Capstone 3 – Image Classification: Fish

A. <u>Objective:</u>

To develop a generalized, predictive model that can identify and differentiate between different types of fish species that share similar characteristics. The ideal model has a myriad of potential use cases that include the following:

1. High volume fish processing plant: ability to dynamically identify and separate different (in tandem with robotics) fish species to optimize the handling and processing of fish
2. Use for identification of fish species for environmental conservation efforts: this would require augmentation of the existing model to account for different environmental contexts
3. Potentially leverage existing model to develop a more sophisticated model that can accurately measure the dimensions (length and width) of the fish as well as the weight

---

B. <u>Assumptions and Exclusions:</u>

B1. Key Assumptions/Comments →

- The context (background noise) has been limited; as one randomly investigates a pool of images, you will notice that the background is relatively consistent; this can most certainly be a problem when attempting to apply this model to a larger population of pictures
- The orientation and general size of the fish varies significantly which supports a more robust model
- No missing values

B2. Key Exclusions →

- N/A

C. <u>Data Wrangling, EDA and Preprocessing:</u>

C1. Data Collection, Wrangling and EDA →

- Downloaded the dataset from Kaggle → citation is denoted below
  - title={A Large-Scale Dataset for Fish Segmentation and Classification}
  - author={Ulucan, Oguzhan and Karakaya, Diclehan and Turkan, Mehmet}
  - booktitle={2020 Innovations in Intelligent Systems and Applications Conference (ASYU)}
  - pages={1--5}
  - year={2020}
  - organization={IEEE}

```
# returning a list of all sub folder names within the directory; these will be the label names
labels = [os.path.basename(x) for x in glob.glob(abs_path + '/*', recursive=True)]

# alternatively, could use os.listdir(abs_path), but that also returns a jupyter notebook checkpoint
labels
```

```
['Sea Bass',
 'Red Mullet',
 'Gilt-Head Bream',
 'Red Sea Bream',
 'Shrimp',
 'Black Sea Sprat',
 'Striped Red Mullet',
 'Hourse Mackerel',
 'Trout']
```

Fig. 1 – Species Included in the Model

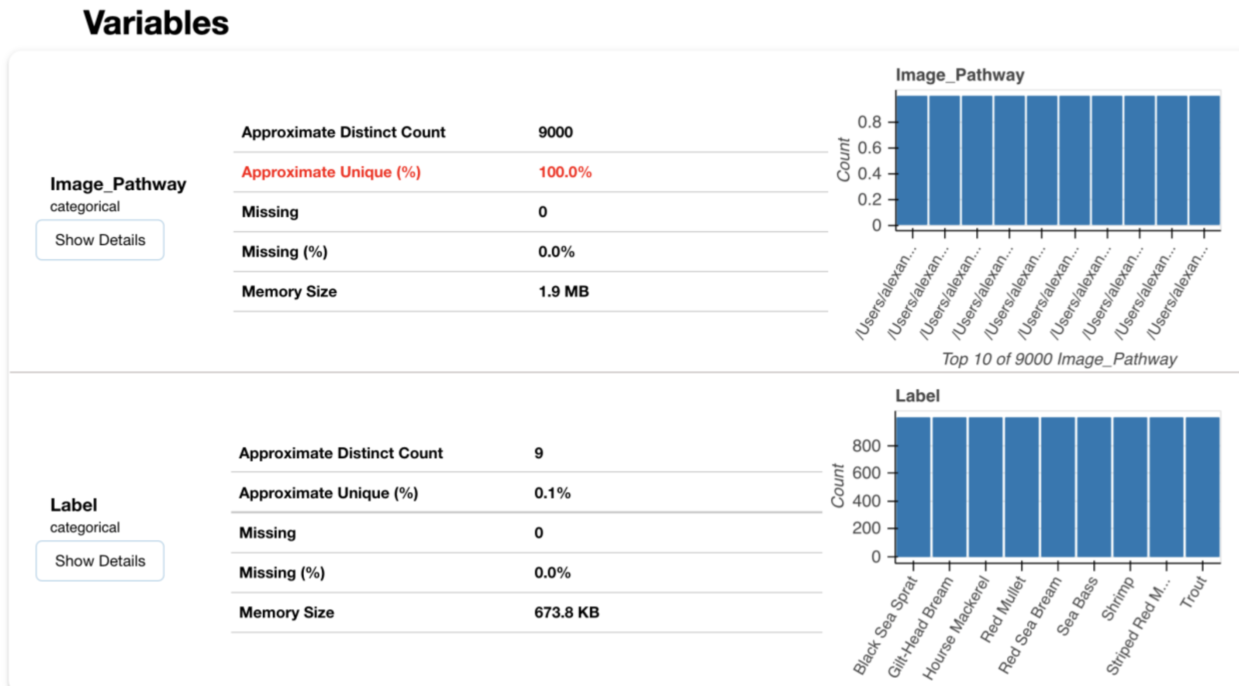

Fig. 2 – Distribution of Images (1000 per Species)

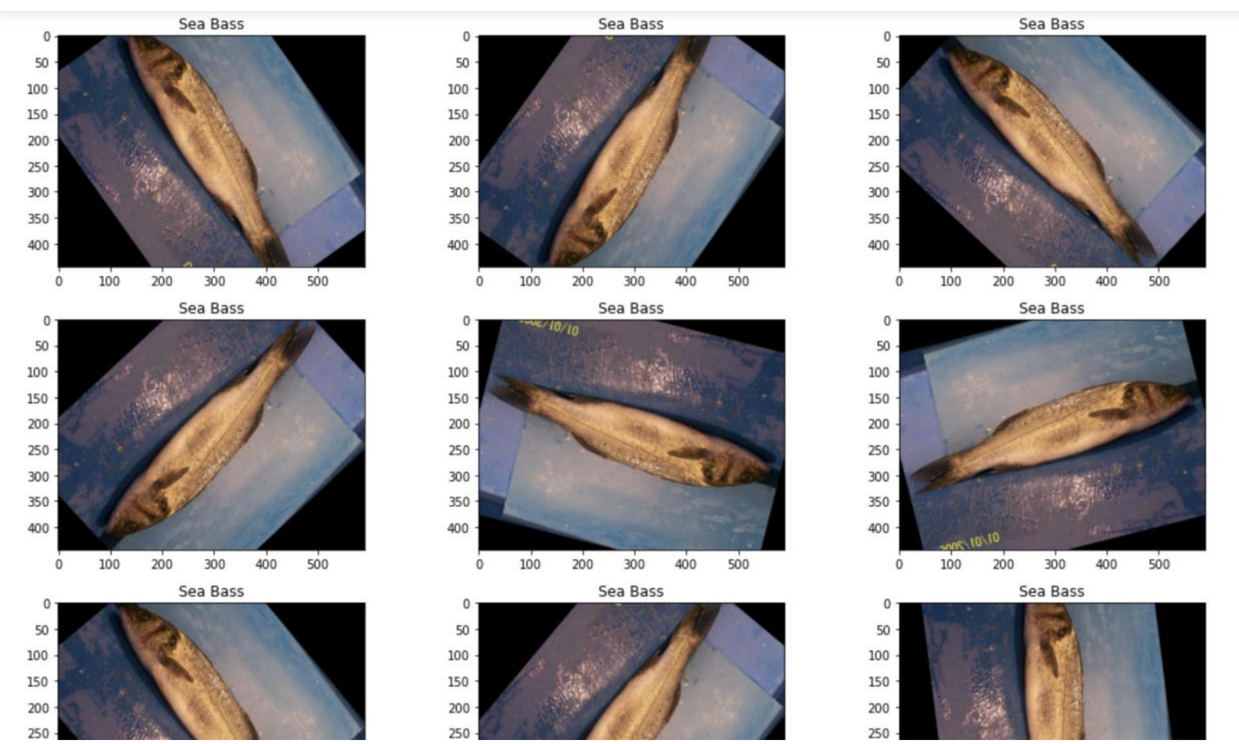

Fig. 3 – Inspection of Sea Bass Images

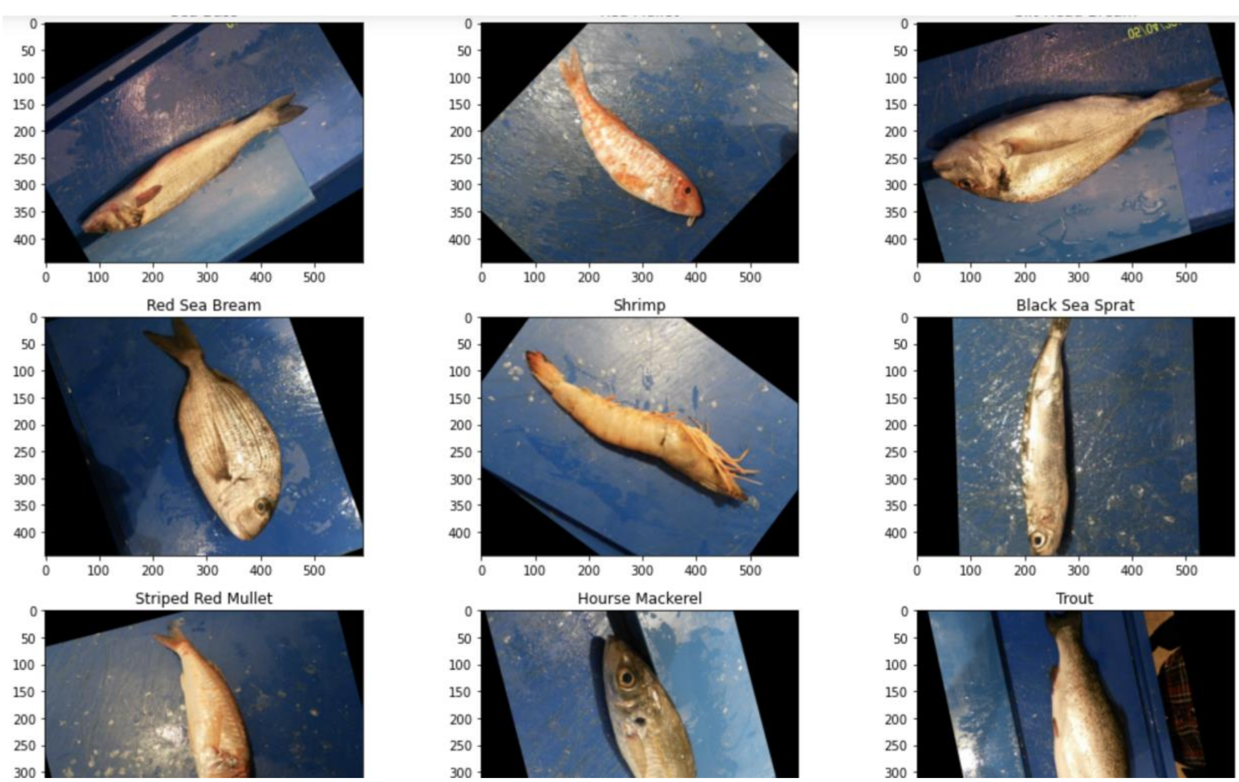

Fig. 4 – One Image of Each Fish Species

C2. Preprocessing →

- After loading in the images and mapping their associated labels and inspecting a small sample, it was time to go ahead and start splitting the data into a training set, validation set and testing set
- First, I started by doing a 70/30% split of training/testing

```
# time for preprocessing
# creating a train and testing split
from sklearn.model_selection import train_test_split

train_df, test_df = train_test_split(image_df, train_size = 0.7, shuffle = True, random_state = 49)
print(train_df.shape, test_df.shape)
```

```
(6300, 2) (2700, 2)
```

Fig. 5 – Splitting into Training and Testing DataFrames

- Next, I went ahead and built out the required generators for processing the images

```
# time to load the images and utilize the ImageDataGenerator

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_generator = ImageDataGenerator(validation_split = 0.25)
test_generator = ImageDataGenerator()
```

```
# creating a training set, validation of the training set and test set for evaluation

training_images = train_generator.flow_from_dataframe(dataframe = train_df, x_col = "Image_Pathway", y_col = "Label"
                , class_mode = 'categorical', target_size = (256, 256), color_mode = "rgb", batch_size = 50, shuffle =
                , seed = 45, subset = "training")

train_val_images = train_generator.flow_from_dataframe(dataframe = train_df, x_col = "Image_Pathway", y_col = "Label"
                , class_mode = 'categorical', target_size = (256, 256), color_mode = "rgb", batch_size = 50, shuffle =
                , seed = 45, subset = "validation")

test_images = test_generator.flow_from_dataframe(dataframe = test_df, x_col = "Image_Pathway", y_col = "Label"
                , class_mode = 'categorical', target_size = (256, 256), color_mode = "rgb", batch_size = 50, shuffle =
                , seed = 45)
```

```
Found 4725 validated image filenames belonging to 9 classes.
Found 1575 validated image filenames belonging to 9 classes.
Found 2700 validated image filenames belonging to 9 classes.
```

```
# checking out the classes to ensure everything is consistent betwe
training_images.class_indices
```

```
{'Black Sea Sprat': 0,
 'Gilt-Head Bream': 1,
 'Hourse Mackerel': 2,
 'Red Mullet': 3,
 'Red Sea Bream': 4,
 'Sea Bass': 5,
 'Shrimp': 6,
 'Striped Red Mullet': 7,
 'Trout': 8}
```

```
train_val_images.class_indices
```

```
{'Black Sea Sprat': 0,
 'Gilt-Head Bream': 1,
 'Hourse Mackerel': 2,
 'Red Mullet': 3,
 'Red Sea Bream': 4,
 'Sea Bass': 5,
 'Shrimp': 6,
 'Striped Red Mullet': 7,
 'Trout': 8}
```

```
test_images.class_indices
```

```
{'Black Sea Sprat': 0,
 'Gilt-Head Bream': 1,
 'Hourse Mackerel': 2,
 'Red Mullet': 3,
 'Red Sea Bream': 4,
 'Sea Bass': 5,
 'Shrimp': 6,
 'Striped Red Mullet': 7,
 'Trout': 8}
```

Fig. 6 – Validating the splitting and classes

- You can see an example of the augmentation below

```python
# visualizing the augmentation of the training dataset
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 5, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip( images_arr, axes):
        ax.imshow(img.astype('uint8'))
    plt.tight_layout()
    plt.show()


augmented_images = [training_images[0][0][i+5] for i in range(5)]
plotImages(augmented_images)
```

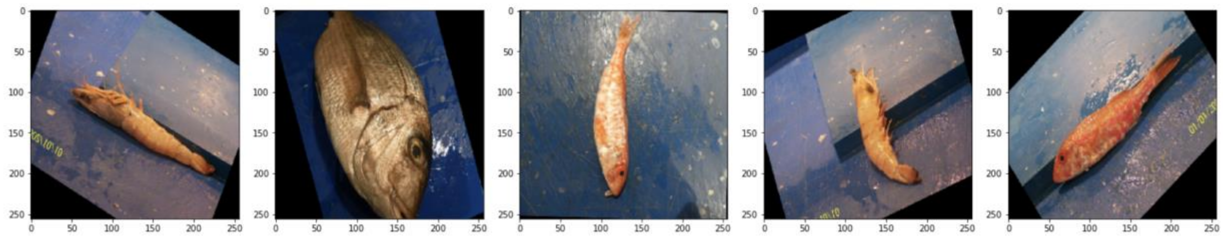

Fig. 7 - Preprocessing Augmentation

- The dataset is now ready for modelling; I elected to build a model without leveraging a pre-trained model. I only did this to get a better understanding of how to compile/build and apply the model

---

D. Modelling:

- The dataset is now ready for modelling; I elected to build a model without leveraging a pre-trained model. I only did this to get a better understanding of how to compile/build and apply the model

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 254, 254, 64)      1792
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 127, 127, 64)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 125, 125, 128)     73856
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 62, 62, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 60, 60, 256)       295168
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 256)       0
_________________________________________________________________
flatten (Flatten)            (None, 230400)            0
_________________________________________________________________
dense (Dense)                (None, 128)               29491328
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 9)                 1161
=================================================================
Total params: 29,879,817
Trainable params: 29,879,817
Non-trainable params: 0
_________________________________________________________________
```

Fig. 8 – Model Setup Params

```
# training the model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

model.compile(optimizer = "adam", loss = "categorical_crossentropy", metrics = ["accuracy"])

history = model.fit(training_images,
    validation_data=train_val_images,
    steps_per_epoch = 20,
    epochs = 25,
    callbacks=[EarlyStopping(monitor='val_loss', patience = 10, restore_best_weights = True)])
```

```
Epoch 1/25
20/20 [==============================] - 223s 11s/step - loss: 59.8392 - accuracy: 0.1949 - val_loss: 1.9908 - val_ac
curacy: 0.3454
Epoch 2/25
20/20 [==============================] - 258s 13s/step - loss: 1.9293 - accuracy: 0.3250 - val_loss: 1.7102 - val_acc
uracy: 0.4222
Epoch 3/25
20/20 [==============================] - 223s 11s/step - loss: 1.7243 - accuracy: 0.3810 - val_loss: 1.4344 - val_acc
uracy: 0.5003
Epoch 4/25
20/20 [==============================] - 203s 10s/step - loss: 1.5356 - accuracy: 0.4410 - val_loss: 1.1922 - val_acc
uracy: 0.6076
Epoch 5/25
20/20 [==============================] - 203s 10s/step - loss: 1.3530 - accuracy: 0.5140 - val_loss: 0.9609 - val_acc
uracy: 0.6870
Epoch 6/25
20/20 [==============================] - 205s 10s/step - loss: 1.1525 - accuracy: 0.5770 - val_loss: 0.8586 - val_acc
uracy: 0.7333
Epoch 7/25
20/20 [==============================] - 214s 11s/step - loss: 1.0564 - accuracy: 0.6328 - val_loss: 0.6930 - val_acc
uracy: 0.8025
Epoch 8/25
20/20 [==============================] - 197s 10s/step - loss: 0.8967 - accuracy: 0.6790 - val_loss: 0.5249 - val_acc
uracy: 0.8432
Epoch 9/25
20/20 [==============================] - 194s 10s/step - loss: 0.8519 - accuracy: 0.6920 - val_loss: 0.4802 - val_acc
```

Fig. 9 – Initial Training

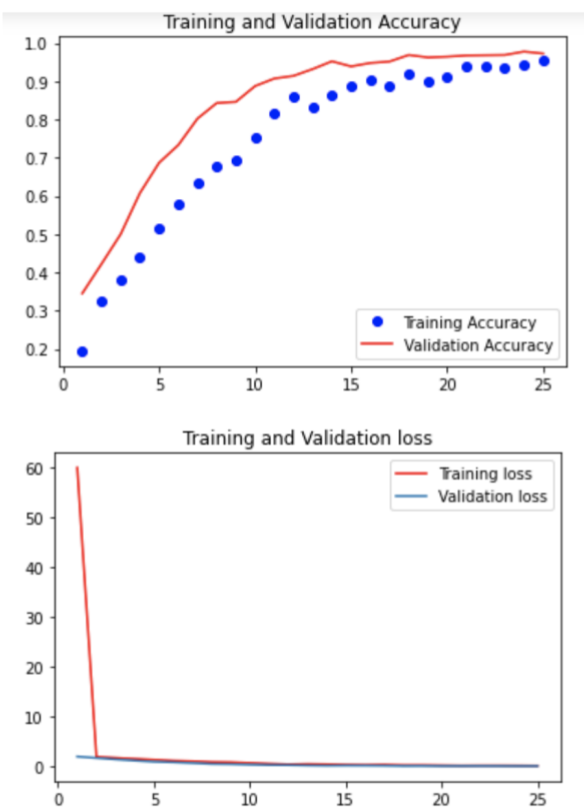

Fig. 10 – Validation/Training Loss and Accuracy

```python
# applying the trained model and making predictions
results = model.evaluate(test_images, verbose=0)

print("    Test Loss: {:.5f}".format(results[0]))
print("Test Accuracy: {:.2f}%".format(results[1] * 100))
```

```
    Test Loss: 0.10351
Test Accuracy: 97.07%
```

Fig. 11 – End Results

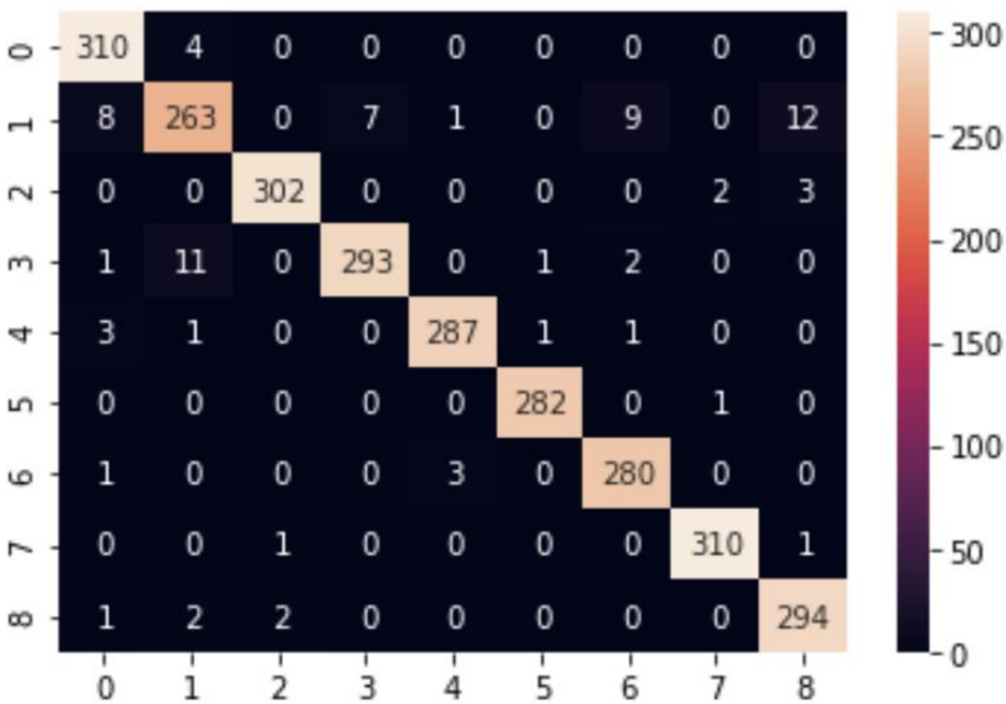

Fig. 12 – Confusion Matrix

	Image_Pathway	Label	pred
2952	/Users/alexanderblaies/Desktop/SpringboardAssi...	Gilt-Head Bream	Shrimp
5624	/Users/alexanderblaies/Desktop/SpringboardAssi...	Black Sea Sprat	Black Sea Sprat
49	/Users/alexanderblaies/Desktop/SpringboardAssi...	Sea Bass	Sea Bass
4668	/Users/alexanderblaies/Desktop/SpringboardAssi...	Shrimp	Shrimp
2504	/Users/alexanderblaies/Desktop/SpringboardAssi...	Gilt-Head Bream	Gilt-Head Bream
7484	/Users/alexanderblaies/Desktop/SpringboardAssi...	Hourse Mackerel	Hourse Mackerel
6942	/Users/alexanderblaies/Desktop/SpringboardAssi...	Striped Red Mullet	Striped Red Mullet
5469	/Users/alexanderblaies/Desktop/SpringboardAssi...	Black Sea Sprat	Black Sea Sprat
6973	/Users/alexanderblaies/Desktop/SpringboardAssi...	Striped Red Mullet	Striped Red Mullet
8640	/Users/alexanderblaies/Desktop/SpringboardAssi...	Trout	Trout
3077	/Users/alexanderblaies/Desktop/SpringboardAssi...	Red Sea Bream	Red Sea Bream
1494	/Users/alexanderblaies/Desktop/SpringboardAssi...	Red Mullet	Red Mullet
6840	/Users/alexanderblaies/Desktop/SpringboardAssi...	Striped Red Mullet	Striped Red Mullet
552	/Users/alexanderblaies/Desktop/SpringboardAssi...	Sea Bass	Sea Bass
8790	/Users/alexanderblaies/Desktop/SpringboardAssi...	Trout	Trout
3478	/Users/alexanderblaies/Desktop/SpringboardAssi...	Red Sea Bream	Red Sea Bream
5461	/Users/alexanderblaies/Desktop/SpringboardAssi...	Black Sea Sprat	Black Sea Sprat
8438	/Users/alexanderblaies/Desktop/SpringboardAssi...	Trout	Trout
4604	/Users/alexanderblaies/Desktop/SpringboardAssi...	Shrimp	Shrimp
6385	/Users/alexanderblaies/Desktop/SpringboardAssi...	Striped Red Mullet	Striped Red Mullet
750	/Users/alexanderblaies/Desktop/SpringboardAssi...	Sea Bass	Sea Bass
8547	/Users/alexanderblaies/Desktop/SpringboardAssi...	Trout	Trout

Fig. 13 – DataFrame Inspection

E. <u>Lessons Learned and Comments:</u>

- This project served as a great introduction into image classification via leveraging Keras
- Based on the results shown above, this model does a great job at identifying individual species with similar characteristics
- It should be noted that I do believe that this model can be improved upon by: including different background contexts (change in lighting, scenery, etc.)
- Also, I did not account for outliers which could potentially further optimize the model
- Due to the computational expense, I chose to limit the epochs and validation steps included within the model; with more convolutional layers, this model could be more accurate