Deep Learning Workshop – Image Classification:

Part1- Exploratory Data Analysis:

Section A+B:

Data Dimensionality- A

The images in the "A Large Scale Fish Dataset" have dimensions of (445, 590, 3), indicating a width of 590 pixels, a height of 445 pixels, and 3 color channels (RGB). The dataset is split into three sets:

-Training set: 6,480 images

(900 images per class in training and validation both)

-Validation set: 1,620 images (20% images from training set)

-Test set: 1,330 images (150 images per class)

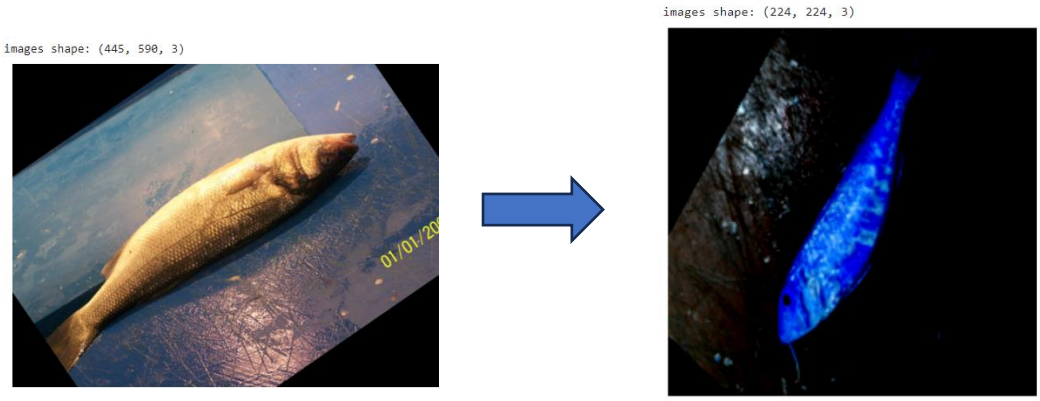
```
Data size:
train - 6480
validation - 1620
test - 1330

# of classes: 9
```

Data Content and Preprocessing-

Each image is at least of size 445*590 with three channels, and there are 9 different classes. Therefore, the images have undergone preprocessing with the following transformations, resizing the images to a fixed size of 224*224 pixels, converting to a tensor and normalizing the pixel values with mean = [0.5,0.5,0.5] and std = [0.5,0.5,0.5].

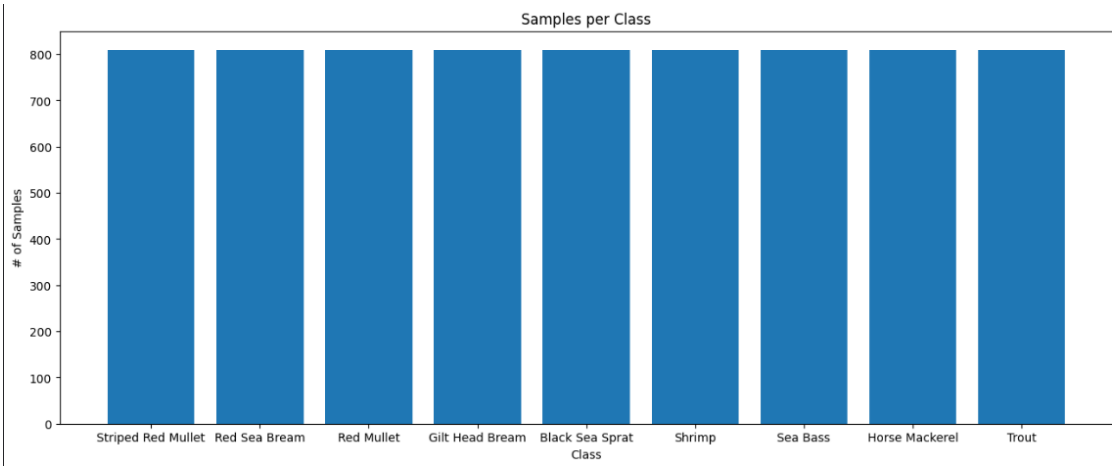
Since the data consists from augmented samples of the same images because there are no many images for same type of fish, we don't need to do further augmentations. The augmentation that has been done on the images is rotation and flip both horizontal and vertical.



Section C:

Data balance-

The dataset is well-balanced, with each of the 9 classes containing 900 samples in both the training set and validation set. This uniform distribution ensures fairness in model training and evaluation. The test set also maintain a similar balance, promoting consistency across all subsets. The balanced class distribution enhances the model's robustness and facilitates unbiased performance assessment across different classes.



Section D:

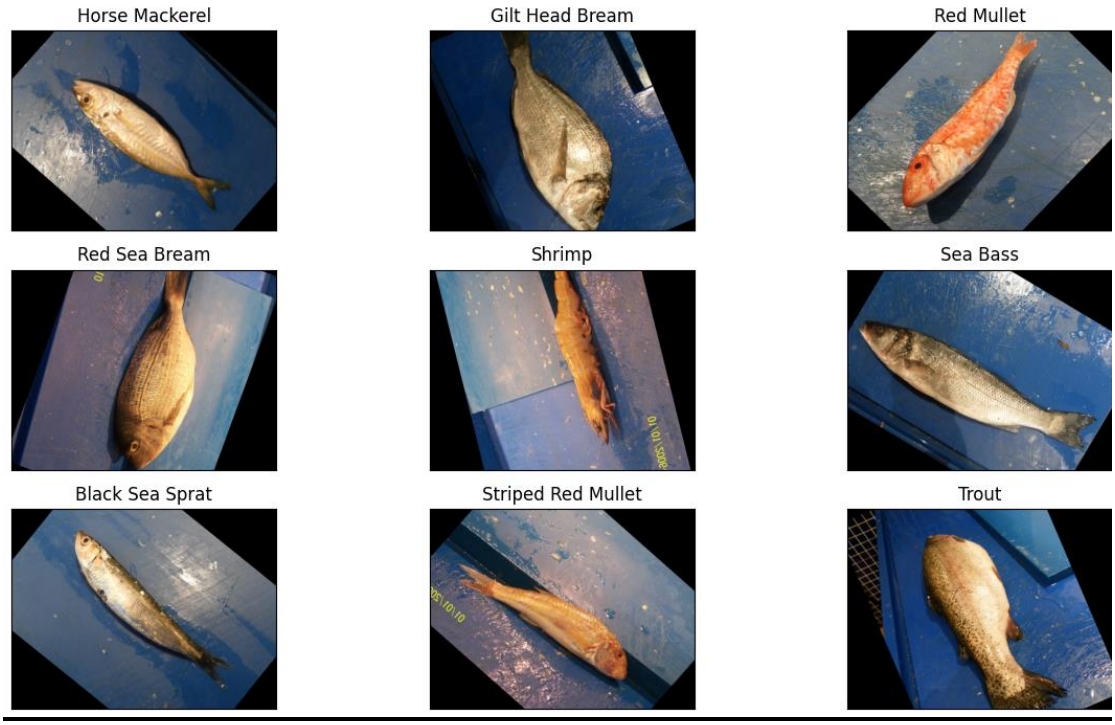
Results batchmark –

We check authors paper that contains benchmark results on datasets. The paper presents MobileNetV2 as the most effective way to classify the fishes. In addition, we saw in Kaggle some examples of using other state-of-the-art CNN architectures such as ResNet50, which achieved good results on this dataset as the authors.

Section E:

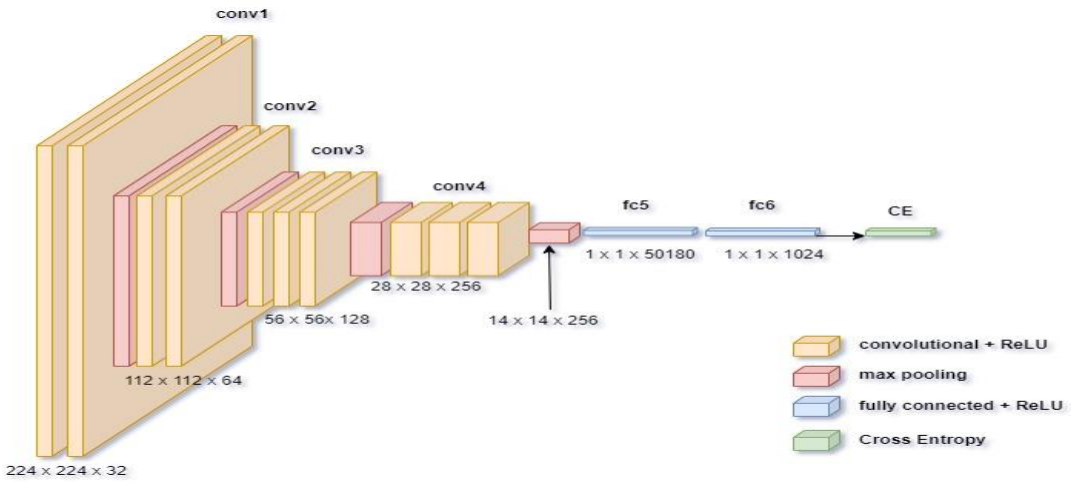
Separability of different classes –

We show samples from each label. As can be seen, the variance between Red Mullet and Shrimp is very high and easy to predict with a basic model. On the other hand, Striped Red Mullet and Red Mullet are two different spices that the variance is very low between them and it can be more difficult for the model to handle this.



Part 2- Implementing Custom CNN Model:

We started by implementing a simple CNN network with four convolution layers (as shown in the figure below) with activation of ReLU and MaxPooling and 2 linear layers for the prediction. In addition, we used batch normalization and add second convolution from same size to same size to examine the impact of batch normalization and additional convolution separately on the simple network performance.



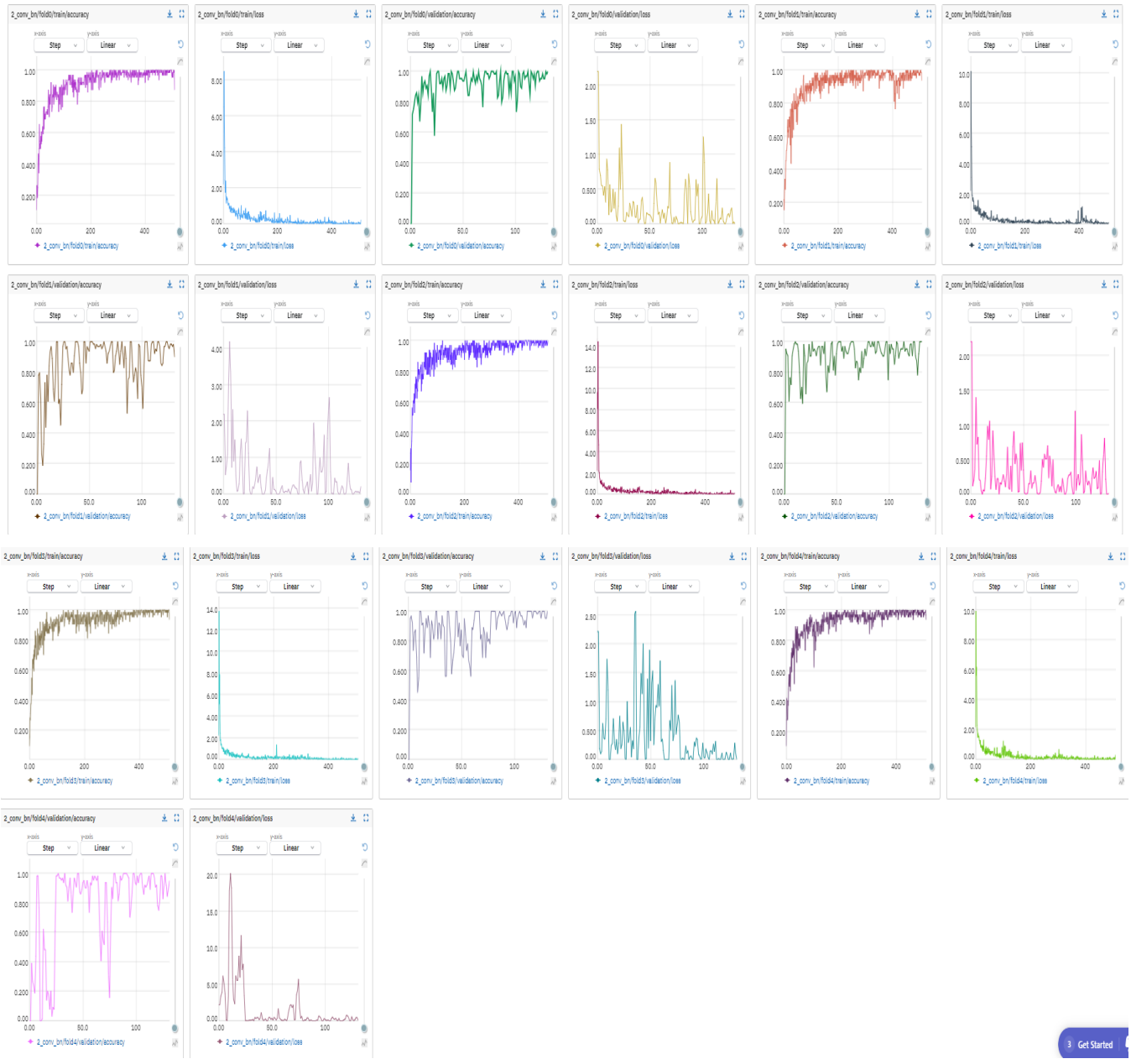
Here is the summary of the model:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 32, 224, 224]             896
              ReLU-2         [-1, 32, 224, 224]               0
         MaxPool2d-3         [-1, 32, 112, 112]               0
          CNNBlock-4         [-1, 32, 112, 112]               0
            Conv2d-5         [-1, 64, 112, 112]          18,496
              ReLU-6         [-1, 64, 112, 112]               0
         MaxPool2d-7           [-1, 64, 56, 56]               0
          CNNBlock-8           [-1, 64, 56, 56]               0
            Conv2d-9          [-1, 128, 56, 56]          73,856
             ReLU-10          [-1, 128, 56, 56]               0
        MaxPool2d-11          [-1, 128, 28, 28]               0
         CNNBlock-12          [-1, 128, 28, 28]               0
           Conv2d-13          [-1, 256, 28, 28]         295,168
             ReLU-14          [-1, 256, 28, 28]               0
        MaxPool2d-15          [-1, 256, 14, 14]               0
         CNNBlock-16          [-1, 256, 14, 14]               0
           Linear-17                [-1, 1024]      51,381,248
             ReLU-18                [-1, 1024]               0
           Linear-19                   [-1, 9]           9,225
================================================================
Total params: 51,778,889
Trainable params: 51,778,889
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 57.44
Params size (MB): 197.52
Estimated Total Size (MB): 255.53
----------------------------------------------------------------
```

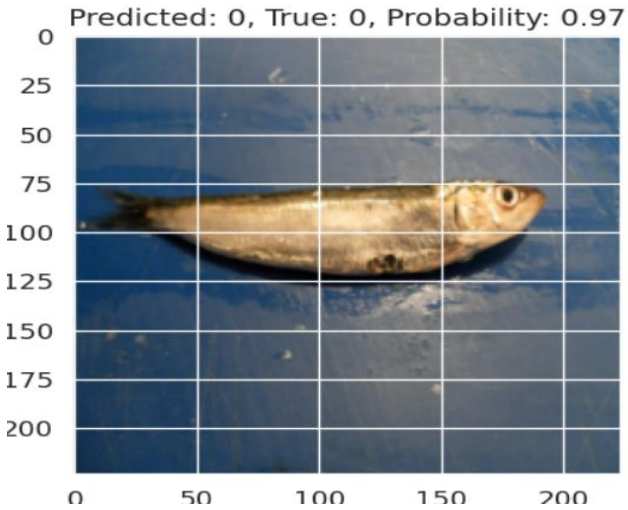
Section A:

Our traditional 5-fold CV experiments were conducted on each training a simple CNN with different combinations of parameters and evaluating them on the validation set: DLWOR-8

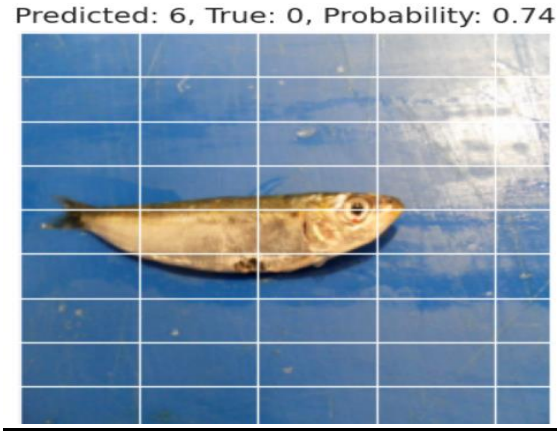
Here, we show the best model (with additional convolution same sized and with batch norm) and his performance (accuracy and loss) on each fold:



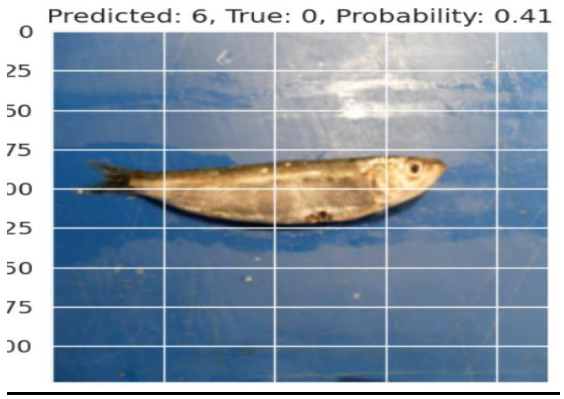
Our best CNN model (with additional convolution same sized and with batch norm) successfully predicts Black Sea Sprat as Black Sea Sprat in the image classification:



On the other hand, when the model failed totally to predict Black Sea Sprat and he classify the fish as Shrimp: (high probability to be Shrimp)



Here is the uncertain prediction for Black Sea Sprat and classify as Shrimp: (low probability to be Shrimp)



The task was to fit a model to the data and analyze the results using various parameters and metrics. In addition, we want to find KFold cross-validation with K=5 that obtain the best accuracy on the validation set. The parameters that we have chosen to test are: "use batch norm "and "additional convolution". All results from Neptune_Link.

Model_Name	1_conv_bn					1_conv_no_bn					2_conv_bn					2_conv_no_bn				
num_folds	fold0	fold1	fold2	fold3	fold4	fold0	fold1	fold2	fold3	fold4	fold0	fold1	fold2	fold3	fold4	fold0	fold1	fold2	fold3	fold4
validation_accuracy	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0.9**	**1**	**1**	**0.95**	**1**	**0**	**1**	**0**	**0**
use_batch_norm	FALSE					TRUE					FALSE					TRUE				
num_layers	6					6					10					10				
Monitoring Time(minutes)	4 min	4 min	4 min	4 min	4 min	4 min	4 min	4 min	4 min	4 min	4 min	4 min	4 min	4 min	4 min	4 min	4 min	4 min	4 min	4 min
train/loss	0.001	0.002	0.00015	0.1247	0.5965	0.0029	0.0077	0.213	0.0057	0.0039	0.224	0.11	0.084	0.107	0.2971	0.045	2.195	0.275	2.2006	2.199
train/acc	1	1	1	0.9375	0.875	1	1	0.9375	1	1	0.875	0.9375	1	0.9375	0.9375	1	0.0625	0.8125	0.125	0.1875
val/loss	0.024	0.013	9.4E-05	6E-05	0	0.011	0.0149	0.017	0.00023	0.025	0.00024	0.215	0.0107	0.000011	0.055	0.0035	2.202	0.201	2.207	2.2075
test/acc	0.863					0.82					0.851	0.905	0.906	0.86	0.893	0.377				
test/loss	1.433					1.098					0.913	0.721	0.96	0.896	0.968	1.974				

The results obtained after training each model with different parameters on each of 5 folds. The model "2_conv_bn" in test set (the basic model with batch norm and with additional convolution) on 3-fold show the highest accuracy.

	model	test_loss	test_acc
0	1_conv_no_bn	1.097148	0.821601
1	1_conv_bn	1.424177	0.864310
2	2_conv_no_bn	1.969679	0.378024
3	2_conv_bn	0.883709	0.883893

	fold	test_loss	test_acc
0	0	0.951400	0.843988
1	1	0.678418	0.904018
2	2	0.949856	0.910506
3	3	0.864345	0.869792
4	4	0.974528	0.891161

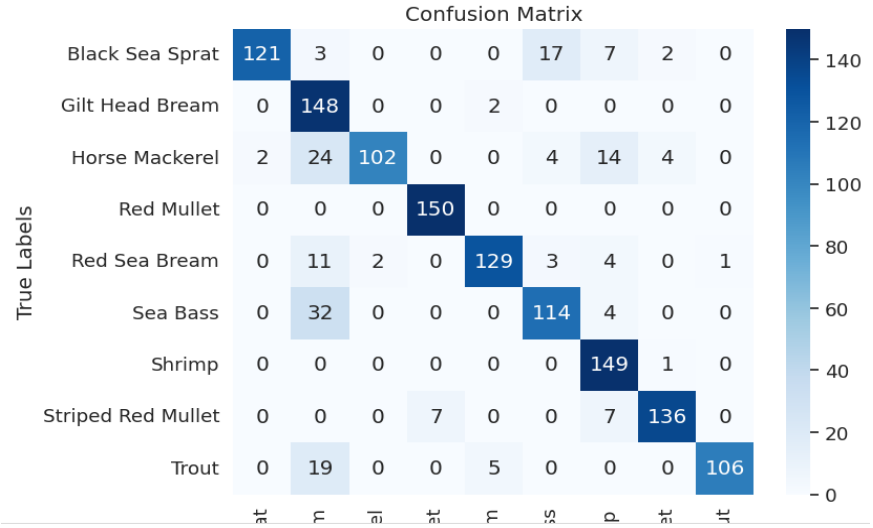
We compare over accuracy because the benchmark that we saw do the comparison on accuracy measure and because the accuracy measures the percentage of correct predictions made by the model.

The metrics obtained from each model all the 5 fold's validation and test data varied depending on their parameters. Therefore, comparing these metrics with the results achieved from the mean of all fold predictions is essential to accurately represent the model's overall performance.

In addition, we measure top-1 and top-5 accuracy the rate of times the correct label is within the top-5 or 1 prediction made by the model. It is essential to consider both metrics to evaluate the model's effectiveness accurately. Also, we measure Recall and Precision to evaluate the model and give Confusion Matrix.

```
top-1 accuracy: 0.868421052631579
top-5 accuracy: 0.9849624060150376

Precision: 0.8962356205660443
Recall: 0.8676353276353277
```



Section B:

It seems like the model is misclassifying samples with relatively high confidence when two samples of fishes seem similar to model but they are different actually. After reviewing the learning curves and the testing results, it appears that the model has gotten overfit with high certainty. We believe that the reason is that the model is too complex for the data and the task, and thus loses the generalization during the training process.

In order to solve this, we suggest 3 solutions:

1. Add a strong regularization to the model (such as dropout).
2. Decrease the complexity of the model architecture by removing layers.
3. Reduce the number of training epochs.

We will implement the first two.

Section C:

We will use the first and second suggested improvements to improve our network's accuracy.

This architecture made by implementing "Dropout" and removing 1

block of CNN that conclude Convolution, ReLU and MaxPooling:

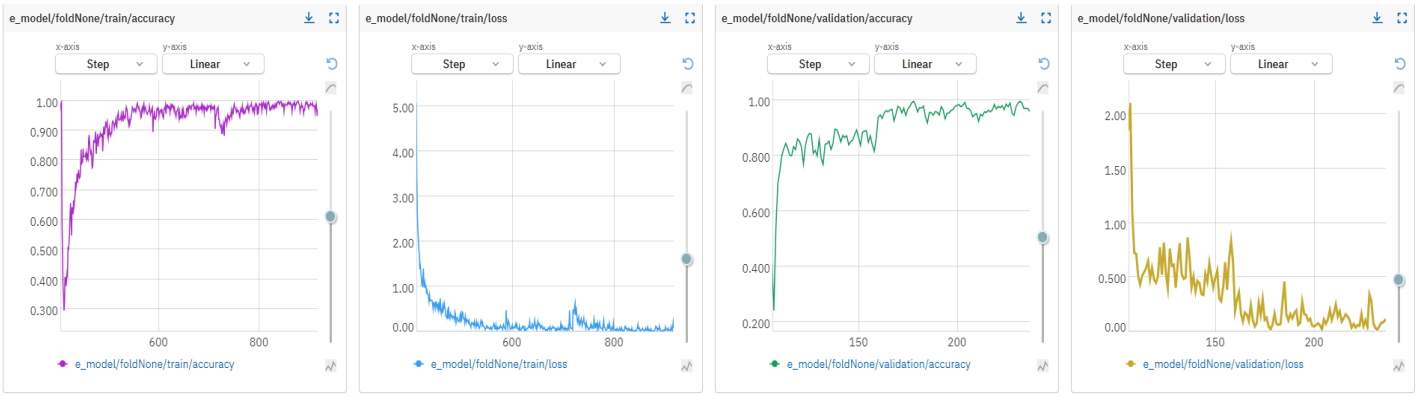
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 32, 224, 224]             896
              ReLU-2         [-1, 32, 224, 224]               0
            Conv2d-3         [-1, 32, 224, 224]           9,248
              ReLU-4         [-1, 32, 224, 224]               0
         MaxPool2d-5         [-1, 32, 112, 112]               0
       BatchNorm2d-6         [-1, 32, 112, 112]              64
          CNNBlock-7         [-1, 32, 112, 112]               0
           Dropout-8         [-1, 32, 112, 112]               0
            Conv2d-9         [-1, 64, 112, 112]          18,496
             ReLU-10         [-1, 64, 112, 112]               0
           Conv2d-11         [-1, 64, 112, 112]          36,928
             ReLU-12         [-1, 64, 112, 112]               0
        MaxPool2d-13           [-1, 64, 56, 56]               0
      BatchNorm2d-14           [-1, 64, 56, 56]             128
         CNNBlock-15           [-1, 64, 56, 56]               0
          Dropout-16           [-1, 64, 56, 56]               0
           Conv2d-17          [-1, 128, 56, 56]          73,856
             ReLU-18          [-1, 128, 56, 56]               0
           Conv2d-19          [-1, 128, 56, 56]         147,584
             ReLU-20          [-1, 128, 56, 56]               0
        MaxPool2d-21          [-1, 128, 28, 28]               0
      BatchNorm2d-22          [-1, 128, 28, 28]             256
         CNNBlock-23          [-1, 128, 28, 28]               0
          Dropout-24          [-1, 128, 28, 28]               0
           Linear-25                [-1, 1024]     102,761,472
             ReLU-26                [-1, 1024]               0
           Linear-27                   [-1, 9]           9,225
================================================================
Total params: 103,058,153
Trainable params: 103,058,153
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 107.20
Params size (MB): 393.14
Estimated Total Size (MB): 500.91
----------------------------------------------------------------
```

The improvement we got was from accuracy= 0.86979 and
loss = 0.864345 to:

```
Avg test loss: 1.3461009103444612
Avg test accuracy: 0.8666071428571429
```

We wanted an improvement, but we don't see the improve from the original model to enhanced model without adding parameters to the model.

We got the following results on training this enhanced model: [DL-WOR30]



Section D:

We used our best model CNN (with batch norm and additional convolution) to examine the effect of inference-time-augmentation on the model performance. When we used ITA, our model test accuracy decreased to 0.8549 (From 0.8697). The reason may be that, as we mentioned, our dataset already contains many augmented images per class. As a result, the decrease was not dramatic and it is already gives us good results.

Section E: (Add Category):

We've added one class to our dataset from:
https://www.kaggle.com/datasets/markdaniellampa/fish-dataset

It had 171 images for the train set and 34 images for the test set. After retraining the model on the "new" dataset, we got the following results:

```
Avg test loss: 2.5179907772690058
Avg test accuracy: 0.7517045454545456
```

The model got worse results compared to the original one. It is possible that it didn't have enough samples to work with (171 vs. ~800 for each class in the original dataset). This resulted poor classification results for the new class and worse results overall.

We can see in neptune.ai the results both the improvement of accuracy and decrease of loss in training set and validation set on this link: [DL-WOR47]

Part3 - Transfer Learning:

Section A+B:

We choose to work with 4 pretrained models from torch on ImageNet data set:

VGG16	MobileNetV2	ResNet50	DenseNet121
134,297,417 parameters	2,235,401 parameters	23,526,473 parameters	6,963,081 parameters

In the fine-tuning process, VGG16, MobileNetV2, ResNet50, andDenseNet121 models were adapted for our fish classification task using PyTorch Lightning.
We selected different pretrained models in terms of complexity of the backbone. As a result, we can see how pretraining's complexity will affect the results of our image classification task.
.
, we replace the last layer of the pretrained model with a linear layer of the output of 9 (the same number of classes). Finally, we trained the network for five epochs.

In fact, the modification involved adjusting output layers to align with task-specific classes with the output of 9 (the same number of classes).

We trained the network for five epochs with initial layers frozen to capitalize on pre-trained knowledge. Then, we trained network for two epochs on all the layers of the pretraining and the output layer.

We specified the following parameters for the trainer-

epochs	devices	accelerator
5	1	auto

In addition, we specified two key two callbacks, Model Checkpoint and LearningRateMonitor.

The Model Checkpoint efficiently saves model weights, focusing on minimizing validation loss (val_loss). This ensures optimal performance on unseen data. Simultaneously, the LearningRateMonitor logs learning rates after each epoch, providing valuable insights into the optimization dynamics.

Together, these callbacks contribute to effective model training, optimizing memory usage and enhancing understanding of the learning rate's evolution throughout the training duration.

The training loop went through each model, printing their names, and training them with specific data.

After the initial training, all layers were slowly unfrozen to fine-tune them for our specific task. This method balances using what the models already know from previous training and adjusting them to perform better on our specific dataset.

We got the results in neptune.ai that show the accuracy improvements and decrease of loss over training and validation set: DL-WOR44

Show example DenseNet:

Section C:

The results we achieved on the testing-set:

	index	# parameters	Validation Loss	Validation Accuracy	Test Loss	Test Accuracy	# unique correct samples	# unique errors
0	vgg16	134297417.0	2.1000	0.178	2.140871	0.164732	0.0	1005.0
1	mobilenet_v2	2235401.0	0.0280	0.991	0.110365	0.967262	7.0	21.0
2	resnet50	23526473.0	0.0980	0.966	0.212286	0.937500	1.0	4.0
3	densenet121	6963081.0	0.0043	0.998	0.037264	0.987351	5.0	0.0

In the evaluation of the models,

VGG16, the observed lower accuracy and higher validation loss indicate a potential misalignment between the model's architecture and the intricacies of the task at hand.

MobileNetV2 demonstrated excellent performance, achieving high accuracy with minimal validation loss and exhibiting robustness with few errors.

ResNet50 The model showcased an ability to adapt to the intricacies of the fine-tuning task, refining its understanding and improving accuracy during the training process.

DenseNet121 stood out with exceptional performance, demonstrating very high accuracy on both validation and test sets and notably, no unique errors, emphasizing its strong classification capabilities.

In summary, we can see that DenseNet121 achieves the best results.

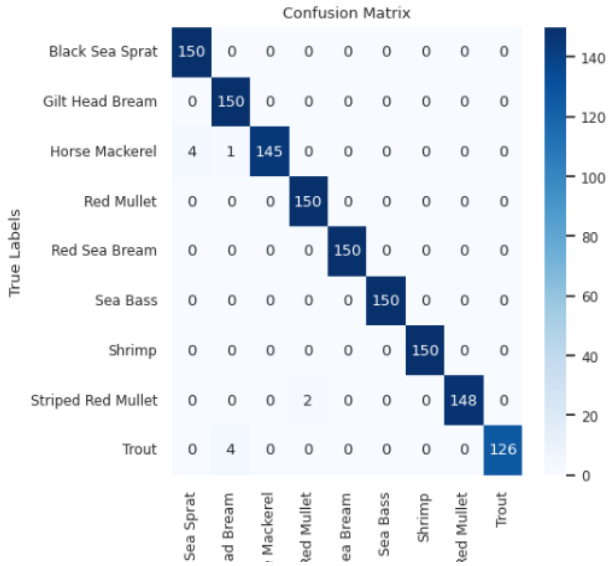
Section D:

Features were extracted using a pre-trained DenseNet121 model and then were evaluated with three classifiers: XGBoost, Random Forest, and SVM. Performance metrics, including accuracy, precision, and recall, were computed to assess each classifier's efficiency. Confusion matrices visually highlighted strengths and weaknesses across different classes. This holistic approach validates transfer learning capabilities and guides potential refinements for pretrained model with ML effectiveness in the task at hand.

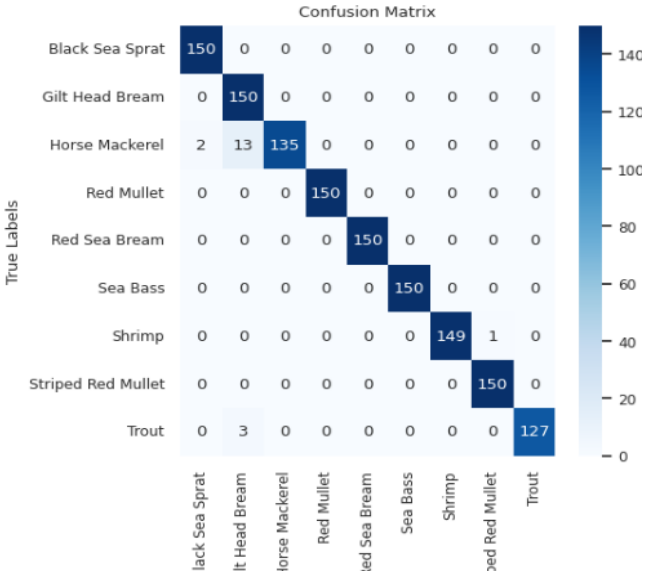
This is the results we achieved on the testing-set:

Model	accuracy	precision	recall
SVC	0.9909	0.9920	0.9913
RandomForestClassifier	0.9857	0.9870	0.9855
XGBClassifier	0.9917	0.9914	0.9906

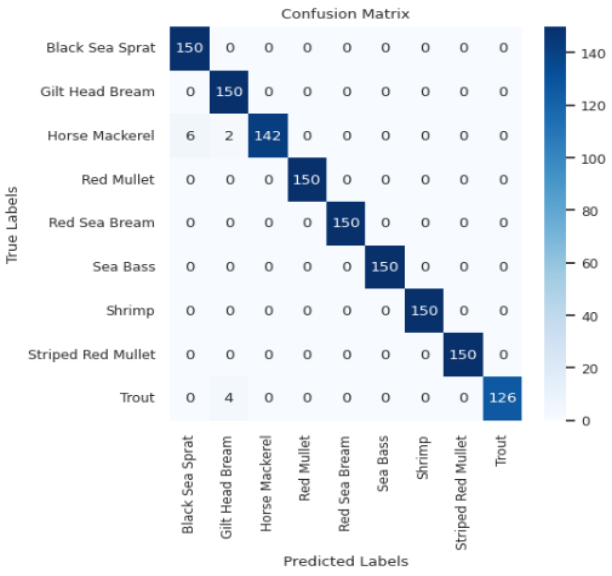
```
XGBClassifier
Test accuracy: 0.9917293233082707
Precision: 0.9920677794192225
Recall: 0.9913960113960114
```




```
RandomForestClassifier
Test accuracy: 0.9857142857142858
Precision: 0.9870926718375769
Recall: 0.9855840455840456
```




```
SVC
Test accuracy: 0.9909774436090225
Precision: 0.9914529914529915
Recall: 0.9906552706552707
```



The results above shows that XGBoost (XGB) achieved the highest overall metrics, with an accuracy of 99.17%, precision of 99.20%, and recall of 99.13%. RandomForestClassifier and SVM classifier also

demonstrated robust performance, with accuracy scores of 98.57% and 99.06%, respectively.

Part4 – Summarization and additional marks:

As we mentioned, Our entire experiment plan and results are in this link (neptune.ai). We also added an HTML of our notebook, which contains all the functions and tasks required in this assignment.

In addition, all of our experiments were performed Kaggle clusters. We took all the datasets directly from there and create our own database in kaggle for models.

In conclusion, after conducting a series of experiments utilizing diverse methodologies such as simple custom CNNs, transfer learning, and feature extraction, we have successfully attained exceptional accuracy in categorizing images belonging to 9 distinct fish species.

These outcomes underscore the significance of exploring a range of architectures and techniques to identify the optimal solution tailored to the unique demands of the task at hand. In our investigation, the fusion of DenseNet with transfer learning emerged as the most potent strategy, yielding superior results.