A PROJECT REPORT ON

Image labeling system using Active Learning and Transfer learning techniques

A project report submitted in fulfillment for the Diploma Degree in AI & ML Under

Applied Roots with University of Hyderabad

Project submitted by

Ankan Mazumdar

Under the Guidance of

Mentor: Neil Fowler

Approved by: Mentor: Neil Fowler

University of Hyderabad

Declaration of Authorship

We hereby declare that this thesis titled ”Image labeling system using Active Learning and Transfer learning techniques” and the work presented by the undersigned candidate, as part of Diploma Degree in AI & ML.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name: Ankan Mazumdar

Thesis Title: Image labeling system using Active Learning and Transfer learning techniques

CERTIFICATE OF RECOMMENDATION

We hereby recommend that the thesis entitled “Image labeling system using Active Learning and Transfer learning techniques” prepared under my supervision be accepted in fulfillment of the requirement for awarding the degree of Diploma in AI & ML Under applied roots with University of Hyderabad. The project, in our opinion, is worthy of its acceptance.

Mentor: Neil Fowler

Under Applied roots with

University of Hyderabad

ACKNOWLEDGEMENT

Every project big or small is successful largely due to the effort of a number of wonderful people who have always given their valuable advice or lent a helping hand. I sincerely appreciate the inspiration; support and guidance of all those people who have been instrumental in making this project a success. I, Ankan Mazumdar student of applied roots, is extremely grateful to my mentor for the confidence bestowed in me and entrusting my project entitled “Image labeling system using Active Learning and Transfer learning techniques” with special reference.

At this juncture, I express my sincere thanks to Mentor: Neil Fowler of applied roots for making the resources available at the right time and providing valuable insights leading to the successful completion of our project who even assisted me in completing the project.

Name: Ankan Mazumdar

Contents

1. Abstract:

Data labeling can be a very time- and/or money-consuming procedure. For this, a domain specialist is occasionally required. Active Learning is an approach which uses fewer training data to achieve better optimization by iteratively training a model. We can use a classification model to perform the majority of the labeling using active learning, requiring us to label samples only when absolutely necessary. Active learning seeks to address this issue by asking to annotate only the most informative data from the unlabeled set.If given the freedom to choose which data to label, the active learning methodology has the potential to significantly reduce the

amount of labeled data needed to train a model with improved accuracy. The data that the model is most unsure of is prioritized through active learning, and labels are only requested for those. As a result, the model picks up new information more quickly.

This is a very important problem ,the reason being, in real world situations, we might want to go ahead to obtain manual labels for new data, but we have constraints on how much labeled data we can actually obtain within a certain time & limited budget.

2. Literature Review

Managing and streamlining a data annotation process is difficult. Businesses face a number of internal and external challenges that make the work of annotating documents inefficient and ineffective. Therefore, the only way to overcome these difficulties is to get to the root of them, comprehend them, and then resolve them appropriately. Let's begin.

Having trouble managing a large workforce-

Data-hungry ML and AI models need a huge amount of labeled data to learn. Businesses employ a sizable staff to manually tag datasets in order to provide the vast amount of labeled data needed to feed the computer models. Additionally, processing Big Data and labeling them with the best quality is crucial to achieving a high level of accuracy.

a lack of access to innovative tools and technologies-

Having a large and skilled workforce does not guarantee the production of high-quality tagged datasets. To carry out the accurate data annotation process, the right equipment and technology are needed. Different technologies and methods are used to tag datasets for deep learning depending on the type of data. It is therefore crucial to deploy the right technology that guarantees the greatest quality at a reasonable price.But businesses frequently fall short in creating the infrastructure necessary for the greatest possible data annotation. Because of the high cost of the tools and a lack of professional process expertise, organizations struggle to choose the best technology to implement.

Lack of reliable and high-quality data tagging-

A precise data annotation model necessitates the highest caliber dataset tagging. There is no room for error at all. Small mistakes can have a major impact on the company's bottom line. The ML model will pick up on incorrect information from your data samples in the same way. As a result, AI won't be able to correctly forecast it or recognise it.Furthermore, obtaining consistent high-quality data is the true challenge; assuring it is not the primary criterion. It is essential for organizations to keep a steady flow of high-quality, labeled datasets for machine learning (ML) training and accurate AI prediction.

Not an inexpensive project-

The process of labeling data for annotation is time-consuming. As a result, businesses find it difficult to determine how much money they need to create an ML & AI training project. Businesses occasionally have to move backwards as a result of paying a large workforce a significant wage for a longer length of time and investing in expensive technologies. Additionally, setting up a sizable, ergonomic office space with all the required amenities is difficult for businesses to handle.

failure to abide by data security rules-

Because of a severe lack of process understanding, data annotation organizations struggle to adhere to international data security standards. As Big Data becomes more and more widespread, the standards governing data privacy compliance are becoming stricter.

When it comes to raw data, it contains incredibly private information such as reading texts, identifying faces, etc. Therefore, tagging misinformation or any little mistakes can have huge repercussions. Besides, the data leak is the most important factor to be addressed here. Hence, data labeling companies are sometimes failing to comply with these compliances with privacy and internal data security standards.

3. Data Description:

CIFAR10 open source dataset will be used. It has 60000 color images with 10 classes, with 6000 images per class. 50000 training images and 10000 test images.The Dataset would divide training images into 5% training set ,5% validation set, 70% unlabelled data & rest 20% data is already clubbed as test data & then we will evaluate accuracy whenever required.

The dataset size is 177 MB. Here are the classes in the dataset

Airplane,automobile,bird,cat,deer,dog,frog,horse,ship,truck

Data type is a dictionary and there are 4 labels- dict_keys([b'batch_label', b'labels', b'data', b'filenames']), among these labels' & b data’ are important features. B Data' consists of an array shape of images & b labels’ contains true labels.

It is high dimensional with 3072 dimensions.

Pandas,Numpy,sklearn etc libraries will be used to process the data.

Labeled dataset is chosen in order to pass labels to the data whenever is required in order to help the model to enhance performance & to estimate the accuracy of the predicted labels by the model.

Tools (Pandas, SQL, Spark etc) that you will use to process this data Here, we will be using Python along with the below-listed libraries

- ➢ Pandas,
- ➢ NumPy,
- ➢ matplotlib,
- ➢ Scikit-learn

4. Approach-Methodology

Theoretical Approach

1. Label a small portion of data.Lets say Labeled data as Dl & unlabeled data as Du where Du>> Dl means Du is significantly larger than Dl.

2. Lets train the initial model M0 with Dl & use M0 to predict class labels. 3. From the unlabeled dataset Du,we‘ll smartly pick /sample a small subset of data s1 which could be hardly 1% of Du & obtain labels for this s1. When we smartly sample data, it means our target is to enhance the performance of our model M0.

4. Now, we’ll take our existing model M0 which is trained on previously labeled data Dl and we’ll add this newly labeled data s1, and we can either retrain the whole model from scratch or can retrain model M0 to obtain M1.

5. Repeat step 3 & step 4 until we obtain all labels correctly.

6. The stopping criteria would be when there is no significant improvement in model M1’s performance compared to M0.

You would have to determine the type of scenario your would like to use (that is, Membership Query Synthesis, Stream-Based Selective Sampling or Pool-Based sampling)

Query strategies

Algorithms for determining which data points should be labeled can be organized into a number of different categories, based upon their purpose-

● Expected error reduction: label those points that would most reduce the model's generalization error.

● Uncertainty sampling: Predict class probabilities for all unlabeled data using the trained classifier.If a classification has a probability higher than a predetermined threshold, it is assumed to be accurate. Include these samples in the training set along with their expected classes.label those points for which the current model is least certain as to what the correct output should be.

When a Supervised Machine Learning model makes a prediction, it often gives a confidence in that prediction. If the model is uncertain (low confidence), then human feedback can help. Getting human feedback when a model is uncertain is a type of Active Learning known as Uncertainty Sampling.

Among types of Uncertainty Sampling,we can use

1. Least Confidence: difference between the most confident prediction and 100% confidence

OR

2. Entropy: difference between all predictions, as defined by information theory

Empirical/Experimental approach

We have not yet finalized on the approach of the experiment. Most probably, the sampling strategy we will use is a combination of uncertainty sampling and pool-based sampling.

And the model to be used is most likely to be an ensemble of CNNs along with Transfer learning at the top of everything. Transfer learning basically leverages the already built neural network architecture & initial dense layers are frozen as per our requirement.

The neural network in multi-class classification has the same number of output nodes as classes. Each output node produces a score for the class to which it belongs.

An activation layer is used to pass through scores from the previous layer. The score is converted into probability values by the activation layer. The data is finally classified to the class with the highest probability value.Activation function could be Softmax or sigmoid activation function in the final layer, but we are yet to finalize it.

Each class's final score should be independent from the others. Most likely, sigmoid should be used because, unlike softmax, which converts each score of the final node between 0 and 1 independently of the previous scores, Sigmoid converts each score between 0 and 1 probabilities considering other’s score.

For Productionisation, We would use separate systems for model training(compute cluster) & serving web application / web API.There would be data store (which is also storing feedback that users are giving , needs to be run periodically in regular intervals) and the model get retrained (either with entire combined dataset or only with recent data or only those incorrect labeled ones corrected by user as per the business needs)

Retrained model is pushed to Model registry/repository with its version.Its redeployed to the API.Use of pretrained weights from previous layer/model to new layer for new dataset and so on, & updating the model as new data arrives.

The majority of ML algorithms demand that any input or output variables be numbers. Any categorical data must therefore be converted to integers.

With one-hot, we create a new category column for each categorical value and give it a binary value of 1 or 0.

A binary vector is used to represent each integer value. The index is denoted by a 1 and all values are zero.

Dataset is balanced ,hence doesn't require any sampling . Moreover it requires image augmentation which we will be performing.

Furthermore, we could explore more for the dataset or can manipulate the balance of our CIFAR-10 dataset by techniques like upsampling,downsampling.

Key Performance metric (KPI) -

In the real world, there are many different types of metrics. We'll examine at a few of the mostly useful used KPIs that we might utilize in this multi class classification project-

- Accuracy
- Confidence score

Interpretability- We will be using LIME.

- Model-Agnostic, Different Data Types (Tabular/Text)
- Easy to interpret for Non-ML users (even non-Techies)
- Relatively faster than SHAP in many case

5. Requirement Analysis

1.Functional Requirements

- Purpose

To build a system which will predict the image labels correctly, basically an automated image annotation system. The system will need retraining the model redundantly until a certain threshold accuracy is reached.

● Inputs

Images are provided as the input to the system.

● Output

The system interprets & predicts the image labels.

● Usability

It reduces manual efforts,saves time, higher accuracy,scalability,streamlined process, on time delivery, reduces cost.

2. Hardware Requirements

● GPU preferable google colab

3. Software Requirements

● Notepad++ or any Code Editor

● Anaconda Navigator , Jupyter Notebook

- Streamlit
- Python 3
- Python libraries like pandas, NumPy etc.
- scikit-learn
- Client environment may be Windows or Linux
- Web Browser

6. Preprocessing - EDA and Feature Extraction

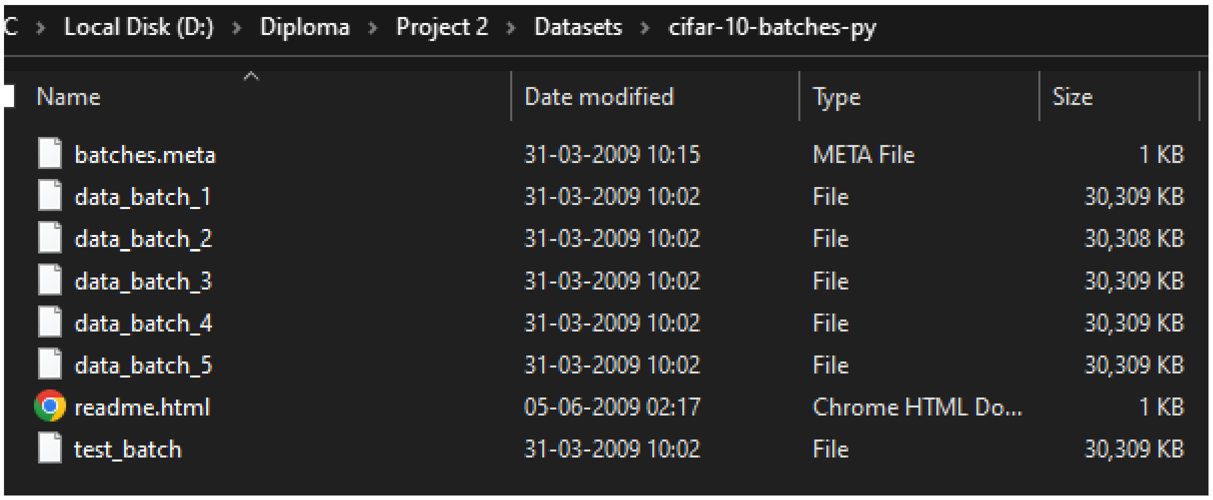
a. Dataset-

Link- https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz

In other way, the dataset can also be downloaded from keras.

CIFAR10 open source dataset will be used. It has 60000 color images with 10 classes, with 6000 images per class. 50000 training images and 10000 test images.

There are a total 7 files inside the dataset- 5 training, 1 test & one metadata file.

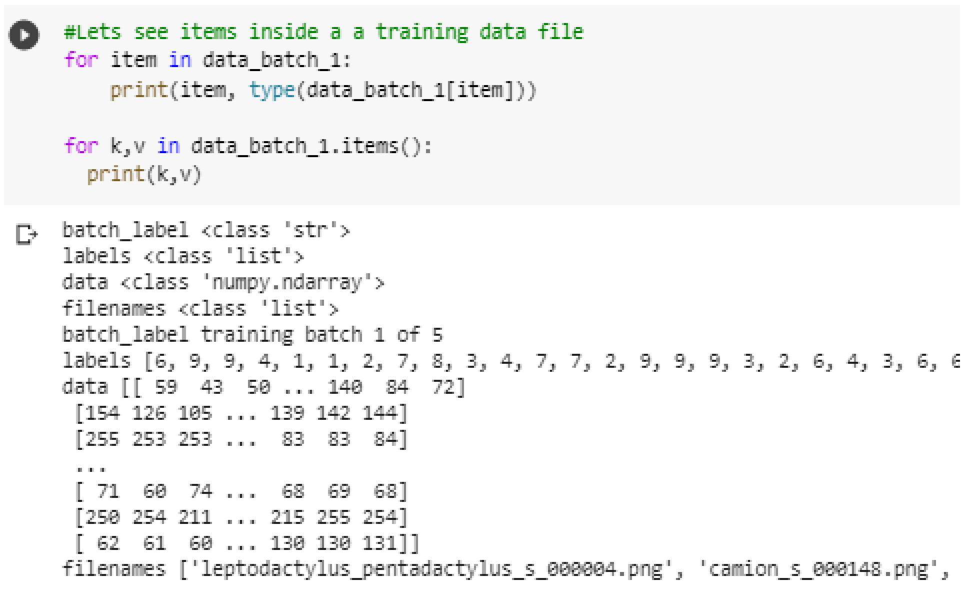


C › Local Disk (D:) › Diploma › Project 2 › Datasets › cifar-10-batches-py

Name	Date modified	Type	Size
batches.meta	31-03-2009 10:15	META File	1 KB
data_batch_1	31-03-2009 10:02	File	30,309 KB
data_batch_2	31-03-2009 10:02	File	30,308 KB
data_batch_3	31-03-2009 10:02	File	30,309 KB
data_batch_4	31-03-2009 10:02	File	30,309 KB
data_batch_5	31-03-2009 10:02	File	30,309 KB
readme.html	05-06-2009 02:17	Chrome HTML Do...	1 KB
test_batch	31-03-2009 10:02	File	30,309 KB

The dataset size is 177 MB. Here are the classes in the dataset -

[Airplane,automobile,bird,cat,deer,dog,frog,horse,ship,truck]



```
#Lets see items inside a a training data file
for item in data_batch_1:
    print(item, type(data_batch_1[item]))

for k,v in data_batch_1.items():
  print(k,v)
```

```
batch_label <class 'str'>
labels <class 'list'>
data <class 'numpy.ndarray'>
filenames <class 'list'>
batch_label training batch 1 of 5
labels [6, 9, 9, 4, 1, 1, 2, 7, 8, 3, 4, 7, 7, 2, 9, 9, 9, 3, 2, 6, 4, 3, 6, 6
data [[ 59  43  50 ... 140  84  72]
 [154 126 105 ... 139 142 144]
 [255 253 253 ...  83  83  84]
 ...
 [ 71  60  74 ...  68  69  68]
 [250 254 211 ... 215 255 254]
 [ 62  61  60 ... 130 130 131]]
filenames ['leptodactylus_pentadactylus_s_000004.png', 'camion_s_000148.png',
```

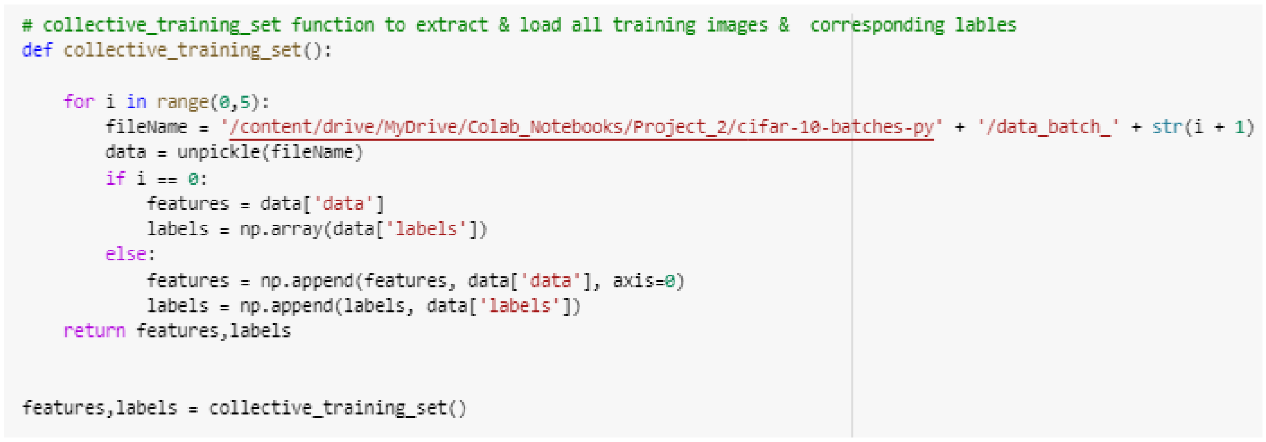
Data type is a dictionary and there are 4 labels- dict_keys([b'batch_label', b'labels', b'data', b'filenames']), among these labels' & b data' are important features. B Data' consists of an array shape of images & b labels' contains true labels.

It is high dimensional with 3072 dimensions.

Pandas,Numpy,sklearn etc libraries will be used to process the data.

Labeled dataset is chosen in order to pass labels to the data whenever is required in order to help the model to enhance performance & to estimate the accuracy of the predicted labels by the model.

b. Loading the whole dataset-



```
# collective_training_set function to extract & load all training images &  corresponding lables
def collective_training_set():

    for i in range(0,5):
        fileName = '/content/drive/MyDrive/Colab_Notebooks/Project_2/cifar-10-batches-py' + '/data_batch_' + str(i + 1)
        data = unpickle(fileName)
        if i == 0:
            features = data['data']
            labels = np.array(data['labels'])
        else:
            features = np.append(features, data['data'], axis=0)
            labels = np.append(labels, data['labels'])
    return features,labels


features,labels = collective_training_set()
```

c. Understanding how reshape & transpose works on single image-

The original batch data is a numpy array with the dimensions (10000 x 3072) of a dimensional tensor, where the column count (10000) represents the number of sample data. The row vector

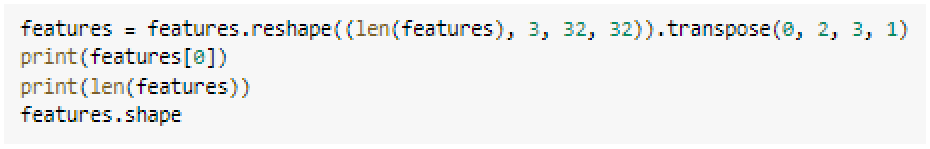
(3072), as indicated in the CIFAR-10/CIFAR-100 dataset, depicts a color image with a 32x32 pixel size. The row vector (3072), which will be used in this project to perform the classification tasks, is an inappropriate type of image data to feed. The dimensions of the tensor representing an image data should be either (num channel x width x height) or (num channel x width x height) in order to input the data into an NN model.

Our image data is a numpy ndarray.

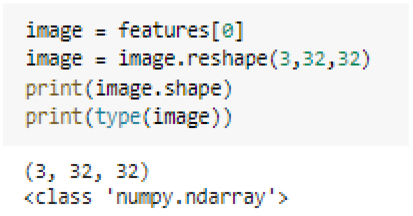
Each image is a 1-D array having 3,072 entries. First 1024 entries for Red, then next 1024 are Green and lastly 1024 entries are Blue channels. In order to visualize the images we have to change the shape of the image as (32,32,3).

Reshape And Transpose a Single Image Our image is a single dimension array of size 3072. First 1024 entries of the array are of Red channel, next 1024 entries are of Green channel, and last 1024 entries are of Blue channel. Total 3072 entries are of three RGB channels.

First we reshape the image/ array into (3,32,32) using image.reshape(3,32,32). (32,32) for 32x32 =1024 entries and 3 for RGB channels.

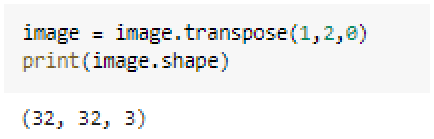


```
features = features.reshape((len(features), 3, 32, 32)).transpose(0, 2, 3, 1)
print(features[0])
print(len(features))
features.shape
```



```
image = features[0]
image = image.reshape(3,32,32)
print(image.shape)
print(type(image))
```

```
(3, 32, 32)
<class 'numpy.ndarray'>
```

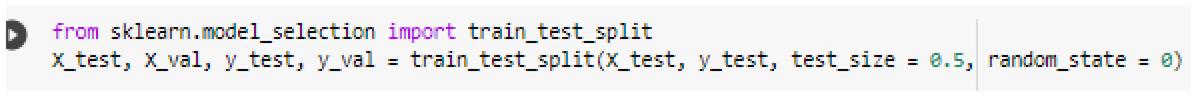


```
image = image.transpose(1,2,0)
print(image.shape)
```

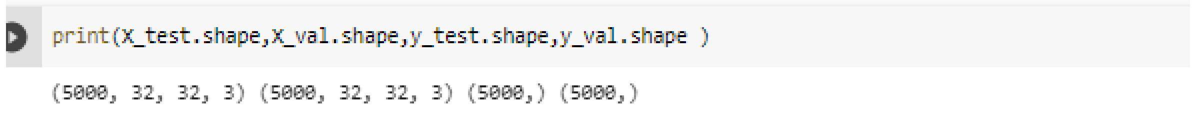
```
(32, 32, 3)
```

d. Creating Validation dataset

Lets split test dataset into 2 halves and use one half 5000 images as validation set



```
from sklearn.model_selection import train_test_split
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size = 0.5, random_state = 0)
```



```
print(X_test.shape,X_val.shape,y_test.shape,y_val.shape )
```

```
(5000, 32, 32, 3) (5000, 32, 32, 3) (5000,) (5000,)
```

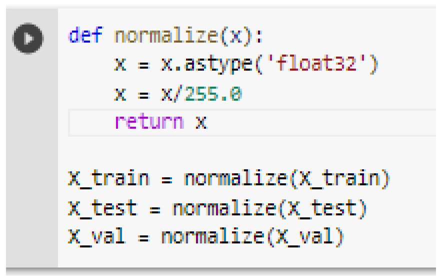
5. Normalize/Rescaling the dataset

The normalize function converts the input image data, x, into a normalized Numpy array. Without altering the array's shape, the original data's values will be changed to fall between 0 and 1, inclusive. An easy explanation for why normalization should be carried out has something to do with activation function.

As an illustration, the sigmoid activation function converts an input value to a new value between 0 and 1. The output value quickly reaches the maximum value 1 when the input value is reasonably large. Similar to this, when the input value is a little low, the output value easily reaches the maximum value of 0.

Pixel value of the image falls between 0 to 255.

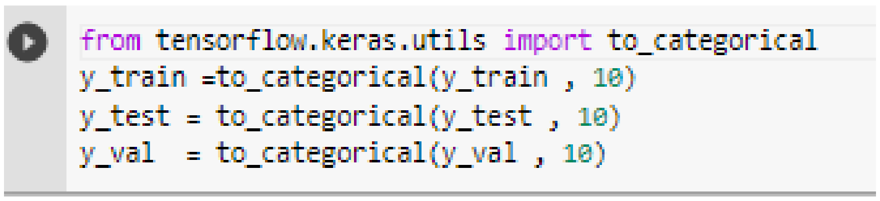
So, we are scaling the value between 0 to 1 by dividing each value by 255



```
def normalize(x):
    x = x.astype('float32')
    x = x/255.0
    return x

X_train = normalize(X_train)
X_test = normalize(X_test)
X_val = normalize(X_val)
```

e. One Hot Encoding-

We would do one-hot encoding using tensorflow’s to_categorical module-



```
from tensorflow.keras.utils import to_categorical
y_train =to_categorical(y_train , 10)
y_test = to_categorical(y_test , 10)
y_val  = to_categorical(y_val , 10)
```

The majority of ML algorithms demand that any input or output variables be numbers. Any categorical data must therefore be converted to integers.

With one-hot, we create a new category column for each categorical value and give it a binary value of 1 or 0.

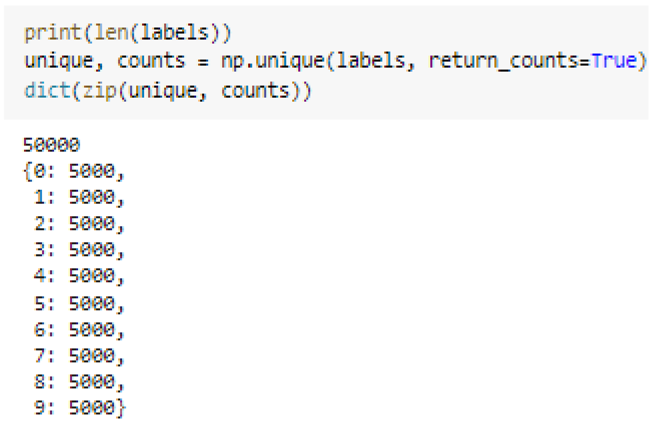
A binary vector is used to represent each integer value. The index is denoted by a 1 and all values are zero.Given that the output of our model would provide the probability of the various categories in which a picture should be classified as a prediction. A vector with the same number of elements as the total number of image classes should exist. For instance, CIFAR-10 offers 10 different image classes, so we also require a vector with a 10 size. Each component represents the class-specific prediction probability.

A comparison between the forecast and the ground truth label should also be possible using our model. It implies that the label data's shape should likewise be converted into a vector with a 10x10 size. Instead, we assign the value 1 to the relevant element because the label is the actual reality.

Even while this is useful in some ordinal circumstances, some input data lacks ranking for category values, which can cause problems with predictions and poor performance. One hot encoding comes to the rescue at that point.

Our training data is more useful and expressive thanks to one hot encoding, and it is also simple to scale. We can more quickly calculate a probability for our values by using numerical values. For our output values in particular, one hot encoding is chosen since it offers more precise predictions than single labels.

This approach can result in a significant issue (too many predictors) if the original column contains a lot of unique values because it creates a lot of additional variables. One-hot encoding also has the drawback of increasing multicollinearity among the numerous variables, which reduces the model's accuracy.



```
print(len(labels))
unique, counts = np.unique(labels, return_counts=True)
dict(zip(unique, counts))
```

```
50000
{0: 5000,
 1: 5000,
 2: 5000,
 3: 5000,
 4: 5000,
 5: 5000,
 6: 5000,
 7: 5000,
 8: 5000,
 9: 5000}
```

Hence either , we could explore more for the dataset or can manipulate the balance of our CIFAR-10 dataset by techniques like upsampling,downsampling.

Secondly, we already have labels in order to assess our model’s accuracy, which is not the case with the real world projects,

f .ImageDataGenerator

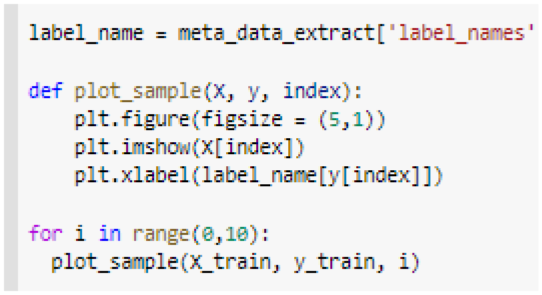
ImageDataGenerator will be used to apply multiple data augmentation techniques to avoid overfitting in the training part.

from tensorflow.keras.preprocessing.image import ImageDataGenerator datagen = ImageDataGenerator(rotation_range=15, width_shift_range=0.1, height_shift_range=0.1,horizontal_flip=True,)

datagen.fit(X_train)

9. Visualization

After preprocessing & transforming dataset, Lets visualize images using matplotlib-



```
label_name = meta_data_extract['label_names'

def plot_sample(X, y, index):
    plt.figure(figsize = (5,1))
    plt.imshow(X[index])
    plt.xlabel(label_name[y[index]])

for i in range(0,10):
  plot_sample(X_train, y_train, i)
```

7. Model Building and training

There are 2 ways to program CNN with keras:

Sequential approach: Here, we generally add layers in sequence.

Modular approach: This is more important. This is more dynamic, customized, molded and easy to explore.

Effectiveness in both approaches will remain the same.

We would be choosing CNN keras sequential model as baseline model which is a deep learning model

Steps of Deep learning Analysis-

Select Dataset

Pre-process Data

Model Generation

Model Compilation

Model Training

Model Evaluation

Model Prediction

a. Build Baseline model-

Here building custom baseline model consisting of 5 blocks-

3 Conv2D layers(Feature Detection / extraction Block),

1 Flatten (Transition Block (from feature detection to classification)) ,

4 Dense layers (Classification Block)

Below parameters are added as different layers in the model in order to tune the model to avoid overfitting and tune the model-

Batch normalization

Dropout

Flatten

```
# Modelling - Model on CNN
import tensorflow as tf
import os
from keras.utils.vis_utils import plot_model
from tensorflow.keras import models, layers
from keras.layers import Conv2D, MaxPooling2D, Flatten , Dense,
Activation,Dropout,BatchNormalization

# create a sequential model i.e. empty neural network which has no layers in it.
model=tf.keras.Sequential([

#==================== Feature Detection / extraction Block ====================#
# Add first convolutional block  we use Conv2D and for colour images and shape
use Conv3D
#Here filters = 64, kernel_size= 3*3 , activation = ‘relu’ and L2 regularizer
used to avoid overfitting
Conv2D(64,(3,3),input_shape=(32,32,3),activation='relu',padding='same',kernel_reg
ularizer=tf.keras.regularizers.l2(0.01)),
BatchNormalization(),
Conv2D(64,(3,3),input_shape=(32,32,3),activation='relu',padding='same',kernel_reg
ularizer=tf.keras.regularizers.l2(0.01)),
BatchNormalization(),
MaxPooling2D(pool_size=(2,2)),
Dropout(0.25),
Dropout(0.25),

# Add Second convolutional block
Conv2D(128,(3,3),activation='relu',padding='same',kernel_regularizer=tf.keras.reg
ularizers.l2(0.01)),
BatchNormalization(),
Conv2D(128,(3,3),activation='relu',padding='same',kernel_regularizer=tf.keras.reg
ularizers.l2(0.01)),
BatchNormalization(),
MaxPooling2D(pool_size=(2,2)),
Dropout(0.25),
Dropout(0.25),

# Add Third convolutional block
Conv2D(256,(3,3),activation='relu',padding='same',kernel_regularizer=tf.keras.reg
ularizers.l2(0.01)),
BatchNormalization(),
MaxPooling2D(pool_size=(2,2)),
Dropout(0.25),
Dropout(0.25),

#==================== Transition Block (from feature detection to classification)
====================#
# Add Flatten layer. Flatten simply converts matrics to array
Flatten(input_shape=(32,32)),

#==================== Classification Block ====================#

# Classification segment - fully connected network
# The Dense layer does classification and is a deep neural network. Dense layer
always accept the array.
Dense(128, activation='relu'),
Dense(100, activation='relu'),
Dense(80, activation='relu'),

# Add the output layer
model.add(layers.Dense(10, activation='softmax')) # as Output layer in above
image. The output layer normally have softmax activation

])
```

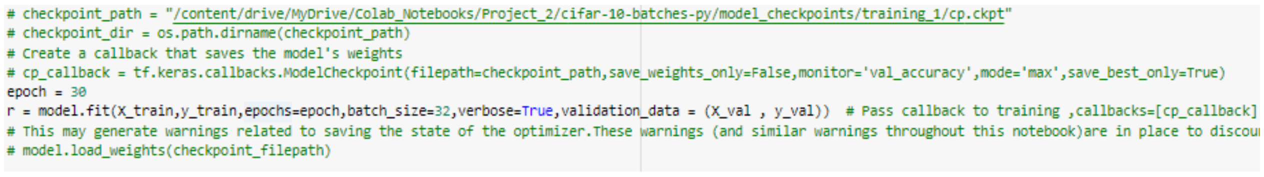
b. Compile model-

A model's behavior is "frozen" when compile() is called on it. The values of the trainable attributes at the time the model is compiled should, according to this, be maintained throughout the model's existence until compile is called again.

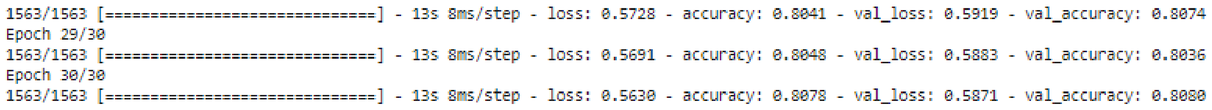
```
#Compile defines the loss function, the optimizer and the metrics.
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

#The summary is textual and includes information about:
#The layers and their order in the model. The output shape of each layer. The
number of parameters (weights) in each layer. The total number of parameters
(weights) in the model.
model.summary()

Parameters summary-
Total params: 1,104,094
Trainable params: 1,102,814
Non-trainable params: 1,280
```

____

c. Training model-



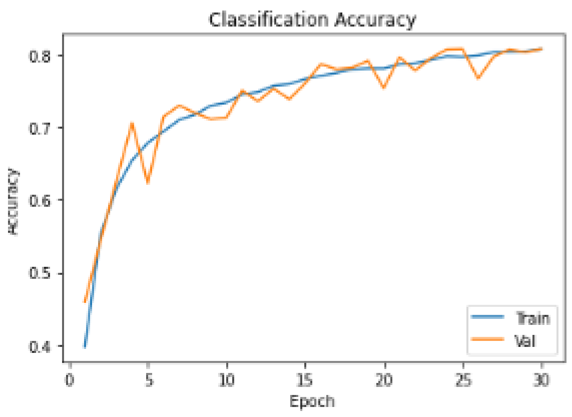
```
# checkpoint_path = "/content/drive/MyDrive/Colab_Notebooks/Project_2/cifar-10-batches-py/model_checkpoints/training_1/cp.ckpt"
# checkpoint_dir = os.path.dirname(checkpoint_path)
# Create a callback that saves the model's weights
# cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,save_weights_only=False,monitor='val_accuracy',mode='max',save_best_only=True)
epoch = 30
r = model.fit(X_train,y_train,epochs=epoch,batch_size=32,verbose=True,validation_data = (X_val , y_val))  # Pass callback to training ,callbacks=[cp_callback]
# This may generate warnings related to saving the state of the optimizer.These warnings (and similar warnings throughout this notebook)are in place to discou
# model.load_weights(checkpoint_filepath)
```

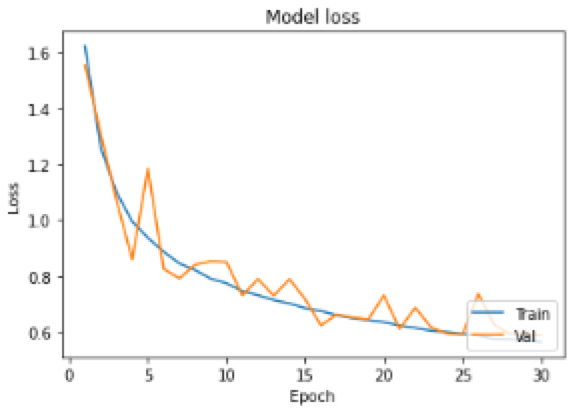


```
1563/1563 [==============================] - 13s 8ms/step - loss: 0.5728 - accuracy: 0.8041 - val_loss: 0.5919 - val_accuracy: 0.8074
Epoch 29/30
1563/1563 [==============================] - 13s 8ms/step - loss: 0.5691 - accuracy: 0.8048 - val_loss: 0.5883 - val_accuracy: 0.8036
Epoch 30/30
1563/1563 [==============================] - 13s 8ms/step - loss: 0.5630 - accuracy: 0.8078 - val_loss: 0.5871 - val_accuracy: 0.8080
```

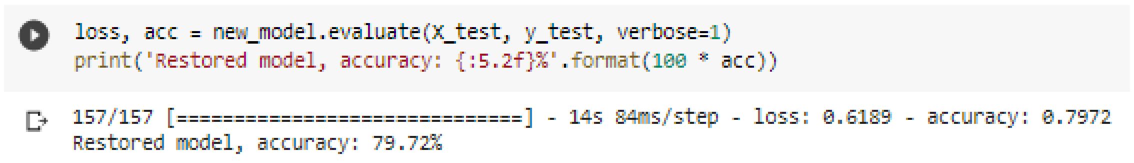
d. Evaluating model on Test data-

Accuracy is approx 80% on test dataset , also there is no gap in graph between accuracy & loss of training & validation datasets.





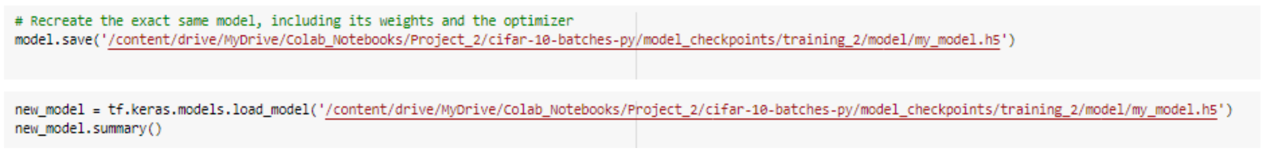
Check its accuracy:



```
loss, acc = new_model.evaluate(X_test, y_test, verbose=1)
print('Restored model, accuracy: {:5.2f}%'.format(100 * acc))
```

```
157/157 [==============================] - 14s 84ms/step - loss: 0.6189 - accuracy: 0.7972
Restored model, accuracy: 79.72%
```

e. Save & Load model

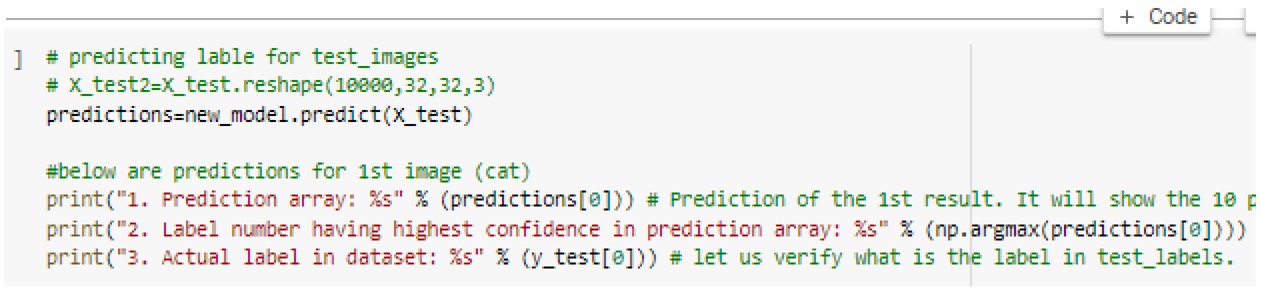


```
# Recreate the exact same model, including its weights and the optimizer
model.save('/content/drive/MyDrive/Colab_Notebooks/Project_2/cifar-10-batches-py/model_checkpoints/training_2/model/my_model.h5')
```

```
new_model = tf.keras.models.load_model('/content/drive/MyDrive/Colab_Notebooks/Project_2/cifar-10-batches-py/model_checkpoints/training_2/model/my_model.h5')
new_model.summary()
```

f. Testing the model & make prediction on test data

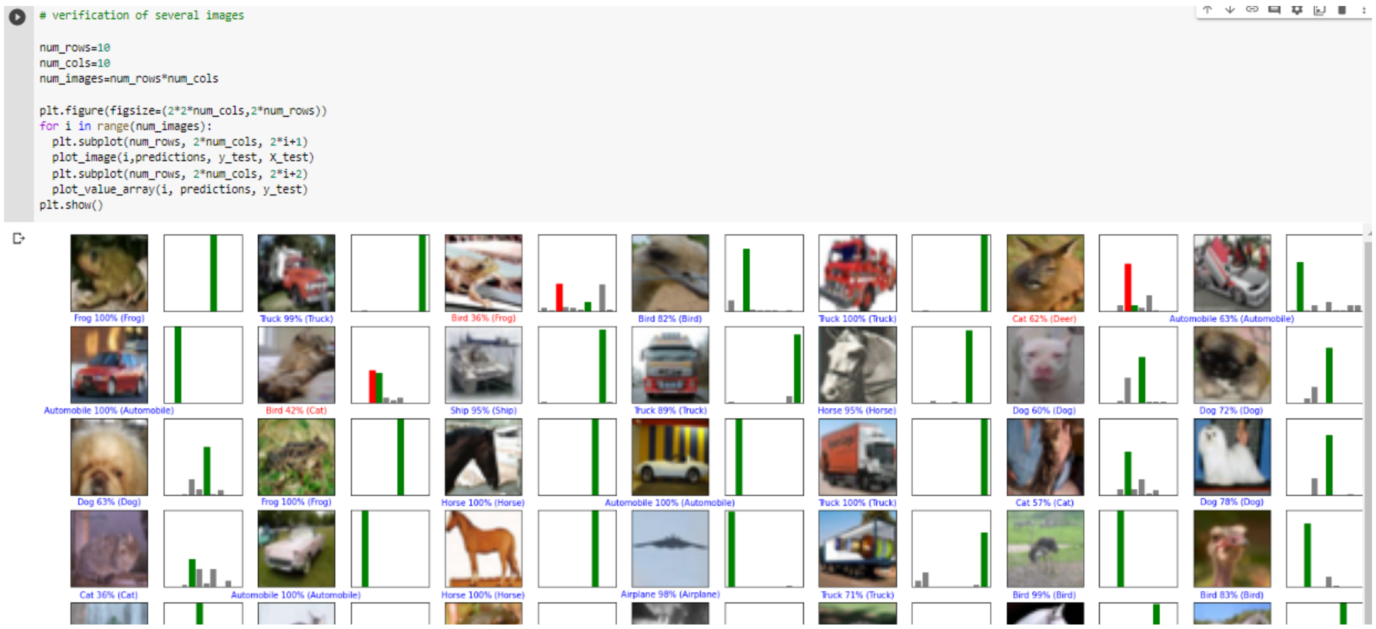
model.predit() function returns confidence score array for an image for all 10 labels, y_pedicted will be the max value in predictions array.



```
# predicting lable for test_images
# X_test2=X_test.reshape(10000,32,32,3)
predictions=new_model.predict(X_test)

#below are predictions for 1st image (cat)
print("1. Prediction array: %s" % (predictions[0])) # Prediction of the 1st result. It will show the 10 p
print("2. Label number having highest confidence in prediction array: %s" % (np.argmax(predictions[0])))
print("3. Actual label in dataset: %s" % (y_test[0])) # let us verify what is the label in test_labels.
```

g. Visualizing Test data prediction

Plot represents the image, predicted label ,(confidence score probability in percentage), ground truth and histogram of prediction array



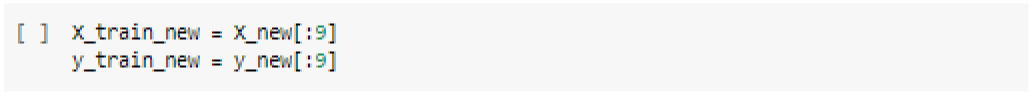
```
# verification of several images

num_rows=10
num_cols=10
num_images=num_rows*num_cols

plt.figure(figsize=(2*2*num_cols,2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i,predictions, y_test, X_test)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions, y_test)
plt.show()
```

h. Model retraining -

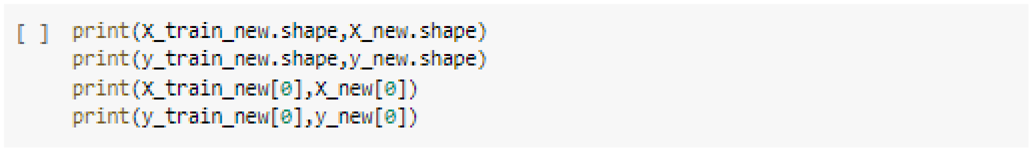
Retrain the model to have them fitted on wrong predicted images & save the retrained model.

We will keep the validation set (along with their ground truth values) same throughout the process & will check whether model’s performance on test dataset is improving or not ,

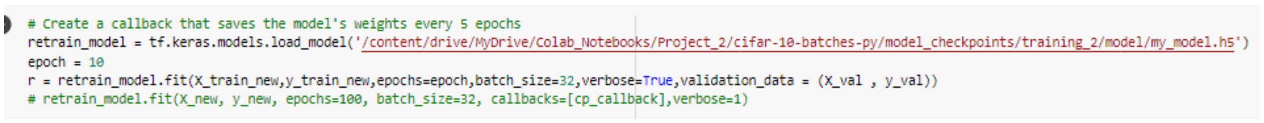
If yes, then we will deploy this new model , otherwise the existing model would be continued.



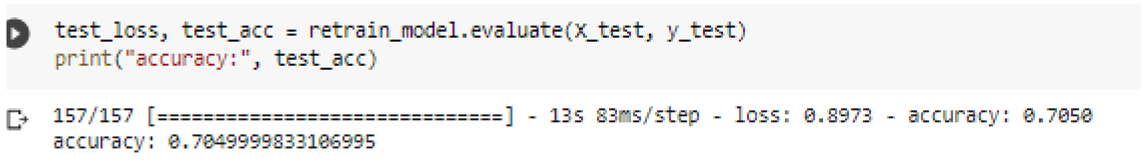
```
[ ]  X_train_new = X_new[:9]
     y_train_new = y_new[:9]
```



```
[ ]  print(X_train_new.shape,X_new.shape)
     print(y_train_new.shape,y_new.shape)
     print(X_train_new[0],X_new[0])
     print(y_train_new[0],y_new[0])
```



```
# Create a callback that saves the model's weights every 5 epochs
retrain_model = tf.keras.models.load_model('/content/drive/MyDrive/Colab_Notebooks/Project_2/cifar-10-batches-py/model_checkpoints/training_2/model/my_model.h5')
epoch = 10
r = retrain_model.fit(X_train_new,y_train_new,epochs=epoch,batch_size=32,verbose=True,validation_data = (X_val , y_val))
# retrain_model.fit(X_new, y_new, epochs=100, batch_size=32, callbacks=[cp_callback],verbose=1)
```



```
test_loss, test_acc = retrain_model.evaluate(X_test, y_test)
print("accuracy:", test_acc)
```

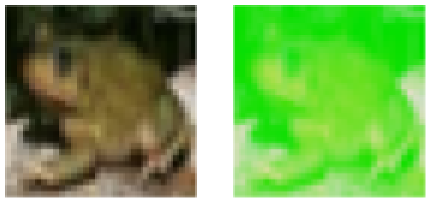
```
157/157 [==============================] - 13s 83ms/step - loss: 0.8973 - accuracy: 0.7050
accuracy: 0.7049999833106995
```

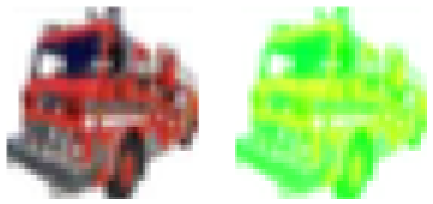
Observation-

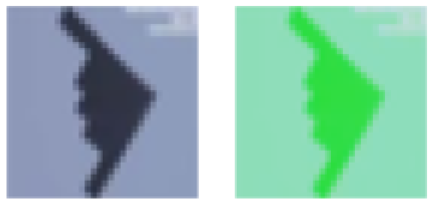
Model's accuracy goes down, hence it can't be considered.

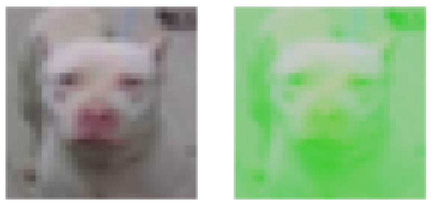
10. Interpretability - Explaining Model predictions

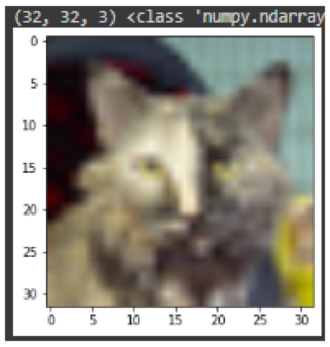
LIME explanation is used here.



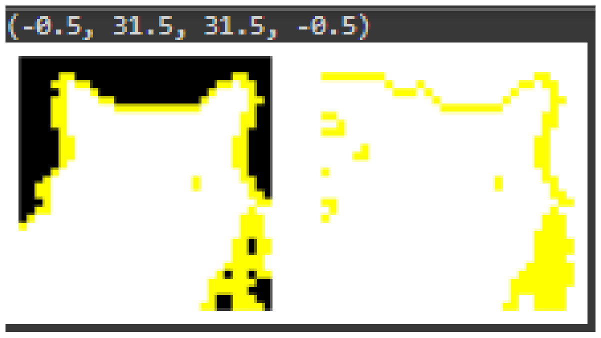








```
(32, 32, 3) <class 'numpy.ndarray
```



```
(-0.5, 31.5, 31.5, -0.5)
```

Observations-

1. For First 2 images, the greener & darker part are reflecting correctly ,however its expected to be more precise in pointing boundaries

2.For next 2 incorrect predictions, The explanations are highlighting correct areas in green to be considered in predictions , however ,the predictions are still wrong..

3.LIME applied on images other than CIFAR 10 dataset projects super pixels rather than depicting green/ red regions.

We would be trying out other approaches to retrain models & improve accuracy.

8. Model Improvement Strategy-

Now here comes the main part, in order to improve the model’s performance, the technique which we would be implementing is active learning & transfer learning which consist of few steps-

loading base model, add few more layers considering it will not harm the model,

retraining model to learn weights from new data only (which includes incorrect predictions).

In addition to that, Ensure to keep the initial training weights from training data the same.

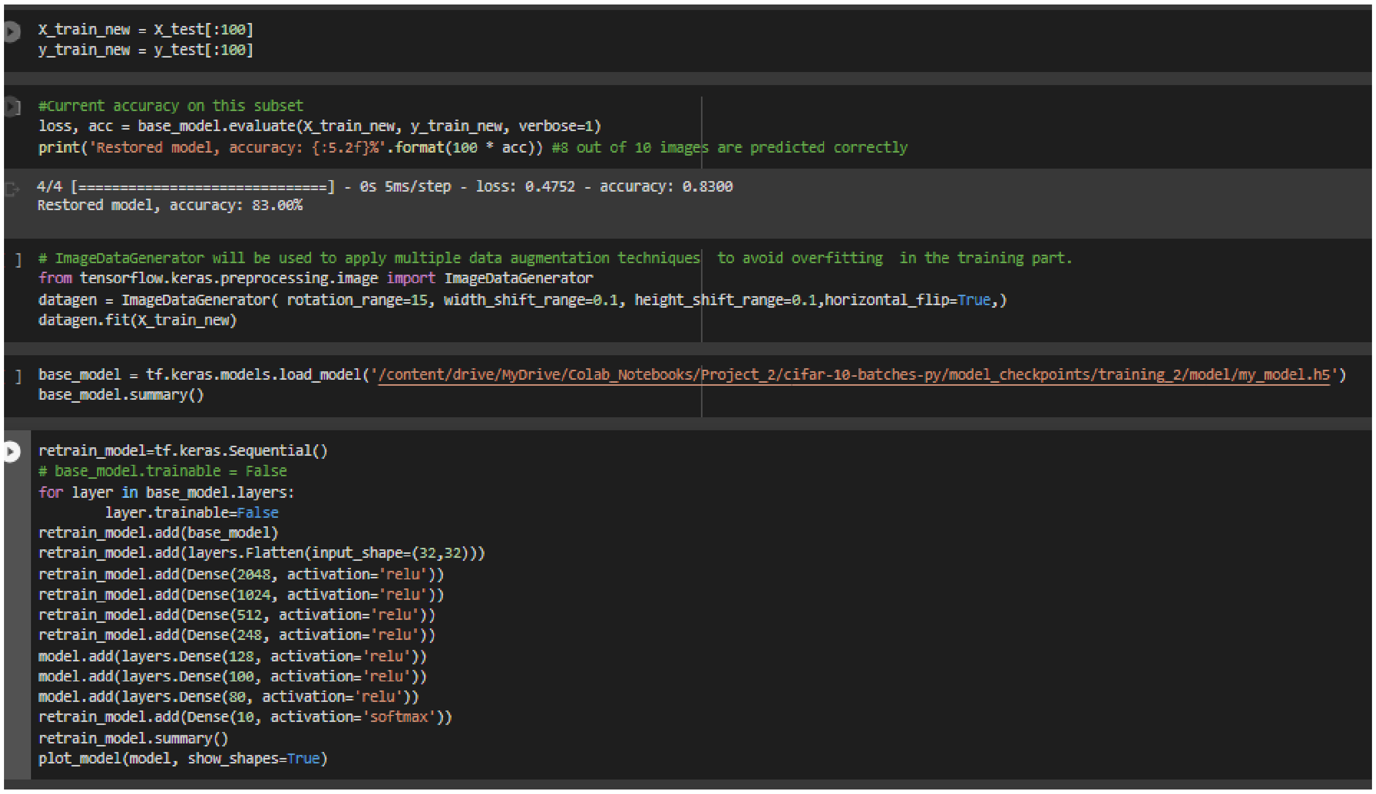
If the model performance gets improved , we will consider it as our new model to be deployed in production , otherwise the existing model will be retained.

a. Active Learning and Transfer Learning

The initial baseline model is restructured by adding a few more dense layers in it.

We will keep its initially trained weights untouched by setting

layer.trainable = False



```
X_train_new = X_test[:100]
y_train_new = y_test[:100]
```

```
#Current accuracy on this subset
loss, acc = base_model.evaluate(X_train_new, y_train_new, verbose=1)
print('Restored model, accuracy: {:5.2f}%'.format(100 * acc)) #8 out of 10 images are predicted correctly
```

```
4/4 [==============================] - 0s 5ms/step - loss: 0.4752 - accuracy: 0.8300
Restored model, accuracy: 83.00%
```

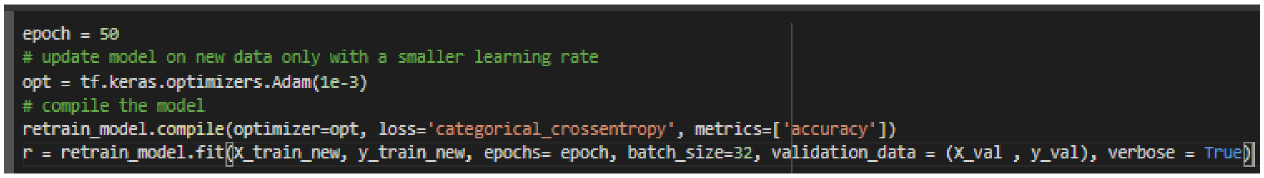
```
# ImageDataGenerator will be used to apply multiple data augmentation techniques  to avoid overfitting  in the training part.
from tensorflow.keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator( rotation_range=15, width_shift_range=0.1, height_shift_range=0.1,horizontal_flip=True,)
datagen.fit(X_train_new)
```

```
base_model = tf.keras.models.load_model('/content/drive/MyDrive/Colab_Notebooks/Project_2/cifar-10-batches-py/model_checkpoints/training_2/model/my_model.h5')
base_model.summary()
```

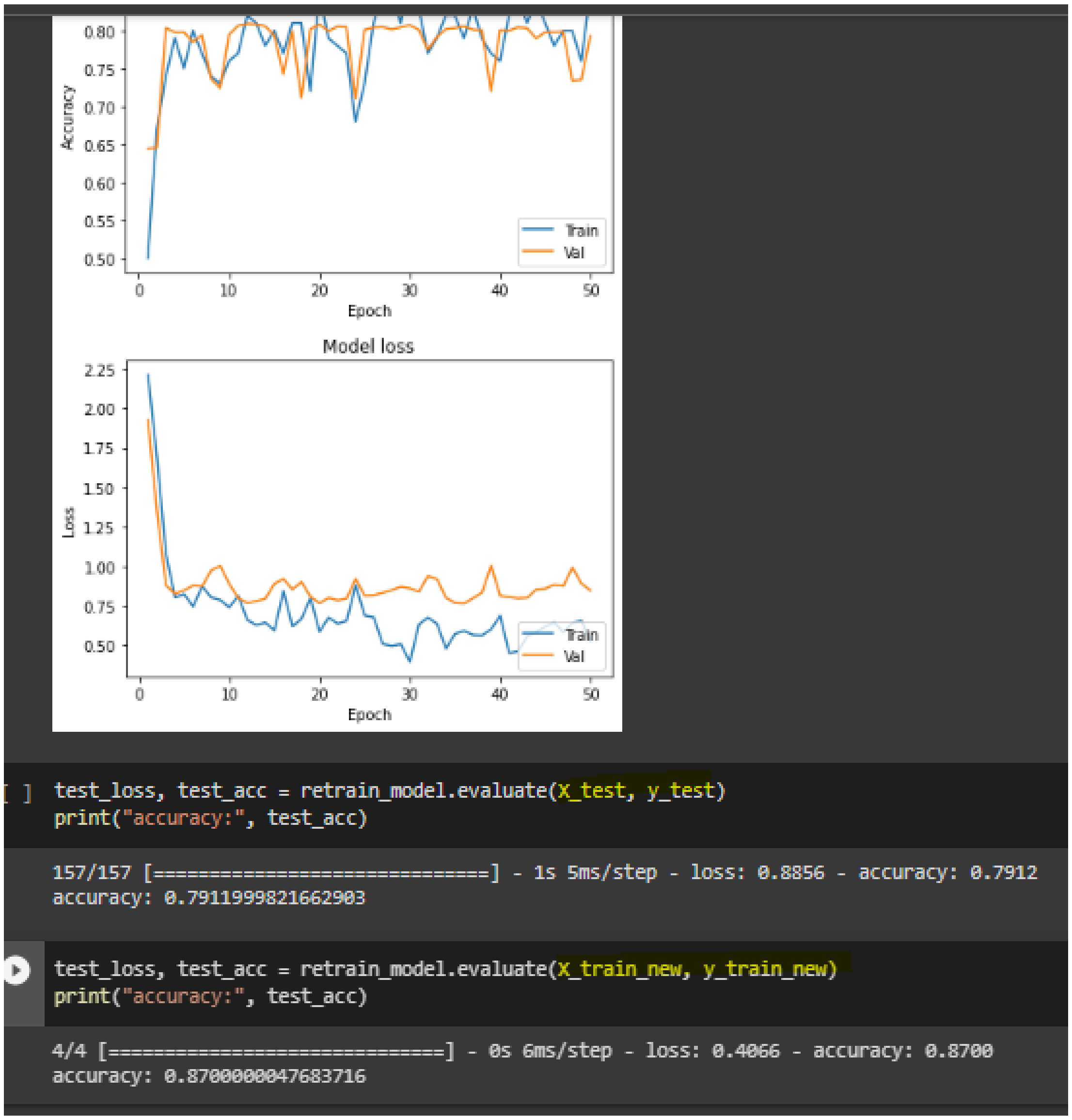
```
retrain_model=tf.keras.Sequential()
# base_model.trainable = False
for layer in base_model.layers:
        layer.trainable=False
retrain_model.add(base_model)
retrain_model.add(layers.Flatten(input_shape=(32,32)))
retrain_model.add(Dense(2048, activation='relu'))
retrain_model.add(Dense(1024, activation='relu'))
retrain_model.add(Dense(512, activation='relu'))
retrain_model.add(Dense(248, activation='relu'))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dense(80, activation='relu'))
retrain_model.add(Dense(10, activation='softmax'))
retrain_model.summary()
plot_model(model, show_shapes=True)
```

This new model will be trained only on new weights on user supplied test data to have it fitted on wrong predicted images.

The model is trained upto 30 epochs , alternatively, it can be trained with early stopping keeping a certain patience threshold.



```
epoch = 50
# update model on new data only with a smaller learning rate
opt = tf.keras.optimizers.Adam(1e-3)
# compile the model
retrain_model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
r = retrain_model.fit(X_train_new, y_train_new, epochs= epoch, batch_size=32, validation_data = (X_val , y_val), verbose = True)
```



```
[ ] test_loss, test_acc = retrain_model.evaluate(X_test, y_test)
    print("accuracy:", test_acc)

    157/157 [==============================] - 1s 5ms/step - loss: 0.8856 - accuracy: 0.7912
    accuracy: 0.7911999821662903

    test_loss, test_acc = retrain_model.evaluate(X_train_new, y_train_new)
    print("accuracy:", test_acc)

    4/4 [==============================] - 0s 6ms/step - loss: 0.4066 - accuracy: 0.8700
    accuracy: 0.8700000047683716
```

b.Observation-

The results are good, performance has improved on previously incorrectly predicted images and overall accuracy is the same.

c. Additionally, a round of fine-tuning of the entire model to give a final touch. unfreeze the base model and train the entire model end-to-end with a very low

learning rate.

base_model.trainable = True

retrain_model.summary()

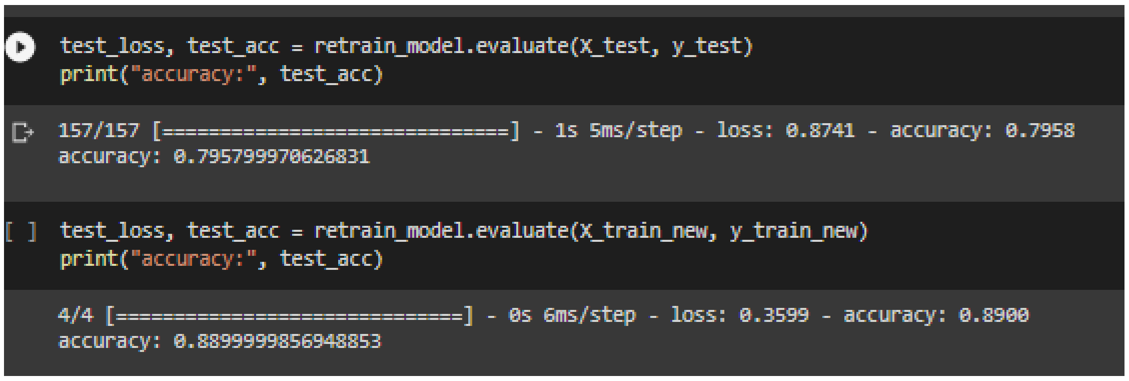
retrain_model.compile(optimizer=tf.keras.optimizers.Adam(1e-5), # Low learning rate

loss='categorical_crossentropy', metrics=['accuracy'])

epoch = 15

r = retrain_model.fit(X_train_new , y_train_new , batch_size = 32, epochs = epoch , validation_data = (X_val , y_val), verbose = True)

If it enhances the performance a bit, then we can save it as the final model , otherwise it can be ignored,



```
test_loss, test_acc = retrain_model.evaluate(X_test, y_test)
print("accuracy:", test_acc)

157/157 [==============================] - 1s 5ms/step - loss: 0.8741 - accuracy: 0.7958
accuracy: 0.795799970626831

[ ] test_loss, test_acc = retrain_model.evaluate(X_train_new, y_train_new)
print("accuracy:", test_acc)

4/4 [==============================] - 0s 6ms/step - loss: 0.3599 - accuracy: 0.8900
accuracy: 0.8899999856948853
```

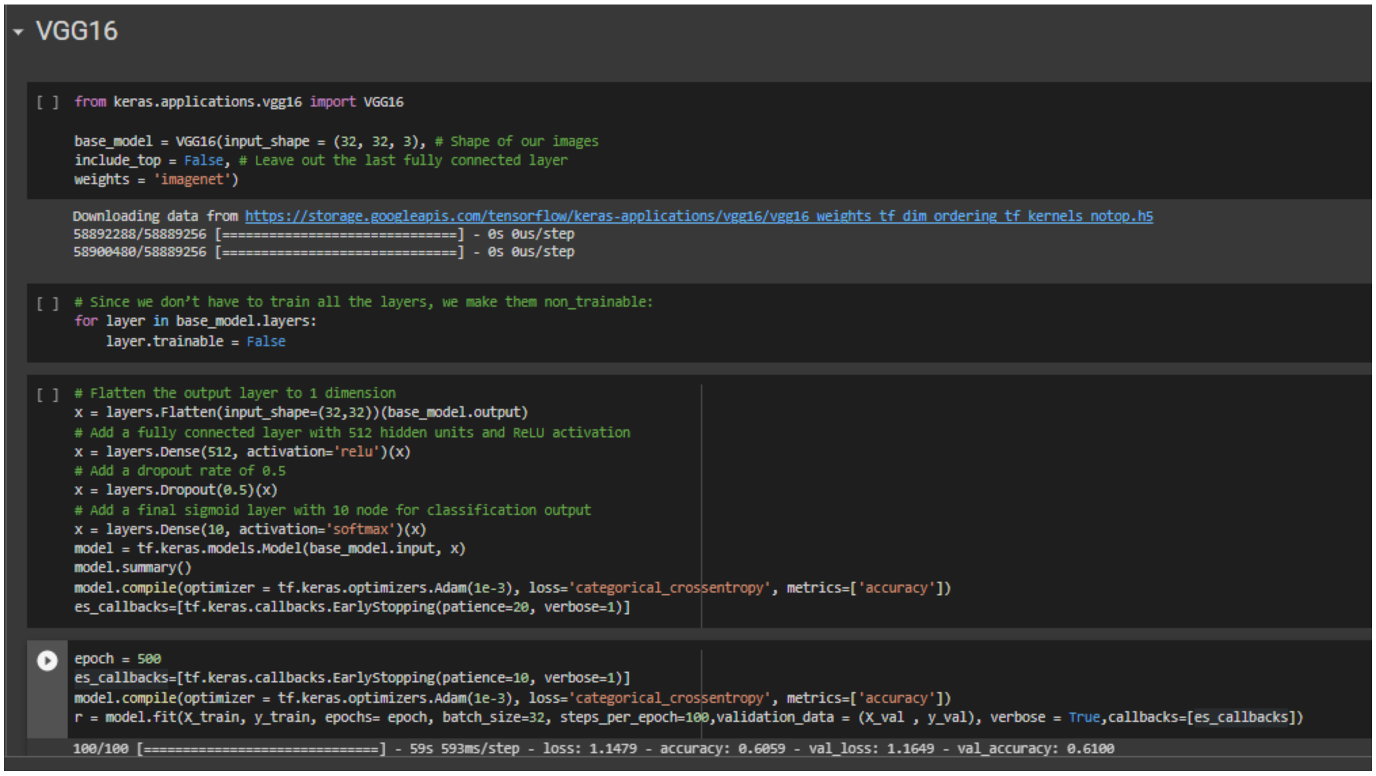
d. Observation-

In our case, it helps in reaching accuracy better than the previous one.

Getting good results on using transfer learning on existing model & later doing finetuning. Accuracy improvement is directly proportional to the amount of data used in transfer learning.

e. Training keras other pretrained models

We tried out our test dataset with VGG16 & RESNET models.



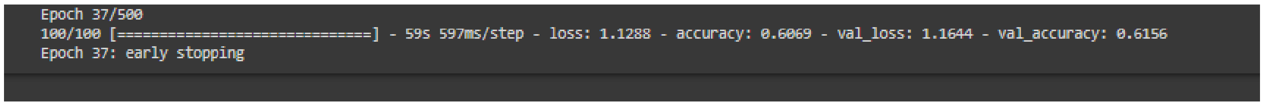
VGG16

```
[ ] from keras.applications.vgg16 import VGG16

    base_model = VGG16(input_shape = (32, 32, 3), # Shape of our images
    include_top = False, # Leave out the last fully connected layer
    weights = 'imagenet')

    Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
    58892288/58889256 [==============================] - 0s 0us/step
    58900480/58889256 [==============================] - 0s 0us/step
```

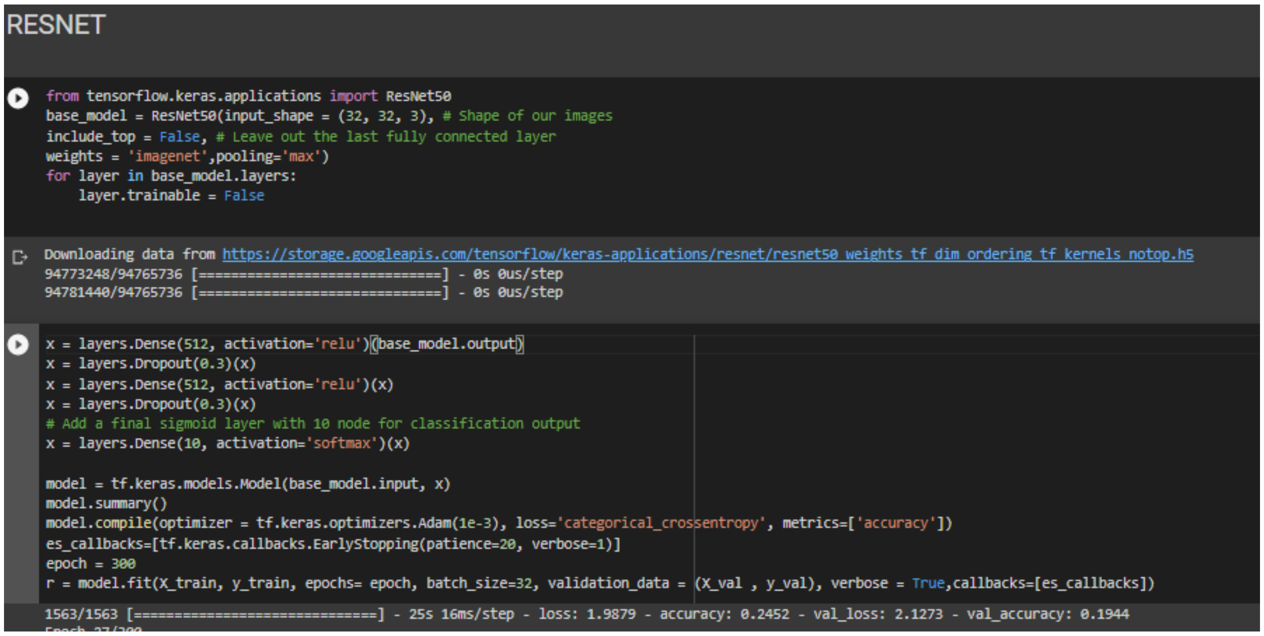
```
[ ] # Since we don't have to train all the layers, we make them non_trainable:
    for layer in base_model.layers:
        layer.trainable = False
```

```
[ ] # Flatten the output layer to 1 dimension
    x = layers.Flatten(input_shape=(32,32))(base_model.output)
    # Add a fully connected layer with 512 hidden units and ReLU activation
    x = layers.Dense(512, activation='relu')(x)
    # Add a dropout rate of 0.5
    x = layers.Dropout(0.5)(x)
    # Add a final sigmoid layer with 10 node for classification output
    x = layers.Dense(10, activation='softmax')(x)
    model = tf.keras.models.Model(base_model.input, x)
    model.summary()
    model.compile(optimizer = tf.keras.optimizers.Adam(1e-3), loss='categorical_crossentropy', metrics=['accuracy'])
    es_callbacks=[tf.keras.callbacks.EarlyStopping(patience=20, verbose=1)]
```

```
    epoch = 500
    es_callbacks=[tf.keras.callbacks.EarlyStopping(patience=10, verbose=1)]
    model.compile(optimizer = tf.keras.optimizers.Adam(1e-3), loss='categorical_crossentropy', metrics=['accuracy'])
    r = model.fit(X_train, y_train, epochs= epoch, batch_size=32, steps_per_epoch=100,validation_data = (X_val , y_val), verbose = True,callbacks=[es_callbacks])
    100/100 [==============================] - 59s 593ms/step - loss: 1.1479 - accuracy: 0.6059 - val_loss: 1.1649 - val_accuracy: 0.6100
```



```
Epoch 37/500
100/100 [==============================] - 59s 597ms/step - loss: 1.1288 - accuracy: 0.6069 - val_loss: 1.1644 - val_accuracy: 0.6156
Epoch 37: early stopping
```



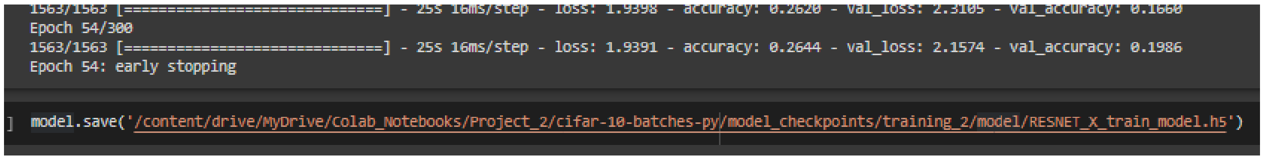
RESNET

```
from tensorflow.keras.applications import ResNet50
base_model = ResNet50(input_shape = (32, 32, 3), # Shape of our images
include_top = False, # Leave out the last fully connected layer
weights = 'imagenet',pooling='max')
for layer in base_model.layers:
    layer.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94773248/94765736 [==============================] - 0s 0us/step
94781440/94765736 [==============================] - 0s 0us/step
```

```
x = layers.Dense(512, activation='relu')(base_model.output)
x = layers.Dropout(0.3)(x)
x = layers.Dense(512, activation='relu')(x)
x = layers.Dropout(0.3)(x)
# Add a final sigmoid layer with 10 node for classification output
x = layers.Dense(10, activation='softmax')(x)

model = tf.keras.models.Model(base_model.input, x)
model.summary()
model.compile(optimizer = tf.keras.optimizers.Adam(1e-3), loss='categorical_crossentropy', metrics=['accuracy'])
es_callbacks=[tf.keras.callbacks.EarlyStopping(patience=20, verbose=1)]
epoch = 300
r = model.fit(X_train, y_train, epochs= epoch, batch_size=32, validation_data = (X_val , y_val), verbose = True,callbacks=[es_callbacks])
```

```
1563/1563 [==============================] - 25s 16ms/step - loss: 1.9879 - accuracy: 0.2452 - val_loss: 2.1273 - val_accuracy: 0.1944
```



```
1563/1563 [==============================] - 25s 16ms/step - loss: 1.9398 - accuracy: 0.2620 - val_loss: 2.3105 - val_accuracy: 0.1660
Epoch 54/300
1563/1563 [==============================] - 25s 16ms/step - loss: 1.9391 - accuracy: 0.2644 - val_loss: 2.1574 - val_accuracy: 0.1986
Epoch 54: early stopping
```

```
model.save('/content/drive/MyDrive/Colab_Notebooks/Project_2/cifar-10-batches-py/model_checkpoints/training_2/model/RESNET_X_train_model.h5')
```

f.Observations-

1. VGG16 & RESNET models are giving approximately 50% and 20% accuracy respectively .

2. Pretrained models are under performing on the test data when compared to our custom model ,

Conclusion-

We will go ahead with our custom model & deploy it in production.

9. Deployment and Productionisation

The final phase of this project.is Deployment, here we are deploying the whole machine learning pipeline into a production system, into a real-time scenario.

In this final stage we need to deploy this machine learning pipeline to put of research available to end users for use. The model will be deployed in real world scenario where it takes continuous raw input from real world and predict the output.

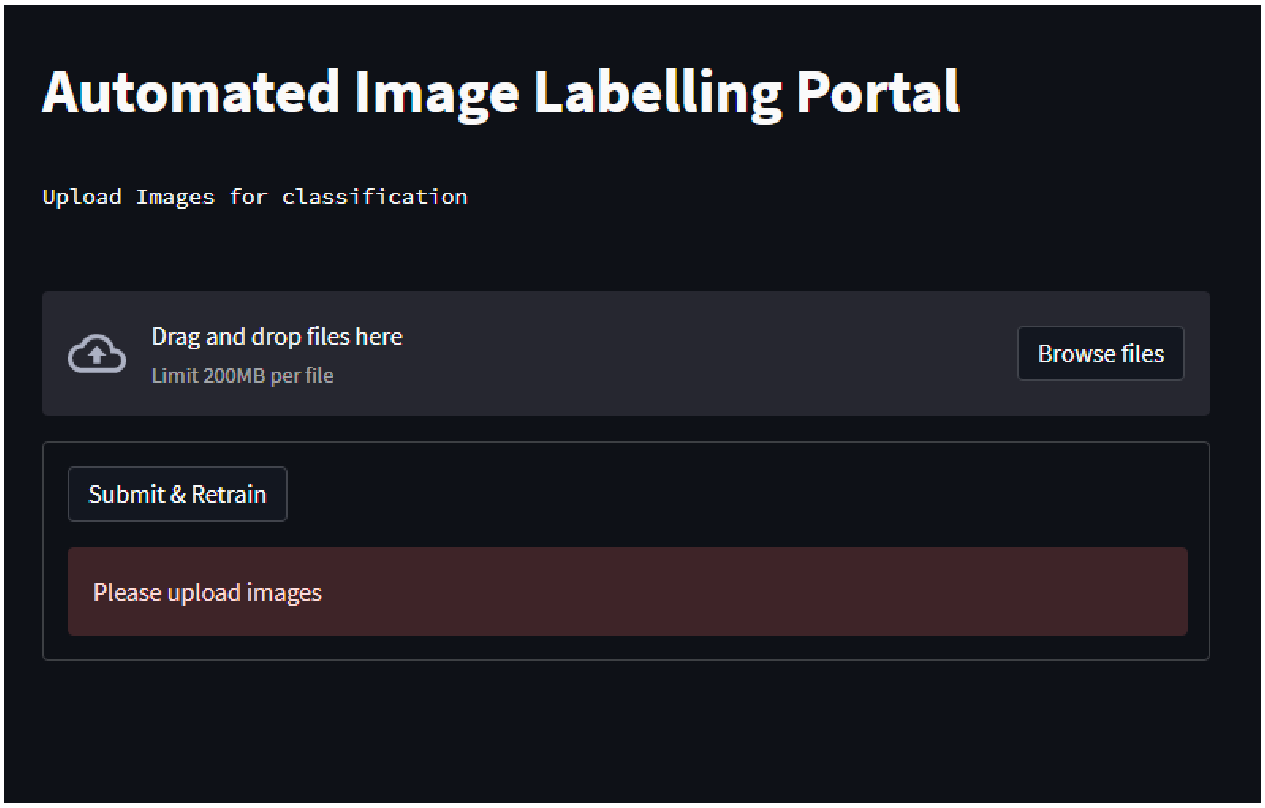
App Building:

In this step a web application developed using Streamlit. It provides a functionality to the user for uploading the input in csv format.

a. User interface

User interface is built on a Streamlit application which is an open source platform.

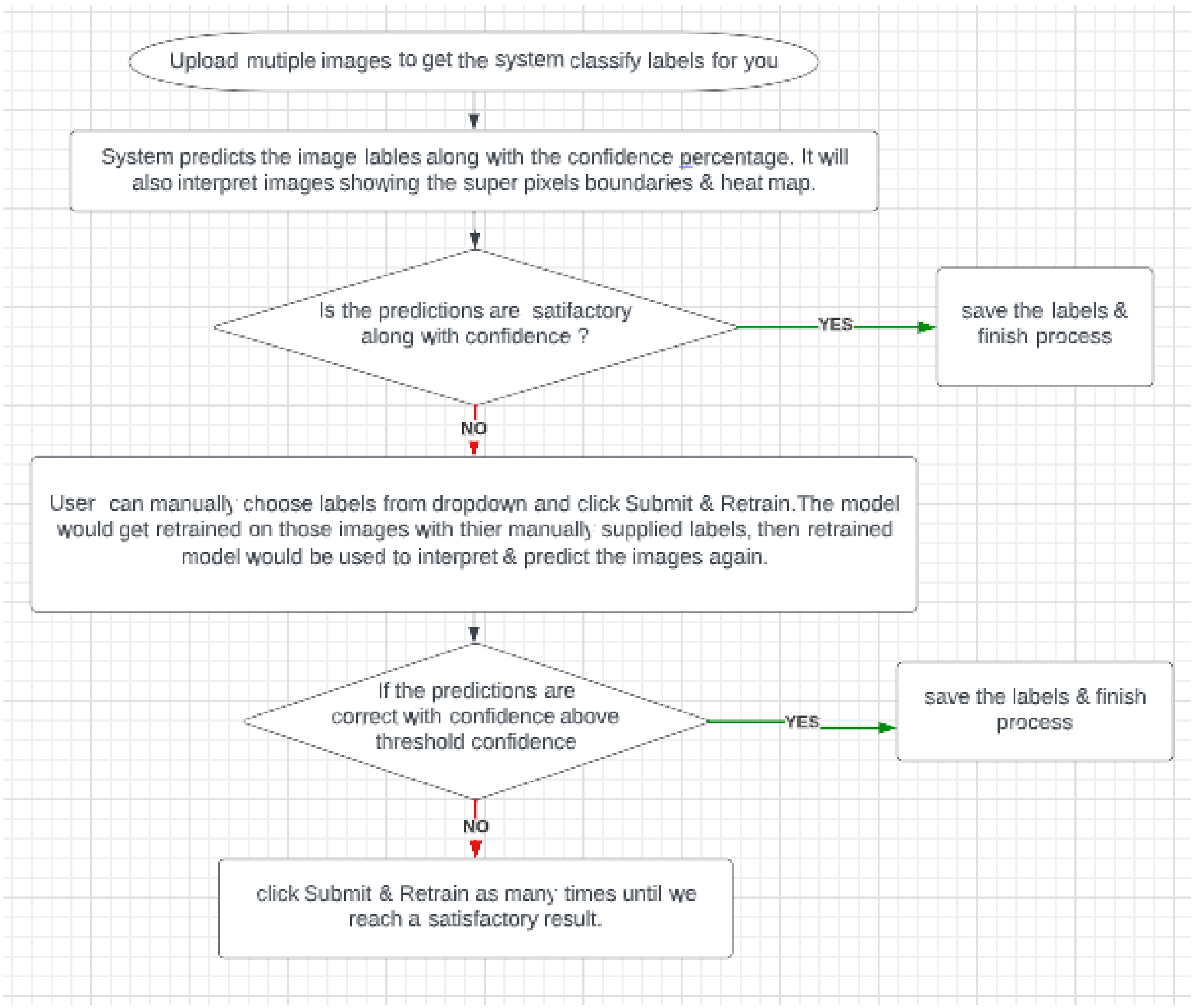
The code is written in stream_app.py file which will use baseline model & retrained model to showcase how active learning improves prediction accuracy & confidence scores on the predicted labels.


Automated Image Labelling Portal

Upload Images for classification

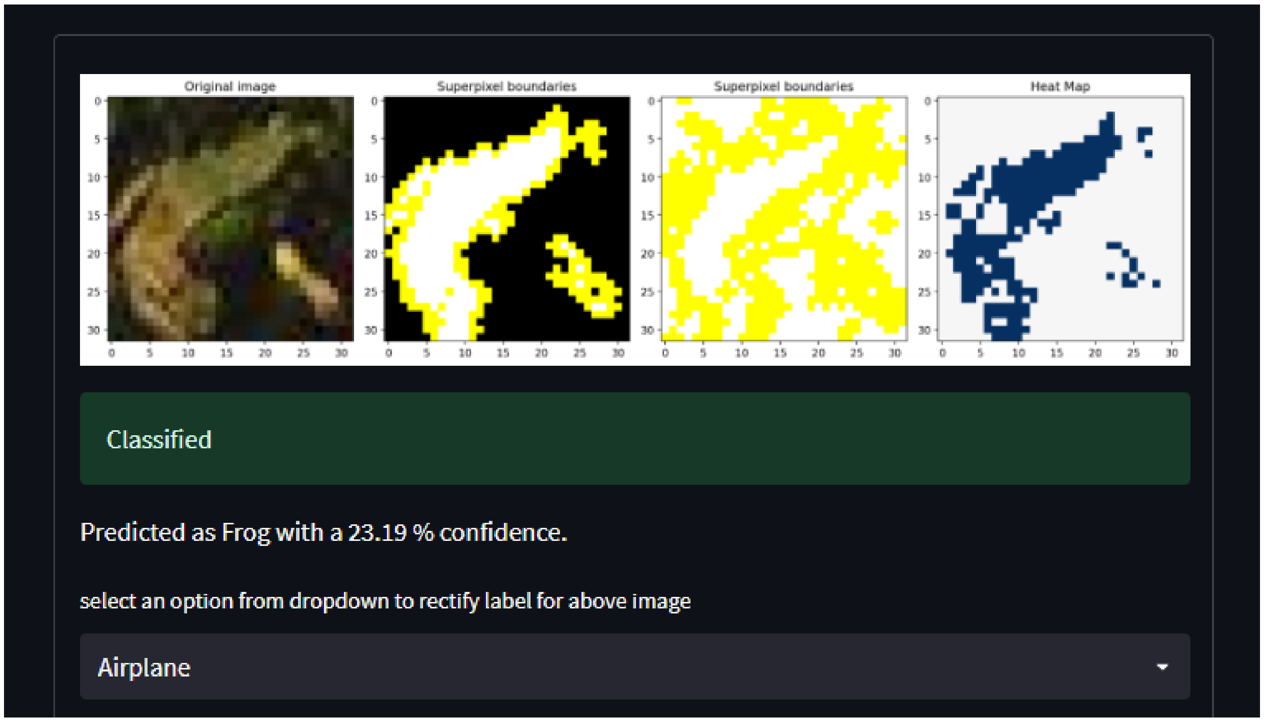
Please upload images

b. Flowchart


Upload mutiple images to get the system classify labels for you
System predicts the image lables along with the confidence percentage. It will also interpret images showing the super pixels boundaries & heat map.
Is the predictions are satifactory along with confidence ?
YES
save the labels & finish process
NO
User can manually choose labels from dropdown and click Submit & Retrain.The model would get retrained on those images with thier manually supplied labels, then retrained model would be used to interpret & predict the images again.
If the predictions are correct with confidence above threshold confidence
YES
save the labels & finish process
NO
click Submit & Retrain as many times until we reach a satisfactory result.

c. Working

Once images are uploaded in order to classify as one of the 10 labels, System predicts the image labels along with the confidence percentage. It will also interpret images showing the super pixels boundaries (positive & negative) & heat map plot. consisting of blue & red areas.

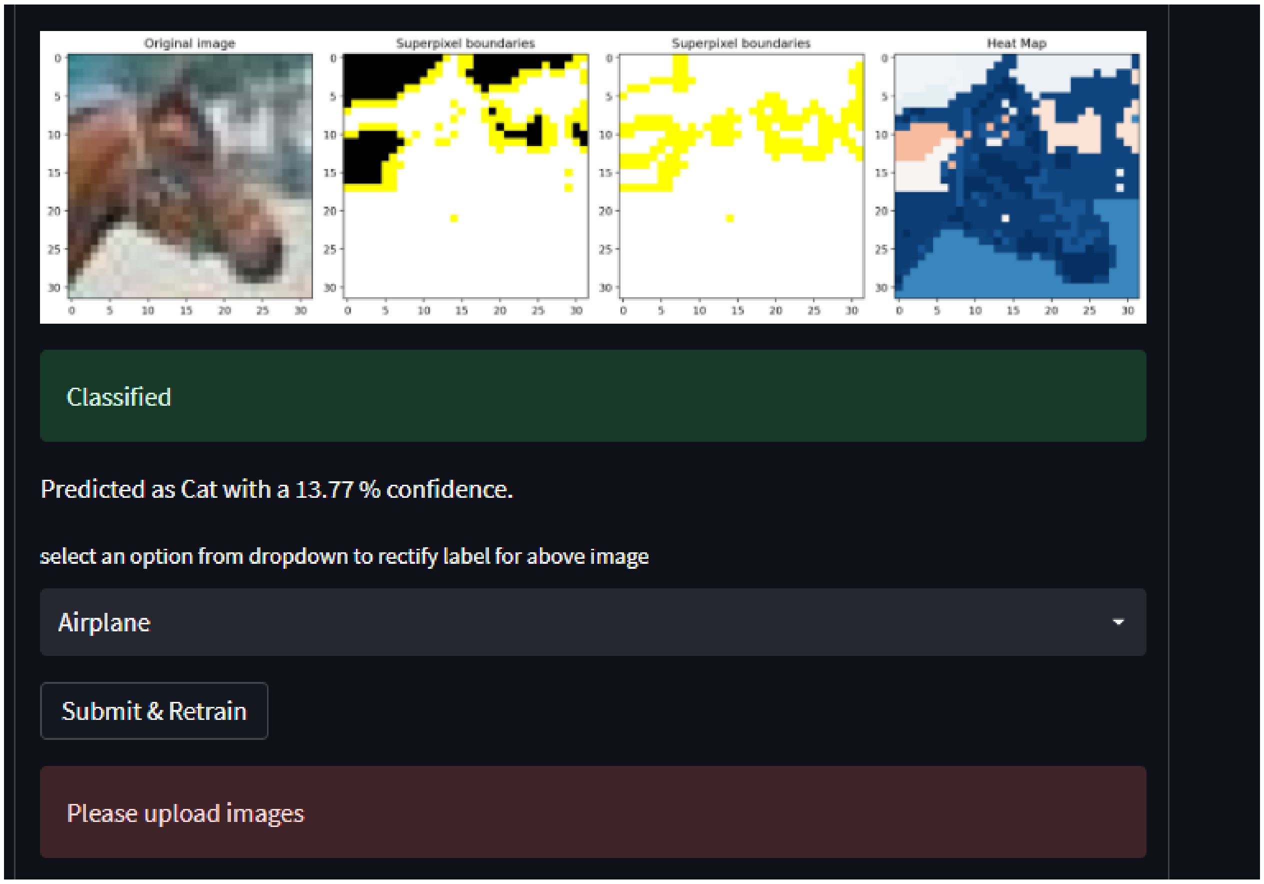


Classified

Predicted as Frog with a 23.19 % confidence.

select an option from dropdown to rectify label for above image

Airplane



Classified

Predicted as Cat with a 13.77 % confidence.

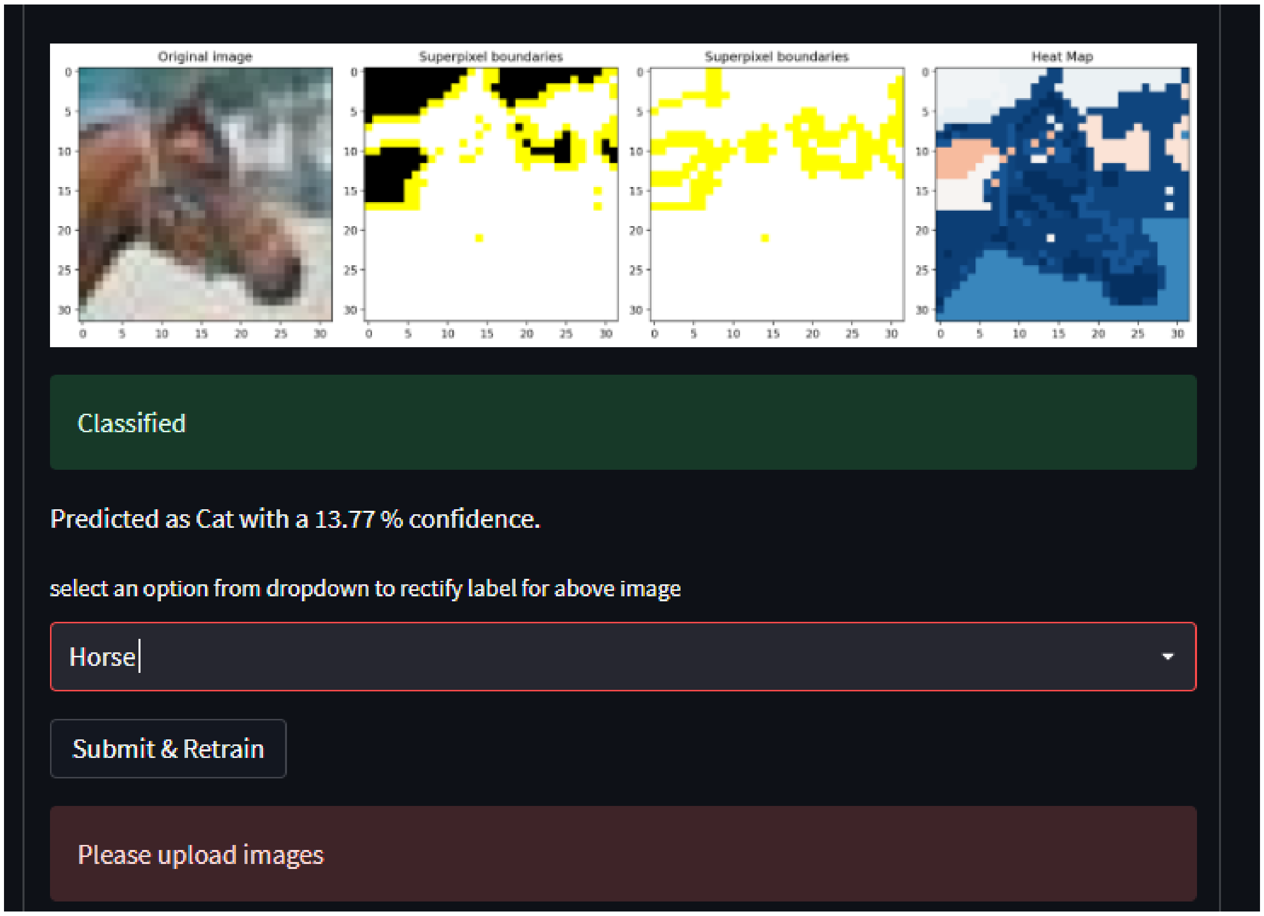
select an option from dropdown to rectify label for above image

Airplane

Please upload images

Whether the predictions are satisfactory along with confidence, then, we can save correctly satisfied labels along with images & can have our unlabelled dataset transformed to a labeled one.

User can manually choose labels from dropdown and click Submit & Retrain.The model would get retrained on those images with their manually supplied labels, then the retrained model would be used to interpret & predict the images again.



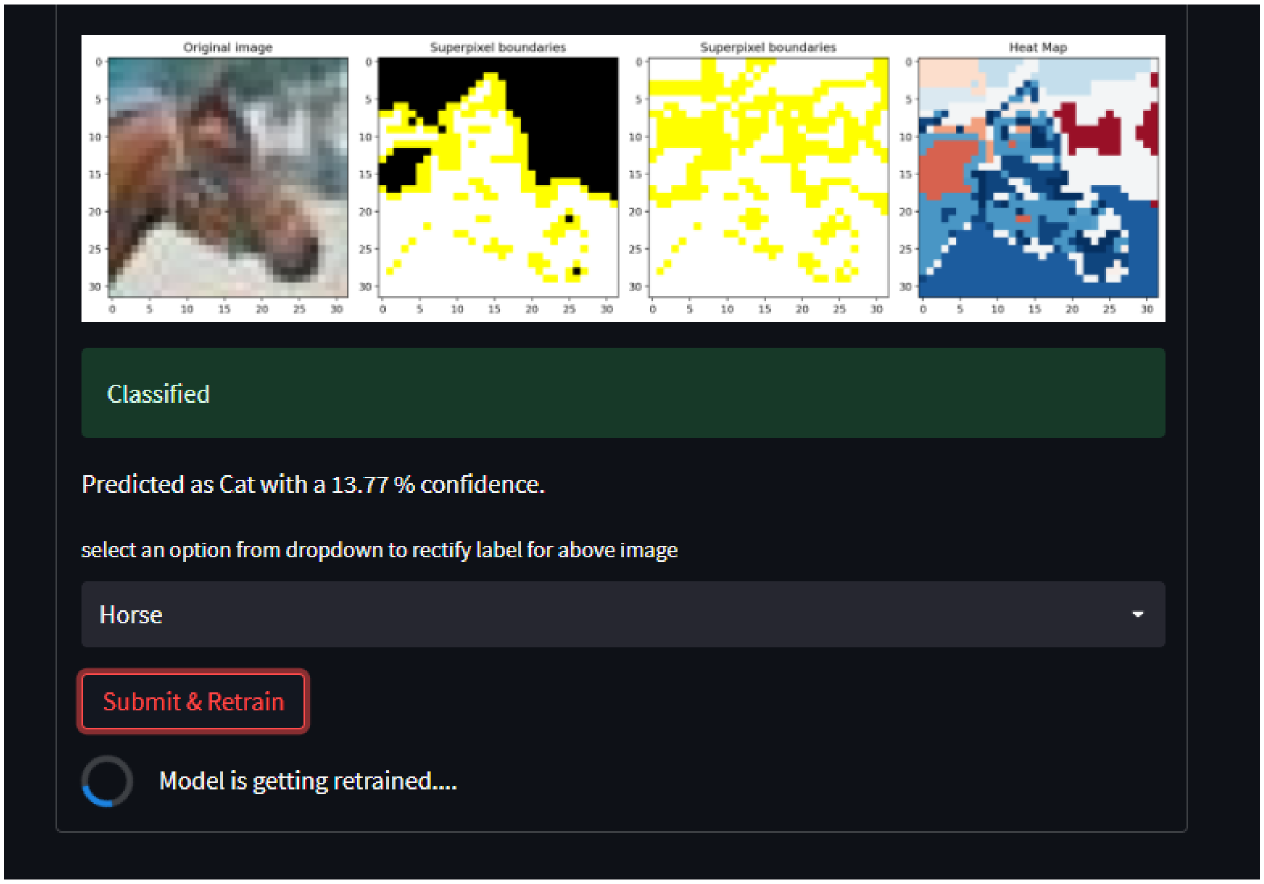
Classified

Predicted as Cat with a 13.77 % confidence.

select an option from dropdown to rectify label for above image

Horse

Please upload images



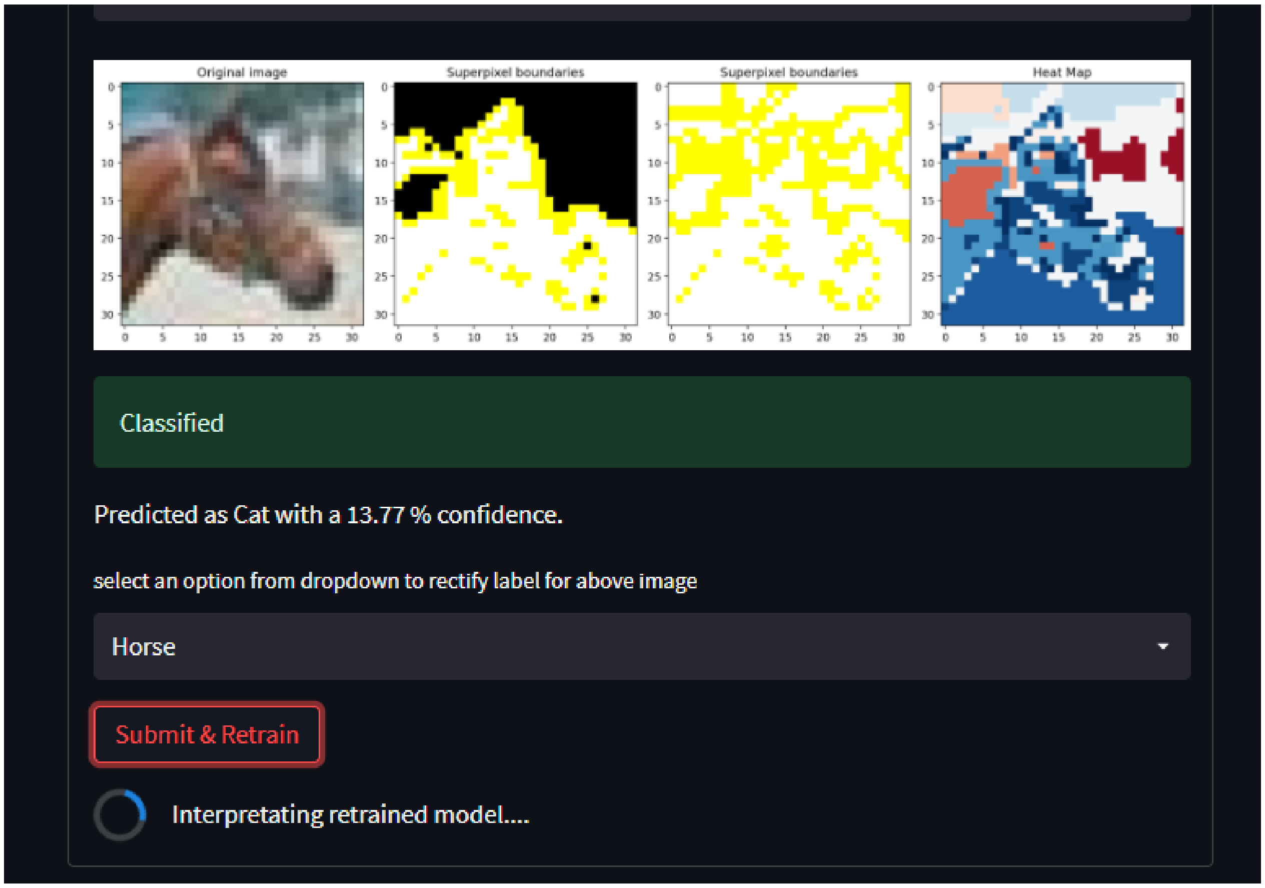
Classified

Predicted as Cat with a 13.77 % confidence.

select an option from dropdown to rectify label for above image

Horse

Model is getting retrained....



Classified

Predicted as Cat with a 13.77 % confidence.

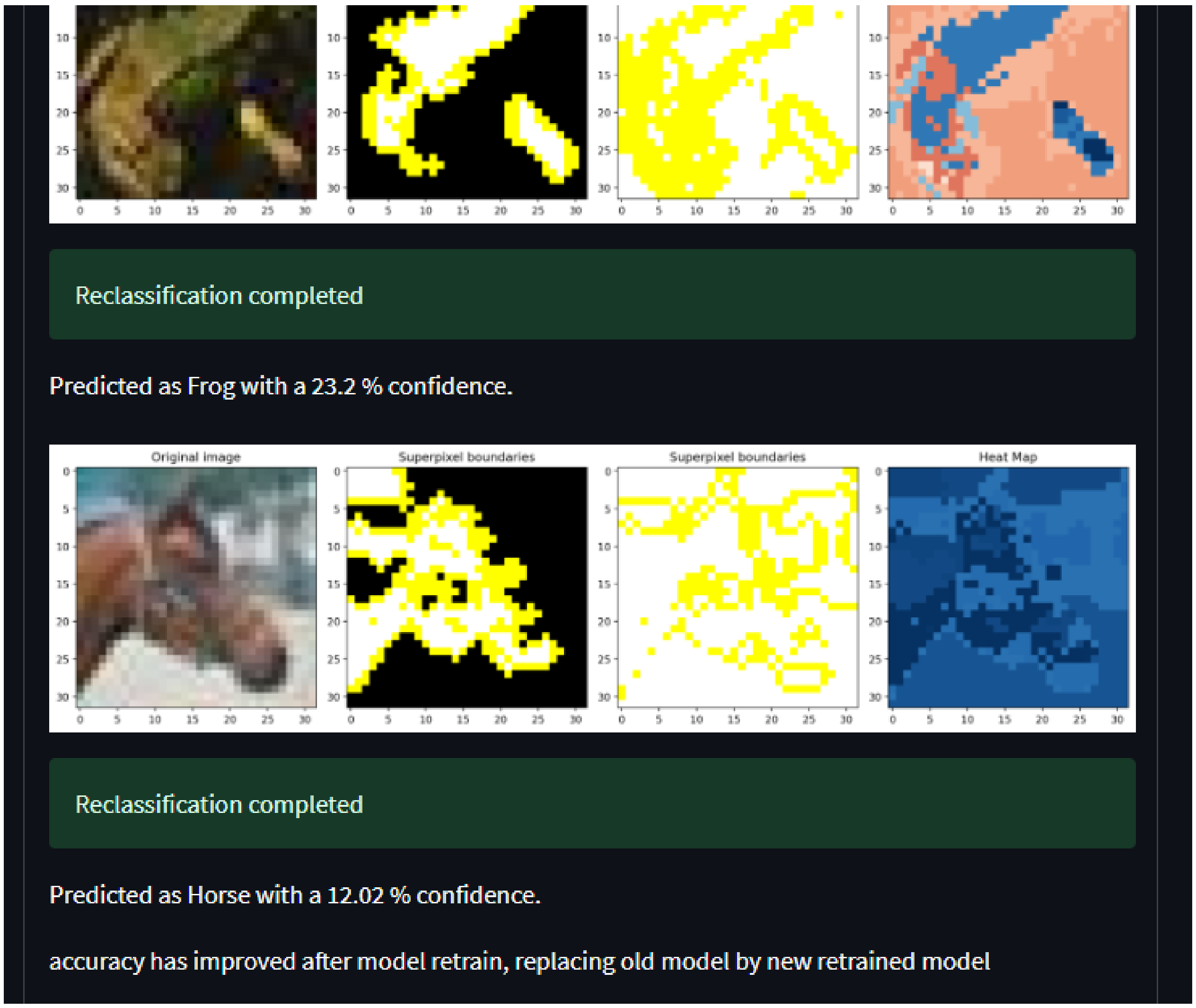
select an option from dropdown to rectify label for above image

Horse

Interpretating retrained model....

If the predictions are correct with confidence above threshold confidence

save the labels & finish process.Otherwise, click Submit & Retrain as many times until we reach a satisfactory result.



Reclassification completed

Predicted as Frog with a 23.2 % confidence.

Reclassification completed

Predicted as Horse with a 12.02 % confidence.

accuracy has improved after model retrain, replacing old model by new retrained model

d. Productionisation-

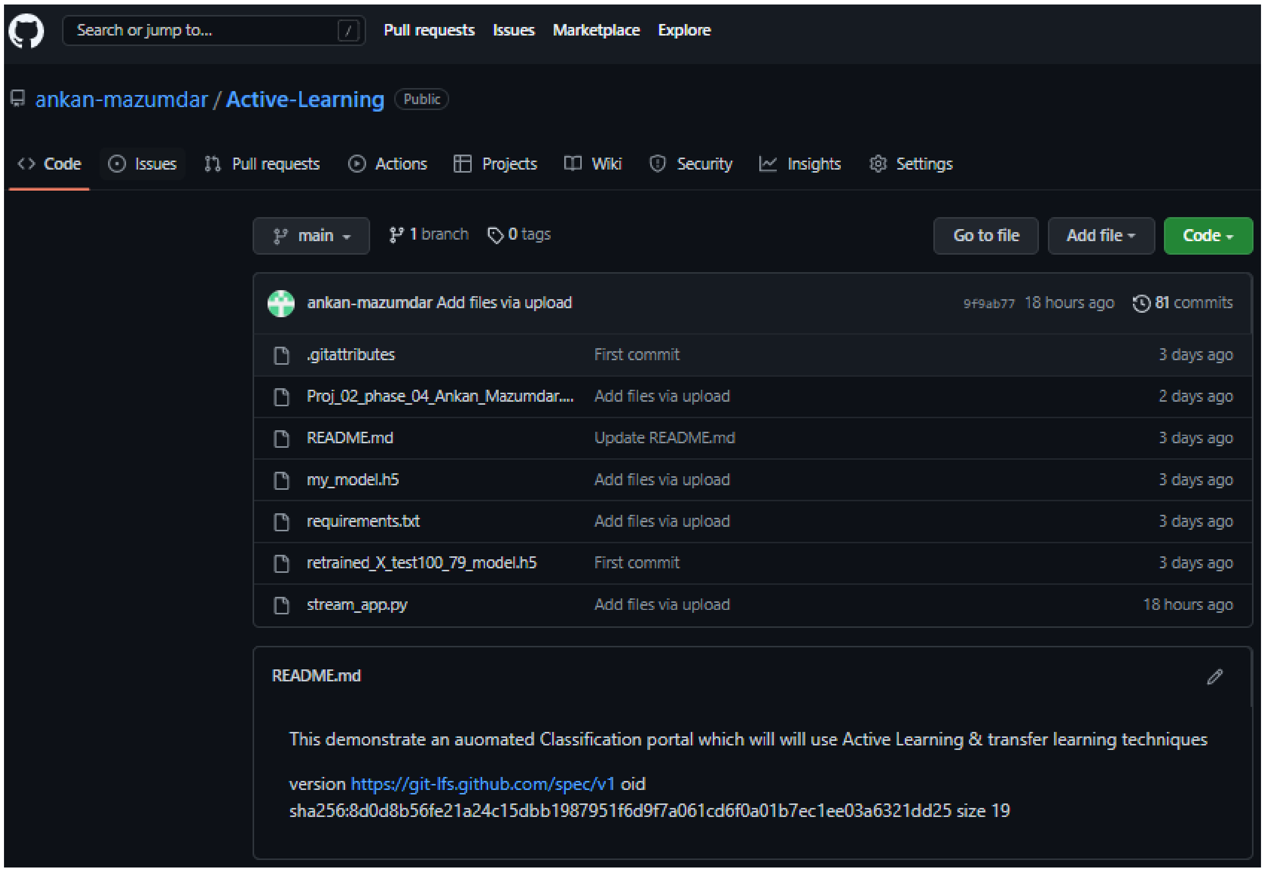
App is Deployed to streamlit cloud, Please access below link-

https://ankan-mazumdar-active-learning-stream-app-kbbcqv.streamlitapp.com/

Steps to deploy-

Upload file from local to github repo-

https://github.com/ankan-mazumdar/Active-Learning



ankan-mazumdar Add files via upload 9f9ab77 18 hours ago 81 commits

.gitattributes	First commit	3 days ago
Proj_02_phase_04_Ankan_Mazumdar....	Add files via upload	2 days ago
README.md	Update README.md	3 days ago
my_model.h5	Add files via upload	3 days ago
requirements.txt	Add files via upload	3 days ago
retrained_X_test100_79_model.h5	First commit	3 days ago
stream_app.py	Add files via upload	18 hours ago

README.md

This demonstrate an auomated Classification portal which will will use Active Learning & transfer learning techniques

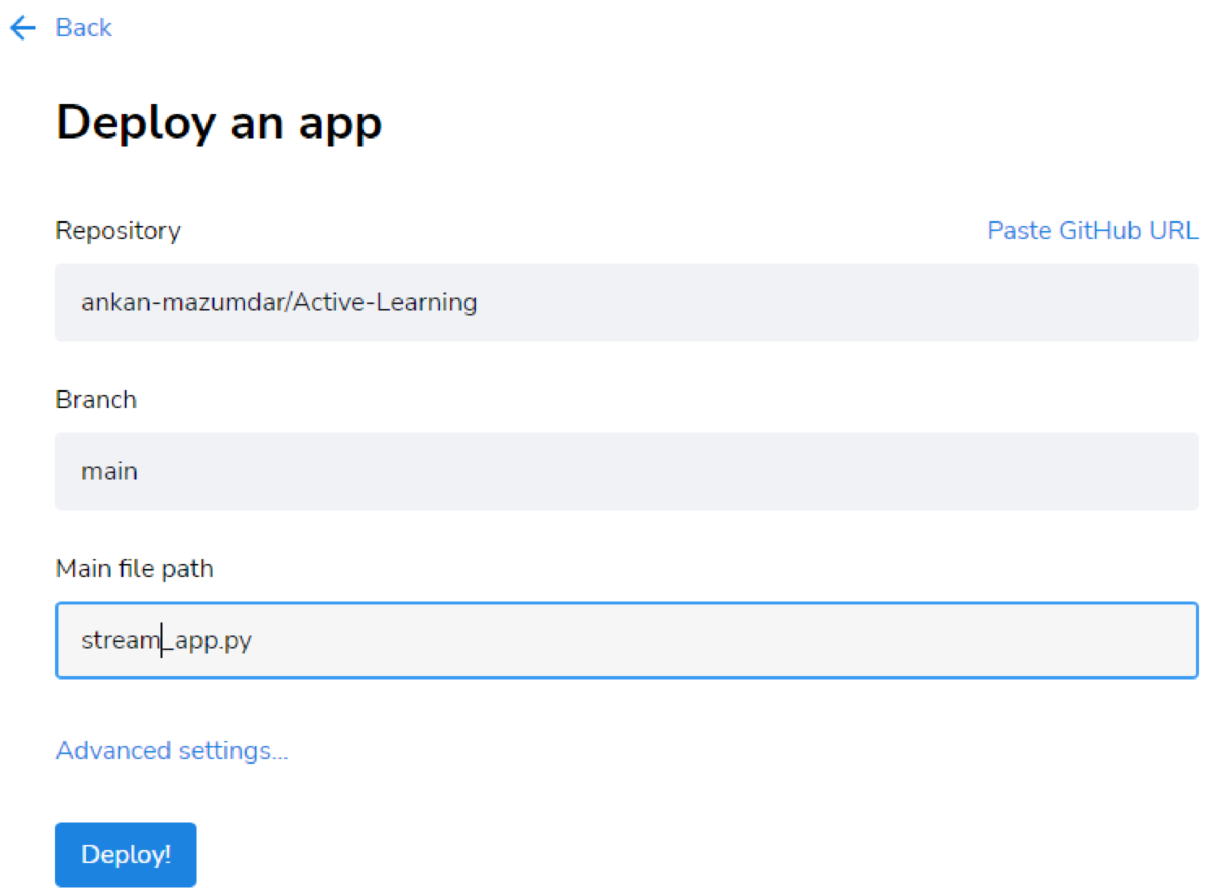
version https://git-lfs.github.com/spec/v1 oid sha256:8d0d8b56fe21a24c15dbb1987951f6d9f7a061cd6f0a01b7ec1ee03a6321dd25 size 19

For uploading large files in git ,have followed the git lfs documentation- https://git-lfs.github.com/

Add a requirements file for Python dependencies.

Next, need to go to streamlit cloud deployment link- https://share.streamlit.io/deploy

and enter the repo details & submit



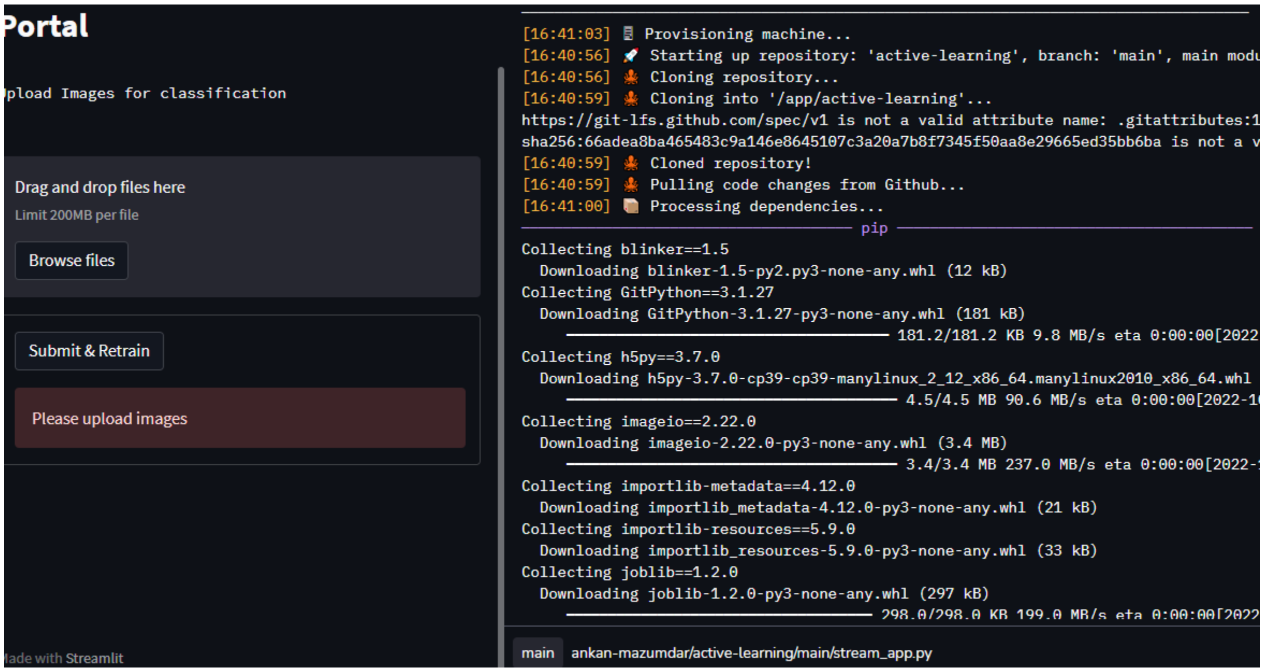
Deploy an app

Repository: ankan-mazumdar/Active-Learning

Branch: main

Main file path: stream_app.py

Once deployment is done, The app will be loaded, the execution logs are loaded side by side which helps to verify if deployment completed successfully or to look for any errors.



```
[16:41:03] 🗄 Provisioning machine...
[16:40:56] 🚀 Starting up repository: 'active-learning', branch: 'main', main modu
[16:40:56] 🐙 Cloning repository...
[16:40:59] 🐙 Cloning into '/app/active-learning'...
https://git-lfs.github.com/spec/v1 is not a valid attribute name: .gitattributes:1
sha256:66adea8ba465483c9a146e8645107c3a20a7b8f7345f50aa8e29665ed35bb6ba is not a v
[16:40:59] 🐙 Cloned repository!
[16:40:59] 🐙 Pulling code changes from Github...
[16:41:00] 📦 Processing dependencies...
──────────────────────────────────── pip ────────────────────────────────────
Collecting blinker==1.5
  Downloading blinker-1.5-py2.py3-none-any.whl (12 kB)
Collecting GitPython==3.1.27
  Downloading GitPython-3.1.27-py3-none-any.whl (181 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 181.2/181.2 KB 9.8 MB/s eta 0:00:00[2022
Collecting h5py==3.7.0
  Downloading h5py-3.7.0-cp39-cp39-manylinux_2_12_x86_64.manylinux2010_x86_64.whl
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.5/4.5 MB 90.6 MB/s eta 0:00:00[2022-1
Collecting imageio==2.22.0
  Downloading imageio-2.22.0-py3-none-any.whl (3.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.4/3.4 MB 237.0 MB/s eta 0:00:00[2022-
Collecting importlib-metadata==4.12.0
  Downloading importlib_metadata-4.12.0-py3-none-any.whl (21 kB)
Collecting importlib-resources==5.9.0
  Downloading importlib_resources-5.9.0-py3-none-any.whl (33 kB)
Collecting joblib==1.2.0
  Downloading joblib-1.2.0-py3-none-any.whl (297 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 298.0/298.0 KB 199.0 MB/s eta 0:00:00[2022
```

ankan-mazumdar/active-learning/main/stream_app.py

e. Observations-

Streamlit provides an easy ,quick & beautiful deployment platform, however it has its own limitation of further customization.

Dependency on github as code repo, makes it difficult to create & save models in github in runtime.

Streamlit is consistent with no outages , however , hasn't tested its scalability nor how many users it can handle at the same time.

Model is trained only on the CIFAR 10 dataset. API is limited to predict only 10 classes present in the dataset itself, any object intended to be classified out of the 10 classes in CIFAR 10 would be invalid .

10. References-

- https://python-data-science.readthedocs.io/en/latest/activelearning.html
- https://paperswithcode.com/task/active-learning/latest
- https://www.diva-portal.org/smash/get/diva2:1586990/FULLTEXT01.pdf

- https://www.datacamp.com/tutorial/active-learning
- https://en.wikipedia.org/wiki/Active_learning_(machine_learning)
- https://drive.google.com/file/d/11Rqk0tGHExrb2LwoecUwPSPr0Y6VE-he/view
- https://towardsdatascience.com/identifying-the-right-classification-metric-for-yourtask-21727fa218a2
- https://www.kaggle.com/code/nkitgupta/evaluation-metrics-for-multi-class-classific ation/notebook
- https://keras.io/api/optimizers/
- https://www.tensorflow.org/api_docs/python/tf/keras/Sequential
- tensorflow conv2d
- Batch Normalization (the original paper)
- Dropout (the original paper)
- Understanding Dropout / deeplearning.ai — Andrew Ng.
- Tensorflow Softmax Cross Entropy with Logits
- Tensorflow Reduce Mean
- Tensorflow Optimizers
- An overview of gradient descent optimization algorithms
- Optimization for Training Deep Models
- Tensorflow tf.train.AdamOptimizer
- https://www.youtube.com/watch?v=HxtBIwfy0kM&t=105s
- https://www.kaggle.com/code/muhammadhafil/image-classification-basics
- http://yann.lecun.com/exdb/publis/pdf/lecun-99.pdf
-
- https://numpy.org/doc/stable/reference/generated/numpy.transpose.html
- https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator
- https://matplotlib.org/stable/tutorials/introductory/images.html
- https://www.tensorflow.org/api_docs/python/tf/keras/utils/to_categorical
- https://medium.com/analytics-vidhya/a-tip-a-day-python-tip-8-why-should-we-normalize-image-pixel-values-or-divide-by-255-4608ac5cd26a
-
-
- https://docs.streamlit.io/
- https://docs.github.com/en/billing/managing-billing-for-git-large-file-storage/viewing-your-git-large-file-storage-usage
- https://www.geeksforgeeks.org/git-lfs-large-file-storage/
- https://discuss.streamlit.io/t/oserror-unable-to-open-file-from-githib/29480/6
- https://git-lfs.github.com/
- https://share.streamlit.io/deploy
- https://keras.io/api/optimizers/
- https://keras.io/api/models/sequential/
- https://keras.io/guides/sequential_model
- https://www.tensorflow.org/api_docs/python/tf/keras/Sequential
- Tensorflow Reduce Mean
- Tensorflow Optimizers

- An overview of gradient descent optimization algorithms
- Optimization for Training Deep Models
- Tensorflow tf.train.AdamOptimizer
- https://www.youtube.com/watch?v=HxtBIwfy0kM&t=105s
- https://www.kaggle.com/code/muhammadhafil/image-classification-basics
- http://yann.lecun.com/exdb/publis/pdf/lecun-99.pdf