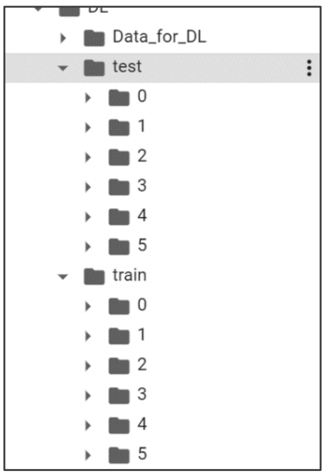
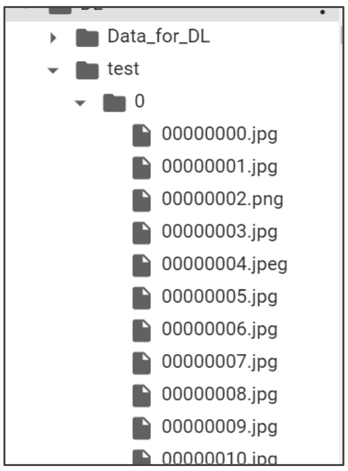
QUESTION 2

- The first task is to access the data in the drive and split the data to test and train and made available to access accordingly.

  - First, I mounted the google drive.
  - I copied the path of the whole data to 'parent_dir' and checked if it was working properly.
  - I assigned, train to be 70% of the data in each category and test to be the rest and ran a loop to create train and test data set for each category and assigned paths as well.



```
Data_for_DL
test
  0
  1
  2
  3
  4
  5
train
  0
  1
  2
  3
  4
  5
```



```
Data_for_DL
test
  0
    00000000.jpg
    00000001.jpg
    00000002.png
    00000003.jpg
    00000004.jpeg
    00000005.jpg
    00000006.jpg
    00000007.jpg
    00000008.jpg
    00000009.jpg
    00000010.jpg
```

  - Here, 0 to 5 can be indexed as the same category names as printed in the previous command, i.e., 0 for rocket to 5 for missile

- The next task was importing and modelling the VGG16 into the dataset

  - I imported necessary tools
  - The ImageDataGenerator is used to convert the images into a usable format to be trained into a model. For the test data, I have only rescaled to create uniform images and for the training data, I have augmented the pictures as given in the code: so a few images, would be horizontally flipped, few zoomed, rotated, etc.
  - Next, I assigned the path, i.e., the test and train data into the ImageDataGenerator, also giving the target size(224,224) since VGG16 has its first layer (224,224).

  - The next step I have done is added the train data and test data into the VGG16 Prediction generator. What I intended to do in this step is create a saved file with train and test data already gone through vgg model and then use this saved file for the future model change.

The reason I did this was, by running VGG16+mytrained model entirely each time, even in GPU it took hours for me, which I couldn't understand the reason or figure out the error. Since, no matter the output layer weights or number, the VGG layers are always pre-trained and fixed with the weights of ' Imagenet' , so running them both in train and test would not make any mistake. Basically what I did is splitted the training process into two parts, to save in between and so, the next processes can be run quicker and easier with appropriate optimiser, batch-size, layers, dropout on however I think the accuracy could be good.

So, the first part of the training, the VGG model is implemented and saved as an npy file on a new path.

- Next, I load this data as the new train data and test data for the rest of the model to be made. However, since earlier we have given the class attribute of ImageDataGenerator to be None, to get the data labels, I ran the data generator once again assigning the class as categorical and then assigned labels.

The next step is modelling the next layers

- Next, I add the trained data features from VGG16 to the sequential model. I added two dense layers and applied activation LeakyRelu, since relu alone didn't give much accuracy. I also added dropout in between these layers
- Now, I complied the model with optimiser Adam running on callback ReduceLR, to reduce learning rate and adapt to the best learning rate as the traning.
- Now the model is fit to training data and in order to see the test data changes per epoch, I assigned test data as validation.
- The model is run and we get 65% accuracy on test data.

Plotting Accuracy

- By using the function history we can access accuracy values on each epoch and plot the corresponding graph of train data, and how test data changes on each epoch

Confusion Matrix

- We can import confusion matrix tool from sklearn and then put in the predictions and original label and then plot, according to how we want to present.