Fish Classification - MO343

Instituto de Computação - UNICAMP

Vitória Maria Carneiro Mathias

I. Introduction

In order to apply my acquired knowledge during the course MO434 - Deep Learning, I made 5 models with different approaches to classify fish types from the Fish Dataset [1], inspired by the notebooks provided by the teacher.

First, a general data process is applied to use the images as inputs and the type of fish as classes - once they're split on train, validation and test. Then, that dataset is used for a Convolutional Neural Network (CNN) created layer by layer and on pretrained VGG, ResNet and DenseNet networks. We will see more about how they work on each of the following sections.

Then we try a different approach using Contrastive Learning, which requires less samples and can identify if two images are from the same class or not. To do so, the data process changes a little, because now the input are tow images and the output is only true or false.

In general, the pretrained networks worked better, due to a large set of train images. But it's interesting to visually see and relevant to understand what is happening on each layer of our built networks. So there are two sections dedicated to present and analyze some scatter plots representing the clusterization each layer is performing and a heat map of the image indicating which part of the image is the most "paid attention" to.

II. Data Process

i. CNN and DNN

To preprocess the images, I reduced their size from 1024x768 to 224x224 - so I could process it, otherwise the memory from the CPU I was using couldn't run it after some tests. Originally there was 430 images, divided on 9 classes of fishes. Since this is a really small dataset to train over, I randomly duplicated 500 more images from the train set and applied an augmentation transformation to make sure each image is somehow unique. The transformations applied are the following:

For test and validation sets:

- Resize, performing a bi-linear interpolation
- Transform the image to tensor
- Normalize the pixel values using the mean and standard deviation from the original set of images

For train set:

- Resize, performing a bi-linear interpolation
- Rotation and translation of the image, by random values
- Transform the image to tensor
- Normalize the pixel values using the mean and standard deviation from the original set of images

ii. Contrastive learning

As said, to this part, the dataset was treated differently: I separated the images on two folders - test and train - and then took randomly two images inside a same folder to build one item to my dataset. Then, I applied the same two composition of transformations to each image and saved the pair of images images and their target as 1 or 0 (same or different images, respectively).

III. Convolutional Neural Network (CNN)

As expected, some factors improved the accuracy:

- Number of epochs (more the better)
- Image size
- Size of train dataset

But it's interesting to note that it's not valid for the batch size and number of convolutions. That's because the batch size is a compromise between accuracy, training time and regularization, which is also limited by the size of the available dataset.

Also, a complex convolutional neural network can has an accuracy worst than one with less layers of convolution, it depends on other manipulation, as dropout, learning rate on gradient descent, and so on.

To keep track of some changes and its effects, I made table of results from the last epoch trained from some tests:

Img Size	Activation function	Batch size	Base cor of CNN	Validation accuracy	Train accuracy
224x224	ReLU	25	conv1, conv2, concat	0.060	0.182
512x384	ReLU	25	conv1, conv2, concat	0.202	0.382
224x224	ReLU	25	conv1, conv2, conv3, conv4	0.120	0.100
224x224	ReLU	25	conv1, conv2, concat, conv3, conv4	0.093	0.162
224x224	ReLU	10	conv1, conv2, conv3, conv4	0.139	0.132
224x224	ReLU	30	conv1, conv2, concat, conv3, conv4,concat, conv5	0.433	0.527
224x224	SoftMax	30	conv1, conv2, concat, conv3, conv4,concat, conv5	0.093	0.172
512x384	SoftMax	30	conv1, conv2, concat, conv3, conv4,concat, conv5	0.122	0.133
512x384	SoftMax	30	conv1, conv2, concat, conv3, conv4,concat, conv5	0.102	0.149

Figure 1: Table with tests

I also tried to change the values of mean and standard deviation when normalizing the images - on the preprocess phase - but it did not change significantly the results. However, changes on the percentage of partition on test, validation and train datasets showed a difference on the results. You can see an image presenting the final architecture on image 2.

Some hyperparameters can influence on the results, so I explored it once I've chosen my CNN architecture. I expected that learning rate and learning step on the optimizer could they stabilize the results over the epochs on accuracy and loss, but it didn't happened. For what I understand, that's because my dataset is not really big. Even if I make an augmentation, the overall is not varied, so the metrics showed no improvement.

The greatest difference I observed was when I changed the train, validation and test proportion. With more data to train, the performance tends to be better, but also the test can mislead the results. That said, the results of mean and standard deviation for Cohen's Kappa and accuracy per class on the test set (with 43 images, since I prioritized the training) are shown on figure 9.

Looking for benchmarks with notebooks availables on kaggle to this dataset, the both top performers used CNN with two dense layers after the convolution, resulting on accuracy about 90%. That gives us a taste of how the performance can improve with the DNN on next section.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Sequential: 1-1                        [-1, 16, 384, 512]        --
|    └─Conv2d: 2-1                       [-1, 16, 384, 512]        432
|    └─BatchNorm2d: 2-2                  [-1, 16, 384, 512]        32
|    └─ReLU: 2-3                         [-1, 16, 384, 512]        --
|    └─MaxPool2d: 2-4                    [-1, 16, 384, 512]        --
├─Sequential: 1-2                        [-1, 64, 192, 256]        --
|    └─Conv2d: 2-5                       [-1, 64, 384, 512]        9,216
|    └─BatchNorm2d: 2-6                  [-1, 64, 384, 512]        128
|    └─ReLU: 2-7                         [-1, 64, 384, 512]        --
|    └─MaxPool2d: 2-8                    [-1, 64, 192, 256]        --
├─Sequential: 1-3                        [-1, 128, 96, 128]        --
|    └─Conv2d: 2-9                       [-1, 128, 192, 256]       92,160
|    └─BatchNorm2d: 2-10                 [-1, 128, 192, 256]       256
|    └─ReLU: 2-11                        [-1, 128, 192, 256]       --
|    └─MaxPool2d: 2-12                   [-1, 128, 96, 128]        --
├─Sequential: 1-4                        [-1, 256, 48, 64]         --
|    └─Conv2d: 2-13                      [-1, 256, 96, 128]        294,912
|    └─BatchNorm2d: 2-14                 [-1, 256, 96, 128]        512
|    └─ReLU: 2-15                        [-1, 256, 96, 128]        --
|    └─MaxPool2d: 2-16                   [-1, 256, 48, 64]         --
├─Sequential: 1-5                        [-1, 512, 24, 32]         --
|    └─Conv2d: 2-17                      [-1, 512, 48, 64]         1,769,472
|    └─BatchNorm2d: 2-18                 [-1, 512, 48, 64]         1,024
|    └─ReLU: 2-19                        [-1, 512, 48, 64]         --
|    └─MaxPool2d: 2-20                   [-1, 512, 24, 32]         --
├─Sequential: 1-6                        [-1, 9]                   --
|    └─Linear: 2-21                      [-1, 512]                 201,327,104
|    └─ReLU: 2-22                        [-1, 512]                 --
|    └─Dropout: 2-23                     [-1, 512]                 --
|    └─Linear: 2-24                      [-1, 9]                   4,617
==========================================================================================
```

Figure 2: CNN architecture with 5 convolutional layers and 2 concatenations between each 2 convolutions

	Mean kappa	Std kappa	Mean accuracy	Std accuracy	Fish class
0	0.8	0.4	0.8	0.4	Black Sea Sprat
1	0.0	0.0	0.0	0.0	Gilt-Head Bream
2	0.0	0.0	0.0	0.0	Hourse Mackerel
3	0.0	0.0	0.0	0.0	Red Mullet
4	0.0	0.0	0.0	0.0	Red Sea Bream
5	0.0	0.0	0.0	0.0	Sea Bass
6	0.9	0.3	0.9	0.3	Shrimp
7	0.0	0.0	0.0	0.0	Striped Red Mullet
8	0.0	0.0	0.0	0.0	Trout

Figure 3: Table with mean and standard deviaton of Cohen's kappa score and accuracy for each class for the CNN

IV. Pre-trained Deep Neural Network (DNN)

i. VGG (Visual Geometry Group)

The VGG Net is a deep CNN, who counts with 16 convolutional layers presenting overall accuracy of 92.7%. For this project I used the pre-trained version trained on the ImageNet dataset, which contains 1000 different

object classes, so it fits our purpose to classify images.

ii. ResNeT(Deep Residual Learning)

The ResNet is a 34 layer convolutional neural network also trained with the ImageNet dataset over 200 classes. The different approach this network brings is that it takes residuals from each layer and uses them in the subsequent connected layers.

It's interesting to note that for train and validation accuracy the means were close to one, but as we will see on coming, the test perfomance indicates this model is really good over fitting.

iii. DenseNet (Dense Convolutional Network)

The DenseNet is also a convolutional neural network with dense connections between each layer to other layers.

Here I summarize the results from them over a test dataset of same size of the test dataset from the CNN section before and created by doing the same data pre-process:

	Mean kappa	Std kappa	Mean accuracy	Std accuracy	Fish class
0	0.444444	0.496904	0.444444	0.496904	Black Sea Sprat
1	0.473684	0.499307	0.473684	0.499307	Gilt-Head Bream
2	0.545455	0.497930	0.545455	0.497930	Hourse Mackerel
3	0.571429	0.494872	0.571429	0.494872	Red Mullet
4	0.117647	0.322190	0.117647	0.322190	Red Sea Bream
5	0.523810	0.499433	0.523810	0.499433	Sea Bass
6	0.937500	0.242061	0.937500	0.242061	Shrimp
7	0.450000	0.497494	0.450000	0.497494	Striped Red Mullet
8	0.454545	0.497930	0.454545	0.497930	Trout

Figure 4: Table with mean and standard deviaton of Cohen's kappa score and accuracy for each class for VGG model

	Mean kappa	Std kappa	Mean accuracy	Std accuracy	Fish class
0	0.111111	0.314270	0.111111	0.314270	Black Sea Sprat
1	0.210526	0.407682	0.210526	0.407682	Gilt-Head Bream
2	0.045455	0.208299	0.045455	0.208299	Hourse Mackerel
3	0.000000	0.000000	0.000000	0.000000	Red Mullet
4	0.117647	0.322190	0.117647	0.322190	Red Sea Bream
5	0.285714	0.451754	0.285714	0.451754	Sea Bass
6	0.187500	0.390312	0.187500	0.390312	Shrimp
7	0.150000	0.357071	0.150000	0.357071	Striped Red Mullet
8	0.090909	0.287480	0.090909	0.287480	Trout

Figure 5: Table with mean and standard deviaton of Cohen's kappa score and accuracy for each class for ResNet model

We can see that the quantity of layers, the size of the train set available and the existence of a dense layer on all the pretrained models reflects on a great performance when we compare to a simple CNN, as the one from the

	Mean kappa	Std kappa	Mean accuracy	Std accuracy	Fish class
0	0.111111	0.314270	0.111111	0.314270	Black Sea Sprat
1	0.210526	0.407682	0.210526	0.407682	Gilt-Head Bream
2	0.045455	0.208299	0.045455	0.208299	Hourse Mackerel
3	0.000000	0.000000	0.000000	0.000000	Red Mullet
4	0.117647	0.322190	0.117647	0.322190	Red Sea Bream
5	0.285714	0.451754	0.285714	0.451754	Sea Bass
6	0.187500	0.390312	0.187500	0.390312	Shrimp
7	0.150000	0.357071	0.150000	0.357071	Striped Red Mullet
8	0.090909	0.287480	0.090909	0.287480	Trout

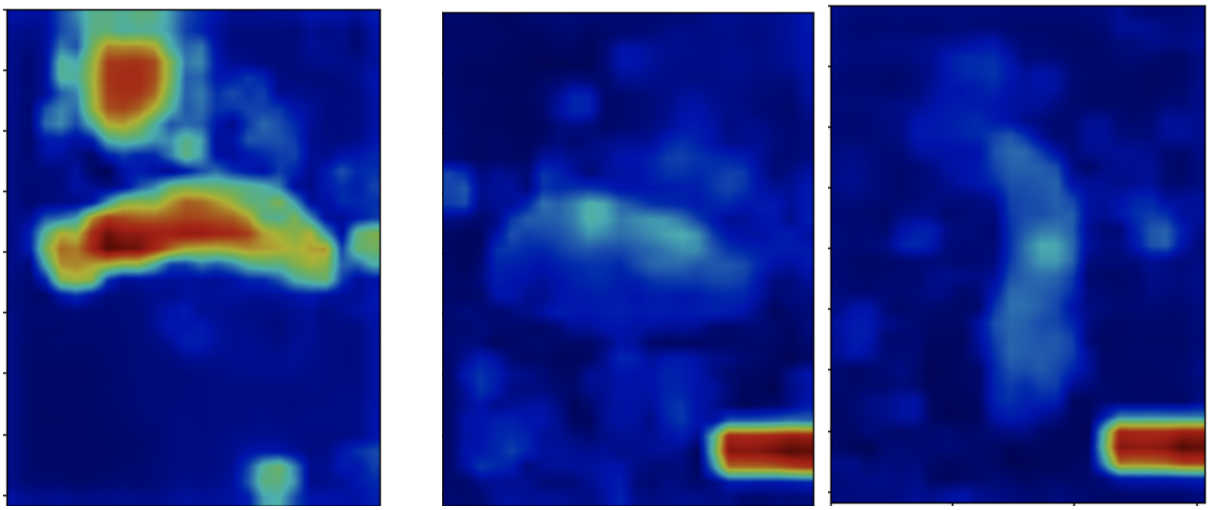
Figure 6: Table with mean and standard deviaton of Cohen's kappa score and accuracy for each class for DenseNet model

first section. Here, the accuracy could be improved by adapting the pre-processing for each pretrained neural network.

But the accuracy from the two last are not great, only VGG did a great job. And beyond that, monitoring the train accuracy I noted that the ResNet was overfitting. That could be due to the train set size, or improvements on the auxiliar functions when using these pretrained models.

V. Areas of activation

Using the GradCAM technique one can take the activated neurons and map the regions on the image that are activated, i.e., the most relevant parts of the image used on the classification. Below I show the superposed heatmaps of activation of three classes, being the first and the last with better performances on the test.



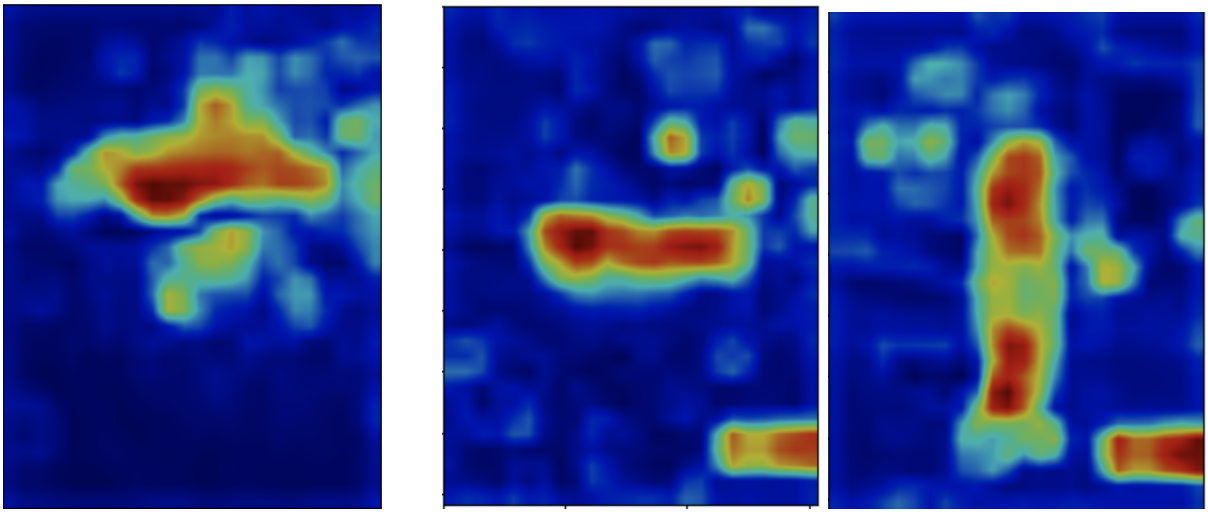
(a) Black Sea Sprat (b) Red Mullet (c) Shrimp

Figure 7: Regions of attention for 3 different classes of fish

It's clear that cutting off the part of the image with the date , my CNN model may be able to focus on the fish. Although that doesn't explain why the model had good performance for the Shrimp class. Maybe it is classifying the most part of images with dates as shrimps, and having at least one class right. From these 3 examples, the only heatmap that makes sense is the first one, even if we can see that it's focusing on some peripheral points -

probably scales with high contrast on the image.

So, to test that hypotesis of the date, I re-trained and re-tested the same CNN with images with 1/6 of the image cropped (from bottom to top), and these are the results - metrics and heat map:



(a) Black Sea Sprat (b) Red Mullet (c) Shrimp

Figure 8: Regions of attention for 3 different classes of fish

	Mean kappa	Std kappa	Mean accuracy	Std accuracy	Fish class
0	0.0	0.0	0.0	0.0	Black Sea Sprat
1	0.0	0.0	0.0	0.0	Gilt-Head Bream
2	0.0	0.0	0.0	0.0	Hourse Mackerel
3	0.0	0.0	0.0	0.0	Red Mullet
4	0.0	0.0	0.0	0.0	Red Sea Bream
5	0.0	0.0	0.0	0.0	Sea Bass
6	1.0	0.0	1.0	0.0	Shrimp
7	1.0	0.0	1.0	0.0	Striped Red Mullet
8	0.0	0.0	0.0	0.0	Trout

Figure 9: Table with mean and standard deviaton of Cohen's kappa score and accuracy for each class for the CNN

From that we see that indeed, the attention focused on the fish this time - even if some images still have the date - but the metrics didn't improve. I understand that this is due to the size of the train set.

VI. Contrastive Learning

In order to improve or equalize the results from the CNNs using a smaller test set, I made a CNN using contrastive learning, that is, the input data are two images and the label is 0 - not the same class - or 1 - same class.

For the pre-processing, for each image, the set of transformations are the same from the CNN, the difference is that to make the dataset, I opened each pair of images - from the list of image paths to train and test separated - and set the target as 0 or 1.

The architecture of this model is simple: three convolutional layers with batch normalization, ReLU and MaxPooling without connections. That said, the mean overall accuracy and loss are:

	Loss mean	Loss std	Accuracy mean	Accuracy std
0	0.246871	0.058135	0.838315	0.072952

Figure 10: Table with overall mean and standard deviation of Cohen's kappa score and accuracy for contrastive learning

Then, I saved the dictionary with the weights of each layer and changed the forward method, so it could be a classifier for the fish images. Also, I changed the input to be just one image and these are the overall results and the results per class:

	Loss mean	Loss std	Accuracy mean	Accuracy std
0	2.4482	0.159535	0.083333	0.056095

Figure 11: Table with overall mean and standard deviation of Cohen's kappa score and accuracy for CNN with contrastive weights

	Mean kappa	Std kappa	Mean accuracy	Std accuracy	Fish class
0	0.000000	0.000000	0.000000	0.000000	Black Sea Sprat
1	0.000000	0.000000	0.000000	0.000000	Gilt-Head Bream
2	0.842105	0.364642	0.842105	0.364642	Hourse Mackerel
3	0.000000	0.000000	0.000000	0.000000	Red Mullet
4	0.000000	0.000000	0.000000	0.000000	Red Sea Bream
5	0.000000	0.000000	0.000000	0.000000	Sea Bass
6	0.000000	0.000000	0.000000	0.000000	Shrimp
7	0.000000	0.000000	0.000000	0.000000	Striped Red Mullet
8	0.000000	0.000000	0.000000	0.000000	Trout

Figure 12: Table with mean and standard deviation of Cohen's kappa score and accuracy for each class for CNN with contrastive weights

The results indicates that, for this architecture at least, using directly the weights that were the best to do a comparison does not lead to a good classifier. If one try to retrain it, it would be just like doing the section CNN with 3 convolutional layers without connecting any of them. For what I tested, more layers and connected are slightly better.

VII. Class separation - CNN

Now, let's see in form of clusters how the CNN separate the classes: We see that it can create clusters, but they are really confused with each other, so the network is not doing a good job by separating the classes.

VIII. Conclusion

From that work I could recognize the importance of a good dataset preparation and size, seeing the difference between results from my simple CNN and dense pre trained networks. There are some variables I didn't change or tested as different initialization for the CNN model weights, play with the dropout size and other arrangements on the architecture.

But it was still possible to study the behavior of the neural network layer by layer with the separation of classes and its attention - shown by the heatmaps. On that part specifically I believe a lot exploration can be done. For

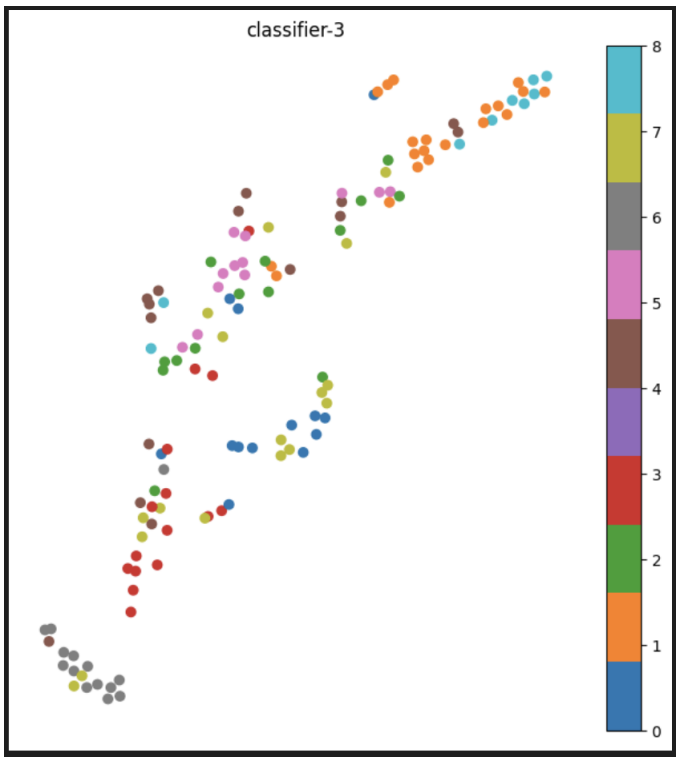


Figure 13: Clusterization from last hidden layer of CNN

example, say the purpose of this model was to be used at sea, the image would be more blue and blured, so the attention would change - and so the results.

During the semester we discussed in class the contrastive learning, which seems to be great to separate different images and faster since it requires less images. But, for the purpose of classification, when using my initial CNN, the metrics are no better overall.

Finally, I understand that doing different tests in parallel can lead one to go back and forward when creating their neural network. Maybe some treatments I made all combined - as cutting of the date from the images, initializing weights with the contrastive learning weights but fine tuning it and expanding the dataset with not only an artifial augmentation - could generate a better neural network.

References

[1] https://www.kaggle.com/datasets/crowww/a-large-scale-fish-dataset