ENPM809T Project 4 Report

Problem 1:

Goal:
To use the concept of optical flow to track the motion of vehicles on a highway. The goals of this problem are to plot the vector field of the optical flow for every frame in the video and to remove the background and show only the movement of cars on the highway

Approach:
Optical flow refers to the per pixel motion estimation between two consecutive frames in a video. Optical flow can be estimated by using the Lucas-Kanade method which works on the assumption that nearby pixels have the same displacement direction. The lucas-kanade algorithm suffers from avrupt movements.

Theory
If the neighboring pixels have the same motion vector a fixed-size window can be used to create a system of equations. Let $p_i = (x_i, y_i)$ be the pixel coordinates in the chosen window. Hence, we can define the equation system as:

$$\begin{cases} I'_x(p_1)u + I'_y(p_1)v = -I'_t(p_1) \\ I'_x(p_2)u + I'_y(p_2)v = -I'_t(p_2) \\ \dots \\ I'_x(p_n)u + I'_y(p_n)v = -I'_t(p_n) \end{cases}$$

This equation system can be rewritten in matrix form:

$$A = \begin{bmatrix} I'_x(p_1) & I'_y(p_1) \\ I'_x(p_2) & I'_y(p_2) \\ \vdots & \vdots \\ I'_x(p_n) & I'_y(p_n) \end{bmatrix}, \ \gamma = \begin{bmatrix} dx \\ dy \end{bmatrix}, b = \begin{bmatrix} -I'_t(p_1)dt \\ -I'_t(p_2)dt \\ \vdots \\ -I'_t(p_n)dt \end{bmatrix}$$

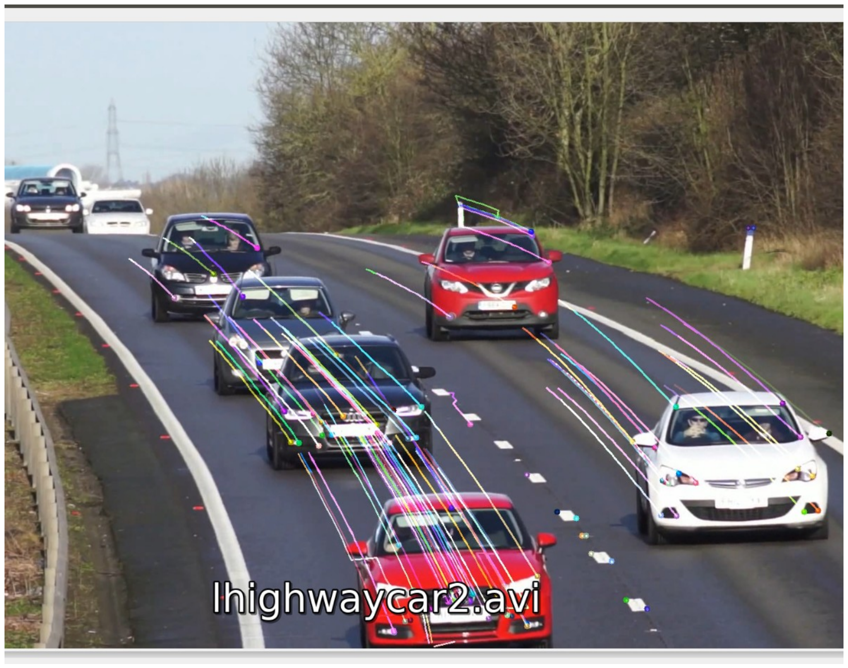
As a result, we have a matrix equation: $A\gamma = b$. Using the Least Squares we can compute the answer vector γ :

$$A^TA\gamma = A^Tb \quad \Rightarrow \quad \underbrace{(A^TA)^{-1}(A^TA)}_{Identity\, matrix}\gamma = (A^TA)^{-1}A^Tb \quad \Rightarrow \quad \gamma = (A^TA)^{-1}A^Tb$$

This can be solved by an approach where every next image will be bigger than the previous one by some scaling factor. The displacement vectors in the smaller images will be used on the larger images for better results.

The steps I have taken to implement the algorithm are as follows:
1. Read the first frame of the video and use the shi-tomasi corner detection algorithm to store the corner points.
2. Read the subsequent frames and use the inbuilt function cv2.calcOpticalFlowPyrLK() to extract the points and the error.
3. The lines are drawn connecting the points from the current frame to the previous frame to represent the vector field.

The below is the result which I obtained with the lines representing the vector field.



4. The background removal was performed by using the inbuilt function cv2.bgsegm.createBackgroundSubtractorMOG() and applying it to every frame of the video. This would create a grayscale image with the background of the highway being darkened and the cars can be clearly seen as it drives along the highway.

Results:
The two videos can be seen in the attached links.
https://youtu.be/KgKM0lsJvDU
https://youtu.be/RiwrNDQIxD0

Problem 2

Goal:
To classify images of 9 different seafood types by implementing a Convolutional Neural Network employing the VGG-16 architecture using Tensorflow and Keras.

Methodology:

Convolutional Neural Network
The Convolution Neural Network(CNN) is a deep learning algorithm which takes an input image, extracts the different features and prioritizes different features through weights and biases and the most important aspect being capable of differentiating different images. Therefore it makes sense to use this algotihm for our project which consists of a dataset with 9 different classes ie 9 different varieties of fish by splitting the data into training and testing, training the network to identify different features and validating on the test dataset to verify the results. CNN being a feedforward network can capture both the sptial and temporal dependencies in an image by using different filters.

The objectives of the algorithm are to extract the high level features. This is where the different layers of the network comes into play.

The first layer is the convolution layer which captures low level features such as edges, color, etc. The results in this process involve a reduction in conditionality as compared to the input and the other in which the dimensionality is increased.

The second layer of the network is known as the pooling layer which reduces the spatial size of the convolved feature. By doing this the performance is improved as less computation is required to process the data. This layer also extracts dominant features which are rotational and positional invariant. Pooling can take place in two ways:
1. Max pooling which returns the greatest value from the portion of the image covered by the kernel.
2. Average pooling which returns the average of all the values from the image covered by the kernel.

The other layer is a fully connected layer which learns non linear combinations of the high level features.

Once the image has passed through these layers the image goes into the feedforward neural network with backpropagation applied in every iteration.

VGG-16 Architecture

The purpose of VGG on the depth of convolutional networks is to understand how the depth of convolutional networks affects the accuracy and accuracy of large-scale image classification and recognition. VGG-16 is a convolution neural network configuration architecture with 16 layers. The width of convolution layers is small, starting from 64 in the first layer and then increasing by a factor of 2 after each max-pooling layer, until it reaches 512. VGG-16 has certain drawbacks due to a large number of computations being carried out in the different layers, The 16 layers of VGG16 are as follows:

1.Convolution using 64 filters
2.Convolution using 64 filters + Max pooling
3.Convolution using 128 filters
4. Convolution using 128 filters + Max pooling
5. Convolution using 256 filters
6. Convolution using 256 filters
7. Convolution using 256 filters + Max pooling
8. Convolution using 512 filters
9. Convolution using 512 filters
10. Convolution using 512 filters+Max pooling
11. Convolution using 512 filters
12. Convolution using 512 filters
13. Convolution using 512 filters+Max pooling
14. Fully connected with 4096 nodes
15. Fully connected with 4096 nodes
16. Output layer with Softmax activation with 1000 nodes.

Algorithm Implementation

Tools

I worked on a google colab notebook using Tensorflow and Keras libraries to generate the VGG16 deep learning model.

Data Split

The project requires 80% of the data to be considered for training purposes and 20% for testing purposes. I put all the images of the data into a single folder and used the command `train_imgs, test_imgs, train_labels, test_labels = train_test_split(img_arr, categories, test_size = .2)` to split up the data into training and test data.

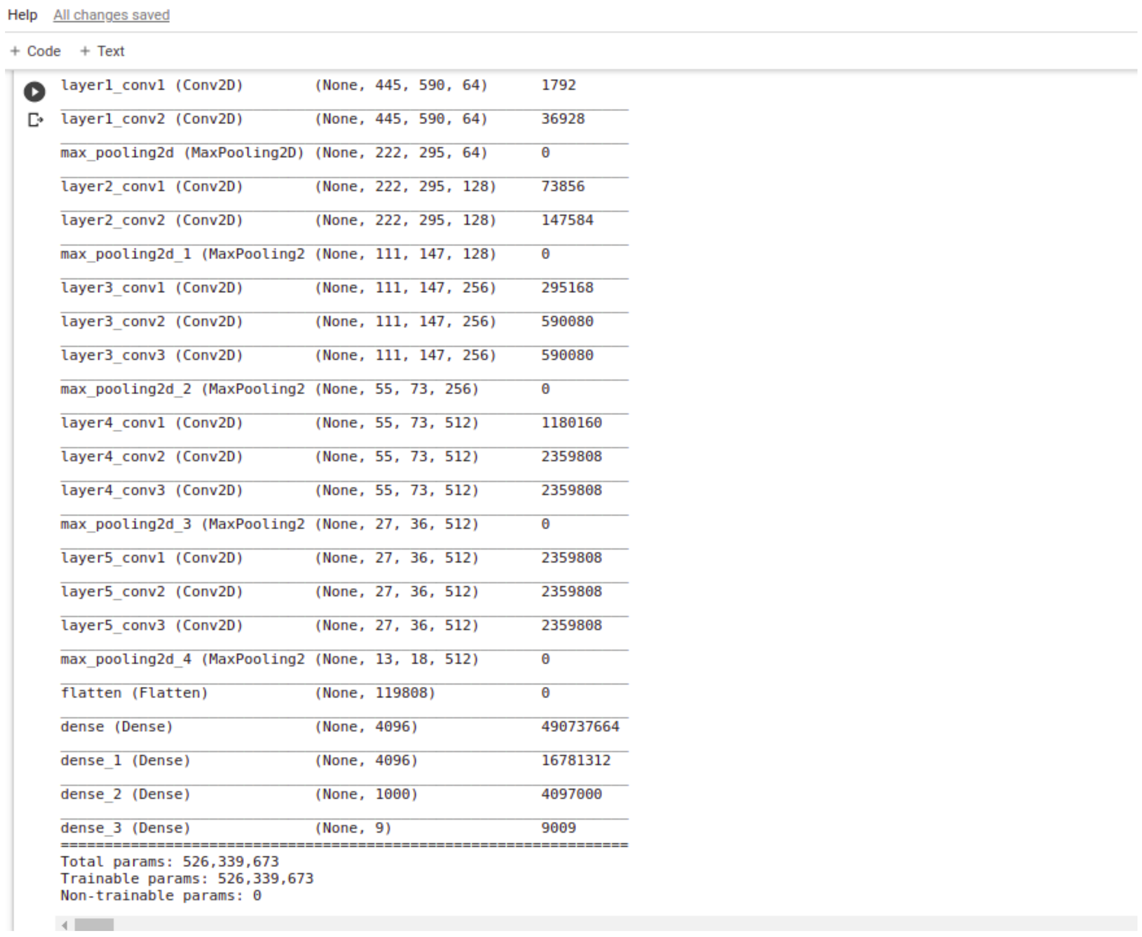
Creating labels

The first step in the pipeline requires creating labels to identify the different types of fish. I assigned labels of 0-8 for the 9 different types of fish. The images are subsequently displayed using matplotlib along with the names of the image. A label binarizer is used to assign the assign the class for which the model has the highest confidence.

Initialization of the model

After the Keras libraries are imported the vgg16 architecture needs to be implemented from scratch. As can be seen in the below lines of code a sequential model is created.] which takes into account the image size and initializes the different layers of the convolutional neural network. The ReLu activate function is used which ensures that negative values are not passed onto the next layer and makes the value zero if that is the case. The Relu activation function uses stochastic gradient descent with backpropagation to train neural networks. Stochastic gradient descent is an approximation of the gradient descent since it replaces the actual graident by an estimated gradient. Post which the data is passed onto the dense layer. A softmax layer is used to output a value based on the number of classes in this case which is 9.

```
model = keras.models.Sequential([
keras.layers.InputLayer(input_shape=img_shape),
# keras.layers.BatchNormalization(),
keras.layers.Conv2D(filters=64,kernel_size=3, activation="relu",
padding="same", name='layer1_conv1'),
keras.layers.Conv2D(filters=64,kernel_size=3, activation="relu",
padding="same", name='layer1_conv2'),
keras.layers.MaxPooling2D(pool_size=2, strides=2),
# keras.layers.BatchNormalization(),
keras.layers.Conv2D(filters=128,kernel_size=3, activation="relu",
padding="same", name='layer2_conv1'),
keras.layers.Conv2D(filters=128,kernel_size=3, activation="relu",
padding="same", name='layer2_conv2'),
keras.layers.MaxPooling2D(pool_size=2, strides=2),
# keras.layers.BatchNormalization(),
keras.layers.Conv2D(filters=256,kernel_size=3, activation="relu",
padding="same", name='layer3_conv1'),
keras.layers.Conv2D(filters=256,kernel_size=3, activation="relu",
padding="same", name='layer3_conv2'),
keras.layers.Conv2D(filters=256,kernel_size=3, activation="relu",
padding="same", name='layer3_conv3'),
keras.layers.MaxPooling2D(pool_size=2, strides=2),
# keras.layers.BatchNormalization(),
keras.layers.Conv2D(filters=512,kernel_size=3, activation="relu",
padding="same", name='layer4_conv1'),
keras.layers.Conv2D(filters=512,kernel_size=3, activation="relu",
padding="same", name='layer4_conv2'),
keras.layers.Conv2D(filters=512,kernel_size=3, activation="relu",
padding="same", name='layer4_conv3'),
keras.layers.MaxPooling2D(pool_size=2, strides=2),
# keras.layers.BatchNormalization(),
keras.layers.Conv2D(filters=512,kernel_size=3, activation="relu",
padding="same", name='layer5_conv1'),
keras.layers.Conv2D(filters=512,kernel_size=3, activation="relu",
padding="same", name='layer5_conv2'),
keras.layers.Conv2D(filters=512,kernel_size=3, activation="relu",
padding="same", name='layer5_conv3'),
keras.layers.MaxPooling2D(pool_size=2, strides=2),
keras.layers.Flatten(),
keras.layers.Dense(4096, activation='relu'),
keras.layers.Dense(4096, activation='relu'),
keras.layers.Dense(1000, activation='relu'),
keras.layers.Dense(num_classes, activation='softmax')
```



```
layer1_conv1 (Conv2D)        (None, 445, 590, 64)      1792
_________________________________________________________________
layer1_conv2 (Conv2D)        (None, 445, 590, 64)      36928
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 222, 295, 64)      0
_________________________________________________________________
layer2_conv1 (Conv2D)        (None, 222, 295, 128)     73856
_________________________________________________________________
layer2_conv2 (Conv2D)        (None, 222, 295, 128)     147584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 111, 147, 128)     0
_________________________________________________________________
layer3_conv1 (Conv2D)        (None, 111, 147, 256)     295168
_________________________________________________________________
layer3_conv2 (Conv2D)        (None, 111, 147, 256)     590080
_________________________________________________________________
layer3_conv3 (Conv2D)        (None, 111, 147, 256)     590080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 55, 73, 256)       0
_________________________________________________________________
layer4_conv1 (Conv2D)        (None, 55, 73, 512)       1180160
_________________________________________________________________
layer4_conv2 (Conv2D)        (None, 55, 73, 512)       2359808
_________________________________________________________________
layer4_conv3 (Conv2D)        (None, 55, 73, 512)       2359808
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 27, 36, 512)       0
_________________________________________________________________
layer5_conv1 (Conv2D)        (None, 27, 36, 512)       2359808
_________________________________________________________________
layer5_conv2 (Conv2D)        (None, 27, 36, 512)       2359808
_________________________________________________________________
layer5_conv3 (Conv2D)        (None, 27, 36, 512)       2359808
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 13, 18, 512)       0
_________________________________________________________________
flatten (Flatten)            (None, 119808)            0
_________________________________________________________________
dense (Dense)                (None, 4096)              490737664
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              4097000
_________________________________________________________________
dense_3 (Dense)              (None, 9)                 9009
=================================================================
Total params: 526,339,673
Trainable params: 526,339,673
Non-trainable params: 0
```

Model Compilation

Once the model has been prepared the model is compiled by making use of an optimizer.

```
optimizer = tf.keras.optimizers.Adam(lr=1e-5)
model.compile(optimizer=optimizer,
loss='categorical_crossentropy',
metrics=['accuracy'])
```

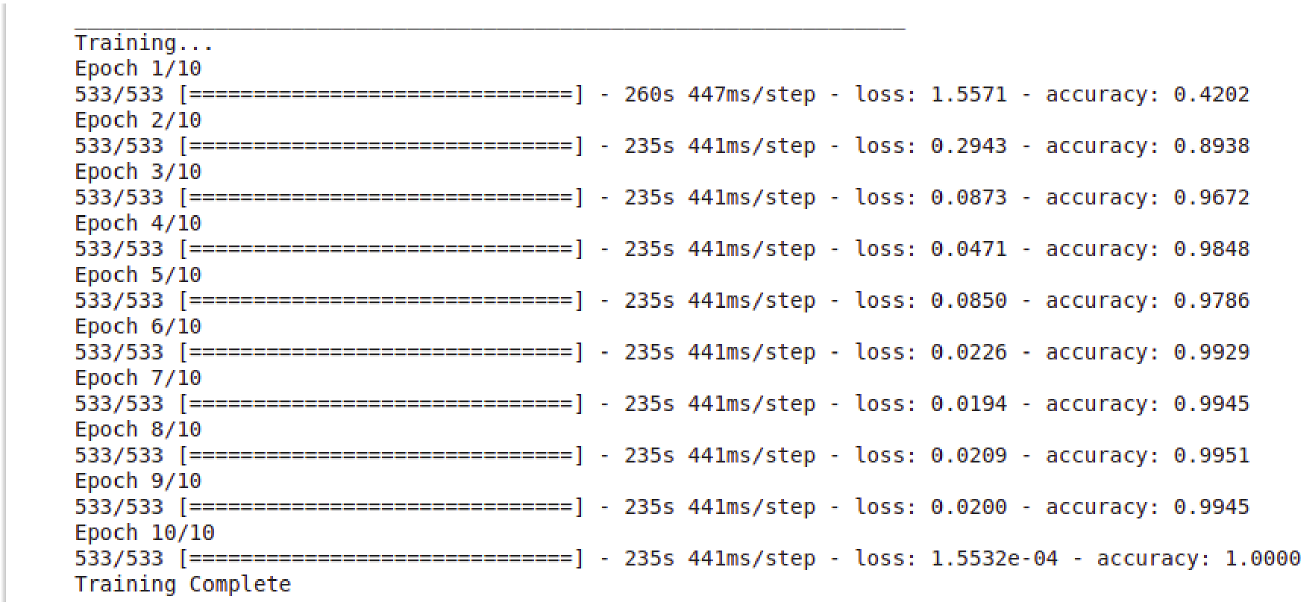
The optimizer I have used is the Adam optimizer which is a stochastic gradient descent method that is based on adaptive estimation of first and second order movements. I just feed in the learning rate as a factor in this algorithm. Initially I considered the SGD optimizer where I had momentum as one of the parameters but I was running into different errors while doing the model fitting post which I experimented with the Adam optimizer. For the loss parameter in model.compile I used a categorical entropy since there are two or more classes to predict.

Model Fitting

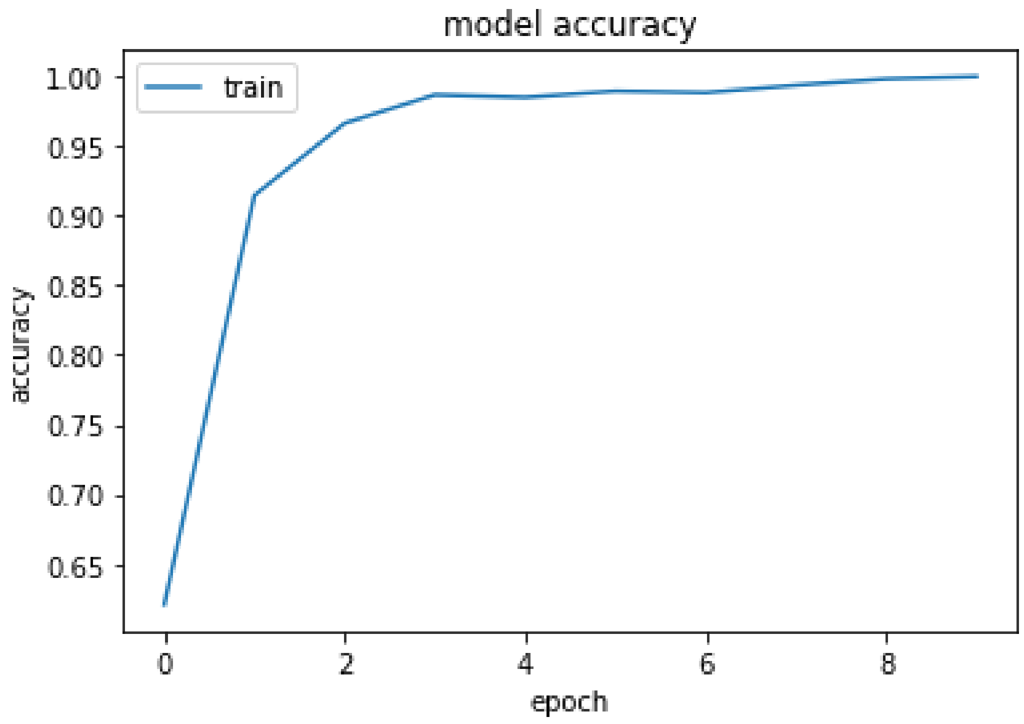
The model train the model on the training data by passing the input images along with the labels for each image. The below function model.fit trains the model for a fixed number of epochs. For training purposes I have used a batch size of 10 and 10 epochs to train the data. I was intially using the model.fit_generator which trains the model on different generators which was used on the training data which did not yield significant results for me.

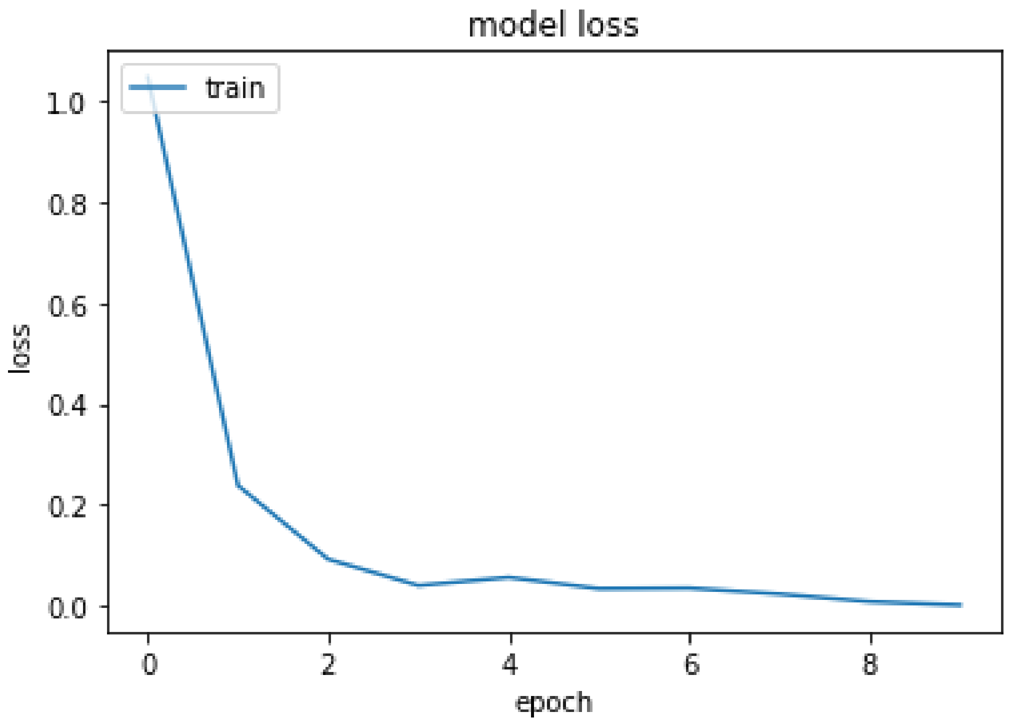
```
history = model.fit(x=np.array(train_imgs),
y=y_train,
epochs=10, batch_size=10)
```

I was only able to experiment with a batch size of 10 after using the model.fit function but I got very good results on the training accuracy. The below figures show the plots for accuracy and epoch and loss and epoch for the training data.



```
_________________________________________________________________
Training...
Epoch 1/10
533/533 [==============================] - 260s 447ms/step - loss: 1.5571 - accuracy: 0.4202
Epoch 2/10
533/533 [==============================] - 235s 441ms/step - loss: 0.2943 - accuracy: 0.8938
Epoch 3/10
533/533 [==============================] - 235s 441ms/step - loss: 0.0873 - accuracy: 0.9672
Epoch 4/10
533/533 [==============================] - 235s 441ms/step - loss: 0.0471 - accuracy: 0.9848
Epoch 5/10
533/533 [==============================] - 235s 441ms/step - loss: 0.0850 - accuracy: 0.9786
Epoch 6/10
533/533 [==============================] - 235s 441ms/step - loss: 0.0226 - accuracy: 0.9929
Epoch 7/10
533/533 [==============================] - 235s 441ms/step - loss: 0.0194 - accuracy: 0.9945
Epoch 8/10
533/533 [==============================] - 235s 441ms/step - loss: 0.0209 - accuracy: 0.9951
Epoch 9/10
533/533 [==============================] - 235s 441ms/step - loss: 0.0200 - accuracy: 0.9945
Epoch 10/10
533/533 [==============================] - 235s 441ms/step - loss: 1.5532e-04 - accuracy: 1.0000
Training Complete
```

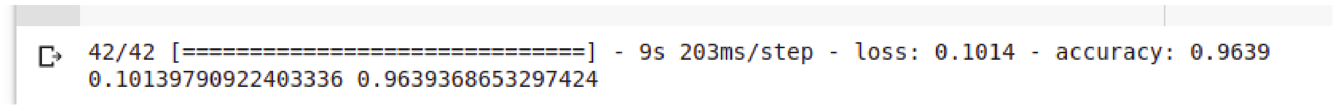

model accuracy
train
accuracy
epoch
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0
2
4
6
8


model loss
train
loss
epoch
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8

Model Evaluation and verification of test data
Now that the model has been trained, it is important to use the model on the testing data to verify the accuracy and loss and also to predict the results for the different test images.

The model is firstly evaluate on the test data to verify the validity of the training. The accuracy of the model and the loss on the test data is displayed as well using the below function which returns the accuracy and loss values which can be plotted with respect to the epochs.

```
model.evaluate(x=np.array(test_imgs), y=y_test)
```

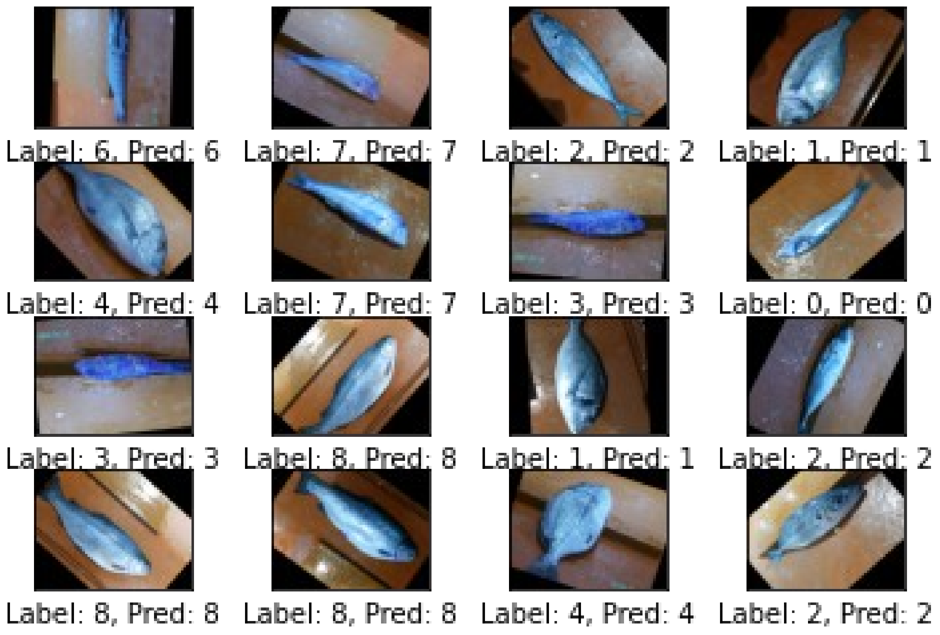


```
42/42 [==============================] - 9s 203ms/step - loss: 0.1014 - accuracy: 0.9639
0.10139790922403336 0.9639368653297424
```

Prediction
A prediction is run on the model with test images to verify the different types of seafood. The below function returns an array of predictions for the different classes of seafood and the index with the maximum value is extracted.

```
np.argmax(model.predict(np.array(test_imgs)), axis=-1)
```

The below figure which consists of 16 images along with its predicted and original label is shown.



I had got an output of integers depicting the labels. But due to time constraints I wasn't able to convert them to strings. But from the diagram we can observe that the predicted label indicated by 'Pred' matches the original label of the test image indicated by 'Label'. This shows that the model is capable of classifying the images of the different types of seafood after being trained on 7200 images and testing being carried out on the remaining 1200 images.

The below are some of the references I used to compile this report.

References

O.Ulucan, D.Karakaya, and M.Turkan.(2020) A large-scale dataset for
fish segmentation and classification. In Conf. Innovations Intell. Syst.
Appli. (ASYU)
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
https://learnopencv.com/optical-flow-in-opencv/
https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c
https://www.tensorflow.org/
https://keras.io/