Investigation Of Deep Batch Active Learning Methods For Marine Species Detection

Master-Thesis

At University of Applied Science Kiel
Department of Computer Science and Electrical Engineering

Degree Program: Master in Information Engineering

Submitted by: Ali Farooq

Matriculation No: 932225

Supervisor: Prof. Dr. Hauke Schramm

First examiner: Prof. Dr. Hauke Schramm

Second examiner: M.Sc. Gordon Böer

Submitted on: 15/08/2022

Abstract

Active learning has been on the discussion in last few years because of its ability to reduce annotation labor and training data. The aim of active learning is to put more information into a training set. There are many techniques to perform active learning which selects either uncertain, diverse or both samples. Active leaning is also suitable to identify outliers in the data. Some techniques work with single sample selection per iteration and some can select batch of images on each iteration.

Two techniques, namely, Diverse Mini-batch active learning and Active learning with gradient lower bound were picked for this experiment because they can select batch of samples for deep learning training. The techniques work with both uncertain and diverse in addition to outlier samples selection. The techniques were tested on pre-trained models InceptionV3 and Yolov5 with pre-existed and custom datasets. The datasets used for pre-existed were cats and dogs. Custom data which were used is underwater grey-scale marine species images.

The challenge for this experiment was to apply active learning methods on Yolov5. Yolov5 performance was not satisfactory because of the grey scale images and unavailability of bounding box features. The second method also did not work for Yolov5 because of missing loss values for each image to calculate lower bound. Therefore, the experiment was conducted with InceptionV3 model. Method Diverse mini-batch active learning works significantly better with InceptionV3 model and with the bounding box images extracted from Yolov5. The accuracy of 97.5% was achieved with a smaller number of training images.

Fachhochschule Kiel
University of Applied Sciences

Declaration

I, Ali Farooq hereby declare that the thesis titled as “Investigation of Deep Batch Active Learning Methods for Marine Species Detection” is an original research work conducted by me. All the resources which were utilized during this research have been acknowledged in the text and the list of references has been provided.

Kiel, 15.08.2022

Ali Farooq	Place, Date

Acknowledgements

I would like to give my gratitude to my thesis supervisor, Prof. Dr. Hauke Schramm for his valuable guidance, support, and advice throughout my academic career. I would also like to thank Mr. Gordon Böer for his counseling, support and inspiration to keep me motivated throughout my thesis. Moreover, his expertise and knowledge in the Marine Species dataset was applicable in caring out my research work.

I would also like to express sincere regards to my family and friends for their moral support and encouragement.

Table of Contents

Introduction

Data annotation for training a supervised machine learning algorithms has always been an important and tricky part. Data must be equally distributed and should have enough information about all possible categories. Supervised machine leaning algorithms always need properly annotated data samples and an algorithm which can learn from those samples. To fulfil this requirement mostly we need thousands of data samples which are annotated. Usually, to annotate that big amount of data is time taking and to get meaningful annotation is difficult. This process is computationally expensive and time consuming because of a big training data. Training data may also contain hundreds of similar data samples which may not help a model to achieve good results and can also contain some outliers and non-relevant data. The question here is how we can improve annotation and selection process of high-quality data which helps a model to learn quickly and efficiently.

Active learning is an emerging technique for data annotation and selection, but it is still in its progressive state. This technique is used to select data for annotation based on some parameters. A good active learning algorithm selects those training data for annotation which helps the model to get better rather than giving a big amount of data sample for training. For example, if we have a constraint to select 100 data samples for annotation. So, active learning main goal is to select best 100 images from a dataset which can present the dataset more precisely.

The main goal which will address the field of marine species detection during this thesis is to investigate deep batch active learning methods on underwater species images. The purpose of this investigation is to speed up the process of training by choosing annotated images which can produce good model accuracy. There are a large number of recorded under water videos available for this experiment. Training and annotation have to be done on frames, therefore, it is better to analyze if active learning can help to choose the most helpful frames from thousands of frames. This whole active learning process of selection will help to detect and classify marine species from those underwater images. InceptionV3 model is used for initial experiments to test active learning methods. YOLO algorithm is used for object classification and detection. Active learning methods were tested on those detected objects. YOLO is a most famous algorithm for real time object detection.

Related work is discussed in chapter 2. The background and definition related to the techniques of active learning are introduced in chapter 3. In chapter 4 there is a glimpse of all algorithms and models which are essential and helpful for this experiment. The initial experiment and the experiment which was conducted on marine species data are explained in chapter 5. The results of those experiments are discussed in detail in Chapter 6. Summary and future work is discussed in chapter 7.

Related Work

To achieve a goal of active learning by reducing the effort of data annotation, many techniques with machine learning and deep learning algorithms has been established in the past and still efforts are made to make current methods better or invent new ones. The purpose of using an active learning on dataset is to reduce the amount of annotated data which was previously most time-consuming task for supervised machine learning algorithms. Through active learning techniques, model try to predict the level of uncertainty or diversity in the dataset samples. It helps the model to train on more informative samples rather than similar samples.

The basic research on deep active learning was carried out with Bayesian method. Yarin Gal researched on deep Bayesian active learning to incorporate active learning with large high dimensional data. The method was tested on MNIST and skin cancer images and it produced significant results (Yarin Gal, 2017). Ozan Sener investigated active learning on convolutional neural networks (CNNs) because huge amount of labelled data is required to train a CNNs model and based on the existing single sample selection methods of active learning it is hard to apply active learning on it. Therefore, new technique core-set approach of active learning was introduced to choose a set of samples based on batch learning. The proposed method performs significantly well on existing datasets and outperforms the existing methods of image classification (Ozan Sener, 2018).

William H. Beluch researched on ensembles for active learning. CNNs with Monte Carlo dropout was compared with ensemble methods and it performs better and predict more accurate uncertain samples from data. It achieves 90% accuracy for MNIST and CIFAR-10 datasets (William H. Beluch, 2018).

Donggeun Yoo examined on active learning new technique using learning loss to detect uncertain samples from the data. The method learns to predict target losses of unlabeled data. The method was tested on different image classification and object detection problems, and it outperforms every time (Donggeun Yoo, 2019). Jordan T. Ash researched on gradient lower bound embedding to perform active learning on batches. This method incorporates both uncertain and diverse samples from a batch. The method performs well compared to other methods and can qualify as a versatile approach for practical active learning problems (Jordan T. Ash, 2020).

Beichen Zhang investigated on state relabeling adversarial active learning to select the most informative samples from the unlabeled pool for labelling. The method used generative and discriminative models to detect uncertain samples (Beichen Zhang, 2020). Recently, in 2022 (Jifan Zhang, 2022) researched on graph based active learning to adapt class balanced samples into a training dataset. It helps to gather uncertain samples accordance to class to make a balanced dataset.

Background

Obtaining labelled data has always been a big challenge in machine learning applications. The process of data labeling can be long and costly. Especially in supervised machine learning algorithms data plays an important part because model learns only from data, and it must be annotated properly. Data annotation usually refers to organizing data in such a way that each sample belongs to a correct class so a model can learn features according to its related class. This issue has always been on discussion to find solutions to reduce data label cost. Therefore, many solutions exist which can help in different provided scenarios to automatically identify which data is going to be labelled.

Active Learning

Active learning is a framework which can help a system to learn on fewer data and to identify data which the user must label. Those techniques make machine learning applications more accurate and applicable because models can produce good results with less amount of data and can easily be applied on any kind of data. The purpose of active learning is to produce higher accuracy on a learning model with less amount of data. A good active learning technique is able to detect samples for annotation which helps a model to train better and to be generalized.

The common starting scenario of active learning starts with having a big pool of unlabeled samples. On those unlabeled samples an algorithm will run which ranks them and the sample with highest rank score will be chosen to label for training dataset. The selected samples will then be annotated and used to train a model. This process is repeated on all samples from unlabeled data until it reaches the required number of training samples. For instance, there are 5000 samples in our population of data from which only 100 samples are required for training seed and 100 extra samples for active learning training. The process will first train a model on 100 samples seed. After that the model will run and train 100 times to select 100 training samples for active learning process. In the active learning case usually, the model trained from scratch but pre trained models can also be used for this process. (Gildenblat, 2020)

The classic approach to select a sample would be to select those samples about which model has wrong prediction. The problem in this approach is that we do not know the actual annotations for those wrongly classified samples. Therefore, there are mostly two main approaches used to identify which samples from unlabeled pool will chose in active learning process. (Jingya Shao, 2019)

Uncertain Samples

Choose samples about which the model is not certain because the model is wrong about that sample. (Jingya Shao, 2019)

Diverse Samples

Choose samples which represent diverse information which the model does not know about. (Jingya Shao, 2019)

Acquisition Function

The selection of uncertain samples from the model is based on a ranking function also called acquisition function. There are many acquisition functions which can help to find uncertain samples such as:

Entropy

The idea behind this function is to select a sample which has a bigger entropy score. The entropy score varies in between 0 and 1. The entropy score can be greater than 1 if number of classes are more than 2. If the probability score of all classes for a sample is high, then that sample has high entropy score because the model is confused about this sample. The sample will qualify for annotation. On the other hand, it will decrease if only one class has higher probability then others. It will show that the model is confident about one class for that the sample will not qualify for annotation. (T, 2019)

$$H(p) = -\Sigma\, p_i log_2(p_i) \quad (1)$$

In equation 1 p_i is a probability of each class i and it take sum of all probabilities to calculate the score.

Variation Ratio

This function selects a sample based on a high variation ratio score. Larger VR shows differentiation or dispersion between classes smaller VR shows similarity and concentration in between classes. (Yarin Gal, 2017)

$$VR[x] = 1 - max(p) \quad (2)$$

In equation 2 p represents the probability of classes and max function will choose one of a class or category with higher probability.

Mean Standard Deviation

This function tries to maximize the mean standard deviation of a sample. It maximizes the value by taking the average of all classes to whom the sample belongs. It's the most used technique now a days to choose uncertain samples. (Yarin Gal, 2017)

$$\sigma(x) = \frac{1}{C} \sum_c \sigma_c \quad (3)$$

In equation 3 x represent a sample and C represents class outputs for the sample.

Random

The random acquisition function returns a value in interval [0, 1] drawn from a uniform distribution. This function is the same as choosing a random sample from an unlabeled pool. The sample will be chosen based on the random uniform distribution value. (Yarin Gal, 2017)

$$a(x) = unif() \quad (4)$$

In equation 4 x is a sample and unif () is a uniform distribution.

Difference Between Outputs

This technique depends on the model outputs. It takes a difference of the first two largest outputs. If the difference is larger, then the model output is more concentrated on first category. On the other hand, if the difference is smaller than model is confused on categories. (Gildenblat, 2020)

$$a = max_1(p_i) - max_2(p_i) \quad (5)$$

In equation 5 p_i are the possible probability of category i.

In classic active learning techniques, the idea is to choose a sample based on multiple models called ensemble models rather than depending on one single model. Single model output may contain some uncertainty. Therefore, it is better to choose ensembles to reduce uncertainty. The decision to choose a sample can be made by ensembles output. If most of the models agree on a particular class, then the sample will not be selected for annotation otherwise if most models have different output classes for a sample than it would be helpful to put it in a training set. The sample will get selected for annotation. This whole process is called a query by committee where a sample is given to models to decide whether to choose it or not. (Gildenblat, 2020)

Techniques of Active Learning

There are many techniques to perform active learning on a dataset. Based on those techniques single or batch of samples can be chosen from the dataset for annotation. The above-mentioned acquisition functions selects only one best sample for each active learning iteration. Therefore, to combine active learning with deep learning models, batch based active learning is needed to speed up the process of deep learning model training. Active learning with deep learning is hard because deep neural networks are computationally high and for that batch active learning will help to achieve active learning (Gildenblat, 2020). In coming sections both single and batch sample selection methodologies will be explained.

Deep Bayesian Active Learning

Deep Bayesian method is a very important starting point for active learning. This method is used to select a single sample. Bayesian networks are good to provide a model uncertainty about a sample. In a Bayesian neural network, the network parameters are drawn from a distribution and ensemble models are used with infinite network settings to compute an output. Ensembles are used because the uncertainty of only one model may be wrong because it may be too confident or under confident about that sample. Therefore, many models are being used to select the correct sample. If more models agree on one sample output, then it produces high confidence about that sample. Otherwise, if models disagree on a sample, then it produces a high uncertainty. With this technique it is hard to incorporate all the infinite parameter values of the distribution, so to achieve that goal Monte Carlo dropout is used. (Ghahramani, 2016)

Monte Carlo Dropout

Bayesian neural network takes an input image x and to get the output for the category c, the network will go over all the possible weight configurations and takes probability for every weight configuration. The real parameter distribution is unknown, but it can be approximated by assuming they belong to the Bernoulli distribution. To simulate the Bernoulli distribution, Monte Carlo Dropout can be used. (Ghahramani, 2016)

The idea behind adding a Monte Carlo dropout into Bayesian network, is that every neuron of the network has a Bernoulli prior value which multiplies with the actual output of the neuron. The prior probability represents the belief about a situation without considering the actual evidence (Prior probability - Wikipedia, n.d.). To run the network with priors of each neuron in the network, with applying many dropouts

at test time and sum up the results is called a Monte Carlo Integration. (Ghahramani, 2016)

$$p(y = c|x) = \int p(y = c|x, w)p(w)dw \approx \int p(y = c|x, w)q*(w)dw \quad (6)$$

In equation 6 y is the output of the category c, x is the input and w is weight configuration. By adding dropout into an equation 6.

$$p(y = c|x) = \frac{1}{T}\sum_t p(y|x, w_t) \approx \frac{1}{T}\sum_t p_c^t \quad (7)$$

In equation 7 where T is the number of test runs. If T gets larger, the approximation of Monte Carlo Integration will get better. Running the model many times while applying dropout and take the average of the results will approximate the probability of each network category. (Gildenblat, 2020)

To calculate the uncertainty in samples, Monte Carlo dropout is inserted into the entropy acquisition function. By taking the average of multiple runs and the entropy of those results the equation will look like:

$$H \approx -\sum_c(\frac{1}{T}\sum_t p_c^t)Log_2(\frac{1}{T}\sum_t p_c^t) \quad (8)$$

If the entropy score is high, the models are uncertain about this sample category and therefor, the sample will be selected for annotation. On the other hand, if the models are confident about the sample category, then the sample will not qualify for tagging. There may be a situation where many models are not certain about a sample. E.g., the sample may contain unclear data to detect or there is a noise.

A possibility to deal with this kind of situation is the concept of Bayesian active learning by disagreement BALD. To select the uncertain sample, the goal is to find those samples where the model is confused about unannotated samples because the actual labels are missing. In a Bayesian network there are many models with infinite parameter settings therefore the goal is to find the sample where most of the models are not agreeing. (Gildenblat, 2020)

$$I(y; w|x, D_{train}) = H(y|x, D_{train}) - E_{p(w|D_{train})}[H(y|x, w, D_{train})] \quad (9)$$

In equation 9 the first term will look for the sample with high entropy score and the second term will find the sample where most of the models are not sure about. To integrate Monte Carlo dropout to approximate the output again the equation will change into:

$$I(y; w|x, D_{train}) \approx -\sum_c \left(\frac{1}{T}\sum_t p_c^t\right) log_2 \left(\frac{1}{T}\sum_t p_c^t\right) + \frac{1}{T}\sum_{t,c} p_c^t \, log_2 p_c^t \quad (10)$$

In the updated equation 10 the first part runs the model many times, averages the output and calculates the entropy. The second part of the equation runs the model many times, calculates and average the entropies.

The Bayesian model is good to select one sample per iteration but it's computationally expensive because there are infinite or many models with different parameter settings so it will take a long time to achieve the goal of selecting samples for annotation. At the same time this method is good to understand how active learning will work on a low level.

Learning Loss for Active Learning

The idea of this method is to predict the loss of the learning algorithm for a given input. If the loss is higher, it means the sample needs to be annotated because the model is not sure about this sample. If the loss is low, then the model will be sure about its prediction so the sample will not be chosen.

To predict the loss, mean square error is added into a final stage. The method has two parts, (1) to predict the target output for a sample and (2) to predict the loss for the output. First the method will predict the target output for the sample and calculate the loss with respect to its target ground truth. This target value will be used as a ground truth for calculating the loss in the loss prediction module. In the loss prediction module, the model will predict loss. The predicted loss and target output loss will used to calculate the final mean square error loss. If the loss is high the sample will select for annotation. (Donggeun Yoo, 2019)

During training, the loss will change a lot and it is difficult to learn therefore, the predicted loss will be compared in between samples. Like Batch of size B will split into two equal parts. For each part the loss will be predicted for sample. After that the loss for both divided parts will be compared with a hinge loss. If the loss is high it shows to choose the sample for annotation otherwise leave the samples. (Gildenblat, 2020)

This method is also suitable in terms of a single sample selection in each active learning iteration. The divided batch will also contain single sample and the loss will be compared in between two samples. So, it is not suitable for batch learning where there are bunch of samples in one batch. In coming sections, the batch base active learning methods will be explained.

Core-Set Approach

This method is used to select a batch of diverse samples from unlabeled samples. The idea behind this is to calculate the distance of an unlabeled sample to a training dataset. The distance was calculated using a distance formula. The Euclidean distance is usually used in case of images. It is applied on the vector extracted from the network and take square root of a distance between two points. If the distance is minimum then, the sample is correctly classified or if the distance is maximum then the sample must be an outlier or contain different information on which the model is not trained yet. The extracted vector can be an image feature vector, image gradient values etc. This method chose a group of samples which have highest distance from the training set on each active learning iteration. (Ozan Sener, 2018)

Batch BALD

This method selected both diverse and uncertain samples from unlabeled data. In single sample selection, disagreement bald with an acquisition function chooses a sample based on the model parameters and model output. Both must agree on the selection of a sample. (Andreas Kirsch J. v., 2019) To extend Bayesian disagreement bald equation 9 to choose batch of samples certain changes are required:

$$I(y_1,..y_B; w|x_1,..x_B, D_{train}) = H(y_1,..y_B|x_1,..x_B, D_{train}) - E_{p(w|D_{train})}[H(y_1,..y_B|x_1,..x_B, w, D_{train})] \quad (11)$$

The updated equation 11 will help to choose different samples rather than choosing similar samples for a batch. If two samples contain similar information one of them will not be selected for labelling. This method encourages more information and diversity into a training set. (Gildenblat, 2020)

Diverse Mini-Batch Active Learning

This method also encourages to choose uncertain and diverse images in the active learning process. To select diverse samples from unlabeled data, Kmeans clustering is used in which it clusters data into K clusters. In case of images, vector can be extracted from the last layer as an image features to perform Kmeans. The image will be selected based on the distance of the feature vector from the cluster center. If the distance is small, the image will not be selected for annotation otherwise if the distance is large or if sample is in the center of clusters, it will be selected because it shows diversity of sample. To select uncertain samples, uncertainty acquisition function will be used to calculate the score like entropy. This score will be used to select uncertain sample. Using this method, a batch of samples can be chosen from each active learning iteration. Those samples are both uncertain and diverse. (Zhdanov, 2019)

Deep Batch Gradient Lower Bound Active Learning

This method is similar to the above explained method but with small change. It also has the benefit to combine both diverse and uncertain samples. As an alternative to use the model output, compute gradient of the predicted category according to the model parameters of the last layer. Since there are many parameters, the gradient becomes a vector called gradient embedding. Using gradients for uncertainty is more understandable since they are used to train networks when gradient norms are large. To be more confident about the certain category, gradients need to change a lot. Initially, to calculate gradient lower bound original labels for the samples are needed but as a substitute consider that the predicted labels from the model are true labels and gradients will calculate according to it. These uncertain gradient embeddings also gave us diverse samples using Kmeans clustering. The distance between samples and cluster centers shows the diversity in samples. (Jordan T. Ash, 2020)

Algorithms and Models

Kmeans

Kmeans is an unsupervised clustering technique. It is an iterative process to produce clustering on data. At beginning it picks some arbitrary points as cluster centers. Assign each sample to its nearest centroid and take an average to adjust the centers of the centroid. This process is repeated until the centers are adjusted and are not changing their positions. The algorithm will converge and create clusters but on local minima or locally optimum solution not to a globally optimum. Therefore, restarting it several times can be useful (K-Means Clustering, n.d.). Kmeans is fast, simple and converges fast but on the other hand it may cause problems with the presence of outliers in data, centers initialization and specifying a value of K if the actual number of categories are missing (sklearn.cluster.KMeans, n.d.).

ImageNet

ImageNet is a dataset available to train models. It contains 14,197,122 annotated images (Papers with Code - ImageNet Dataset, n.d.). Many models are pre-trained on it to perform image classification and detection tasks. The pre-trained models are trained on ImageNet dataset, and they put a benchmark in image classification and detection field. The purpose of models is to correctly classify input images into different categories of objects. Those different categories represent classes that belongs to our daily life objects. Animals, vehicles, appliances and many more objects are available on ImageNet and models are trained on it. The pre-trained models are available on Keras core library. Using transfer learning, feature extraction or fine tuning those pre-trained models can be generalized on images outside from ImageNet dataset. There are many models such as ResNet, VGG, InceptionV3 or many more which are trained on ImageNet dataset. InceptionV3 is used to extract multilevel feature information from the image. It computes three different convolutions with the dimensions of 1*1, 3*3 and 5*5 in the model. This different feature information with dimension channel is fed into each layer in the network. Originally InceptionV3 called GoogleNet but after that it is called inception with version number v3. This model is also available on Keras core library. (Rosebrock, 2017)

T-distributed Stochastic Neighbor Embedding

T-distributed stochastic neighbor embedding is also abbreviated as T-SNE. T-SNE is a visualization tool to envisage high dimensional data into lower dimensions. It calculates joint probabilities based on data points similarities. Using Kullback-Leibler divergence it tries to minimize the joint probabilities of high dimension data into low dimension (2.2. Manifold learning, n.d.). It is recommended to use PCA to minimize the dimensions of the data before using T-SNE. If the embedding is not initialized, it uses PCA as default to reduce the dimensions. It can reduce the dimensions on 2 to 3 components (sklearn.manifold.TSNE, n.d.). The reduced components can lose a major part of the data information therefore, those graphs are only good for visualizing the high dimension data not to analyze or predict something from those graphs. The process is computationally expensive and can take hours to reduce the dimensions of data. The algorithm is also stochastic, it will produce different results on each restart. It also restricts to only 2 or 3 components reduction which loses most of the information for analysis.

Principal Component Analysis (PCA)

PCA is a well-known dimensionality reduction method. It reduces the large data dimensions into fewer dimensions which can still contain most of the information related to the original dimensions. PCA is a statistical analysis tool to detect missing values, categorical data and many more information from the data. It extracts the information and represents it in summary called principal components. To find the principal components, PCA first normalizes the input variable variances into the same continuous range. After that it computes the covariance matrix of these variables and finds eigenvectors and eigenvalues from the computed covariance matrix. Those eigenvectors are arranged in descending order according to their eigenvalues which determines the principal components. The first component always contains most of the information. Those components are not correlated to each other and independent from the original data. It combines all correlated variables from data into one component which helps in analysis and visualization. Every component variance ratio will explain the quantity of information contained in this component. Therefore, a best practice is to choose number of the components which can explain at least 90% of overall data. Those reduced dimensions can help to visualize data better and more clearly to analyze it. (Jaadi, 2022)

Yolov5

Yolo stands for 'You Only Look Once'. It is one of the famous algorithms for detecting objects in an image. It divides an image into small parts where each part is responsible for detecting an object. Its speed and accuracy also makes it more reliable to be used for object detection. Glenn Jocher introduced Yolov5 which is the fifth version of Yolo. He used the Pytorch framework for Yolov5 and made it open source available on GitHub. Yolov5 has different configurations which are varied in size, parameters and speed. The models are trained on the COCO dataset and have simple functionality to include model ensembling and hyper parameter analysis. (YOLOv5 Documentation, n.d.)

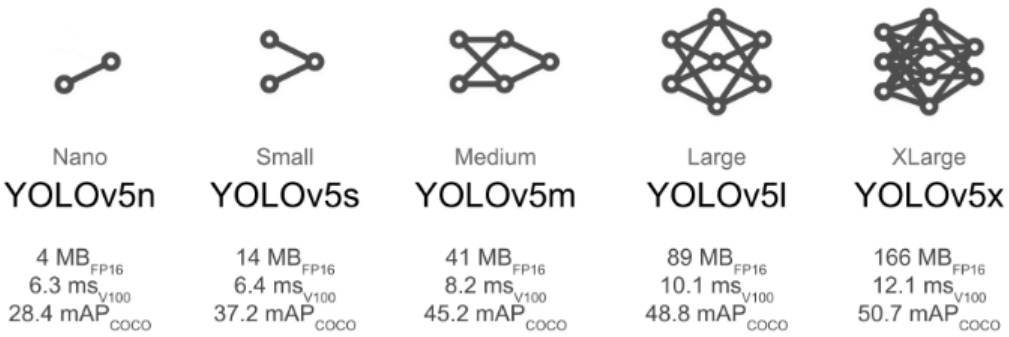

Figure 1: Yolov5 models configuration (Ultralytics, n.d.)

The pre-trained model can be loaded using Pytorch hub and use to detect objects in images. The model can receive input in a form of filename, OpenCV, numpy or Pytorch object and return detected object in JSON, pandas data frame or torch output format. The output consists of bounding box dimensions width and height for both axis and respective class name for that detected object. One additional output parameter is confidence or probability of detected object based on its assigned class. The model can also plot bounding boxes on image and each box represents a detected object. (YOLOv5 Documentation, n.d.)

The model can also be fine-tuned or retrained on custom data. To train the model on custom data, different parameters have to be set like epochs, image size, batch size and hyper parameters. Image size can be varied based on the presence of objects in the training images, if the training data has small objects it benefits to train model on higher resolution. To achieve good testing result it is recommended to use the same resolution on which the model was trained. Hyper parameters can be selected from the pre-trained models like small, medium or large yolov5. (YOLOv5 Documentation, n.d.)

Inside the yolov5 there are many layers based on convolutions and detections etc. the model contains four detection layers and each layer is responsible for detecting different sizes of objects. Every layer has its own dimension for object detection like 4*4, 8*8, 16*16 and 32*32. Using those different detection layers with different sizes Yolov5 produce good results and achieve best accuracy. (Glenn-Jocher, n.d.)

Experiment

As shown in section background, there are many methods of active learning which can be used according to their need. This research is based on deep batch active learning methods (Diverse mini-batch active learning and Deep batch active learning with gradient lower bound). Both methods can choose diverse as well as uncertain images from data and support batch active learning.

This section will describe the experiments in detail. Including the preparations, the difficulties faced during experiments and the experiments outcomes.

Initial Experiment

Before diving into the main problem, a preliminary experiment was conducted. The purpose of this experiment was to test the active learning process on a pre-trained model (Existing datasets, publicly and easily available).

Dataset and Model Preparation

300 images of cats and dogs were collected from online available kaggle dataset (https://www.kaggle.com/datasets/nafisur/dogs-vs-cats). The actual dataset size is bigger hence, only 300 images were selected randomly for the initial experiment.

Figure 2: Cats and Dogs sample images from kaggle dataset

A pre-trained model inceptionV3 with ImageNet weight was used to extract features from selected images.

```
Model = InceptionV3(weights='imagenet', include_top=False)
```

To extract features from last layer, include_top parameter was set to false so that it can return features before last fully connected layer. Those features were then used for the active learning process. Each image feature vector consists of 51200 features. Kmeans algorithm was used on the extracted features to perform clustering. There are only two classes cat and dog therefore k=2 was chosen for number of clusters. The clustering was performed based on extracted features.

The feature vector dimensions are very large therefore, based on the Kmeans centroids, the Euclidean distance was calculated for each image feature. The distances were plotted on the scatter plot which can be seen in the figure 3. The image shows distance and shape of both clusters. The plot can help to visualize the distribution of images towards the clusters and distances will help to choose diverse and outlier samples from unlabeled pool.

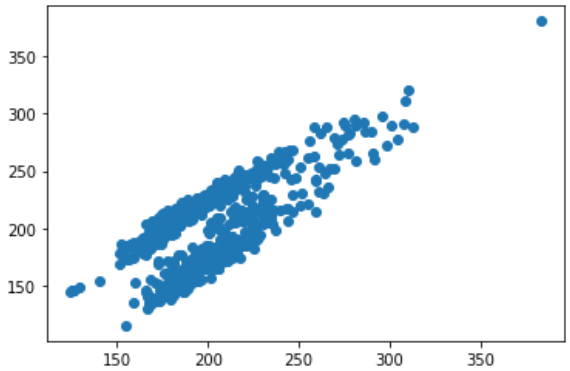

Figure 3: Distance plot on cats and dogs dataset image features

To test on new classes, new images were added into the dataset on which the model was not trained previously. This test is to check if model can produce different features for those new unseen images, then seen images.

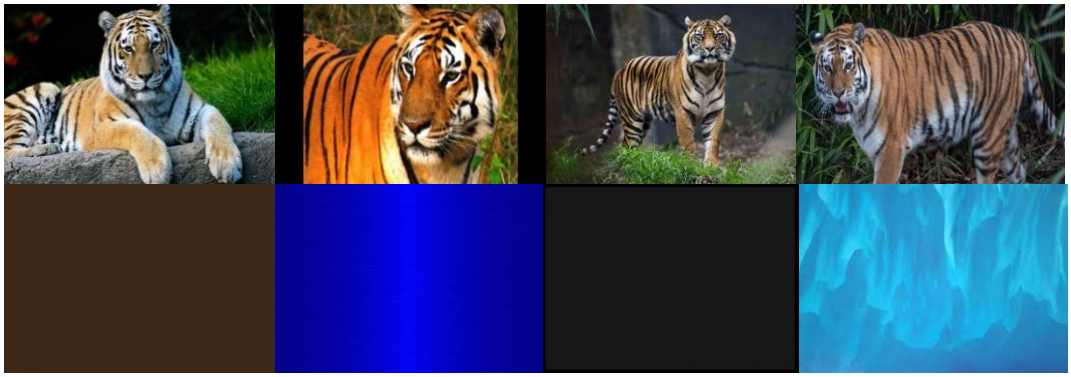

Figure 4: Newly added random images into a dataset to test Kmeans clustering

The new images were added into dataset and the same process was executed on the new data. As the known classes are only 2 so Kmeans was performed on new data with same 2 clusters value. The distances were calculated again, and the updated plot is shown in figure 5. The plot shows that new images were clustered very well from cat and dog images.

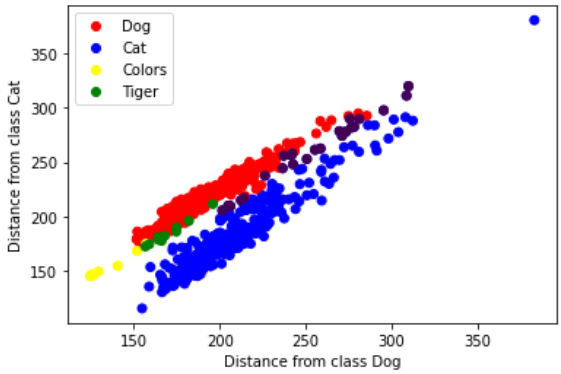

Figure 5: Distance plot with trained and not trained classes

The scatter plot shows 4 different clusters cat, dog, tiger and colors. They all cluster significantly. The purple points are showing the misclassified samples of cats and dogs images. Moreover, the plot showing an outlier from cat dataset. The experiment shows how to proceed to next steps and choose images based on the image features. The distances can help to find images which are far from their centers to choose outliers and the images which lies in between cluster centers to select diverse images. Entropy or any acquisition function can be calculated to choose uncertain images. Images with high entropy score can qualify for annotation from unlabeled pool.

Experiment with Yolov5

In this experiment, the chosen active learning methods were applied on the output of Yolov5. Yolov5 is an open-source code and can be adapted according to the need. Therefore, to extract features during training of yolov5 on required underwater fish data is the main task for this experiment. The process is explained below:

- Firstly, a seed was prepared for initial training of yolov5. For the seed only three types of fishes were selected (Fish_unspecified, Fish_cod and Jellyfish_aurelia). For each class only 100 images corresponding to their labels were selected randomly from the main dataset. To train yolov5 following command was used:

```
train.py -data (data.yaml filepath) -project (project name) -image (image
size) --weights (path of pre-trained yolov5 model weights) -batch-size
(size of batch)
```

- The features can be extracted from the detect method of yolo.py file during training. In the detect method all four detect layer's input will be used and it will produce bounding box size for detected objects. The features can be extracted in two ways: before or after final convolution. Therefore, for this experiment features were extracted by both ways to check the difference. After convolution, the feature dimensions decreased than before convolution. With batch size of 16 for each epoch of training 16 images were passed for training. Each iteration has four different detect layers to detect different size of objects therefore, for each batch there are four different features vectors before convolution and four feature vectors after convolution to track all detection.
- It is necessary to track the images back to the batch because images were selected randomly for each batch. The image names were saved according to their batch number. Overall, there are 9 numpy files for each batch in each iteration. There is no need to store features and image names for every epoch because features keep on updating all the time. Therefore, only last epoch data is needed to be stored because those features are updated over the epochs.
- The extracted features and names were loaded back to perform active learning method (Diverse mini-batch active learning). All respective features were combined into one list such as detection layer1, detection layer2, detection layer3 and detection layer4 features also detection convolution layer1, detection convolution layer2, detection convolution layer3 and detection convolution layer4 features.

- Kmeans was conducted on all 8 extracted features with 3 clusters. The feature dimensions were very high therefore, based on those clusters center's distances were calculated. Scatter plot was plotted on those distances.

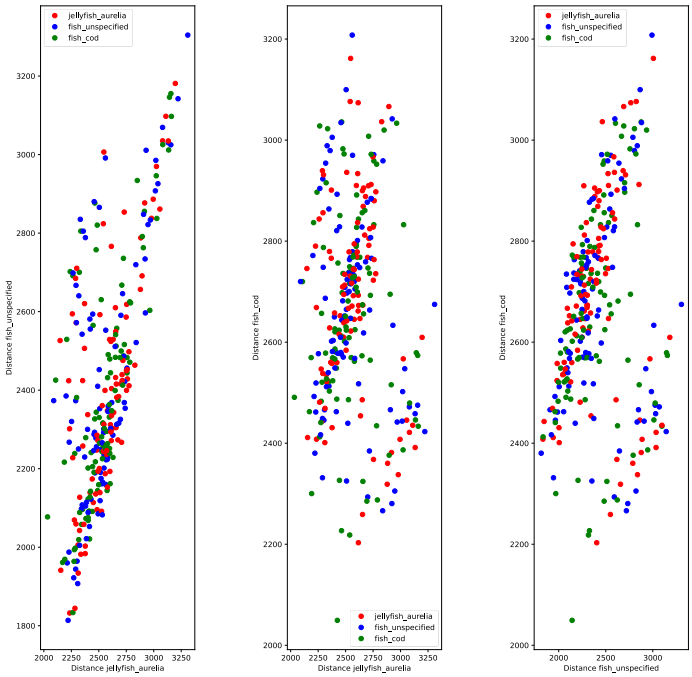

Figure 6: Distance plot for Yolov5 detect feature stage1

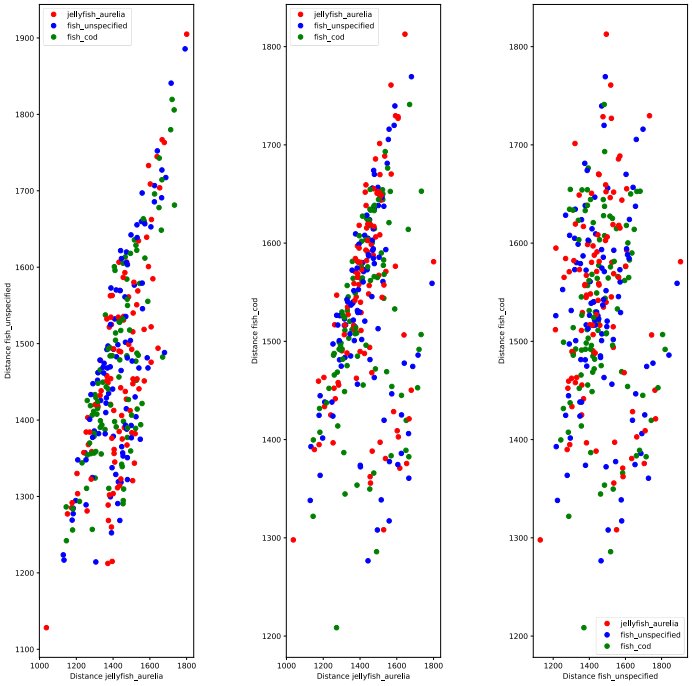

Figure 7: Distance plot for Yolov5 detect feature stage2

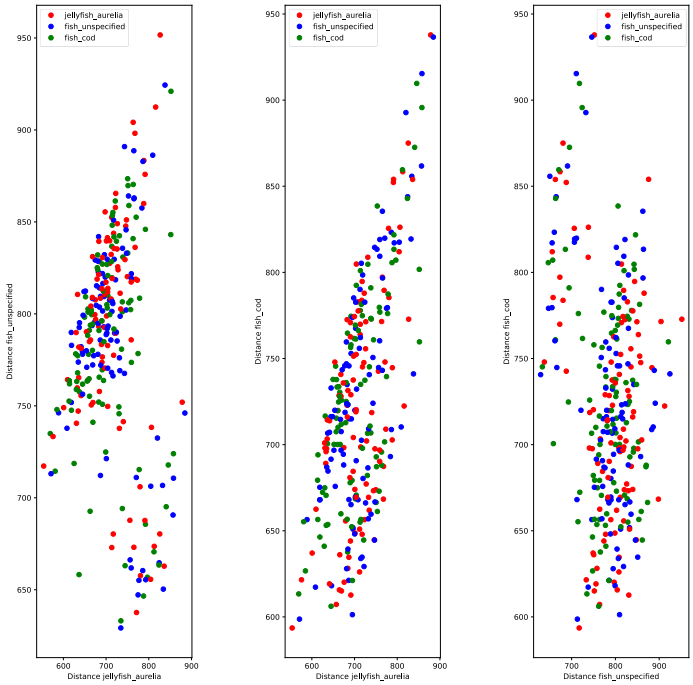

Figure 8: Distance plot for Yolov5 feature stage3

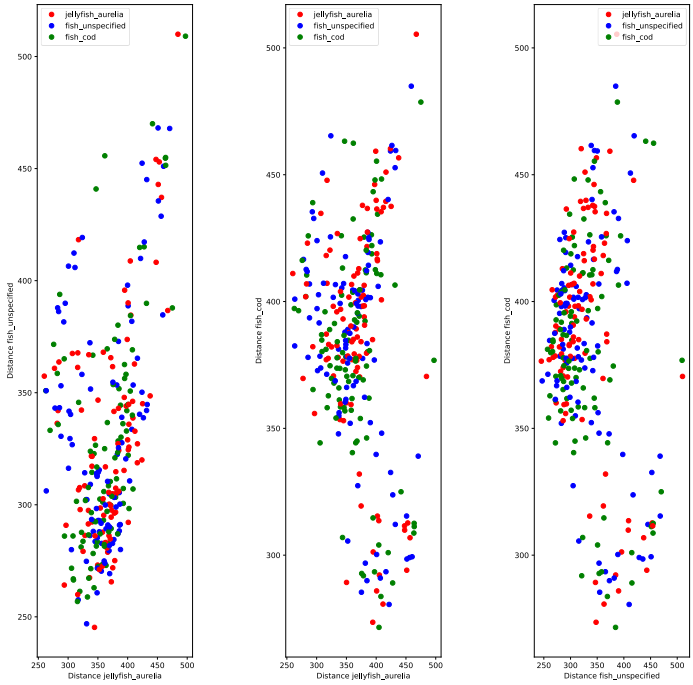

Figure 9: Distance plot for Yolov5 feature stage4

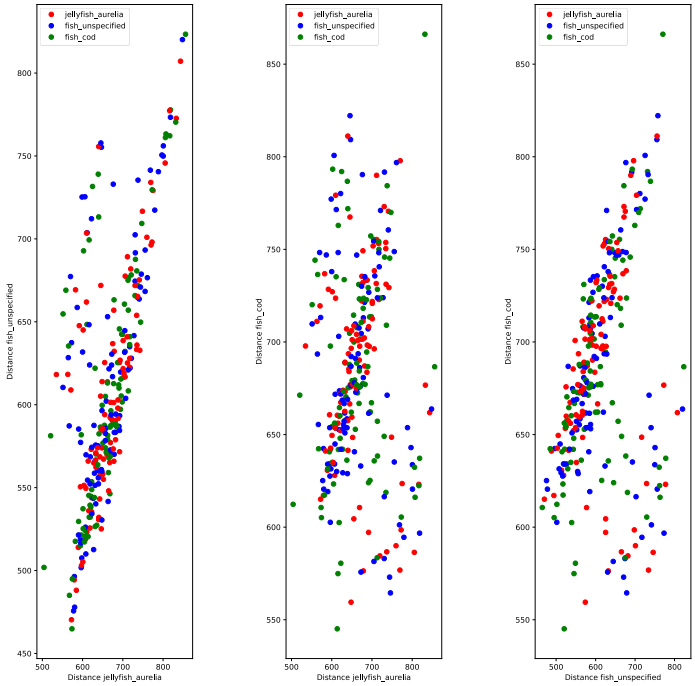

Figure 10: Distance plot for Yolov5 detect feature after convolution stage1

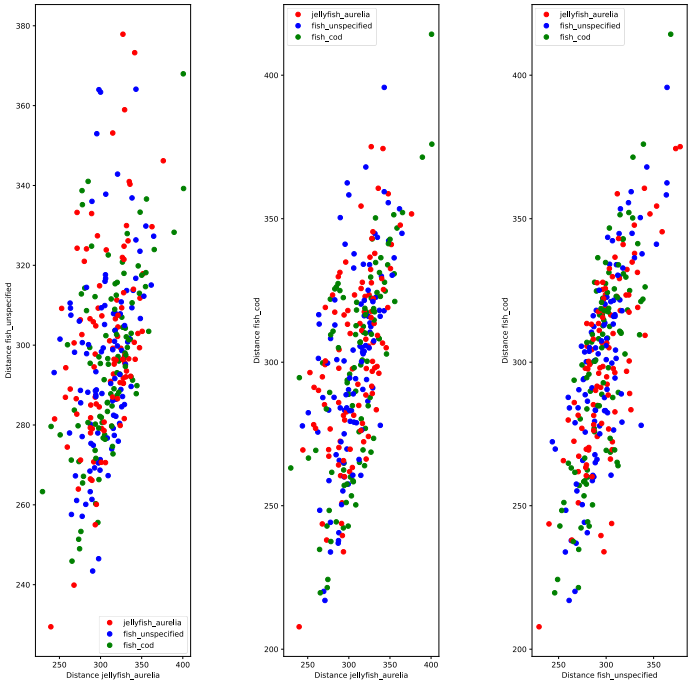

Figure 11: Distance plot for Yolov5 detect features after convolution stage2

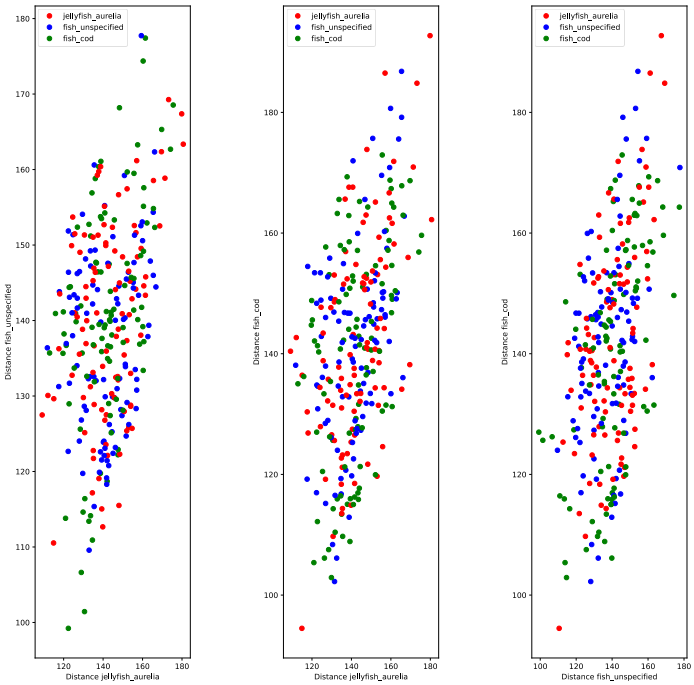

Figure 12: Distance plot for Yolov5 detect features after convolution stage3

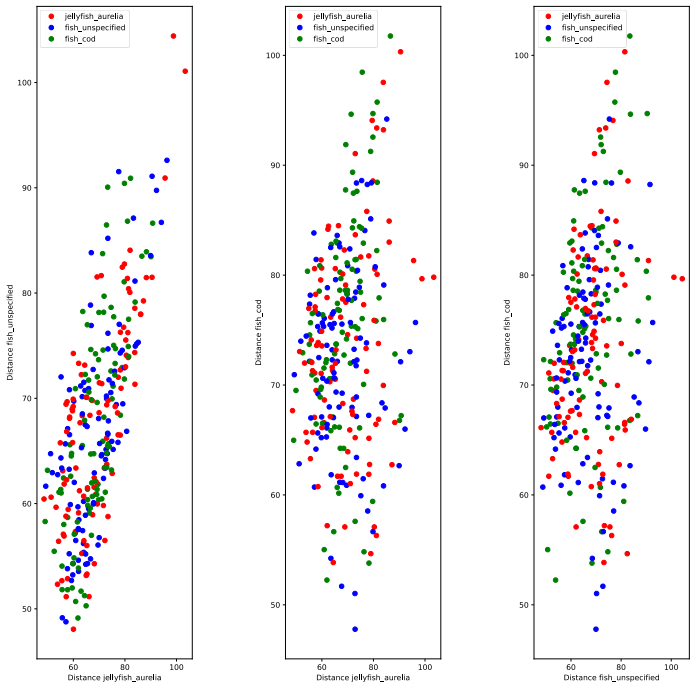

Figure 13: Distance plot for Yolov5 detect features after convolution stage4

- The plots in figure 6-13 show that the clusters are not scattered they are merged for both detect features and convolution features. The reason of this is because the fish images are on grey scale and all of them shared mutual pixel values. Therefore, PCA was applied to reduce the dimensions to check if it can make clusters better.

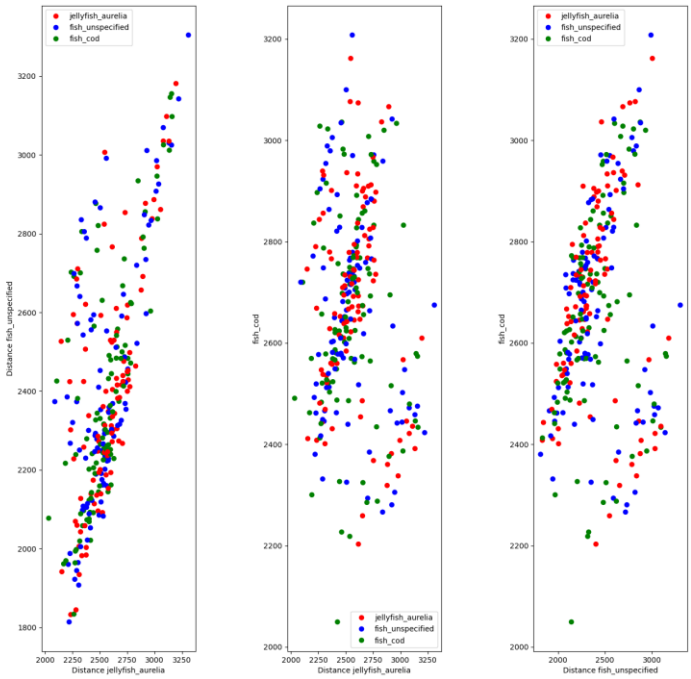

Figure 14: PCA distance plot for Yolov5 detect features stage1

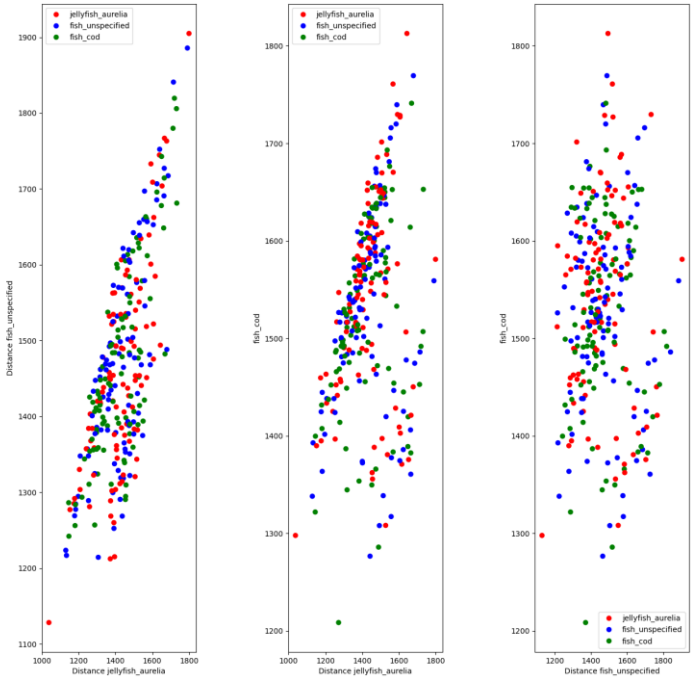

Figure 15: PCA distance plot for Yolov5 detect features stage2

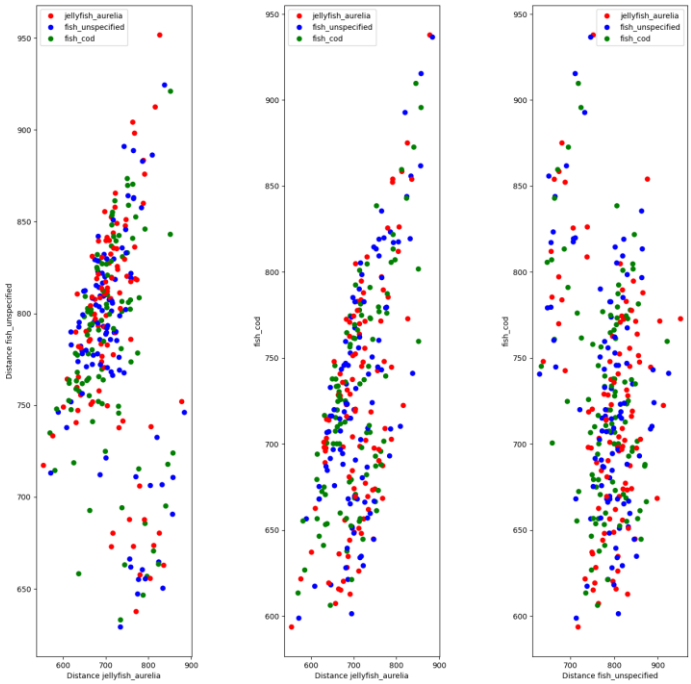

Figure 16: PCA distance plot for Yolov5 detect features stage3

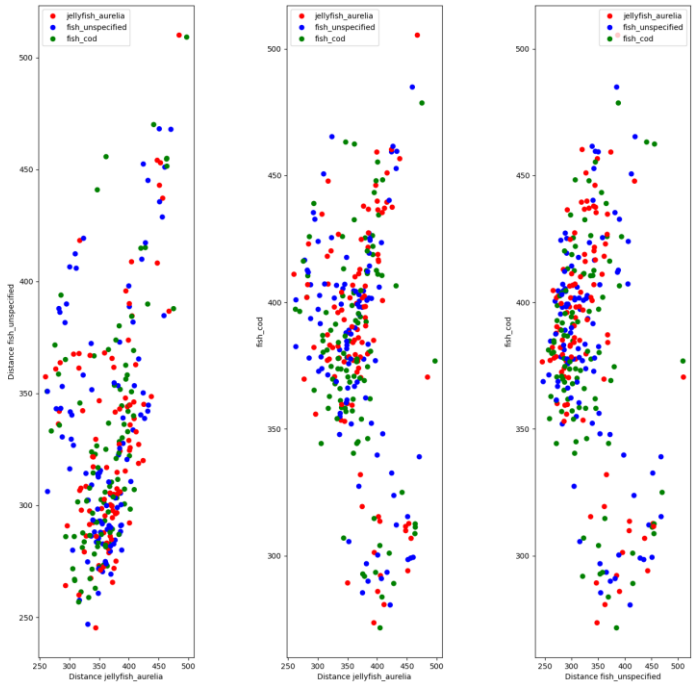

Figure 17: PCA distance plot for Yolov5 detect features stage4

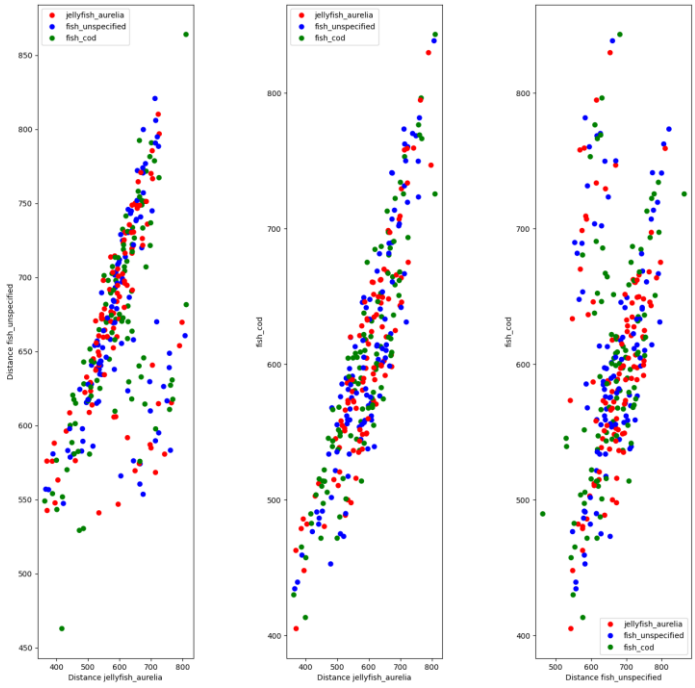

Figure 18: PCA distance plot for Yolov5 detect features after convolution stage1

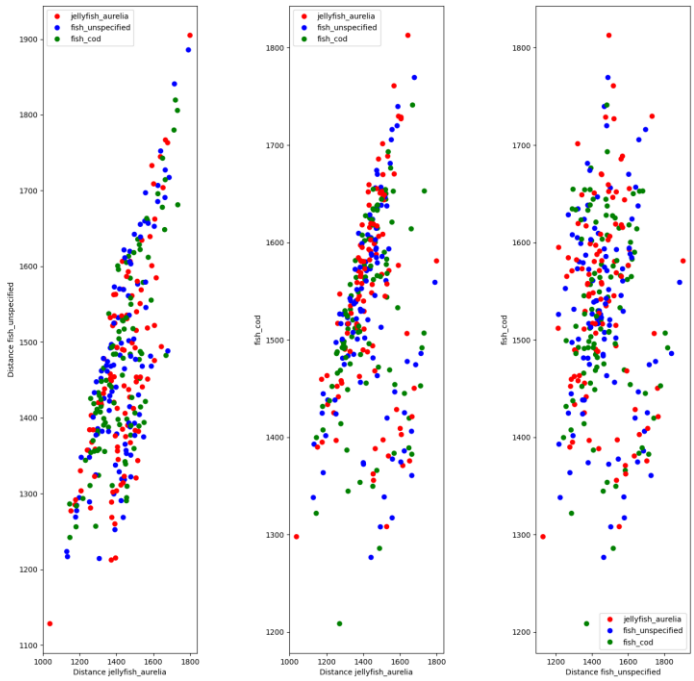

Figure 19: PCA distance plot for Yolov5 detect features after convolution stage2

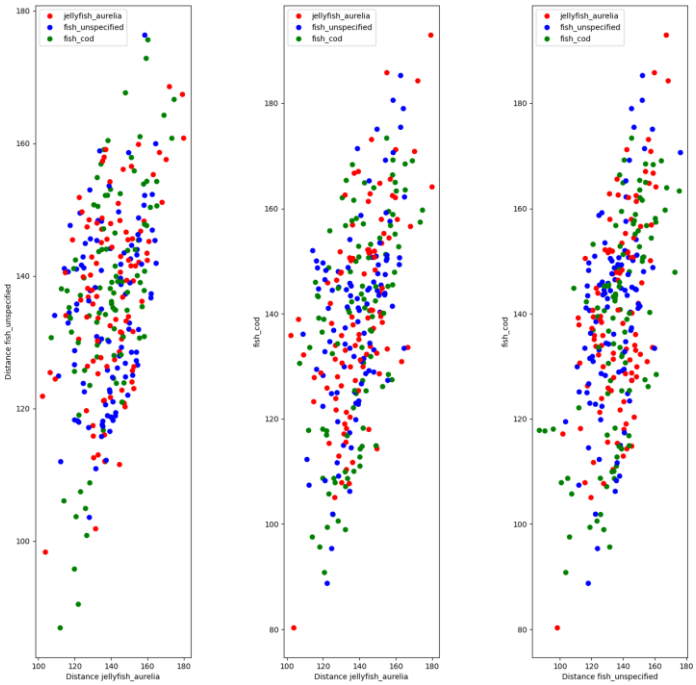

Figure 20: PCA distance plot for Yolov5 detect features after convolution stage3

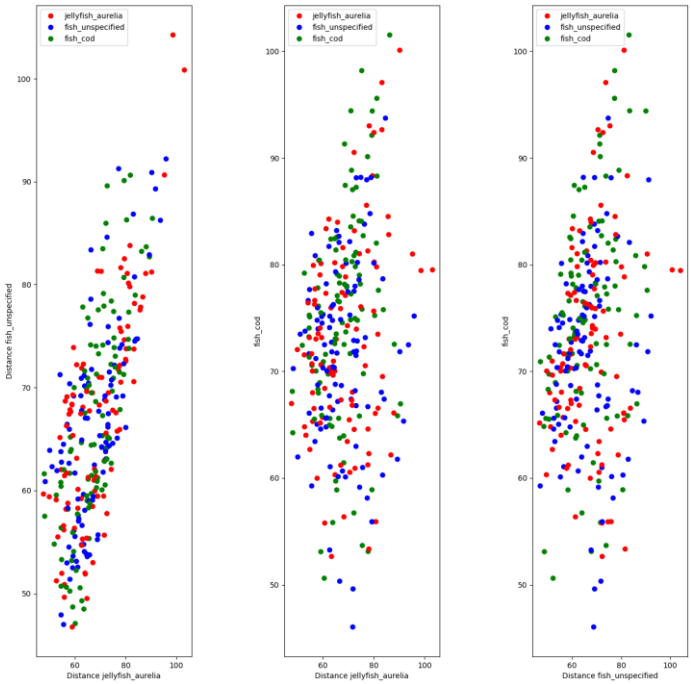

Figure 21: PCA distance plot for Yolov5 detect features after convolution stage4

- As seen in figures 14-21, PCA results are still not satisfactory. The clusters are still merged with each other. It shows that reducing the dimensions will not help to extract relevant features from the image. The next approach would be to find features of the bounding boxes which may help to find the relevant features to plot clustering. Yolov5 provides a way to manipulate its code according to the user's need, but its complex structure will make things a bit difficult. Some attempts were made to find those bounding box features but, unfortunately, it was unsuccessful. To find gradient values of last layer on Yolov5 is also not possible because Yolov5 calculates loss at the end as an average of whole batch. So, to find the gradient value for each image is not possible. Therefore, due to the restriction of time the experiment was stopped because both methods of active learning could not be possible on Yolov5. The initial experiment was proceeded with fish data using InceptonV3 and ImageNet to perform active learning method which will be discussed in the next part.

Proceeding Initial Experiment

Yolov5 was unsuccessful with active learning experiment because the bounding box features could not be extracted. So, the initial experiment was proceeded with few changes to perform active learning.

- The dataset was changed to fish data bounding box images. The bounding box images were extracted from yolov5. Those images contain only required part of the image which is needed for training and feature extraction. 1500 images for each class were added into training and active learning dataset. Additional 3500 images from each class were used for testing purpose.
- Those 1500 images were divided into training and active learning process. Only 200 images were chosen randomly from each class to use for training. In total 600 Images were in initial training seed.
- To train an InceptionV3 model on custom fish data additional layers were added into the model. The structure of the model is given below:

```
pre_model=InceptionV3(weights='imagenet',include_top=False,input_shape=(150,150,3))
```

- ImageNet weights were used for training and last fully connected layers were removed from the model to put new layers. The new layers were used to predict probabilities for each class according to the image. The image shape for the model was set to (150, 150, 3). The layers which were added into the model are:

```
x = pre_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation="relu")(x)
x = Dropout(0.5)(x)
x = Dense(512, activation="relu")(x)
Predictions = Dense(3, activation="softmax")(x)
```

  New layers were extended into the InceptionV3 model. The layers were fully connected, and the final layer predicts probabilities for each class because only three fish categories were used for this experiment that is why node value is 3. Softmax activation function produced confidence for each class with sum together 1.
- To keep the essence of the original model InceptionV3, few layers were set to trainable false other were trained while the custom model was training.

```
for layer in self.model_.layers[:52]:
      layer.trainable = False
```

  First, 52 layers out of 311 layers were not trained with custom model others were trained accordingly.

- The active learning pipeline was followed by training a model. The pipeline was computed for each active learning epoch. For active learning there were 1300 images from each class and for each epoch 100 images were selected from each fish category to perform active learning. On each epoch there were 300 images and active learning chose images from them for annotation. The pipeline is shown in figure 22:

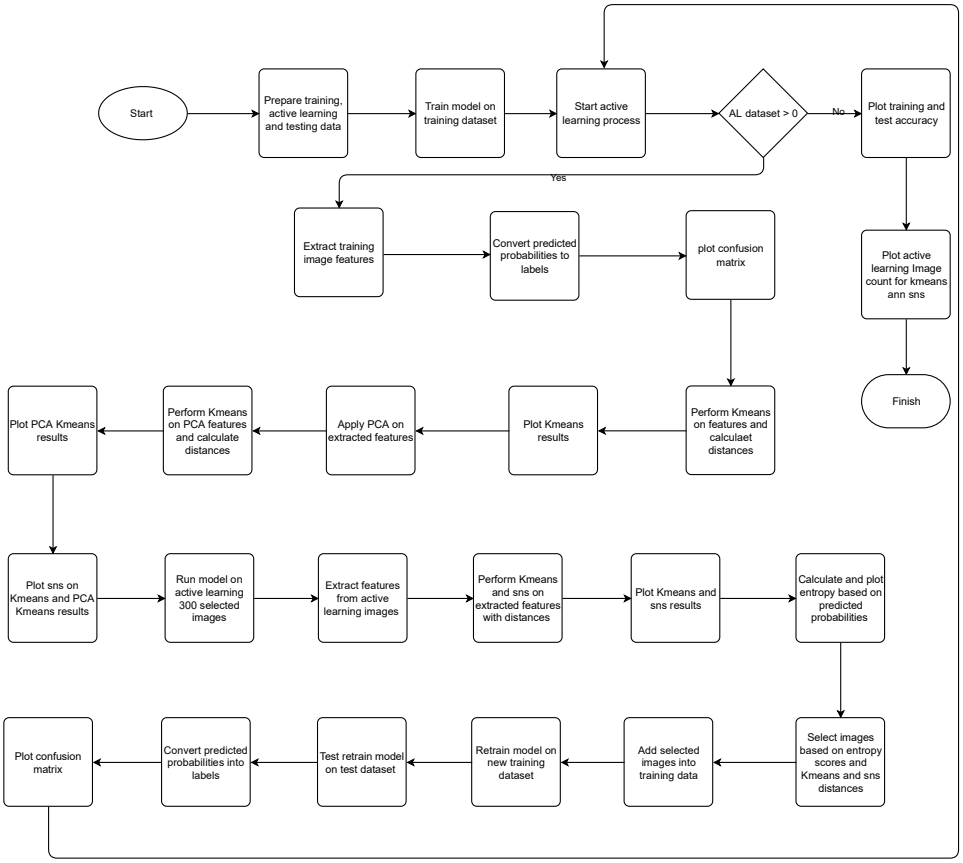

Figure 22: Active Learning pipeline

- The pipeline saved the results on each stage to check the accuracy of model on newly added images. The active learning process test on both Kmeans and Sns to check the difference and reliability for analysis. The results will be discussed in detail in the next chapter.
- Unknown class images were also tested to observe the behavior of the trained model that how it can incorporate unknown images.

Results

Results were saved for each epoch to see the rate of change and progress on data clustering and accuracy. In this section only initial and final results will be discussed thoroughly to observe the difference however, remaining results of Kmeans and Sns on training data can be found in Appendix B.

As per the model, training starts; it's better to see the accuracy and loss of both training and validation sets. It shows that if model is converging properly with the provided data and how the accuracy is increasing over training epochs. It also shows how the loss will decrease over the training time and is able to produce minimum error.

Figure 23: Training and validation accuracy plot for active learning first and last iteration

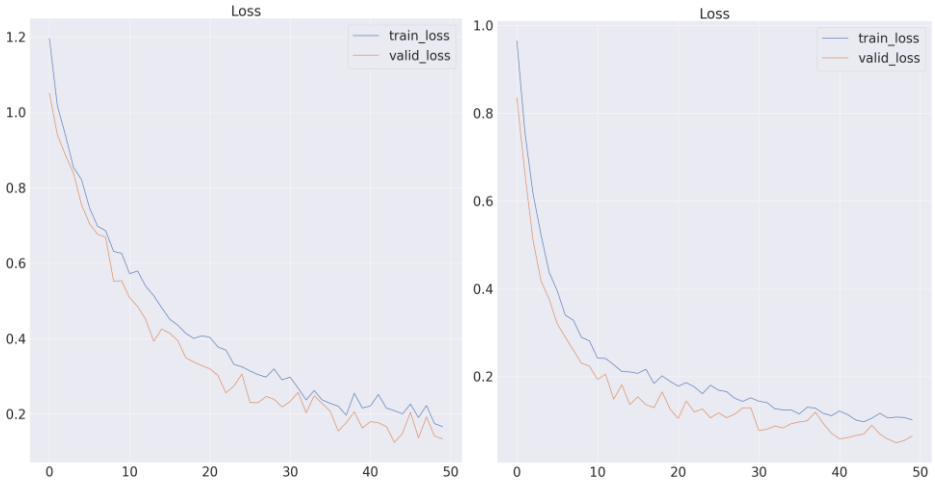

Figure 24: Training and validation loss for active learning first and last iteration

As seen in figure 23 the accuracy of the model is reasonable at the start of process. Both training and validation accuracy achieved good results. It reaches more than 90% easily with first set of training data. The accuracy fluctuates a bit for both validation and training but during active learning process the new chosen images were added into the dataset which make the model more stable. At the end of process, it seems that accuracy increased a bit than before. On the other hand, in figure 24 the loss score for both validation and training are less than 2%. At the end of the process, it decreased a bit more. It shows that model trained very well on the given data, and it converges properly. As new data were added into the training dataset, it maintains and improves the model accuracy and loss.

The accuracy plot in figure 23 can be seen more clearly in a confusion matrix. It shows how many images were categorized correctly by the model and how many were classified wrongly.

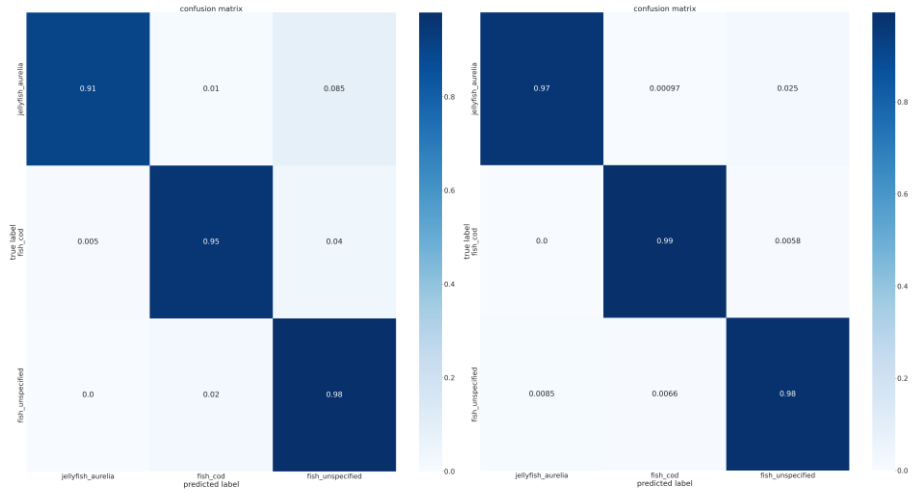

Figure 25: Confusion matrix for active learning first and last iteration

As shown in figure 25 confusion matrixes the accuracy is satisfactory all over the process. It increased also for class jellyfish_aurelia and fish_cod because at the start of the process their accuracy is 91% and 95% but it increased during active learning process to 97% and 99%. For class fish_unspecified it remains same which means the model is trained well on that class.

The entropy scores were also calculated for each batch of active learning. Which can be shown in figure 26. The entropy score will help to find which images the model is not certain about therefore the images with high entropy score will be selected for annotation.

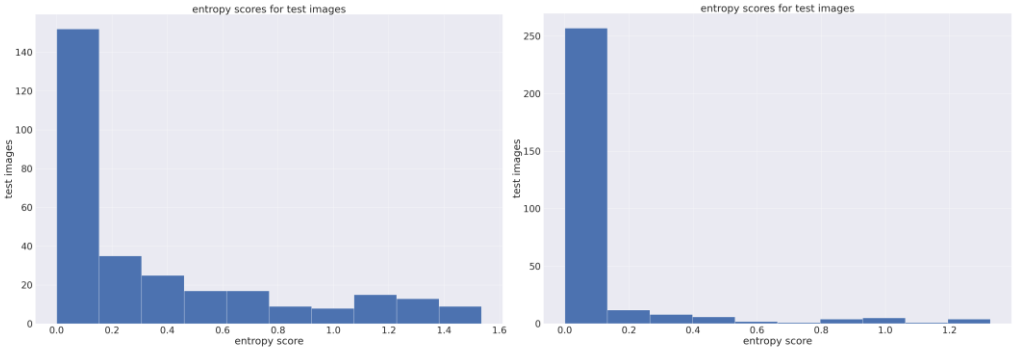

Figure 26: Entropy scores for active learning first and last iteration selected images

The entropy graphs in figure 26 are showing the scores of all batch images. In the starting of the process, the number of uncertain images is more because more images have high entropy scores. As the active learning process proceeds it starts decreasing and on last stage the number of images with high scores are not abundant.

Kmeans plot were also created to identify the diversity in the images based on the distances. The graphs below show clustering of the images on Kmeans from starting and ending process.

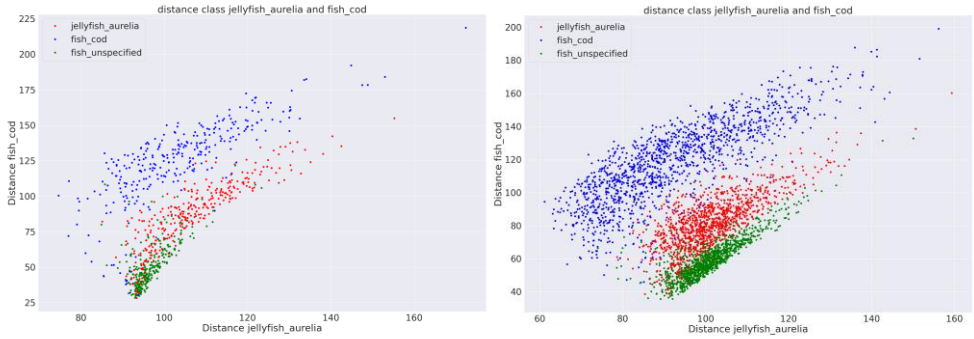

Figure 27: Distance plot of class jellyfish_aurelia and fish_cod for active learning first and last iteration

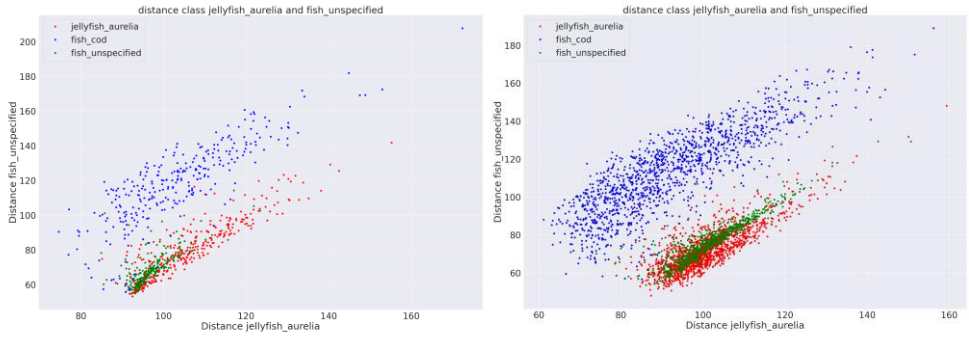

Figure 28: Distance plot of class jellyfish_aurelia and fish_unspecified for active learning first and last iteration

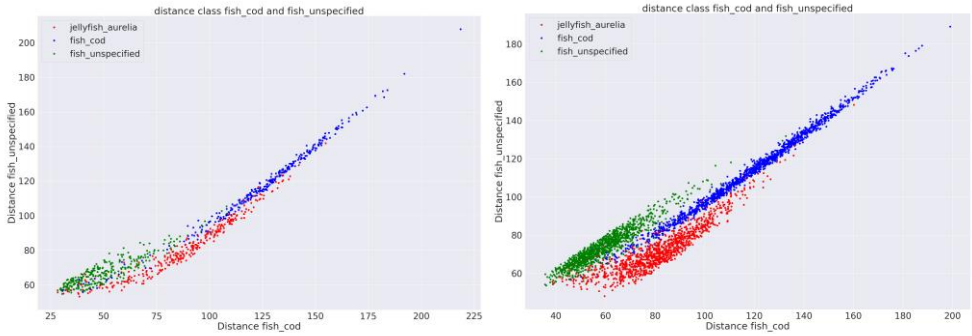

Figure 29: Distance plot of class fish_cod and fish_unspecified for active learning first and last iteration

In above figures 27-29, the distances were plotted for all three classes. As visible in the plots the clusters were much merged at the start of the process because of the smaller number of images. As the process increased and more images were added into the data, it starts separating a little. Diverse images were selected which have maximum distance from their class centers or are in the middle of centers.

Sns plots were also created to view all class clusters in one plot to identify the diversity in the images with better view. The graphs below will show the clustering of the images on Sns from starting and ending process.

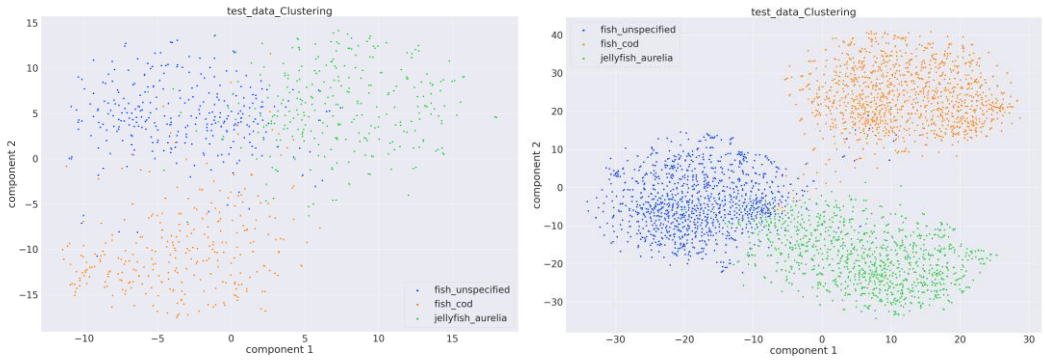

Figure 30: Sns plot for active learning first and last iteration

In figure 30, Sns reduced the dimension of the data to two components which are easy to plot and view. The graphs show that at the start of the process the clusters are very apart and scattered because dataset is very minimum but as the new images were added into the dataset by active learning process. It starts creating centered and merged clusters. The class fish cod is separated well from the other clusters but class jellyfish_aurelia and fish unspecified are attached to each other. Still the clusters are clear enough to separate them. This graph is just for visualization but not for analysis because most of the information about data was discarded with dimension reduction.

Active learning process is to choose the images which are diverse and uncertain. Those images were added into dataset to increase the model prediction with less but important images. The graph in figure 31 will shows how many images were selected from above Kmeans and Sns results on each iteration.

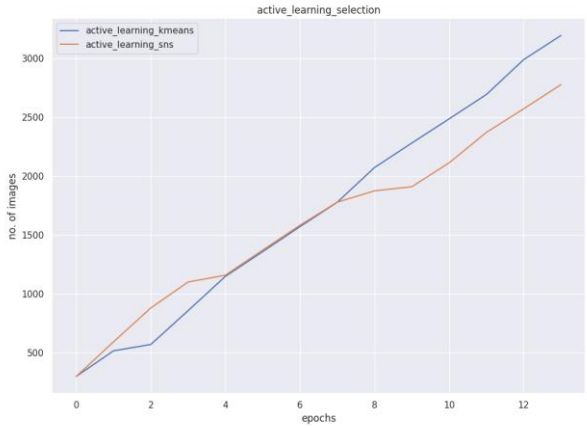

Figure 31: Selected images count for Kmeans and Sns during active learning process

For both Kmeans and Sns methods selected images can be seen in the figure 31. For information, only Kmeans selected images were added into dataset. It can be seen in the graph total number of images for Sns over the active learning epochs were increased for first 4 epochs. For next three epochs the number is same for both methods but after 7th epoch the image count was decreased compared to Kmeans count. Sns has very low information about the original data that's why the reliability of it is not sure. On the other hand, images for Kmeans method were increased after 7th epoch. Kmeans preserve the real information about the data that's why its reliability is higher than Sns.

To visualize the accuracy with respect to number of images will be shown in figure 32. It shows that how the accuracy increased over the size of training data.

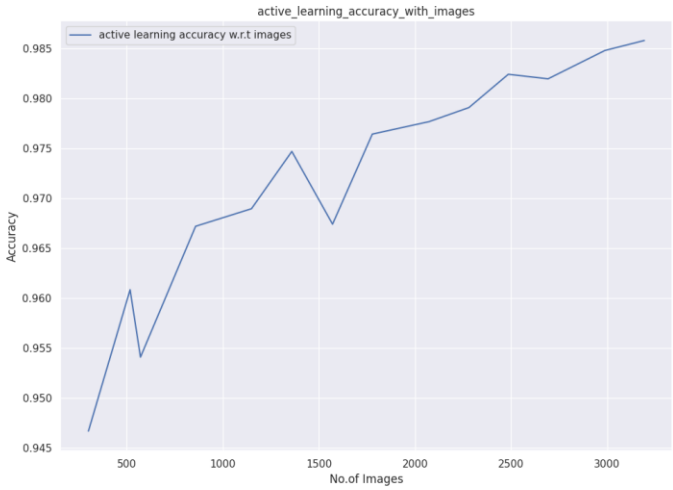

Figure 32: Active learning accuracy w.r.t number of images

In figure 32, the plot shows that training accuracy increased as the number of images start increasing in the training dataset. After 1500 images in training set accuracy decreased a bit from 97% to 96% but it rises again and reached the accuracy of 98%. It shows that active learning selected images help a model to train better and to achieve good accuracy.

The total number images are a combination of all three classes. In figure 33 it can be seen that how many images were selected from each class.

Figure 33: Number of selected images for each class per active learning iteration

In figure 33 it shows that for first iteration the number of images for each class is same because initial training dataset contains 100 images for each class. For class jellyfish_aurelia and fish_unspecified the number of selected images were mostly high except for some iterations where selected images are 20 or less. For class fish_cod the number of images is always high except third and tenth iteration. Which shows that for model training fish_cod images required more number than other fish classes. It also shows that for some iterations active learning select almost all the images from the batch which means those images have high distances or high entropy scores. On third iteration active learning select a smaller number of images compare to other iterations which means the third iteration batch has images which are similar to the training dataset.

Test data were prepared to evaluate the retrained model on each active learning epoch. Therefore, the confusion matrix in figure 34 shows the testing accuracy for start and end of the active learning process.

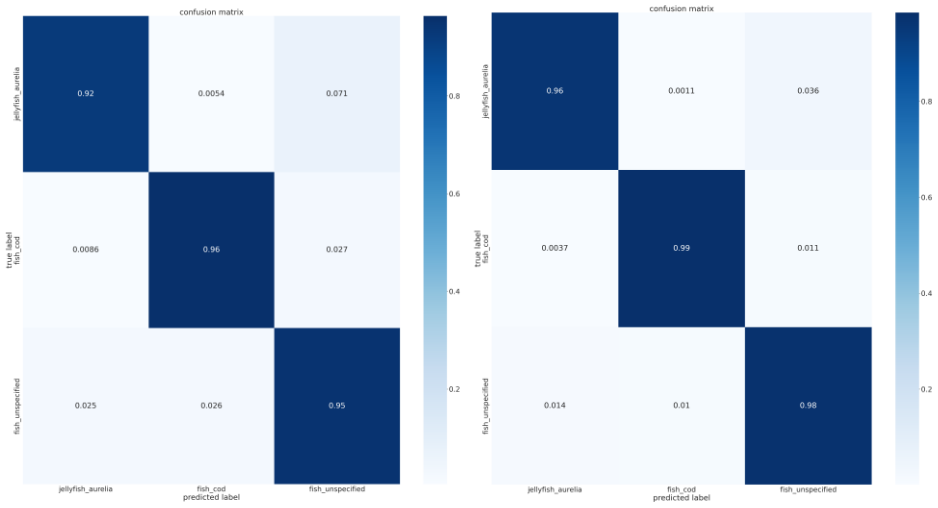

Figure 34: Test data confusion matrix for active learning first and last epoch

Confusion matrix helps to see the accuracy of model in terms of correct classification of the images. The above figure 34, reveal that accuracy of classes for first active learning epoch is acceptable but compared to the last epoch, it is less. It increased over the epochs as images were added into the training data to retrain the model. For class fish_cod, the accuracy was increased from 96% to 99% and for class fish_unspecified accuracy increased from 95% to 98%. Lastly, for class jellyfish_aurelia accuracy increased from 92% to 96%. It shows that active learning helps the model to increase its accuracy.

Process can be validated more precisely if both test and train dataset accuracy can be checked together to see the improvement of accuracy.

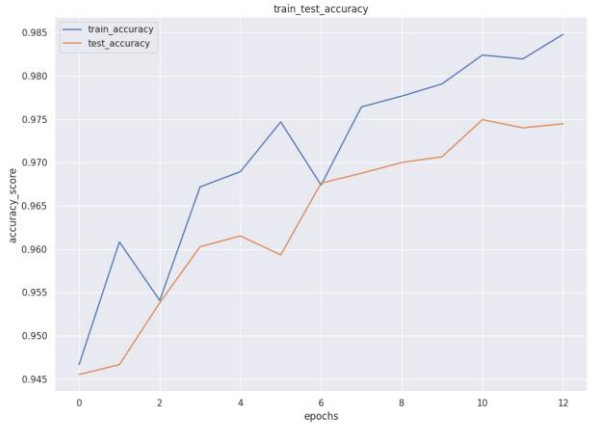

Figure 35: Train and Test accuracy

Training and test accuracy are shown in figure 35. Over the active learning epochs training accuracy increased up to 98% and test accuracy increased till 97.5% which resembles to good model prediction. As seen in the graph the accuracy for both testing and training matches at 2nd and 6th epoch. Overall, both training and testing accuracy are increasing with iterations which shows that model was trained well.

New images which belong to other classes were also examined on the trained model. To check how the model reacts in unknown situations. Below Sns graph in figure 36 will show how the clusters were made with new unknown data.

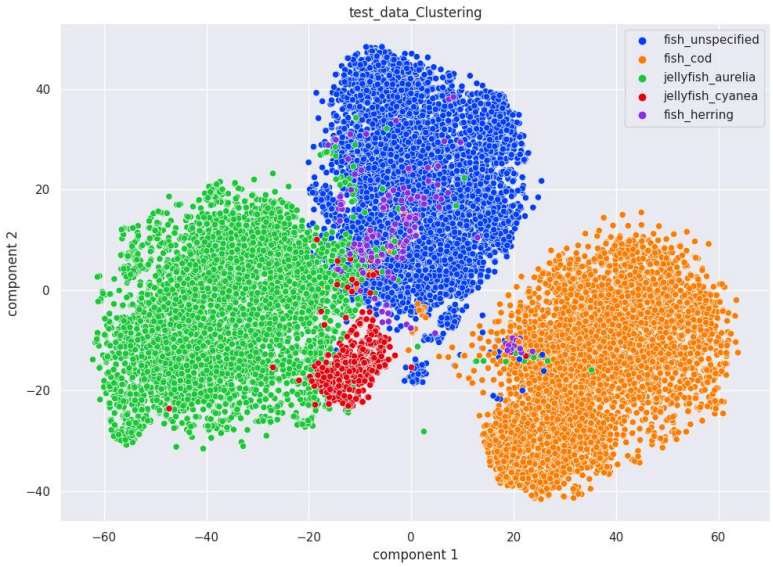

Figure 36: Sns plot for unknown classes

As shown in figure 34, new fish data fish_herring and jellyfish_cyanea were tested with the model. The Sns graph helps to visualize it better rather than Kmeans. Fish_cod cluster is separated very well from other clusters. Jellyfish_aurelia and jellyfish_cyanea are clustered well but very close to each other because both are the types of jellyfish and both have some similar characteristics. On the other, hand fish_unspecified and fish_herring are mixed up. The model may confuse herring and unspecified fish, as the latter is a rather ambiguous class that contains many poorly visible fish that may also be present in a school of fish that has been labeled as herring.

Based on those clusters active learning cannot be performed because the new clusters information is missing. To initialize Sns or Kmeans in the beginning cluster information is unknown hence it is not clear how many new clusters will be added into the model. Only based on existing groups clustering can be performed. Therefore, for active learning only those class images were used on which model was trained. Of course, there are some methodologies to find clusters dynamically without knowing the actual number. Those methodologies can be used to incorporate unknown clusters.

Discussion

Summary

In this research work deep batch active learning techniques were used to speed up the process of annotation. The purpose of deep batch methods was to select more than one images in single iteration. Deep learning algorithms work with batch learning and computationally high therefore, deep batch methods will speed up the process of training.

Models that were tested for active learning ware Yolov5 and InceptionV3. Both models are pre-trained and have good satisfactory state of the art results in image classification, detection and computer vision fields. An Initial experiment was performed on InceptionV3, and it gave a confidence to proceed with active learning with Yolov5. For diverse mini-batch active learning, the features were extracted from the images and clustered using Kmeans to select diverse images, but the results were not satisfactory because the full image features were used for active learning method. The fish images were in gray scale and were hard to identify fishes on feature level. Therefore, bounding box features were needed but those features were not available easily, using the provided Yolov5 implementation. Different tests were computed to find the required features, but no success was achieved. For deep active learning with lower bound, gradient values are required for last layer which were also not available on Yolov5 according to the research. Therefore, the subsequent experiments were carried out using a classification network, namely the InceptionV3.

For the final test, the detected bounding box images from Yolov5 were used as a dataset because those bounding box include the required part of the image, i.e., the image region showing a fish or jellyfish. InceptionV3 was modified according to the fish data and trained on it. A good classification accuracy was achieved using the model. Several series of steps were applied to compute mini-batch active learning and results were satisfying and active learning computed well on InvceptionV3 updated model.

Even though the intended model Yolov5 was unsuccessful at first, it should be possible to apply similar methods as have been investigated if the image features could be extracted on a bounding box basis. The InceptionV3 model worked well in the active learning process which shows that it can really help to increase model accuracy in less time. Due to the time constraint only one method testifies on InceptionV3 but deep batch active learning with lower bound can also be tried in a future work.

Future Work

Active learning methods were applied on Yolov5 and InceptionV3. Yolov5 can be improved to work on active learning specified methods if bounding box features are available. Deep batch active leaning with lower bound can also be tested on Yolov5 if gradient values of last layer are able to calculate properly. This method can also be tested on InceptionV3.

New unknown images can also be testified on active learning by making Kmeans or Sns clusters dynamic. Other possibilities to test active learning include R-CNN, faster R-CNN or matrix learning to make training efficient with less data if it is required.

Literature

2.2. Manifold learning. (n.d.). Retrieved from scikit-learn: https://scikit-learn.org/stable/modules/manifold.html#t-sne

Analytics, D. (2020, Aug 18). What Is Principal Component Analysis (PCA) and How It Is Used? Retrieved from Sartorius: https://www.sartorius.com/en/knowledge/science-snippets/what-is-principal-component-analysis-pca-and-how-it-is-used-507186

Andreas Kirsch, J. v. (2019). BatchBALD: Efficient and Diverse Batch Acquisition for Deep Bayesian Active Learning. Arxiv:1906.08158, 16.

Andreas Kirsch, J. v. (2019). Human in the Loop: Deep Learning without Wasteful Labelling. Retrieved from OATML: https://oatml.cs.ox.ac.uk/blog/2019/06/24/batchbald.html

Beichen Zhang, L. L. (2020). State-Relabeling Adversarial Active Learning. IEEE/CVF Conference on Computer Vision and Pattern Recognition, 10.

Donggeun Yoo, I. S. (2019). Learning Loss for Active Learning. Arxiv:1905.03677, 10.

Ghahramani, Y. G. (2016). BAYESIAN CONVOLUTIONAL NEURAL NETWORKS WITH BERNAOULLI APPROXIMATION VARIATIONAL INFERENCE. Arxiv: 1506.02158, 12.

Gildenblat, J. (2020). Overview of Active Learning for Deep Learning. Retrieved from jacobgil.github.io: https://jacobgil.github.io/deeplearning/activelearning

Glenn-Jocher. (n.d.). GitHub - ultralytics/yolov5: YOLOv5 🚀 in PyTorch > ONNX > CoreML > TFLite. Retrieved from GitHub: https://github.com/ultralytics/yolov5

ImageNet Models — Neural Network Libraries 1.29.0 documentation. (n.d.). Retrieved from Nnabla.readthedocs.io: https://nnabla.readthedocs.io/en/latest/python/api/models/imagenet.html

Jaadi, Z. (2022, July 14). A Step-by-Step Explanation of Principal Component Analysis (PCA). Retrieved from Built In: https://builtin.com/data-science/step-step-explanation-principal-component-analysis

Jifan Zhang, J. K.-S. (2022). GALAXY: Graph-based Active Learning at the Extreme. Arxiv: 2202.01402, 15.

Jingya Shao, Q. W. (2019). Learning to Sample: an Active Learning Framework. Arxiv: 1909.03585, 10.

Jordan T. Ash, C. Z. (2020). DEEP BATCH ACTIVE LEARNING BY DIVERSE, uNCERTAIN GRADIENT LOWER BOUNDS. Arxiv: 1906.03671, 26.

K-Means Clustering. (n.d.). Retrieved from Learnbymarketing.com: https://www.learnbymarketing.com/methods/k-means-clustering/#:~:text=K%2Dmeans%20clustering%20uses%20%E2%80%9Ccentroids,the%20points%20assigned%20to%20it.

Ozan Sener, S. S. (2018). ACTIVE LEARNING FOR CONVOLUTIONAL NEURAL NETWORKS: A CORE-SET APPROACH. Arxiv:1708.00489, 13.

Papers with Code - ImageNet Dataset. (n.d.). Retrieved from Papers with Code: https://paperswithcode.com/dataset/imagenet

Prior probability - Wikipedia. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Prior_probability

Remus Pop, P. F. (2018). DEEP ENSEMBLE BAYESIAN ACTIVE LEARNING: ADDRESSING THE MODE COLLAPSE ISSUE IN MONTE. Arxiv:1811.03897, 15.

Rosebrock, A. (2017, March 20). ImageNet: VGGNet, ResNet, Inception, and Xception with Keras - PyImageSearch. Retrieved from PyImageSearch: https://pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/

sklearn.cluster.KMeans. (n.d.). Retrieved from scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

sklearn.manifold.TSNE. (n.d.). Retrieved from scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

T, S. (2019, Jan 11). Retrieved from Towards Data Sceince: https://towardsdatascience.com/entropy-how-decision-trees-make-decisions-2946b9c18c8#:~:text=Entropy%20is%20measured%20between%200,very%20high%20level%20of%20disorder.

Ultralytics. (n.d.). PyTorch. Retrieved from Pytorch.org: https://pytorch.org/hub/ultralytics_yolov5/

William H. Beluch, T. G. (2018). The Power of Ensembles for Active Learning in Image Classification. IEEE/CVF Conference on Computer Vision and Pattern Recognition, 10.

Yarin Gal, R. I. (2017). Deep Bayesian Active Learning with Image Data. Arxiv:1703.02910, 10.

YOLOv5 Documentation. (n.d.). Retrieved from Docs.ultralytics.com: https://docs.ultralytics.com/

Zhdanov, F. (2019). Diverse mini-batch Active Learning. Arxiv:1901.05954, 9.

List of Figures

Appendix A

Yolov5 feature extraction

```
def forward(self, x):
       # save features
       path = "/home/afarooq/ali/yolov5/features/"
       filelist = []
       for f in os.listdir(path):
              file=path+f
              if (os.stat(file).st_size == 132):
                     filelist.append(f)
       filelist.sort()
       if len(filelist) > 0:
              np.save(path + filelist[0], x[0].cpu().detach().numpy())
              np.save(path + filelist[1], x[1].cpu().detach().numpy())
              np.save(path + filelist[2], x[2].cpu().detach().numpy())
              np.save(path + filelist[3], x[3].cpu().detach().numpy())
              nam = filelist[0].rsplit('.', 1)[0]
              res = nam.rsplit('_',2)
       z = []  # inference output
       count = 0
       for i in range(self.nl):
              x[i] = self.m[i](x[i])  # conv
              print("conv x", x[i].size())
              if len(filelist) > 0:
                     outfile = path + res[0] + "_" + str(count) + "_" +
                                   "conv_features" + ".npy"
                     np.save(outfile, x[i].cpu().detach().numpy())
                     count = count + 1
              bs, _, ny, nx = x[i].shape  # x(bs,255,20,20) to x(bs,3,20,20,85)
              x[i] = x[i].view(bs, self.na, self.no, ny, nx).permute(0, 1, 3, 4,
                                                                  2).contiguous()
              if not self.training:  # inference
                     if self.onnx_dynamic or self.grid[i].shape[2:4] !=
                            x[i].shape[2:4]:
                            self.grid[i], self.anchor_grid[i] =
                                                self._make_grid(nx, ny, i)
                     y = x[i].sigmoid()
                     if self.inplace:
                            y[..., 0:2] = (y[..., 0:2] * 2 - 0.5 + self.grid[i]) *
                                          self.stride[i]  # xy
                            y[..., 2:4] = (y[..., 2:4] * 2) ** 2 *
                                          self.anchor_grid[i]  # wh
                     else:
                            xy = (y[..., 0:2] * 2 - 0.5 + self.grid[i]) *
                                          self.stride[i]  # xy
                            wh = (y[..., 2:4] * 2) ** 2 * self.anchor_grid[i]
                            y = torch.cat((xy, wh, y[..., 4:]), -1)
                     z.append(y.view(bs, -1, self.no))
       return x if self.training else (torch.cat(z, 1), x)
```

InceptionV3 model retraining

```
def retrain_inception_model(self, epoch):
      pre_model = InceptionV3(weights='imagenet', include_top=False,
                              input_shape = (150, 150, 3))
      x = pre_model.output
      x = GlobalAveragePooling2D()(x)
      x = Dense(1024, activation="relu", name = "features1")(x)
      x = Dropout(0.5)(x)
      x = Dense(512, activation="relu", name = "features2")(x)
      predictions = Dense(3, activation="softmax")(x)
      self.model_ = Model(inputs=pre_model.input, outputs=predictions)
      for layer in self.model_.layers[:52]:
            layer.trainable = False
      # compile the model
      self.model_.compile(optimizer=SGD(lr=0.0001, momentum=0.9)
                          , loss='categorical_crossentropy'
                          , metrics=['accuracy'])
      train_datagen =  ImageDataGenerator(
                          rotation_range=40,
                          width_shift_range=0.2,
                          height_shift_range=0.2,
                          shear_range=0.2,
                          zoom_range=0.2,
                          horizontal_flip=True)
      val_datagen = ImageDataGenerator(
                          rotation_range=40,
                          width_shift_range=0.2,
                          height_shift_range=0.2,
                          shear_range=0.2,
                          zoom_range=0.2,
                          horizontal_flip=True)
      self.x_train, self.x_val, self.y_train, self.y_val =
            train_test_split(self.train_images, self.train_image_labels
                              , test_size = 0.15, random_state = 41)
      self.x_train = np.asarray(self.x_train)
      self.y_train = np.asarray(self.y_train)
      self.y_train = tf.one_hot(self.y_train, depth=3)
      self.x_val = np.asarray(self.x_val)
      self.y_val = np.asarray(self.y_val)
      self.y_val = tf.one_hot(self.y_val, depth=3)
      train_generator = train_datagen.flow(self.x_train, self.y_train,20)
      val_generator = val_datagen.flow(self.x_val, self.y_val, 20)
      self.hist = self.model_.fit(train_generator
                              , validation_data = val_generator
                              , epochs= 50
                              , verbose = 1)
      self.model_.save(self.modelname)
```

Kmeans and distance calculation and plot

```
def kmeans(self, epoch):
       clusters = KMeans(self.k, random_state = 40)
       clusters.fit_predict(self.train_image_features)
       self.centroids = clusters.cluster_centers_
       self.labels = clusters.labels_
       image_distance = pd.DataFrame([])
       for center in range(len(self.centroids)):
             distance = []
             for feature in range(len(self.train_image_features)):
                   temp = self.centroids[center] - self.train_image_features[feature]
                   distance += [np.linalg.norm(temp)]
             image_distance["class"+str(center)] = distance
       self.class0_distance = image_distance.loc[:,'class0']
       self.class1_distance = image_distance.loc[:,'class1']
       self.class2_distance = image_distance.loc[:,'class2']
       self.class0_distance = self.class0_distance.values
       self.class1_distance = self.class1_distance.values
       self.class2_distance = self.class2_distance.values
       plt.figure(figsize=(30,20))
       plt.tight_layout()
       plt.xlabel("Distance jellyfish_aurelia")
       plt.ylabel("Distance fish_cod")
       plt.title("distance class jellyfish_aurelia and fish_cod")
       scatter = plt.scatter(self.class0_distance, self.class1_distance
                          , c=self.train_image_labels
                          , cmap=matplotlib.colors.ListedColormap(self.colors))
       plt.legend(handles=scatter.legend_elements()[0], labels=self.label)
       plt.savefig("/home/afarooq/ali/yolov5/exp_results/kmeans_class01 " +
                   str(epoch) + ".png")
       plt.figure(figsize=(30,20))
       plt.tight_layout()
       plt.xlabel("Distance jellyfish_aurelia")
       plt.ylabel("Distance fish_unspecified")
       plt.title("distance class jellyfish_aurelia and fish_unspecified")
       scatter = plt.scatter(self.class0_distance, self.class2_distance
                          , c=self.train_image_labels
                          , cmap=matplotlib.colors.ListedColormap(self.colors))
       plt.legend(handles=scatter.legend_elements()[0], labels=self.label)
       plt.savefig("/home/afarooq/ali/yolov5/exp_results/kmeans_class02 " +
                   str(epoch) + ".png")
       plt.figure(figsize=(30,20))
       plt.tight_layout()
       plt.xlabel("Distance fish_cod")
       plt.ylabel("Distance fish_unspecified")
       plt.title("distance class fish_cod and fish_unspecified")
       scatter = plt.scatter(self.class1_distance, self.class2_distance
                          , c=self.train_image_labels
                          , cmap=matplotlib.colors.ListedColormap(self.colors))
       plt.legend(handles=scatter.legend_elements()[0], labels=self.label)
       plt.savefig("/home/afarooq/ali/yolov5/exp_results/kmeans_class12 " +
                   str(epoch) + ".png")
```

T-SNE plot

```
def sns_plot(self, epoch):
       palette = sns.color_palette("bright", 3)
       tsne = TSNE()
       X_embedded = tsne.fit_transform(self.train_image_features)
       plt.figure(figsize=(30,20))
       plt.tight_layout()
       sns.scatterplot(X_embedded[:,0], X_embedded[:,1],
       hue=self.train_labels_sns, legend="full", palette=palette)
       plt.title("data_Clustering")
       plt.xlabel("component 1")
       plt.ylabel("component 2")
       plt.savefig("/home/afarooq/ali/yolov5/exp_results/
              sns_clusters_feature " + str(epoch) + ".png")
```

Appendix B

Active learning trained model Kmeans distance plot for all classes

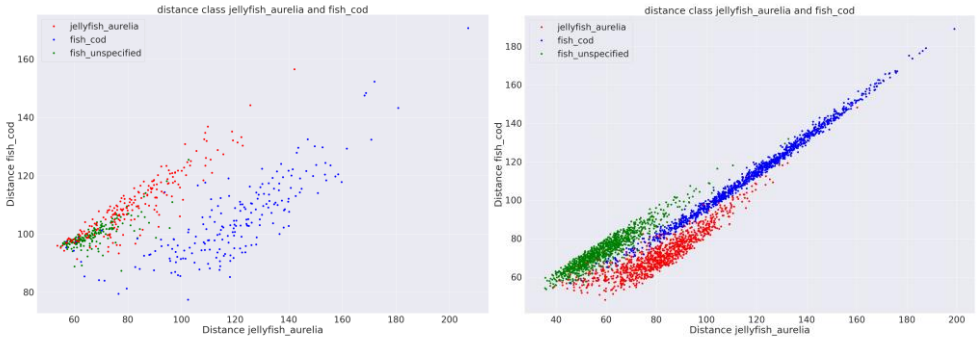

Figure 37: Training data distance plot of class jellyfish_aurelia and fish_cod for active learning first and last iteration

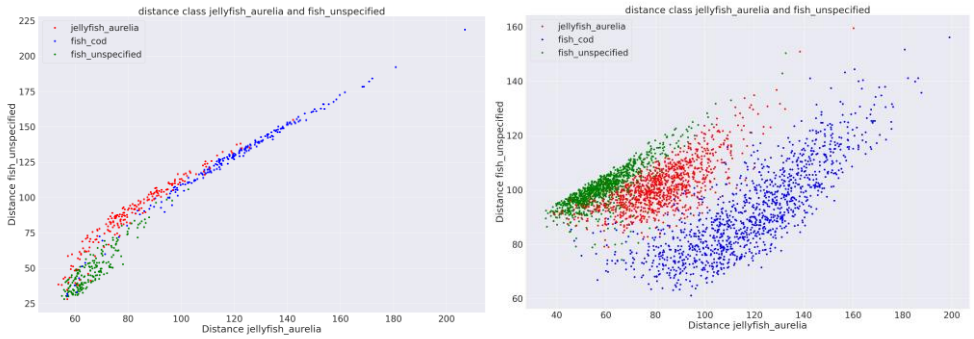

Figure 38: Training data distance plot of class jellyfish_aurelia and fish_unspecified for active learning first and last iteration

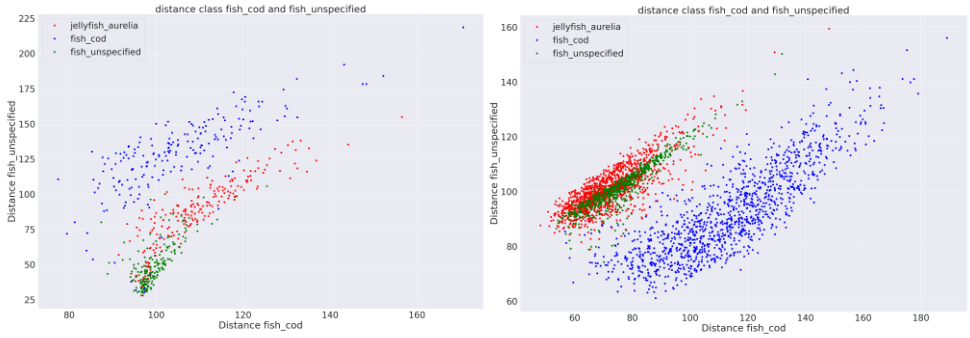

Figure 39: Training data distance plot of class fish_cod and fish_unspecified for active learning first and last iteration

Active learning trained model Sns plot

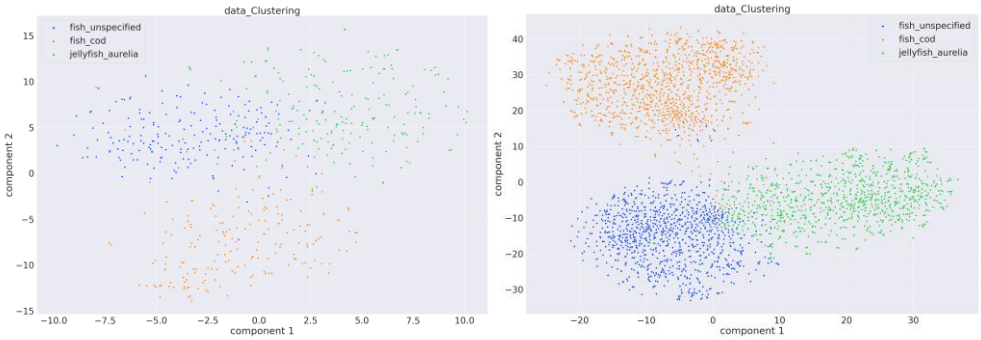

Figure 40: Training data Sns plot for active learning first and last iteration

Active learning trained model PCA Kmeans distance plot for all classes

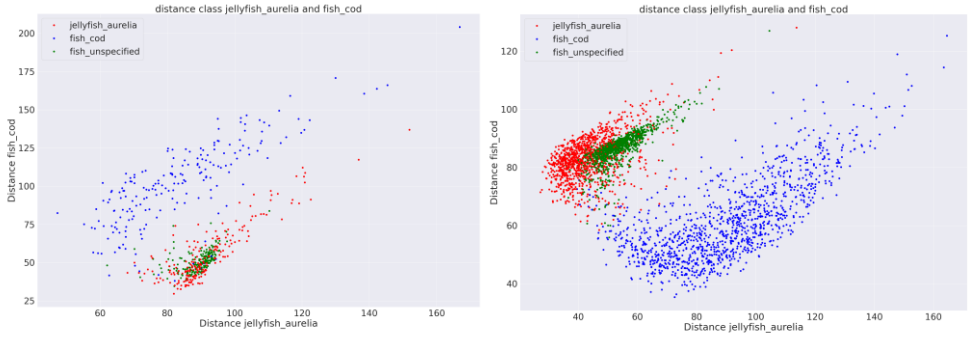

Figure 41: Training data PCA distance plot of class jellyfish_aurelia and fish_cod for active learning first and last iteration

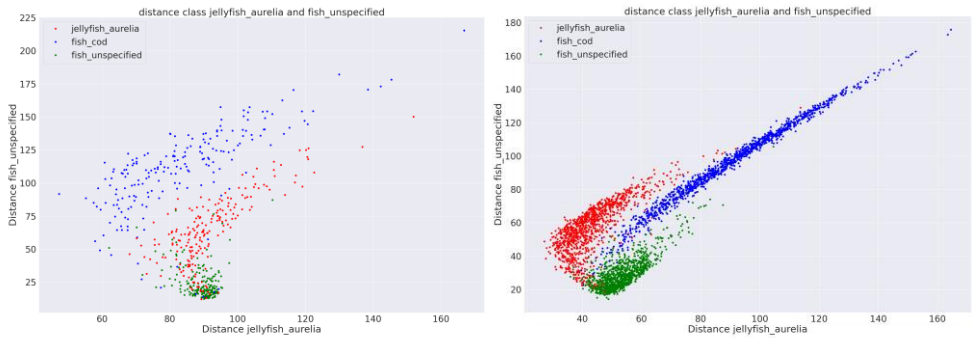

Figure 42: Training data PCA distance plot of class jellyfish_aurelia and fish_unspecified for active learning first and last iteration

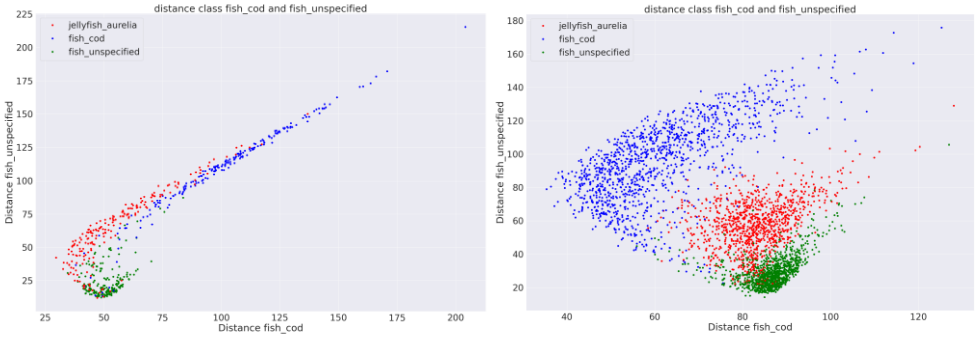

Figure 43: Training data PCA distance plot of class fish_cod and fish_unspecified for active learning first and last iteration

Active learning trained model PCA Sns plot

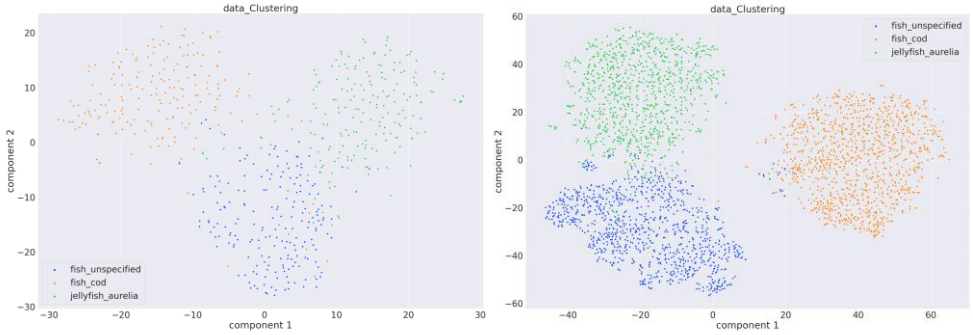

Figure 44: Training data PCA Sns plot for active learning first and last iteration