**Fisheries Localization & Classification**

Collector: 1305037

Coder: 1305089

Trainer: 1305034

Writer: 1305098

Leader: 1305007

**Project overview**

Nearly half of the world depends on seafood for their main source of protein. In the Western and Central Pacific, where 60% of the world’s tuna is caught, illegal, unreported, and unregulated fishing practices are threatening marine ecosystems, global seafood supplies and local livelihoods. [The Nature Conservancy](http://www.conserveca.org/tuna) is working with local, regional and global partners to preserve this fishery for the future.

Currently, the Conservancy is looking to the future by using cameras to dramatically scale the monitoring of fishing activities to fill critical science and compliance monitoring data gaps. Although these electronic monitoring systems work well and are ready for wider deployment, the amount of raw data produced is cumbersome and expensive to process manually.

We have to develop algorithms to automatically detect and classify species of tunas, sharks and more that fishing boats catch, which will accelerate the video review process. Faster review and more reliable data will enable countries to reallocate human capital to management and enforcement activities which will have a positive impact on conservation and our planet.

Machine learning has the ability to transform what we know about our oceans and how we manage them.

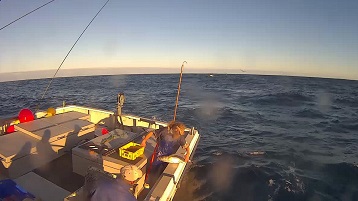


Figure: Images from training set

**Problem statement**

The main task is to localize and detect which species of fish appears on a fishing boat, based on images captured from boat cameras of various angles.

The goal is to predict the likelihood that a fish is from a certain class from the provided classes, thus making it a multiclass classification problem in machine learning terms.

Eight target classes are provided in this dataset : Albacore tuna, Bigeye tuna, Yellowfin tuna, Mahi Mahi, Opah, Sharks, Other (meaning that there are fish present but not in the above categories), and No Fish (meaning that no fish is in the picture).

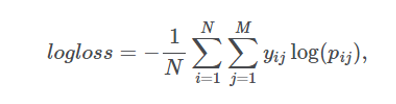
The goal is to train a CNN that would be able to classify fishes into these eight classes.

**Performance metrics**

The metric used for this Kaggle competition is multiclass logarithmic loss (also known as

categorical cross entropy).

In multi-class classification (M>2), we take the sum of log loss values for each class prediction in the observation.



Here each image has been labeled with one true class and for each image a set of predicted

probabilities should be submitted. N is the number of images in the test set, M is the number of

image class labels,log is the natural logarithm, yij is 1 if observation belongs to class and 0

otherwise, and pij is the predicted probability that observation belongs to class .



Further we have also measured accuracy as a metric as we had to output accuracy of the model as required.

**Data description**

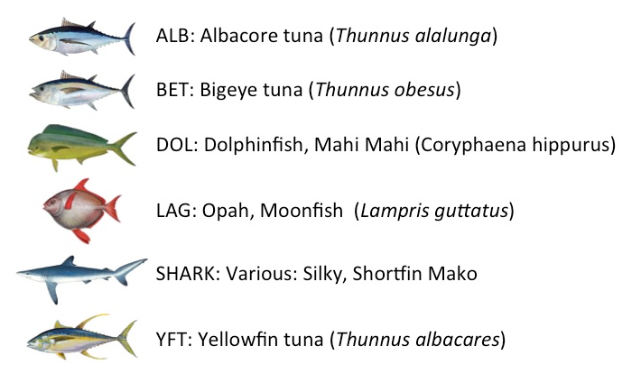
We are using the data set of “The Nature Conservancy Fisheries Monitoring” for this project.

Data Set link: <https://www.kaggle.com/c/the-nature-conservancy-fisheries-monitoring/data>

Eight target categories are available in this dataset:

1. Albacore tuna
2. Bigeye tuna
3. Yellowfin tuna
4. Mahi Mahi,
5. Opah
6. Sharks
7. Other (meaning that there are fish present but not in the above categories)
8. No Fish (meaning that no fish is in the picture).

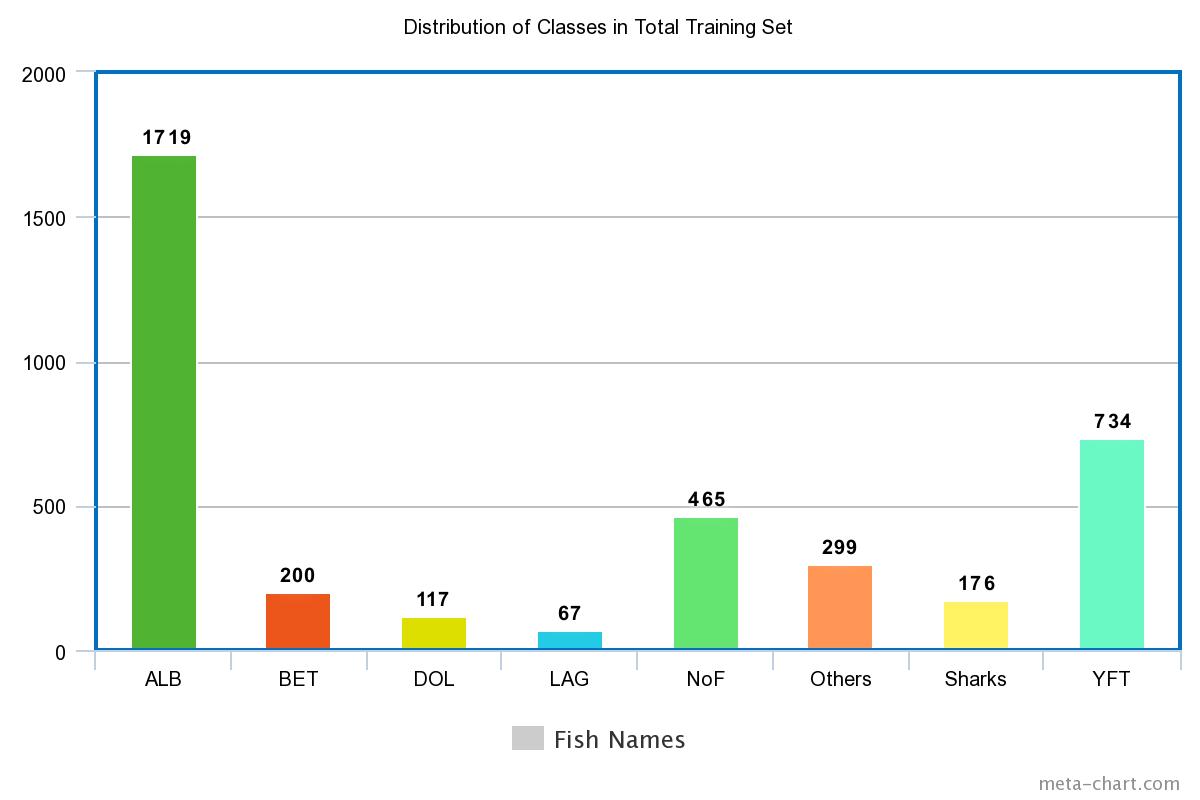
Each image has only one fish category, except that there are sometimes very small fish in the pictures that are used as bait. The train data is labeled according to fish species labels. We have to predict labels of test data.



There are 3777 images in the training set.

Among those,

1. 1719 images of ALB
2. 200 images of BET
3. 117 images of DOL
4. 67 images of LAG
5. 465 images of NoF
6. 299 images of Others
7. 176 images of Sharks
8. 734 images of YFT



**Data preprocessing**

1. Fish Localization:

We resize training files to 145x145x3. Training data was splited to train : validation : test data in the proportion of 80:10:10. Then we apply various augmentations on the images such as rotation, width shift, height shift, shear, zoom, horizontal flip, rescale. Moreover, we shuffle images in the respective directories.

1. Fish Classification:

All the images are resized to 145x145x3 as inceptionv3 model takes this specific sized files generally.

The localization part gives output a mask that is then ‘AND’ed with the image to find the image of fish only in the real image. This masked image is then send to InceptionV3 to classify the fish found in the image.

The images below are a real image, its mask, and the masked image.



Figure: Real Image



Figure: Mask



Figure: Masked image

**Model description**

1. Fish Localization:

We used a vgg16 model to localize fish in the image. This model was created by  VGG (Visual Geometry Group, University of Oxford) for the ILSVRC-2014.

1. Fish Classification:

We used inceptionV3 model to categorize the fish localized in the image.

**Architecture description**

1. Vgg16 architecture:

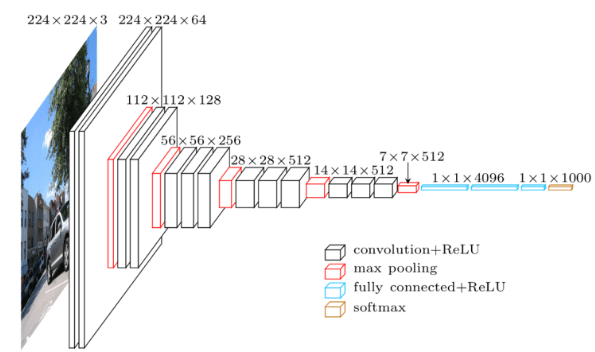


Figure: vgg16 internal architecture

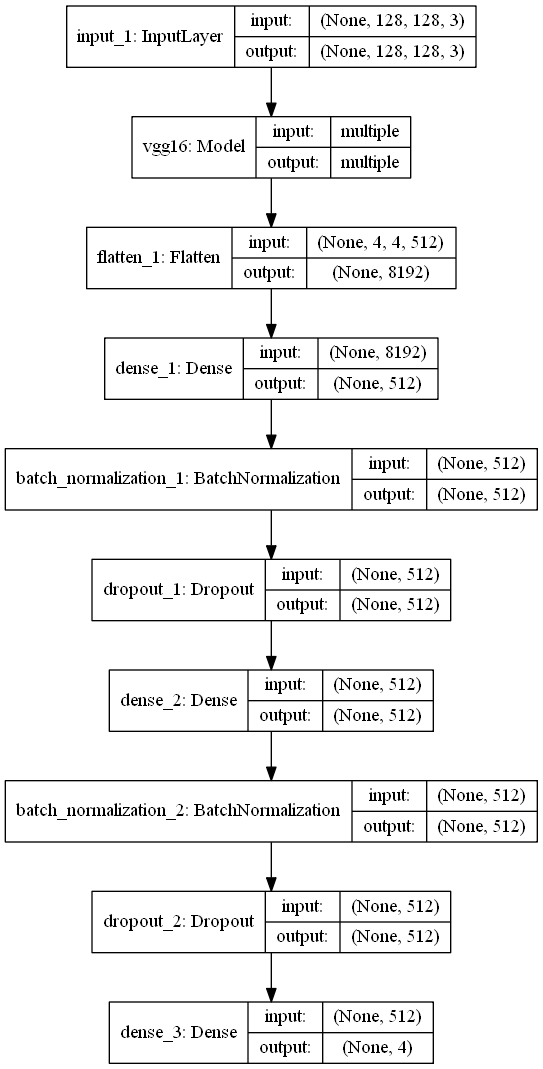
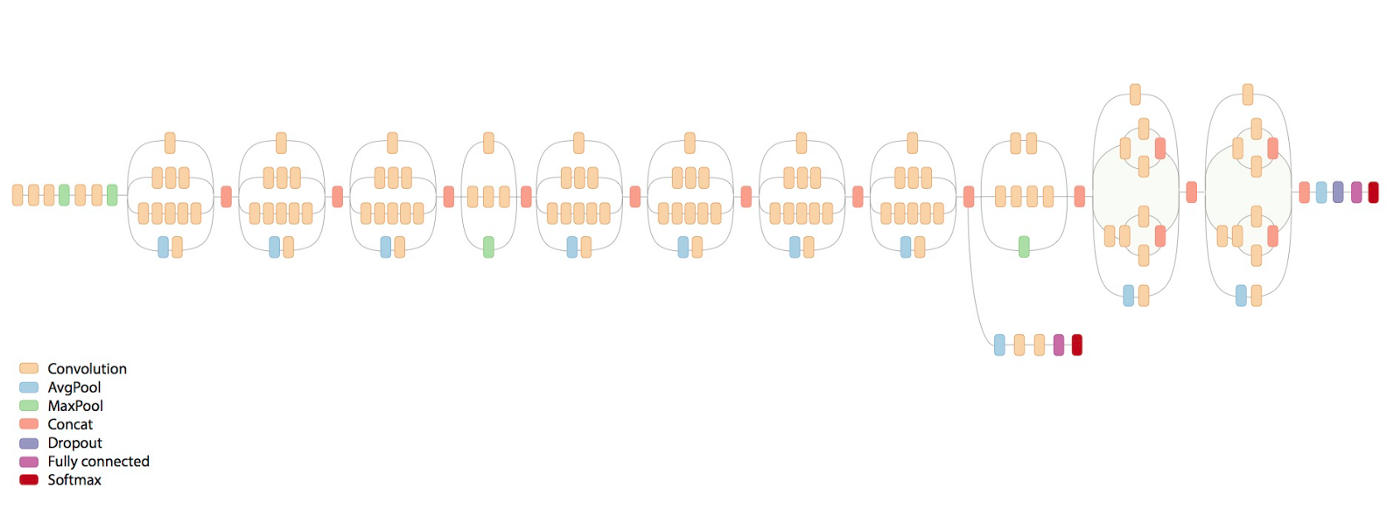


Figure: Our model architecture

2)InceptionV3 architecture:



As, we have used topless inceptionV3 model, we added few layers at the top:

1. AveragePooling2D layer (1)
2. Flatten layer (1)
3. Dense layer (1)

**Algorithm and techniques**

**VGG(16) Architecture**

The VGG architecture is composed entirely of 3x3 convolutional and maxpooling

layers, with a fully connected block at the end. The pretrained model is available in Caffe, Torch,

Keras, Tensorflow and many other popular DL libraries for public use.

Layers :

1. Convolution :

Convolutional layers convolve around the image to detect edges, lines, blobs of

colors and other visual elements. Convolutional layers hyperparameters are the number of

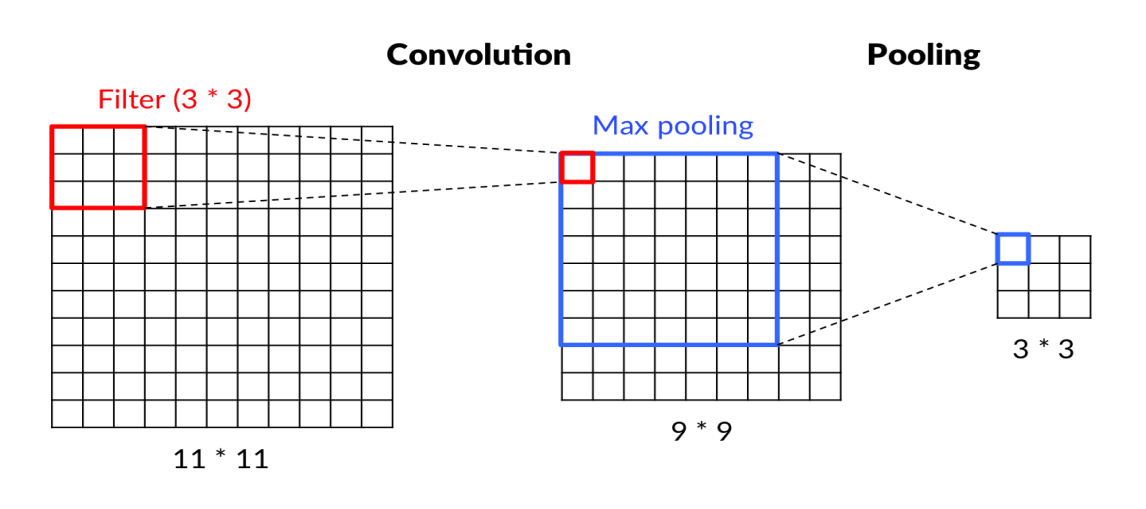
filters, filter size, stride, padding and activation functions for introducing nonlinearity.

1. MaxPooling :

Pooling layers reduces the dimensionality of the images by removing some of

the pixels from the image. Maxpooling replaces a n x n area of an image with the maximum

pixel value from that area to downsample the image.



1. Dropout :

Dropout is a simple and effective technique to prevent the neural network from

overfitting during the training. Dropout is implemented by only keeping a neuron active with

some probability p and setting it to 0 otherwise. This forces the network to not learn redundant

information.

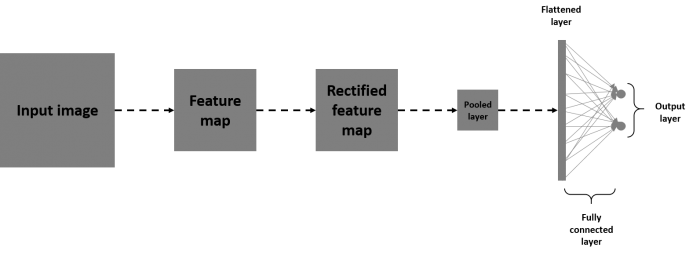
1. Flatten :

Flattens the output of the convolution layers to feed into the Dense layers.

1. Dense (Fully connected) :

Dense layers are the traditional fully connected networks that maps the scores of the

convolutional layers into the correct labels with some activation function(softmax used here).



Activation functions :

Activation layers apply a nonlinear operation to the output of the other layers such as convolutional layers or dense layers.

1. ReLu Activation :

ReLu or Rectified Linear Unit computes the function f(x)=max(0,x) to threshold the activation at 0.

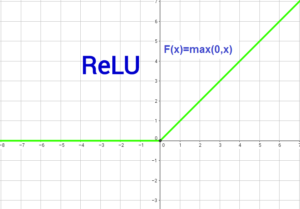


Figure: Relu function

1. Softmax Activation :

Softmax function is applied to the output layer to convert the scores into probabilities that

sum to 1.

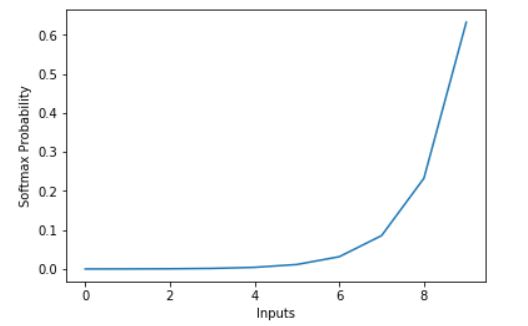
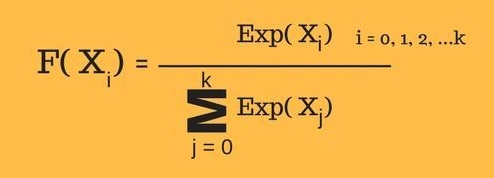


Figure: Softmax function

Optimizers :

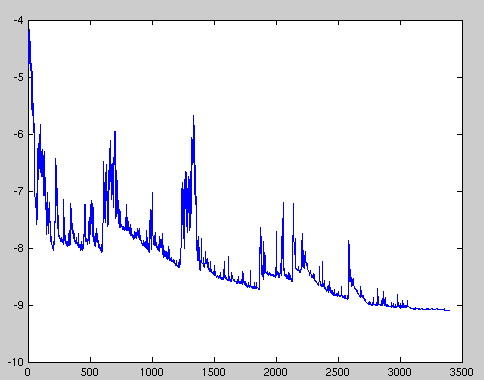
1. SGD:

Stochastic gradient descent (SGD) in contrast performs a parameter update for each training example x(i) and label y(i):

θ = θ − η.∇J(θ;x(i);y(i))

Batch gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. SGD does away with this redundancy by performing one update at a time. It is therefore usually much faster and can also be used to learn online.

SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily as in image below.

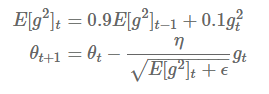


While batch gradient descent converges to the minimum of the basin the parameters are placed in, SGD's fluctuation, on the one hand, enables it to jump to new and potentially better local minima. On the other hand, this ultimately complicates convergence to the exact minimum, as SGD will keep overshooting. However, it has been shown that when we slowly decrease the learning rate, SGD shows the same convergence behaviour as batch gradient descent, almost certainly converging to a local or the global minimum for non-convex and convex optimization respectively.

1. RMSProp:

RMSprop is an unpublished, adaptive learning rate method proposed by Geoff Hinton in [Lecture 6e of his Coursera Class](http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf).

RMSprop and Adadelta have both been developed independently around the same time stemming from the need to resolve Adagrad's radically diminishing learning rates. RMSprop in fact is identical to the first update vector of Adadelta:



Here decaying average over past squared gradients is E[g2]t . We use gt to denote the gradient at time step t.

RMSprop as well divides the learning rate by an exponentially decaying average of squared gradients. Hinton suggests γ to be set to 0.9, while a good default value for the learning rate η is 0.001.

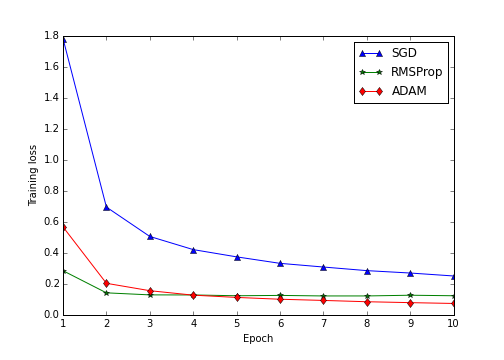
3.Adam :

Adam (Adaptive moment estimation) is an update to RMSProp optimizer in which the

running average of both the gradients and their magnitude is used. In practice Adam is

currently recommended as the default algorithm to use, and often works slightly better than

RMSProp. Adam also shows general high accuracy while adadelta learns too fast. We have used Adam in the experiment because we felt having similar optimizer would be a better baseline for comparing the experiments.



Data Augmentation :

Data augmentation is a regularization technique where we produce more

images from the training data provided with random jitter, crop, rotate, reflect, scaling etc to change the pixels while keeping the labels intact. CNNs generally perform better with more data as it prevents overfitting.

Batch Normalization :

Batch Normalization is a recently developed technique by Ioffe and Szegedy which tries to properly initializing neural networks by explicitly forcing the activations throughout a network to take on a unit gaussian distribution at the beginning of the training. In practice we put the Batchnorm layers right after Dense or convolutional layers. Networks that use Batch Normalization are significantly more robust to bad initialization. Because normalization greatly reduces the ability of a small number of outlying inputs to over influence

the training, it also tends to reduce overfitting. Additionally, batch normalization can be interpreted as doing preprocessing at every layer of the network, but integrated into the network itself.

We have batch size of 32 in our project.

**InceptionV3 architecture:**

Layers:

Convolution, MaxPooling, Dense, Flatten, Dropout layers are same as described earlier. Here 2 more types of layers are used also. Those are described below:

AvgPooling:

Pooling layers reduces the dimensionality of the images by removing some of

the pixels from the image. Avgpooling replaces a n x n area of an image with the average

pixel value from that area to downsample the image.

Concat:

The concatenation of the inputs (at least 2 tensor) alongside provided axis.

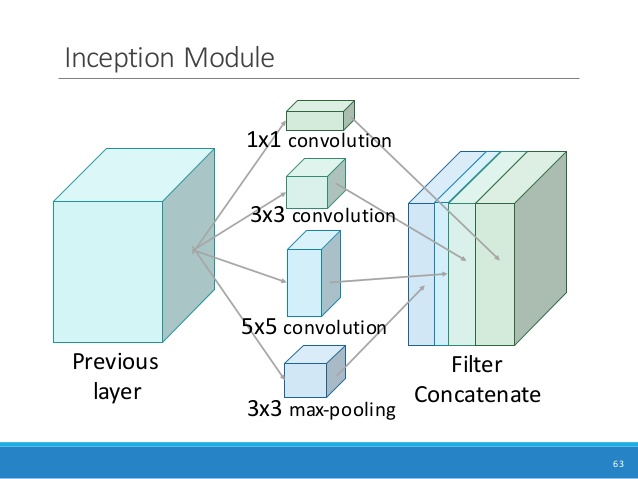


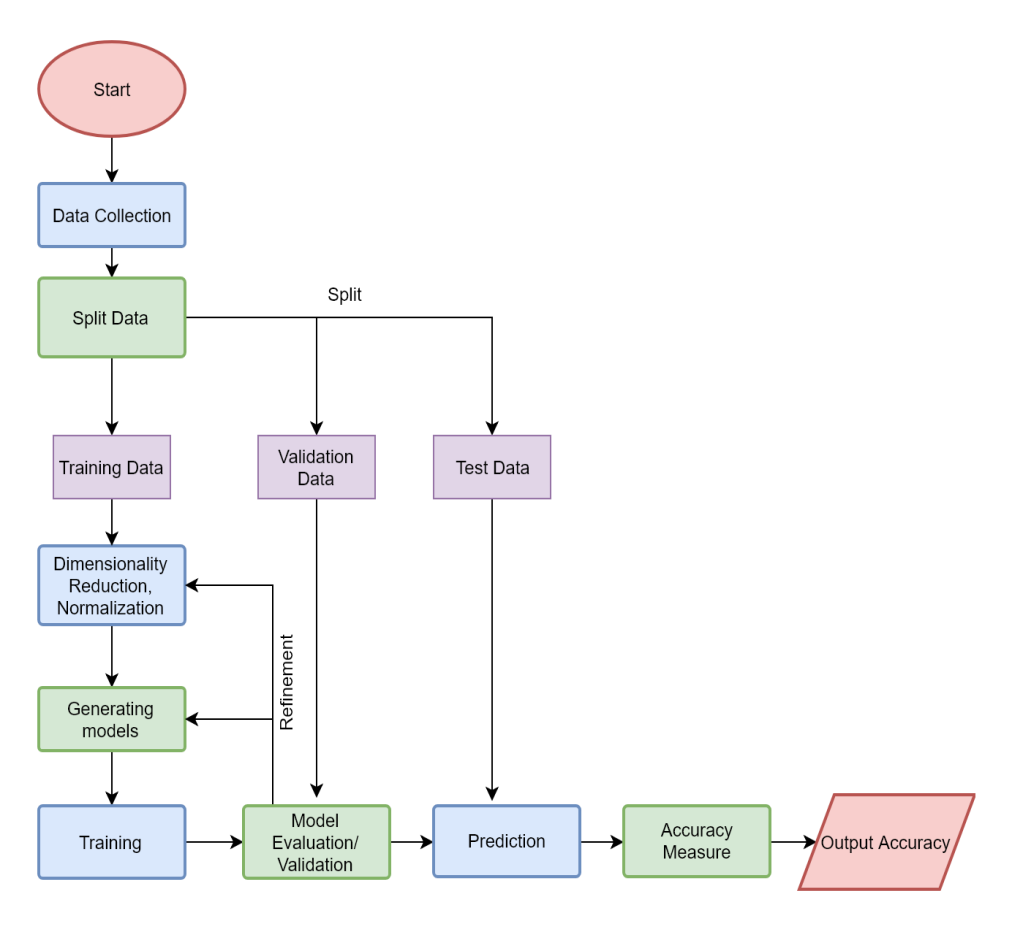
Figure: Concat layer

Optimizer:

Same as those used in the previous model.

Data augmentation and activation functions are same as described earlier.

**Workflow:**



**Hyperparameter tuning**

Hyperparameters we have tuned are:

1. Optimizer: Adam, RMSProp, SGD
2. Learning Rate:0.001, 0.01,0.1

**For Model1 (Fish Localization):**

|  |  |  |  |
| --- | --- | --- | --- |
| Optimizer | Learning Rate | Drop Ratio | Intersection over Union (IoU) |
| RMSProp | 0.001 | 0.5 | 0.05916595828038941 |
| RMSProp | 0.001 | 0.7 | 0.06697892062915273 |
| RMSProp | 0.001 | 0.8 | 0.027552860364718882 |
| SGD | 0.001 | 0.5 | 0.04918212063674736 |
| SGD | 0.001 | 0.7 | 0.03084469545491272 |
| SGD | 0.001 | 0.8 | 0.04635634851225526 |
| ADAM | 0.001 | 0.5 | 0.009074315957180164 |
| ADAM | 0.001 | 0.7 | 0.013103915474366392 |
| ADAM | 0.001 | 0.8 | 0.009850858354809816 |
| RMSProp | 0.01 | 0.5 | 0.017100379249480338 |
| RMSProp | 0.01 | 0.7 | 0.007527862805291865 |
| RMSProp | 0.01 | 0.8 | 0.018675593822314058 |
| SGD | 0.01 | 0.5 | 0.05836305138366047 |
| SGD | 0.01 | 0.7 | 0.005256330440361491 |
| SGD | 0.01 | 0.8 | 0.09444243644789639 |
| ADAM | 0.01 | 0.5 | 0.006079822701755217 |
| ADAM | 0.01 | 0.7 | 0.04960293784174007 |
| ADAM | 0.01 | 0.8 | 0.00695820973055734 |
| RMSProp | 0.1 | 0.5 | 0.01226741332893041 |
| RMSProp | 0.1 | 0.7 | 0.0062498116810106264 |
| RMSProp | 0.1 | 0.8 | 0.006043347785166556 |
| SGD | 0.1 | 0.5 |  |
| SGD | 0.1 | 0.7 |  |
| SGD | 0.1 | 0.8 |  |
| ADAM | 0.1 | 0.5 |  |
| ADAM | 0.1 | 0.7 |  |
| ADAM | 0.1 | 0.8 |  |

**ForModel2 (Fish Classification):**

|  |  |  |
| --- | --- | --- |
| **Optimizer** | **Learning Rate** | **Log Loss** |
| RMSProp | 0.001 | 2.461715246314433 |
| SGD | 0.001 | 3.595411736507719 |
| Adam | 0.001 | 2.676518795239816 |
| RMSProp | 0.01 | 6.10315287503572 |
| SGD | 0.01 | 3.2645895830286085 |
| Adam | 0.01 | 2.70410889689602 |
| RMSProp | 0.1 | 10.522447322125708 |
| SGD | 0.1 | 3.0928670043563227 |
| Adam | 0.1 | 2.98967542104468 |

The result of hyperparameter tuning is separately given in tuning\_results.txt and Result.docx.

**Result description**

**Intersection Over Union(IoU)**  (for model 1)

0.11

**Log loss:** (for model 2)

2.5134533

The result is not as satisfactory as expected because

* Our devices do not have GPU, so we couldn’t run enough epochs to reach minima.
* Images are large in size, take a lot of time to run even small number of epochs.
* We had to train with a very small number of training data as it takes huge amount of time to train large dataset.
* For SGD optimizer and learning rate 0.1, model did not converge.

**Conclusion**

We hope and believe that we will be able to get better accuracy if we use tensorflow object detection api such as yolo, ssd etc to detect fish images. We will further carry out this project to get satisfactory result. If we could run the project with bigger dataset and more epochs the result would be much better.