# APPLICATION OF DEEP LEARNING

TO

# FISH RECOGNITION

A Project

Presented to

The Faculty of the Graduate School

At the University of Missouri

---

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

---

Implemented and Defended by:


## Fuming Xiang


Prof. Yi Shang, Advisor

May 2018

# TABLE OF CONTENTS

# List of Figures

# List of Tables

# ACKNOWLEDGEMENTS

Firstly, I would like to thank Prof. Yi Shang for his mentorship, guidance, and support throughout my graduate school and through this research project. I would also like to thank Joel for the opportunity to collaborate with the Missouri Department of Conservation and their feedback on this project.

I would also like to thank everyone from the Computer Science lab, especially Guang Chen, Peng Sun, Yang Liu, and Nickolas Wergeles, for sharing their experiences and being an integral part of this project. I would also like to show appreciation to Jodie Lenser, Shirley Holdmeier for guiding me through my application process to Mizzou as well as advising me on my courses.

Finally, I would like to thank my family. Without their support, trust and belief in me, I would not be where I am today. I hope I have done you proud.

# ABSTRACT

Over the past few years, deep learning has been widely used and obtained very good results in image recognition. In this project, several state-of-the-art deep learning models and their combinations have been applied to fish recognition in images, in particular 9 common species of fish in Missouri rivers. Four different data processing and machine learnings pipelines have been developed and extensive experiments have been conducted to evaluate their performances. The deep convolutional neural network (CNN) models used in these pipelines include SSD, VGG16, ResNet50, etc. The four pipelines are image-based, instance-based, instance rotation based, and ensemble, with increasing complexity. Without doing any preprocessing, the image-based pipeline takes an entire image as input to classify the image into one of the target classes using deep CNNs. This pipeline achieved up to 75.57% classification accuracy on our test dataset. The instance-based pipeline consists of object detection by one deep CNN followed by classification by another deep CNN. This method achieved up to 80.03% accuracy on our test dataset. The instance rotation based pipeline adds a deep CNN to do pose estimation between object detection and classification. The posture-adjusted fish image is used as the input to the classification model, which help the pipeline to achieve up to 82.83% accuracy on the same dataset. Finally, the ensemble pipeline is a combination of two instance rotation based pipelines. The difference of these two instance rotation based pipelines is in the classification model: one is VGG16 and the other ResNet50. The ensemble pipeline achieved up to 87.22% accuracy, outperforming all other pipelines significantly.

# INTRODUCTION

The project is proposed by Missouri Department of Conservation (MDC), to solve the issue that counting and classifying fish in image manually is a huge amount of work and time-consuming. Since the Convolutional neural network performance better and better, and for now, the CNN in some case could get higher accuracy than human recognition, we are going to use existing CNN models to solve the fish classification problem. The primary task of this project is aiming to classify 9 different species of fish in Missouri river.

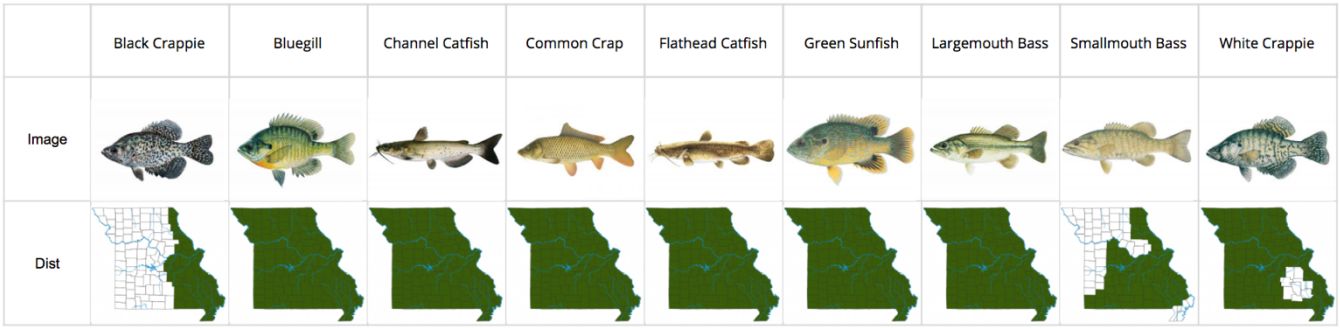


*Figure 1 Fish Species and Distribution in Missouri*

The classification task is taking a or a batch of RGB images, and return the species or the probability of fish species.

All the data used in this project is download from ImageNet, which is a large image dataset with categorical labels, In the dataset, there are 4842 images distribute in 9 classes, and each image may contain several fish instances, but only one species.

| Abbreviation | BCP | BLG | CCF | CMC | FCF | GSF | LMB | SMB | WCP |
|---|---|---|---|---|---|---|---|---|---|
| Fish (9) | Black Crappie | Bluegill | Channel Catfish | Common Crappie | Flathead Catfish | Green Sunfish | Largemouth Bass | Smallmouth Bass | White Crappie |
| Images (4842) | 538 | 680 | 606 | 418 | 413 | 574 | 888 | 557 | 168 |

*Table 1 Dataset downloaded from ImageNet*

Among the whole dataset, most of the image contains 1 fish. To visualize the size of fish, we define the ratio by using this formula the ratio equals to the labeled box area of fish, divided by the image area in pixel measurement.

$$Ratio_{fish} = Area_{fish\ box} / Area_{image}$$

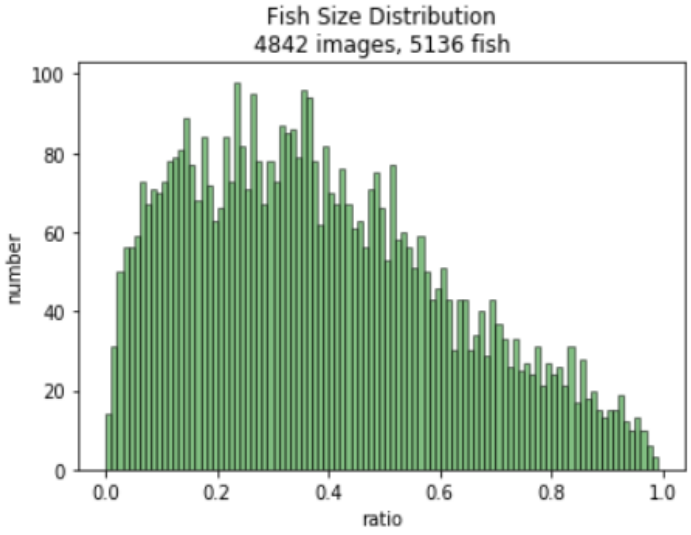


*Figure 2 Fish size ratio distribution*

# 1. RELATED WORKS

## 1.1 Convolutional Neural Network (CNN)

The Convolutional Neural Network is a kind of artificial neural network, which has become a research hotspot in the field of speech analysis and image recognition. Its weight-sharing network structure makes it more similar to biological neural networks, reducing the complexity of the network model and reducing the number of weights. This advantage is more obvious when the input of the network is a multi-dimensional image so that the image can be directly used as the input of the network, which avoids the complicated feature extraction and data reconstruction process in the traditional recognition algorithm. A convolutional network is a multi-layer perceptron specially designed to identify two-dimensional shapes. This network structure is highly invariant to translation, scaling, tilting, or other forms of deformation.

CNN is the first learning algorithm to train multi-layer network structures successfully. It uses spatial relationships to reduce the number of parameters that need to be learned to improve the training performance of general back propagation algorithms. CNNs are proposed as a deep learning framework to minimize data preprocessing requirements. In CNN, a small part of the image (local receptive area) is used as the lowest level input to the hierarchical structure. The information is then transmitted to different layers. Each layer passes a digital filter to obtain the most significant features of the observed data. This method can obtain significant features of the observation data that are invariant to translation, scaling, and rotation, because the local perception region of the image allows the neuron or the processing unit to access the most basic features, such as oriented edges or corner points.

## 1.2 VGG16

In this project, different CNNs ware used, such as VGG16, ResNet50, and SSD Caffe. The original title of the VGG paper was “VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION”. The paper was published on ICLR 2015. VGG16 and VGG19 are two very good DCNN (Deep Convolutional Neural Network) proposed by this article. This series of models achieved best-performance in ILSVRC-2013 and achieved good results in other DCNN-based work (eg FCN). Compared to ALEXNET, VGG has a more accurate estimate of the picture and more space. VGG16 has 3 convolutional blocks with max pooling, and 3 fully connected layers with ReLU functions, finally following a softmax function. Each convolutional block contains 2 or 3 convolutional layers.

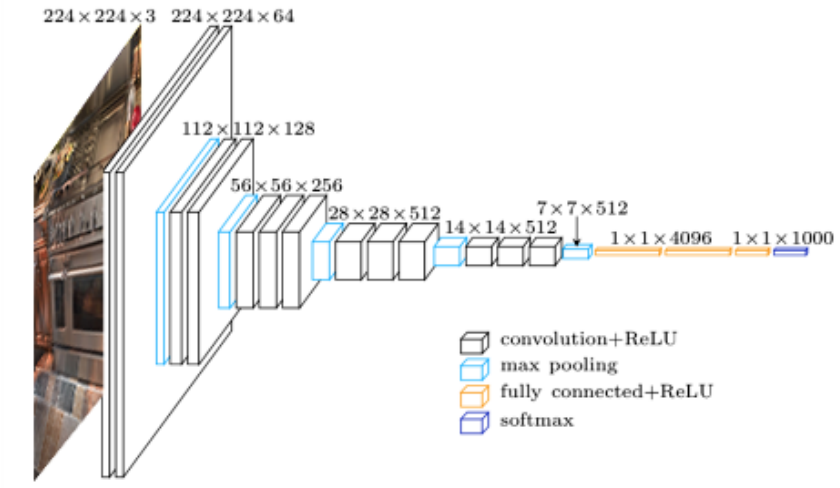


*Figure 3 VGG 16 Architecture*

## 1.3 ResNet 50

An original paper of the ResNet model: Deep Residual Learning for Image Recognition. It can be clearly seen that the expressiveness of the network is enhanced as the depth of the network increases. The experiments of He Kaiming and others also proved that the network structure with the

same complexity, the relative deeper network performs better. However, it is not. Regardless of the issue of computational cost, when the depth of the network is deeper, increasing the number of layers will not improve performance, but will cause the gradient diffusion. Under the guidance of this idea, Deep Residual Learning (ResNet) was proposed. ResNet has two basic blocks, one is Identity Block, the input and output dimensions are the same, another basic block is Conv Block, the input and output dimensions are not the same, so cannot be connected in series, its role was initially to change the dimension of the feature vector.

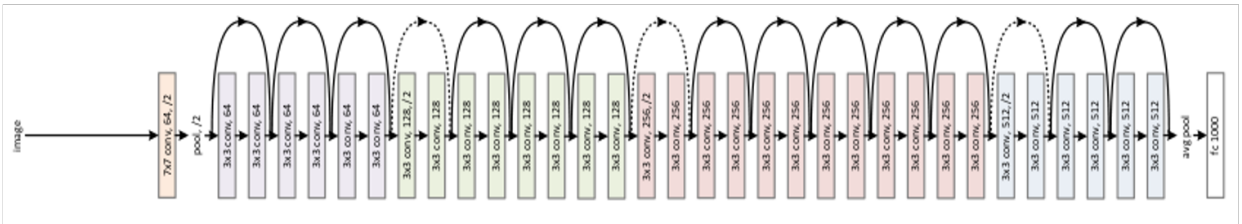


*Figure 4 ResNet 50 Architecture*

## 1.4 SSD Caffe

The SSD algorithm proposed in the paper “SSD: single shot multibox detector”, is an object detection algorithm that directly predicts the coordinates and categories of the bounding box, and does not generate a proposal process. For the detection of objects of different sizes, the traditional approach is to convert the images into different sizes, then process them separately, and finally synthesize the results. In this paper, the ssd can synthesize different convolutional layer feature maps to achieve the same effect. The main network structure of the algorithm is VGG16, which transforms two fully connected layers into a convolutional layer and then adds four convolutional layers to

construct a network structure. The output of five different convolutional layers is convolved with two 3*3 convolution kernels, one for the output classification, each default box generates 21 confidences, and the other for localization, each default box generates 4 coordinate values (x, y, w, h). In addition, these 5 convolutional layers also generate the default box (the generated coordinates) via the prior box layer. Finally, the first three calculation results are respectively merged and then passed to the loss layer.

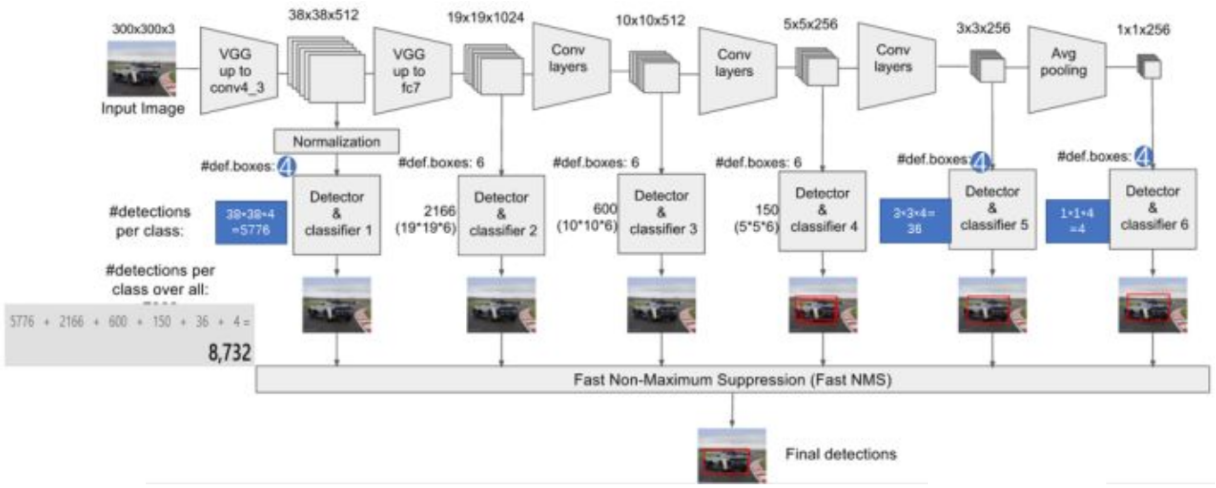


*Figure 5 SSD Caffe Architecture*

# 2. DESIGN AND IMPLEMENTATION

## 2.1 Design

### *2.1.1 Data Preprocessing*

Images are downloaded from ImageNet by the links with keywords. Thus, there would be some data that is not suitable for our project, such as grayscale images, non-fish or missing-fish image, and ambiguity image. What we need to do is to remove these images to clean the dataset for this project.

*Figure 6 Examples of grayscale, non-fish or missing-fish, and ambiguity images*

Since the images downloaded from ImageNet only provide categorical labels, in order to train the network, especially for the object detection and pose estimation, we need to label the fish with an annotation tool: Sloth.

Sloth's purpose is to provide a versatile tool for various labeling tasks in the context of computer vision research. In this project, for every image, the label we need is a position bounding box (red in the figure 7) containing a particular species of fish, and its head and tail coordinates (green points in figure 7).

*Figure 7 Example of labeled image*

### 2.1.2 Image Pipeline

Image Pipeline is the basic pipeline in this project. It takes the non-processed images as the input. For this pipeline, we used VGG16 as the prediction model. Since the VGG16 was trained in ImageNet and the output of the pre-trained model is 1000, in order to use the pre-trained weights to get a higher accuracy, we used pre-trained VGG16 as the basic model and added another dense layer (fully connected layer) with the output 9.

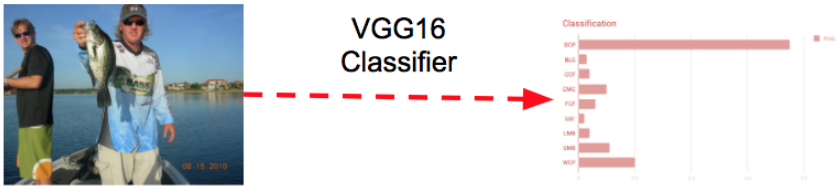


*Figure 8 Image Pipeline*

We randomly split 4842 images into training (80%), validation (10%), and testing (10%) dataset by species.

| | Species | BCP | BLG | CCF | CMC | FCF | GSF | LMB | SMB | WCP | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Image Pipeline Dataset | train (80%) | 431 | 544 | 485 | 335 | 331 | 460 | 711 | 446 | 135 | 3878 |
| | val (10%) | 54 | 68 | 61 | 42 | 41 | 57 | 89 | 56 | 17 | 485 |
| | test (10%) | 53 | 68 | 60 | 41 | 41 | 57 | 88 | 55 | 16 | 479 |

*Table 2 Image Pipeline dataset*

### 2.1.3 Instance Pipeline

However, in our project, like we saw in data overview part, the fish in the image is small, mostly smaller than half of the image size, which means, in the image, there are plenty of useless information, which would influence the performance of classification. Thus, in order to avoid it, fish detection is necessary. In Instance Pipeline, detection is added before the VGG16 classification, which would output the box position of fish, and we do the padding to avoid distortion and crop it.


*Figure 9 Instance Pipeline*

Based on the pipeline we designed, cropped images are needed for the training stage to the classifier. The generated images are coming from the Image Pipeline. We padded and cropped the fish into square instance images. If the border of the cropped image is out of the image border, we should do padding after cropping, and finally, we got 5136 instance images.

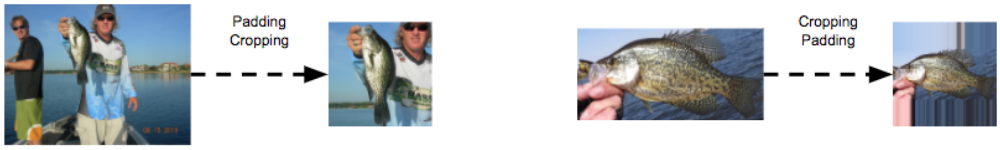


*Figure 10 Instance images generation*

| | Species | BCP | BLG | CCF | CMC | FCF | GSF | LMB | SMB | WCP | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Instance Pipeline Dataset | train (80%) | 507 | 574 | 507 | 366 | 350 | 474 | 747 | 467 | 144 | 4136 |
| | val (10%) | 58 | 69 | 61 | 44 | 41 | 59 | 91 | 58 | 18 | 499 |
| | test (10%) | 56 | 69 | 60 | 44 | 44 | 61 | 92 | 59 | 16 | 501 |

*Table 3 Instance Pipeline dataset*

### *2.1.4 Instance Rotation Pipeline*

In the Instance Pipeline, we did object detection, and image classification to filter out the useless information by cropping the fish out. However, the cropped images still have some useless information which we could filter out: the orientation of fish. Thus, we added a pose estimator after the object detection to align the fish into a particular orientation.

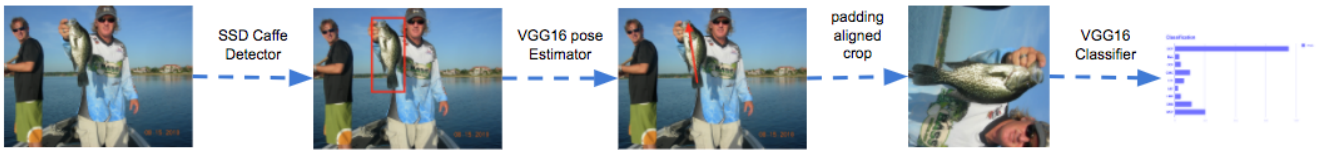


*Figure 11 Instance Rotation Pipeline*

Amount 5136 instance images, we assign the pose of fish into 16 orientations in a circle, and every orientation takes 22.5 degrees.

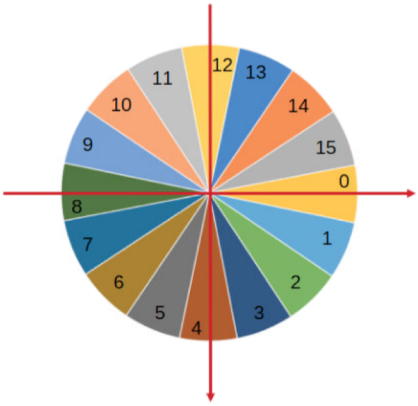


*Figure 12 Orientation split*

To get which orientation that the fish should be assigned to, we used the head and tail coordinates to calculate the vector of each fish.

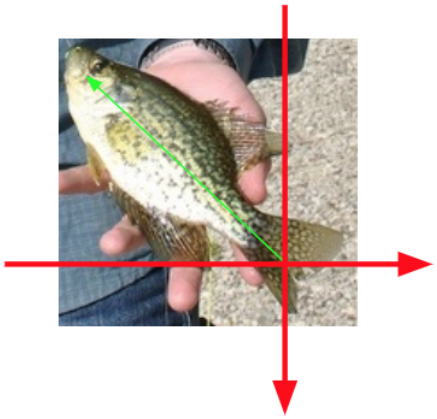

*Figure 13 Orientation vector*

$$vector = coordinate_{head} - coordinate_{tail}$$

For the training, validation, and testing data in pose estimator, we did data augmentation to enlarge the dataset, such as flipping and rotating. Finally, we have 7687 for the estimator dataset.

| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Instance Rotation Pipeline Pose Estimator | train | 776 | 508 | 137 | 116 | 817 | 124 | 140 | 528 | 994 | 404 | 122 | 108 | 793 | 106 | 110 | 376 | 6159 |
| | val | 97 | 63 | 17 | 14 | 102 | 15 | 17 | 66 | 124 | 50 | 15 | 13 | 99 | 13 | 14 | 47 | 766 |
| | test | 96 | 63 | 17 | 14 | 102 | 15 | 17 | 65 | 124 | 50 | 15 | 13 | 99 | 13 | 13 | 46 | 762 |

*Table 4 Pose estimator dataset*

For the training, validation, and testing data in VGG16 classifier, we do the alignment from the Instance Pipeline VGG16 classifier dataset.


*Figure 14 Instance Rotation images generation*

After alignment, we got the training, validation, and testing data for the VGG16 classifier in Instance Rotation Pipeline.

| | Species | BCP | BLG | CCF | CMC | FCF | GSF | LMB | SMB | WCP | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Instance Rotation Pipeline | train (80%) | 507 | 574 | 507 | 366 | 350 | 474 | 747 | 467 | 144 | 4136 |
| | val (10%) | 58 | 69 | 61 | 44 | 41 | 59 | 91 | 58 | 18 | 499 |
| | test (10%) | 56 | 69 | 60 | 44 | 44 | 61 | 92 | 59 | 16 | 501 |

*Table 5 Instance Rotation Pipeline dataset*

### *2.1.5 Ensemble Pipeline*

In the results of a massive machine learning competition, the best result is usually the ensemble model rather than the single model. For example, ILSVRC 2015's highest-scoring single model structure achieved the 13th place. The 1st to 12th have used different types of model ensemble.

There are many different types of ensemble model: one of them is stacking. This type is more general and can theoretically characterize any other ensemble technique. For this pipeline, I will use the purest form of stacking, which involves averaging the ensemble model outputs. Since the averaging process does not include any parameters, this process does not require training (only the trained model is needed).

In the Ensemble Pipeline, we train another classifier by using ImageNet pre-trained model: ResNet 50, and take the weight average to get the ensemble result.


*Figure 15 Ensemble Pipeline*

When we got the prediction results for VGG16 and ResNet50, we applied weight average to get the final result for this pipeline.

$$Pred_{ensemble} = ratio_{VGG} * pred_{VGG} + (1 - ratio_{VGG}) * pred_{ResNet}$$

The only difference for the Ensemble Pipeline is that we need to train the ResNet50 model similarly as the training stage for the VGG16 model in Instance Rotation Pipeline. Thus, the dataset is same.

| | Species | BCP | BLG | CCF | CMC | FCF | GSF | LMB | SMB | WCP | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Ensemble Pipeline | train (80%) | 507 | 574 | 507 | 366 | 350 | 474 | 747 | 467 | 144 | 4136 |
| | val (10%) | 58 | 69 | 61 | 44 | 41 | 59 | 91 | 58 | 18 | 499 |
| | test (10%) | 56 | 69 | 60 | 44 | 44 | 61 | 92 | 59 | 16 | 501 |

*Table 6 Ensemble Pipeline dataset*

## 2.2 Implementation

### *2.2.1 Configuration*

The project's development environment is Linux, Ubuntu 16.04, using python as the only development language. Python dependency package version information is as follows.

| Package Name | Version | Description |
|---|---|---|
| h5py | 2.7.1 | tools for *.h5 file (keras weights) |
| imutils | 0.4.3 | image processing tools |
| jupyter | 1.0.0 | python GUI (in browser) |
| Keras | 1.2.0 | high level machine learning API |
| matplotlib | 2.1.0 | data visualization |
| numpy | 1.14.0 | python scientific computing library |
| pandas | 0.20.3 | python data analysis library |
| pillow | 3.1.2 | image processing tools |
| pip | 9.0.1 | python package management tool |

| | | |
|---|---|---|
| progressbar | 2.3.0 | progress bar view components |
| python | 2.7 | language |
| scikit-image | 0.13.1 | image processing tools |
| scikit-learn | 0.19.1 | machine learning toolkit |
| scipy | 1.0.0 | python algorithm library and math toolkit |
| six | 1.11.0 | library for compatibility with python 2 and python 3 |
| sloth | 1.0 | annotation tools |
| Theano | 0.9.0 | Used to define, optimize, and efficiently solve analog estimation problems of multi-dimensional array data corresponding mathematical expressions |

*Table 7 Development environment configuration*

### *2.2.2 Evaluation Metric*

In the whole project, we use object detection, and classification (pose estimator and fish classifier). In object detection, precision and recall are the primary evaluation metrics, and for classification, calculating the accuracy is a simple method to evaluate the model.

#### *2.2.2.1 Object Detection Accuracy*

Traditionally in the evaluation of object detection algorithms, for a single detection file and its corresponding ground truth file, two values, recall, and precision, can be calculated. Usually, in the object detection task, we use confusion matrix to get the prediction results. A confusion matrix is a summary of prediction results on a detection or classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. To calculate the confusion matrix:

1. You need a test dataset or a validation dataset with expected outcome values.

2. Make a prediction for each row in your test dataset.
3. From the expected outcomes and predictions count:
    a. The number of correct predictions for each class.
    b. The number of incorrect predictions for each class, organized by the class that was predicted.

Efficiently, scikit-learn provides a function to get confusion matrix by inputting the label and prediction in order. For object detection or binary classification, for example, we have the confusion matrix as follows:

| | | Ground Truth | |
|---|---|---|---|
| | | class | Non-class |
| Predicted Class | class | True Positives | False Positives |
| | Non-class | False Negatives | True Negatives |

*Figure 16 Confusion Matrix Example*

- True Positives (TP): These are cases in which we predicted yes, and their label is yes.
- True Negatives (TN): We predicted no, and their label is no.
- False Positives (FP): We predicted yes, but their label is no.
- False Negatives (FN): We predicted no, but their label is yes.

$$Precision = TP / (TP + FP)$$

$$Recall = TP / (TP + FN)$$

Intuitively, recall tells us how many of our objects have been detected, and precision gives us information on the number of false alarms. Both are higher if better and should be close to 1 for perfect systems.

In order to be able to calculate these two measures (recall and precision), an evaluation tool needs to find out two things:

1. for each ground truth rectangle, we need to know whether it has been correctly detected or not
2. for each detected rectangle, we need to know whether it corresponds to a valid ground truth rectangle or not

In order to measure whether the object is correctly detected, we use Intersection over Union.

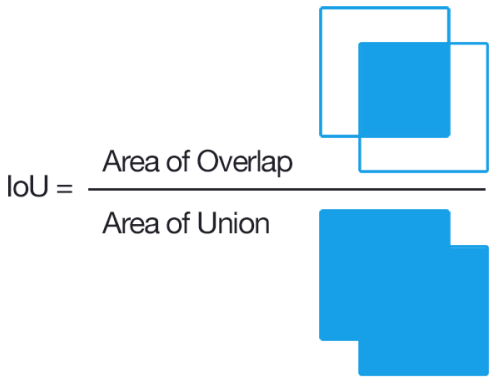


*Figure 17 Intersection over Union*

Examining this equation, you can see that Intersection over Union is simply a ratio. In the numerator, we compute the area of overlap between the predicted bounding box and the ground-truth bounding box. The denominator is the area of union, or more simply, the area encompassed by

both the predicted bounding box and the ground-truth bounding box. Dividing the area of overlap by the area of union yields our final score — the Intersection over Union.

In all reality, it's extremely unlikely that the (x, y)-coordinates of our predicted bounding box are going to exactly match the (x, y)-coordinates of the ground-truth bounding box. Due to varying parameters of our model (image pyramid scale, sliding window size, feature extraction method, etc.), a complete and total match between predicted and ground-truth bounding boxes is simply unrealistic. Because of this, we need to define an evaluation metric that rewards predicted bounding boxes for heavily overlapping with the ground-truth:


*Figure 18 Threshold example of Intersection over Union*

As you can see, predicted bounding boxes that heavily overlap with the ground-truth bounding boxes have higher scores than those with less overlap. This makes Intersection over Union an excellent metric for evaluating custom object detectors.

#### 2.2.2.2 Classification Accuracy

Accuracy is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions our model got right. Formally, accuracy has the following definition:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

Usually, in classification task, we use confusion matrix similarly to the object detection but using multiple-class.

By looking at the confusion matrix, we could calculate the accuracy easily.

$$Accuracy = \frac{Trace_{matrix}}{Sum_{matrix}}$$

#### 2.2.2.3 Pose Estimation Accuracy

Pose Estimation is a classification task, but we apply two different measurements to calculate the accuracy of the model:

1. Traditional classification accuracy measurement.
2. Offsetting classification accuracy measurement.

In traditional classification accuracy measurement, we could calculate the confusion matrix, and get the accuracy by using the above formula. However, in some cases, we allow some offset to the rotated fish, which means the nearby orientation is acceptable.

$$Accuracy\ 0 = \frac{True\ Positives}{Number\ of\ predictions}$$

$$Accuracy\ 1\ =\ \frac{Offsetting\ True\ Positives}{Number\ of\ predictions}$$

Where:

$True\ Positives$: $(prediction\ ==\ ground\ truth)$ and

$Offsetting\ True\ Positives$: $(prediction\ ==\ ground\ truth\ ||\ prediction\ ==\ ground\ truth_{adjoining}\ )$

Usually, in Numpy, we could do it easily by using the following code (cm is the confusion matrix):

```
accuracy0 = np.trace(cm) / float(np.sum(cm))
accuracy1 = float(np.trace(cm)+np.trace(np.roll(cm,-1,0))+np.trace(np.roll(cm,+1,0)))/np.sum(cm)
```

*Figure 19 Python code for accuracy calculation*

For example, in the following figure, if the ground truth orientation is 14, both of those two measurements are acceptable.

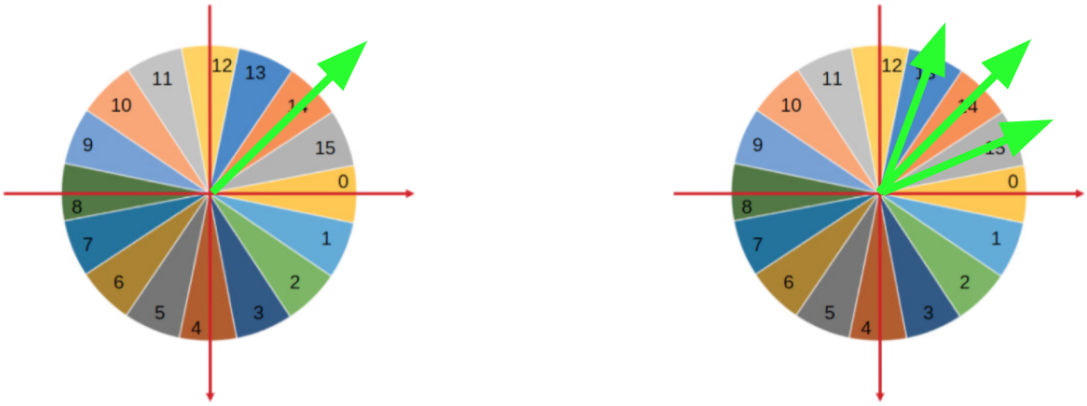


*Figure 20 Accuracy_0 and Accuracy_1 orientation distribution*

### 2.2.3 Image Pipeline

In the Image Pipeline, as we mentioned, we used VGG16 pre-trained model and added another fully connected layer with output 9. Since the pre-trained model is trained on ImageNet dataset, and our data is also coming from ImageNet, we only train last two layers (transfer learning).

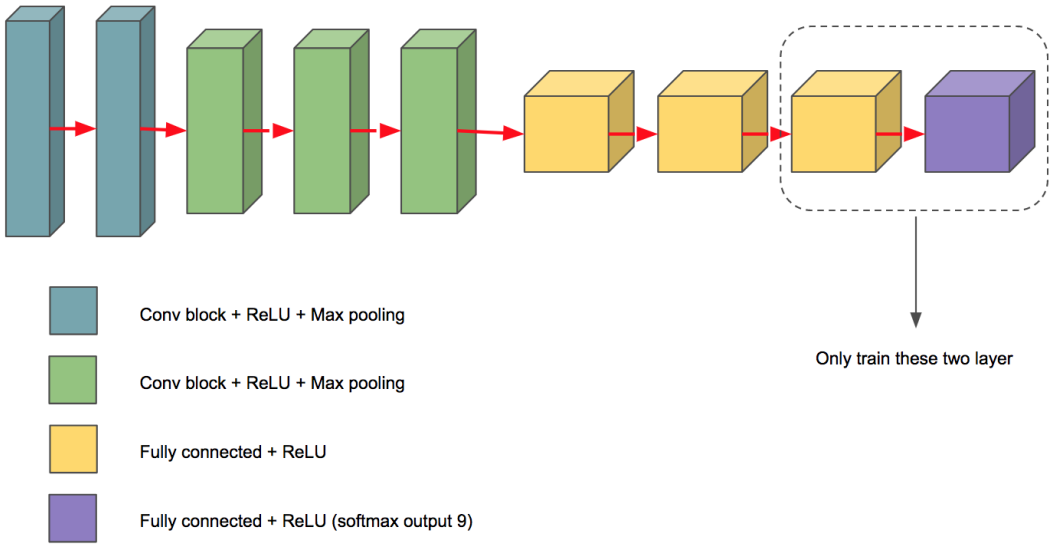


*Figure 21 Modified VGG 16 Architecture*

For the loss function, we chose cross entropy, which is usually used in multi-class classification.

$$l(\hat{y}, y) = -y^T log\hat{y}$$

In Keras, it is easy to set up the loss function by using the following code

```
model.compile(optimizer=SGD(lr=1e-4, momentum=0.9, nesterov=True),
              loss='categorical_crossentropy', metrics=['accuracy'])
```

*Figure 22 Compile model in Keras with cross-entropy loss function*

The input of the network is a batch of RGB images with the size of (3, 224, 224). 3 is for R, G, and B channel, and 224 is the input of traditional VGG structure. Any images should be resized to 224 by 224 before it is given to the network. Since we used Theano as the backend in Keras, the channel

should be the first parameter. Anyone who wants to use Tensorflow as the Keras backend, the only modification is using the (224, 224, 3) as the input size. Usually, in training step of traditional deep learning problem, choosing 1e-4 as the initial learning rate is better, and when the loss value no longer declines in a certain epoch, or the accuracy of the verification dataset no longer rises, we can reduce the learning rate by 0.1 to achieve better results.

In this pipeline, we used (3, 224, 224) as the input, initial learning rate 1e-4, batch size 64, optimizer SGD, and trained the network by reducing the learning rate by 0.1 if the validation accuracy does not improve in 30 epochs.

| input | lr | # epoch | batch size | time / epoch | total time | last update epoch | loss | val_loss | val_Acc |
|---|---|---|---|---|---|---|---|---|---|
| 3,224,224 | 1e-4 | 56 | 64 | 39 s | 36 min 24 s | 25 | 0.1155 | 0.9666 | 0.77 |

*Table 8 Image Pipeline training phase – VGG16 Classifier*

After 25 epochs, we achieve 77% accuracy on validation dataset with the training loss 0.1155 and validation loss 0.9666. The figures show the curve of loss and accuracy on training and validation dataset.

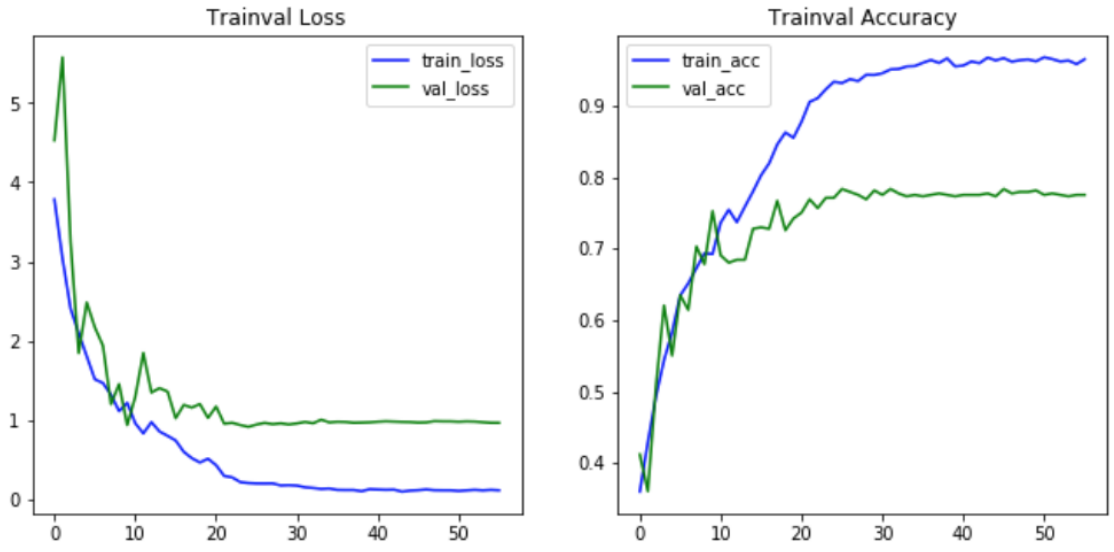


*Figure 23 Image Pipeline loss and accuracy curve – VGG16 Classifier*

For the training dataset, we get the following confusion matrix with the accuracy 0.9866

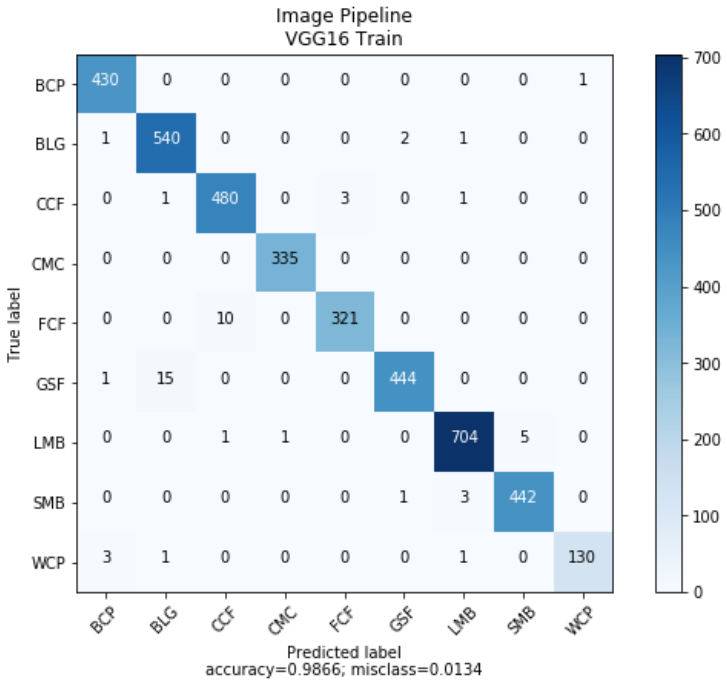


*Figure 24 Image Pipeline training confusion matric – VGG16 Classifier*

For the validation dataset, we get the following confusion matrix with the accuracy 0.7711

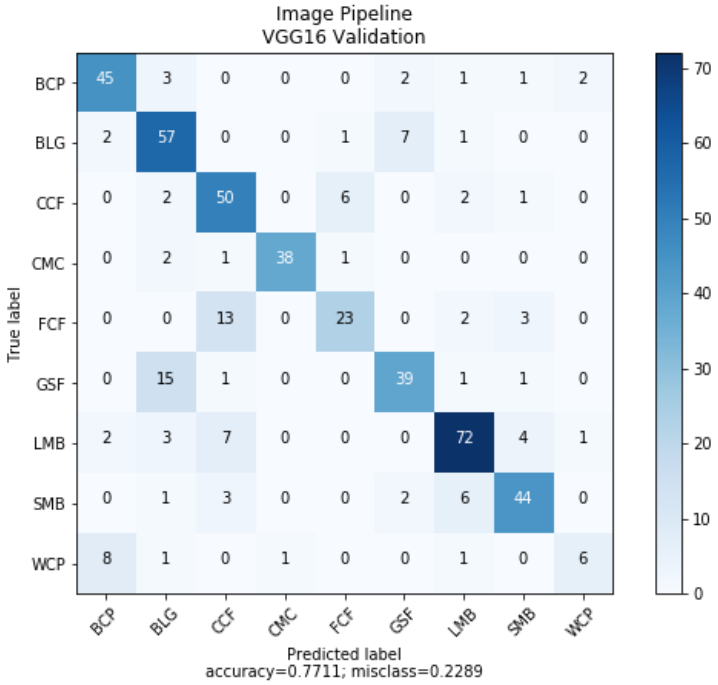


*Figure 25 Image Pipeline validation confusion matric – VGG16 Classifier*

For the test dataset, we get the following confusion matrix with the accuracy 0.7557

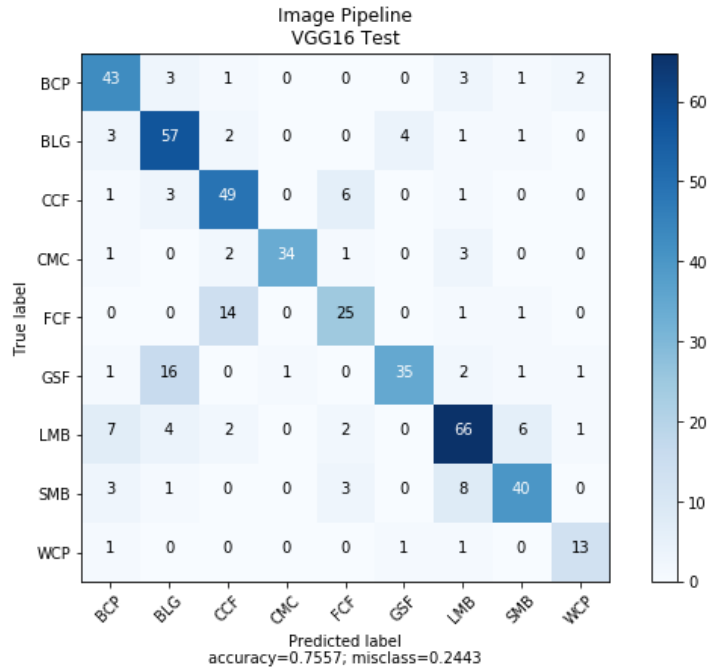


*Figure 26 Image Pipeline testing confusion matric – VGG16 Classifier*

### 2.2.4 Instance Pipeline

In the Instance Pipeline, there are two models which should be trained on the instance dataset: SSD Caffe detector and VGG16 classifier.

For the SSD Caffe detector, we resize the images to 512 by 512 and use the size (512, 512, 3) as the input with bounding box labels. Training phase took 20,000 iterations in 12 hours. In SSD Caffe, the loss function is the combination of confidence loss and box location loss.

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

Where N is the number of matched box with ground truth box.

After 20,000 iterations, we got the final loss 1.28

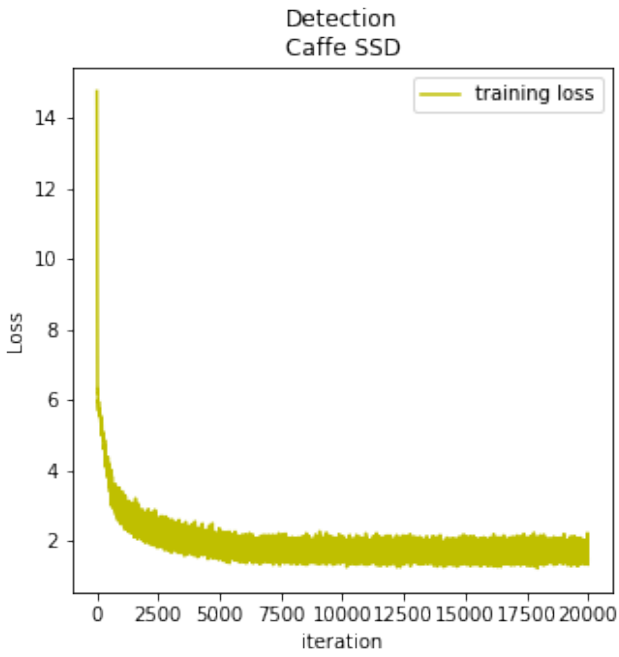


*Figure 27 SSD Caffe training loss*

On test dataset, we got the prediction results without confidence threshold. In order to choose the threshold that is best for the model to get relatively high precision and recall, we use the precision-recall curve to measure the performance. For each confidence threshold from 0.0 to 1.0, we could have a point with the x of recall and y of precision.

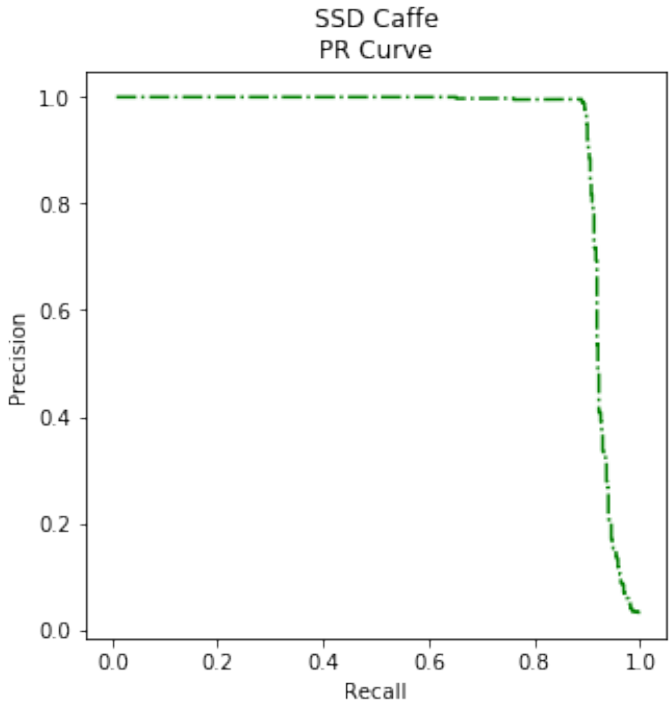


*Figure 28 SSD Caffe precision-recall curve on the test dataset*

The AUC (Area Under Curve) of this figure is the average precision. When the confidence threshold is 0.5, we could have a relatively high precision and recall with the average precision 0.9275

For the VGG16 classifier in instance dataset, we use the similar method to modify the network structure: using ImageNet pre-trained model, and adding a fully connected layer with output 9, freeze the layers except for the last two layers. We used (3, 224, 224) as the input, initial learning rate 1e-4,

batch size 64 and trained the network by reducing the learning rate by 0.1 if the validation accuracy does not improve in 30 epochs.

| input | lr | # epoch | batch size | time / epoch | total time | last update epoch | loss | val_loss | val_Acc |
|---|---|---|---|---|---|---|---|---|---|
| 3,224,224 | 1e-4 | 114 | 64 | 37 s | 1 h 10 m 18 s | 83 | 0.3386 | 0.4751 | 0.8236 |

*Table 9 Instance Pipeline training phase – VGG16 Classifier*

After 83 epochs, we achieved 82.36% accuracy on validation dataset with the training loss 0.3386 and validation loss 0.4751. The figures show the curve of loss and accuracy on training and validation dataset.

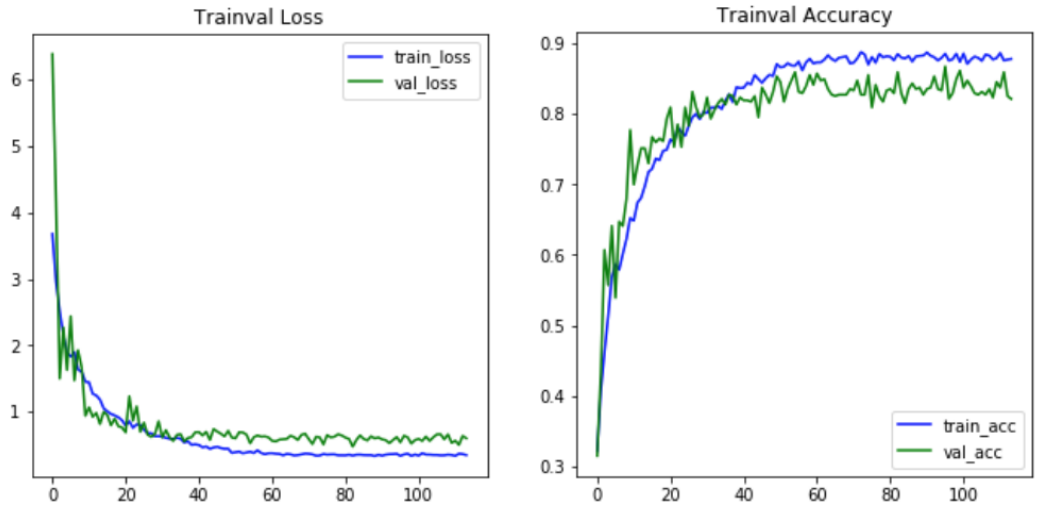


*Figure 29 Instance Pipeline loss and accuracy curve – VGG16 Classifier*

For the training dataset, we get the following confusion matrix with the accuracy 0.9589

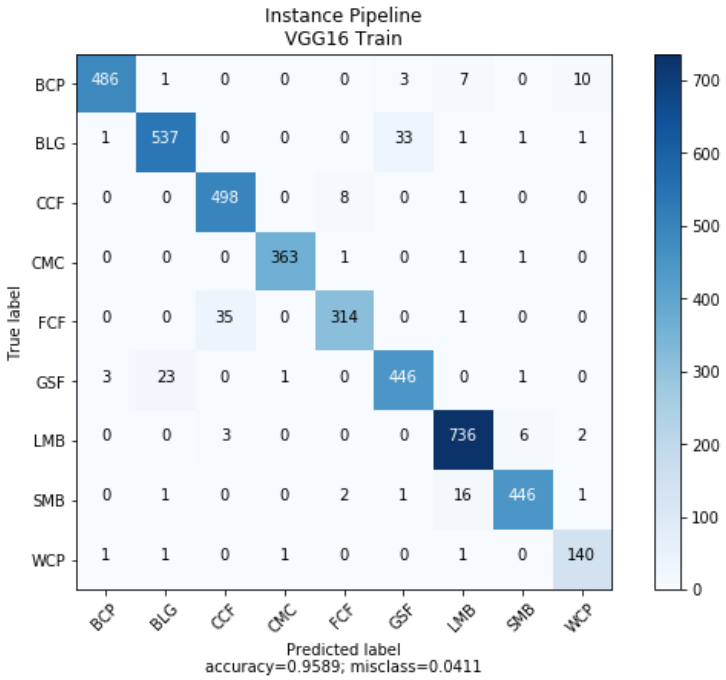


*Figure 30 Instance Pipeline training confusion matric – VGG16 Classifier*

For the validation dataset, we get the following confusion matrix with the accuracy 0.8236

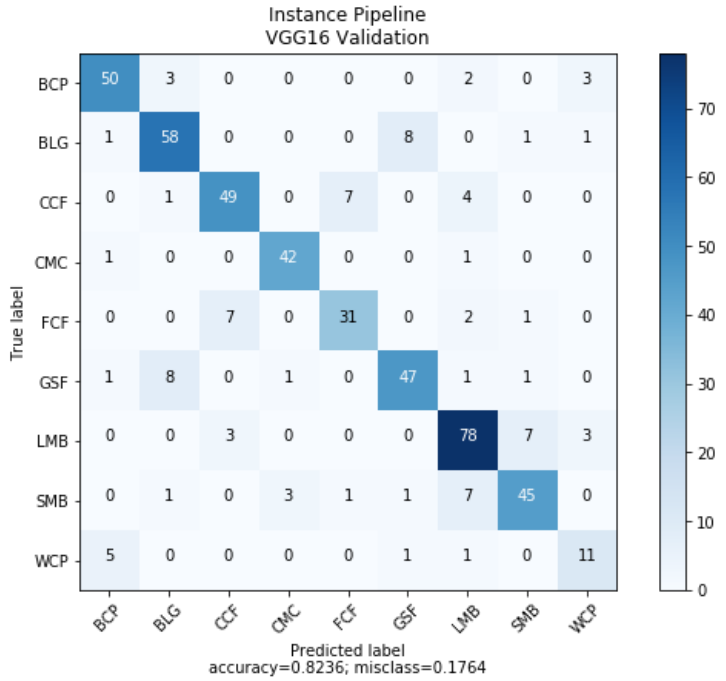


*Figure 31 Instance Pipeline validation confusion matric – VGG16 Classifier*

For the testing dataset, we get the following confusion matrix with the accuracy 0.8064

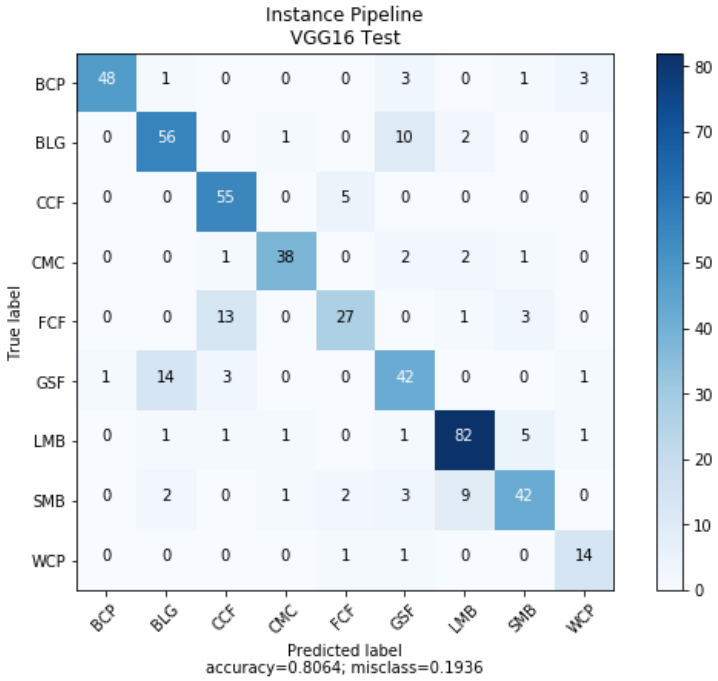


*Figure 32 Instance Pipeline testing confusion matric – VGG16 Classifier*

### 2.2.5 Instance Rotation Pipeline

In Instance Rotation Pipeline, the detector is same as Instance Pipeline. Two models are needed: Pose Estimator and VGG16 Classifier in Instance rotated dataset. The pose estimator is a traditional classification problem, so we still use the same method to modify the neural network based on VGG16: using ImageNet pre-trained model, and adding a fully connected layer with output 16, freeze the layers except for the last two layers. We used (3, 224, 224) as the input, initial learning rate 1e-4, batch size 64, and trained the network by reducing the learning rate by 0.1 if the validation accuracy does not improve in 30 epochs.

| input | lr | # epoch | batch size | time / epoch | total time | last update epoch | loss | val_loss | val_Acc |
|---|---|---|---|---|---|---|---|---|---|
| 3,224,224 | 1e-4 | 53 | 64 | 54 s | 47 min 42 s | 23 | 0.2951 | 0.2944 | 0.9138 |

*Table 10 Pose Estimator training phase*

After 23 epochs, we achieved 91.38% accuracy on validation dataset with the training loss 0.2951 and validation loss 0.2944. The figures show the curve of loss and accuracy on training and validation dataset.

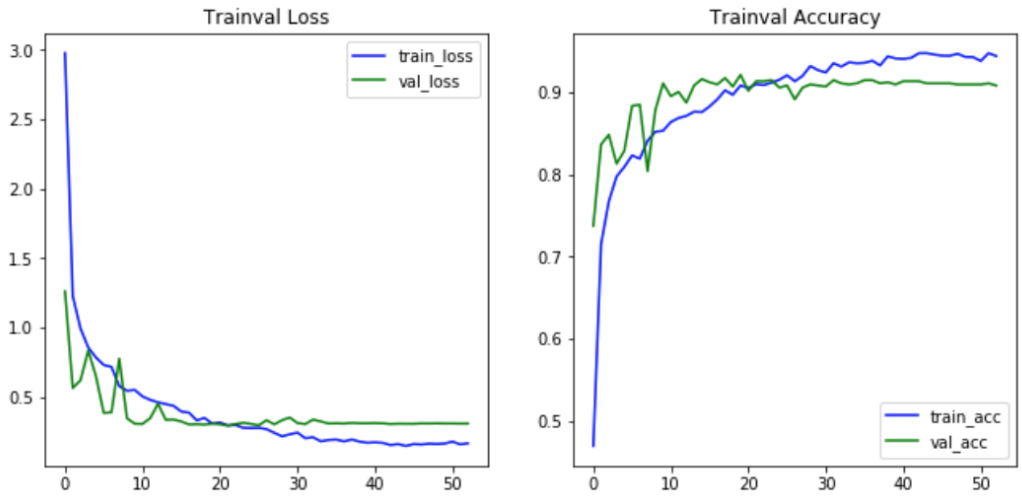


*Figure 33 Pose Estimator loss and accuracy curve*

For the accuracy, we use two different measurement methods as mentioned in Evaluation Metric part. accuracy_0 for traditional classification accuracy, accuracy_1 for offsetting classification accuracy. For the training dataset, we get the following confusion matrix with the accuracy_0: 0.9768, and accuracy_1: 0.9985

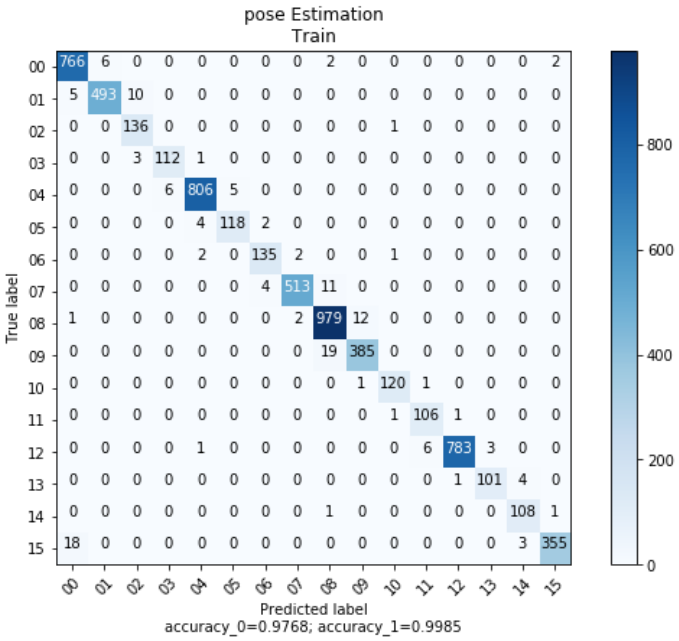


*Figure 34 Pose Estimator training confusion matrix*

For the validation dataset, we get the following confusion matrix with the accuracy_0: 0.9138, and accuracy_1: 0.9948

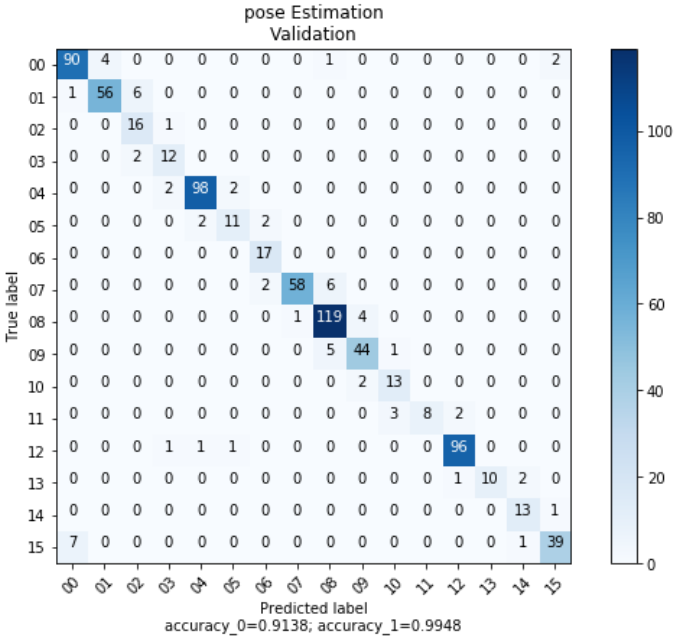


*Figure 35 Pose Estimator validation confusion matrix*

For the testing dataset, we get the following confusion matrix with the accuracy_0: 0.9239, and accuracy_1: 0.9961

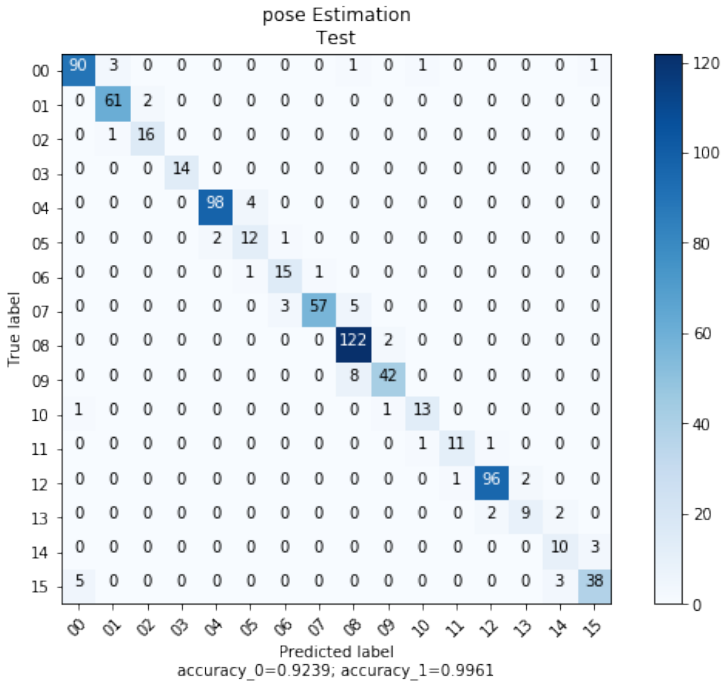


*Figure 36 Pose Estimator testing confusion matrix*

In the VGG16 classifier for instance rotation classification, we did the same step, modify the VGG16 structure with a basic VGG model and added a fully connected layer (output shape 9). We used (3, 224, 224) as the input, initial learning rate 1e-4, batch size 64, and trained the network by reducing the learning rate by 0.1 if the validation accuracy does not improve in 30 epochs.

| input | lr | # epoch | batch size | time / epoch | total time | last update epoch | loss | val_loss | val_Acc |
|---|---|---|---|---|---|---|---|---|---|
| 3,224,224 | 1e-4 | 126 | 64 | 40 s | 1h 24 min | 96 | 0.1039 | 0.3305 | 0.8938 |

*Table 11 Instance Rotation Pipeline training phase – VGG16 Classifier*

After 96 epochs, we achieved 89.38% accuracy on validation dataset with the training loss 0.1039 and validation loss 0.3305. The figures show the curve of loss and accuracy on training and validation dataset.

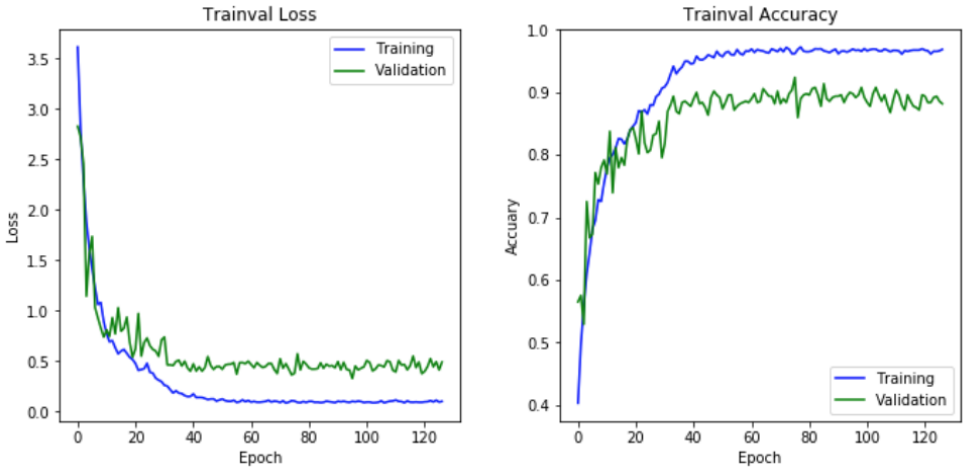


*Figure 37 Instance Rotation Pipeline loss and accuracy curve – VGG16 Classifier*

For the training dataset, we get the following confusion matrix with the accuracy 0.9915

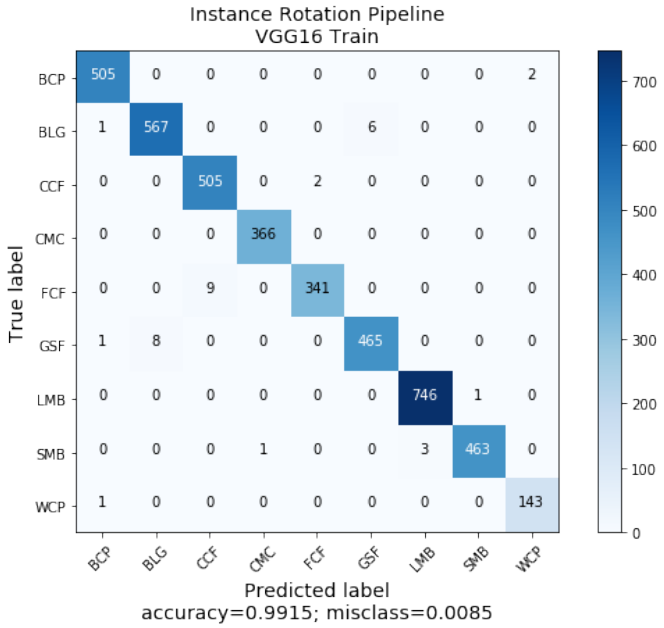


*Figure 38 Instance Rotation Pipeline training confusion matrix – VGG16 Classifier*

For the validation dataset, we get the following confusion matrix with the accuracy 0.8938

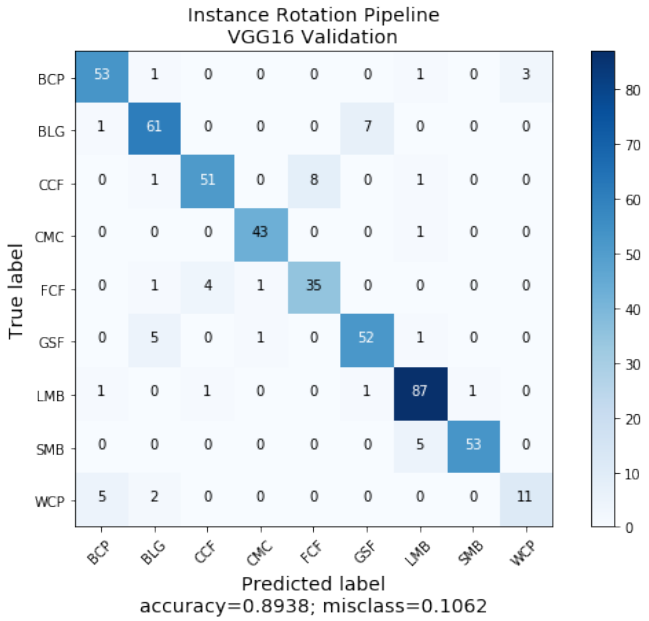


*Figure 39 Instance Rotation Pipeline validation confusion matrix – VGG16 Classifier*

For the testing dataset, we get the following confusion matrix with the accuracy 0.8383

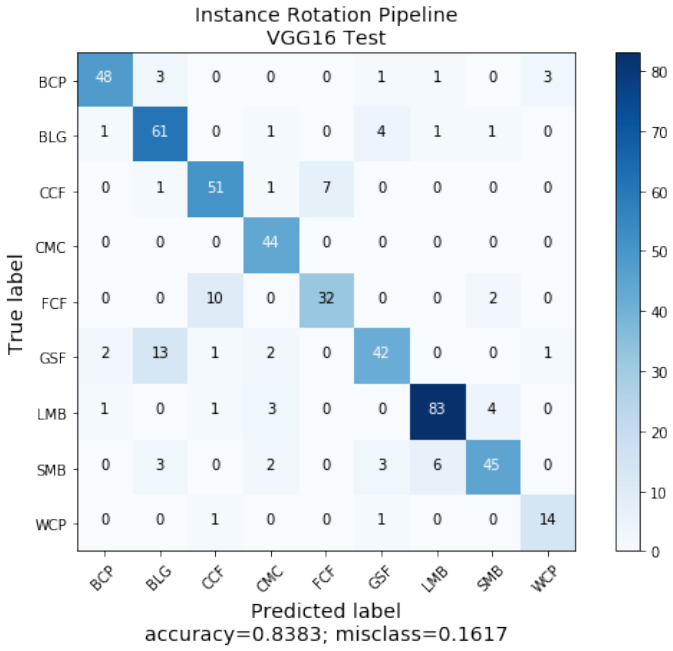


*Figure 40 Instance Rotation Pipeline testing confusion matrix – VGG16 Classifier*

### 2.2.6 Ensemble Pipeline

For the Ensemble Pipeline, obviously, we need another CNN classifier, ResNet50, and train it on the instance rotation dataset. Similar to the modification in VGG16, we load the ImageNet pre-trained ResNet50 model and added another fully connected layer with output 9. We used (3, 224, 224) as the input, initial learning rate 1e-4, batch size 64 and trained the network by reducing the learning rate by 0.1 if the validation accuracy does not improve in 30 epochs.

| input | lr | # epoch | batch size | time / epoch | total time | last update epoch | loss | val_loss | val_Acc |
|---|---|---|---|---|---|---|---|---|---|
| 3,224,224 | 1e-4 | 98 | 64 | 46 s | 1h 15m 8s | 68 | 0.1912 | 0.27 | 0.8717 |

*Table 12 ResNet 50 training phase*

After 98 epochs, we achieved 87.17% accuracy on validation dataset with the training loss 0.1912 and validation loss 0.27. The figures show the curve of loss and accuracy on training and validation dataset.

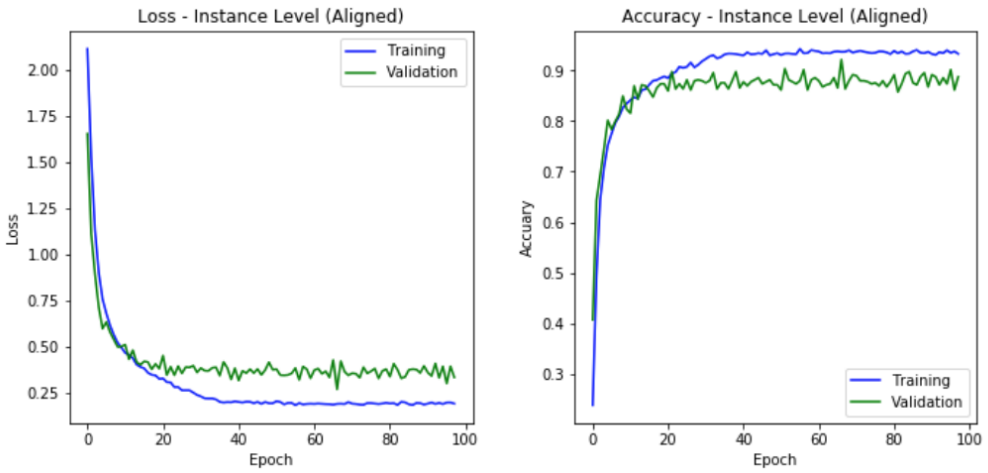


*Figure 41 Ensemble Pipeline - ResNet 50 loss and accuracy curve*

For the training dataset, we get the following confusion matrix with the accuracy

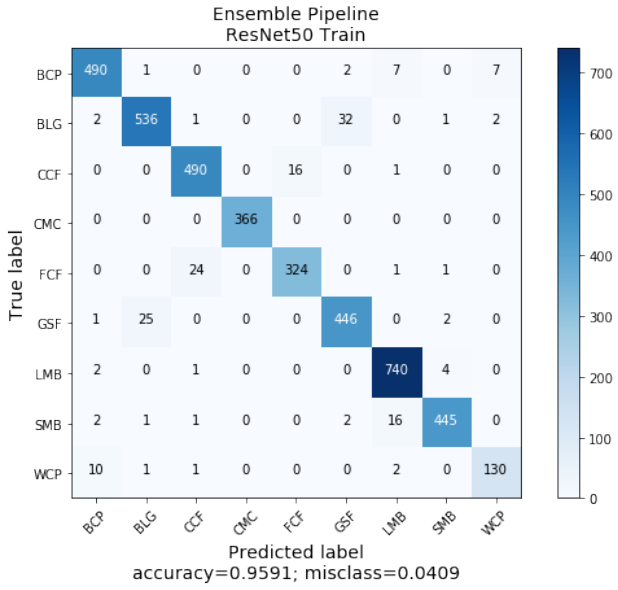


*Figure 42 Ensemble Pipeline training confusion matrix – ResNet50 Classifier*

For the validation dataset, we get the following confusion matrix with the accuracy

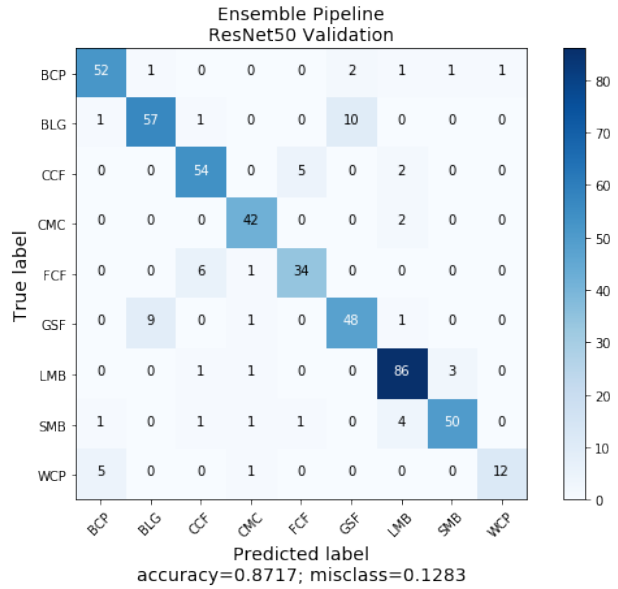


*Figure 43 Ensemble Pipeline validation confusion matrix – ResNet50 Classifier*

For the testing dataset, we get the following confusion matrix with the accuracy 0.8683

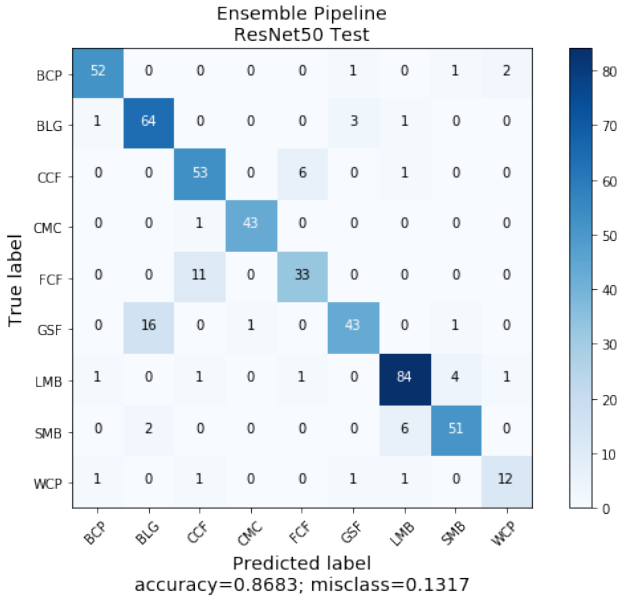


*Figure 44 Ensemble Pipeline  testing confusion matrix – ResNet50 Classifier*

For the ensemble model, we use a weighted average instead of the average algorithm. In order to find the best ratio, we use VGG as a standard, and the ratio is increased by 0.001 from 0 to 1 as we mentioned in design part. On the validation dataset, we have the following curve.

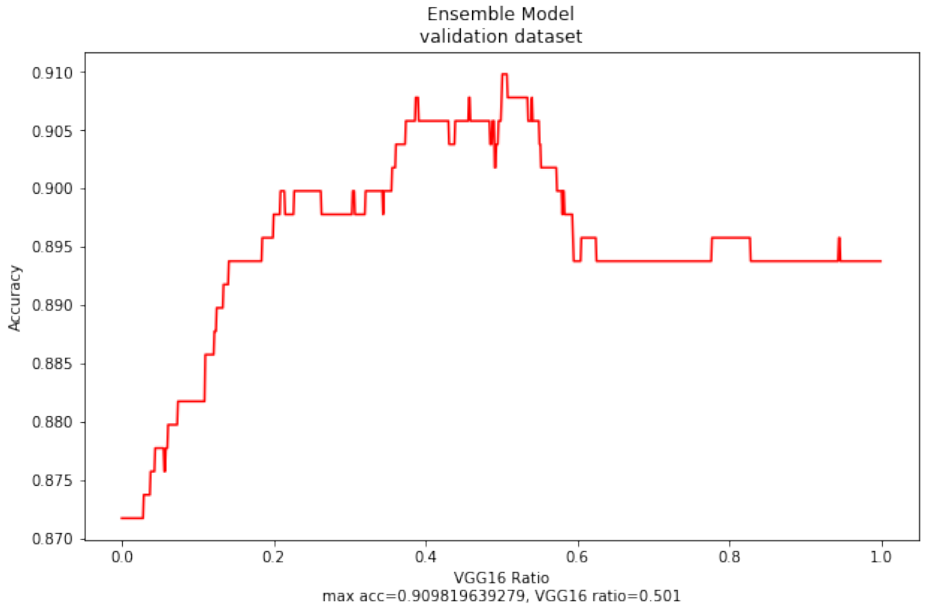


*Figure 45 VGG ratio - validation accuracy curve*

When $ratio_{VGG}$ = 0.501, the maximum accuracy of the ensemble model is 0.9098.

For the training dataset, we get the following confusion matrix with the accuracy 0.9915

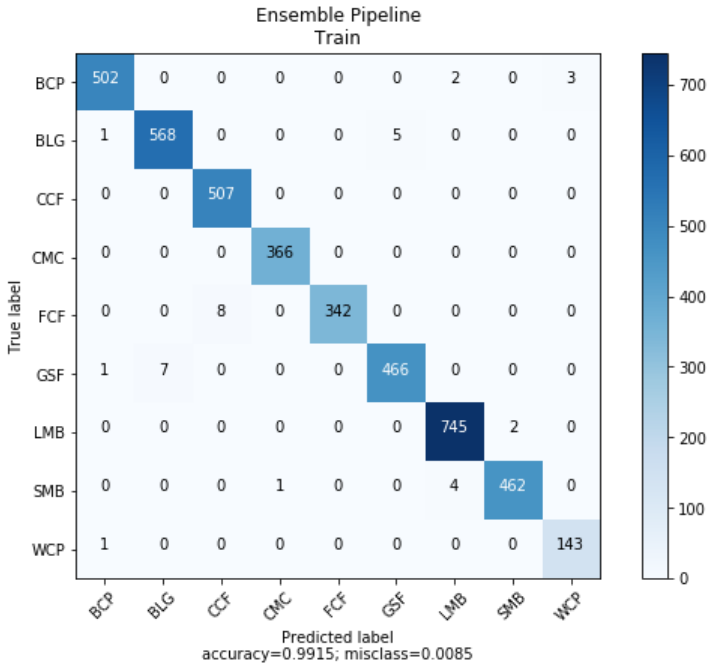


*Figure 46 Ensemble Pipeline training confusion matrix – Weighted Classifier*

For the validation dataset, we get the following confusion matrix with the accuracy 0.9098

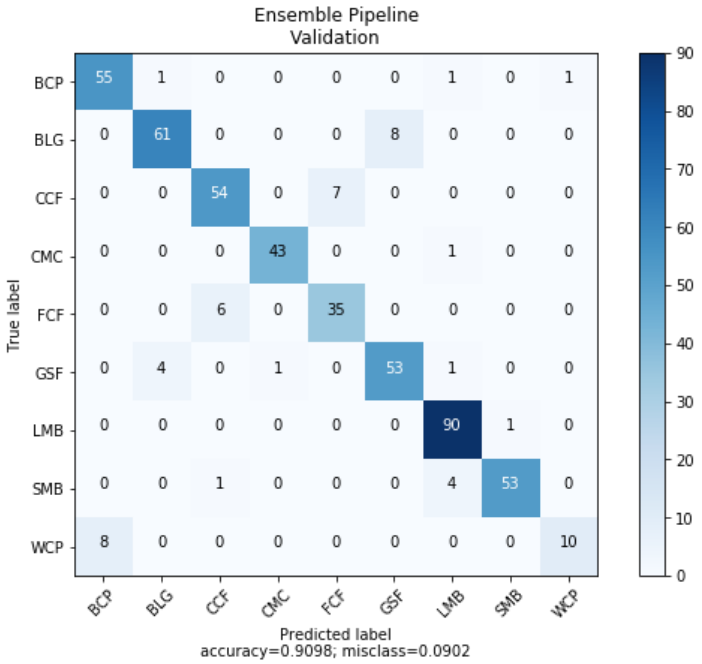


*Figure 47 Ensemble Pipeline validation confusion matrix – Weighted Classifier*

For the testing dataset, we get the following confusion matrix with the accuracy 0.8802

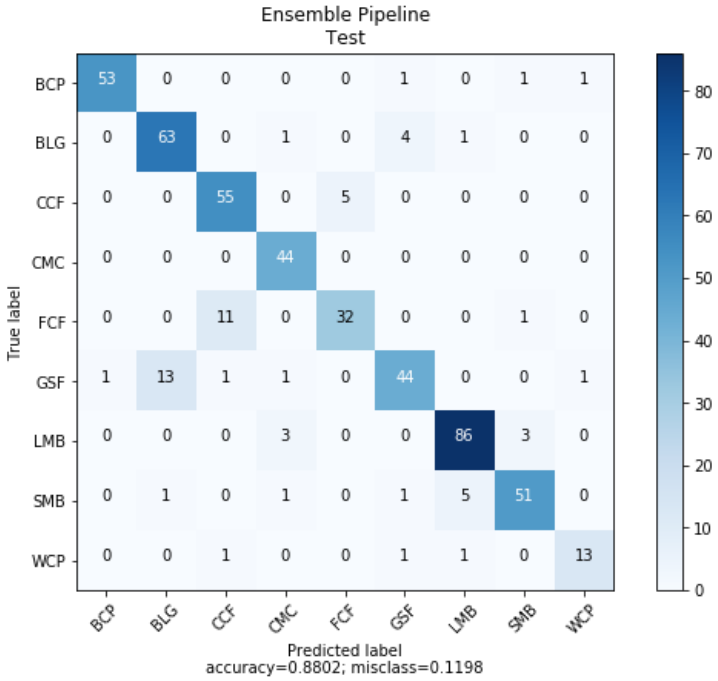


*Figure 48 Ensemble Pipeline testing confusion matrix – Weighted Classifier*

# 3. RESULTS

In the implementation part, we use different image dataset for training on VGG16 and ResNet50 model. For each of the image-classifier combinations we use, we have the following accuracy chart.

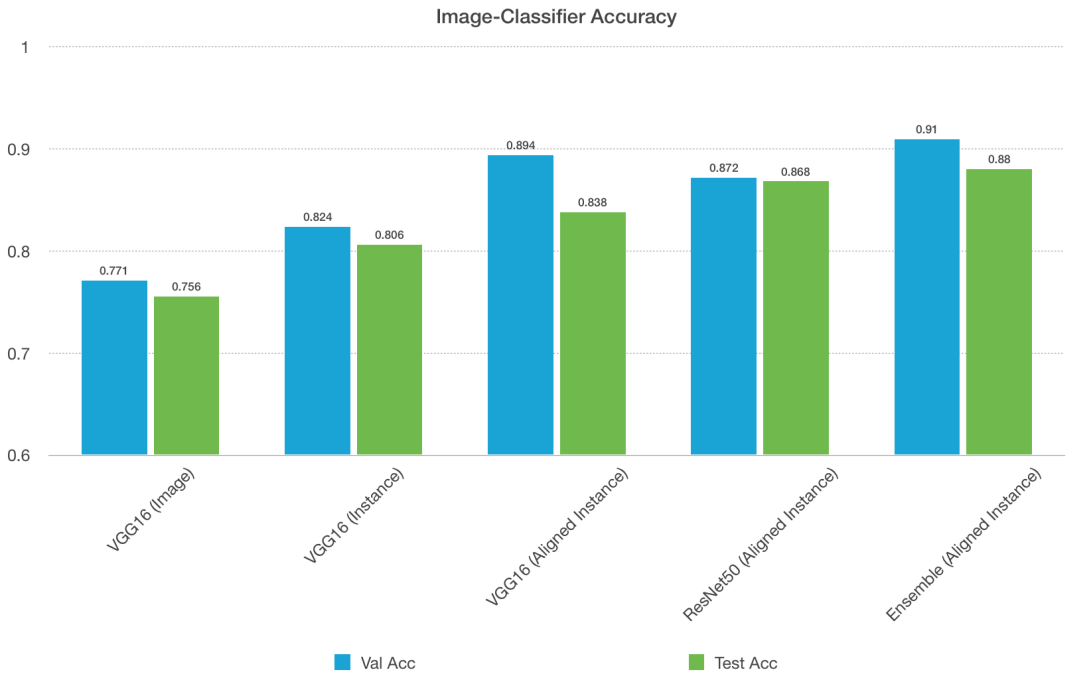


*Figure 49 Image-Classifier Accuracy*

By preprocessing the images and letting the classifier focus on the fish instance itself, we can improve the accuracy more effectively. For all pairs of image-classifier, the validation accuracy is higher than test accuracy. Only considering the classifier, the Ensemble Pipeline performs best both on validation and test dataset.

Here we regard each pipeline as a system and execute them on the test set. For each pipeline, we get the following confusion matrix.

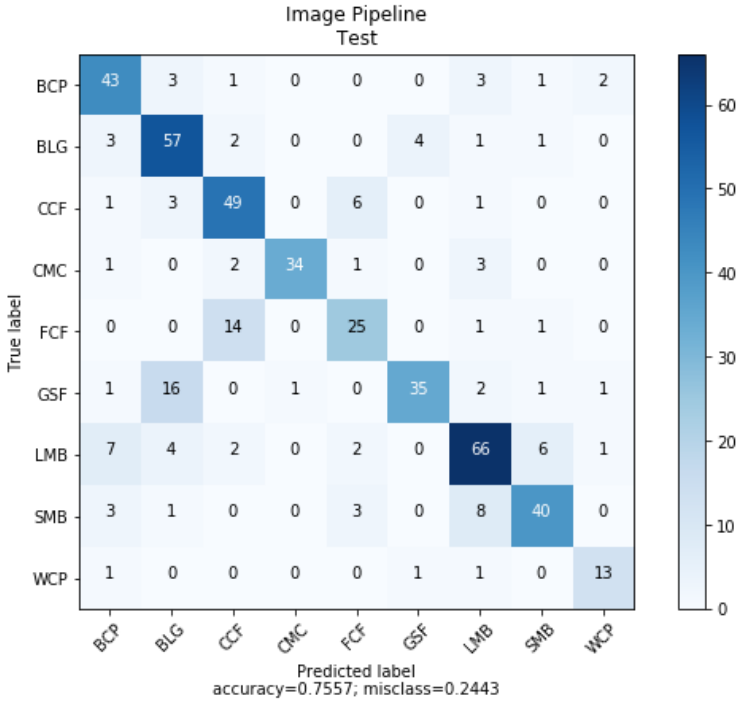


*Figure 50 Image Pipeline test accuracy*

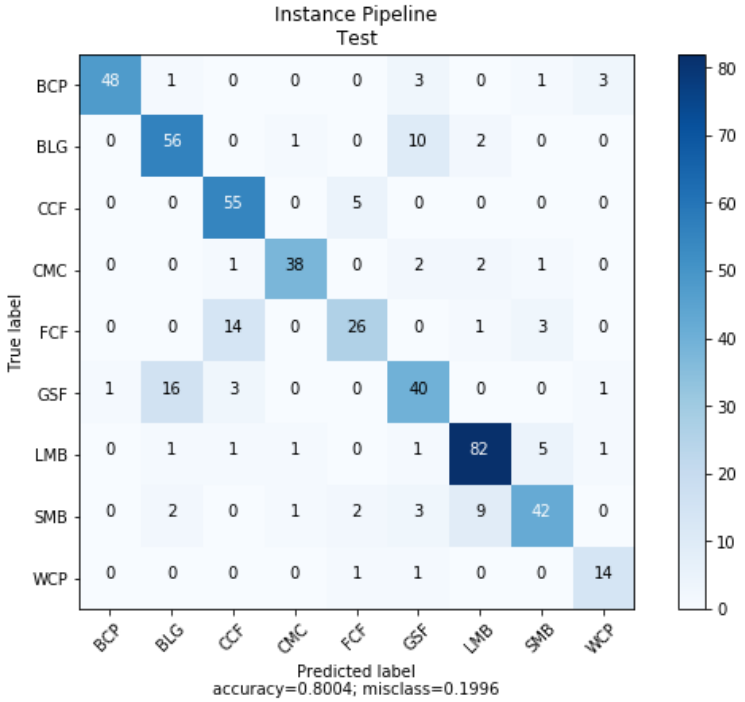


*Figure 51 Instance Pipeline test accuracy*

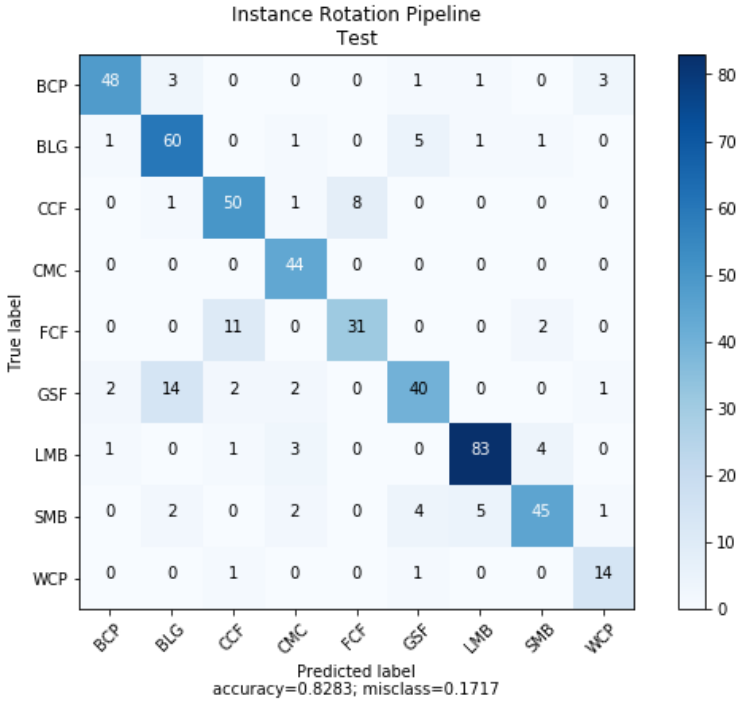


*Figure 52 Instance Rotation Pipeline test accuracy*

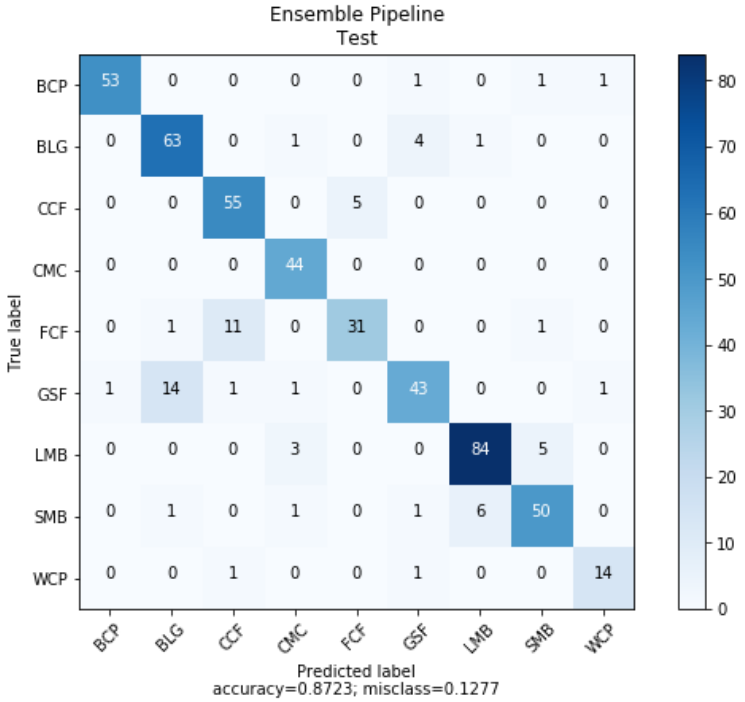


*Figure 53 Ensemble Pipeline test accuracy*

For each system, execution time is another standard for measuring system efficiency on the same data. Finally, combined with all the information obtained, we have the following results.

| | | # of Test Image | # of fish | # of detected 'fish' | tp in detection | correct classification | Accuracy correct classification / # fish | Time (s) |
|---|---|---|---|---|---|---|---|---|
| Image Pipeline | | 479 | - | - | - | 362 | 75.57 % | 13.16 |
| Instance Pipeline | | 479 | 501 | 500 | 494 | 401 | 80.03 % | 53.16 |
| Instance Rotation Pipeline | VGG16 | 479 | 501 | 500 | 494 | 415 | 82.83 % | 132.15 |
| | ResNet50 | 479 | 501 | 500 | 494 | 428 | 85.43 % | 138.02 |
| Ensemble Pipeline | | 479 | 501 | 500 | 494 | **437** | **87.22 %** | 140.58 |

*Table 13 Final test results*

Ensemble Pipeline gets highest classification accuracy, 87.22%, on the same test dataset, but takes the longest time, 140.58 s. Even if Ensemble Pipeline spends the longest time, we still have to choose ensemble as the final model, because in our project, accuracy is the most important.

# 4. CONCLUSION AND FUTURE WORK

## 4.1 Conclusion and Contribution

Through the step-by-step purification of the images, the classifier can focus on the fish instance itself, filter out unwanted information such as background and orientation, and our classifier can obtain higher accuracy.

In this project, we came up with fish classification pipeline on mess images. For our selected pipeline, Ensemble Pipeline, given the input RGB image, we can return the position, category, and confidence of fish to the user as a system. Finally, we achieved 87.22% accuracy for the classification task.

## 4.2 Future Work

In the future, we will improve classification accuracy by trying Automatic Hyper-Parameter Adjustment. To more focus on the information we need and classify the fish by its prominent feature, we are going to extract the tail or fins of fish, and use another model to classify them for supporting the prediction result. For the ensemble method, weighted average is not the best, we could try tiny network instead of it.

Moreover, we would like to use a data visualization method to return the user an output image with the information on it to easily grab the position, species, and confidence of fish.

# 5. REFERENCES


[1] Liu, Wei, et al. "Ssd: Single shot multibox detector." *European conference on computer vision. Springer, Cham*, 2016.

[2] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *In ICLR*, 2015.

[3] He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016.

[4] Chen, Guang, et al. "Automatic Fish Classification System Using Deep Learning." *In ICTAI*, 2017

[5] He, Kaiming, et al. "Mask r-cnn." *Computer Vision (ICCV), 2017 IEEE International Conference on.* IEEE, 2017.

[6] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems.* 2012.

[7] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2016.

[8] M. Ravanbakhsh, M. R. Shortis, F. Shafait, A. Mian, E. S. Harvey, and J. W. Seager, "Automated fish detection in underwater images using shape-based level sets," *The Photogrammetric Record*, vol. 30, no. 149, pp. 46–62, 2015.