Seafood images classification

Author: Mislav Spajić

https://github.com/MySlav/seafood-classification-cnn

Contents

1. Problem description

I have chosen to deal with image classification and will try to solve this problem of Seafood images classification utilizing a Convolutional Neural Network (CNN). I will utilize transfer learning method, I will load some of the pretrained models like “DenseNet”, “MobileNetV2” and “ResNet” and change the final layer. The parameters for the rest of the network will be frozen. I have chosen a dataset which contains images of 9 different seafood types.

This dataset contains 9 different seafood types collected from a supermarket in Izmir, Turkey for a university-industry collaboration project at Izmir University of Economics, and this work was published in ASYU 2020. The dataset includes gilt head bream, red sea bream, sea bass, red mullet, horse mackerel, black sea sprat, striped red mullet, trout, shrimp image samples.

Images were collected via 2 different cameras, Kodak Easyshare Z650 and Samsung ST60. Therefore, the resolution of the images are: 2832 x 2128, 1024 x 768, respectively.

Although authors provide a set in which they have done resizing of pictures to a fixed size and they have generated more pictures using data augmentation techniques, I will use the original set and try to do these things myself.

2. Methodology

2.1. Importing data

Since I worked on a kaggle kernel, importing data was quite easy with the help of python standard “os” library.

```
# Defining directory with data
Dir = '/kaggle/input/a-large-scale-fish-dataset/NA_Fish_Dataset'
os.listdir(Dir)
```

```
# Creating a dataframe of paths and labels
data = {"path": [],
        "label": []
        }

for i in os.listdir(Dir):
  path = os.path.join(Dir, i)
  for j in os.listdir(path):
    data['path'].append(os.path.join(path, j))
    data['label'].append(i)

data_df=pd.DataFrame.from_dict(data)
```

2.2. Exploratory data analysis

Basic EDA was done using some basic “pandas” functions.

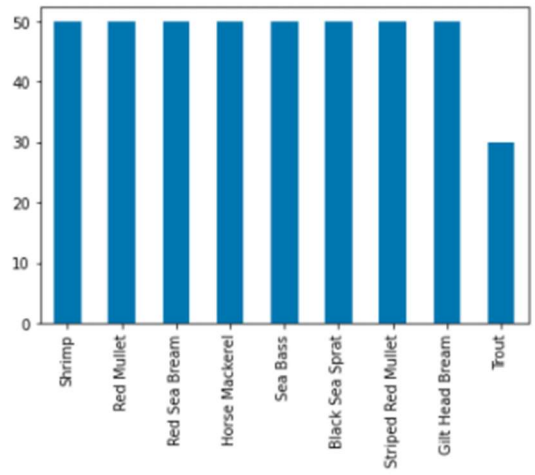
```
data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 430 entries, 0 to 429
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   path    430 non-null    object
 1   label   430 non-null    object
dtypes: object(2)
memory usage: 6.8+ KB
```

```
data_df["label"].value_counts()
```

```
Shrimp               50
Red Mullet           50
Red Sea Bream        50
Horse Mackerel       50
Sea Bass             50
Black Sea Sprat      50
Striped Red Mullet   50
Gilt Head Bream      50
Trout                30
Name: label, dtype: int64
```

```
data_df["label"].value_counts().plot.bar()
plt.show()
```



We can see that dataset is very balanced, all the classes except “Trout” have equal number of pictures (50). I won't have a problem with class imbalance and accuracy can be used as a measure of performance.

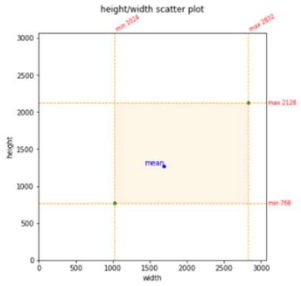
```
# Finding all the unique formats and dimensions of images in the dataset
dimensions=[]
formats=[]

for row in data_df.iterrows():
    img = PIL.Image.open(row[1]["path"])

    dimensions.append(np.asarray(img).shape)
    formats.append(img.format)

print("Dimensions of the pictures in the dataset are:", set(dimensions))
print("Formats of the pictures in the dataset are:", set(formats))
```

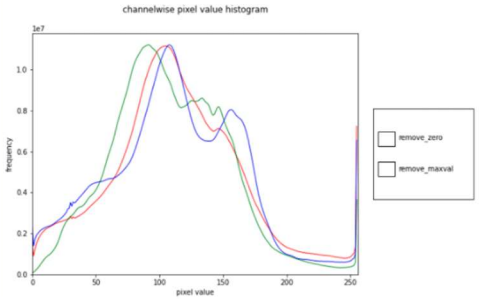
```
Dimensions of the pictures in the dataset are: {(2128, 2832, 3), (768, 1024, 3)}
Formats of the pictures in the dataset are: {'JPEG', 'PNG', 'MPO'}
```

We can see that all pictures have 3 channels but they have different dimensions, we will deal with that later on when we initialize dataloaders.

I have also done some more image EDA, using neat library called “basic-image-eda”.

```
found 430 images.

  0%|          | 0/430 [00:00<?, ?it/s]

Using 2 threads. (max:2)


100%|          | 430/430 [01:25<00:00,  5.02it/s]

*-------------------------------------------------------------------------------*
number of images                    |  430

dtype                               |  uint8
channels                            |  [3]
extensions                          |  ['jpeg', 'png', 'JPG']

min height                          |  768
max height                          |  2128
mean height                         |  1267.7209302325582
median height                       |  768

min width                           |  1024
max width                           |  2832
mean width                          |  1688.3348837209303
median width                        |  1024

mean height/width ratio             |  0.7508705426014899
median height/width ratio           |  0.75
recommended input size(by mean)     |  [1264 1688] (h x w, multiples of 8)
recommended input size(by mean)     |  [1264 1696] (h x w, multiples of 16)
recommended input size(by mean)     |  [1280 1696] (h x w, multiples of 32)

channel mean(0~1)                   |  [0.4109598  0.40690383 0.4242128 ]
channel std(0~1)                    |  [0.21329764 0.18420503 0.20046176]
*-------------------------------------------------------------------------------*
eda ended in 00 hours 01 minutes 25 seconds
```





2.3. Data preparation

Using „sckit-learn“ label encoder I encoded class names to integers.

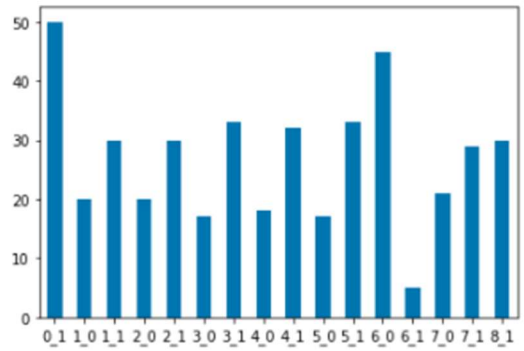
```
# Encoding labels to integers
le = LabelEncoder()
data_df['label'] = le.fit_transform(data_df['label'])
```

```
# We can pull class names with this
le.classes_
```

```
array(['Black Sea Sprat', 'Gilt Head Bream', 'Horse Mackerel',
       'Red Mullet', 'Red Sea Bream', 'Sea Bass', 'Shrimp',
       'Striped Red Mullet', 'Trout'], dtype=object)
```

I also added camera column which represents with which camera was the picture taken. I will encode that column also and use it to create another one called “label_camera” which represents combination of class and camera, that column will be used to stratify the split. More info in the notebook.

```
# Distribution of combinations of camera and class
data_df["label_camera"].value_counts().sort_index().plot.bar(rot=0)
plt.show()
```

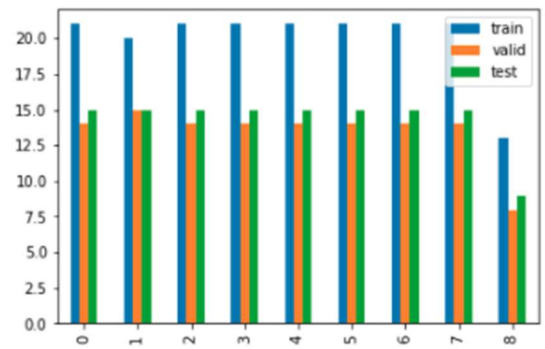


Train – valid -test split was done using “train_test_split” from “sklearn” library.

```
# Splitting train - test, 70-30 %, using stratified split by label_camera combination
train,  test = train_test_split(data_df, test_size = 0.3, random_state = 42, stratify=data_df
["label_camera"])

# Had trouble implementing k-fold CV with pytorch, so I decided to go with regular train-valid-te
st split
train,  valid = train_test_split(train, test_size = 0.4, random_state = 42, stratify=train["lab
el_camera"])

print(f"train set shape: {train.shape}")
print(f"test set shape: {test.shape}")
print(f"valid set shape: {valid.shape}")

# Final ratios of sets are train (42 %) - valid (28 %) - test (30 %)
# Decided to go this route since I deal with low number of rows, and I will augment the test data
several times
```

```
train set shape: (180, 4)
test set shape: (129, 4)
valid set shape: (121, 4)
```

Class distribution after split



Data will be augmented before training and stored in RAM, later on I will push it to GPU memory during training. It was implemented using “albumentations” library. For each picture in the train set I generated 6 augmented ones which resulted in 140 images in each class. Augmented train data was therefore consisted of 1260 pictures in total.

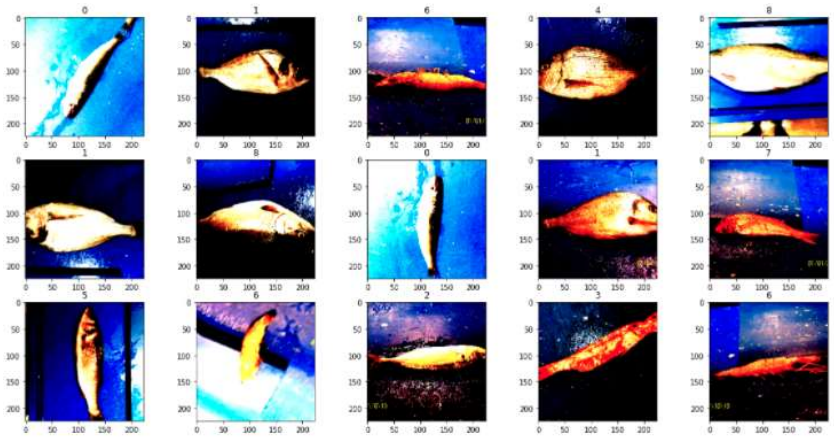
```
# Defining augmentations
"""All pre-trained models expect input images normalized in the same way, i.e. mini-batches of 3-
channel RGB images of shape (3 x H x W), where H and W are expected to be at least 224.
The images have to be loaded in to a range of [0, 1] and then normalized using mean = [0.485, 0.4
56, 0.406] and std = [0.229, 0.224, 0.225]."""

albumentations_transform = albumentations.Compose([
    albumentations.SmallestMaxSize(224),
    albumentations.CenterCrop(224, 224),
    albumentations.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))
    ])

albumentations_transform2 = albumentations.Compose([
    albumentations.SmallestMaxSize(224),
    albumentations.CenterCrop(224, 224),
    albumentations.Rotate(p=0.3),
    albumentations.RandomBrightnessContrast(brightness_limit=0.3, contrast_limit=0.3, p=1.0),
    albumentations.OneOf([
                          albumentations.RandomRotate90(p=1),
                          albumentations.HorizontalFlip(p=1),
                          albumentations.VerticalFlip(p=1)
    ], p=1),
    albumentations.OneOf([
                          albumentations.MotionBlur(p=1),
                          albumentations.OpticalDistortion(p=1),
                          albumentations.GaussNoise(p=1)
    ], p=1),
    albumentations.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))
])
```

Split ratios in practice were changed now, and they are shown below.

```
# Split ratios now in practice
print("Training: {:.2f}".format(len(augmented_train)/(len(augmented_train)+len(test)+len(vali
d))))
print("Validation: {:.2f}".format(len(valid)/(len(augmented_train)+len(test)+len(valid))))
print("Test: {:.2f}".format(len(test)/(len(augmented_train)+len(test)+len(valid))))
```

```
Training: 0.83
Validation: 0.08
Test: 0.09
```

Augmented images can be seen below.



Next step was defining pytorch dataset and dataloader classes.

```
# Defining dataset classes
class MyTrainDataset(torch.utils.data.Dataset):
    def __init__(self, dataframe,transform=None):
        self.dataframe = dataframe
        self.transform = transform

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, index):
        row = self.dataframe.iloc[index]
        image=torchvision.transforms.functional.to_tensor(row["image"])

        return (
            image,
            row["label"]
        )


class MyTestDataset(torch.utils.data.Dataset):
    def __init__(self, dataframe,transform=None):
        self.dataframe = dataframe
        self.transform = transform

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, index):
        row = self.dataframe.iloc[index]
        image=np.asarray(PIL.Image.open(row["path"]))

        if self.transform:
            image = self.transform(image=image)["image"]
            image = torchvision.transforms.functional.to_tensor(image)
        return (
            image,
            row["label"]
        )
```

```
# Initializing datasets
dataset_train = MyTrainDataset(augmented_train)
dataset_valid = MyTestDataset(valid,transform=albumentations_transform)
dataset_test = MyTestDataset(test,transform=albumentations_transform)
```

```
# Initializing dataloaders
trainloader = torch.utils.data.DataLoader(dataset_train, batch_size=16, shuffle=True)
validloader = torch.utils.data.DataLoader(dataset_valid, batch_size=16)
testloader = torch.utils.data.DataLoader(dataset_test, batch_size=16)
```

I have used a smaller batch size of 16 but to small size of dataset and so that models generalize better.

3. Experiments and evaluation

I decided to utilize transfer learning, I setteled on these 3 pretrained models: “DenseNet161”, “ MobileNet v2” and “ResNet152”. For all 3 of them I changed the last layer and added the same same Sequential container instead of their default one. Structure of the layer can be seen below in the code.

```
# https://pytorch.org/vision/stable/models.html
densenet=models.densenet161(pretrained=True)
mobilenet = models.mobilenet_v2(pretrained=True)
resnet = models.resnet152(pretrained=True)
all_models = [densenet,mobilenet,resnet]
```

```
# Modifying last layers, addign the same Seqential container as last element to all 3

for model in all_models:
    classifier_name, old_classifier = model._modules.popitem()
    # Needed because resnet  has no "classifier" as name of the last layer but "fc" and to store
input size

    # Freeze parameters so we don't backprop through them - code taken from HW
    for param in model.parameters():
        param.requires_grad = False

    # Some of the models have a seqential container as last element
    if type(old_classifier) == torch.nn.modules.container.Sequential:
        classifier_input_size = old_classifier[len(old_classifier)-1].in_features
    else:
        classifier_input_size = old_classifier.in_features
    classifier = torch.nn.Sequential(OrderedDict([
                            ('fc1', torch.nn.Linear(classifier_input_size, 512)),
                            ('relu', torch.nn.ReLU()),
                            ('dropout', torch.nn.Dropout(p=0.5)),
                            ('fc2', torch.nn.Linear(512, 9)),
                            ('output', torch.nn.LogSoftmax(dim=1))
                            ]))
    model.add_module(classifier_name, classifier)


    model.to(device)
```

The same loss criterion, optimizer and learning rate was used for all 3 models.

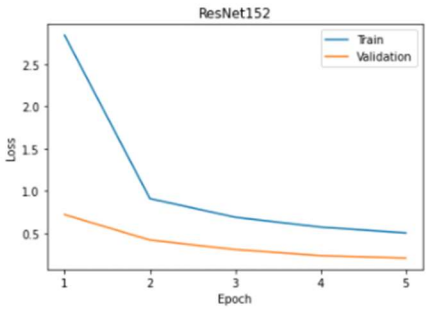
```
# Using the same loss criterion and optimizer for all of the models, also the same learning rate
and number of epochs will be used

criterion = torch.nn.NLLLoss()
optimizer_densenet = torch.optim.Adagrad(densenet.classifier.parameters(), lr=0.01)
optimizer_mobilenet = torch.optim.Adagrad(mobilenet.classifier.parameters(), lr=0.01)
optimizer_resnet = torch.optim.Adagrad(resnet.fc.parameters(), lr=0.01)
```

Training was done using a helper function which is code modified from one of the homeworks, all 3 models were trained over just 5 epochs not to overfit due to small size of dataset.

```
# ResNet 152 Training
train_losses,valid_losses = training(resnet, optimizer_resnet, criterion, trainloader, validloa
der)
```

```
Epoch: 1/5..  Training Loss: 2.840..  Validation Loss: 0.720..  Validation Accuracy: 0.938
Epoch: 2/5..  Training Loss: 0.910..  Validation Loss: 0.420..  Validation Accuracy: 0.938
Epoch: 3/5..  Training Loss: 0.690..  Validation Loss: 0.308..  Validation Accuracy: 0.953
Epoch: 4/5..  Training Loss: 0.573..  Validation Loss: 0.236..  Validation Accuracy: 0.992
Epoch: 5/5..  Training Loss: 0.504..  Validation Loss: 0.208..  Validation Accuracy: 0.977
```

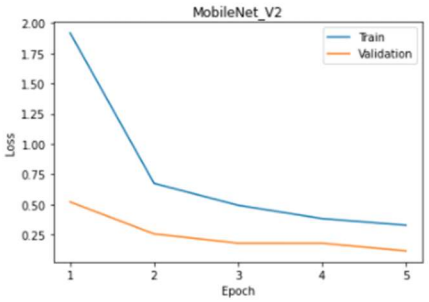
```
plt.figure()
plt.title("ResNet152")
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.plot(train_losses)
plt.plot(valid_losses)
plt.xticks(np.arange(len(train_losses)), np.arange(1, len(train_losses)+1))
plt.legend(['Train','Validation'], loc = 'upper right')
plt.show()
```



```
# MobileNet_V2
train_losses,valid_losses = training(mobilenet, optimizer_mobilenet, criterion, trainloader, va
lidloader)
```

```
Epoch: 1/5..  Training Loss: 1.919..  Validation Loss: 0.520..  Validation Accuracy: 0.898
Epoch: 2/5..  Training Loss: 0.674..  Validation Loss: 0.256..  Validation Accuracy: 0.984
Epoch: 3/5..  Training Loss: 0.493..  Validation Loss: 0.179..  Validation Accuracy: 0.977
Epoch: 4/5..  Training Loss: 0.382..  Validation Loss: 0.178..  Validation Accuracy: 0.945
Epoch: 5/5..  Training Loss: 0.327..  Validation Loss: 0.115..  Validation Accuracy: 0.984
```

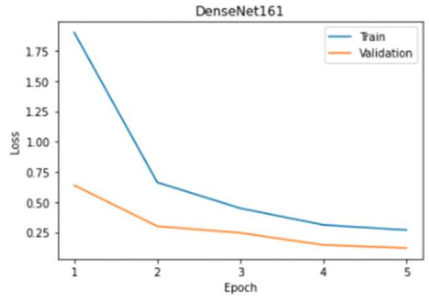
```
plt.figure()
plt.title("MobileNet_V2")
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.plot(train_losses)
plt.plot(valid_losses)
plt.xticks(np.arange(len(train_losses)), np.arange(1, len(train_losses)+1))
plt.legend(['Train','Validation'], loc = 'upper right')
plt.show()
```




```
# DenseNet161
train_losses,valid_losses = training(densenet, optimizer_densenet, criterion, trainloader, vali
dloader)
```

```
Epoch: 1/5..  Training Loss: 1.901..  Validation Loss: 0.639..  Validation Accuracy: 0.836
Epoch: 2/5..  Training Loss: 0.663..  Validation Loss: 0.300..  Validation Accuracy: 0.961
Epoch: 3/5..  Training Loss: 0.449..  Validation Loss: 0.245..  Validation Accuracy: 0.945
Epoch: 4/5..  Training Loss: 0.311..  Validation Loss: 0.145..  Validation Accuracy: 0.984
Epoch: 5/5..  Training Loss: 0.269..  Validation Loss: 0.121..  Validation Accuracy: 0.984
```

```
plt.figure()
plt.title("DenseNet161")
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.plot(train_losses)
plt.plot(valid_losses)
plt.xticks(np.arange(len(train_losses)), np.arange(1, len(train_losses)+1))
plt.legend(['Train','Validation'], loc = 'upper right')
plt.show()
```



Since MobileNet_V2 based model has shown to be the best on validation set, predictions and evaluation on the test set was done with it.

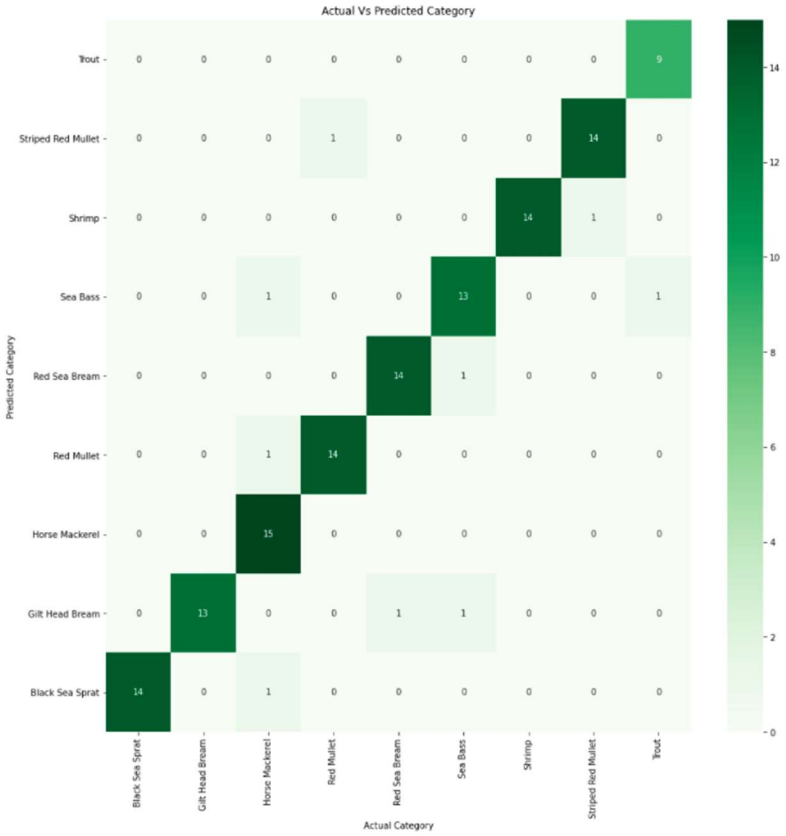
```
# Predictions - on mobilenet
# https://www.kaggle.com/abubakaryagob/fish-classification-with-pytorch-resnet - Modified code fr
om this notebook
y_pred_list = []
y_true_list = []
with torch.no_grad():
    for inputs, labels in testloader:
            inputs, labels = inputs.to(device), labels.to(device)
            y_test_pred = mobilenet(inputs)
            _, y_pred_tag = torch.max(y_test_pred, dim = 1)
            y_pred_list.append(y_pred_tag.cpu().numpy())
            y_true_list.append(labels.cpu().numpy())

flat_pred = []
flat_true = []
for i in range(len(y_pred_list)):
    for j in range(len(y_pred_list[i])):
        flat_pred.append(y_pred_list[i][j])
        flat_true.append(y_true_list[i][j])
```

```
# Evaluation of predictions on test set

print(f"number of testing samples results: {len(flat_pred)}")
print(f"Testing accuracy is: {accuracy_score(flat_true, flat_pred) * 100:.2f}%")
```

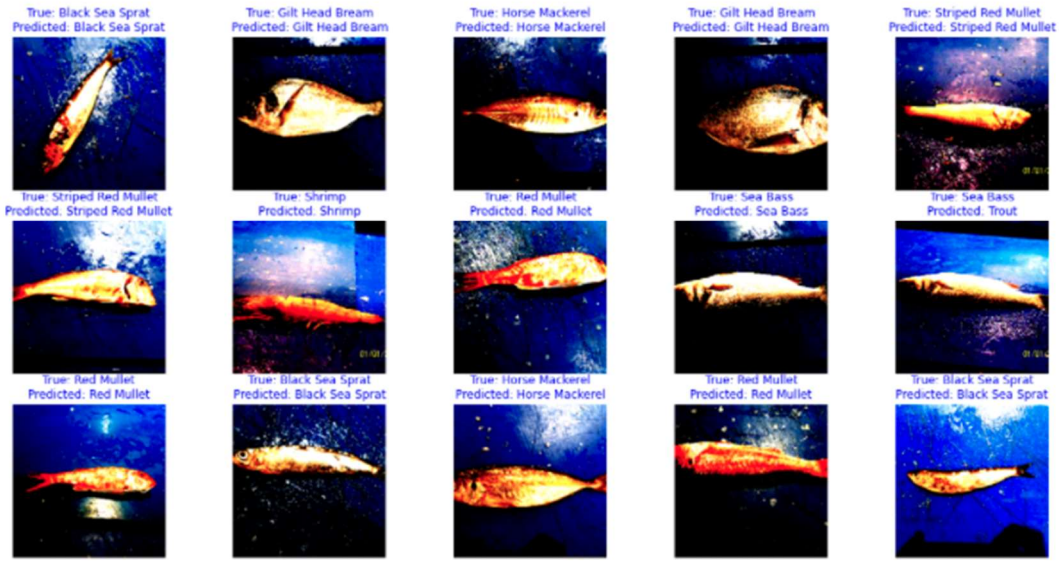
```
number of testing samples results: 129
Testing accuracy is: 93.02%
```

```
# Classification report
print(classification_report(flat_true, flat_pred, target_names=le.classes_))
```

```
                  precision    recall  f1-score   support

   Black Sea Sprat       1.00      0.93      0.97        15
   Gilt Head Bream       1.00      0.87      0.93        15
    Horse Mackerel       0.83      1.00      0.91        15
        Red Mullet       0.93      0.93      0.93        15
     Red Sea Bream       0.93      0.93      0.93        15
         Sea Bass        0.87      0.87      0.87        15
           Shrimp        1.00      0.93      0.97        15
Striped Red Mullet       0.93      0.93      0.93        15
            Trout        0.90      1.00      0.95         9

         accuracy                           0.93       129
        macro avg       0.93      0.93      0.93       129
     weighted avg       0.93      0.93      0.93       129
```




```
# Displays 15 random picture of the train dataset with their respective real and predicted labels

inds = np.random.randint(len(dataset_test), size=15)
j=1
plt.figure(figsize=(20,10))
for i in inds:
    img, label = dataset_test[i]
    plt.subplot(3, 5, j)
    plt.gca().set_title(f"True: {le.inverse_transform([label])[0]}\nPredicted: {le.inverse_tran
sform([flat_pred[i]])[0]}",color='blue')
    plt.imshow(img.permute(1, 2, 0))
    plt.axis('off')
    j+=1

plt.show()
```



4. Conclusion

By utilizing transfer learning I have easily gotten a Convolutional Neural Network with high accuracy for my seafood classification problem. I have downloaded 3 pretrained models from pytorch repository and changed only the last layer. Layer of same structure has been added to all 3. Weights of rest of the layers of the models were frozen.

MobileNet_V2 based model has shown to be the best on validation set and after testing it on test set, it has shown accuracy of 93 %, which is a great result given the size of the dataset and small number of optimizations I have done to the pretrained models.

5. References

O. Ulucan, D. Karakaya and M. Turkan, "A Large-Scale Dataset for Fish Segmentation and Classification," 2020 Innovations in Intelligent Systems and Applications Conference (ASYU), 2020, pp. 1-5, doi: 10.1109/ASYU50717.2020.9259867.

https://pypi.org/project/basic-image-eda/

https://pytorch.org/tutorials/beginner/data_loading_tutorial.html

https://pytorch.org/vision/stable/models.html

https://www.kaggle.com/abubakaryagob/fish-classification-with-pytorch-resnet