ENPM 673
Project 4 Report

- **Problem 1**:

  - **Optical Flow:**

Optical flow in Computer Vision is an indispensable technique to obtain information about object motion which finds its use object detection and video compression. Optical Flow is the distribution of the apparent velocities of objects in an image. It is a set of techniques used to estimate the pattern of motion of the moving objects in an image. The main idea of Optical Flow is to estimate the object's displacement vector caused by its motion or camera movements.

Consider a pixel I (x, y, t) in a video frame which defines the exact pixel intensity at the frame *t* . It moves by distance (dx, dy) in next frame taken after time *dt* . Assuming object displacement does not change the pixels intensity and dt = 1, we can say,

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

We take Taylor series approximation of RHS, remove common terms and divide by dt to get the following equation:

$$f_x u + f_y v + f_t = 0$$

where:

$$f_x = \partial f / \partial x \; ; \; f_y = \partial f / \partial y$$

$$u = dx / dt \; ; \; v = dy / dt \qquad ..(1)$$

The equation above is called the Optical Flow equation. Here, f_x and f_y are the image gradients and f_t is the gradient along time. But u and v are unknown, and one equation will not be solved by two unknown variables. Hence, various determination methods such as Lucas-Kanade, Horn-Schunck are used to solve this problem.

- Types of Optical Flow:
  1) Sparse Optical Flow: In this type, motion vectors for a specific set of objects like the corners of an image are computed. Only the sparse features are considered, and the pixels not contained in these features are ignored. Hence it requires some preprocessing to extract the features.

2) Dense Optical Flow: Contrary to Sparse optical flow, the Dense Optical Flow uses every pixel in the image and calculates the motion vectors. Dense pyramid Lucas-Kanade, PCAFlow, Farnback and RLOF are some of the examples of Dense Optical Flow. I have implemented the Farnback Dense Optical Flow for this problem. A 2-channel array with optical flow vectors, (u,v) expressed in equation 1 above is obtained by the opencv function for the Farnback algorithm. After this their magnitude and angle are obtained by the opencv cartesian to polar coordinate conversion function. The Hue channel and Value channel of the HSV image correspond to angle and magnitude of the vectors. Then, start and end points are provided for these vectors which are spaced by a 25x25 window.

- **<u>Results</u>**:

  YouTube Video Problem 1.1: Link

  YouTube Video Problem 1.2: Link1, Link2

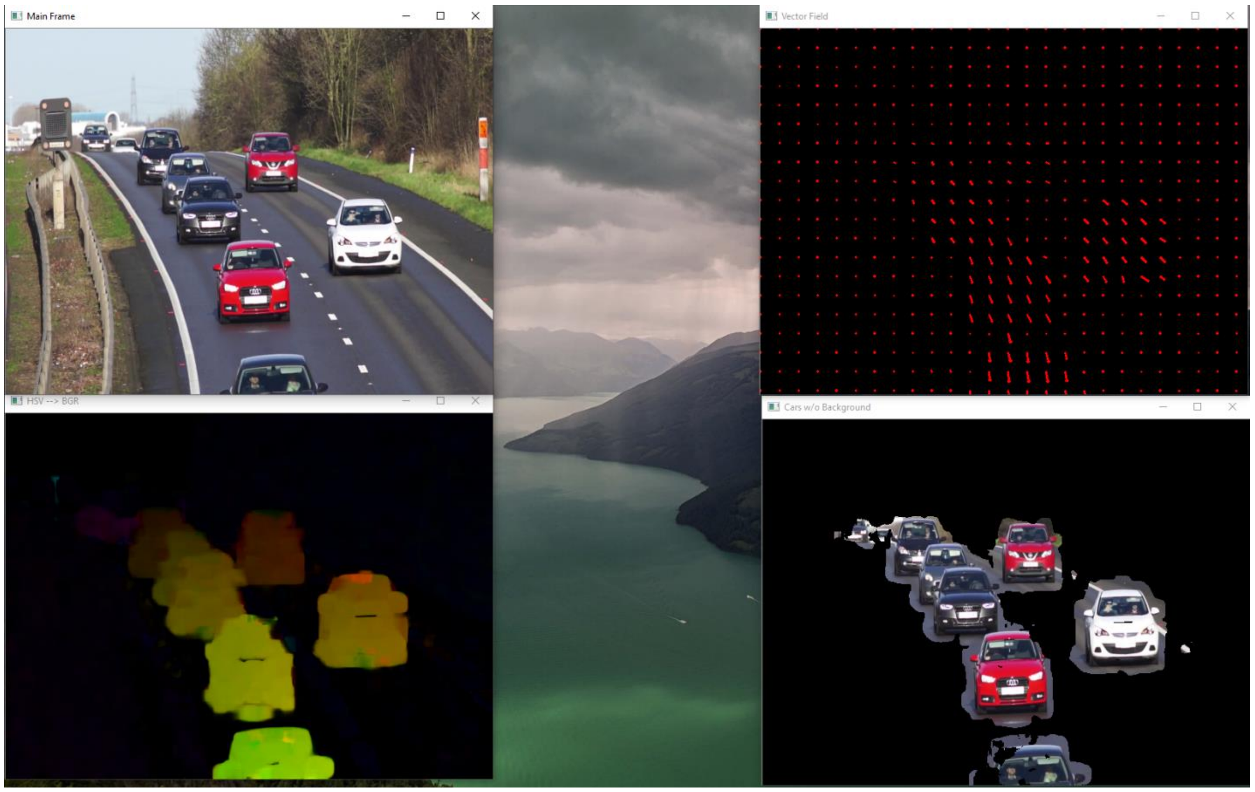


Figure 1: Main frame, Vector Field, Motion Output and Moving Features w/o the background are shown.

The results above show the main frame of the video along with the output at three stages of the solution. The right top corner shows the Motion Vector field with spaced (vertically and horizontally 25 pixels apart) Red arrows showing the direction of the motion features in the current frame. The bottom left window shows the BGR image converted from the HSV image which has angle and magnitude values in its Hue and Value channels, respectively. Lastly, the right bottom corner shows the main from with only the motion features.

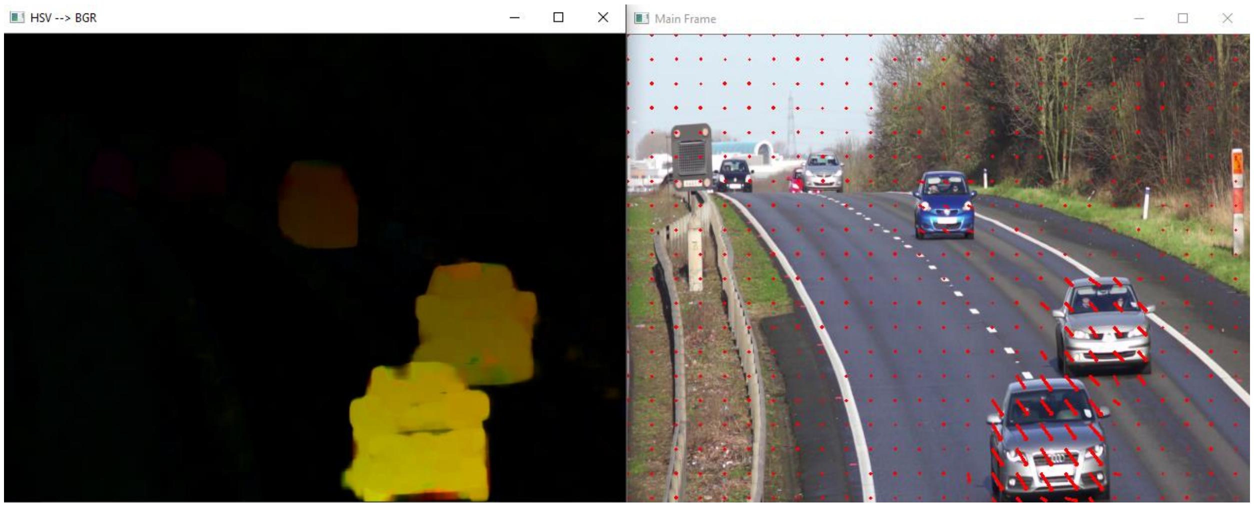


Figure 2: Final output of the problem with the vector field imposed on the main frame and the RGB image of angle and magnitude on the left.

➢ **Problem 2**:

• **Convolutional Neural Networks**:

A convolutional neural network is a class of deep neural networks which is used to analyze visual imagery. It takes an image as input and studying its different aspects, provides a verdict of the type of that image. They are also known as shift invariant artificial neural networks, based on the shared-wight architecture of the convolution kernels or filters that slide along the input features and provide feature maps. CNNs are inspired by the neural connectivity in biological processes meaning each neuron in one layer connected to all the neurons in the next layer. CNNs possess the ability to learn features of the images it is provided.

The dataset used for the problem is “A large-scale dataset for fish segmentation and classification” cited in [1].

- The structure of a Convolutional Neural Network is shown below:
  It contains one input layer, one output layer and two or more hidden layers.

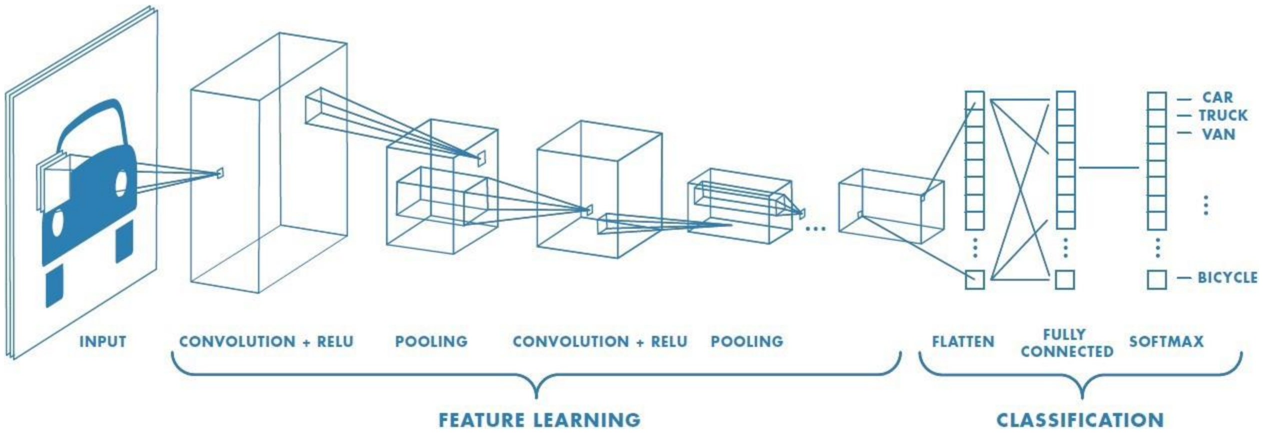


Figure 3: The general structure of a CNN

The hidden layers are a combination of convolution layers, pooling layers, activation functions and fully connected layers.

Input Layer: The main difference between a neural network and a convolutional neural network is that the CNN handles input and output in 3 dimensions. Hence, the input to the input layer is an image as a three-dimensional matrix. Generally, the image is resized as desired by the CNN.

Convolutional Layer: This layer consists of the kernel which is a 2-D filter and the image which is a 3-D matrix. Convolution operation takes place in this layer between the image and the kernel. The kernel elements are obtained from the gaussian distribution.

Pooling Layer: The purpose of pooling layer is to reduce the size of the image by retaining only the important features. It also prevents the neural network from overfitting the data. Average and Max pooling is the most commonly used.

Activation function: This is the function which decides if to activate a neuron or not based on calculating the weighted sum and the bias to it. It is applied on the output if the convolutional layer to decide whether to activate a particular pixel or not.

- **VGG-16**:

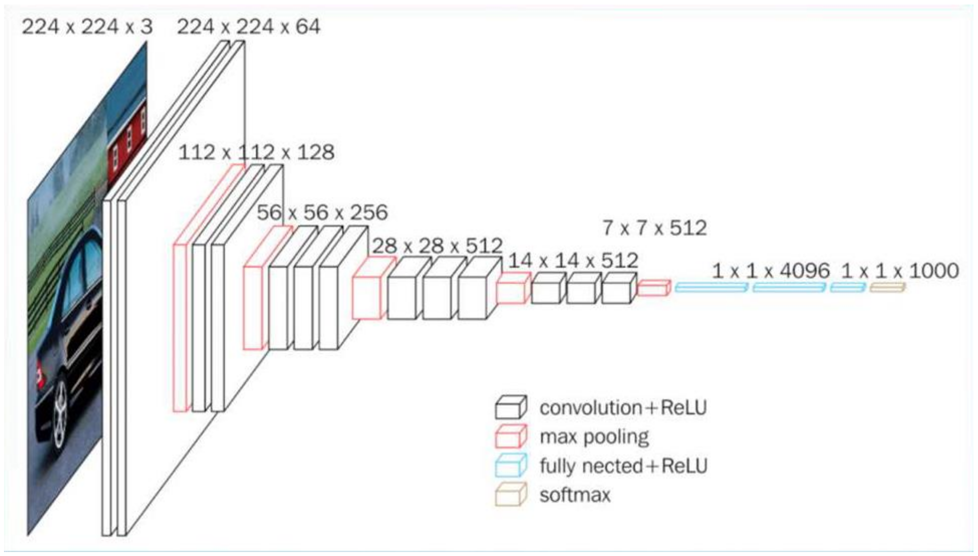
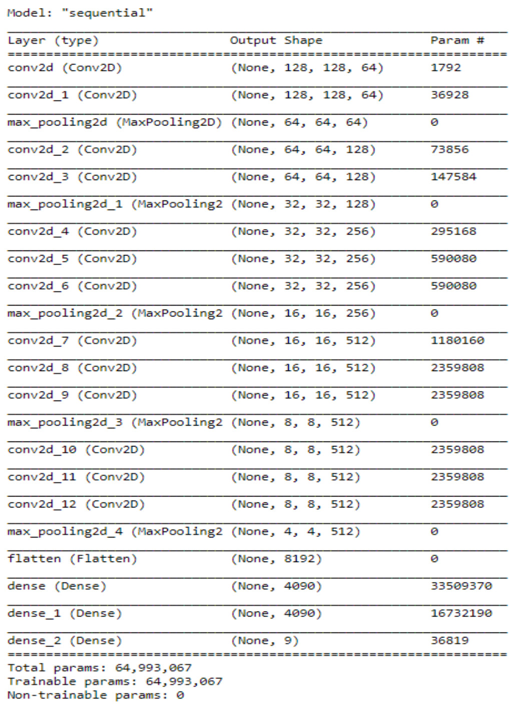


Figure 4: VGG-16 Architecture

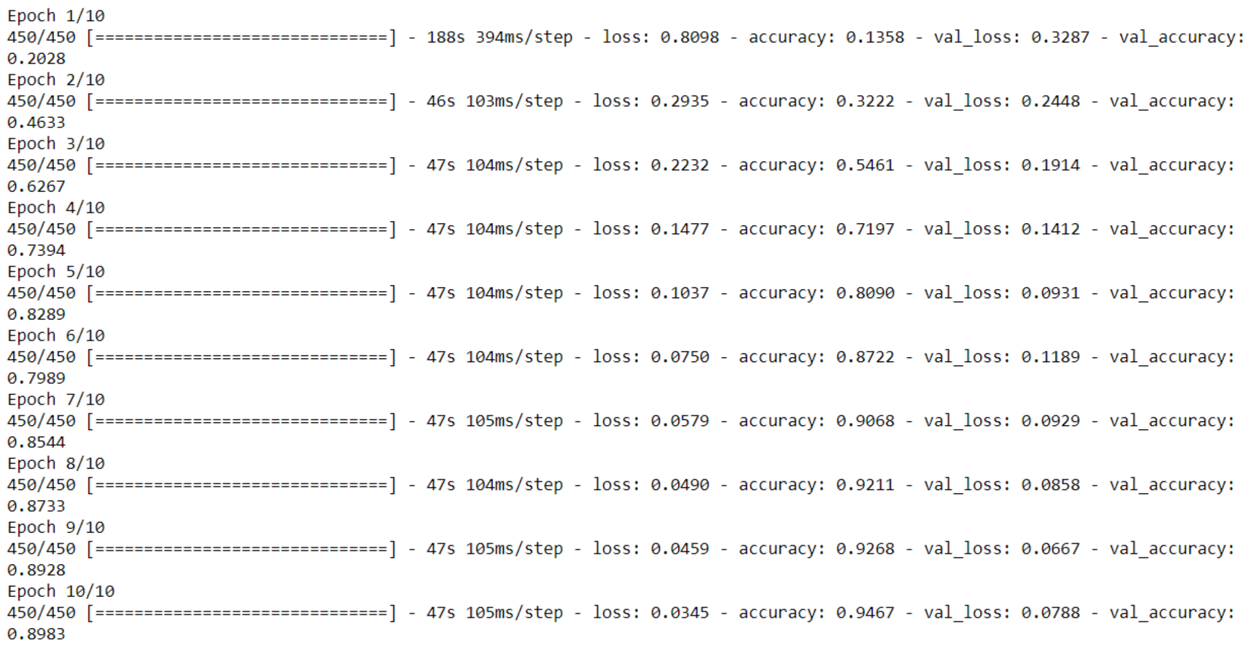
The VGG-16 CNN is a architecture implemented by Zimmerman which consists of 16 layers such as convolutional layers, max pooling layers, activation function layers and fully connected layers. The implemented VGG-16 architecture contains 13 convolutional layers, 5 max-pooling layers, 3 dense layers and one flatten layer.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 128, 128, 64)      1792
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 128, 128, 64)      36928
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 128)       73856
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 64, 128)       147584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 32, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 256)       295168
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 32, 32, 256)       590080
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 32, 32, 256)       590080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 256)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 16, 16, 512)       1180160
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 16, 16, 512)       2359808
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 16, 16, 512)       2359808
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 8, 8, 512)         0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 8, 8, 512)         2359808
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 8, 8, 512)         2359808
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 8, 8, 512)         2359808
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0
_________________________________________________________________
dense (Dense)                (None, 4090)              33509370
_________________________________________________________________
dense_1 (Dense)              (None, 4090)              16732190
_________________________________________________________________
dense_2 (Dense)              (None, 9)                 36819
=================================================================
Total params: 64,993,067
Trainable params: 64,993,067
Non-trainable params: 0
```

Figure 5: Summary of the implemented VGG-16 architecture

- **Results**:



```
Epoch 1/10
450/450 [==============================] - 188s 394ms/step - loss: 0.8098 - accuracy: 0.1358 - val_loss: 0.3287 - val_accuracy:
0.2028
Epoch 2/10
450/450 [==============================] - 46s 103ms/step - loss: 0.2935 - accuracy: 0.3222 - val_loss: 0.2448 - val_accuracy:
0.4633
Epoch 3/10
450/450 [==============================] - 47s 104ms/step - loss: 0.2232 - accuracy: 0.5461 - val_loss: 0.1914 - val_accuracy:
0.6267
Epoch 4/10
450/450 [==============================] - 47s 104ms/step - loss: 0.1477 - accuracy: 0.7197 - val_loss: 0.1412 - val_accuracy:
0.7394
Epoch 5/10
450/450 [==============================] - 47s 104ms/step - loss: 0.1037 - accuracy: 0.8090 - val_loss: 0.0931 - val_accuracy:
0.8289
Epoch 6/10
450/450 [==============================] - 47s 104ms/step - loss: 0.0750 - accuracy: 0.8722 - val_loss: 0.1189 - val_accuracy:
0.7989
Epoch 7/10
450/450 [==============================] - 47s 105ms/step - loss: 0.0579 - accuracy: 0.9068 - val_loss: 0.0929 - val_accuracy:
0.8544
Epoch 8/10
450/450 [==============================] - 47s 104ms/step - loss: 0.0490 - accuracy: 0.9211 - val_loss: 0.0858 - val_accuracy:
0.8733
Epoch 9/10
450/450 [==============================] - 47s 105ms/step - loss: 0.0459 - accuracy: 0.9268 - val_loss: 0.0667 - val_accuracy:
0.8928
Epoch 10/10
450/450 [==============================] - 47s 105ms/step - loss: 0.0345 - accuracy: 0.9467 - val_loss: 0.0788 - val_accuracy:
0.8983
```

Figure 6: Training results for 10 epochs

1. Accuracy vs Epoch:

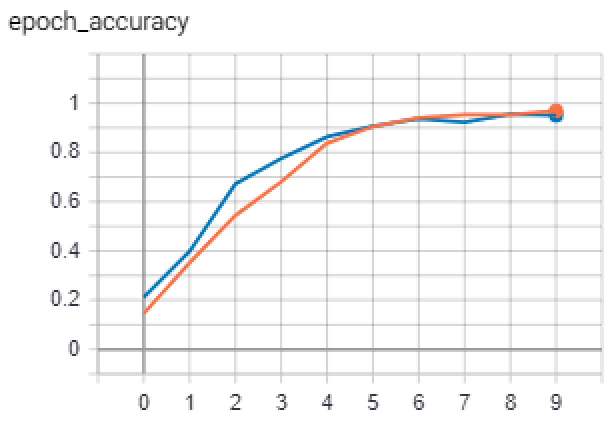


Figure 7: Accuracy vs Epoch plot

The accuracy of the network increases as the number of epochs increases and with it the data provided to the network increases.

2. Loss vs Epoch:

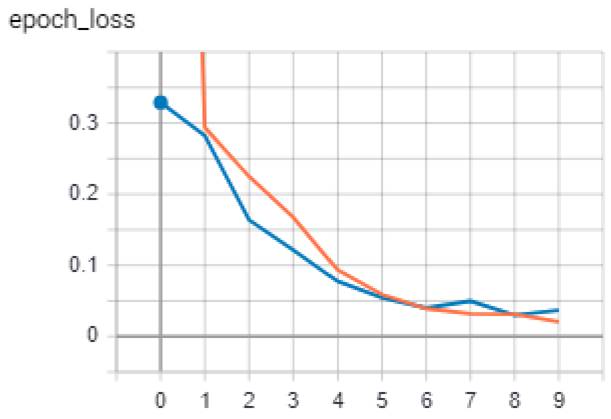


Figure 8: Loss vs Epoch plot

Contrary to accuracy, the loss decreases as the data and the number of epochs increase.

1) After Standardizing the data input, the accuracy of the network was increased.

2) Increasing the batch size and the number of epochs increased the accuracy of the network.

3)The accuracy increased and the fluctuations in every epoch were decreased after decaying the learning rate.

<u>References</u>:

[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)