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.

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

```
!unzip "/content/drive/My Drive/CV-CW1/Fish_Dataset.zip" -d "/content"
```

```
Archive:  /content/drive/My Drive/CV-CW1/Fish_Dataset.zip
replace /content/Fish_Dataset/Black Sea Sprat/00001.png? [y]es, [n]o, [A]ll, [N]one,
```

```
# We will start by building a dataset class using the following 5 species of fishes
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):
    #### ADD YOUR CODE HERE ####
    indices = np.arange(lendata)
    np.random.shuffle(indices)
    split = int(lendata*percentage)
    idxs_train = indices[:split]
    idxs_test = indices[split:]
    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,
                 subset=1):
        # 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
        # subset: 1 -> first 5 classes, 2 -> last 4 classes

        # Load all the images and their labels & Resize all images to a standard size
        #### ADD YOUR CODE HERE ####
        training_num = len(idxs_train)
        test_num = len(idxs_test)
        total_num = training_num + test_num

        # Initialise empty picture and label arrays
        pictures = torch.empty((total_num, 3, img_size, img_size))
        labels = torch.empty((total_num,1), dtype=torch.int)

        # Select the appropriate subset of species
        if subset == 1:
          fish = ["Black Sea Sprat", "Gilt-Head Bream", "Shrimp", "Striped Red Mullet", "T
        else:
          fish = ["Hourse Mackerel", "Red Mullet", "Red Sea Bream", "Sea Bass"]

        # Hack so the corrupted image doesn't cause issues
        seenCorruptPicture = False

        # Create an image to tensor transformer
        transform = transforms.ToTensor()

        # Load all the pictures and convert to
        for j,species in enumerate(fish):
          label = Multiclass_labels_correspondances[species]
          for i in range(1,1001):
            # Open the image
            img = "/"+species+"/"+(5-len(str(i)))*"0"+str(i)+".png"
            if img == "/Red Mullet/00081.png": # This file is corrupted!
              seenCorruptPicture = True
              continue
            try:
              tmp_picture = Image.open(path_to_images+img)
            except IOError:
              print("Error opening file")

            # Create a copy and close the image
            picture = tmp_picture.copy()
            tmp_picture.close()

            # Resize the image
            picture = picture.resize((img_size,img_size))

            # Store the image and label
            if not seenCorruptPicture:
              index = j*1000+i-1
            else:
              index = j*1000+i-2
            pictures[index] = transform(picture)
            labels[index] = label

        # Extract the images and labels with the specified file indexes
        #### ADD YOUR CODE HERE ####
        if train:
          self.images = torch.empty((training_num, 3, img_size, img_size))
          self.labels = torch.empty((training_num,1), dtype=torch.int)
          indices = idxs_train
        else:
          self.images = torch.empty((test_num, 3, img_size, img_size))
          self.labels = torch.empty((test_num,1), dtype=torch.int)
          indices = idxs_test

        for i,indx in enumerate(indices):
          self.images[i] = pictures[indx]
          self.labels[i] = labels[indx]

    def __len__(self):
        # Return the number of samples
        #### ADD YOUR CODE HERE ####

        return len(self.labels)

    def __getitem__(self, idx):
        # Get an item using its index
        # Return the image and its label
        #### ADD YOUR CODE HERE ####

        return self.images[idx], self.labels[idx]
```

Step 2: Explore the data. (15 Points)

Step 2.1: Data visualisation. (5 points)

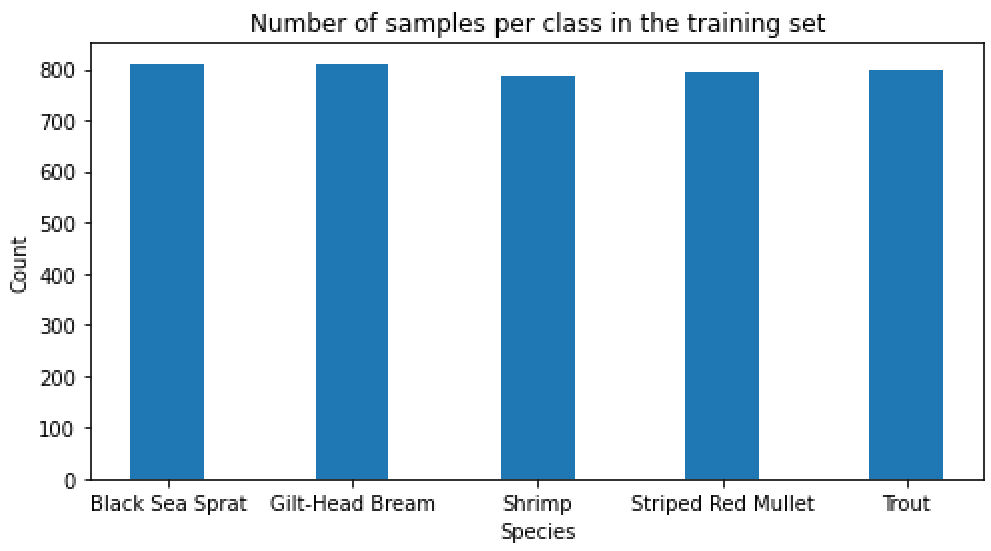
- Plot data distribution, i.e. the number of samples per class.
- Plot 1 sample from each of the five classes in the training set.

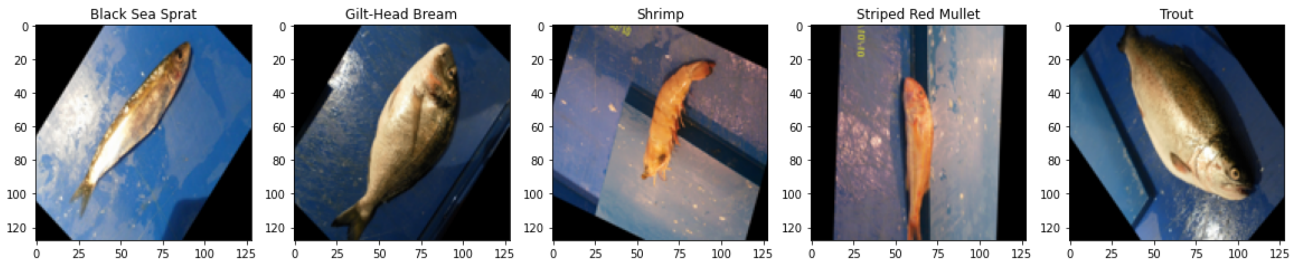
```
# Training set
img_path = '/content/Fish_Dataset'
dataset  = FishDataset(img_path, idxs_train, idxs_test, None, img_size=128, train=True, su


# Plot the number of samples per class
#### ADD YOUR CODE HERE ####
counts = [0,0,0,0,0]
fish = ["Black Sea Sprat", "Gilt-Head Bream", "Shrimp", "Striped Red Mullet", "Trout"]
for i in range(len(dataset)):
  img, label = dataset[i]
  counts[label] += 1

plt.figure(figsize=(8, 4))
plt.bar([2*i for i,_ in enumerate(counts)], counts)
plt.xlabel("Species")
plt.ylabel("Count")
plt.title("Number of samples per class in the training set")
plt.xticks([2*i for i,_ in enumerate(counts)], fish)
plt.show

# Plot 1 sample from each of the five classes in the training set
#### ADD YOUR CODE HERE ####
sample_images = []
toPIL = transforms.ToPILImage()
idx = 0
for i in range(5):
  while True:
    img, label = dataset[idx]
    idx += 1
    if label == i:
      img = toPIL(img)
      sample_images.append(img)
      break
fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(1,5, figsize=(20,4))
ax1.imshow(sample_images[0])
ax1.title.set_text("Black Sea Sprat")
ax2.imshow(sample_images[1])
ax2.title.set_text("Gilt-Head Bream")
ax3.imshow(sample_images[2])
ax3.title.set_text("Shrimp")
ax4.imshow(sample_images[3])
ax4.title.set_text("Striped Red Mullet")
ax5.imshow(sample_images[4])
ax5.title.set_text("Trout")
```





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?

The training dataset is relatively balanced as there is a similar number of each species. (Indeed in the full dataset there is the same number of each)

If the dataset weren't balanced, 3 options would be:

- Over-sampling the dataset, i.e. making copies of under-represented species
- Under-sampling the dataset, i.e reducing the number of images of an over-represented species
- Collecting more data/images for under-represented species (the hardest of the solutions, as there is probably a reason why the species is under-represented in the first place)

The dataset has already been pre-processed in a way. Looking at the images it is clear that data augmentation has been used to increase the number of samples. This has involved transforming/translating the intial images to create more.

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.

```
class Net(nn.Module):
    def __init__(self, output_dims = 1):
        super(Net, self).__init__()
        #### ADD YOUR CODE HERE ####
        self.conv1 = nn.Conv2d(3,6,3)
        self.conv2 = nn.Conv2d(6,16,3)
        self.conv3 = nn.Conv2d(16,26,3)
        self.conv4 = nn.Conv2d(26,36,3)
        self.fc1 = nn.Linear(1296,256)
        self.fc2 = nn.Linear(256,128)
        self.fc3 = nn.Linear(128,output_dims)
        self.dropout = nn.Dropout(0.25)

    def forward(self, x):
        # Forward propagation
        #### ADD YOUR CODE HERE ####
        x = F.max_pool2d(F.relu(self.conv1(x)), 2, 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2, 2)
        x = F.max_pool2d(F.relu(self.conv3(x)), 2, 2)
        x = F.max_pool2d(F.relu(self.conv4(x)), 2, 2)
        x = torch.flatten(x,1)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(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 = 'gpu'
```

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

- Loss function
- Optimizer
- Learning Rate

- Number of iterations
- Batch Size
- Other relevant hyperparameters

```
# Network
model = Net(5)

# Loss function
criterion = nn.CrossEntropyLoss()

# Optimiser and learning rate
lr = 0.1
optimizer = torch.optim.SGD(model.parameters(), lr)

# Number of iterations for training
epochs = 32

# Training batch size
train_batch_size = 64


# Based on the FishDataset, use the PyTorch DataLoader to load the data during model train
train_dataset = FishDataset(img_path, idxs_train, idxs_test, None, img_size=128, train=Tru
train_dataloader = DataLoader(train_dataset, batch_size=train_batch_size, shuffle=True)
test_dataset = FishDataset(img_path, idxs_train, idxs_test, None, img_size=128, train=Fals
test_dataloader = DataLoader(test_dataset, batch_size=train_batch_size, shuffle=True)
```

Step 3.3: Train the model. (15 points)

Complete the training loop.

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

    for imgs, labs in train_dataloader:
        # Get a batch of training data and train the model
        #### ADD YOUR CODE HERE ####
        optimizer.zero_grad()
        outputs = model(imgs)
        labs = labs.squeeze(1).long()
        loss = criterion(outputs,labs)
        loss.backward()
        optimizer.step()

        loss_curve += [loss.item()]
    print('--- Iteration {0}: training loss = {1:.4f} ---'.format(epoch + 1, np.array(loss
```

```
  3%|▎         | 1/32 [00:17<09:08, 17.69s/it]--- Iteration 1: training loss = 1.61
  6%|▋         | 2/32 [00:35<08:45, 17.52s/it]--- Iteration 2: training loss = 1.60
  9%|▉         | 3/32 [00:52<08:26, 17.45s/it]--- Iteration 3: training loss = 1.60
 12%|█▎        | 4/32 [01:09<08:08, 17.44s/it]--- Iteration 4: training loss = 1.60
 16%|█▌        | 5/32 [01:27<07:50, 17.42s/it]--- Iteration 5: training loss = 1.581
 19%|█▉        | 6/32 [01:44<07:30, 17.33s/it]--- Iteration 6: training loss = 1.567
 22%|██▏       | 7/32 [02:01<07:11, 17.25s/it]--- Iteration 7: training loss = 1.55
 25%|██▌       | 8/32 [02:18<06:49, 17.08s/it]--- Iteration 8: training loss = 1.366
 28%|██▊       | 9/32 [02:34<06:30, 16.98s/it]--- Iteration 9: training loss = 1.13
 31%|███▏      | 10/32 [02:51<06:11, 16.87s/it]--- Iteration 10: training loss = 1.
 34%|███▍      | 11/32 [03:08<05:53, 16.82s/it]--- Iteration 11: training loss = 0.
 38%|███▊      | 12/32 [03:25<05:36, 16.81s/it]--- Iteration 12: training loss = 0.
 41%|████      | 13/32 [03:41<05:18, 16.77s/it]--- Iteration 13: training loss = 0.7
 44%|████▍     | 14/32 [03:58<05:01, 16.75s/it]--- Iteration 14: training loss = 0.
 47%|████▋     | 15/32 [04:15<04:44, 16.75s/it]--- Iteration 15: training loss = 0.
 50%|█████     | 16/32 [04:31<04:27, 16.73s/it]--- Iteration 16: training loss = 0.4
 53%|█████▎    | 17/32 [04:48<04:11, 16.74s/it]--- Iteration 17: training loss = 0.
 56%|█████▋    | 18/32 [05:05<03:54, 16.74s/it]--- Iteration 18: training loss = 0.
 59%|█████▉    | 19/32 [05:22<03:37, 16.75s/it]--- Iteration 19: training loss = 0.
 62%|██████▎   | 20/32 [05:39<03:21, 16.81s/it]--- Iteration 20: training loss = 0.
 66%|██████▌   | 21/32 [05:57<03:09, 17.19s/it]--- Iteration 21: training loss = 0.1
 69%|██████▉   | 22/32 [06:14<02:50, 17.08s/it]--- Iteration 22: training loss = 0.1
 72%|███████▏  | 23/32 [06:30<02:32, 16.99s/it]--- Iteration 23: training loss = 0.1
 75%|███████▌  | 24/32 [06:47<02:15, 16.93s/it]--- Iteration 24: training loss = 0.1
 78%|███████▊  | 25/32 [07:04<01:58, 16.96s/it]--- Iteration 25: training loss = 0.1
 81%|████████▏ | 26/32 [07:21<01:41, 16.96s/it]--- Iteration 26: training loss = 0.
 84%|████████▍ | 27/32 [07:38<01:24, 16.97s/it]--- Iteration 27: training loss = 0.
 88%|████████▊ | 28/32 [07:55<01:07, 16.98s/it]--- Iteration 28: training loss = 0.
 91%|█████████ | 29/32 [08:16<00:54, 18.04s/it]--- Iteration 29: training loss = 0.1
 94%|█████████▍| 30/32 [08:33<00:35, 17.78s/it]--- Iteration 30: training loss = 0.
 97%|█████████▋| 31/32 [08:50<00:17, 17.57s/it]--- Iteration 31: training loss = 0.
100%|██████████| 32/32 [09:07<00:00, 17.11s/it]--- Iteration 32: training loss = 0.0
```

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

```
# Deploy the model
#### ADD YOUR CODE HERE ####
predictions = []
targets = []
with torch.no_grad():
  for imgs,labs in test_dataloader:
    preds = model(imgs)
    preds = torch.argmax(preds,1)
    preds = preds.detach().numpy()
    labs = labs.detach().numpy()
    for i in range(len(preds)):
      predictions.append(preds[i])
      targets.append(labs[i])
targets = list(targets)
```

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

You can use sklearns related function.

```
#### ADD YOUR CODE HERE ####
cm = np.zeros((5,5))
for i in range(len(predictions)):
  output = predictions[i]
  target = targets[i]
  cm[output,target] += 1

print(cm)
accuracy = cm.trace()/1000
print("Accuracy:",accuracy)
```

```
[[186.   0.   7.   5.   6.]
 [  0. 182.   2.   7.   6.]
 [  2.   0. 197.   2.   0.]
 [  1.   3.   7. 189.   2.]
 [  0.   6.   0.   1. 189.]]
Accuracy: 0.943
```

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.

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

LENDATA = 3999 # Not 4000 because of the corrupt image
idxs_train,idxs_test = split_train_test(LENDATA, 0.8)

# Training batch size - MOVED FROM 4.2
train_batch_size = 64

# Dataloaders
#### ADD YOUR CODE HERE ####
```

```
train_dataset = FishDataset(img_path, idxs_train, idxs_test, None, img_size=128, train=Tru
train_dataloader = DataLoader(train_dataset, batch_size=train_batch_size, shuffle=True)
test_dataset = FishDataset(img_path, idxs_train, idxs_test, None, img_size=128, train=Fals
test_dataloader = DataLoader(test_dataset, batch_size=train_batch_size, shuffle=True)
```

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`.

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

freeze_till_last(model)
# Modify the last layer. This layer is not freezed.
#### ADD YOUR CODE HERE ####
model.fc3 = nn.Linear(128,4)

# Loss function
criterion = nn.CrossEntropyLoss()

# Optimiser and learning rate
lr = 0.1
optimizer = torch.optim.SGD(model.parameters(), lr)

# Number of iterations for training
epochs = 8
```

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

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

    for imgs, labs in train_dataloader:
        # Get a batch of training data and train the model
        #### ADD YOUR CODE HERE ####
        optimizer.zero_grad()
        outputs = model(imgs)
        labs = labs.squeeze(1).long()
        loss = criterion(outputs,labs)
        loss.backward()
        optimizer.step()

        loss_curve += [loss.item()]
    print('--- Iteration {0}: training loss = {1:.4f} ---'.format(epoch + 1, np.array(loss

# Deploy the model on the test set
# Deploy the model on the test set
#### ADD YOUR CODE HERE ####
predictions = []
targets = []
with torch.no_grad():
  for imgs,labs in test_dataloader:
    preds = model(imgs)
    preds = torch.argmax(preds,1)
    preds = preds.detach().numpy()
    labs = labs.detach().numpy()
    for i in range(len(preds)):
      predictions.append(preds[i])
      targets.append(labs[i])
targets = list(targets)

# Evaluate the performance
#### ADD YOUR CODE HERE ####
cm = np.zeros((4,4))
for i in range(len(predictions)):
  output = predictions[i]
  target = targets[i]
  cm[output,target] += 1

print(cm)
accuracy = cm.trace()/800
print("Accuracy:",accuracy)
```

```
 12%|█          | 1/8 [00:06<00:47,  6.79s/it]--- Iteration 1: training loss = 1.105
 25%|██         | 2/8 [00:13<00:40,  6.77s/it]--- Iteration 2: training loss = 0.9952
 38%|███▊       | 3/8 [00:20<00:33,  6.74s/it]--- Iteration 3: training loss = 0.958
 50%|█████      | 4/8 [00:26<00:26,  6.74s/it]--- Iteration 4: training loss = 0.9234
 62%|██████▎    | 5/8 [00:33<00:20,  6.74s/it]--- Iteration 5: training loss = 0.924
 75%|███████▌   | 6/8 [00:40<00:13,  6.74s/it]--- Iteration 6: training loss = 0.9174
 88%|████████▊  | 7/8 [00:47<00:06,  6.73s/it]--- Iteration 7: training loss = 0.897
100%|██████████| 8/8 [00:53<00:00,  6.74s/it]--- Iteration 8: training loss = 0.9052

[[144.  28.  17.  39.]
 [ 22. 114.  22.  39.]
 [ 20.  20. 140.  18.]
 [ 37.  35.  12.  93.]]
Accuracy: 0.61375
```

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

The purpose of this final task was to freeze all but the last layer of the trained model. The idea is that the earlier layers learn the different features of the fish, and the final layer selects the fish based on those features. As the types of features should be somewhat consistent between species, the earlier layers can be frozen. The final layer is then trained to learn which features are associated with each of the new species. However, the accuracy achieved is much lower than previously.

✓ 55s completed at 8:21 PM ● ×