 6 layers: 3 convolution layers and 3 linear layers. The first two convolution layers have a pooling layer and all 3 convolution layers uses batch normalization for training. ReLU activation function is used in all layers.

- 1st layer: uses convolution on 128x128 image with kernel size 11x11 depth 16 with zero padding followed by pooling layer of size 3x3 and stride 2
- 2nd layer: input of previous layer with depth 32 annd kernel size 7x7 with zero padding followed by pooling layer of size 3x3 and stride 2

- 3rd layer: input of previous layer with depth 32 annd kernel size 3x3 with zero padding
- 4th,5th,6th layer: dense layer with output 256 -> 128 -> 9

ResNet50 Model ResNet-50 is a 50-layer convolutional neural network (48 convolutional layers, one MaxPool layer, and one average pool layer).

The 50-layer ResNet architecture includes the following elements, as shown in the table below:

- A 7×7 kernel convolution alongside 64 other kernels with a 2-sized stride.
- A max pooling layer with a 2-sized stride.
- 9 more layers—3×3,64 kernel convolution, another with 1×1,64 kernels, and a third with 1×1,256 kernels. These 3 layers are repeated 3 times.
- 12 more layers with 1×1,128 kernels, 3×3,128 kernels, and 1×1,512 kernels, iterated 4 times.
- 18 more layers with 1×1,256 cores, and 2 cores 3×3,256 and 1×1,1024, iterated 6 times.
- 9 more layers with 1×1,512 cores, 3×3,512 cores, and 1×1,2048 cores iterated 3 times. (up to this point the network has 50 layers)
- Average pooling, followed by a fully connected layer with 1000 nodes, using the softmax activation function.

[8]:
```python
class CustomCNN(nn.Module):
    def __init__(self, num_classes=9):
        super(CustomCNN, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 16, kernel_size=11, stride=1, padding=0),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 3, stride = 2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=7, stride=1, padding=0),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 3, stride = 2))
        self.layer3 = nn.Sequential(
            nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=0),
            nn.BatchNorm2d(32),
            nn.ReLU())
        self.fc = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(16928, 256),
            nn.ReLU())
        self.fc1 = nn.Sequential(
            nn.Dropout(0.5),
                nn.Linear(256, 128),
            nn.ReLU())
        self.fc2= nn.Sequential(
            nn.Linear(128, num_classes))

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        out = self.fc1(out)
        out = self.fc2(out)
        return out

class ResNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.nn = resnet50(pretrained=True)

        # do not allow backpropagation through
        # pre-trained model layers
        for param in self.nn.parameters():
            param.requires_grad = False

        self.nn.fc = nn.Sequential(nn.Linear(2048, 512),
                                   nn.ReLU(),
                                   nn.Dropout(0.2),
                                   nn.Linear(512, NUM_CLASSES))
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        return self.sigmoid(self.nn(x))
```

1.0.11 Methods for Evaluation of Model

[9]:
```
@torch.no_grad()
def evaluate_model(model,loader):
    model.to('cpu')
    # set model to evaluation mode before evaluation
    model.eval()
    predictions = torch.tensor([])
    targets = torch.tensor([])
    for _, (images, labels) in enumerate(loader):
        out = model(images)
        _, predicted = torch.max(out.data, 1)
        targets = torch.cat((targets, labels))
        predictions = torch.cat((predictions, predicted))

    accuracy = accuracy_score(targets, predictions) * 100
    f1 = f1_score(targets, predictions, average='macro')
    recall = recall_score(targets, predictions, average='macro')
    precision = precision_score(targets, predictions, average='macro')
    conf_matrix = confusion_matrix(targets, predictions)
    return conf_matrix,[accuracy, f1, recall, precision]

def pretty_print_metrics(metrics):
    print('-- TEST SET METRICS --')
    print(f'Accuracy: {metrics[0]:.3f}%\nF-score: {metrics[1]:.3f}\nRecall:␣
↪{metrics[2]:.3f}\nPrecision: {metrics[3]:.3f}')

def run_testing(model_to_test):
    model =None
    if model_to_test=="CustomCNN":
        model = CustomCNN()
    if model_to_test=="ResNet":
        model = ResNet()
    model.load_state_dict(torch.load(f'models/{model_to_test}-model.
↪pt'),strict=False)
    model_to_test_conf_mat, model_to_test_metrics = evaluate_model(model,␣
↪test_set_loader)
    pretty_print_metrics(model_to_test_metrics)
    plot_cm(np.array(model_to_test_conf_mat), CLASSES)
```

1.0.12 Plotting Utilities

[10]:
```
# Utility method to plot the confusion matrix over k-folds
def plot_cm(cm, classes, normalize=False, title='Visualization of the confusion␣
↪matrix', cmap=plt.cm.Reds):
    plt.figure(figsize=(14,7))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("confusion matrix")
    else:
        print('Confusion matrix')

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()

    plt.ylabel('Actual True label')
    plt.xlabel('Predicted label')

# Utility method to plot both accuracy vs num epochs and loss vs num epochs
def plot_training_metrics(training_metrics_df):
    fig, ax = plt.subplots(2, figsize=(20,10))

    ax[0].set_xlabel('Number of epochs')
    ax[1].set_xlabel('Number of epochs')
    ax[0].set_ylabel('Training accuracy %')
    ax[1].set_ylabel('Training loss')

    ax[0].set_title('Training Accuracy vs Number of Epochs')
    ax[1].set_title('Training Loss vs Number of Epochs')
    ax[0].plot(training_metrics_df.epoch.values, training_metrics_df.accuracy.
 ↪values)
    ax[1].plot(training_metrics_df.epoch.values, training_metrics_df.loss.
 ↪values)
```

1.0.13 Method for training model

[11]:
```
# the actual training loop given a model, data loader, optimizer
def training_loop(model, train_loader, optimizer, total_step,name):
    epoch_metrics = []

    for epoch in range(NUM_EPOCHS):
        loss_val = 0.0
        acc_val = 0.0
        for i, (images, labels) in enumerate(train_loader):
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            loss = LOSS_FN(outputs, labels)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

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

            loss_val = loss.item()
            acc_val = (correct / total) * 100

            epoch_metrics.append({'epoch':epoch + (i/total_step),'loss':
 ↪loss_val, 'accuracy':acc_val})

            print(f'Epoch [{epoch + 1}/{NUM_EPOCHS}], Step [{i + 1}/
 ↪{total_step}], Loss: {loss.item():.4f}, Accuracy: {(correct / total) * 100:.
 ↪2f}')

    torch.save(model.state_dict(), f'models/{name}-model.pt')
    return epoch_metrics
```

[12]:
```
def train(model,save):
    epoch_metrics_df = pd.DataFrame(columns=['epoch','loss','accuracy'])
    model.to(device)
    optimizer = Adam(model.parameters(), lr=LEARNING_RATE)
    # run training
    epoch_metrics = training_loop(model, train_sampled_dataloader, optimizer,␣
 ↪len(train_sampled_dataloader),save)
    epoch_metrics_df = epoch_metrics_df.append(epoch_metrics, ignore_index=True)
    epoch_metrics_df.to_pickle(f'metric/{save}-metric.pkl')
    return epoch_metrics_df
```

1.0.14 Training Data using ResNet50(Pre-Trained) Model

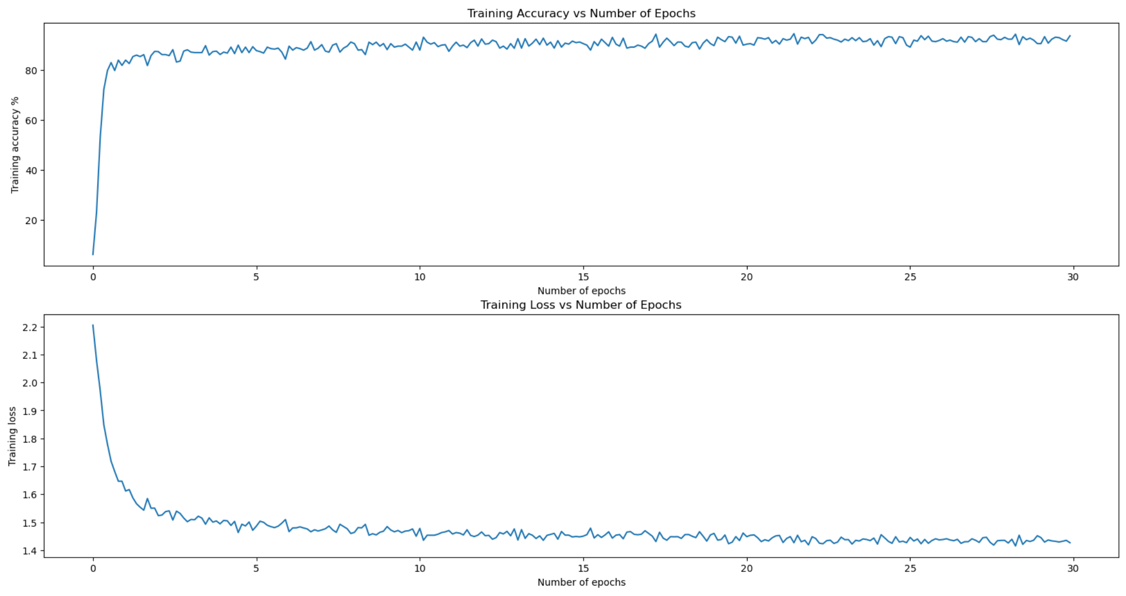
[13]:
```
LEARNING_RATE = 0.001
NUM_EPOCHS = 30
LOSS_FN = nn.CrossEntropyLoss()

#epoch_metrics_df =train(ResNet(),'ResNet')
```

```
C:\Users\kshit\anaconda3\lib\site-packages\torchvision\models\_utils.py:208:
UserWarning: The parameter 'pretrained' is deprecated since 0.13 and will be
removed in 0.15, please use 'weights' instead.
  warnings.warn(
C:\Users\kshit\anaconda3\lib\site-packages\torchvision\models\_utils.py:223:
UserWarning: Arguments other than a weight enum or `None` for 'weights' are
deprecated since 0.13 and will be removed in 0.15. The current behavior is
equivalent to passing `weights=ResNet50_Weights.IMAGENET1K_V1`. You can also use
`weights=ResNet50_Weights.DEFAULT` to get the most up-to-date weights.
```

```
  warnings.warn(msg)
Epoch [1/30], Step [1/9], Loss: 2.2047, Accuracy: 6.00
Epoch [1/30], Step [2/9], Loss: 2.0781, Accuracy: 23.00
Epoch [1/30], Step [3/9], Loss: 1.9713, Accuracy: 52.80
Epoch [1/30], Step [4/9], Loss: 1.8485, Accuracy: 72.20
Epoch [1/30], Step [5/9], Loss: 1.7793, Accuracy: 79.80
Epoch [1/30], Step [6/9], Loss: 1.7183, Accuracy: 83.00
Epoch [1/30], Step [7/9], Loss: 1.6814, Accuracy: 79.80
Epoch [1/30], Step [8/9], Loss: 1.6469, Accuracy: 84.00
Epoch [1/30], Step [9/9], Loss: 1.6468, Accuracy: 81.88
Epoch [2/30], Step [1/9], Loss: 1.6117, Accuracy: 84.00
Epoch [2/30], Step [2/9], Loss: 1.6169, Accuracy: 82.60
Epoch [2/30], Step [3/9], Loss: 1.5873, Accuracy: 85.40
Epoch [2/30], Step [4/9], Loss: 1.5663, Accuracy: 86.00
Epoch [2/30], Step [5/9], Loss: 1.5539, Accuracy: 85.40
Epoch [2/30], Step [6/9], Loss: 1.5434, Accuracy: 86.20
Epoch [2/30], Step [7/9], Loss: 1.5850, Accuracy: 81.80
Epoch [2/30], Step [8/9], Loss: 1.5503, Accuracy: 85.80
Epoch [2/30], Step [9/9], Loss: 1.5506, Accuracy: 87.50
Epoch [3/30], Step [1/9], Loss: 1.5238, Accuracy: 87.40
Epoch [3/30], Step [2/9], Loss: 1.5258, Accuracy: 86.20
Epoch [3/30], Step [3/9], Loss: 1.5383, Accuracy: 86.20
Epoch [3/30], Step [4/9], Loss: 1.5409, Accuracy: 85.80
Epoch [3/30], Step [5/9], Loss: 1.5079, Accuracy: 88.20
Epoch [3/30], Step [6/9], Loss: 1.5399, Accuracy: 83.20
Epoch [3/30], Step [7/9], Loss: 1.5317, Accuracy: 83.60
Epoch [3/30], Step [8/9], Loss: 1.5152, Accuracy: 87.60
Epoch [3/30], Step [9/9], Loss: 1.5021, Accuracy: 88.12
Epoch [4/30], Step [1/9], Loss: 1.5101, Accuracy: 87.20
Epoch [4/30], Step [2/9], Loss: 1.5091, Accuracy: 87.00
Epoch [4/30], Step [3/9], Loss: 1.5218, Accuracy: 87.00
Epoch [4/30], Step [4/9], Loss: 1.5150, Accuracy: 87.00
Epoch [4/30], Step [5/9], Loss: 1.4932, Accuracy: 89.80
Epoch [4/30], Step [6/9], Loss: 1.5160, Accuracy: 86.00
Epoch [4/30], Step [7/9], Loss: 1.5007, Accuracy: 87.40
Epoch [4/30], Step [8/9], Loss: 1.5047, Accuracy: 87.60
Epoch [4/30], Step [9/9], Loss: 1.4945, Accuracy: 86.25
Epoch [5/30], Step [1/9], Loss: 1.5066, Accuracy: 87.20
Epoch [5/30], Step [2/9], Loss: 1.5052, Accuracy: 86.80
Epoch [5/30], Step [3/9], Loss: 1.4887, Accuracy: 89.20
Epoch [5/30], Step [4/9], Loss: 1.5032, Accuracy: 86.60
Epoch [5/30], Step [5/9], Loss: 1.4634, Accuracy: 90.00
Epoch [5/30], Step [6/9], Loss: 1.4930, Accuracy: 87.00
Epoch [5/30], Step [7/9], Loss: 1.4871, Accuracy: 89.20
Epoch [5/30], Step [8/9], Loss: 1.5010, Accuracy: 87.00
Epoch [5/30], Step [9/9], Loss: 1.4714, Accuracy: 89.38
Epoch [6/30], Step [1/9], Loss: 1.4864, Accuracy: 87.80
Epoch [6/30], Step [2/9], Loss: 1.5038, Accuracy: 87.40
Epoch [6/30], Step [3/9], Loss: 1.4998, Accuracy: 86.80
Epoch [6/30], Step [4/9], Loss: 1.4895, Accuracy: 89.20
Epoch [6/30], Step [5/9], Loss: 1.4844, Accuracy: 88.60
Epoch [6/30], Step [6/9], Loss: 1.4810, Accuracy: 88.40
Epoch [6/30], Step [7/9], Loss: 1.4861, Accuracy: 88.80
Epoch [6/30], Step [8/9], Loss: 1.4968, Accuracy: 87.20
Epoch [6/30], Step [9/9], Loss: 1.5098, Accuracy: 84.38
Epoch [7/30], Step [1/9], Loss: 1.4665, Accuracy: 89.60
Epoch [7/30], Step [2/9], Loss: 1.4798, Accuracy: 88.00
Epoch [7/30], Step [3/9], Loss: 1.4801, Accuracy: 89.00
Epoch [7/30], Step [4/9], Loss: 1.4838, Accuracy: 88.60
Epoch [7/30], Step [5/9], Loss: 1.4797, Accuracy: 88.00
Epoch [7/30], Step [6/9], Loss: 1.4762, Accuracy: 88.80
Epoch [7/30], Step [7/9], Loss: 1.4661, Accuracy: 91.40
Epoch [7/30], Step [8/9], Loss: 1.4731, Accuracy: 88.00
Epoch [7/30], Step [9/9], Loss: 1.4689, Accuracy: 88.75
Epoch [8/30], Step [1/9], Loss: 1.4726, Accuracy: 90.20
Epoch [8/30], Step [2/9], Loss: 1.4770, Accuracy: 87.60
Epoch [8/30], Step [3/9], Loss: 1.4867, Accuracy: 87.20
Epoch [8/30], Step [4/9], Loss: 1.4729, Accuracy: 90.00
Epoch [8/30], Step [5/9], Loss: 1.4639, Accuracy: 90.60
Epoch [8/30], Step [6/9], Loss: 1.4929, Accuracy: 87.20
Epoch [8/30], Step [7/9], Loss: 1.4849, Accuracy: 88.80
Epoch [8/30], Step [8/9], Loss: 1.4769, Accuracy: 89.60
Epoch [8/30], Step [9/9], Loss: 1.4600, Accuracy: 91.25
Epoch [9/30], Step [1/9], Loss: 1.4640, Accuracy: 90.60
Epoch [9/30], Step [2/9], Loss: 1.4810, Accuracy: 88.00
Epoch [9/30], Step [3/9], Loss: 1.4802, Accuracy: 88.20
Epoch [9/30], Step [4/9], Loss: 1.4926, Accuracy: 86.20
Epoch [9/30], Step [5/9], Loss: 1.4537, Accuracy: 91.20
Epoch [9/30], Step [6/9], Loss: 1.4591, Accuracy: 90.20
Epoch [9/30], Step [7/9], Loss: 1.4550, Accuracy: 91.20
Epoch [9/30], Step [8/9], Loss: 1.4641, Accuracy: 89.60
Epoch [9/30], Step [9/9], Loss: 1.4683, Accuracy: 90.62
Epoch [10/30], Step [1/9], Loss: 1.4848, Accuracy: 88.20
Epoch [10/30], Step [2/9], Loss: 1.4724, Accuracy: 90.60
Epoch [10/30], Step [3/9], Loss: 1.4659, Accuracy: 89.20
Epoch [10/30], Step [4/9], Loss: 1.4708, Accuracy: 89.60
Epoch [10/30], Step [5/9], Loss: 1.4631, Accuracy: 89.60
Epoch [10/30], Step [6/9], Loss: 1.4685, Accuracy: 90.40
Epoch [10/30], Step [7/9], Loss: 1.4700, Accuracy: 89.20
Epoch [10/30], Step [8/9], Loss: 1.4763, Accuracy: 88.00
Epoch [10/30], Step [9/9], Loss: 1.4502, Accuracy: 91.25
Epoch [11/30], Step [1/9], Loss: 1.4781, Accuracy: 88.00
Epoch [11/30], Step [2/9], Loss: 1.4357, Accuracy: 93.20
Epoch [11/30], Step [3/9], Loss: 1.4537, Accuracy: 91.20
Epoch [11/30], Step [4/9], Loss: 1.4538, Accuracy: 90.40
Epoch [11/30], Step [5/9], Loss: 1.4537, Accuracy: 91.00
Epoch [11/30], Step [6/9], Loss: 1.4574, Accuracy: 89.40
Epoch [11/30], Step [7/9], Loss: 1.4632, Accuracy: 90.00
Epoch [11/30], Step [8/9], Loss: 1.4658, Accuracy: 90.20
Epoch [11/30], Step [9/9], Loss: 1.4705, Accuracy: 87.50
Epoch [12/30], Step [1/9], Loss: 1.4581, Accuracy: 89.60
Epoch [12/30], Step [2/9], Loss: 1.4633, Accuracy: 91.20
Epoch [12/30], Step [3/9], Loss: 1.4611, Accuracy: 89.60
Epoch [12/30], Step [4/9], Loss: 1.4548, Accuracy: 90.00
Epoch [12/30], Step [5/9], Loss: 1.4733, Accuracy: 89.00
Epoch [12/30], Step [6/9], Loss: 1.4531, Accuracy: 91.00
Epoch [12/30], Step [7/9], Loss: 1.4491, Accuracy: 92.00
Epoch [12/30], Step [8/9], Loss: 1.4541, Accuracy: 89.60
Epoch [12/30], Step [9/9], Loss: 1.4655, Accuracy: 92.50
Epoch [13/30], Step [1/9], Loss: 1.4520, Accuracy: 90.40
Epoch [13/30], Step [2/9], Loss: 1.4533, Accuracy: 90.60
Epoch [13/30], Step [3/9], Loss: 1.4394, Accuracy: 92.00
Epoch [13/30], Step [4/9], Loss: 1.4452, Accuracy: 91.40
Epoch [13/30], Step [5/9], Loss: 1.4633, Accuracy: 88.80
Epoch [13/30], Step [6/9], Loss: 1.4584, Accuracy: 89.60
Epoch [13/30], Step [7/9], Loss: 1.4676, Accuracy: 88.40
Epoch [13/30], Step [8/9], Loss: 1.4517, Accuracy: 90.60
Epoch [13/30], Step [9/9], Loss: 1.4763, Accuracy: 88.75
Epoch [14/30], Step [1/9], Loss: 1.4366, Accuracy: 92.60
Epoch [14/30], Step [2/9], Loss: 1.4737, Accuracy: 88.80
Epoch [14/30], Step [3/9], Loss: 1.4424, Accuracy: 92.60
Epoch [14/30], Step [4/9], Loss: 1.4594, Accuracy: 89.60
Epoch [14/30], Step [5/9], Loss: 1.4536, Accuracy: 90.80
Epoch [14/30], Step [6/9], Loss: 1.4421, Accuracy: 92.40
Epoch [14/30], Step [7/9], Loss: 1.4515, Accuracy: 90.20
Epoch [14/30], Step [8/9], Loss: 1.4355, Accuracy: 92.80
Epoch [14/30], Step [9/9], Loss: 1.4533, Accuracy: 90.00
Epoch [15/30], Step [1/9], Loss: 1.4566, Accuracy: 91.20
Epoch [15/30], Step [2/9], Loss: 1.4603, Accuracy: 88.80
Epoch [15/30], Step [3/9], Loss: 1.4395, Accuracy: 91.80
Epoch [15/30], Step [4/9], Loss: 1.4670, Accuracy: 89.20
Epoch [15/30], Step [5/9], Loss: 1.4541, Accuracy: 90.80
Epoch [15/30], Step [6/9], Loss: 1.4541, Accuracy: 90.40
Epoch [15/30], Step [7/9], Loss: 1.4476, Accuracy: 91.60
Epoch [15/30], Step [8/9], Loss: 1.4494, Accuracy: 91.00
Epoch [15/30], Step [9/9], Loss: 1.4479, Accuracy: 91.25
Epoch [16/30], Step [1/9], Loss: 1.4505, Accuracy: 90.60
Epoch [16/30], Step [2/9], Loss: 1.4559, Accuracy: 90.00
Epoch [16/30], Step [3/9], Loss: 1.4791, Accuracy: 88.00
Epoch [16/30], Step [4/9], Loss: 1.4438, Accuracy: 91.40
Epoch [16/30], Step [5/9], Loss: 1.4556, Accuracy: 89.80
Epoch [16/30], Step [6/9], Loss: 1.4458, Accuracy: 92.40
Epoch [16/30], Step [7/9], Loss: 1.4548, Accuracy: 91.00
Epoch [16/30], Step [8/9], Loss: 1.4661, Accuracy: 89.60
Epoch [16/30], Step [9/9], Loss: 1.4434, Accuracy: 93.12
Epoch [17/30], Step [1/9], Loss: 1.4541, Accuracy: 90.40
Epoch [17/30], Step [2/9], Loss: 1.4561, Accuracy: 89.60
Epoch [17/30], Step [3/9], Loss: 1.4411, Accuracy: 92.80
Epoch [17/30], Step [4/9], Loss: 1.4649, Accuracy: 88.80
Epoch [17/30], Step [5/9], Loss: 1.4666, Accuracy: 89.20
Epoch [17/30], Step [6/9], Loss: 1.4571, Accuracy: 89.20
Epoch [17/30], Step [7/9], Loss: 1.4555, Accuracy: 90.00
Epoch [17/30], Step [8/9], Loss: 1.4573, Accuracy: 89.60
Epoch [17/30], Step [9/9], Loss: 1.4698, Accuracy: 88.75
Epoch [18/30], Step [1/9], Loss: 1.4602, Accuracy: 90.60
Epoch [18/30], Step [2/9], Loss: 1.4505, Accuracy: 91.60
Epoch [18/30], Step [3/9], Loss: 1.4311, Accuracy: 94.40
Epoch [18/30], Step [4/9], Loss: 1.4645, Accuracy: 89.20
Epoch [18/30], Step [5/9], Loss: 1.4433, Accuracy: 91.20
Epoch [18/30], Step [6/9], Loss: 1.4359, Accuracy: 92.80
Epoch [18/30], Step [7/9], Loss: 1.4485, Accuracy: 91.40
Epoch [18/30], Step [8/9], Loss: 1.4482, Accuracy: 89.80
Epoch [18/30], Step [9/9], Loss: 1.4486, Accuracy: 91.25
Epoch [19/30], Step [1/9], Loss: 1.4424, Accuracy: 91.20
Epoch [19/30], Step [2/9], Loss: 1.4561, Accuracy: 89.60
Epoch [19/30], Step [3/9], Loss: 1.4563, Accuracy: 89.20
Epoch [19/30], Step [4/9], Loss: 1.4500, Accuracy: 91.00
Epoch [19/30], Step [5/9], Loss: 1.4452, Accuracy: 91.20
Epoch [19/30], Step [6/9], Loss: 1.4664, Accuracy: 88.40
Epoch [19/30], Step [7/9], Loss: 1.4502, Accuracy: 90.80
Epoch [19/30], Step [8/9], Loss: 1.4327, Accuracy: 92.20
Epoch [19/30], Step [9/9], Loss: 1.4547, Accuracy: 90.62
Epoch [20/30], Step [1/9], Loss: 1.4603, Accuracy: 89.80
Epoch [20/30], Step [2/9], Loss: 1.4362, Accuracy: 93.20
Epoch [20/30], Step [3/9], Loss: 1.4387, Accuracy: 92.20
Epoch [20/30], Step [4/9], Loss: 1.4546, Accuracy: 91.40
Epoch [20/30], Step [5/9], Loss: 1.4233, Accuracy: 93.40
Epoch [20/30], Step [6/9], Loss: 1.4290, Accuracy: 93.20
Epoch [20/30], Step [7/9], Loss: 1.4488, Accuracy: 90.80
Epoch [20/30], Step [8/9], Loss: 1.4347, Accuracy: 93.60
Epoch [20/30], Step [9/9], Loss: 1.4616, Accuracy: 90.00
Epoch [21/30], Step [1/9], Loss: 1.4487, Accuracy: 90.40
Epoch [21/30], Step [2/9], Loss: 1.4535, Accuracy: 90.60
Epoch [21/30], Step [3/9], Loss: 1.4551, Accuracy: 90.00
Epoch [21/30], Step [4/9], Loss: 1.4450, Accuracy: 93.00
Epoch [21/30], Step [5/9], Loss: 1.4309, Accuracy: 92.80
Epoch [21/30], Step [6/9], Loss: 1.4377, Accuracy: 92.40
Epoch [21/30], Step [7/9], Loss: 1.4332, Accuracy: 93.00
Epoch [21/30], Step [8/9], Loss: 1.4445, Accuracy: 90.80
Epoch [21/30], Step [9/9], Loss: 1.4514, Accuracy: 91.88
Epoch [22/30], Step [1/9], Loss: 1.4530, Accuracy: 90.40
Epoch [22/30], Step [2/9], Loss: 1.4278, Accuracy: 92.60
Epoch [22/30], Step [3/9], Loss: 1.4423, Accuracy: 92.00
Epoch [22/30], Step [4/9], Loss: 1.4487, Accuracy: 92.40
Epoch [22/30], Step [5/9], Loss: 1.4270, Accuracy: 94.60
Epoch [22/30], Step [6/9], Loss: 1.4537, Accuracy: 90.40
Epoch [22/30], Step [7/9], Loss: 1.4304, Accuracy: 93.20
Epoch [22/30], Step [8/9], Loss: 1.4357, Accuracy: 92.60
Epoch [22/30], Step [9/9], Loss: 1.4192, Accuracy: 93.12
Epoch [23/30], Step [1/9], Loss: 1.4483, Accuracy: 90.60
Epoch [23/30], Step [2/9], Loss: 1.4420, Accuracy: 92.00
Epoch [23/30], Step [3/9], Loss: 1.4253, Accuracy: 94.20
Epoch [23/30], Step [4/9], Loss: 1.4233, Accuracy: 94.20
Epoch [23/30], Step [5/9], Loss: 1.4341, Accuracy: 92.80
Epoch [23/30], Step [6/9], Loss: 1.4360, Accuracy: 93.00
Epoch [23/30], Step [7/9], Loss: 1.4244, Accuracy: 92.40
Epoch [23/30], Step [8/9], Loss: 1.4291, Accuracy: 92.00
Epoch [23/30], Step [9/9], Loss: 1.4469, Accuracy: 91.25
Epoch [24/30], Step [1/9], Loss: 1.4378, Accuracy: 92.40
Epoch [24/30], Step [2/9], Loss: 1.4381, Accuracy: 91.80
Epoch [24/30], Step [3/9], Loss: 1.4222, Accuracy: 93.00
Epoch [24/30], Step [4/9], Loss: 1.4355, Accuracy: 91.80
Epoch [24/30], Step [5/9], Loss: 1.4331, Accuracy: 93.00
Epoch [24/30], Step [6/9], Loss: 1.4404, Accuracy: 91.40
Epoch [24/30], Step [7/9], Loss: 1.4386, Accuracy: 91.60
Epoch [24/30], Step [8/9], Loss: 1.4345, Accuracy: 92.60
Epoch [24/30], Step [9/9], Loss: 1.4436, Accuracy: 90.00
Epoch [25/30], Step [1/9], Loss: 1.4219, Accuracy: 91.80
Epoch [25/30], Step [2/9], Loss: 1.4556, Accuracy: 89.40
Epoch [25/30], Step [3/9], Loss: 1.4441, Accuracy: 92.60
Epoch [25/30], Step [4/9], Loss: 1.4319, Accuracy: 93.40
Epoch [25/30], Step [5/9], Loss: 1.4248, Accuracy: 93.20
Epoch [25/30], Step [6/9], Loss: 1.4484, Accuracy: 90.60
Epoch [25/30], Step [7/9], Loss: 1.4297, Accuracy: 93.40
Epoch [25/30], Step [8/9], Loss: 1.4324, Accuracy: 93.00
Epoch [25/30], Step [9/9], Loss: 1.4271, Accuracy: 90.00
Epoch [26/30], Step [1/9], Loss: 1.4460, Accuracy: 89.20
Epoch [26/30], Step [2/9], Loss: 1.4330, Accuracy: 92.00
Epoch [26/30], Step [3/9], Loss: 1.4403, Accuracy: 91.60
Epoch [26/30], Step [4/9], Loss: 1.4233, Accuracy: 93.80
Epoch [26/30], Step [5/9], Loss: 1.4387, Accuracy: 92.20
Epoch [26/30], Step [6/9], Loss: 1.4249, Accuracy: 93.60
Epoch [26/30], Step [7/9], Loss: 1.4350, Accuracy: 91.60
Epoch [26/30], Step [8/9], Loss: 1.4409, Accuracy: 91.40
Epoch [26/30], Step [9/9], Loss: 1.4371, Accuracy: 91.88
Epoch [27/30], Step [1/9], Loss: 1.4384, Accuracy: 92.60
Epoch [27/30], Step [2/9], Loss: 1.4411, Accuracy: 91.60
Epoch [27/30], Step [3/9], Loss: 1.4369, Accuracy: 92.00
Epoch [27/30], Step [4/9], Loss: 1.4341, Accuracy: 91.40
Epoch [27/30], Step [5/9], Loss: 1.4391, Accuracy: 91.20
Epoch [27/30], Step [6/9], Loss: 1.4247, Accuracy: 93.20
Epoch [27/30], Step [7/9], Loss: 1.4305, Accuracy: 91.20
Epoch [27/30], Step [8/9], Loss: 1.4308, Accuracy: 93.40
Epoch [27/30], Step [9/9], Loss: 1.4410, Accuracy: 93.12
Epoch [28/30], Step [1/9], Loss: 1.4358, Accuracy: 91.40
Epoch [28/30], Step [2/9], Loss: 1.4281, Accuracy: 92.60
Epoch [28/30], Step [3/9], Loss: 1.4449, Accuracy: 91.40
Epoch [28/30], Step [4/9], Loss: 1.4464, Accuracy: 91.40
Epoch [28/30], Step [5/9], Loss: 1.4285, Accuracy: 93.40
Epoch [28/30], Step [6/9], Loss: 1.4183, Accuracy: 94.00
Epoch [28/30], Step [7/9], Loss: 1.4340, Accuracy: 92.40
Epoch [28/30], Step [8/9], Loss: 1.4352, Accuracy: 92.20
Epoch [28/30], Step [9/9], Loss: 1.4355, Accuracy: 93.12
Epoch [29/30], Step [1/9], Loss: 1.4259, Accuracy: 92.40
Epoch [29/30], Step [2/9], Loss: 1.4393, Accuracy: 92.40
Epoch [29/30], Step [3/9], Loss: 1.4153, Accuracy: 94.40
Epoch [29/30], Step [4/9], Loss: 1.4539, Accuracy: 90.20
Epoch [29/30], Step [5/9], Loss: 1.4211, Accuracy: 93.40
Epoch [29/30], Step [6/9], Loss: 1.4349, Accuracy: 92.20
Epoch [29/30], Step [7/9], Loss: 1.4314, Accuracy: 92.80
Epoch [29/30], Step [8/9], Loss: 1.4359, Accuracy: 92.00
Epoch [29/30], Step [9/9], Loss: 1.4520, Accuracy: 90.62
Epoch [30/30], Step [1/9], Loss: 1.4452, Accuracy: 90.60
Epoch [30/30], Step [2/9], Loss: 1.4293, Accuracy: 93.40
Epoch [30/30], Step [3/9], Loss: 1.4367, Accuracy: 90.80
Epoch [30/30], Step [4/9], Loss: 1.4335, Accuracy: 92.40
Epoch [30/30], Step [5/9], Loss: 1.4320, Accuracy: 93.20
Epoch [30/30], Step [6/9], Loss: 1.4296, Accuracy: 93.00
Epoch [30/30], Step [7/9], Loss: 1.4323, Accuracy: 92.20
Epoch [30/30], Step [8/9], Loss: 1.4354, Accuracy: 91.60
Epoch [30/30], Step [9/9], Loss: 1.4269, Accuracy: 93.75
```

```
C:\Users\kshit\AppData\Local\Temp\ipykernel_12700\60527815.py:7: FutureWarning:
The frame.append method is deprecated and will be removed from pandas in a
future version. Use pandas.concat instead.
  epoch_metrics_df = epoch_metrics_df.append(epoch_metrics, ignore_index=True)
```

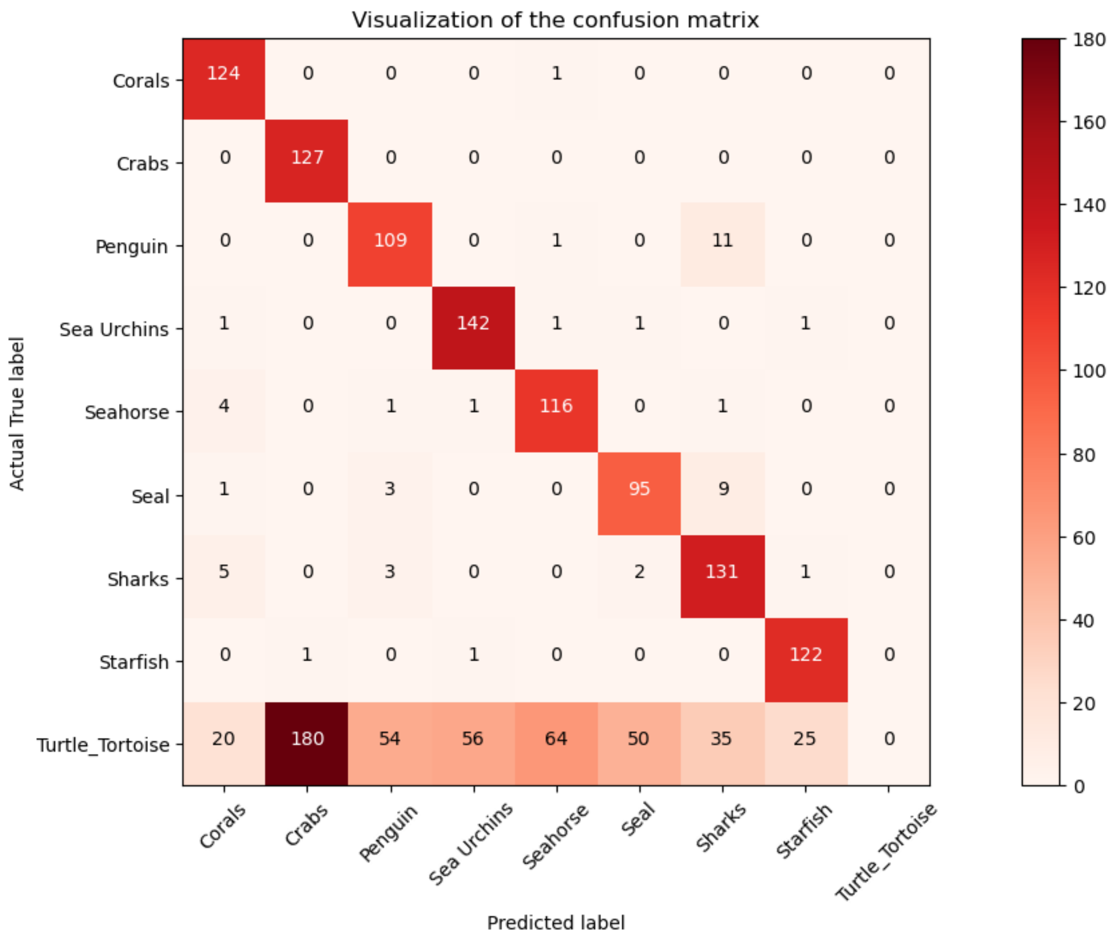
[14]:
```
#plot_training_metrics(pd.read_pickle("metric/ResNet-metric.pkl"))
```



```
[12]: #run_testing('ResNet')
```

```
C:\Users\kshit\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
-- TEST SET METRICS --
Accuracy: 64.400%
F-score: 0.692
Recall: 0.844
Precision: 0.595
Confusion matrix
```

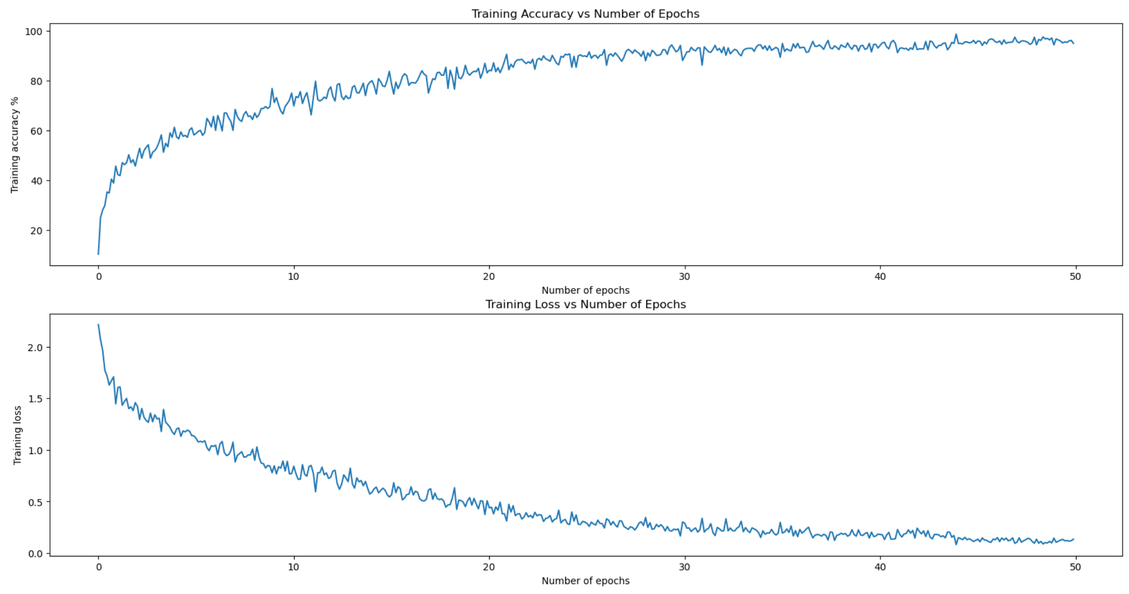


1.0.15 Training Data using Custom-CNN Model

[16]:
```
LEARNING_RATE = 0.001
NUM_EPOCHS = 50
LOSS_FN = nn.CrossEntropyLoss()

#epoch_metrics_df =train(CustomCNN(),'CustomCNN')
```

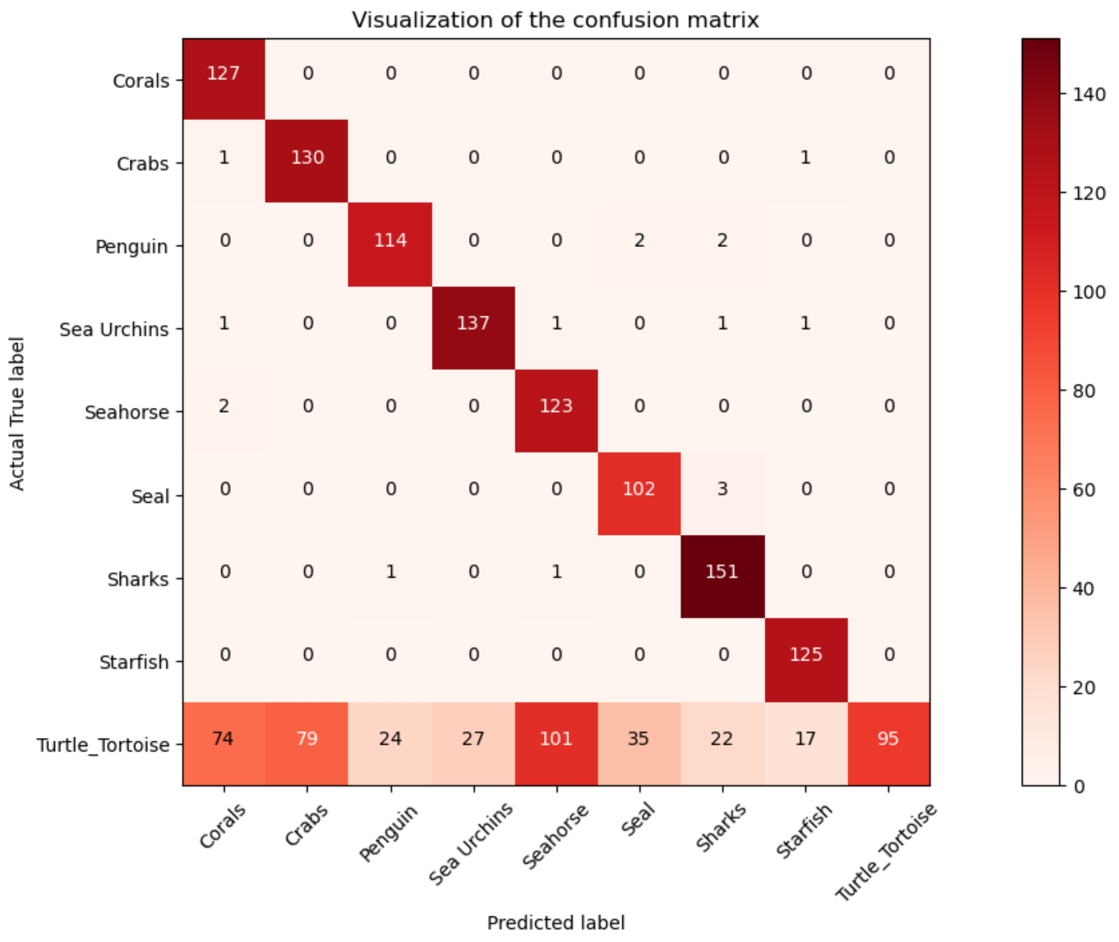
```
Epoch [1/50], Step [1/9], Loss: 2.2140, Accuracy: 10.20
Epoch [1/50], Step [2/9], Loss: 2.0662, Accuracy: 25.00
Epoch [1/50], Step [3/9], Loss: 1.9624, Accuracy: 28.00
Epoch [1/50], Step [4/9], Loss: 1.7703, Accuracy: 29.80
Epoch [1/50], Step [5/9], Loss: 1.7156, Accuracy: 35.20
Epoch [1/50], Step [6/9], Loss: 1.6282, Accuracy: 34.80
Epoch [1/50], Step [7/9], Loss: 1.6701, Accuracy: 40.40
Epoch [1/50], Step [8/9], Loss: 1.7080, Accuracy: 38.80
Epoch [1/50], Step [9/9], Loss: 1.4456, Accuracy: 45.62
Epoch [2/50], Step [1/9], Loss: 1.6052, Accuracy: 42.20
Epoch [2/50], Step [2/9], Loss: 1.6097, Accuracy: 41.80
Epoch [2/50], Step [3/9], Loss: 1.4320, Accuracy: 47.00
Epoch [2/50], Step [4/9], Loss: 1.4692, Accuracy: 46.20
Epoch [2/50], Step [5/9], Loss: 1.4983, Accuracy: 47.00
Epoch [2/50], Step [6/9], Loss: 1.3997, Accuracy: 50.20
Epoch [2/50], Step [7/9], Loss: 1.4175, Accuracy: 47.00
Epoch [2/50], Step [8/9], Loss: 1.3806, Accuracy: 48.20
Epoch [2/50], Step [9/9], Loss: 1.4577, Accuracy: 45.62
Epoch [3/50], Step [1/9], Loss: 1.4220, Accuracy: 49.40
Epoch [3/50], Step [2/9], Loss: 1.2945, Accuracy: 52.80
Epoch [3/50], Step [3/9], Loss: 1.4007, Accuracy: 48.80
Epoch [3/50], Step [4/9], Loss: 1.3172, Accuracy: 51.80
Epoch [3/50], Step [5/9], Loss: 1.2857, Accuracy: 53.20
Epoch [3/50], Step [6/9], Loss: 1.2670, Accuracy: 54.20
Epoch [3/50], Step [7/9], Loss: 1.3565, Accuracy: 48.80
Epoch [3/50], Step [8/9], Loss: 1.2721, Accuracy: 51.20
Epoch [3/50], Step [9/9], Loss: 1.3403, Accuracy: 51.88
Epoch [4/50], Step [1/9], Loss: 1.3005, Accuracy: 53.20
Epoch [4/50], Step [2/9], Loss: 1.3069, Accuracy: 55.40
Epoch [4/50], Step [3/9], Loss: 1.1784, Accuracy: 58.20
Epoch [4/50], Step [4/9], Loss: 1.3922, Accuracy: 51.20
Epoch [4/50], Step [5/9], Loss: 1.2687, Accuracy: 54.80
Epoch [4/50], Step [6/9], Loss: 1.2449, Accuracy: 53.40
Epoch [4/50], Step [7/9], Loss: 1.2184, Accuracy: 59.00
Epoch [4/50], Step [8/9], Loss: 1.1745, Accuracy: 57.20
Epoch [4/50], Step [9/9], Loss: 1.1496, Accuracy: 61.25
Epoch [5/50], Step [1/9], Loss: 1.2022, Accuracy: 57.40
Epoch [5/50], Step [2/9], Loss: 1.2124, Accuracy: 56.60
Epoch [5/50], Step [3/9], Loss: 1.1310, Accuracy: 59.40
Epoch [5/50], Step [4/9], Loss: 1.1837, Accuracy: 57.60
Epoch [5/50], Step [5/9], Loss: 1.1746, Accuracy: 58.00
Epoch [5/50], Step [6/9], Loss: 1.1912, Accuracy: 57.20
Epoch [5/50], Step [7/9], Loss: 1.1810, Accuracy: 60.20
Epoch [5/50], Step [8/9], Loss: 1.1385, Accuracy: 61.00
Epoch [5/50], Step [9/9], Loss: 1.1361, Accuracy: 58.13
Epoch [6/50], Step [1/9], Loss: 1.1116, Accuracy: 58.80
Epoch [6/50], Step [2/9], Loss: 1.0764, Accuracy: 59.60
Epoch [6/50], Step [3/9], Loss: 1.0834, Accuracy: 60.00
Epoch [6/50], Step [4/9], Loss: 1.0749, Accuracy: 58.00
Epoch [6/50], Step [5/9], Loss: 1.0892, Accuracy: 59.20
Epoch [6/50], Step [6/9], Loss: 1.0225, Accuracy: 64.80
Epoch [6/50], Step [7/9], Loss: 0.9921, Accuracy: 63.40
Epoch [6/50], Step [8/9], Loss: 1.0407, Accuracy: 61.40
Epoch [6/50], Step [9/9], Loss: 1.0338, Accuracy: 65.62
Epoch [7/50], Step [1/9], Loss: 1.0448, Accuracy: 60.00
Epoch [7/50], Step [2/9], Loss: 0.9532, Accuracy: 66.00
Epoch [7/50], Step [3/9], Loss: 1.0579, Accuracy: 63.40
Epoch [7/50], Step [4/9], Loss: 1.0812, Accuracy: 59.80
Epoch [7/50], Step [5/9], Loss: 0.9755, Accuracy: 67.00
Epoch [7/50], Step [6/9], Loss: 0.9451, Accuracy: 67.00
Epoch [7/50], Step [7/9], Loss: 0.9534, Accuracy: 65.00
Epoch [7/50], Step [8/9], Loss: 0.9916, Accuracy: 63.60
Epoch [7/50], Step [9/9], Loss: 1.0747, Accuracy: 60.00
Epoch [8/50], Step [1/9], Loss: 0.8828, Accuracy: 68.40
Epoch [8/50], Step [2/9], Loss: 0.9468, Accuracy: 65.60
Epoch [8/50], Step [3/9], Loss: 0.9636, Accuracy: 64.20
Epoch [8/50], Step [4/9], Loss: 0.9822, Accuracy: 63.60
Epoch [8/50], Step [5/9], Loss: 0.9300, Accuracy: 66.40
Epoch [8/50], Step [6/9], Loss: 0.9331, Accuracy: 67.60
Epoch [8/50], Step [7/9], Loss: 0.9511, Accuracy: 65.60
Epoch [8/50], Step [8/9], Loss: 0.9522, Accuracy: 65.80
Epoch [8/50], Step [9/9], Loss: 1.0069, Accuracy: 64.38
Epoch [9/50], Step [1/9], Loss: 0.8998, Accuracy: 67.00
Epoch [9/50], Step [2/9], Loss: 1.0292, Accuracy: 65.20
Epoch [9/50], Step [3/9], Loss: 0.9292, Accuracy: 66.40
Epoch [9/50], Step [4/9], Loss: 0.8714, Accuracy: 68.80
Epoch [9/50], Step [5/9], Loss: 0.8670, Accuracy: 68.80
Epoch [9/50], Step [6/9], Loss: 0.8245, Accuracy: 69.60
Epoch [9/50], Step [7/9], Loss: 0.8494, Accuracy: 68.80
Epoch [9/50], Step [8/9], Loss: 0.8429, Accuracy: 69.60
Epoch [9/50], Step [9/9], Loss: 0.7787, Accuracy: 76.88
Epoch [10/50], Step [1/9], Loss: 0.8464, Accuracy: 71.20
Epoch [10/50], Step [2/9], Loss: 0.7679, Accuracy: 73.20
Epoch [10/50], Step [3/9], Loss: 0.8363, Accuracy: 70.20
Epoch [10/50], Step [4/9], Loss: 0.8234, Accuracy: 67.80
Epoch [10/50], Step [5/9], Loss: 0.8920, Accuracy: 66.60
Epoch [10/50], Step [6/9], Loss: 0.7945, Accuracy: 69.60
Epoch [10/50], Step [7/9], Loss: 0.8890, Accuracy: 70.80
Epoch [10/50], Step [8/9], Loss: 0.7672, Accuracy: 72.20
Epoch [10/50], Step [9/9], Loss: 0.7709, Accuracy: 75.00
Epoch [11/50], Step [1/9], Loss: 0.8417, Accuracy: 69.80
Epoch [11/50], Step [2/9], Loss: 0.7685, Accuracy: 73.60
Epoch [11/50], Step [3/9], Loss: 0.7150, Accuracy: 73.20
Epoch [11/50], Step [4/9], Loss: 0.7169, Accuracy: 75.60
Epoch [11/50], Step [5/9], Loss: 0.8570, Accuracy: 70.80
Epoch [11/50], Step [6/9], Loss: 0.7644, Accuracy: 73.40
Epoch [11/50], Step [7/9], Loss: 0.7450, Accuracy: 75.20
Epoch [11/50], Step [8/9], Loss: 0.8383, Accuracy: 71.00
Epoch [11/50], Step [9/9], Loss: 0.8489, Accuracy: 66.25
Epoch [12/50], Step [1/9], Loss: 0.7721, Accuracy: 73.20
Epoch [12/50], Step [2/9], Loss: 0.5944, Accuracy: 79.80
Epoch [12/50], Step [3/9], Loss: 0.7787, Accuracy: 72.40
Epoch [12/50], Step [4/9], Loss: 0.7791, Accuracy: 71.80
Epoch [12/50], Step [5/9], Loss: 0.8334, Accuracy: 72.40
Epoch [12/50], Step [6/9], Loss: 0.7588, Accuracy: 73.40
Epoch [12/50], Step [7/9], Loss: 0.7768, Accuracy: 72.80
Epoch [12/50], Step [8/9], Loss: 0.7238, Accuracy: 76.20
Epoch [12/50], Step [9/9], Loss: 0.7354, Accuracy: 77.50
Epoch [13/50], Step [1/9], Loss: 0.7939, Accuracy: 73.60
Epoch [13/50], Step [2/9], Loss: 0.8027, Accuracy: 71.80
Epoch [13/50], Step [3/9], Loss: 0.6728, Accuracy: 78.40
Epoch [13/50], Step [4/9], Loss: 0.6187, Accuracy: 78.80
Epoch [13/50], Step [5/9], Loss: 0.6659, Accuracy: 73.60
Epoch [13/50], Step [6/9], Loss: 0.7581, Accuracy: 72.40
Epoch [13/50], Step [7/9], Loss: 0.7278, Accuracy: 74.00
Epoch [13/50], Step [8/9], Loss: 0.6924, Accuracy: 72.80
Epoch [13/50], Step [9/9], Loss: 0.8230, Accuracy: 73.12
Epoch [14/50], Step [1/9], Loss: 0.6692, Accuracy: 77.40
Epoch [14/50], Step [2/9], Loss: 0.6299, Accuracy: 78.00
Epoch [14/50], Step [3/9], Loss: 0.7296, Accuracy: 75.20
Epoch [14/50], Step [4/9], Loss: 0.6911, Accuracy: 75.00
Epoch [14/50], Step [5/9], Loss: 0.7029, Accuracy: 76.80
Epoch [14/50], Step [6/9], Loss: 0.6523, Accuracy: 79.00
Epoch [14/50], Step [7/9], Loss: 0.6949, Accuracy: 74.00
Epoch [14/50], Step [8/9], Loss: 0.6273, Accuracy: 78.20
Epoch [14/50], Step [9/9], Loss: 0.5714, Accuracy: 79.38
Epoch [15/50], Step [1/9], Loss: 0.5901, Accuracy: 80.00
Epoch [15/50], Step [2/9], Loss: 0.6238, Accuracy: 78.00
Epoch [15/50], Step [3/9], Loss: 0.6396, Accuracy: 74.60
Epoch [15/50], Step [4/9], Loss: 0.5842, Accuracy: 80.80
Epoch [15/50], Step [5/9], Loss: 0.6067, Accuracy: 79.60
Epoch [15/50], Step [6/9], Loss: 0.6292, Accuracy: 77.80
Epoch [15/50], Step [7/9], Loss: 0.6105, Accuracy: 77.60
Epoch [15/50], Step [8/9], Loss: 0.5663, Accuracy: 80.20
Epoch [15/50], Step [9/9], Loss: 0.5440, Accuracy: 83.75
Epoch [16/50], Step [1/9], Loss: 0.5647, Accuracy: 78.60
Epoch [16/50], Step [2/9], Loss: 0.6819, Accuracy: 74.60
Epoch [16/50], Step [3/9], Loss: 0.5851, Accuracy: 79.40
Epoch [16/50], Step [4/9], Loss: 0.6430, Accuracy: 76.80
Epoch [16/50], Step [5/9], Loss: 0.6247, Accuracy: 78.80
Epoch [16/50], Step [6/9], Loss: 0.5161, Accuracy: 81.60
Epoch [16/50], Step [7/9], Loss: 0.5336, Accuracy: 82.80
Epoch [16/50], Step [8/9], Loss: 0.5677, Accuracy: 82.00
Epoch [16/50], Step [9/9], Loss: 0.5706, Accuracy: 78.12
Epoch [17/50], Step [1/9], Loss: 0.6430, Accuracy: 79.20
Epoch [17/50], Step [2/9], Loss: 0.5634, Accuracy: 79.20
Epoch [17/50], Step [3/9], Loss: 0.5993, Accuracy: 79.00
Epoch [17/50], Step [4/9], Loss: 0.5890, Accuracy: 80.20
Epoch [17/50], Step [5/9], Loss: 0.5256, Accuracy: 82.20
Epoch [17/50], Step [6/9], Loss: 0.5090, Accuracy: 84.00
Epoch [17/50], Step [7/9], Loss: 0.5047, Accuracy: 82.60
Epoch [17/50], Step [8/9], Loss: 0.5203, Accuracy: 82.00
Epoch [17/50], Step [9/9], Loss: 0.6111, Accuracy: 75.00
Epoch [18/50], Step [1/9], Loss: 0.6240, Accuracy: 78.20
Epoch [18/50], Step [2/9], Loss: 0.5213, Accuracy: 80.80
Epoch [18/50], Step [3/9], Loss: 0.5812, Accuracy: 80.40
Epoch [18/50], Step [4/9], Loss: 0.5289, Accuracy: 83.20
Epoch [18/50], Step [5/9], Loss: 0.5179, Accuracy: 83.40
Epoch [18/50], Step [6/9], Loss: 0.5259, Accuracy: 82.20
Epoch [18/50], Step [7/9], Loss: 0.5047, Accuracy: 82.20
Epoch [18/50], Step [8/9], Loss: 0.4451, Accuracy: 85.40
Epoch [18/50], Step [9/9], Loss: 0.4670, Accuracy: 76.88
Epoch [19/50], Step [1/9], Loss: 0.4692, Accuracy: 84.20
Epoch [19/50], Step [2/9], Loss: 0.5344, Accuracy: 81.40
Epoch [19/50], Step [3/9], Loss: 0.6332, Accuracy: 76.60
Epoch [19/50], Step [4/9], Loss: 0.4234, Accuracy: 85.40
Epoch [19/50], Step [5/9], Loss: 0.5139, Accuracy: 81.20
Epoch [19/50], Step [6/9], Loss: 0.5087, Accuracy: 80.80
Epoch [19/50], Step [7/9], Loss: 0.4941, Accuracy: 82.40
Epoch [19/50], Step [8/9], Loss: 0.4501, Accuracy: 86.20
Epoch [19/50], Step [9/9], Loss: 0.5077, Accuracy: 83.12
Epoch [20/50], Step [1/9], Loss: 0.5374, Accuracy: 82.20
Epoch [20/50], Step [2/9], Loss: 0.4644, Accuracy: 83.20
Epoch [20/50], Step [3/9], Loss: 0.5303, Accuracy: 83.80
Epoch [20/50], Step [4/9], Loss: 0.4763, Accuracy: 83.60
Epoch [20/50], Step [5/9], Loss: 0.4291, Accuracy: 85.00
Epoch [20/50], Step [6/9], Loss: 0.5071, Accuracy: 81.00
Epoch [20/50], Step [7/9], Loss: 0.5038, Accuracy: 83.40
Epoch [20/50], Step [8/9], Loss: 0.3739, Accuracy: 87.00
Epoch [20/50], Step [9/9], Loss: 0.5074, Accuracy: 83.12
Epoch [21/50], Step [1/9], Loss: 0.4391, Accuracy: 84.20
Epoch [21/50], Step [2/9], Loss: 0.4447, Accuracy: 84.00
Epoch [21/50], Step [3/9], Loss: 0.3804, Accuracy: 87.20
Epoch [21/50], Step [4/9], Loss: 0.4483, Accuracy: 83.60
Epoch [21/50], Step [5/9], Loss: 0.4155, Accuracy: 85.20
Epoch [21/50], Step [6/9], Loss: 0.4934, Accuracy: 83.20
Epoch [21/50], Step [7/9], Loss: 0.3811, Accuracy: 85.20
Epoch [21/50], Step [8/9], Loss: 0.3805, Accuracy: 87.80
Epoch [21/50], Step [9/9], Loss: 0.3109, Accuracy: 90.62
Epoch [22/50], Step [1/9], Loss: 0.4731, Accuracy: 84.40
Epoch [22/50], Step [2/9], Loss: 0.3980, Accuracy: 86.60
Epoch [22/50], Step [3/9], Loss: 0.4592, Accuracy: 85.40
Epoch [22/50], Step [4/9], Loss: 0.3632, Accuracy: 87.60
Epoch [22/50], Step [5/9], Loss: 0.3799, Accuracy: 88.40
Epoch [22/50], Step [6/9], Loss: 0.3815, Accuracy: 88.40
Epoch [22/50], Step [7/9], Loss: 0.3302, Accuracy: 88.60
Epoch [22/50], Step [8/9], Loss: 0.3493, Accuracy: 87.60
Epoch [22/50], Step [9/9], Loss: 0.3919, Accuracy: 86.88
Epoch [23/50], Step [1/9], Loss: 0.3509, Accuracy: 87.60
Epoch [23/50], Step [2/9], Loss: 0.3658, Accuracy: 87.00
Epoch [23/50], Step [3/9], Loss: 0.3416, Accuracy: 88.60
Epoch [23/50], Step [4/9], Loss: 0.3953, Accuracy: 84.60
Epoch [23/50], Step [5/9], Loss: 0.3612, Accuracy: 88.40
Epoch [23/50], Step [6/9], Loss: 0.3729, Accuracy: 89.00
Epoch [23/50], Step [7/9], Loss: 0.3684, Accuracy: 88.20
Epoch [23/50], Step [8/9], Loss: 0.3064, Accuracy: 90.00
Epoch [23/50], Step [9/9], Loss: 0.3356, Accuracy: 88.75
Epoch [24/50], Step [1/9], Loss: 0.3415, Accuracy: 88.40
Epoch [24/50], Step [2/9], Loss: 0.3641, Accuracy: 87.80
Epoch [24/50], Step [3/9], Loss: 0.3024, Accuracy: 90.20
Epoch [24/50], Step [4/9], Loss: 0.3259, Accuracy: 88.60
Epoch [24/50], Step [5/9], Loss: 0.3351, Accuracy: 87.00
Epoch [24/50], Step [6/9], Loss: 0.4157, Accuracy: 86.40
Epoch [24/50], Step [7/9], Loss: 0.2931, Accuracy: 89.80
Epoch [24/50], Step [8/9], Loss: 0.3162, Accuracy: 89.40
Epoch [24/50], Step [9/9], Loss: 0.3264, Accuracy: 90.62
Epoch [25/50], Step [1/9], Loss: 0.2851, Accuracy: 90.40
Epoch [25/50], Step [2/9], Loss: 0.2764, Accuracy: 90.80
Epoch [25/50], Step [3/9], Loss: 0.3991, Accuracy: 85.40
Epoch [25/50], Step [4/9], Loss: 0.3097, Accuracy: 89.80
Epoch [25/50], Step [5/9], Loss: 0.3695, Accuracy: 85.40
Epoch [25/50], Step [6/9], Loss: 0.2789, Accuracy: 90.00
Epoch [25/50], Step [7/9], Loss: 0.2774, Accuracy: 90.40
Epoch [25/50], Step [8/9], Loss: 0.3076, Accuracy: 89.80
Epoch [25/50], Step [9/9], Loss: 0.3014, Accuracy: 90.00
Epoch [26/50], Step [1/9], Loss: 0.2881, Accuracy: 89.60
Epoch [26/50], Step [2/9], Loss: 0.2591, Accuracy: 91.60
Epoch [26/50], Step [3/9], Loss: 0.3001, Accuracy: 89.00
Epoch [26/50], Step [4/9], Loss: 0.2821, Accuracy: 90.00
Epoch [26/50], Step [5/9], Loss: 0.2715, Accuracy: 90.20
Epoch [26/50], Step [6/9], Loss: 0.3198, Accuracy: 89.00
Epoch [26/50], Step [7/9], Loss: 0.2870, Accuracy: 90.20
Epoch [26/50], Step [8/9], Loss: 0.2860, Accuracy: 90.40
Epoch [26/50], Step [9/9], Loss: 0.2435, Accuracy: 92.50
Epoch [27/50], Step [1/9], Loss: 0.3320, Accuracy: 86.20
Epoch [27/50], Step [2/9], Loss: 0.3186, Accuracy: 90.20
Epoch [27/50], Step [3/9], Loss: 0.2721, Accuracy: 90.80
Epoch [27/50], Step [4/9], Loss: 0.3058, Accuracy: 89.80
Epoch [27/50], Step [5/9], Loss: 0.2741, Accuracy: 91.20
Epoch [27/50], Step [6/9], Loss: 0.2522, Accuracy: 90.20
Epoch [27/50], Step [7/9], Loss: 0.3131, Accuracy: 89.00
Epoch [27/50], Step [8/9], Loss: 0.3099, Accuracy: 87.80
Epoch [27/50], Step [9/9], Loss: 0.2585, Accuracy: 89.38
Epoch [28/50], Step [1/9], Loss: 0.2448, Accuracy: 91.80
Epoch [28/50], Step [2/9], Loss: 0.2304, Accuracy: 92.60
Epoch [28/50], Step [3/9], Loss: 0.2562, Accuracy: 92.00
Epoch [28/50], Step [4/9], Loss: 0.2471, Accuracy: 91.00
Epoch [28/50], Step [5/9], Loss: 0.2243, Accuracy: 92.40
Epoch [28/50], Step [6/9], Loss: 0.2522, Accuracy: 91.60
Epoch [28/50], Step [7/9], Loss: 0.2916, Accuracy: 91.00
Epoch [28/50], Step [8/9], Loss: 0.3033, Accuracy: 89.80
Epoch [28/50], Step [9/9], Loss: 0.2681, Accuracy: 91.88
Epoch [29/50], Step [1/9], Loss: 0.3458, Accuracy: 88.00
Epoch [29/50], Step [2/9], Loss: 0.2488, Accuracy: 91.20
Epoch [29/50], Step [3/9], Loss: 0.2943, Accuracy: 89.80
Epoch [29/50], Step [4/9], Loss: 0.2303, Accuracy: 92.20
Epoch [29/50], Step [5/9], Loss: 0.2376, Accuracy: 91.40
Epoch [29/50], Step [6/9], Loss: 0.2786, Accuracy: 90.20
Epoch [29/50], Step [7/9], Loss: 0.2551, Accuracy: 90.20
Epoch [29/50], Step [8/9], Loss: 0.2756, Accuracy: 90.60
Epoch [29/50], Step [9/9], Loss: 0.2548, Accuracy: 92.50
Epoch [30/50], Step [1/9], Loss: 0.2149, Accuracy: 92.40
Epoch [30/50], Step [2/9], Loss: 0.2570, Accuracy: 90.60
Epoch [30/50], Step [3/9], Loss: 0.2187, Accuracy: 93.40
Epoch [30/50], Step [4/9], Loss: 0.2141, Accuracy: 94.40
Epoch [30/50], Step [5/9], Loss: 0.2328, Accuracy: 93.00
Epoch [30/50], Step [6/9], Loss: 0.2258, Accuracy: 91.60
Epoch [30/50], Step [7/9], Loss: 0.2340, Accuracy: 92.20
Epoch [30/50], Step [8/9], Loss: 0.1655, Accuracy: 94.20
Epoch [30/50], Step [9/9], Loss: 0.3016, Accuracy: 88.12
Epoch [31/50], Step [1/9], Loss: 0.2900, Accuracy: 89.80
Epoch [31/50], Step [2/9], Loss: 0.2417, Accuracy: 91.60
Epoch [31/50], Step [3/9], Loss: 0.2438, Accuracy: 91.60
Epoch [31/50], Step [4/9], Loss: 0.2097, Accuracy: 93.40
Epoch [31/50], Step [5/9], Loss: 0.2295, Accuracy: 92.40
Epoch [31/50], Step [6/9], Loss: 0.2446, Accuracy: 92.00
Epoch [31/50], Step [7/9], Loss: 0.2032, Accuracy: 93.20
Epoch [31/50], Step [8/9], Loss: 0.2273, Accuracy: 93.20
Epoch [31/50], Step [9/9], Loss: 0.3375, Accuracy: 86.25
Epoch [32/50], Step [1/9], Loss: 0.2041, Accuracy: 93.60
Epoch [32/50], Step [2/9], Loss: 0.2293, Accuracy: 92.60
Epoch [32/50], Step [3/9], Loss: 0.2405, Accuracy: 91.60
Epoch [32/50], Step [4/9], Loss: 0.2852, Accuracy: 91.40
Epoch [32/50], Step [5/9], Loss: 0.1993, Accuracy: 92.60
Epoch [32/50], Step [6/9], Loss: 0.1697, Accuracy: 94.20
Epoch [32/50], Step [7/9], Loss: 0.2487, Accuracy: 92.20
Epoch [32/50], Step [8/9], Loss: 0.2356, Accuracy: 93.00
Epoch [32/50], Step [9/9], Loss: 0.2149, Accuracy: 91.25
Epoch [33/50], Step [1/9], Loss: 0.2172, Accuracy: 93.60
Epoch [33/50], Step [2/9], Loss: 0.3331, Accuracy: 90.20
Epoch [33/50], Step [3/9], Loss: 0.2154, Accuracy: 93.20
Epoch [33/50], Step [4/9], Loss: 0.2438, Accuracy: 90.80
Epoch [33/50], Step [5/9], Loss: 0.2177, Accuracy: 91.80
Epoch [33/50], Step [6/9], Loss: 0.2153, Accuracy: 92.60
Epoch [33/50], Step [7/9], Loss: 0.2409, Accuracy: 92.20
Epoch [33/50], Step [8/9], Loss: 0.2556, Accuracy: 90.80
Epoch [33/50], Step [9/9], Loss: 0.3090, Accuracy: 90.00
Epoch [34/50], Step [1/9], Loss: 0.2070, Accuracy: 92.20
Epoch [34/50], Step [2/9], Loss: 0.2471, Accuracy: 92.80
Epoch [34/50], Step [3/9], Loss: 0.2172, Accuracy: 93.00
Epoch [34/50], Step [4/9], Loss: 0.1983, Accuracy: 93.00
Epoch [34/50], Step [5/9], Loss: 0.2450, Accuracy: 93.00
Epoch [34/50], Step [6/9], Loss: 0.2344, Accuracy: 91.80
Epoch [34/50], Step [7/9], Loss: 0.2227, Accuracy: 93.60
Epoch [34/50], Step [8/9], Loss: 0.1987, Accuracy: 94.40
Epoch [34/50], Step [9/9], Loss: 0.1515, Accuracy: 94.38
Epoch [35/50], Step [1/9], Loss: 0.2165, Accuracy: 92.80
Epoch [35/50], Step [2/9], Loss: 0.1834, Accuracy: 94.00
Epoch [35/50], Step [3/9], Loss: 0.1953, Accuracy: 92.20
Epoch [35/50], Step [4/9], Loss: 0.1939, Accuracy: 94.00
Epoch [35/50], Step [5/9], Loss: 0.2300, Accuracy: 92.20
Epoch [35/50], Step [6/9], Loss: 0.1947, Accuracy: 92.80
Epoch [35/50], Step [7/9], Loss: 0.1755, Accuracy: 93.40
Epoch [35/50], Step [8/9], Loss: 0.1914, Accuracy: 92.80
Epoch [35/50], Step [9/9], Loss: 0.2982, Accuracy: 89.38
Epoch [36/50], Step [1/9], Loss: 0.1943, Accuracy: 95.00
Epoch [36/50], Step [2/9], Loss: 0.2044, Accuracy: 91.80
Epoch [36/50], Step [3/9], Loss: 0.2340, Accuracy: 92.80
Epoch [36/50], Step [4/9], Loss: 0.2012, Accuracy: 92.20
Epoch [36/50], Step [5/9], Loss: 0.2633, Accuracy: 92.00
Epoch [36/50], Step [6/9], Loss: 0.1657, Accuracy: 94.80
Epoch [36/50], Step [7/9], Loss: 0.2176, Accuracy: 93.20
Epoch [36/50], Step [8/9], Loss: 0.1603, Accuracy: 95.00
Epoch [36/50], Step [9/9], Loss: 0.2288, Accuracy: 93.12
Epoch [37/50], Step [1/9], Loss: 0.1936, Accuracy: 93.40
Epoch [37/50], Step [2/9], Loss: 0.2105, Accuracy: 93.40
Epoch [37/50], Step [3/9], Loss: 0.2337, Accuracy: 91.40
Epoch [37/50], Step [4/9], Loss: 0.2508, Accuracy: 91.60
Epoch [37/50], Step [5/9], Loss: 0.1930, Accuracy: 93.80
Epoch [37/50], Step [6/9], Loss: 0.1480, Accuracy: 95.80
Epoch [37/50], Step [7/9], Loss: 0.1738, Accuracy: 93.80
Epoch [37/50], Step [8/9], Loss: 0.1803, Accuracy: 93.80
Epoch [37/50], Step [9/9], Loss: 0.1771, Accuracy: 94.38
Epoch [38/50], Step [1/9], Loss: 0.1643, Accuracy: 93.60
Epoch [38/50], Step [2/9], Loss: 0.1824, Accuracy: 92.60
Epoch [38/50], Step [3/9], Loss: 0.1727, Accuracy: 94.60
Epoch [38/50], Step [4/9], Loss: 0.1277, Accuracy: 96.20
Epoch [38/50], Step [5/9], Loss: 0.2050, Accuracy: 93.20
Epoch [38/50], Step [6/9], Loss: 0.2017, Accuracy: 92.80
Epoch [38/50], Step [7/9], Loss: 0.1233, Accuracy: 94.00
Epoch [38/50], Step [8/9], Loss: 0.1719, Accuracy: 93.40
Epoch [38/50], Step [9/9], Loss: 0.1783, Accuracy: 92.50
Epoch [39/50], Step [1/9], Loss: 0.1927, Accuracy: 94.40
Epoch [39/50], Step [2/9], Loss: 0.1822, Accuracy: 93.80
Epoch [39/50], Step [3/9], Loss: 0.1904, Accuracy: 93.40
Epoch [39/50], Step [4/9], Loss: 0.1636, Accuracy: 95.20
Epoch [39/50], Step [5/9], Loss: 0.1764, Accuracy: 93.60
Epoch [39/50], Step [6/9], Loss: 0.2271, Accuracy: 92.60
Epoch [39/50], Step [7/9], Loss: 0.1798, Accuracy: 94.20
Epoch [39/50], Step [8/9], Loss: 0.1644, Accuracy: 94.00
Epoch [39/50], Step [9/9], Loss: 0.2260, Accuracy: 91.88
Epoch [40/50], Step [1/9], Loss: 0.1756, Accuracy: 94.00
Epoch [40/50], Step [2/9], Loss: 0.1711, Accuracy: 94.20
Epoch [40/50], Step [3/9], Loss: 0.1895, Accuracy: 92.60
Epoch [40/50], Step [4/9], Loss: 0.1996, Accuracy: 93.00
Epoch [40/50], Step [5/9], Loss: 0.1459, Accuracy: 95.00
Epoch [40/50], Step [6/9], Loss: 0.2031, Accuracy: 91.60
Epoch [40/50], Step [7/9], Loss: 0.1608, Accuracy: 94.60
Epoch [40/50], Step [8/9], Loss: 0.1658, Accuracy: 94.60
Epoch [40/50], Step [9/9], Loss: 0.1847, Accuracy: 93.12
Epoch [41/50], Step [1/9], Loss: 0.1804, Accuracy: 94.20
Epoch [41/50], Step [2/9], Loss: 0.1780, Accuracy: 95.20
Epoch [41/50], Step [3/9], Loss: 0.1311, Accuracy: 95.40
Epoch [41/50], Step [4/9], Loss: 0.1765, Accuracy: 93.60
Epoch [41/50], Step [5/9], Loss: 0.2008, Accuracy: 92.80
Epoch [41/50], Step [6/9], Loss: 0.1345, Accuracy: 95.40
Epoch [41/50], Step [7/9], Loss: 0.1362, Accuracy: 96.20
Epoch [41/50], Step [8/9], Loss: 0.1397, Accuracy: 95.00
Epoch [41/50], Step [9/9], Loss: 0.2283, Accuracy: 91.25
Epoch [42/50], Step [1/9], Loss: 0.1827, Accuracy: 93.00
Epoch [42/50], Step [2/9], Loss: 0.1556, Accuracy: 93.20
Epoch [42/50], Step [3/9], Loss: 0.1852, Accuracy: 92.80
Epoch [42/50], Step [4/9], Loss: 0.1896, Accuracy: 93.00
Epoch [42/50], Step [5/9], Loss: 0.2252, Accuracy: 92.20
Epoch [42/50], Step [6/9], Loss: 0.1935, Accuracy: 93.20
Epoch [42/50], Step [7/9], Loss: 0.2198, Accuracy: 92.60
Epoch [42/50], Step [8/9], Loss: 0.1455, Accuracy: 95.40
Epoch [42/50], Step [9/9], Loss: 0.2417, Accuracy: 92.50
Epoch [43/50], Step [1/9], Loss: 0.2114, Accuracy: 92.80
Epoch [43/50], Step [2/9], Loss: 0.1865, Accuracy: 92.80
Epoch [43/50], Step [3/9], Loss: 0.2157, Accuracy: 92.80
Epoch [43/50], Step [4/9], Loss: 0.1607, Accuracy: 95.80
Epoch [43/50], Step [5/9], Loss: 0.2173, Accuracy: 92.40
Epoch [43/50], Step [6/9], Loss: 0.1552, Accuracy: 96.00
Epoch [43/50], Step [7/9], Loss: 0.1382, Accuracy: 95.60
Epoch [43/50], Step [8/9], Loss: 0.1812, Accuracy: 94.00
Epoch [43/50], Step [9/9], Loss: 0.1811, Accuracy: 93.12
Epoch [44/50], Step [1/9], Loss: 0.1766, Accuracy: 94.20
Epoch [44/50], Step [2/9], Loss: 0.1589, Accuracy: 94.20
Epoch [44/50], Step [3/9], Loss: 0.1679, Accuracy: 95.00
Epoch [44/50], Step [4/9], Loss: 0.1482, Accuracy: 95.20
Epoch [44/50], Step [5/9], Loss: 0.2046, Accuracy: 92.40
Epoch [44/50], Step [6/9], Loss: 0.2032, Accuracy: 93.60
Epoch [44/50], Step [7/9], Loss: 0.1540, Accuracy: 95.40
Epoch [44/50], Step [8/9], Loss: 0.1702, Accuracy: 95.00
Epoch [44/50], Step [9/9], Loss: 0.0821, Accuracy: 98.75
Epoch [45/50], Step [1/9], Loss: 0.1501, Accuracy: 95.00
Epoch [45/50], Step [2/9], Loss: 0.1408, Accuracy: 95.00
Epoch [45/50], Step [3/9], Loss: 0.1307, Accuracy: 94.80
Epoch [45/50], Step [4/9], Loss: 0.1531, Accuracy: 95.60
Epoch [45/50], Step [5/9], Loss: 0.1299, Accuracy: 95.60
Epoch [45/50], Step [6/9], Loss: 0.1392, Accuracy: 95.20
Epoch [45/50], Step [7/9], Loss: 0.1269, Accuracy: 95.40
Epoch [45/50], Step [8/9], Loss: 0.1126, Accuracy: 96.20
Epoch [45/50], Step [9/9], Loss: 0.1234, Accuracy: 95.00
Epoch [46/50], Step [1/9], Loss: 0.1351, Accuracy: 96.00
Epoch [46/50], Step [2/9], Loss: 0.1101, Accuracy: 95.80
Epoch [46/50], Step [3/9], Loss: 0.1478, Accuracy: 94.20
Epoch [46/50], Step [4/9], Loss: 0.1269, Accuracy: 95.60
Epoch [46/50], Step [5/9], Loss: 0.1243, Accuracy: 94.80
Epoch [46/50], Step [6/9], Loss: 0.1108, Accuracy: 96.40
Epoch [46/50], Step [7/9], Loss: 0.1042, Accuracy: 96.80
Epoch [46/50], Step [8/9], Loss: 0.1355, Accuracy: 96.60
Epoch [46/50], Step [9/9], Loss: 0.1298, Accuracy: 95.62
Epoch [47/50], Step [1/9], Loss: 0.1507, Accuracy: 95.40
Epoch [47/50], Step [2/9], Loss: 0.1116, Accuracy: 96.00
Epoch [47/50], Step [3/9], Loss: 0.1466, Accuracy: 94.60
Epoch [47/50], Step [4/9], Loss: 0.1277, Accuracy: 96.40
Epoch [47/50], Step [5/9], Loss: 0.1459, Accuracy: 95.00
Epoch [47/50], Step [6/9], Loss: 0.1192, Accuracy: 95.40
Epoch [47/50], Step [7/9], Loss: 0.1264, Accuracy: 95.40
Epoch [47/50], Step [8/9], Loss: 0.1480, Accuracy: 95.60
Epoch [47/50], Step [9/9], Loss: 0.0948, Accuracy: 97.50
Epoch [48/50], Step [1/9], Loss: 0.1068, Accuracy: 95.80
Epoch [48/50], Step [2/9], Loss: 0.1483, Accuracy: 95.20
Epoch [48/50], Step [3/9], Loss: 0.1054, Accuracy: 96.00
Epoch [48/50], Step [4/9], Loss: 0.1189, Accuracy: 96.00
Epoch [48/50], Step [5/9], Loss: 0.1298, Accuracy: 95.80
Epoch [48/50], Step [6/9], Loss: 0.1431, Accuracy: 95.40
Epoch [48/50], Step [7/9], Loss: 0.1378, Accuracy: 94.60
Epoch [48/50], Step [8/9], Loss: 0.1132, Accuracy: 95.20
Epoch [48/50], Step [9/9], Loss: 0.0953, Accuracy: 97.50
Epoch [49/50], Step [1/9], Loss: 0.1351, Accuracy: 94.40
Epoch [49/50], Step [2/9], Loss: 0.0955, Accuracy: 96.60
Epoch [49/50], Step [3/9], Loss: 0.1136, Accuracy: 96.20
Epoch [49/50], Step [4/9], Loss: 0.0889, Accuracy: 97.60
Epoch [49/50], Step [5/9], Loss: 0.1014, Accuracy: 96.80
Epoch [49/50], Step [6/9], Loss: 0.0980, Accuracy: 97.00
Epoch [49/50], Step [7/9], Loss: 0.1144, Accuracy: 96.40
Epoch [49/50], Step [8/9], Loss: 0.0985, Accuracy: 97.20
Epoch [49/50], Step [9/9], Loss: 0.1452, Accuracy: 94.38
Epoch [50/50], Step [1/9], Loss: 0.1051, Accuracy: 96.80
Epoch [50/50], Step [2/9], Loss: 0.1144, Accuracy: 96.40
Epoch [50/50], Step [3/9], Loss: 0.1257, Accuracy: 96.00
Epoch [50/50], Step [4/9], Loss: 0.1338, Accuracy: 95.20
Epoch [50/50], Step [5/9], Loss: 0.1198, Accuracy: 95.60
Epoch [50/50], Step [6/9], Loss: 0.1211, Accuracy: 95.40
Epoch [50/50], Step [7/9], Loss: 0.1155, Accuracy: 96.00
Epoch [50/50], Step [8/9], Loss: 0.1205, Accuracy: 96.20
Epoch [50/50], Step [9/9], Loss: 0.1350, Accuracy: 95.00

C:\Users\kshit\AppData\Local\Temp\ipykernel_12700\60527815.py:7: FutureWarning:
The frame.append method is deprecated and will be removed from pandas in a
future version. Use pandas.concat instead.
  epoch_metrics_df = epoch_metrics_df.append(epoch_metrics, ignore_index=True)
```

[17]:
```
#plot_training_metrics(pd.read_pickle("metric/CustomCNN-metric.pkl"))
```



[14]:
```
#run_testing('CustomCNN')
```

```
-- TEST SET METRICS --
Accuracy: 73.600%
F-score: 0.780
Recall: 0.896
Precision: 0.765
Confusion matrix
```



Actual \ Predicted	Corals	Crabs	Penguin	Sea Urchins	Seahorse	Seal	Sharks	Starfish	Turtle_Tortoise
Corals	127	0	0	0	0	0	0	0	0
Crabs	1	130	0	0	0	0	0	1	0
Penguin	0	0	114	0	0	2	2	0	0
Sea Urchins	1	0	0	137	1	0	1	1	0
Seahorse	2	0	0	0	123	0	0	0	0
Seal	0	0	0	0	0	102	3	0	0
Sharks	0	0	1	0	1	0	151	0	0
Starfish	0	0	0	0	0	0	0	125	0
Turtle_Tortoise	74	79	24	27	101	35	22	17	95

1.0.16 HOG (Histogram of Gradients) with SVM

A class of sea animals vary so much in color. Structural cues like shape may give a more robust representation. Gradients of specific directions captures some notion of shape. To allow for some variability in shape, we'll use features known as Histogram of Oriented Gradients (HOG).

The idea of HOG is instead of using each individual gradient direction of each individual pixel of an image, we group the pixels into small cells. For each cell, we compute all the gradient directions and group them into a number of orientation bins. We sum up the gradient magnitude in each sample. So stronger gradients contribute more weight to their bins, and effects of small random orientations due to noise is reduced. This histogram gives us a picture of the dominant orientation of that cell. Doing this for all cells gives us a representation of the structure of the image. The HOG features keep the representation of an object distinct but also allow for some variations in shape.

1.0.17 Classifier

We use multiclass support vector machines as our classifier. The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.

1.0.18 Steps for training

- Extract HOG features from training and test images
- Use Multiclass SVM to train model to classify the HOG features
- Test the model using test set

[15]:
```
from skimage.feature import hog
import imageio
from sklearn.svm import SVC,LinearSVC
from sklearn.metrics import accuracy_score
import _pickle as pickle
from skimage.feature import match_descriptors, plot_matches, SIFT
from scipy.spatial.distance import cdist
from sklearn.cluster import KMeans
```

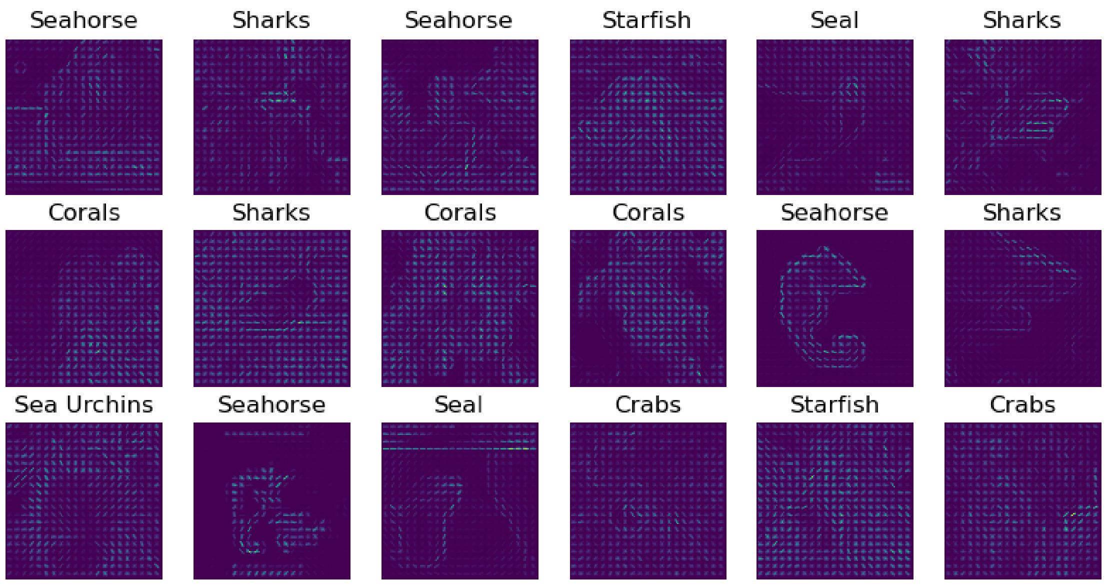
1.0.19 Method to extract HOG features

[20]:
```
def create_features_dataset(train=True,loader=None):
    train_test = "train"
    if(not train):
        train_test="test"

    hog_features=[]
    hog_images=[]
    landmarks=[]
    labels_list=[]
    for i, (images, labels) in enumerate(loader):
        for i in (range(len(labels.numpy()))):
            img = images.numpy()[i,1]
            labels_list.append(labels.numpy()[i])
            features, hog_image = hog(img, orientations=9, pixels_per_cell=(6,␣
↪6),
                                    cells_per_block=(3, 3), visualize=True)
            hog_features.append(features)
            hog_images.append(hog_image)

    np.save(f'data/hog_features_{train_test}.npy',hog_features)
    np.save(f'data/labels_{train_test}.npy',labels_list)
    np.save(f'data/hog_images_{train_test}.npy',hog_images)

    print(f'Hog Features created in data/hog_features{train_test}.npy')
    print(f'Hog Images created in data/hog_images_{train_test}.npy')
    print(f'Labels created in data/labels_{train_test}.npy')
```

[16]:
```
#create_features_dataset(train=True,loader=train_sampled_dataloader)
#create_features_dataset(train=False,loader=test_set_loader)
```

1.0.20 Visualizing HOG Images

```
[66]: def visualize_hog_images(n_row=10,n_col=10):

          dict={0:'Corals',1:'Crabs',2:'Penguin',3:'Sea Urchins',4:'Seahorse',5:
       ↪'Seal',6:'Sharks',7:'Starfish',8:'Turtle_Tortoise'}

          plt.figure(figsize=(10,5))
          hog_img = np.load("data/hog_images_train.npy",allow_pickle=True)
          hog_labels = np.load("data/labels_train.npy",allow_pickle=True)
          for i in range(n_row*n_col):
              plt.subplot(n_row,n_col,i+1)
              plt.title(dict[hog_labels[i]])
              plt.imshow(hog_img[i])
              plt.axis('off')

      visualize_hog_images(n_row=3,n_col=6)
```



1.0.21 Method to load, train and evaluate data

```
[31]: def load_data(hog_features_f,labels):
          data_dict = dict()

          data_dict['X'] = np.load(hog_features_f)
          data_dict['Y'] = np.load(labels)

          return data_dict
```

```
def evaluate(model, X, Y):
        predicted_Y = model.predict(X)
        accuracy = accuracy_score(Y, predicted_Y)
        return accuracy

def train_SVC(data,epochs,random_state,kernel,decision_function,gamma):

    model = SVC(random_state=random_state, max_iter=epochs, kernel=kernel,␣
 ↪decision_function_shape=decision_function, gamma=gamma)

    print( "start training...")
    print( "--")
    print( "kernel: {}".format(kernel))
    print( "decision function: {} ".format(decision_function))
    print( "max epochs: {} ".format(epochs))
    print( "gamma: {} ".format(gamma))
    print( "--")
    print( "Training samples: {}".format(len(data['Y'])))

    model.fit(data['X'], data['Y'])

    with open("models/svm.bin", 'wb') as f:
        pickle.dump(model, f)


    print( "calculating accuracy...")
    accuracy = evaluate(model, data['X'], data['Y'])
    print( "accuracy = {0:.1f}".format(accuracy*100))
    return accuracy


def run_svm_testing(test):

    targets = test['Y']

    with open("models/svm.bin", 'rb') as f:
        model = pickle.load(f)

    predictions = model.predict(test['X'])

    accuracy = accuracy_score(targets, predictions) * 100
    f1 = f1_score(targets, predictions, average='macro')
    recall = recall_score(targets, predictions, average='macro')
    precision = precision_score(targets, predictions, average='macro')
    conf_matrix = confusion_matrix(targets, predictions)
    pretty_print_metrics([accuracy,f1,recall,precision])
    plot_cm(np.array(conf_matrix), CLASSES)
```

[23]:
```
train_dict = load_data("data/hog_features_train.npy","data/labels_train.npy")
test_dict = load_data("data/hog_features_test.npy","data/labels_test.npy")
```

[24]:
```
epochs = 1000
random_state = 0
kernel = 'rbf'
decision_function = 'ovr'
gamma = 0.001

#train_SVC(train_dict,epochs=epochs,random_state=random_state,kernel=kernel,decision_function=
```

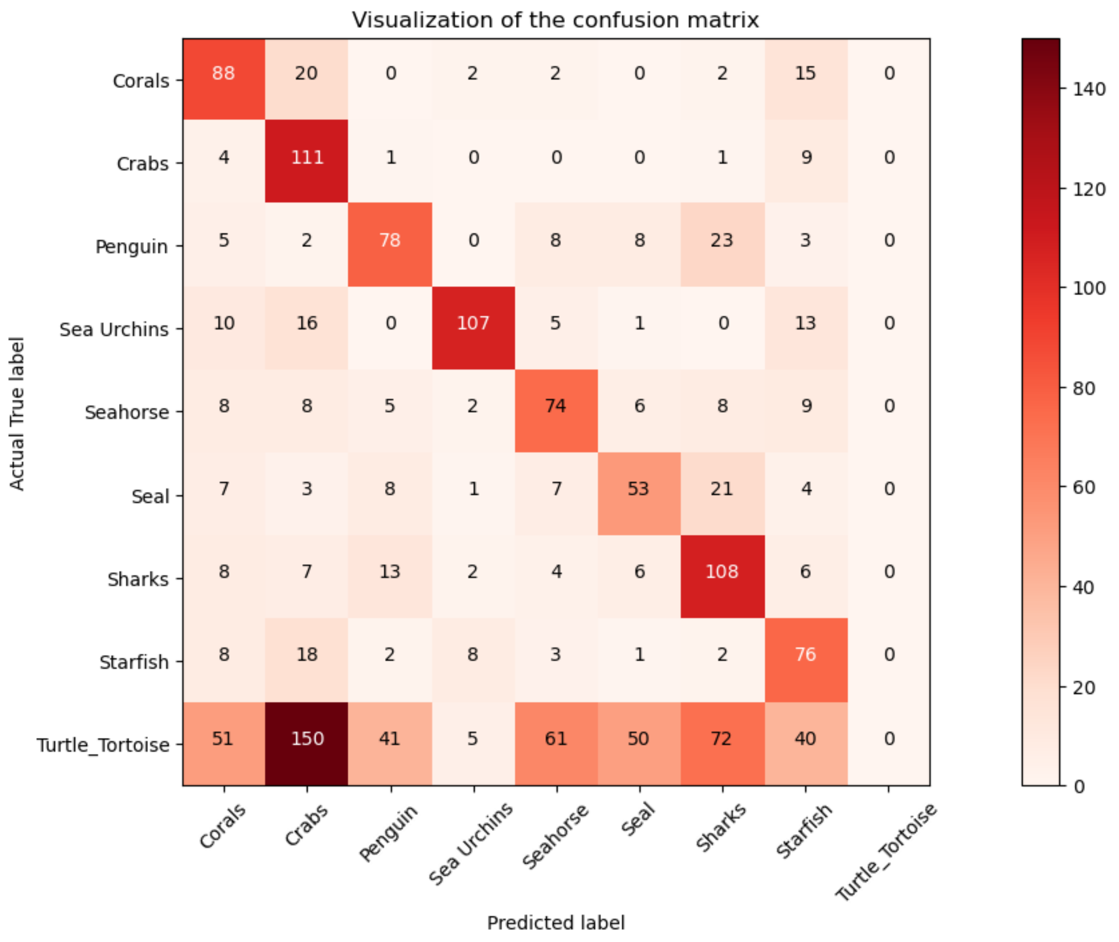
```
start training…
--
kernel: rbf
decision function: ovr
max epochs: 1000
gamma: 0.001
--
Training samples: 4160
calculating accuracy…
accuracy = 81.8
```

[24]:
```
0.8180288461538462
```

[32]:
```
run_svm_testing(test_dict)
```

```
C:\Users\kshit\anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
-- TEST SET METRICS --
Accuracy: 46.333%
F-score: 0.492
Recall: 0.595
Precision: 0.437
Confusion matrix
```



	Corals	Crabs	Penguin	Sea Urchins	Seahorse	Seal	Sharks	Starfish	Turtle_Tortoise
Corals	88	20	0	2	2	0	2	15	0
Crabs	4	111	1	0	0	0	1	9	0
Penguin	5	2	78	0	8	8	23	3	0
Sea Urchins	10	16	0	107	5	1	0	13	0
Seahorse	8	8	5	2	74	6	8	9	0
Seal	7	3	8	1	7	53	21	4	0
Sharks	8	7	13	2	4	6	108	6	0
Starfish	8	18	2	8	3	1	2	76	0
Turtle_Tortoise	51	150	41	5	61	50	72	40	0

1.0.22 SIFT with SVM Bag of visual words

1.0.23 Scale Invariant Feature Transform (SIFT)

SIFT is a feature extraction method that reduces the image content to a set of points used to detect similar patterns in other images. This algorithm is usually related to computer vision applications, including image matching and object detection.

1.0.24 Classifier

We use multiclass support vector machines as our classifier to classify SIFT features. The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.

1.0.25 Steps to training algorithm

- Extract SIFT features from train and test images.
- Quantize the feature space. Kmeans algorithm is used for clustering the SIFT features and the center points, we get from the clustering algorithm, are our visual words.

- Extract local features and compare these features with visual words to create histograms for each image both for the test and train dataset.
- Predict the class of test images comparing with each histogram of train images. We will used multiclass SVM for this task.
- Evaluate performance metrics.

1.0.26 Methods to extract SIFT features and create Histogram from bag of words

[24]:
```python
def extract_SIFT_features(loader=None,train=True):

    train_test = "train"
    if(not train):
        train_test="test"

    descriptor_extractor = SIFT()

    image_descriptors = []

    for i, (images, labels) in enumerate(loader):
        for i in (range(len(labels.numpy()))):
            img = images.numpy()[i,1]
            descriptor_extractor.detect_and_extract(img)
            image_descriptors.append(descriptor_extractor.descriptors)

    np.save(f'data/sift_features_{train_test}.npy',image_descriptors)

    print(f'SIFT Features created in data/sift_features_{train_test}.npy')

#method to create bag of visual words using k-means algorithm
def kmean_bow(all_descriptors, num_cluster):
    bow_dict = []

    kmeans = KMeans(n_clusters = num_cluster)
    kmeans.fit(all_descriptors)

    bow_dict = kmeans.cluster_centers_

    with open("models/bow_kmeans.bin", 'wb') as f:
        pickle.dump(bow_dict, f)

    print('Bag of wrods created')

    return bow_dict

#method to create histogram from local features by mathcing with bag of words.
#Euclidiean distance is used to measure the similarity.
def create_feature_bow(image_descriptors, BoW, num_cluster,train):
    train_test = "train"
    if(not train):
        train_test="test"

    X_features = []

    for i in range(len(image_descriptors)):
        features = np.array([0] * num_cluster)

        if image_descriptors[i] is not None:
            distance = cdist(image_descriptors[i], BoW)

            argmin = np.argmin(distance, axis = 1)

            for j in argmin:
                features[j] += 1
        X_features.append(features)

    np.save(f'data/sift_hist_features_{train_test}.npy',X_features)

    print(f'SIFT Hist Features created in data/sift_hist_features_{train_test}.
↪npy')

    return X_features
```

[13]:
```
#extract_SIFT_features(loader=train_sampled_dataloader,train=True)
#extract_SIFT_features(loader=test_set_loader,train=False)
```

```
SIFT Features created in data/sift_featurestest.npy
```

1.0.27 Creating Bag of visual words

[25]:
```
image_descriptors = np.load("data/sift_features_train.npy",allow_pickle=True)
all_descriptors = []
#for descriptor in image_descriptors:
#    if descriptor is not None:
#        for des in descriptor:
#            all_descriptors.append(des)

num_cluster = 100
#BoW = kmean_bow(all_descriptors, num_cluster)
```

```
C:\Users\kshit\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
Bag of wrods created
```

1.0.28 Creating Histogram from Bag of visual words

[36]:
```
image_test_descriptors = np.load("data/sift_features_test.
↪npy",allow_pickle=True)
#X_features = create_feature_bow(image_descriptors, BoW, num_cluster,True)
#X_test_features = create_feature_bow(image_test_descriptors, BoW,␣
↪num_cluster,False)
```

```
SIFT Hist Features created in data/sift_hist_features_train.npy
SIFT Hist Features created in data/sift_hist_features_test.npy
```

[36]:
```
train_dict = load_data("data/hog_features_train.npy","data/labels_train.npy")
test_dict = load_data("data/hog_features_test.npy","data/labels_test.npy")
```

1.0.29 Training SVM with SIFT

[49]:
```
model_svm = SVC(C = 30, max_iter=1000, random_state = 0)
```

[50]:
```
#model_svm.fit(np.load("data/sift_hist_features_train.npy"), train_dict['Y'])
#with open("models/bow_svm_kmeans.bin", 'wb') as f:
    #pickle.dump(model_svm, f)
```

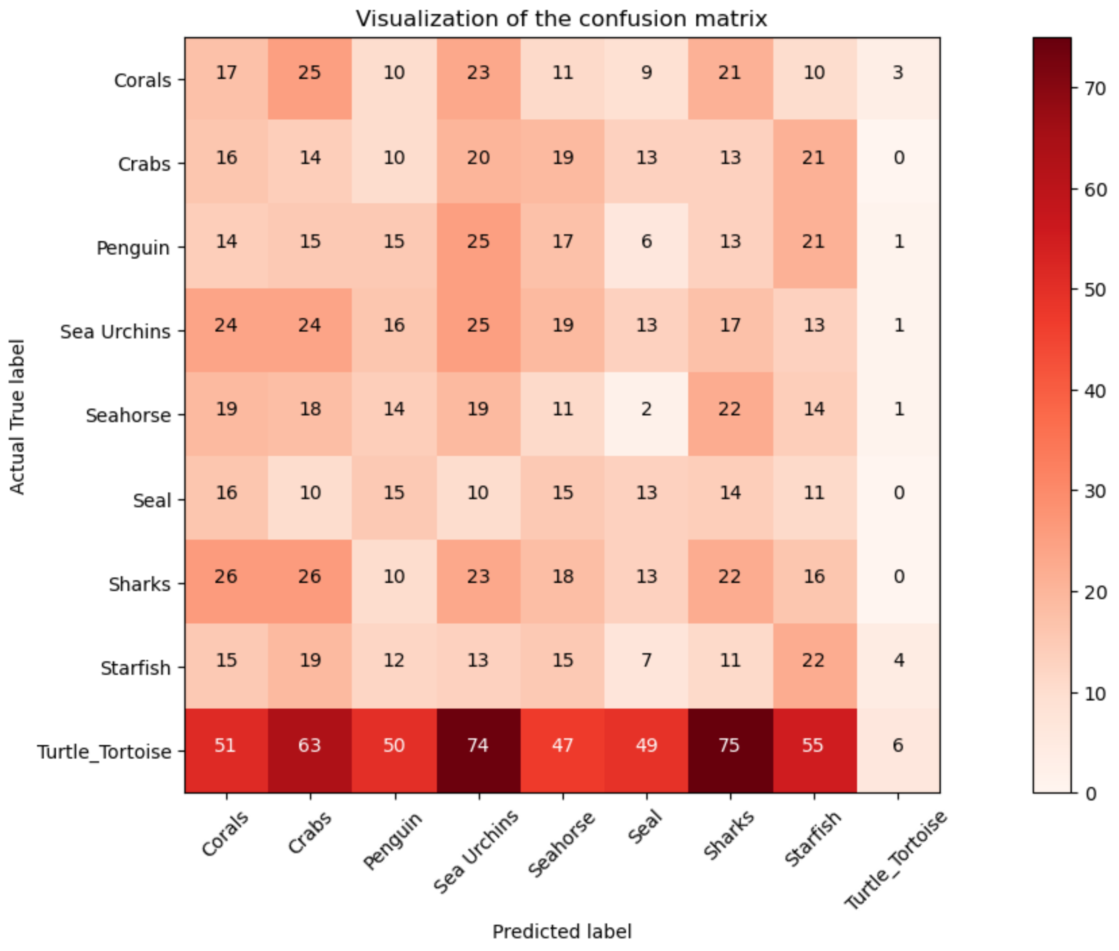
```
C:\Users\kshit\anaconda3\lib\site-packages\sklearn\svm\_base.py:299:
ConvergenceWarning: Solver terminated early (max_iter=1000).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
```

1.0.30 Testing SVM model trained with SIFT

[39]:
```
def run_bow_svm_test(test_features,test_labels):

    with open("models/bow_svm_kmeans.bin", 'rb') as f:
        model = pickle.load(f)

    targets=test_labels
    predictions = model.predict(test_features)

    accuracy = accuracy_score(targets, predictions) * 100
    f1 = f1_score(targets, predictions, average='macro')
    recall = recall_score(targets, predictions, average='macro')
    precision = precision_score(targets, predictions, average='macro')
    conf_matrix = confusion_matrix(targets, predictions)

    pretty_print_metrics([accuracy,f1,recall,precision])
    plot_cm(np.array(conf_matrix), CLASSES)
```

```
run_bow_svm_test(np.load("data/sift_hist_features_test.npy"),test_dict['Y'])
```

```
-- TEST SET METRICS --
Accuracy: 9.667%
F-score: 0.101
Recall: 0.120
Precision: 0.125
Confusion matrix
```



1.1 Results and Conclusion

We have trained total four models with 4 different approaches:

- Using Pre-trained models
- Using our own CNN architecture
- Using HOG features with SVM
- Using SIFT features with SVM

Following are the experimental results given in the table below:

1.1.1 Experimental Results

Model/Metric	Accuracy	F-Socre	Recall	Precision
ResNet50(Pretrained)	64.400	0.692	0.844	0.595
Custom-CNN	73.600	0.780	0.896	0.765
SVM with HOG features	46.333	0.492	0.595	0.437
SVM with SIFT features	9.667	0.101	0.120	0.125

Interpreting Results From the results we can observe that the Custom-CNN model achieves highest accuracy on test dataset followed by the pretrained model. Models trained using HOG and SIFT features does not perform well. The Custom-CNN trains on the training data from beginning and the layers learn the data from scratch therefore it is able to predict the data more accurately. The resnet50 model is pre-trained on ImageNet dataset which contains natural images. Therefore the resnet50 model tries to correlate with the knowledge it gained from ImageNet dataset to the sea animals dataset. The resnet model is able to classify most of the classes correctly except the turtle class. The custom CNN model is able to classify turtle class better. However models trained using hand crafted features like HOG and SIFT fails to classify most of the classes correctly. If we analyze the HOG images we can see that the HOG images of corals and turtle is similar and most of the images are similar so the model is not able to accurately classify most of the images. SVM with SIFT feature model fails with the least accuracy becuase most of the images contain water and the colors are similar in most of the images therefore there are no distinction between SIFT features among classes and therefore model is unable to classify correctly.

1.1.2 Conclusion

In conclusion we can say that Deep CNN architecture are better suited for image classification problems on natural images than training on extracted features like HOG and SIFT. Deep CNN architectures are very powerful in predicting images if the number of training samples are large. However extracted features are useful if the number of training data is small and it is easy to distinguish shapes in the images. Therefore deep learning models are able to generalize better for real world problems than hand crafted features.

1.1.3 References

[1] https://github.com/myconcordia/COMP478

[2] Z. Cao, J. C. Principe, B. Ouyang, F. Dalgleish and A. Vuorenkoski, “Marine animal classification using combined CNN and hand-designed image features,” OCEANS 2015 - MTS/IEEE Washington, Washington, DC, USA, 2015, pp. 1-6, doi: 10.23919/OCEANS.2015.7404375.

[3] N. N, A. Siva Kumaran K, A. A, A. V. S and B. M. J, “Convolutional Neural Networks (CNN) based Marine Species Identification,” 2022 International Conference on Automation, Computing and Renewable Systems (ICACRS), Pudukkottai, India, 2022, pp. 602-607, doi: 10.1109/ICACRS55517.2022.10029109.

[4] N. N, A. Siva Kumaran K, A. A, A. V. S and B. M. J, “Convolutional Neural Networks (CNN) based Marine Species Identification,” 2022 International Conference on Automa-

tion, Computing and Renewable Systems (ICACRS), Pudukkottai, India, 2022, pp. 602-607, doi: 10.1109/ICACRS55517.2022.10029109.

[5] X. Li and X. Guo, “A HOG Feature and SVM Based Method for Forward Vehicle Detection with Single Camera,” 2013 5th International Conference on Intelligent Human-Machine Systems and Cybernetics, Hangzhou, China, 2013, pp. 263-266, doi: 10.1109/IHMSC.2013.69.

[6] A. P. Puspaningrum et al., “Waste Classification Using Support Vector Machine with SIFT-PCA Feature Extraction,” 2020 4th International Conference on Informatics and Computational Sciences (ICICoS), Semarang, Indonesia, 2020, pp. 1-6, doi: 10.1109/ICICoS51170.2020.9298982.

[7] Q. Li and X. Wang, “Image Classification Based on SIFT and SVM,” 2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS), Singapore, 2018, pp. 762-765, doi: 10.1109/ICIS.2018.8466432.