

Five sea animals I used: ['Eel', 'Octopus', 'Otter', 'Seal', 'Sharks']

(i)

First model:

Layer (type)	Output Shape	Param #
rescaling (Rescaling)	(None, 100, 100, 3)	0
conv2d (Conv2D)	(None, 98, 98, 32)	896
max_pooling2d (MaxPooling2D)	(None, 49, 49, 32)	0
dropout (Dropout)	(None, 49, 49, 32)	0
conv2d_1 (Conv2D)	(None, 47, 47, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 23, 23, 64)	0
dropout_1 (Dropout)	(None, 23, 23, 64)	0
conv2d_2 (Conv2D)	(None, 21, 21, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 10, 10, 128)	0
dropout_2 (Dropout)	(None, 10, 10, 128)	0
flatten (Flatten)	(None, 12800)	0
dense (Dense)	(None, 128)	1638528
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 5)	645
=====		
Total params: 1,732,421		
Trainable params: 1,732,421		
Non-trainable params: 0		

This is a CNN (Convolutional Neural Network) architecture with the following layers:

- Rescaling: This layer rescales the pixel values of the input images between 0 and 1.
- Conv2D: This layer applies 32 filters of size 3x3 to the input image to extract 32 feature maps.
- MaxPooling2D: This layer performs a 2x2 max pooling operation on the feature maps to downsample them and reduce the spatial size of the output.
- Dropout: This layer randomly drops some of the neurons during training to prevent overfitting.
- Conv2D: This layer applies 64 filters of size 3x3 to the output of the previous layer to extract 64 feature maps.
- MaxPooling2D: This layer performs a 2x2 max pooling operation on the feature maps to downsample them.
- Dropout: This layer randomly drops some of the neurons during training.
- Conv2D: This layer applies 128 filters of size 3x3 to the output of the previous layer to extract 128 feature maps.
- MaxPooling2D: This layer performs a 2x2 max pooling operation on the feature maps to downsample them.

- Dropout: This layer randomly drops some of the neurons during training.
- Flatten: This layer flattens the output of the previous layer into a 1D vector.
- Dense: This layer applies 128 neurons with ReLU activation to the flattened output.
- Dropout: This layer randomly drops some of the neurons during training.
- Dense: This layer applies 5 neurons with softmax activation as the output layer to classify the input image into one of five classes.

The total number of trainable parameters in this CNN architecture is 1,732,421.

Second Model:

Layer (type)	Output Shape	Param #
rescaling (Rescaling)	(None, 100, 100, 3)	0
conv2d (Conv2D)	(None, 98, 98, 32)	896
max_pooling2d (MaxPooling2D)	(None, 49, 49, 32)	0
dropout (Dropout)	(None, 49, 49, 32)	0
conv2d_1 (Conv2D)	(None, 47, 47, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 23, 23, 64)	0
dropout_1 (Dropout)	(None, 23, 23, 64)	0
conv2d_2 (Conv2D)	(None, 21, 21, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 10, 10, 128)	0
dropout_2 (Dropout)	(None, 10, 10, 128)	0
flatten (Flatten)	(None, 12800)	0
dense (Dense)	(None, 128)	1638528
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 5)	645
=====		
Total params: 1,732,421		
Trainable params: 1,732,421		
Non-trainable params: 0		

This is a CNN (Convolutional Neural Network) architecture with the following layers:

- rescaling" layer: This layer rescales the input image to a size of 100 x 100 with 3 color channels (RGB).
- "conv2d" layer: This is a convolutional layer with 32 filters and a kernel size of 3x3. This layer extracts 32 different features from the input image.
- "max_pooling2d" layer: This is a max pooling layer that reduces the spatial dimensions of the feature maps output by the previous layer by taking the maximum value within each 2x2 region. This reduces the number of parameters and helps prevent overfitting.
- "dropout" layer: This is a regularization layer that randomly drops out (sets to zero) some of the activations of the previous layer during training. This helps prevent overfitting.

- "conv2d_1" layer: This is another convolutional layer with 64 filters and a kernel size of 3x3. This layer extracts more complex features from the input image.
- "max_pooling2d_1" layer: Another max pooling layer that further reduces the spatial dimensions of the feature maps.
- Another "dropout" layer.
- "conv2d_2" layer: This is another convolutional layer with 128 filters and a kernel size of 3x3.
- "max_pooling2d_2" layer: Another max pooling layer.
- Another "dropout" layer.
- "flatten" layer: This layer flattens the 3D feature maps output by the previous layer into a 1D vector.
- "dense" layer: This is a fully connected layer with 128 neurons, which applies a linear transformation to the input vector.
- Another "dropout" layer.
- "dense_1" layer: This is the final layer, a fully connected layer with 5 neurons corresponding to the 5 classes in the dataset. This layer outputs the probabilities of each class based on the input image.

(ii)

First Model:

In this training run, I used 50 epochs. Training accuracy started at 40.76% in the first epoch and gradually increased over the epochs. By the end of training, in the 50th epoch, the training accuracy reached 97.51%.

The training accuracy generally increased as the number of epochs increased, with some fluctuations in between. The model started to overfit the training data as the number of epochs increased, which is evident from the gradual decrease in the rate of increase in accuracy in the later epochs.

It is worth noting that, while high training accuracy is desirable, it doesn't necessarily mean the model will perform well on unseen data. It is important to evaluate the model's performance on a separate validation set to check for overfitting and select the best model based on its performance on the validation set.

Second Model:

The model was trained for 50 epochs. Initially, the training accuracy was around 43% in the first epoch, and it gradually increased in the following epochs. The training accuracy continued to improve until it reached 100% in the 48th epoch. However, it is important to note that a model with 100% training accuracy may not always generalize well on unseen data. The accuracy

fluctuations in some of the epochs, such as in epochs 29 and 37, could be due to random variations in the data or noise in the model. Overall, the trend is that the accuracy increased with each epoch, indicating that the model is learning and improving over time.

- ❖ It seems that the two models have converged. In general, convergence occurs when the performance metric (in this case, training accuracy) stabilizes and stops improving significantly with additional epochs.
- ❖ Note: The number of epochs used in training is dependent on various factors such as the complexity of the model, the size of the dataset, the batch size, and the learning rate used for optimization.

Based on the training accuracy changing over epochs, it appears that the model continues to improve and the training accuracy continues to increase with more epochs up to a point where the accuracy plateaus. This suggests that increasing the number of epochs beyond the point where the accuracy plateaus may not result in significant improvements in the model's performance.

Additionally, it's important to note that training for too many epochs can lead to overfitting, where the model becomes too specialized in the training data and may not generalize well to unseen data. Therefore, it's important to monitor both training and validation accuracy and employ techniques such as early stopping to prevent overfitting.

In summary, the number of epochs used for training depends on the specific model and dataset. Still, based on the training accuracy changes over epochs and the possibility of overfitting, it's important to monitor the model's performance and apply appropriate techniques to optimize the number of epochs.

(iii)

	Test Accuracy
First CNN model	0.65
Second CNN model	0.67
Fine-tuned model	0.78



Based on the test accuracy results, the fine-tuned model outperformed the first and second CNN models. This is expected as fine-tuning a pre-trained model such as VGG16 can lead to better performance than training a CNN from scratch.



The first CNN model has the lowest test accuracy of the three models. This may be because it was not trained for a sufficient number of epochs or had a suboptimal architecture.



The second CNN model has a slightly higher test accuracy compared to the first CNN model, but lower than the fine-tuned model. It is possible that the architecture of the second CNN model is better suited for the sea animals dataset than the first model, but still not as good as VGG16 when fine-tuned.



Overall, the results show that using a pre-trained model like VGG16 and fine-tuning it can lead to significant improvements in accuracy compared to training a CNN from scratch.



(iii)

Second Model	
	<pre>[[0. 0. 0. 0. 1.]]</pre>
	<pre>[[0.999 0.001 0. 0. 0.]]</pre>

 <p>A photograph of a white seal pup lying on a snowy surface. The pup has dark eyes and a dark nose. The image is framed with a vertical axis on the left (0 to 80) and a horizontal axis at the bottom (0 to 80).</p>	<pre>[[0.001 0. 0.068 0.93 0.001]]</pre>
 <p>A photograph of a dark seal pup lying on a wet, sandy beach. The pup has a dark body and a lighter face. The image is framed with a vertical axis on the left (0 to 80) and a horizontal axis at the bottom (0 to 80).</p>	<pre>[[0. 0. 0.035 0.964 0.001]]</pre>

	<pre>[[0. 1. 0. 0. 0.]]</pre>
	<pre>[[0.024 0.776 0. 0.197 0.003]]</pre>

 <p>A cuttlefish is shown swimming over a sandy ocean floor. The cuttlefish has a light-colored, mottled body with some darker spots. Its tentacles are visible, and it appears to be moving towards the right. The background is a textured, light-colored sand or seabed.</p>	<pre>[[0.991 0.009 0. 0. 0.]]</pre>
 <p>A shark is shown swimming in clear blue water. The shark is facing the camera, with its mouth open, revealing sharp teeth. Its fins are visible, and the water is a deep blue color.</p>	<pre>[[0. 0. 0. 0. 1.]]</pre>

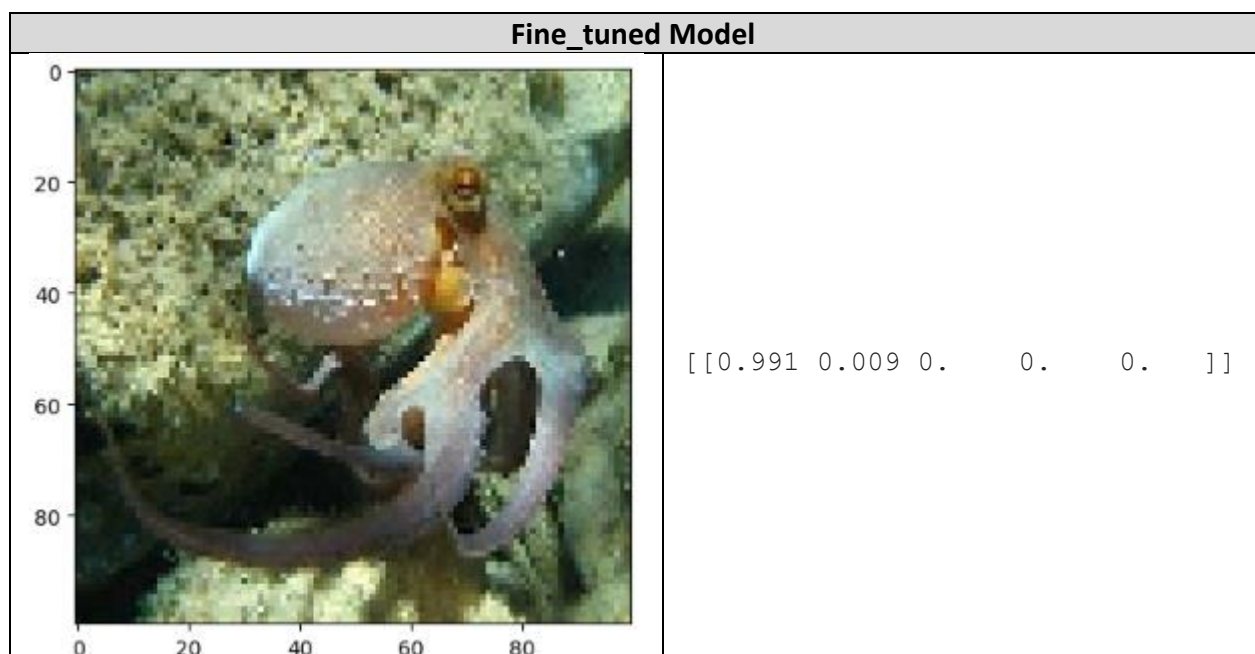
	<pre>[[0. 0. 1. 0. 0.]]</pre>
	<pre>[[0. 0. 1. 0. 0.]]</pre>



Based on the results, it seems like the second model correctly identified the most images of the animals I used for testing. The output of the model consists of a probability distribution across the five different animal categories, and in each case, the highest probability corresponds to the actual animal in the image.



Overall, the model appears to have performed well on this limited set of test images. However, it is important to note that the performance of the model may vary depending on the quality and diversity of the images used for training and testing. It is also possible that the model may perform differently when tested on a larger or more varied set of images.

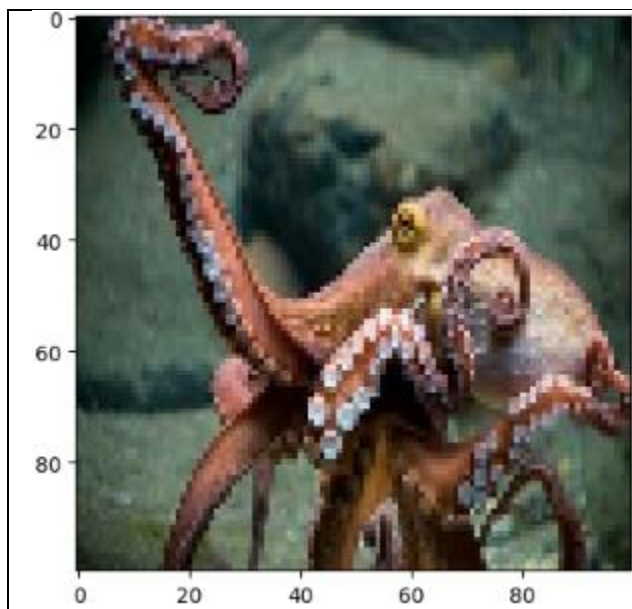
Therefore, further evaluation and testing of the model may be necessary to fully assess its performance and suitability for practical applications.

- ❖ Note: The results show that the model was particularly good at identifying sharks and otters, as these images received a score of 1 for the corresponding class. It's worth noting that there were a few instances where the model was less certain in its predictions, particularly for the images of eels and octopuses. Similarly, in one of the images of octopus, the model predicted a relatively low likelihood of the image belonging to the correct class.

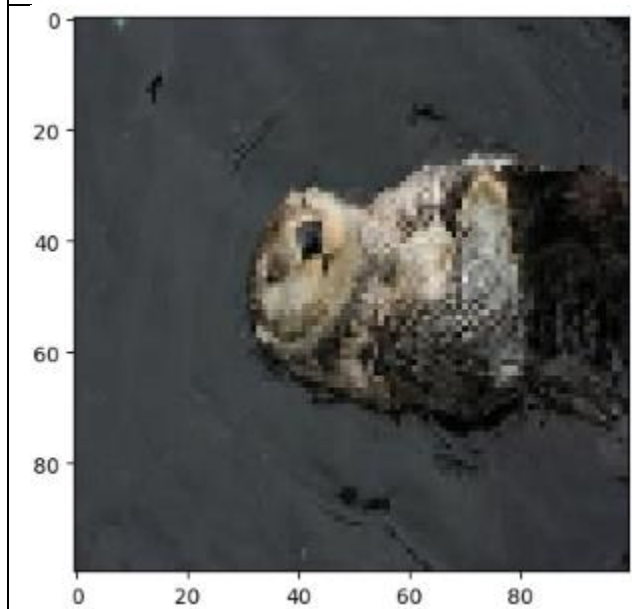


	<pre>[[0.999 0.001 0. 0. 0.]]</pre>
	<pre>[[0. 0. 0. 0. 1.]]</pre>



 <p>A photograph of a dark-colored seal, possibly a Mediterranean monk seal, resting on a sandy beach. The seal is facing the camera with its head slightly raised. The background shows some wet sand and a small puddle.</p>	<pre>[[0. 0. 0.035 0.964 0.001]]</pre>
 <p>A photograph of a white seal pup, likely a ringed seal, resting on a snowy or icy surface. The pup is facing the camera with its head down. The background is a bright, snowy landscape.</p>	<pre>[[0.001 0. 0.068 0.93 0.001]]</pre>

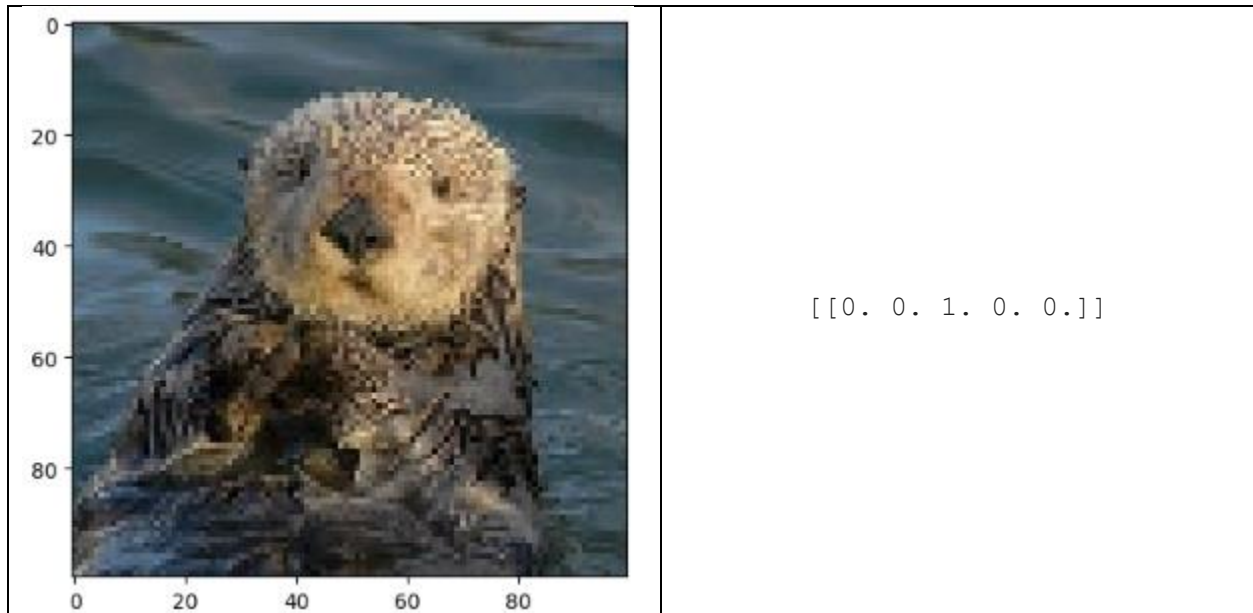


`[[0. 1. 0. 0. 0.]]`



`[[0. 0. 1. 0. 0.]]`

	<pre>[[0. 0. 0. 0. 1.]]</pre>
	<pre>[[0.024 0.776 0. 0.197 0.003]]</pre>



Based on the results, the fine-tuned model performed relatively well in recognizing the different sea animals. The model correctly identified the images of sharks with high confidence, which is a good indication that the model is good at recognizing sharks and otters. The model also correctly identified the images of seals with a high level of confidence, which suggests that the model is reasonably good at recognizing these animals as well.

However, the model did make some mistakes, particularly in recognizing the images of octopus and eel, where the model misclassified the images. This could be due to the fact that these animals have a more complex and unique shape, which may make them harder to recognize compared to other animals like seals and sharks.

Overall, the fine-tuned model showed promising results in recognizing the different sea animals, but there is still room for improvement, particularly in recognizing more complex animals.