readme

Training The Model

Installing the dependencies

`conda create -n cifar10`

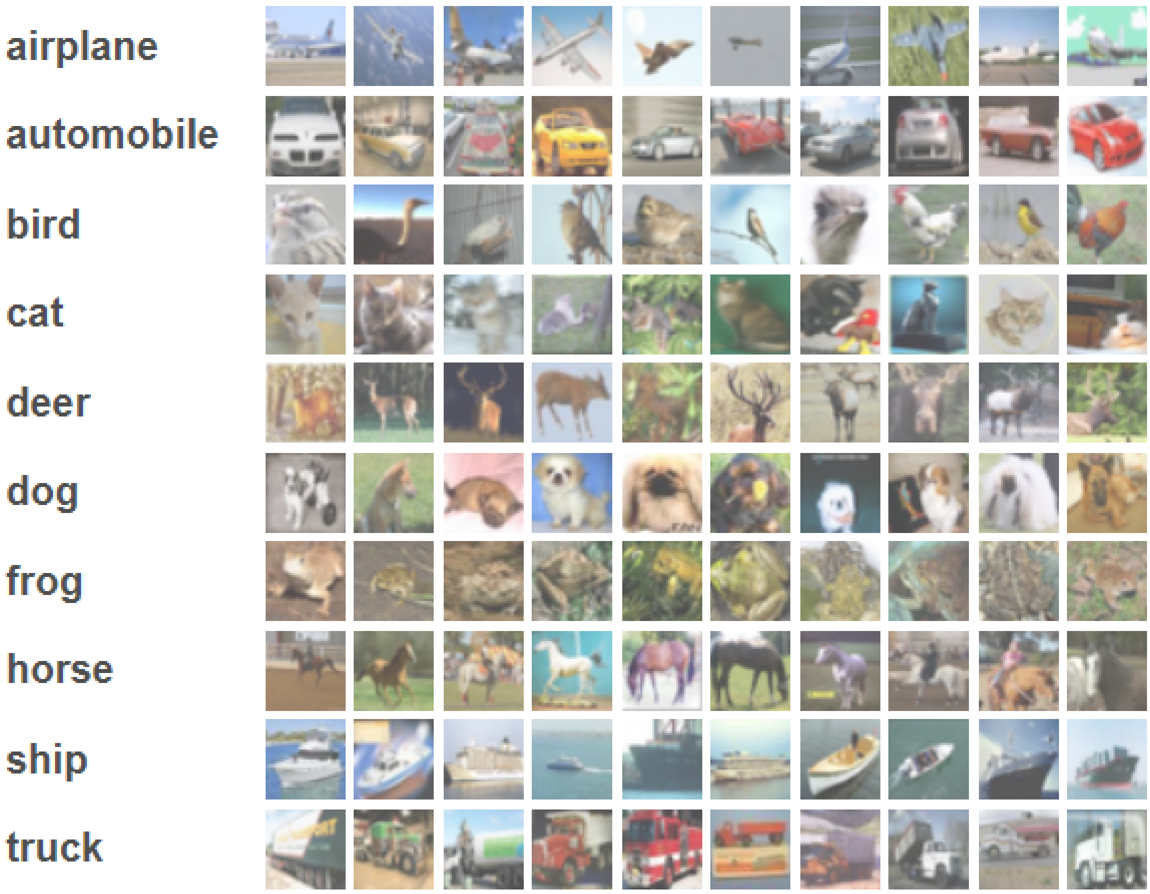
`pip install tensorflow tensorflow-gpu opencv-python os matplotlib seaborn`

`jupyter notebook`

```
The classes are:

| Label | Description |
|:-----:|-------------|
|   0   | airplane    |
|   1   | automobile  |
|   2   | bird        |
|   3   | cat         |
|   4   | deer        |
|   5   | dog         |
|   6   | frog        |
|   7   | horse       |
|   8   | ship        |
|   9   | truck       |

Returns:
  Tuple of NumPy arrays: `(x_train, y_train), (x_test, y_test)`.

**x_train**: uint8 NumPy array of grayscale image data with shapes
  `(50000, 32, 32, 3)`, containing the training data. Pixel values range
  from 0 to 255.

**y_train**: uint8 NumPy array of labels (integers in range 0-9)
  with shape `(50000, 1)` for the training data.

**x_test**: uint8 NumPy array of grayscale image data with shapes
  `(10000, 32, 32, 3)`, containing the test data. Pixel values range
  from 0 to 255.

**y_test**: uint8 NumPy array of labels (integers in range 0-9)
  with shape `(10000, 1)` for the test data.
```

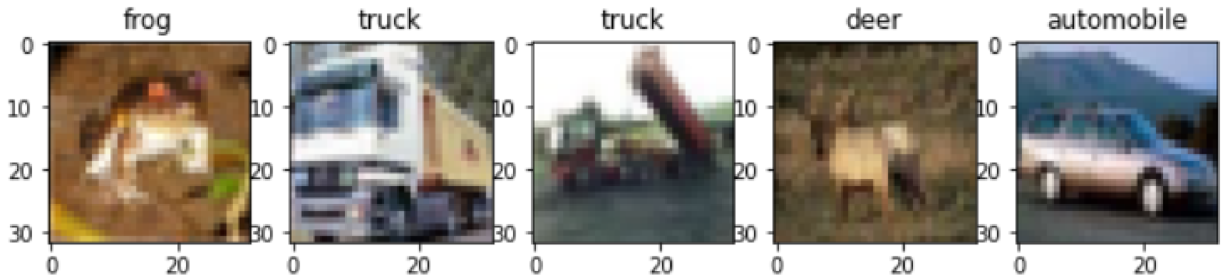
> Cifar10 is a dataset of 50,000 32x32 color training images and 10,000 test images, labeled over 10 categories. See more info at the [CIFAR homepage](https://www.cs.toronto.edu/~kriz/cifar.html)

Turns out that the images are already splitted

> The classes of the datasets are
> `classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]`
>
> 
>

Plotting the image

Image can be plotted with `matplotlib` library



The weird part is the image looks like Japanese video 😂😂😂

Pre-processing the data

1. Normalizing The Image's Arra
2. Reshapping the y to fit into the model

Architecture

***The most important part of the project is* CNN Model**

- The efficiency is tested with three different models

Model 0

- This is the first model which has

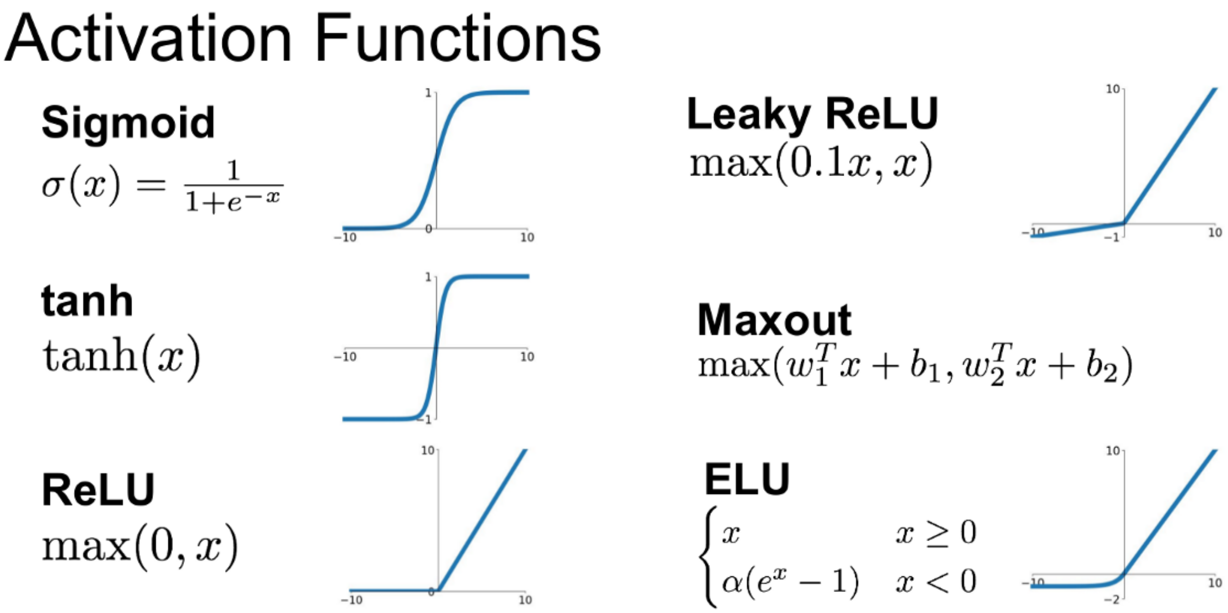
1. Convolutional Layer with 16 filters of size (3,3)
2. MaxPooling
3. Convolution Layer with 32 filters of (3,3) size with stride = 1
4. MaxPooling
5. Flatten
6. Fully Connected Layer with neurons =256
7. Final Output Layer with sigmoid activation

In [9]:
```
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_23 (Conv2D)          (None, 30, 30, 16)        448       
                                                                 
 max_pooling2d_21 (MaxPoolin  (None, 15, 15, 16)       0         
 g2D)                                                            
                                                                 
 conv2d_24 (Conv2D)          (None, 13, 13, 32)        4640      
                                                                 
 max_pooling2d_22 (MaxPoolin  (None, 6, 6, 32)         0         
 g2D)                                                            
                                                                 
 conv2d_25 (Conv2D)          (None, 4, 4, 64)          18496     
                                                                 
 max_pooling2d_23 (MaxPoolin  (None, 2, 2, 64)         0         
 g2D)                                                            
                                                                 
 flatten_7 (Flatten)         (None, 256)               0         
                                                                 
 dense_17 (Dense)            (None, 256)               65792     
                                                                 
 dense_18 (Dense)            (None, 10)                2570      
                                                                 
=================================================================
Total params: 91,946
Trainable params: 91,946
Non-trainable params: 0
_________________________________________________________________
```

`model.add(Conv2D(16, (3,3), 1, activation='relu', input_shape=(32,32,3)))`

- $$\lfloor \frac{n - f + 2p}{s} + 1 \rfloor$$

- Here n = 32 ,f =3 ,p =0 ,s=1
- $$\lfloor \frac{32-3}{1} + 1 \rfloor$$
- 30 is the width of the output layer of the first convolutional layer
- `activation = 'relu'` is the rectified linear unit which
- $$f(x) = max(0, x)$$
- 

Parameters

- First parameter = $$((333)+1)*16 = 448$$

```
MaxPooling2D(
    pool_size=(2, 2),
    strides=None,
    padding='valid',
    data_format=None,
    **kwargs,
)
```

- For **maxpooling**
- $$\lfloor \frac{254}{2} \rfloor = 127$$
- There wont be parameter for the max pooling.

For second conv2D

`model.add(Conv2D(32,(3,3),1,activation='relu'))`

means that **32** 3*3 filters of RGB(3) channels have stride =1

- $$\lfloor \frac{n-f+2p}{s} + 1 \rfloor$$

- Here n = 127 ,f =3 ,p =0 ,s=1
- $$\lfloor\frac{127-3}{1}+1\rfloor$$
- 125 is the width and height of its output layer
- The chanels would be 32
- Hence, the output shape =(None,125,125,32)
  Parameters

$$((f_H * f_W * n^{[C-1]}) + 1) * n^{[C]}$$

$$((3*3*16)+1)*32 = 4640$$

For max pooling

- $$\lfloor\frac{125}{2}\rfloor = 62$$

For third conv2D

`model.add(Conv2D(64,(3,3),1,activation='relu'))`
means that **64** 3*3 filters of RGB(3) channels have stride =1

- $$\lfloor\frac{n-f+2p}{s}+1\rfloor$$
- Here n = 62 ,f =3 ,p =0 ,s=1
- $$\lfloor\frac{62-3}{1}+1\rfloor$$
- 60 is the width and height of its output layer
- The chanels would be 64
- Hence, the output shape =(None,60,60,64)
  Parameters

$$((f_H * f_W * n^{[C-1]}) + 1) * n^{[C]}$$

$$((3*3*32)+1)*64 = 18496$$

`dense_17`

$$Parameters = 256*256+256*1 = 65792$$

`dense_18`

$$Parameters = 256*10+10*1 = 2570$$

`model.compile('adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])`
Adaptive Moment Estimation Algorithm optmiizer is used with loss = sparse_categorical_entropy

$$\text{Loss} = -\frac{1}{\substack{\text{output}\\ \text{size}}} \sum_{i=1}^{\substack{\text{output}\\ \text{size}}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

model 1 and model 2 have same architecture but they differ on the architecture only

Model 1

In [10]:

```
1 model1.summary()
```

```
Model: "sequential_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_32 (Conv2D)          (None, 30, 30, 8)         224

 max_pooling2d_30 (MaxPoolin  (None, 15, 15, 8)        0
 g2D)

 conv2d_33 (Conv2D)          (None, 13, 13, 16)        1168

 max_pooling2d_31 (MaxPoolin  (None, 6, 6, 16)         0
 g2D)

 conv2d_34 (Conv2D)          (None, 4, 4, 32)          4640

 max_pooling2d_32 (MaxPoolin  (None, 2, 2, 32)         0
 g2D)

 flatten_10 (Flatten)        (None, 128)               0

 dense_25 (Dense)            (None, 3000)              387000

 dense_26 (Dense)            (None, 1000)              3001000

 dense_27 (Dense)            (None, 10)                10010

=================================================================
Total params: 3,404,042
Trainable params: 3,404,042
Non-trainable params: 0
_________________________________________________________________
```

Model 2

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_35 (Conv2D)          (None, 30, 30, 8)         224

 max_pooling2d_33 (MaxPoolin  (None, 15, 15, 8)        0
 g2D)

 conv2d_36 (Conv2D)          (None, 13, 13, 16)        1168

 max_pooling2d_34 (MaxPoolin  (None, 6, 6, 16)         0
 g2D)

 conv2d_37 (Conv2D)          (None, 4, 4, 32)          4640

 max_pooling2d_35 (MaxPoolin  (None, 2, 2, 32)         0
 g2D)

 flatten_11 (Flatten)        (None, 128)               0

 dense_28 (Dense)            (None, 3000)              387000

 dense_29 (Dense)            (None, 1000)              3001000

 dense_30 (Dense)            (None, 10)                10010

=================================================================
Total params: 3,404,042
Trainable params: 3,404,042
Non-trainable params: 0
_________________________________________________________________
```

```
model_name.add(MaxPooling2D())
model_name.add(Conv2D(16,(3,3),1,activation='relu'))
model_name.add(MaxPooling2D())
model_name.add(Conv2D(32,(3,3),1,activation='relu'))
model_name.add(MaxPooling2D())
model_name.add(Flatten())
model_name.add(Dense(3000,activation='relu'))
model_name.add(Dense(1000,activation='relu'))
model_name.add(Dense(10,activation='softmax'))
```

Model is saved and loaded to/from the disk

```
model.save(os.path.join('models','model.h5'))
model = load_model(os.path.join('models','model.h5'))**strong text**
```

model.h5	1/29/2023 10:45 PM	H5 File	1,131 KB
model1.h5	1/29/2023 10:45 PM	H5 File	39,952 KB
model2.h5	1/29/2023 10:45 PM	H5 File	13,333 KB

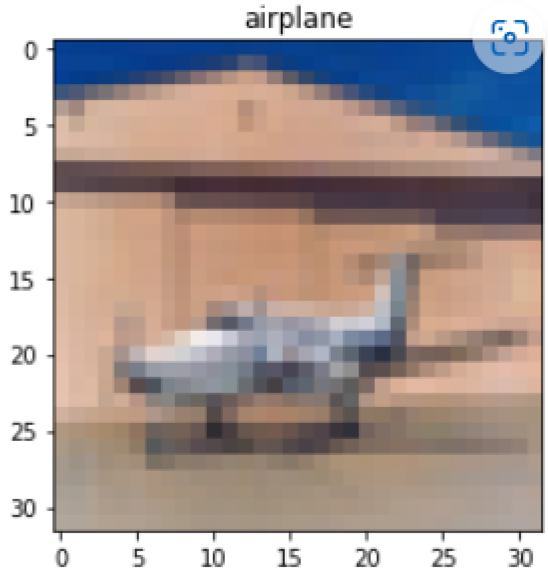
GitHub allows upto 100 MB files so the h5 files cant be included but please let me know if you need this

Testing the result 😄😄

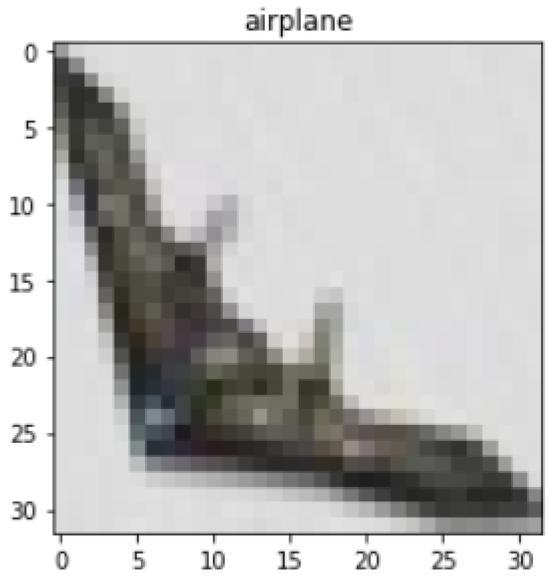
Parameter	Model 0	Model 1	Model 2
Accuracy After 10th	0.7955	0.7925	0.5813
Loss After 10th epoch	0.5730	0.5719	1.1919
Parameters	91,946	3,404,042	3,404,042

Randomly testing with test-images

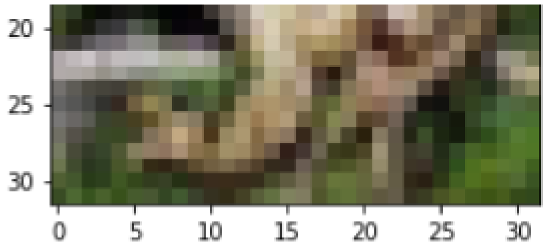
```
Actual image is  airplane 742 index
```



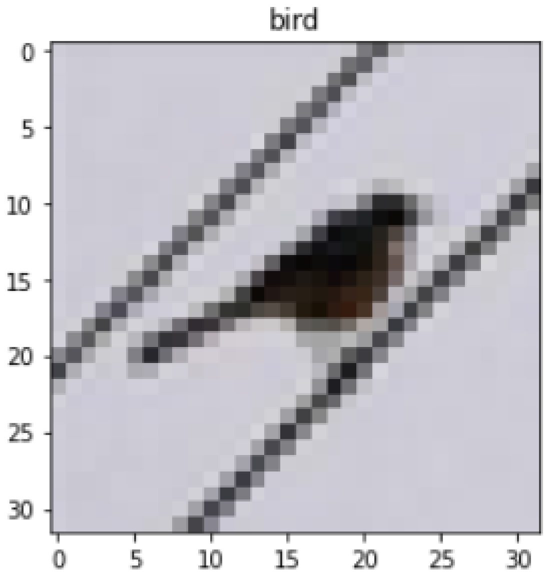

```
Model 0 predicted airplane
Model 1 predicted airplane
Model 2 predicted truck
********
Actual image is  airplane 180 index
```




```
Model 0 predicted airplane
Model 1 predicted airplane
Model 2 predicted airplane
********
Actual image is  frog 379 index
```

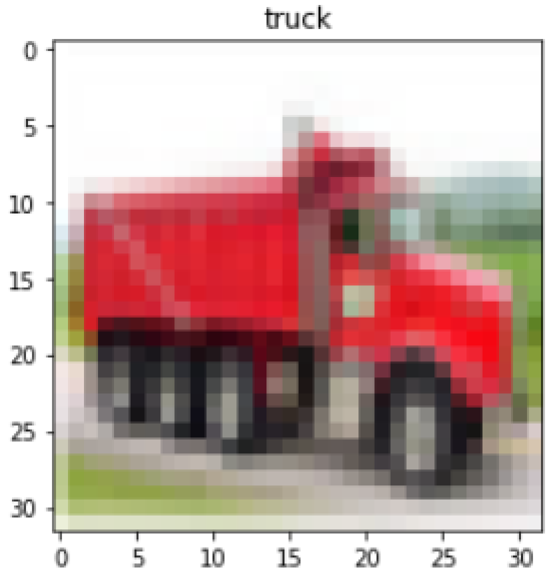


```
Model 0 predicted frog
Model 1 predicted frog
Model 2 predicted frog
********
Actual image is  bird 731 index
```

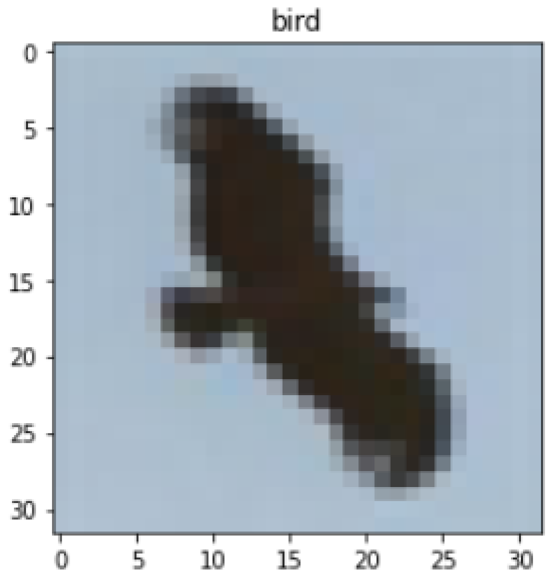



```
Model 0 predicted airplane
Model 1 predicted bird
Model 2 predicted bird
********
```

```
********
Actual image is  truck 609 index
```

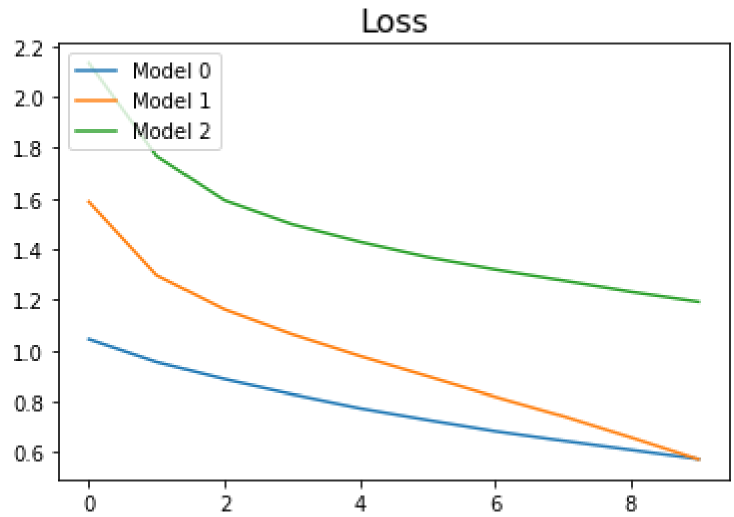



```
Model 0 predicted truck
Model 1 predicted truck
Model 2 predicted truck
********
Actual image is  bird 160 index
```

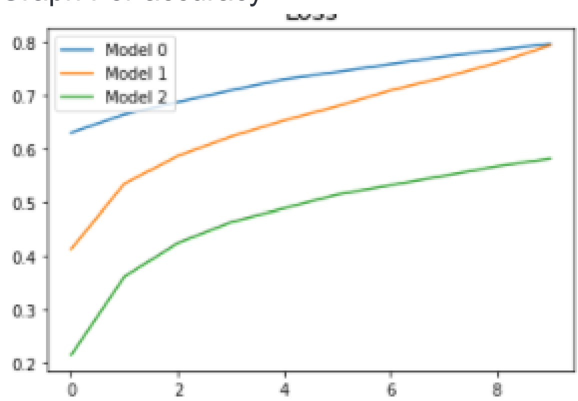



```
Model 0 predicted bird
Model 1 predicted bird
Model 2 predicted airplane
********
```

Graph of accuracy and loss

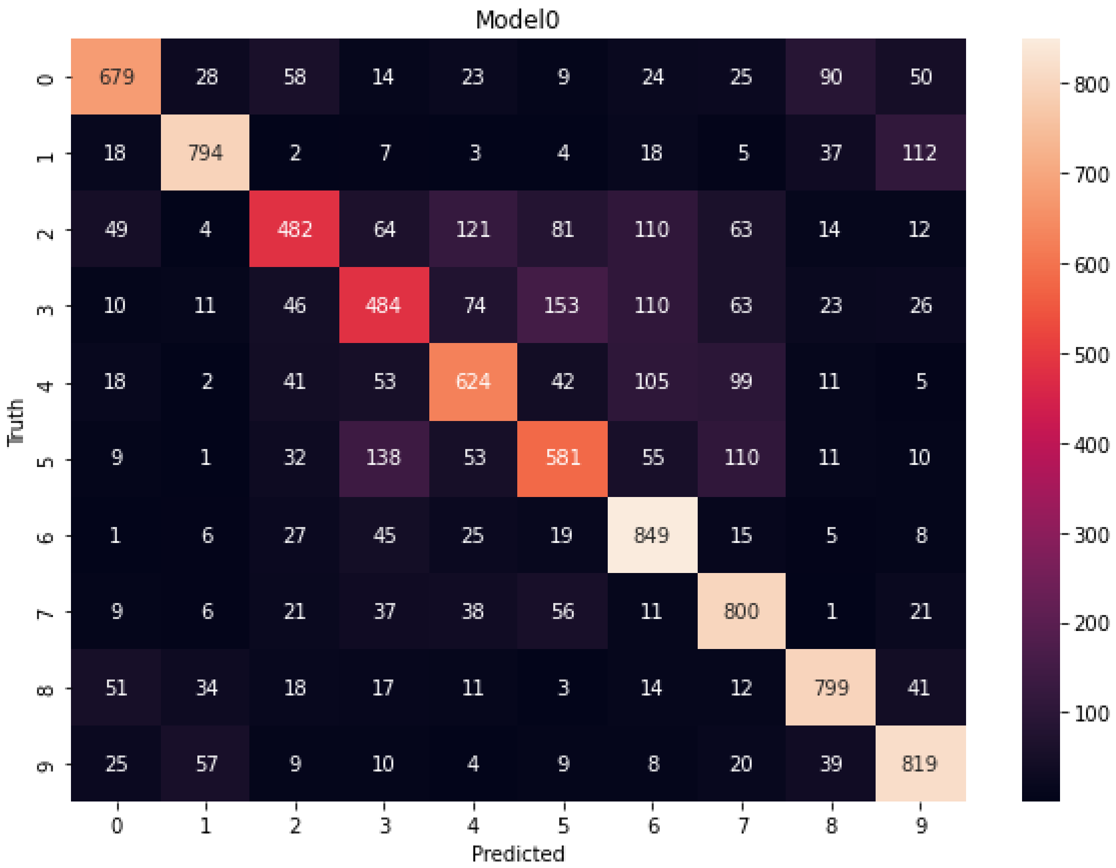


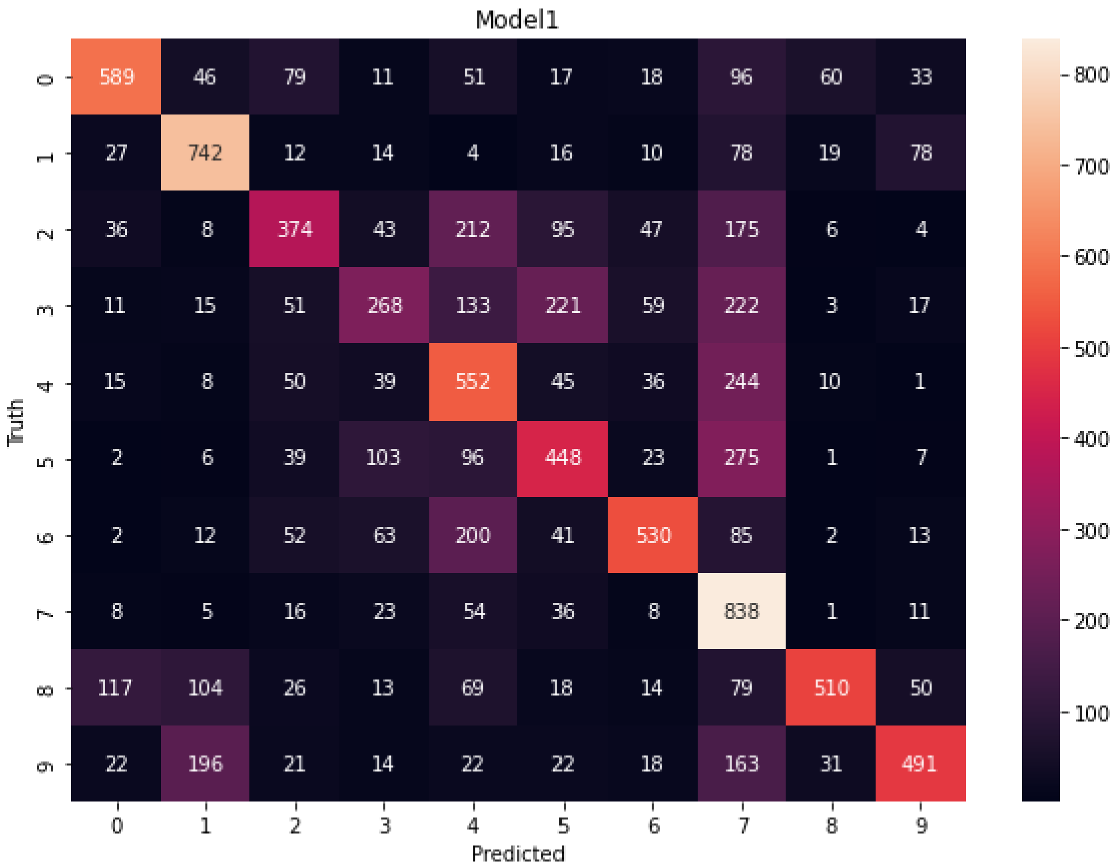
Graph For accuracy

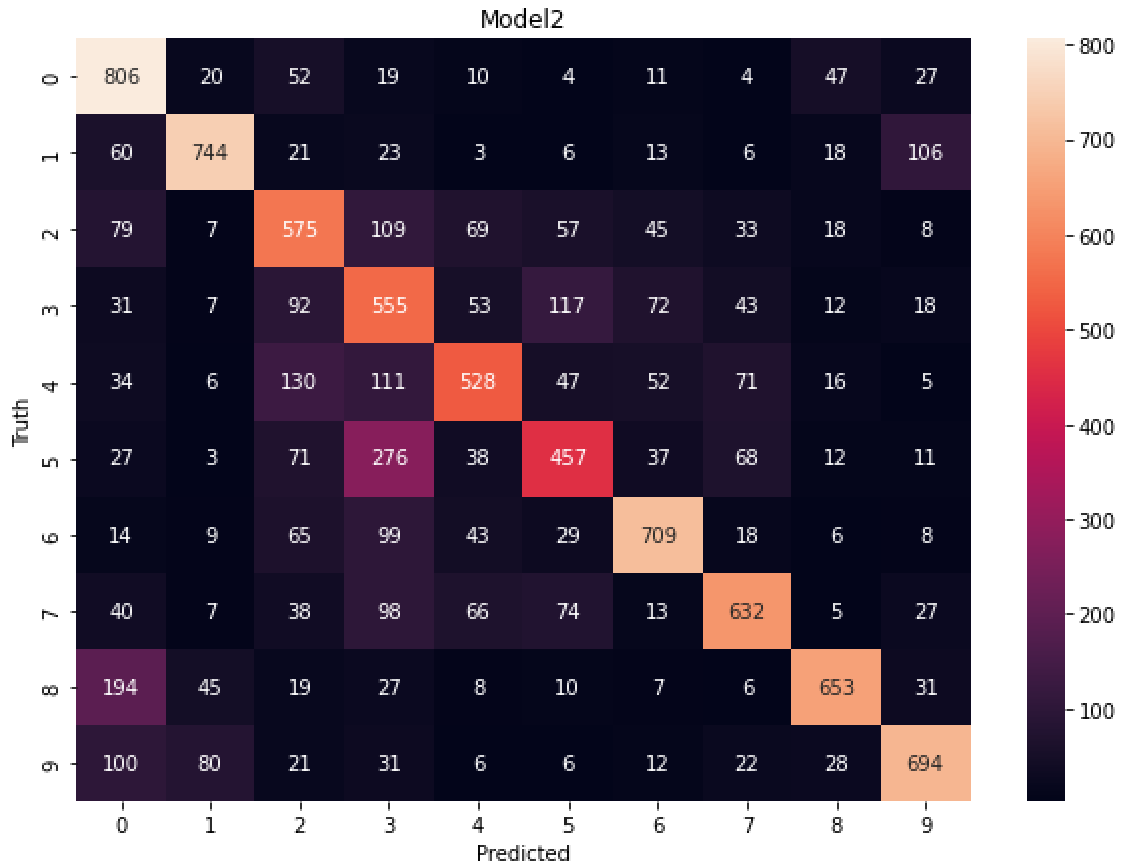


> Model 0 seems to be appropriate one

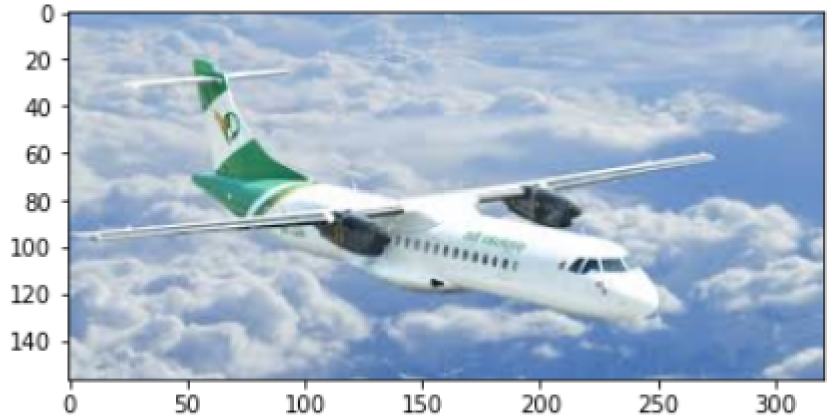
Confusion Matrix


Model0
679 28 58 14 23 9 24 25 90 50
18 794 2 7 3 4 18 5 37 112
49 4 482 64 121 81 110 63 14 12
10 11 46 484 74 153 110 63 23 26
18 2 41 53 624 42 105 99 11 5
9 1 32 138 53 581 55 110 11 10
1 6 27 45 25 19 849 15 5 8
9 6 21 37 38 56 11 800 1 21
51 34 18 17 11 3 14 12 799 41
25 57 9 10 4 9 8 20 39 819
Truth
0 1 2 3 4 5 6 7 8 9
Predicted
800
700
600
500
400
300
200
100

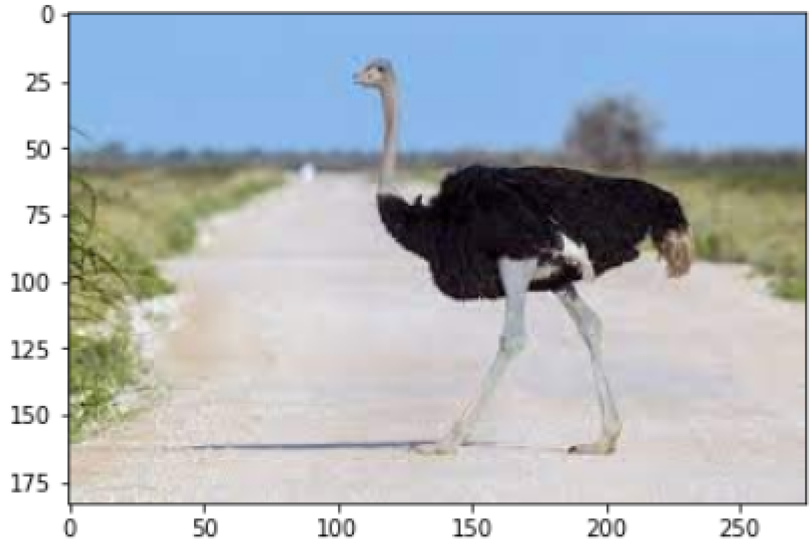

Model1
589 46 79 11 51 17 18 96 60 33
27 742 12 14 4 16 10 78 19 78
36 8 374 43 212 95 47 175 6 4
11 15 51 268 133 221 59 222 3 17
15 8 50 39 552 45 36 244 10 1
2 6 39 103 96 448 23 275 1 7
2 12 52 63 200 41 530 85 2 13
8 5 16 23 54 36 8 838 1 11
117 104 26 13 69 18 14 79 510 50
22 196 21 14 22 22 18 163 31 491
Truth
0 1 2 3 4 5 6 7 8 9
Predicted
800
700
600
500
400
300
200
100



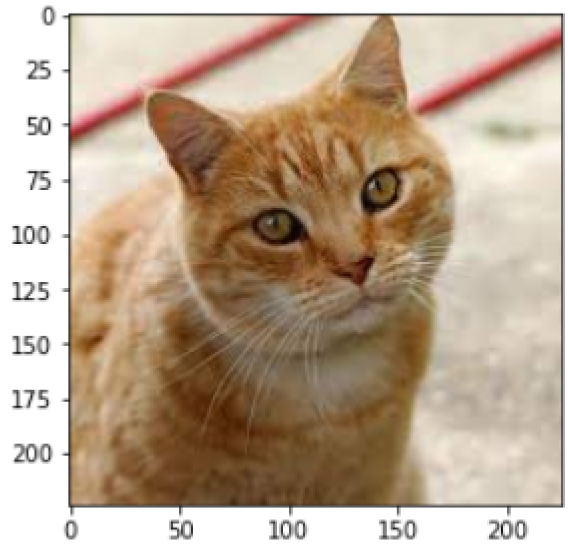
My favourite part is testing with new images typically



```
1/1 [==============================] - 0s 24ms/step
airplane
```



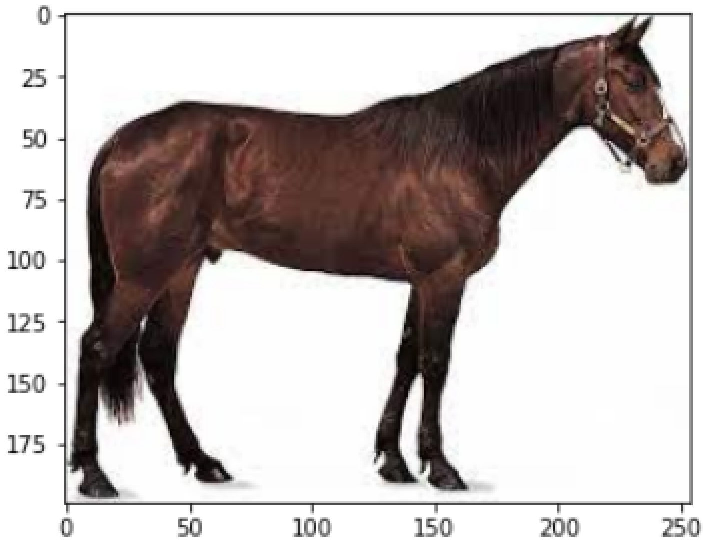
```
1/1 [==============================] - 0s 16ms/step
bird
```



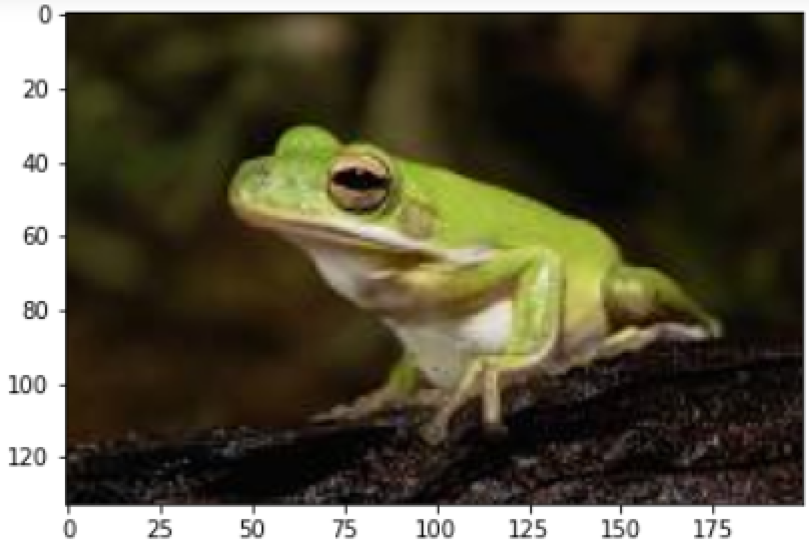
```
1/1 [==============================] - 0s 16ms/step
cat
```


```
1/1 [==============================] - 0s 16ms/step
airplane
```


```
1/1 [==============================] - 0s 16ms/step
truck
```



```
1/1 [==============================] - 0s 16ms/step
horse
```




```
1/1 [==============================] - 0s 16ms/step
frog
```


```
1/1 [==============================] - 0s 27ms/step
horse
```

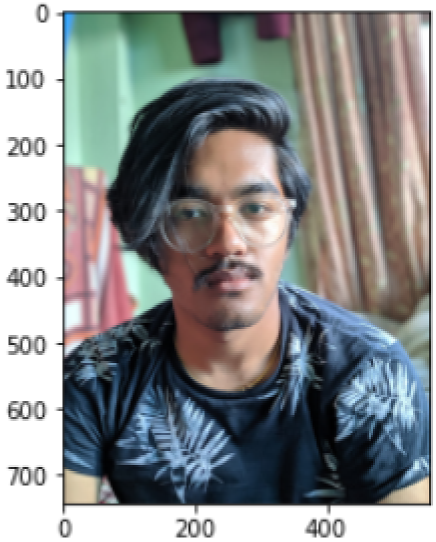
The model does have error

Fun Thing I did was even though there is not class for a human I wanted to know how it will classify my image

🥁🥁🥁

In [9]:

```
image_test('images/10.jpg')
```




```
1/1 [==============================] - 0s 16ms/step
truck
```

TURNS OUT I'M A TRUCK🚚🚚