Coursework 2: Fish Classification

Created by Athanasios Vlontzos and Wenjia Bai

In this coursework, you will be exploring the application of convolutional neural networks for image classification tasks. As opposed to standard applications such as object or face classification, we will be dealing with a slightly different domain, fish classification for precision fishing.

In precision fishing, engineers and fishmen collaborate to extract a wide variety of information about the fish, their species and wellbeing etc. using data from satellite images to drones surveying the fisheries. The goal of precision fishing is to provide the marine industry with information to support their decision making processes.

Here your will develop an image classification model that can classify fish species given input images. It consists of two tasks. The first task is to train a model for the following species:

- Black Sea Sprat
- Gilt-Head Bream
- Shrimp
- Striped Red Mullet
- Trout

The second task is to finetune the last layer of the trained model to adapt to some new species, including:

- Hourse Mackerel
- Red Mullet
- Red Sea Bream
- Sea Bass

You will be working using a large-scale fish dataset [1].

[1] O. Ulucan, D. Karakaya and M. Turkan. A large-scale dataset for fish segmentation and classification. Innovations in Intelligent Systems and Applications Conference (ASYU). 2020.

Step 0: Download data.

Download the Data from here -- make sure you access it with your Imperial account.

It is a ~2.5GB file. You can save the images and annotations directories in the same directory as this notebook or somewhere else.

The fish dataset contains 9 species of fishes. There are 1,000 images for each fish species, named as %05d.png in each subdirectory.

Step 1: Load the data. (15 Points)

- Complete the dataset class with the skeleton below.
- Add any transforms you feel are necessary.

Your class should have at least 3 elements

- An `__init__` function that sets up your class and all the necessary parameters.
- An `__len__` function that returns the size of your dataset.
- An `__getitem__` function that given an index within the limits of the size of the dataset returns the associated image and label in tensor form.

You may add more helper functions if you want.

In this section we are following the Pytorch dataset class structure. You can take inspiration from their documentation.

In [1]:
```
# Dependencies
import pandas as pd
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
import os
from PIL import Image
import numpy as np
from tqdm import tqdm
import torch
import torch.nn as nn
import torch.nn.functional as F
import matplotlib.pyplot as plt
import glob
from sklearn.metrics import accuracy_score, ConfusionMatrixDisplay
```

In [2]:
```
# We will start by building a dataset class using the following 5 species of
Multiclass_labels_correspondances = {
    'Black Sea Sprat': 0,
    'Gilt-Head Bream': 1,
    'Shrimp': 2,
    'Striped Red Mullet': 3,
    'Trout': 4
}

# The 5 species will contain 5,000 images in total.
# Let us split the 5,000 images into training (80%) and test (20%) sets
def split_train_test(lendata, percentage=0.8):
    indices = np.arange(0, lendata - 1, 1)
    np.random.shuffle(indices)

    limit = int(percentage * lendata)

    idxs_train = indices[:limit]
    idxs_test = indices[limit:]

    return idxs_train, idxs_test

LENDATA = 5000
np.random.seed(42)
idxs_train, idxs_test = split_train_test(LENDATA, 0.8)

# Implement the dataset class
class FishDataset(Dataset):
    def __init__(self,
                 path_to_images,
                 idxs_train,
                 idxs_test,
                 transform_extra=None,
                 img_size=128,
                 train=True):
        # path_to_images: where you put the fish dataset
        # idxs_train: training set indexes
        # idxs_test: test set indexes
        # transform_extra: extra data transform
        # img_size: resize all images to a standard size
        # train: return training set or test set

        # Load all the images and their labels
        images = []
        labels = []

        for label in Multiclass_labels_correspondances:
            print(path_to_images + '/' + label)

            files = glob.glob(path_to_images + '/' + label + '/*.png')

            print("Number of files: ", len(files))

            for filename in files:
                image = Image.open(filename)

                # Resize all images to a standard size
                new_image = image.resize((img_size, img_size))
```

```
        idxs = idxs_train if train else idxs_test

        # Extract the images and labels with the specified file indexes
        self.images = np.array(images)[idxs]
        self.labels = np.array(labels)[idxs]

        self.transform = transform_extra

        self.no_labels = len(Multiclass_labels_correspondances)
        self.one_hot = F.one_hot(torch.arange(0, self.no_labels) % self.no_l


    def __len__(self):
        # Return the number of samples
        return len(self.images)


    def get_one_hot_vector(self, label):
        pos = Multiclass_labels_correspondances[label]
        return self.one_hot[pos]


    def __getitem__(self, idx):
        # Get an item using its index
        # Return the image and its label
        image = self.images[idx]

        if self.transform is not None:
            image = self.transform(image)

        label_one_hot_vector = self.get_one_hot_vector(self.labels[idx])

        return image, label_one_hot_vector


    def get_number_samples_from_class(self, label):
        images_with_labels = np.array([self.images, self.labels]).reshape(se
        idxs = np.array([l == label for l in self.labels])
        images_with_label = images_with_labels[idxs]
        return (images_with_label.shape[0], images_with_label[0][0])
```

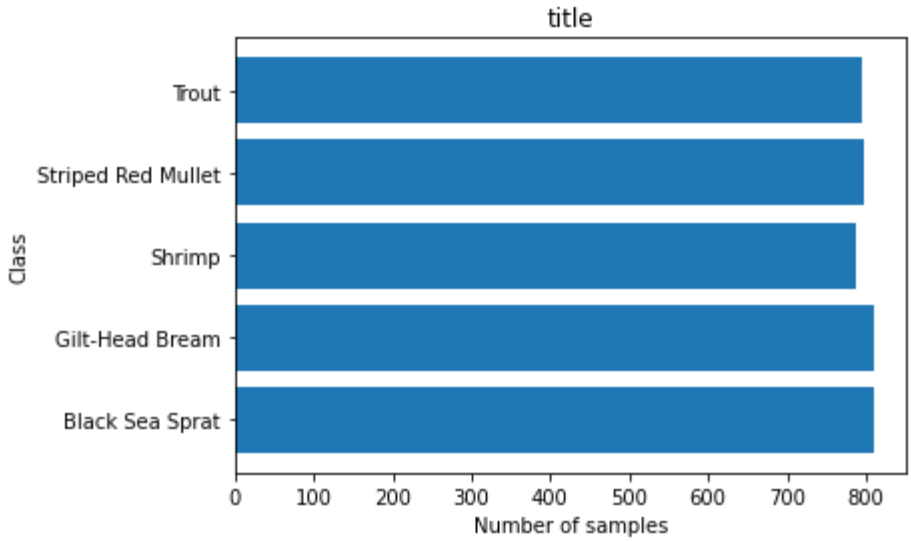
Step 2: Explore the data. (15 Points)

Step 2.1: Data visualisation. (5 points)

- Plot data distribution, i.e. the number of samples per class.
- Plot 1 sample from each of the five classes in the training set.

In [3]:
```
# Training set
img_path = 'Fish_Dataset'
dataset = FishDataset(img_path, idxs_train, idxs_test, None, img_size=128, t

# Plot the number of samples per class
samples_per_class = []
sample_per_class = []


for label in Multiclass_labels_correspondances:
    number, image = dataset.get_number_samples_from_class(label)
    samples_per_class.append(number)
    sample_per_class.append(image)

plt.barh([*Multiclass_labels_correspondances], samples_per_class)
plt.title("title")
plt.ylabel("Class")
plt.xlabel("Number of samples")
plt.show()

# Plot 1 sample from each of the five classes in the training set
fig, axs = plt.subplots(1, len(Multiclass_labels_correspondances))
fig.set_size_inches(18.5, 10.5, forward=True)
for img, ax, label in zip(sample_per_class, axs, Multiclass_labels_correspor
    ax.imshow(img)
    ax.axis('off')
    ax.set_title(label)
plt.show()
```

```
Fish_Dataset/Black Sea Sprat
Number of files:  1000
Fish_Dataset/Gilt-Head Bream
Number of files:  1000
Fish_Dataset/Shrimp
Number of files:  1000
Fish_Dataset/Striped Red Mullet
Number of files:  1000
Fish_Dataset/Trout
Number of files:  1000
/tmp/ipykernel_62299/593882310.py:71: FutureWarning: The input object of ty
pe 'Image' is an array-like implementing one of the corresponding protocols
(`__array__`, `__array_interface__` or `__array_struct__`); but not a seque
nce (or 0-D). In the future, this object will be coerced as if it was first
converted using `np.array(obj)`. To retain the old behaviour, you have to e
ither modify the type 'Image', or assign to an empty array created with `n
p.empty(correct_shape, dtype=object)`.
  self.images = np.array(images)[idxs]
/tmp/ipykernel_62299/593882310.py:71: VisibleDeprecationWarning: Creating a
n ndarray from ragged nested sequences (which is a list-or-tuple of lists-o
r-tuples-or ndarrays with different lengths or shapes) is deprecated. If yo
u meant to do this, you must specify 'dtype=object' when creating the ndarr
ay.
  self.images = np.array(images)[idxs]
```



Step 2.2: Discussion. (10 points)

- Is the dataset balanced?
- Can you think of 3 ways to make the dataset balanced if it is not?
- Is the dataset already pre-processed? If yes, how?

Yes, the dataset is balanced because we have approximately 1000 images per class (aproximatively 800 for each class in the training set and 200 in the testing set).

If the dataset is not balanced we could:

- use weights: classes that have more images have a lower weight than the classes with less images
- over-sampling: increase the dataset of the classes with less images by repeating them
- under-sampling: decrease the dataset of the classes with more images by randomly dropping some of them

Yes, the dataset is already pre-processed (the images are associated labels as they are separated in different folders).

Step 3: Multiclass classification. (55 points)

In this section we will try to make a multiclass classifier to determine the species of the fish.

Step 3.1: Define the model. (15 points)

Design a neural network which consists of a number of convolutional layers and a few fully connected ones at the end.

The exact architecture is up to you but you do NOT need to create something complicated. For example, you could design a LeNet insprired network.

In [4]:
```
class Net(nn.Module):
    def __init__(self, output_dims = 1):
        super(Net, self).__init__()

        # I followed the LeNet design and I used ReLU as the activation func
        self.features = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=6, kernel_size=5),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.classifier = nn.Sequential(
            nn.Linear(in_features=16 * 5 * 5, out_features=120),
            nn.ReLU(inplace=True),
            nn.Linear(in_features=120, out_features=84),
            nn.ReLU(inplace=True),
            nn.Linear(in_features=84, out_features=output_dims),
        )


    def forward(self, x):
        # Forward propagation
        x = self.features(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x

# Since most of you use laptops, you may use CPU for training.
# If you have a good GPU, you can set this to 'gpu'.
device = 'cpu'
```

Step 3.2: Define the training parameters. (10 points)

- Loss function
- Optimizer
- Learning Rate
- Number of iterations
- Batch Size
- Other relevant hyperparameters

In [5]:
```
# Network
model = Net(len(Multiclass_labels_correspondances))
model = model.to(device)

# Loss function
criterion = nn.CrossEntropyLoss()

# Optimiser and learning rate
lr = 0.01
optimizer = torch.optim.SGD(model.parameters(), lr=lr)

# Number of iterations for training
epochs = 100

# Training batch size
train_batch_size = 100

# Based on the FishDataset, use the PyTorch DataLoader to load the data duri
train_dataset = FishDataset(img_path, idxs_train, idxs_test, transforms.ToTe
train_dataloader = DataLoader(train_dataset, batch_size=train_batch_size)
test_dataset = FishDataset(img_path, idxs_train, idxs_test, transforms.ToTer
test_dataloader = DataLoader(test_dataset)
```

```
Fish_Dataset/Black Sea Sprat
Number of files:  1000
Fish_Dataset/Gilt-Head Bream
Number of files:  1000
Fish_Dataset/Shrimp
Number of files:  1000
Fish_Dataset/Striped Red Mullet
Number of files:  1000
Fish_Dataset/Trout
Number of files:  1000
```

```
/tmp/ipykernel_62299/593882310.py:71: FutureWarning: The input object of ty
pe 'Image' is an array-like implementing one of the corresponding protocols
(`__array__`, `__array_interface__` or `__array_struct__`); but not a seque
nce (or 0-D). In the future, this object will be coerced as if it was first
converted using `np.array(obj)`. To retain the old behaviour, you have to e
ither modify the type 'Image', or assign to an empty array created with `n
p.empty(correct_shape, dtype=object)`.
  self.images = np.array(images)[idxs]
/tmp/ipykernel_62299/593882310.py:71: VisibleDeprecationWarning: Creating a
n ndarray from ragged nested sequences (which is a list-or-tuple of lists-o
r-tuples-or ndarrays with different lengths or shapes) is deprecated. If yo
u meant to do this, you must specify 'dtype=object' when creating the ndarr
ay.
  self.images = np.array(images)[idxs]
Fish_Dataset/Black Sea Sprat
Number of files:  1000
Fish_Dataset/Gilt-Head Bream
Number of files:  1000
Fish_Dataset/Shrimp
Number of files:  1000
Fish_Dataset/Striped Red Mullet
Number of files:  1000
Fish_Dataset/Trout
Number of files:  1000
```

Step 3.3: Train the model. (15 points)

Complete the training loop.

In [6]:
```
for epoch in tqdm(range(epochs)):
    model.train()
    loss_curve = []

    for images, targets in train_dataloader:

        images = images.to(device)
        targets = targets.to(device)

        optimizer.zero_grad()
        outputs = model.forward(images)

        loss = criterion(outputs, targets)
        loss.backward()
        loss_curve += [loss.item()]

        optimizer.step()

    print('--- Iteration {0}: training loss = {1:.4f} ---'.format(epoch + 1,
```

```
  1%|
| 1/100 [00:00<00:55,  1.78it/s]
--- Iteration 1: training loss = 1.6108 ---
  2%|
| 2/100 [00:01<00:54,  1.81it/s]
--- Iteration 2: training loss = 1.6100 ---
```

```
  3%|███
| 3/100 [00:01<00:52,  1.86it/s]
--- Iteration 3: training loss = 1.6094 ---
  4%|████
| 4/100 [00:02<00:51,  1.86it/s]
--- Iteration 4: training loss = 1.6087 ---
  5%|█████
| 5/100 [00:02<00:49,  1.92it/s]
--- Iteration 5: training loss = 1.6080 ---
  6%|██████
| 6/100 [00:03<00:49,  1.90it/s]
--- Iteration 6: training loss = 1.6073 ---
  7%|███████
| 7/100 [00:03<00:48,  1.93it/s]
--- Iteration 7: training loss = 1.6066 ---
  8%|████████
| 8/100 [00:04<00:47,  1.95it/s]
--- Iteration 8: training loss = 1.6059 ---
  9%|█████████
| 9/100 [00:04<00:46,  1.96it/s]
--- Iteration 9: training loss = 1.6052 ---
 10%|██████████
| 10/100 [00:05<00:45,  1.97it/s]
--- Iteration 10: training loss = 1.6045 ---
 11%|███████████
| 11/100 [00:05<00:45,  1.97it/s]
--- Iteration 11: training loss = 1.6036 ---
 12%|████████████
| 12/100 [00:06<00:44,  1.98it/s]
--- Iteration 12: training loss = 1.6027 ---
 13%|█████████████
| 13/100 [00:06<00:44,  1.93it/s]
--- Iteration 13: training loss = 1.6015 ---
 14%|██████████████
| 14/100 [00:07<00:44,  1.92it/s]
--- Iteration 14: training loss = 1.6002 ---
 15%|███████████████
| 15/100 [00:07<00:43,  1.95it/s]
--- Iteration 15: training loss = 1.5986 ---
 16%|████████████████
| 16/100 [00:08<00:43,  1.93it/s]
--- Iteration 16: training loss = 1.5965 ---
 17%|█████████████████
| 17/100 [00:08<00:42,  1.95it/s]
--- Iteration 17: training loss = 1.5940 ---
 18%|██████████████████
| 18/100 [00:09<00:42,  1.92it/s]
--- Iteration 18: training loss = 1.5907 ---
 19%|███████████████████
| 19/100 [00:09<00:41,  1.93it/s]
--- Iteration 19: training loss = 1.5861 ---
 20%|████████████████████
| 20/100 [00:10<00:41,  1.91it/s]
```

```
--- Iteration 20: training loss = 1.5796 ---
 21%|
| 21/100 [00:10<00:41,  1.88it/s]
--- Iteration 21: training loss = 1.5699 ---
 22%|
| 22/100 [00:11<00:42,  1.84it/s]
--- Iteration 22: training loss = 1.5548 ---
 23%|
| 23/100 [00:12<00:41,  1.84it/s]
--- Iteration 23: training loss = 1.5299 ---
 24%|
| 24/100 [00:12<00:41,  1.85it/s]
--- Iteration 24: training loss = 1.4870 ---
 25%|
| 25/100 [00:13<00:41,  1.81it/s]
--- Iteration 25: training loss = 1.4188 ---
 26%|
| 26/100 [00:13<00:41,  1.78it/s]
--- Iteration 26: training loss = 1.3406 ---
 27%|
| 27/100 [00:14<00:40,  1.80it/s]
--- Iteration 27: training loss = 1.2962 ---
 28%|
| 28/100 [00:14<00:39,  1.81it/s]
--- Iteration 28: training loss = 1.2826 ---
 29%|
| 29/100 [00:15<00:38,  1.85it/s]
--- Iteration 29: training loss = 1.2671 ---
 30%|
| 30/100 [00:15<00:37,  1.85it/s]
--- Iteration 30: training loss = 1.2518 ---
 31%|
| 31/100 [00:16<00:37,  1.82it/s]
--- Iteration 31: training loss = 1.2379 ---
 32%|
| 32/100 [00:16<00:36,  1.87it/s]
--- Iteration 32: training loss = 1.2256 ---
 33%|
| 33/100 [00:17<00:35,  1.87it/s]
--- Iteration 33: training loss = 1.2156 ---
 34%|
| 34/100 [00:18<00:34,  1.89it/s]
--- Iteration 34: training loss = 1.2061 ---
 35%|
| 35/100 [00:18<00:33,  1.92it/s]
--- Iteration 35: training loss = 1.1972 ---
 36%|
| 36/100 [00:19<00:33,  1.90it/s]
--- Iteration 36: training loss = 1.1875 ---
 37%|
| 37/100 [00:19<00:33,  1.91it/s]
--- Iteration 37: training loss = 1.1791 ---
```

```
 38%|
| 38/100 [00:20<00:32,  1.91it/s]
--- Iteration 38: training loss = 1.1695 ---
 39%|
| 39/100 [00:20<00:31,  1.92it/s]
--- Iteration 39: training loss = 1.1608 ---
 40%|
| 40/100 [00:21<00:30,  1.94it/s]
--- Iteration 40: training loss = 1.1519 ---
 41%|
| 41/100 [00:21<00:30,  1.94it/s]
--- Iteration 41: training loss = 1.1424 ---
 42%|
| 42/100 [00:22<00:30,  1.91it/s]
--- Iteration 42: training loss = 1.1318 ---
 43%|
| 43/100 [00:22<00:29,  1.92it/s]
--- Iteration 43: training loss = 1.1206 ---
 44%|
| 44/100 [00:23<00:29,  1.92it/s]
--- Iteration 44: training loss = 1.1101 ---
 45%|
| 45/100 [00:23<00:29,  1.85it/s]
--- Iteration 45: training loss = 1.0984 ---
 46%|
| 46/100 [00:24<00:29,  1.82it/s]
--- Iteration 46: training loss = 1.0861 ---
 47%|
| 47/100 [00:24<00:28,  1.85it/s]
--- Iteration 47: training loss = 1.0724 ---
 48%|
| 48/100 [00:25<00:27,  1.87it/s]
--- Iteration 48: training loss = 1.0589 ---
 49%|
| 49/100 [00:25<00:27,  1.84it/s]
--- Iteration 49: training loss = 1.0416 ---
 50%|
| 50/100 [00:26<00:27,  1.83it/s]
--- Iteration 50: training loss = 1.0203 ---
 51%|
| 51/100 [00:27<00:26,  1.82it/s]
--- Iteration 51: training loss = 0.9975 ---
 52%|
| 52/100 [00:27<00:25,  1.86it/s]
--- Iteration 52: training loss = 0.9807 ---
 53%|
| 53/100 [00:28<00:25,  1.87it/s]
--- Iteration 53: training loss = 0.9578 ---
 54%|
| 54/100 [00:28<00:24,  1.88it/s]
--- Iteration 54: training loss = 0.9347 ---
 55%|
| 55/100 [00:29<00:23,  1.90it/s]
```

```
--- Iteration 55: training loss = 0.9041 ---
 56%|
| 56/100 [00:29<00:22,  1.93it/s]
--- Iteration 56: training loss = 0.8779 ---
 57%|
| 57/100 [00:30<00:21,  1.98it/s]
--- Iteration 57: training loss = 0.8193 ---
 58%|
| 58/100 [00:30<00:20,  2.02it/s]
--- Iteration 58: training loss = 0.8390 ---
 59%|
| 59/100 [00:31<00:19,  2.07it/s]
--- Iteration 59: training loss = 0.7642 ---
 60%|
| 60/100 [00:31<00:19,  2.09it/s]
--- Iteration 60: training loss = 0.7554 ---
 61%|
| 61/100 [00:31<00:18,  2.11it/s]
--- Iteration 61: training loss = 0.7344 ---
 62%|
| 62/100 [00:32<00:18,  2.11it/s]
--- Iteration 62: training loss = 0.7159 ---
 63%|
| 63/100 [00:32<00:17,  2.11it/s]
--- Iteration 63: training loss = 0.6813 ---
 64%|
| 64/100 [00:33<00:18,  1.99it/s]
--- Iteration 64: training loss = 0.6668 ---
 65%|
| 65/100 [00:33<00:17,  2.00it/s]
--- Iteration 65: training loss = 0.6395 ---
 66%|
| 66/100 [00:34<00:16,  2.04it/s]
--- Iteration 66: training loss = 0.6236 ---
 67%|
| 67/100 [00:34<00:15,  2.07it/s]
--- Iteration 67: training loss = 0.6026 ---
 68%|
| 68/100 [00:35<00:15,  2.08it/s]
--- Iteration 68: training loss = 0.5853 ---
 69%|
| 69/100 [00:35<00:14,  2.10it/s]
--- Iteration 69: training loss = 0.5673 ---
 70%|
| 70/100 [00:36<00:14,  2.10it/s]
--- Iteration 70: training loss = 0.5444 ---
 71%|
| 71/100 [00:36<00:14,  2.03it/s]
--- Iteration 71: training loss = 0.5166 ---
 72%|
| 72/100 [00:37<00:13,  2.06it/s]
--- Iteration 72: training loss = 0.5338 ---
```

```
 73%|                                                                                | 73/100 [00:37<00:12,  2.09it/s]
--- Iteration 73: training loss = 0.4827 ---
 74%|                                                                                | 74/100 [00:38<00:12,  2.10it/s]
--- Iteration 74: training loss = 0.4785 ---
 75%|                                                                                | 75/100 [00:38<00:11,  2.10it/s]
--- Iteration 75: training loss = 0.4796 ---
 76%|                                                                                | 76/100 [00:39<00:11,  2.08it/s]
--- Iteration 76: training loss = 0.4790 ---
 77%|                                                                                | 77/100 [00:39<00:11,  2.00it/s]
--- Iteration 77: training loss = 0.4575 ---
 78%|                                                                                | 78/100 [00:40<00:11,  1.94it/s]
--- Iteration 78: training loss = 0.4197 ---
 79%|                                                                                | 79/100 [00:40<00:10,  1.93it/s]
--- Iteration 79: training loss = 0.4353 ---
 80%|                                                                                | 80/100 [00:41<00:10,  1.95it/s]
--- Iteration 80: training loss = 0.4273 ---
 81%|                                                                                | 81/100 [00:41<00:09,  1.98it/s]
--- Iteration 81: training loss = 0.3855 ---
 82%|                                                                                | 82/100 [00:42<00:09,  1.98it/s]
--- Iteration 82: training loss = 0.3787 ---
 83%|                                                                                | 83/100 [00:42<00:08,  1.97it/s]
--- Iteration 83: training loss = 0.3796 ---
 84%|                                                                                | 84/100 [00:43<00:08,  1.91it/s]
--- Iteration 84: training loss = 0.3853 ---
 85%|                                                                                | 85/100 [00:43<00:07,  1.91it/s]
--- Iteration 85: training loss = 0.3342 ---
 86%|                                                                                | 86/100 [00:44<00:07,  1.92it/s]
--- Iteration 86: training loss = 0.3383 ---
 87%|                                                                                | 87/100 [00:44<00:06,  1.95it/s]
--- Iteration 87: training loss = 0.3092 ---
 88%|                                                                                | 88/100 [00:45<00:06,  1.96it/s]
--- Iteration 88: training loss = 0.3371 ---
 89%|                                                                                | 89/100 [00:45<00:05,  1.96it/s]
--- Iteration 89: training loss = 0.3357 ---
 90%|                                                                                | 90/100 [00:46<00:05,  1.93it/s]
```

```
--- Iteration 90: training loss = 0.2866 ---
 91%|██████████████████████████████████████████████████████████████████████
███████████████████████████          | 91/100 [00:47<00:04,  1.93it/s]
--- Iteration 91: training loss = 0.2909 ---
 92%|██████████████████████████████████████████████████████████████████████
████████████████████████████         | 92/100 [00:47<00:04,  1.93it/s]
--- Iteration 92: training loss = 0.2774 ---
 93%|██████████████████████████████████████████████████████████████████████
█████████████████████████████        | 93/100 [00:48<00:03,  1.95it/s]
--- Iteration 93: training loss = 0.2624 ---
 94%|██████████████████████████████████████████████████████████████████████
██████████████████████████████       | 94/100 [00:48<00:03,  1.95it/s]
--- Iteration 94: training loss = 0.3169 ---
 95%|██████████████████████████████████████████████████████████████████████
███████████████████████████████      | 95/100 [00:49<00:02,  1.96it/s]
--- Iteration 95: training loss = 0.2434 ---
 96%|██████████████████████████████████████████████████████████████████████
████████████████████████████████     | 96/100 [00:49<00:02,  1.97it/s]
--- Iteration 96: training loss = 0.2292 ---
 97%|██████████████████████████████████████████████████████████████████████
█████████████████████████████████    | 97/100 [00:50<00:01,  1.96it/s]
--- Iteration 97: training loss = 0.2727 ---
 98%|██████████████████████████████████████████████████████████████████████
██████████████████████████████████   | 98/100 [00:50<00:01,  1.97it/s]
--- Iteration 98: training loss = 0.2386 ---
 99%|██████████████████████████████████████████████████████████████████████
███████████████████████████████████  | 99/100 [00:51<00:00,  1.95it/s]
--- Iteration 99: training loss = 0.2066 ---
100%|██████████████████████████████████████████████████████████████████████
███████████████████████████████████| 100/100 [00:51<00:00,  1.94it/s]
--- Iteration 100: training loss = 0.2363 ---
```

Step 3.4: Deploy the trained model onto the test set. (10 points)

In [7]:

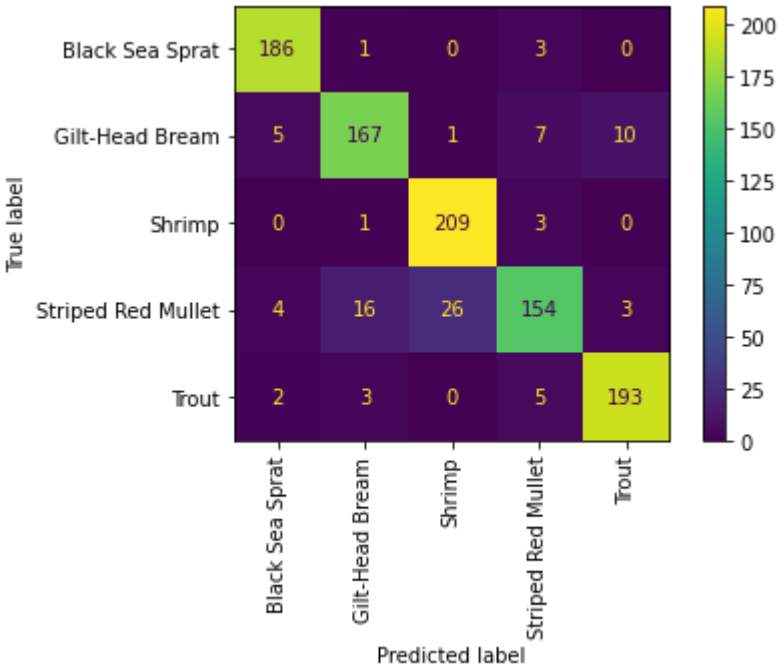
```python
# Deploy the model
for images, targets in test_dataloader:

    images = images.to(device)
    targets = targets.to(device)

    outputs = model.forward(images)
```

Step 3.5: Evaluate the performance of the model and visualize the confusion matrix. (5 points)

You can use sklearns related function.

In [8]:
```
y_test = test_dataset.labels
y_pred = []

for images, targets in test_dataloader:

    images = images.to(device)
    targets = targets.to(device)

    outputs = model.forward(images)

    _, predicted_label = torch.max(outputs.data, 1)
    predicted_label = predicted_label.item()
    predicted_label = [*Multiclass_labels_correspondances.keys()][predicted_
    y_pred.append(predicted_label)

ConfusionMatrixDisplay.from_predictions(y_test, y_pred, xticks_rotation="ver

print("Accuracy: ", accuracy_score(y_test, y_pred))
```

```
Accuracy:  0.9099099099099099
```



Step 4: Finetune your classifier. (15 points)

In the previous section, you have built a pretty good classifier for certain species of fish. Now we are going to use this trained classifier and adapt it to classify a new set of species:

```
'Hourse Mackerel
'Red Mullet',
'Red Sea Bream'
'Sea Bass'
```

Step 4.1: Set up the data for new species. (2 points)

Overwrite the labels correspondances so they only incude the new classes and regenerate the datasets and dataloaders.

In [9]:
```
Multiclass_labels_correspondances ={
    'Hourse Mackerel': 0,
    'Red Mullet': 1,
    'Red Sea Bream': 2,
    'Sea Bass': 3}

LENDATA = 4000
idxs_train,idxs_test = split_train_test(LENDATA, 0.8)

# Dataloaders
train_dataset = FishDataset(img_path, idxs_train, idxs_test, transforms.ToTe
train_dataloader = DataLoader(train_dataset, batch_size=train_batch_size)
test_dataset = FishDataset(img_path, idxs_train, idxs_test, transforms.ToTer
test_dataloader = DataLoader(test_dataset)
```

```
Fish_Dataset/Hourse Mackerel
Number of files:  1000
Fish_Dataset/Red Mullet
Number of files:  1000
Fish_Dataset/Red Sea Bream
Number of files:  1000
Fish_Dataset/Sea Bass
Number of files:  1000
/tmp/ipykernel_62299/593882310.py:71: FutureWarning: The input object of ty
pe 'Image' is an array-like implementing one of the corresponding protocols
(`__array__`, `__array_interface__` or `__array_struct__`); but not a seque
nce (or 0-D). In the future, this object will be coerced as if it was first
converted using `np.array(obj)`. To retain the old behaviour, you have to e
ither modify the type 'Image', or assign to an empty array created with `n
p.empty(correct_shape, dtype=object)`.
  self.images = np.array(images)[idxs]
/tmp/ipykernel_62299/593882310.py:71: VisibleDeprecationWarning: Creating a
n ndarray from ragged nested sequences (which is a list-or-tuple of lists-o
r-tuples-or ndarrays with different lengths or shapes) is deprecated. If yo
u meant to do this, you must specify 'dtype=object' when creating the ndarr
ay.
  self.images = np.array(images)[idxs]
```

```
Fish_Dataset/Hourse Mackerel
Number of files:  1000
Fish_Dataset/Red Mullet
Number of files:  1000
Fish_Dataset/Red Sea Bream
Number of files:  1000
Fish_Dataset/Sea Bass
Number of files:  1000
```

Step 4.2: Freeze the weights of all previous layers of the network except the last layer. (5 points)

You can freeze them by setting the gradient requirements to `False` .

In [10]:
```
def freeze_till_last(model):
    for param in model.parameters():
        param.requires_grad = False

model.classifier = nn.Sequential(
    nn.Linear(in_features=16 * 5 * 5, out_features=120),
    nn.ReLU(inplace=True),
    nn.Linear(in_features=120, out_features=84),
    nn.ReLU(inplace=True),
    nn.Linear(in_features=84, out_features=len(Multiclass_labels_corresponda
)

freeze_till_last(model)

for name, param in model.named_parameters():
    if "classifier" in name:
        param.requires_grad = True
        print(name)


# model.

# for layer in model.classifier:
#     layer.requires_grad = True

# Loss function
criterion = nn.CrossEntropyLoss()

# Optimiser and learning rate
lr = 0.01
optimizer = torch.optim.SGD(model.parameters(), lr=lr)

# Number of iterations for training
epochs = 100

# Training batch size
train_batch_size = 100
```

```
classifier.0.weight
classifier.0.bias
classifier.2.weight
classifier.2.bias
classifier.4.weight
classifier.4.bias
```

Step 4.3: Train and test your finetuned model. (5 points)

In [11]:
```
# Finetune the model
for epoch in tqdm(range(epochs)):
    model.train()
    loss_curve = []

    for images, targets in train_dataloader:

        images = images.to(device)
        targets = targets.to(device)

        optimizer.zero_grad()
        outputs = model.forward(images)

        loss = criterion(outputs, targets)
        loss.backward()
        loss_curve += [loss.item()]

        optimizer.step()

    print('--- Iteration {0}: training loss = {1:.4f} ---'.format(epoch + 1,

# Deploy the model on the test set
y_test = test_dataset.labels
y_pred = []

for images, targets in test_dataloader:

    images = images.to(device)
    targets = targets.to(device)

    outputs = model.forward(images)

    _, predicted_label = torch.max(outputs.data, 1)
    predicted_label = predicted_label.item()
    predicted_label = [*Multiclass_labels_correspondances.keys()][predicted_
    y_pred.append(predicted_label)

# Evaluate the performance
ConfusionMatrixDisplay.from_predictions(y_test, y_pred, xticks_rotation="ver

print("Accuracy: ", accuracy_score(y_test, y_pred))
```

```
  1%|
| 1/100 [00:00<00:39,  2.52it/s]
--- Iteration 1: training loss = 1.2883 ---
  2%|
| 2/100 [00:00<00:36,  2.67it/s]
--- Iteration 2: training loss = 1.1296 ---
```

```
  3%|███
| 3/100 [00:01<00:36,  2.68it/s]
--- Iteration 3: training loss = 1.0167 ---
  4%|████
| 4/100 [00:01<00:36,  2.64it/s]
--- Iteration 4: training loss = 0.9369 ---
  5%|█████
| 5/100 [00:01<00:34,  2.73it/s]
--- Iteration 5: training loss = 0.8790 ---
  6%|██████
| 6/100 [00:02<00:33,  2.84it/s]
--- Iteration 6: training loss = 0.8337 ---
  7%|███████
| 7/100 [00:02<00:32,  2.87it/s]
--- Iteration 7: training loss = 0.7960 ---
  8%|████████
| 8/100 [00:02<00:32,  2.83it/s]
--- Iteration 8: training loss = 0.7639 ---
  9%|█████████
| 9/100 [00:03<00:31,  2.85it/s]
--- Iteration 9: training loss = 0.7354 ---
 10%|██████████
| 10/100 [00:03<00:30,  2.95it/s]
--- Iteration 10: training loss = 0.7098 ---
 11%|███████████
| 11/100 [00:03<00:29,  3.01it/s]
--- Iteration 11: training loss = 0.6870 ---
 12%|████████████
| 12/100 [00:04<00:28,  3.06it/s]
--- Iteration 12: training loss = 0.6669 ---
 13%|█████████████
| 13/100 [00:04<00:28,  3.08it/s]
--- Iteration 13: training loss = 0.6493 ---
 14%|██████████████
| 14/100 [00:04<00:27,  3.10it/s]
--- Iteration 14: training loss = 0.6334 ---
 15%|███████████████
| 15/100 [00:05<00:27,  3.09it/s]
--- Iteration 15: training loss = 0.6167 ---
 16%|████████████████
| 16/100 [00:05<00:27,  3.07it/s]
--- Iteration 16: training loss = 0.6012 ---
 17%|█████████████████
| 17/100 [00:05<00:26,  3.07it/s]
--- Iteration 17: training loss = 0.5851 ---
 18%|██████████████████
| 18/100 [00:06<00:26,  3.06it/s]
--- Iteration 18: training loss = 0.5705 ---
 19%|███████████████████
| 19/100 [00:06<00:26,  3.06it/s]
--- Iteration 19: training loss = 0.5554 ---
 20%|████████████████████
| 20/100 [00:06<00:26,  3.07it/s]
```

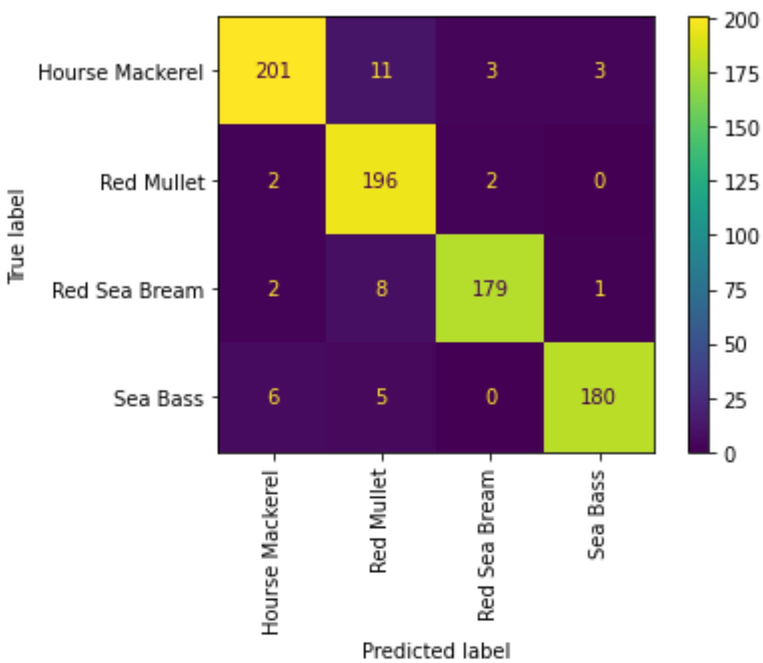
```
--- Iteration 20: training loss = 0.5404 ---
 21%|
| 21/100 [00:07<00:26,  3.00it/s]
--- Iteration 21: training loss = 0.5272 ---
 22%|
| 22/100 [00:07<00:26,  2.92it/s]
--- Iteration 22: training loss = 0.5136 ---
 23%|
| 23/100 [00:07<00:26,  2.89it/s]
--- Iteration 23: training loss = 0.4987 ---
 24%|
| 24/100 [00:08<00:25,  2.94it/s]
--- Iteration 24: training loss = 0.4866 ---
 25%|
| 25/100 [00:08<00:25,  2.95it/s]
--- Iteration 25: training loss = 0.4747 ---
 26%|
| 26/100 [00:08<00:25,  2.96it/s]
--- Iteration 26: training loss = 0.4643 ---
 27%|
| 27/100 [00:09<00:25,  2.85it/s]
--- Iteration 27: training loss = 0.4544 ---
 28%|
| 28/100 [00:09<00:25,  2.84it/s]
--- Iteration 28: training loss = 0.4460 ---
 29%|
| 29/100 [00:09<00:25,  2.82it/s]
--- Iteration 29: training loss = 0.4383 ---
 30%|
| 30/100 [00:10<00:24,  2.85it/s]
--- Iteration 30: training loss = 0.4277 ---
 31%|
| 31/100 [00:10<00:23,  2.92it/s]
--- Iteration 31: training loss = 0.4149 ---
 32%|
| 32/100 [00:10<00:23,  2.91it/s]
--- Iteration 32: training loss = 0.4082 ---
 33%|
| 33/100 [00:11<00:23,  2.86it/s]
--- Iteration 33: training loss = 0.4042 ---
 34%|
| 34/100 [00:11<00:23,  2.85it/s]
--- Iteration 34: training loss = 0.3968 ---
 35%|
| 35/100 [00:12<00:23,  2.82it/s]
--- Iteration 35: training loss = 0.3865 ---
 36%|
| 36/100 [00:12<00:22,  2.84it/s]
--- Iteration 36: training loss = 0.3780 ---
 37%|
| 37/100 [00:12<00:21,  2.90it/s]
--- Iteration 37: training loss = 0.3681 ---
```

```
 38%|
| 38/100 [00:13<00:21,  2.94it/s]
--- Iteration 38: training loss = 0.3604 ---
 39%|
| 39/100 [00:13<00:20,  2.98it/s]
--- Iteration 39: training loss = 0.3501 ---
 40%|
| 40/100 [00:13<00:20,  2.95it/s]
--- Iteration 40: training loss = 0.3458 ---
 41%|
| 41/100 [00:14<00:20,  2.85it/s]
--- Iteration 41: training loss = 0.3374 ---
 42%|
| 42/100 [00:14<00:20,  2.81it/s]
--- Iteration 42: training loss = 0.3268 ---
 43%|
| 43/100 [00:14<00:21,  2.63it/s]
--- Iteration 43: training loss = 0.3185 ---
 44%|
| 44/100 [00:15<00:21,  2.66it/s]
--- Iteration 44: training loss = 0.3137 ---
 45%|
| 45/100 [00:15<00:20,  2.68it/s]
--- Iteration 45: training loss = 0.2993 ---
 46%|
| 46/100 [00:15<00:19,  2.75it/s]
--- Iteration 46: training loss = 0.2995 ---
 47%|
| 47/100 [00:16<00:18,  2.82it/s]
--- Iteration 47: training loss = 0.3006 ---
 48%|
| 48/100 [00:16<00:18,  2.86it/s]
--- Iteration 48: training loss = 0.2703 ---
 49%|
| 49/100 [00:16<00:17,  2.92it/s]
--- Iteration 49: training loss = 0.2783 ---
 50%|
| 50/100 [00:17<00:17,  2.94it/s]
--- Iteration 50: training loss = 0.2986 ---
 51%|
| 51/100 [00:17<00:16,  2.94it/s]
--- Iteration 51: training loss = 0.2587 ---
 52%|
| 52/100 [00:17<00:16,  2.97it/s]
--- Iteration 52: training loss = 0.2501 ---
 53%|
| 53/100 [00:18<00:15,  2.96it/s]
--- Iteration 53: training loss = 0.2523 ---
 54%|
| 54/100 [00:18<00:16,  2.86it/s]
--- Iteration 54: training loss = 0.2819 ---
 55%|
| 55/100 [00:19<00:16,  2.80it/s]
```

```
--- Iteration 55: training loss = 0.2336 ---
 56%|
| 56/100 [00:19<00:15,  2.82it/s]
--- Iteration 56: training loss = 0.2283 ---
 57%|
| 57/100 [00:19<00:15,  2.79it/s]
--- Iteration 57: training loss = 0.2382 ---
 58%|
| 58/100 [00:20<00:14,  2.84it/s]
--- Iteration 58: training loss = 0.2457 ---
 59%|
| 59/100 [00:20<00:14,  2.87it/s]
--- Iteration 59: training loss = 0.2070 ---
 60%|
| 60/100 [00:20<00:14,  2.83it/s]
--- Iteration 60: training loss = 0.2049 ---
 61%|
| 61/100 [00:21<00:13,  2.79it/s]
--- Iteration 61: training loss = 0.2387 ---
 62%|
| 62/100 [00:21<00:13,  2.81it/s]
--- Iteration 62: training loss = 0.2033 ---
 63%|
| 63/100 [00:21<00:13,  2.73it/s]
--- Iteration 63: training loss = 0.1905 ---
 64%|
| 64/100 [00:22<00:13,  2.65it/s]
--- Iteration 64: training loss = 0.1822 ---
 65%|
| 65/100 [00:22<00:13,  2.61it/s]
--- Iteration 65: training loss = 0.3085 ---
 66%|
| 66/100 [00:23<00:12,  2.64it/s]
--- Iteration 66: training loss = 0.2027 ---
 67%|
                                   | 67/100 [00:23<00:12,  2.69it/s]
--- Iteration 67: training loss = 0.1821 ---
 68%|
                                   | 68/100 [00:23<00:11,  2.73it/s]
--- Iteration 68: training loss = 0.1725 ---
 69%|
                                   | 69/100 [00:24<00:11,  2.80it/s]
--- Iteration 69: training loss = 0.1709 ---
 70%|
                                   | 70/100 [00:24<00:10,  2.74it/s]
--- Iteration 70: training loss = 0.1826 ---
 71%|
                                   | 71/100 [00:24<00:11,  2.63it/s]
--- Iteration 71: training loss = 0.1505 ---
 72%|
                                   | 72/100 [00:25<00:10,  2.64it/s]
--- Iteration 72: training loss = 0.1600 ---
```

```
 73%|                                                                                | 73/100 [00:25<00:10,  2.68it/s]
--- Iteration 73: training loss = 0.1491 ---
 74%|                                                                                | 74/100 [00:26<00:09,  2.76it/s]
--- Iteration 74: training loss = 0.1447 ---
 75%|                                                                                | 75/100 [00:26<00:09,  2.74it/s]
--- Iteration 75: training loss = 0.1567 ---
 76%|                                                                                | 76/100 [00:26<00:08,  2.74it/s]
--- Iteration 76: training loss = 0.4739 ---
 77%|                                                                                | 77/100 [00:27<00:08,  2.77it/s]
--- Iteration 77: training loss = 0.1527 ---
 78%|                                                                                | 78/100 [00:27<00:08,  2.75it/s]
--- Iteration 78: training loss = 0.1656 ---
 79%|                                                                                | 79/100 [00:27<00:07,  2.75it/s]
--- Iteration 79: training loss = 0.1601 ---
 80%|                                                                                | 80/100 [00:28<00:07,  2.71it/s]
--- Iteration 80: training loss = 0.1365 ---
 81%|                                                                                | 81/100 [00:28<00:07,  2.71it/s]
--- Iteration 81: training loss = 0.1281 ---
 82%|                                                                                | 82/100 [00:28<00:06,  2.67it/s]
--- Iteration 82: training loss = 0.1227 ---
 83%|                                                                                | 83/100 [00:29<00:06,  2.70it/s]
--- Iteration 83: training loss = 0.1186 ---
 84%|                                                                                | 84/100 [00:29<00:05,  2.70it/s]
--- Iteration 84: training loss = 0.1150 ---
 85%|                                                                                | 85/100 [00:30<00:05,  2.69it/s]
--- Iteration 85: training loss = 0.1109 ---
 86%|                                                                                | 86/100 [00:30<00:05,  2.67it/s]
--- Iteration 86: training loss = 0.1080 ---
 87%|                                                                                | 87/100 [00:30<00:04,  2.63it/s]
--- Iteration 87: training loss = 0.1041 ---
 88%|                                                                                | 88/100 [00:31<00:04,  2.70it/s]
--- Iteration 88: training loss = 0.1022 ---
 89%|                                                                                | 89/100 [00:31<00:04,  2.72it/s]
--- Iteration 89: training loss = 0.0984 ---
 90%|                                                                                | 90/100 [00:31<00:03,  2.72it/s]
```

```
--- Iteration 90: training loss = 0.0977 ---
 91%|██████████████████████████████████████████████████████████████████████████████████████████          | 91/100 [00:32<00:03,  2.79it/s]
--- Iteration 91: training loss = 0.3735 ---
 92%|███████████████████████████████████████████████████████████████████████████████████████████         | 92/100 [00:32<00:02,  2.92it/s]
--- Iteration 92: training loss = 0.1026 ---
 93%|████████████████████████████████████████████████████████████████████████████████████████████        | 93/100 [00:32<00:02,  2.84it/s]
--- Iteration 93: training loss = 0.0928 ---
 94%|█████████████████████████████████████████████████████████████████████████████████████████████       | 94/100 [00:33<00:02,  2.94it/s]
--- Iteration 94: training loss = 0.0883 ---
 95%|██████████████████████████████████████████████████████████████████████████████████████████████      | 95/100 [00:33<00:01,  3.02it/s]
--- Iteration 95: training loss = 0.0860 ---
 96%|███████████████████████████████████████████████████████████████████████████████████████████████     | 96/100 [00:33<00:01,  2.92it/s]
--- Iteration 96: training loss = 0.0835 ---
 97%|████████████████████████████████████████████████████████████████████████████████████████████████    | 97/100 [00:34<00:01,  2.84it/s]
--- Iteration 97: training loss = 0.0805 ---
 98%|█████████████████████████████████████████████████████████████████████████████████████████████████   | 98/100 [00:34<00:00,  2.83it/s]
--- Iteration 98: training loss = 0.0783 ---
 99%|██████████████████████████████████████████████████████████████████████████████████████████████████  | 99/100 [00:35<00:00,  2.87it/s]
--- Iteration 99: training loss = 0.0763 ---
100%|████████████████████████████████████████████████████████████████████████████████████████████████████| 100/100 [00:35<00:00,  2.83it/s]
--- Iteration 100: training loss = 0.0741 ---

Accuracy:  0.9461827284105131
```



Step 4.4: Did finetuning work? Why did we freeze the first few layers? (3 points)

ADD YOUR RESPONSE HERE

In [ ]:
```
Yes, finetuning work. My algorithm keeps updating the weights and biases onl
classification. This way we save on the computational time without sacrifici
the same image through a model without frozen layers will result in the same
freeze it and not update the weights and biases anymore.
```