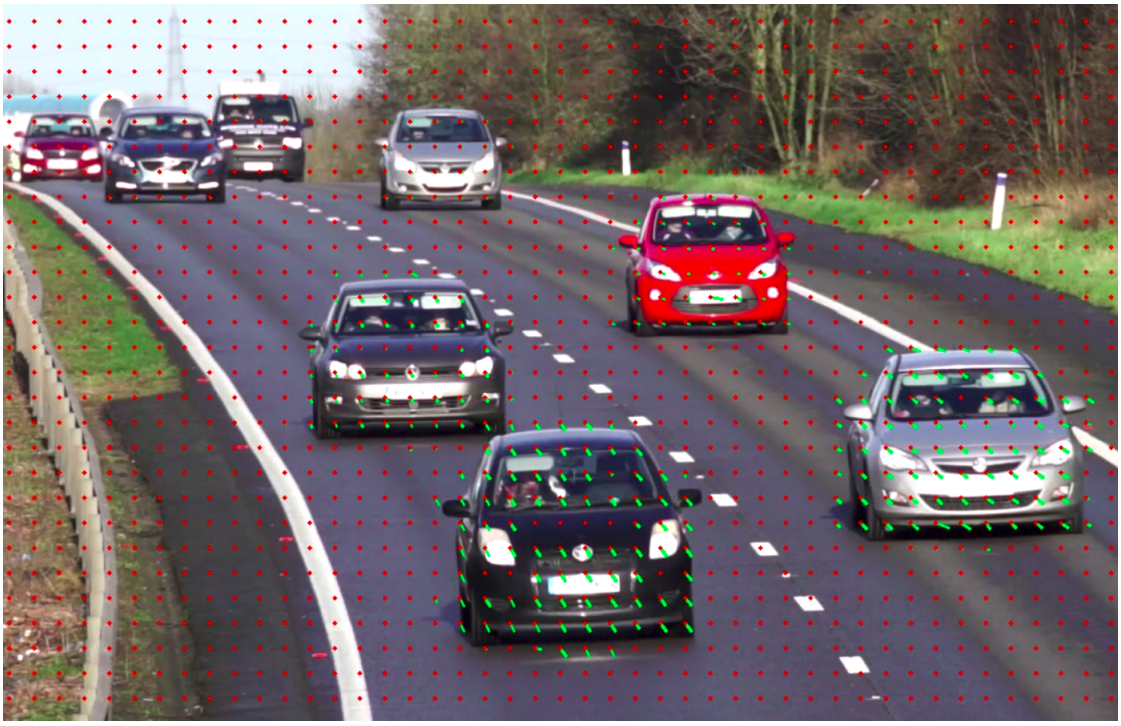
PROJ4 REPORT

ENPM673 - Perception For Autonomous Robots - Spring 21



Paras Savnani

05.17.2021

University Of Maryland College Park

Problem 1:

Objective: Track the motion of the vehicles on a highway using the concept of Optical Motion Flow.

CODE EXPLANATION AND PROCEDURE:

main.py

1. Firstly, the video file is loaded using the **cv2.VideoCapture()** function and the frame is resized to 75% to improve video processing time.
2. For optical flow a dummy hsv image is created and all the saturation values are assigned the value 255.
3. In the main while(1) loop the dense optical flow is calculated between the current frame and the previous frame using the function **cv2.calcOpticalFlowFarneback().**
4. Next, the flow array is converted from cartesian coordinates to polar and the results are assigned to the dummy hsv image that was created before.
5. Further the hsv image is converted to bgr for visualization.
6. For vector flow visualization the **draw_flow** function is used to show the vectors and their magnitudes.

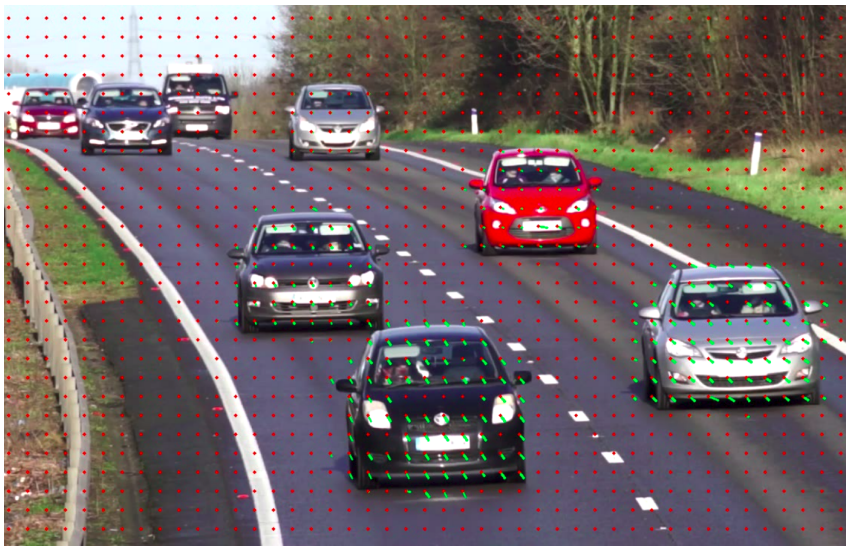
def draw_flow(img, flow, step):

This function is responsible for drawing the vector flow field, it takes image, flow array and step size as the input to draw the flow vector field.

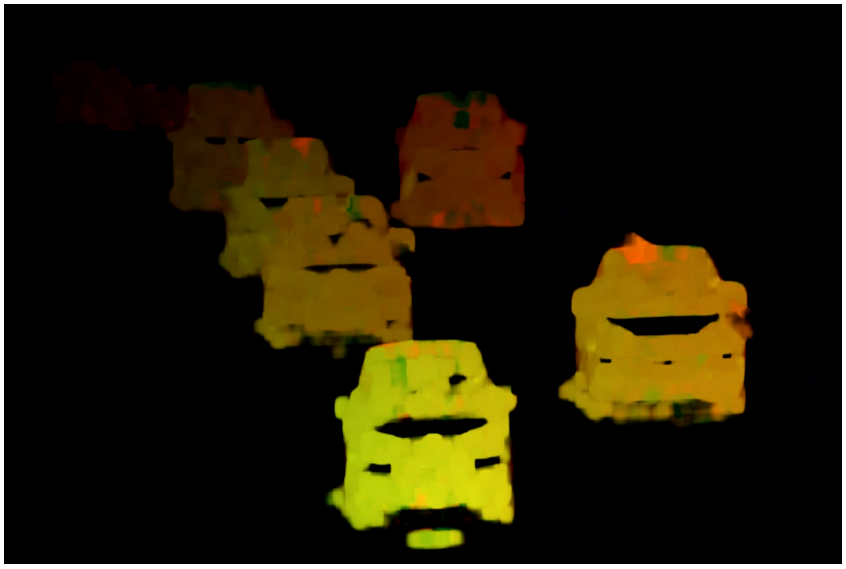
1. Firstly, it extracts the height and width of the image.
2. Then, it constructs a numpy meshgrid using the image size and the step size information.
3. Next, it takes the x and y vectors from the flow array by transposing it.
4. After this it creates a lines array by vertically stacking the meshgrid points and the flow vector coordinates.

5. Following this it iterates through all the lines and plots them using **cv2.polylines(),** we can also use **cv2.arrowedLine()** too. It also plots the circles at the step size distance for visualization.

RESULTS:



Vector field of the Optical flow



Movement of Cars after removing the background

Problem 2

Objective: Implement a CNN to perform classification on seafood dataset using deep learning frameworks such as tensorflow/pytorch and analyse the training process using the hyperparameter tuning.

CODE EXPLANATION AND PROCEDURE:

We will use the kearas framework to build the CNN for this problem. It uses tensorflow as its backend to create the different layers in the network. Aslo, we will train this on google colab to use the GPU compute power google provides to the users.

Problem2.ipynb

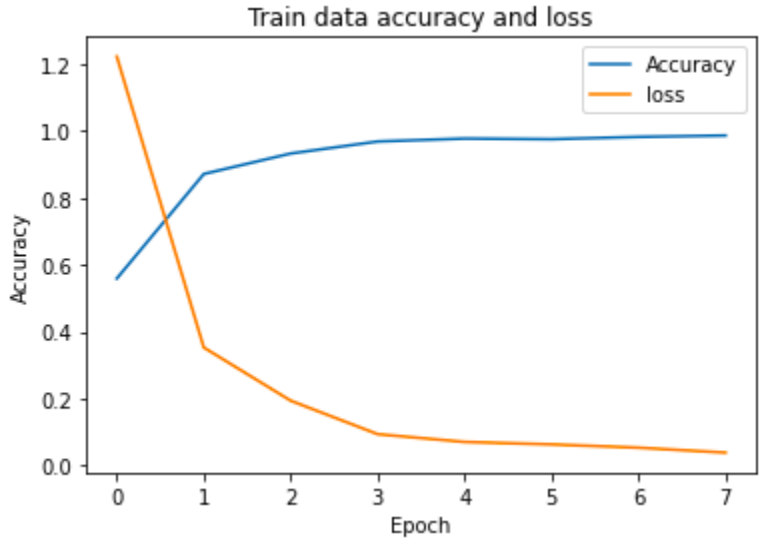
1. Firstly, we import the required library packages from keras module. It includes the functions to create convolution layers, max pool layers, imagedatagenerator(for preprocessing) etc.
2. Next, we will mount the google drive storage to the colab location using drive.mount(path) function.
3. Then, for the preprocessing we will create an object of ImageDataGenerator() with arguments rescale factor (1/255) and validation_split (0.2) to separate the data into training and test set.
4. After this, we use **flow_from_directory()** method to input the directory, batch size and input image size for the image dataset.
5. Following this we define the architecture of the CNN as:
    a. 2 Convolution layers
    b. 1 Max Pool layer
    c. 2 Convolution layers
    d. 1 Max Pool layer
    e. 1 Flattened layer
    f. 1 Dense layer
6. Next in mode.compile(), we define the loss function and the optimizer.

7. Now, when we check mode.summary(), we find the trainable and non-trainable parameters available.
8. Finally, we call model.fit_generator() to pass data one by one in the model and we also define the number of epochs for training and the early stopping condition.
9. After the training is complete we save the model using mode.save().
10. To evaluate the performance of the model, we plot the accuracy and loss vs number of epochs for both training and test data.
11. At last, if we are satisfied with the model architecture, we can use it to predict the classes. We load the model and randomly select an image from test data and predict the class in which it falls.

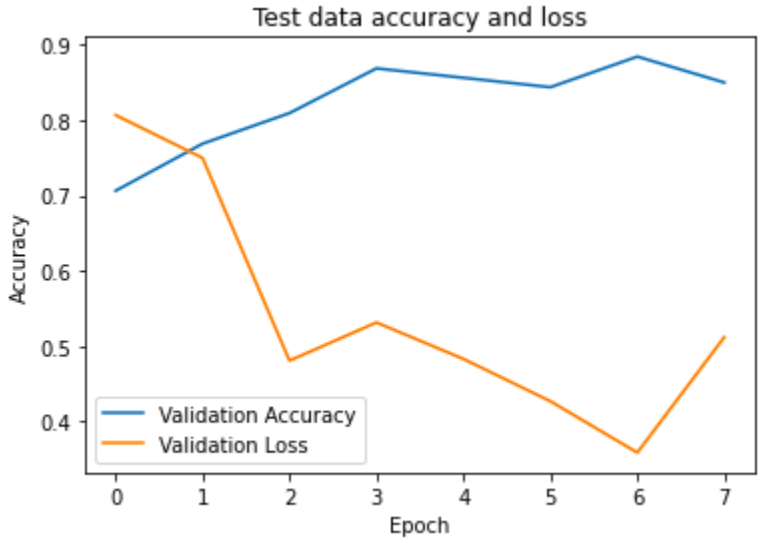
ANALYSIS:

- To improve the accuracy of the training images need to be preprocessed correctly. So, we **scale** the input images to maintain standardization in data.
- We also **shuffle** the train data and test data to avoid introducing any bias during the training of the data.
- Furthermore, we input the images in the CNN in **batches of size 32** which means we apply gradient descent on a batch of images and also we don't load all the images at once in memory.
- As the data is already augmented we **do not augment** it further because it will only increase redundancy in the training.
- Also, the original VGG16 architecture took very long to train because it had almost ~1e+7 parameters to train and the accuracy was also not stable enough, so we use a **custom smaller architecture** (941929 params) for our problem.
- The learning rate is kept at default (i.e **1e-3**) and we use **adam optimizer** with categorical_crossentropy loss.
- Apart from this, we train the network for 8 epochs with 100 steps per epoch, because the validation accuracy was not changing much after this and it was decreasing due to the **overtraining of the data**.

RESULTS:



Train data accuracy vs Loss Plot

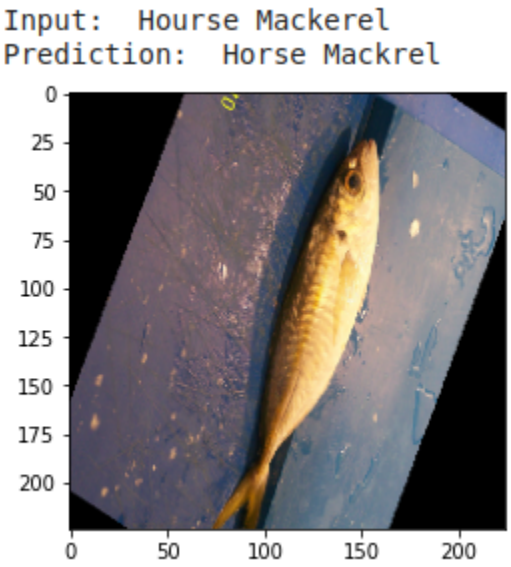


Test data accuracy vs Loss Plot

We get a train accuracy of **98.65%** and a train loss of **0.0383.** Also, we get a validation accuracy of **85.00%** and a validation loss of **0.5120.** It is evident from the training results that the validation loss is not decreasing further and if we continue the training it may result in the overfitting of the data. Thus, we stop the training at **8 epochs** with a

reasonable validation accuracy.

Below is the model output when we Input a **Hourse Mackrel** image and it also predicts the **Hourse Mackrel** image.



REFERENCES

- https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_video/py_lucas_kanade/py_lucas_kanade.html
- https://learnopencv.com/optical-flow-in-opencv/
- https://github.com/Hvass-Labs/TensorFlow-Tutorials
- https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c
- Dataset Citation: O.Ulucan, D.Karakaya, and M.Turkan.(2020) A large-scale dataset for fish segmentation and classification. In Conf. Innovations Intell. Syst. Appli. (ASYU)