ENPM-673 PERCEPTION FOR AUTONOMOUS ROBOTS

# **<u>Project-4</u>**

Jeet Patel (117430479)

## Contents

## 1. Introduction

The principle of optical flow in two variants (sparse and dense) is implemented in this project in part 1. The dataset of the cars running on highway is provided. Estimation of the velocity and directions in which cars move on a highway are done using optical flow methods. In part 2, We are provided with 9 different seafood type (Fish) dataset [1], collected from a Supermarket in Turkey. Using the concept of classification, we need to classify according to the respective fish breed.

## 2. Optical Flow

The aim is to track the motion of vehicles across frames to estimate its current velocity and predict its position in the next frame. For this relationship between consecutive frames is required.

### 2.1 What is Optical flow?

Optical flow is the movement of objects in consecutive frames, induced by the relative movement of the object and the camera. The distribution of apparent velocities of movement of a brightness pattern in an image is also known as optical flow. The problem of optical flow is expressed as:

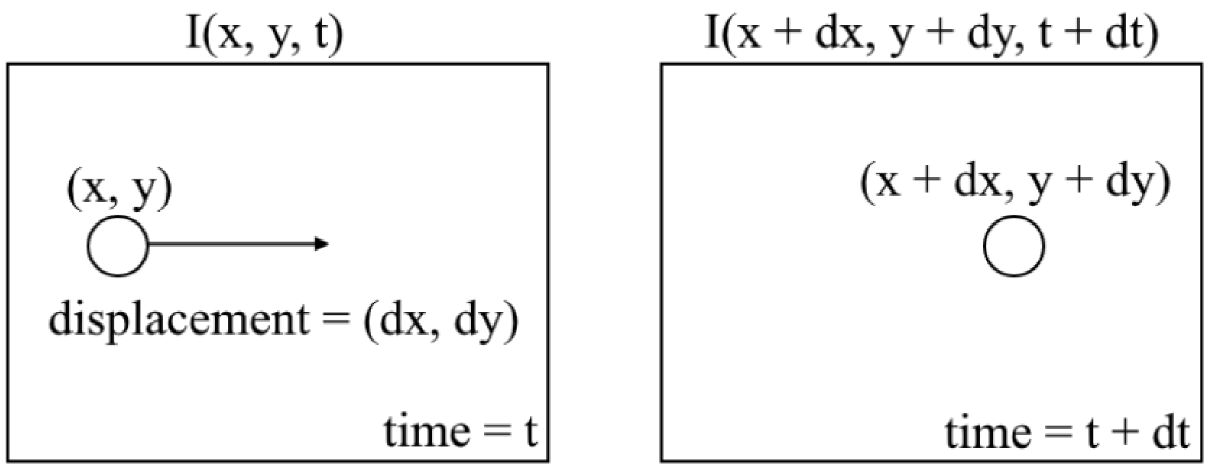


Figure-1: Optical flow problem

Figure-1 show two consecutive frames, which is expressed as image intensity (*I*) as function of (x, y) and time (t). A new image I (x + dx, y + dy, t + dt) is obtained, if we take the first image I (x, y, t) and move the pixels by (dx, dy) over time t.

First, we assume that pixel intensities of an object are constant between consecutive frames.

$$I\ (x, y, t) = I\ (x + dx, y + dy, t + dt)$$

Second, Taylor Series Approximation is taken of the RHS and common terms are removed.

$$f_x\, U + f_y\, V + f_t = 0$$

Where, $f_x$ *and* $f_y$ are image gradients,

$f_t$ are gradients along time and

$$U = \frac{dx}{dt} \text{ and } V = \frac{dy}{dt}$$

Here solving U and V to determine movement over time. We cannot directly solve the optical flow equation for U and V since there is only one equation for two unknown variables. Methods such as the Lucas-Kanade method are used to address this issue.

## 2.2 Sparse Optical Flow

Sparse optical flow tracks the velocity vectors using a sparse feature collection of pixels (e.g., interesting features like edges and corners) (motion). The extracted features are transferred from frame to frame in the optical flow mechanism to ensure that the same points are monitored. The method for plotting vector field is as follows:

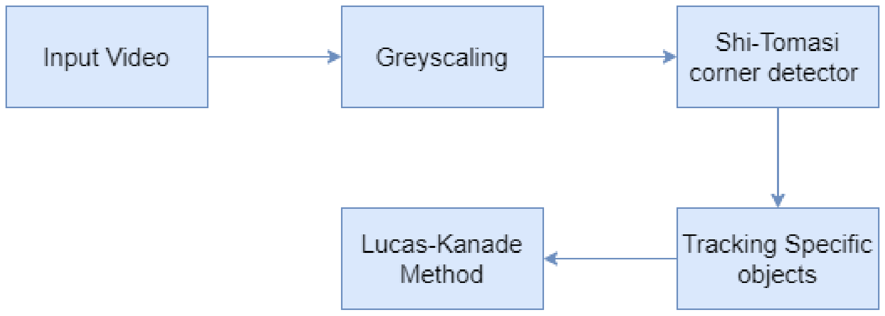


Figure-2: Overview of pipeline for sparse optical flow

First, each frame is capture from the video. The first frame is converted to greyscale because we only need the luminance channel for detecting edges which is less computationally expensive. Then, each frame is converted into greyscale after the first frame. In the given dataset, we only need to detect motion of cars. For a given set of pixels of cars, we have used Shi-Tomasi Corner Detector or any other method to track features. This method for corner detection is applied as follows:

1. Determining windows (small image patches) with large gradients (variations in image intensity) when translated in both x and y directions.
2. For each window, a score of R is computed.
3. Depending on the value of R, each window is classified as a flat, edge, or corner.

In Shi-Tomasi proposed the scoring function is R = min (λ1, λ2). If R is greater than threshold than it is classified as corner.

Here pixels spaced at 25 are selected manually to draw the optical flow and plotting vector field.

Then Lucas Kanade method is applied to estimate the motion of interesting features by comparing two consecutive frames. The Lucas-Kanade method works under the following assumptions:

1. Two consecutive frames are separated by a small-time increment (dt) such that objects are not displaced significantly.
2. A frame portrays a “natural” scene with textured objects exhibiting shades of gray that change smoothly.

Lucas Kanade method iteratively calculates an optical flow for a sparse feature set with pyramids. The function used is cv2.calcOpticalFlowPyrLK(). It takes input of previous frame, next frame, window size, and 2d vector point and gives us 2D vector points of new frame. After that for plotting vector field, cv2.arrowedlines() function is used.

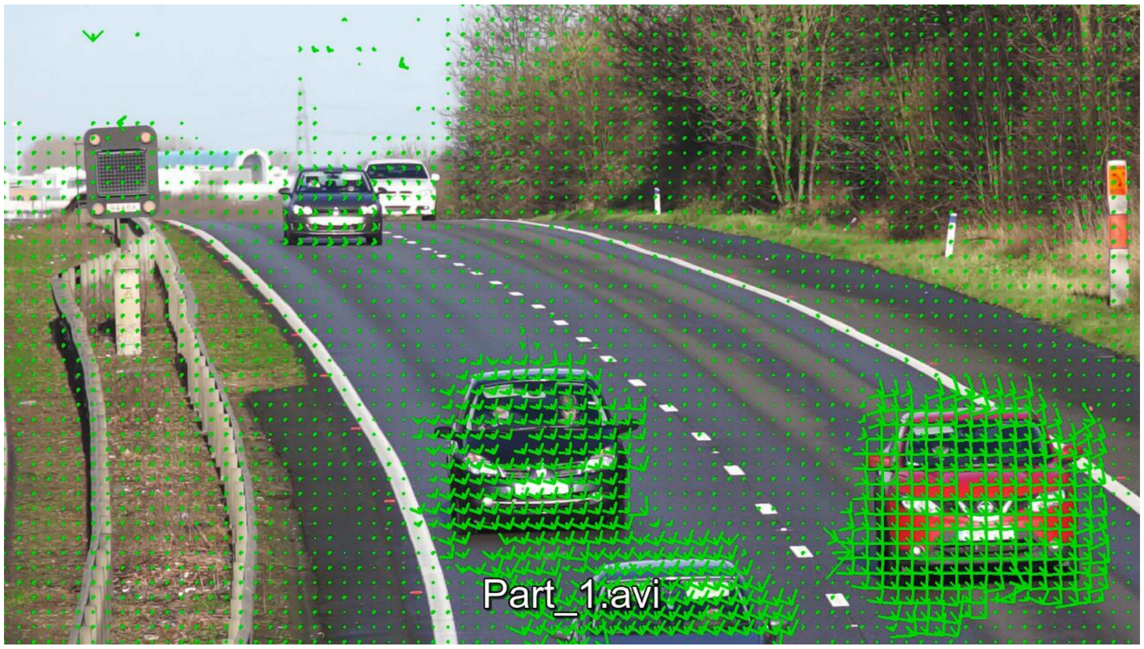


Figure-3: Results of Vector field

## 2.3 Dense optical flow

Dense optical flow tries to find the optical flow vector for each pixel in each frame. While this method is slower, it produces a more precise and dense performance, which is ideal for applications like learning form from motion and video segmentation. Here Farneback method is used for the dense optical flow.

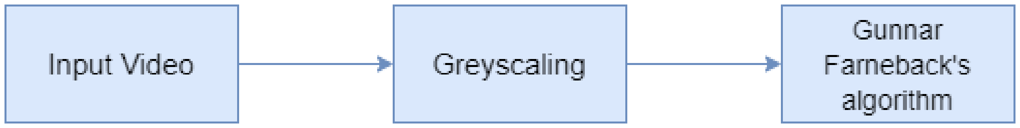


Figure-4: Overview of pipeline for dense optical flow

In Farneback Method, the procedure begins by using the polynomial expansion transform to calculate the windows of image frames with a quadratic polynomial. Then, observe how the polynomial transforms as the state of motion changes. To put it another way, to estimate displacement fields. After a sequence of refinements, dense optical flow is computed.

The magnitude and direction of optical flow are computed for this using a 2-D channel array of flow vectors. The distance (magnitude) of flow is visualized by the value of HSV color representation, while the angle (direction) of flow is visualized by hue. For optimum visibility, the HSV intensity is often set to a limit of 255. This is implemented using cv2.calcOpticalFlowFarneback.

## 2.4 Results of Dense Optical Flow

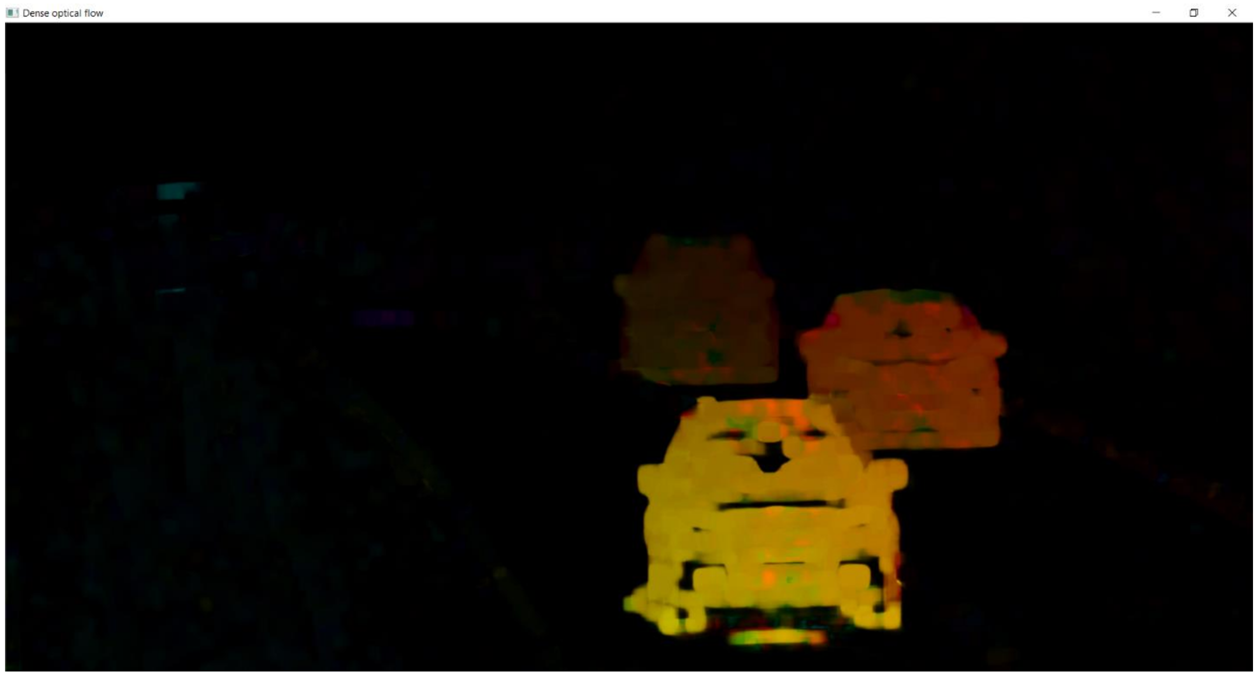


Figure-5: Results of Dense Optical Flow

## 3 Classification of Fish Dataset

In this part, the aim to classify the large dataset of species of fish by developing the Convolution Neural Network (CNN). The convolutional neural network (CNN) is a class of **deep learning neural networks**. They are most widely used to analyze visual imagery and are often used in image classification behind the scenes. Image classification is the process of taking an input (such as a picture) and generating a class (such as "cat") or a probability that the input belongs to that class ("this input has a 90% chance of being a cat").

A CNN has:

- **Convolutional layers**: The main purpose of the convolution step is to extract features from the input image. Here usage of multiple convolution filters or kernels that run over the image and compute a dot product. Each filter extracts different features from the image.
- **ReLU layers:** It is an activation function applied onto feature map to increase non-linearity in network.
- **Pooling layers:** Pooling simply means down sampling of an image. It takes small region of the convolutional output as input and sub-samples it to produce a single output. Max pooling layer helps reduce the spatial size of the convolved features and helps reduce over-fitting by providing an abstracted representation of them. It is a sample-based discretization process. It works in the same way as the convolution layer, except instead of taking the dot product of the input and the kernel, we take the maximum of the input overlapped by the kernel.
- **Fully connected layer:** This layer takes input from all neurons in the previous layer and performs operation with individual neuron in the current layer to generate output.

CNNs have an input layer, and output layer, and hidden layers. The hidden layers usually consist of convolutional layers, ReLU layers, pooling layers, and fully connected layers.

- Convolutional layers apply a convolution operation to the input. This passes the information on to the next layer.
- Pooling combines the outputs of clusters of neurons into a single neuron in the next layer.
- Fully connected layers connect every neuron in one layer to every neuron in the next layer.

Only a subarea of the previous layer's input is received by neurons in a convolutional layer. Each neuron in a completely connected layer receives input from all elements of the previous layer. A CNN is a machine that extracts features from photographs. It is no longer necessary to manually extract features. They are picked up as the network works through a series of photos. Feature detection is taught to CNNs across tens or hundreds of secret layers. The process of CNN for image classification is as follows:

- starts with an input image
- applies many different filters to it to create a feature map
- applies a ReLU function to increase non-linearity
- applies a pooling layer to each feature map

- flattens the pooled images into one long vector.
- inputs the vector into a fully connected artificial neural network.
- processes the features through the network. The final fully connected layer provides the "voting" of the classes that we are after.
- trains through forward propagation and backpropagation for many, many epochs. These repeats until we have a well-defined neural network with trained weights and feature detectors.

## 3.1 VGG-16 Architecture

VGG16 is a convolution neural net (CNN) architecture which is one of the excellent vision model architecture till date. Most unique thing about VGG16 is that instead of having many hyper-parameters they focused on having convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 2 FC (fully connected layers) followed by a SoftMax for output. The 16 in VGG16 refers to it has 16 layers that have weights.

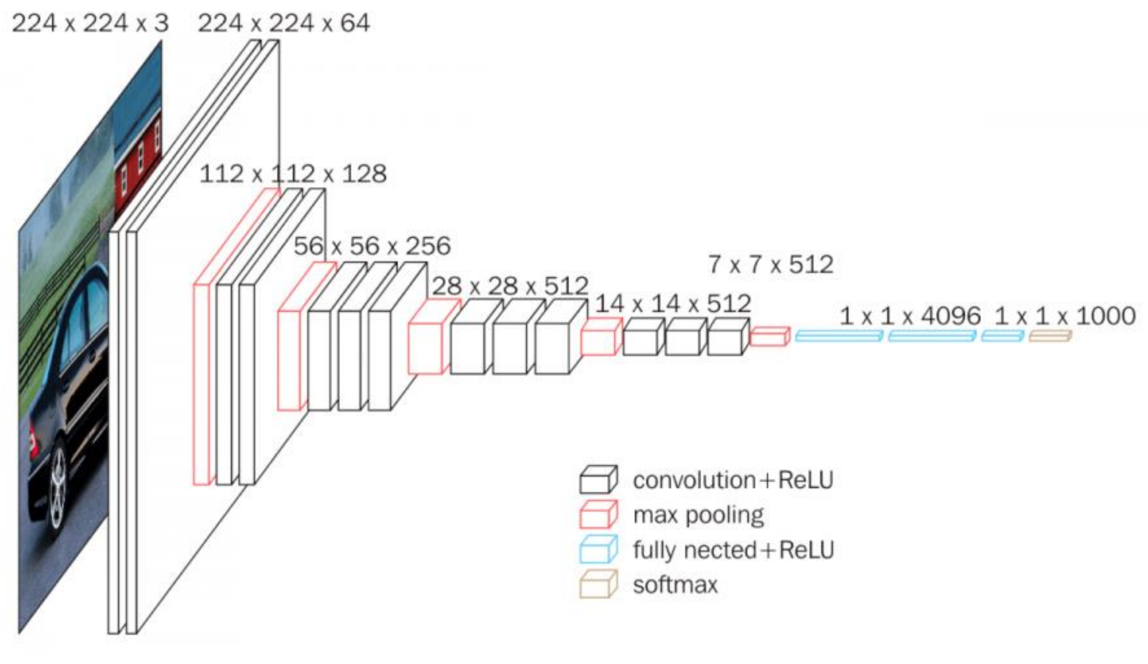
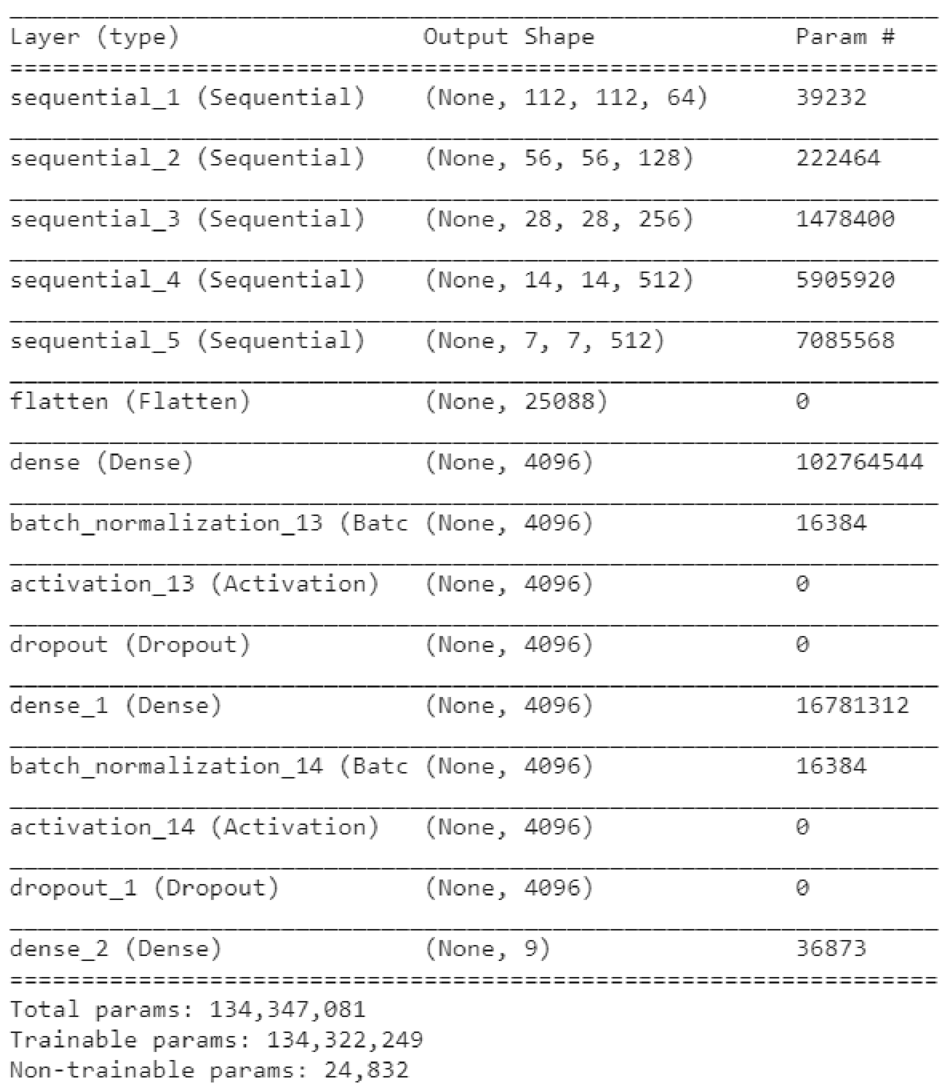


Figure- 6: VGG-16 architecture

Here I have started with initializing the model by specifying that the model is a sequential model.

The summary of the model is as below:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_1 (Sequential)    (None, 112, 112, 64)      39232
_________________________________________________________________
sequential_2 (Sequential)    (None, 56, 56, 128)       222464
_________________________________________________________________
sequential_3 (Sequential)    (None, 28, 28, 256)       1478400
_________________________________________________________________
sequential_4 (Sequential)    (None, 14, 14, 512)       5905920
_________________________________________________________________
sequential_5 (Sequential)    (None, 7, 7, 512)         7085568
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 4096)              102764544
_________________________________________________________________
batch_normalization_13 (Batc (None, 4096)              16384
_________________________________________________________________
activation_13 (Activation)   (None, 4096)              0
_________________________________________________________________
dropout (Dropout)            (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
batch_normalization_14 (Batc (None, 4096)              16384
_________________________________________________________________
activation_14 (Activation)   (None, 4096)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)              (None, 9)                 36873
=================================================================
Total params: 134,347,081
Trainable params: 134,322,249
Non-trainable params: 24,832
```

Figure-7: Summary of layers of implemented model

## 3.2 Pipeline

The overview of pipeline for classifying the fish dataset is as follows:

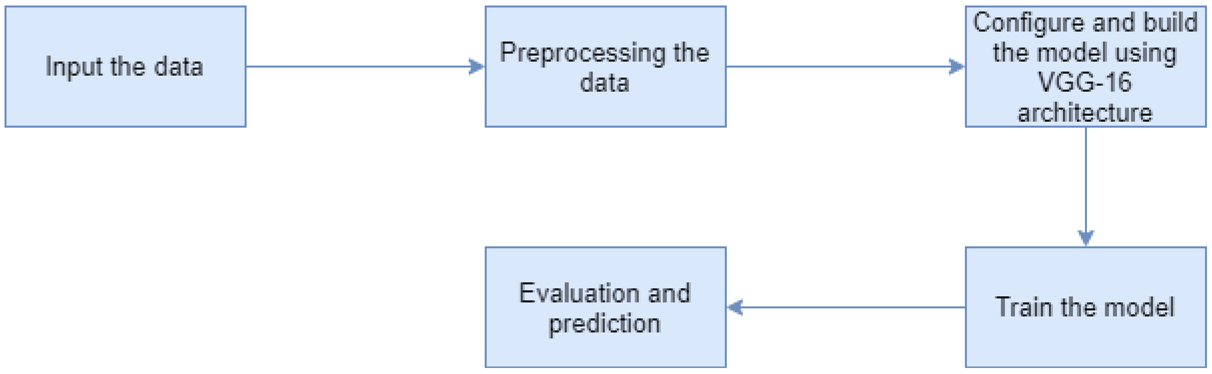

Input the data
Preprocessing the data
Configure and build the model using VGG-16 architecture
Train the model
Evaluation and prediction

Figure-8: Overview of pipeline for image classification

First, a large-scale fish dataset is loaded. Then, color channels are separated into three from the images and then data is labeled into 9 different categories. Then data is prepared for training and testing by splitting 80% of data for training and remaining 20% for testing. Here, there is a limit of 500 for data from each class. After splitting the dataset, data is augmented to prevent overfitting and improve generalization by generating similar images of the original data differing in linear translation, reflection, rotation, and shear. Then VVG-16 architecture is employed for classification. One of the very useful functions in Keras is callbacks which are used for the training of the model for monitoring, logging, and debugging purposes. Here ModelCheckpoint, LearningRateScheduler, TensorBoard, EarlyStopping, ReduceLROnPlateau are used.

After configuring the model, we train it on the data generated one batch at a time. The model is trained over 30 epochs with Adam optimizer and patience is set to 5 which means that the model will stop to train if it does not see any rise in validation accuracy in 30 epochs. The batch size is set as 16.

The training and validation loss and accuracy is plotted using matplotlib visualizing the increasing accuracy and decreasing loss as shown in image below.

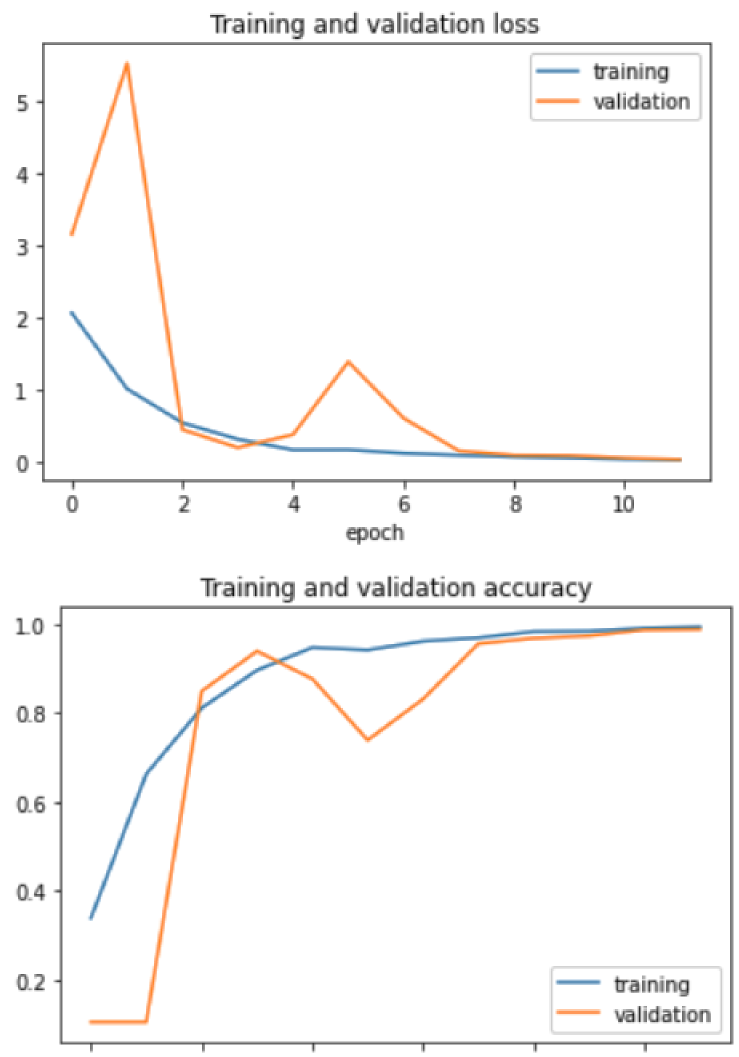
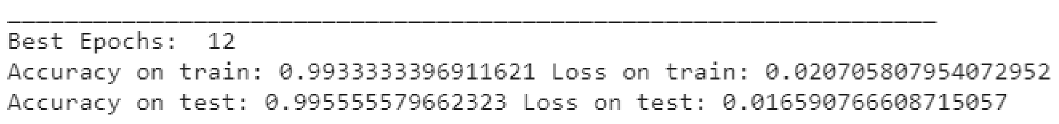


Figure-9: Accuracy vs Epoch & Loss vs Epoch Plots

Result summary of model accuracy is shown as below:



```
________________________________________________________________
Best Epochs:  12
Accuracy on train: 0.9933333396911621 Loss on train: 0.020705807954072952
Accuracy on test: 0.995555579662323 Loss on test: 0.016590766608715057
```

Figure-10: Accuracy results of model

## 4. Challenges Faced

- To run the code, it requires GPU for train and testing the model. Initial task was to load the data of larger size in Google Drive and then use it in google Collaboratory for speed up the processing.
- The dataset contains noisy backgrounds and distinct exposures. For this, various preprocess is done to filter the image and make it suitable for image classification.
- Significant time was spent in understanding various architecture and deciding the best architecture after testing and changing various parameters.

## 5. References

- [1] O.Ulucan, D.Karakaya, and M.Turkan.(2020) A large-scale dataset for fish segmentation and classification. In Conf. Innovations Intell. Syst.Appli. (ASYU)
- "OpenCV: Optical Flow," *Opencv.org*, 2021. https://docs.opencv.org/master/d4/dee/tutorial_optical_flow.html.
- Wikipedia Contributors, "Convolutional neural network," Wikipedia, May 13, 2021. https://en.wikipedia.org/wiki/Convolutional_neural_network.
- X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating Very Deep Convolutional Networks for Classification and Detection," arXiv.org, 2015. https://arxiv.org/abs/1505.06798.
- freeCodeCamp.org, "Image Classification with Convolutional Neural Networks | Deep Learning with PyTorch: Zero to GANs |,". [Online]. Available: https://www.youtube.com/watch?v=d9QHNkD_Pos.

## 6. Output Video link

- https://drive.google.com/drive/folders/1K0A71BH9bOnGGN9O6BwSE15ab-JbAJbi?usp=sharing