

ASSIGNMENT 3 / MINI PROJECT

VISUAL RECOGNITION(VR)

Name of Group Members:-

HUSSNAIN ASHRAF (MT2021055)

KUNAL (MT2021067)

REPORT OF ASSIGNMENT –

3a

Implementation of AlexNet Convolutional Neural Network(CNN) architecture on CIFAR -10 using TensorFlow and Keras.

AlexNet Implementation and Discussion:-

AlexNet is the name of a convolutional neural network which has had a large impact on the field of machine learning, specifically in the application of deep learning to machine vision.

It consisted of 11×11 , 5×5 , 3×3 , convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum. It attached ReLU activations after every convolutional and fully-connected layer.

Some Key Points of AlexNet:-

1. Relu activation function is used instead of Tanh to add non-linearity. It accelerates the speed by 6 times at the same accuracy.
2. Use dropout instead of regularization to deal with overfitting. However, the training time is doubled with the dropout rate of 0.5.
3. Overlap pooling to reduce the size of the network. It reduces the top-1 and top-5 error rates by 0.4% and 0.3%, respectively.

Dataset:-

We are using the CIFAR-10 dataset which contains 60,000 color images, each with dimensions 32x32px. The content of the images within the dataset is sampled from 10 classes.

These 10 classes are depicted as follows:-

1. Airplane

2. automobile

3. Bird

4. Cat

5. Deer

6. Dog

7. Frog

8. Horse

9. Ship

10. Truck

The CIFAR dataset is partitioned into 50,000 training data and 10,000 test data by default. The last partition of the dataset we require is the validation data.so here we have taken some small amount of training ,validation and test data because our training time was taking so much time.

The validation data is obtained by taking the last some amount of images within the training data.

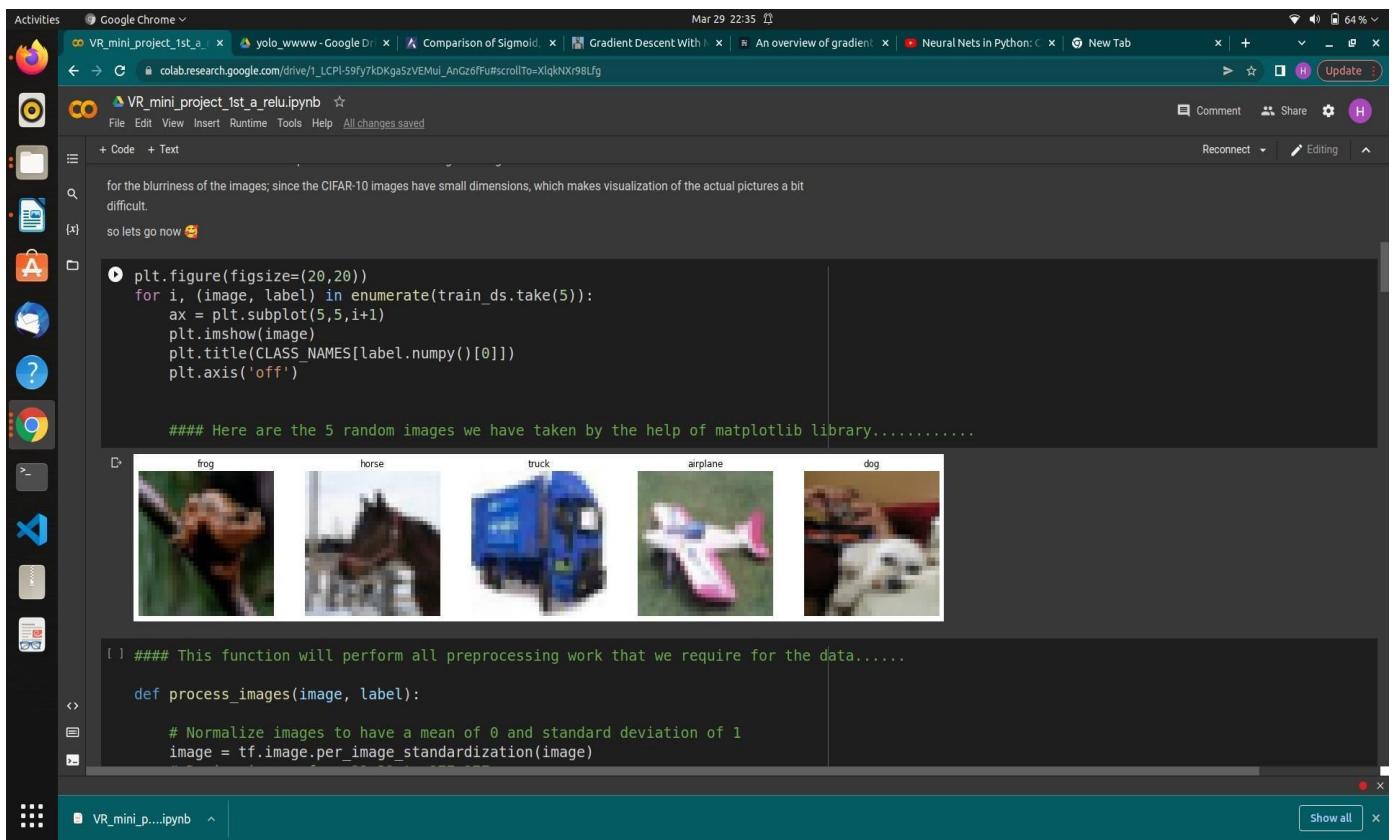
Preprocessing:-

Preprocessing in any machine learning is associated with the transformation of data from one form to another.

Usually, preprocessing is conducted to ensure the data utilized is within an appropriate format.

First, let's visualize the images within the CIFAR-10 dataset. The code picture below uses the Matplotlib library to present the pixel information of the data from five training images into actual images. There is also an **indicator of the class each depicted content within the images belongs to**.

Here the blurriness of the images are coming in the CIFAR-10 images because of having small dimensions, which makes visualization of the actual pictures a bit difficult.



The screenshot shows a Google Colab notebook titled "VR_mini_project_1st_a_relu.ipynb". The code cell contains the following Python code:

```
plt.figure(figsize=(20,20))
for i, (image, label) in enumerate(train_ds.take(5)):
    ax = plt.subplot(5,5,i+1)
    plt.imshow(image)
    plt.title(CLASS_NAMES[label.numpy()[0]])
    plt.axis('off')

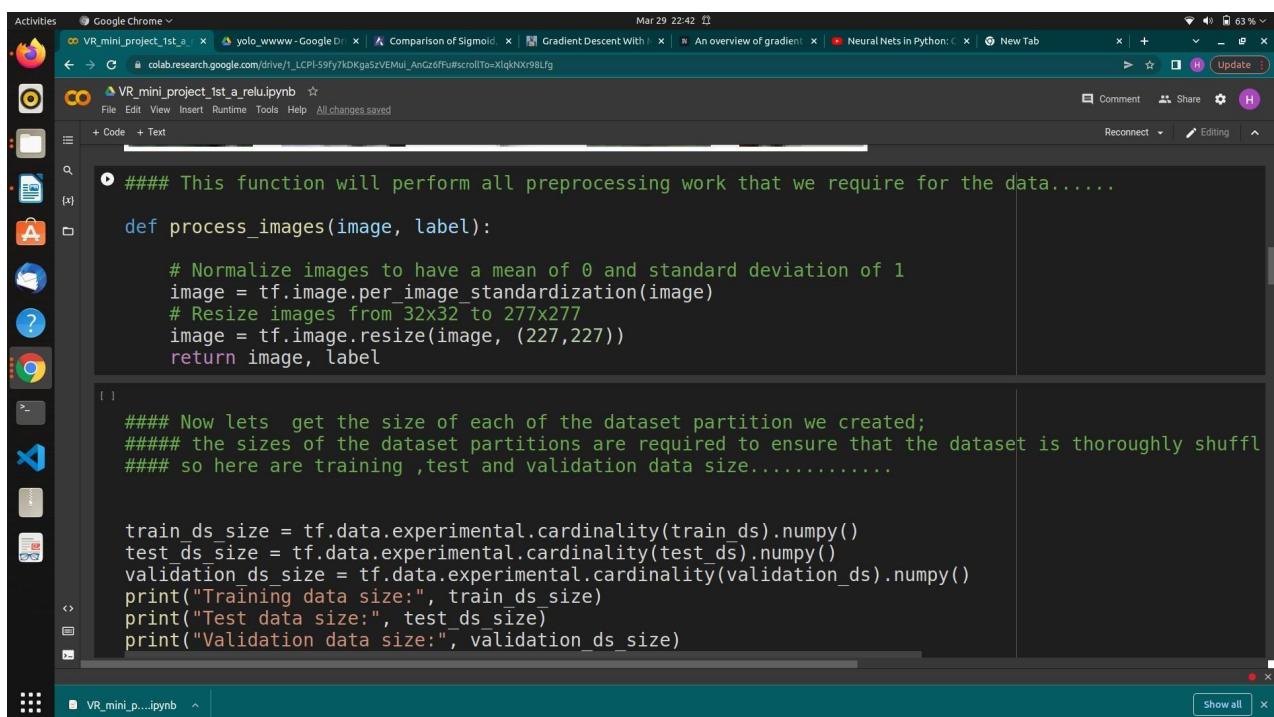
##### Here are the 5 random images we have taken by the help of matplotlib library.....
```

Below the code, five small images are displayed with their respective labels: frog, horse, truck, airplane, and dog. The "frog" image shows a close-up of a frog's head. The "horse" image shows a dark horse's head. The "truck" image shows a blue toy truck. The "airplane" image shows a white and pink toy airplane. The "dog" image shows a white dog's head.

The notebook interface includes a sidebar with various icons for file operations, a toolbar with a "Comment" button, and a status bar at the bottom indicating "VR_mini_p...ipynb" and "Show all".

The primary preprocessing transformations which we have done on the data are:

- **Normalizing and standardizing the images.**
- **Resizing of the images from 32x32 to 227x227. The AlexNet network input expects a 227x227 image.**



A screenshot of a Google Colab notebook titled "VR_mini_project_1st_a_relu.ipynb". The code in the cell is as follows:

```
#### This function will perform all preprocessing work that we require for the data.....  
def process_images(image, label):  
    # Normalize images to have a mean of 0 and standard deviation of 1  
    image = tf.image.per_image_standardization(image)  
    # Resize images from 32x32 to 227x227  
    image = tf.image.resize(image, (227,227))  
    return image, label  
  
#### Now lets get the size of each of the dataset partition we created;  
##### the sizes of the dataset partitions are required to ensure that the dataset is thoroughly shuffled  
##### so here are training ,test and validation data size.....  
  
train_ds_size = tf.data.experimental.cardinality(train_ds).numpy()  
test_ds_size = tf.data.experimental.cardinality(test_ds).numpy()  
validation_ds_size = tf.data.experimental.cardinality(validation_ds).numpy()  
print("Training data size:", train_ds_size)  
print("Test data size:", test_ds_size)  
print("Validation data size:", validation_ds_size)
```

For our basic input/data , we will do three primary operations.

1. Preprocessing the data within the dataset
2. Shuffle the dataset
3. Batch data within the dataset

These what we have done after preprocessing that is shuffling and batching the data are attached in the below picture.

The screenshot shows a Google Colab notebook titled 'VR_mini_project_1st_a_relu.ipynb'. The code implements a data pipeline for three datasets: train, test, and validation. It uses TensorFlow's Dataset API to map images, shuffle them, and batch them into 32-element batches. A note indicates that the code represents the Keras implementation of the AlexNet CNN architecture, using ReLU activation for most layers and softmax for the final layer.

```
##### For our basic input/data pipeline, we will conduct three primary operations which are here:-  
##### 1. Preprocessing the data within the dataset  
##### 2 Shuffle the dataset  
##### 3. Batch data within the dataset  
  
train_ds = (train_ds  
            .map(process_images)  
            .shuffle(buffer_size=train_ds_size)  
            .batch(batch_size=32, drop_remainder=True))  
  
test_ds = (test_ds  
            .map(process_images)  
            .shuffle(buffer_size=train_ds_size)  
            .batch(batch_size=32, drop_remainder=True))  
  
validation_ds = (validation_ds  
            .map(process_images)  
            .shuffle(buffer_size=train_ds_size)  
            .batch(batch_size=32, drop_remainder=True))  
  
## now this code represents the Keras implementation of the AlexNet CNN architecture:-  
## we have used relu activation function here and last layer as softmax activation function...
```

Model Implementation:-

Here are the types of layers the **AlexNet CNN** architecture is composed of, along with a brief description:-

Convolutional layer: A convolution is a mathematical term that describes a dot product multiplication between two sets of elements. Within deep learning the convolution operation acts on the filters/kernels and image data array within the convolutional layer. Therefore a convolutional layer is simply a layer that houses the convolution operation that occurs between the filters and the images passed through a convolutional neural network.

Batch Normalisation layer: Batch Normalization is a technique that mitigates the effect of unstable *gradients*

within a neural network through the introduction of an additional layer that performs operations on the inputs from the previous layer. The operations standardize and normalize the input values, after that the input values are transformed through scaling and shifting operations.

MaxPooling layer: Max pooling is a variant of sub-sampling where the maximum pixel value of pixels that fall within the receptive field of a unit within a sub-sampling layer is taken as the output. The max-pooling operation below has a window of 2x2 and slides across the input data, outputting an average of the pixels within the receptive field of the kernel.

Flatten layer: Takes an input shape and flattens the input image data into a one-dimensional array.

Dense Layer: A dense layer has an embedded number of arbitrary units/neurons within. Each neuron is a perceptron.

The code picture represents the Keras implementation of the AlexNet CNN architecture:-

A screenshot of a Google Colab notebook titled "VR_mini_project_1st_a_relu.ipynb". The code cell contains the definition of a Sequential model for AlexNet. The model consists of two main stages of convolutional layers followed by three fully connected layers at the end.

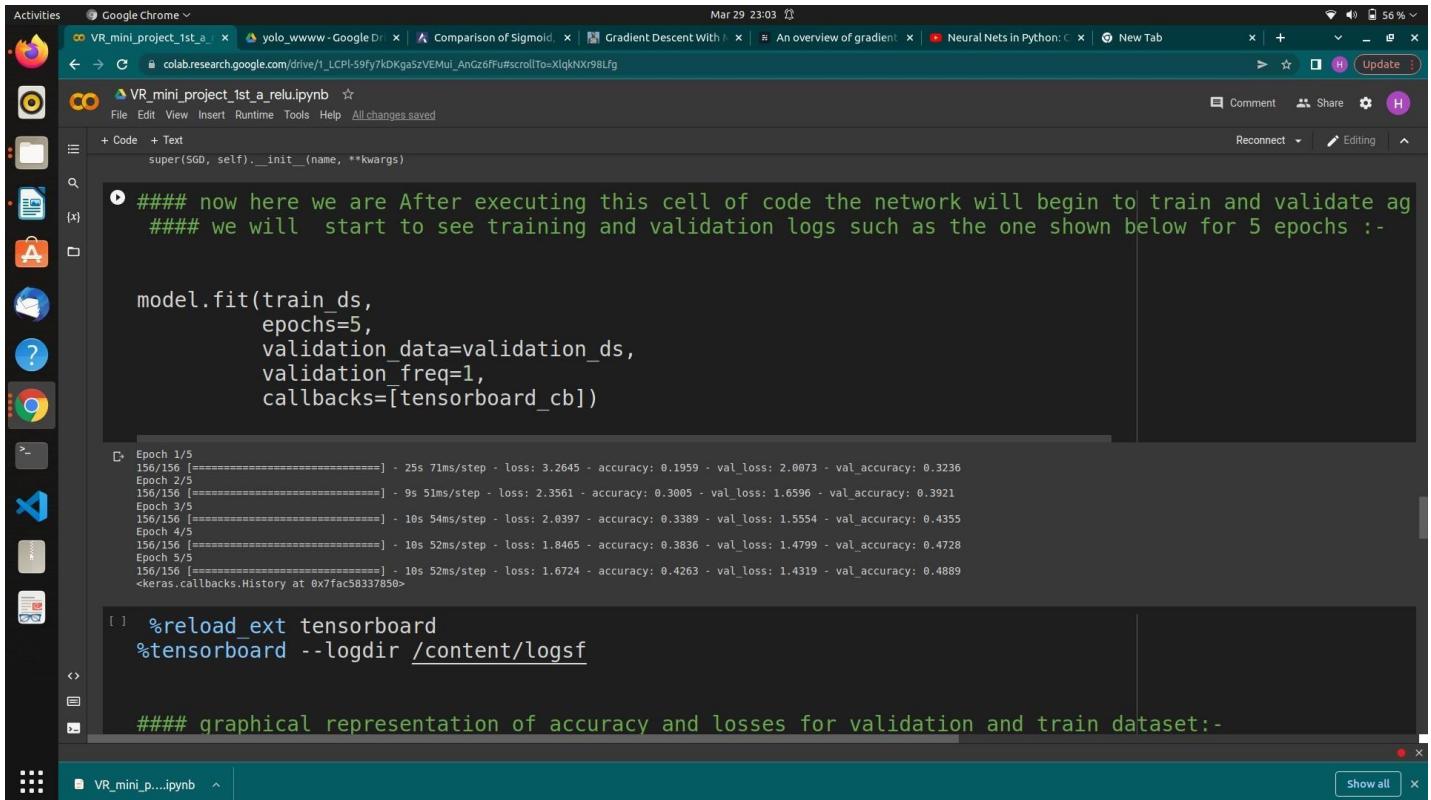
```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='relu', input_shape=(227, 227, 3)),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Flatten(),
    keras.layers.Dense(4096, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(4096, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation='softmax')
])
```

Here are the model summary visualization for more insight about all the layers of AlexNet CNN architecture:-

A screenshot of a Google Colab notebook titled "VR_mini_project_1st_a_relu.ipynb". The code cell displays the model summary, which provides detailed information about each layer's type, output shape, and parameter count.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 55, 55, 96)	34944
batch normalization (BatchNormaliza	(None, 55, 55, 96)	384
max pooling2d (MaxPooling2D)	(None, 27, 27, 96)	0
conv2d_1 (Conv2D)	(None, 27, 27, 256)	614656
batch normalization_1 (BatchN	(None, 27, 27, 256)	1024
max pooling2d_1 (MaxPooling2D)	(None, 13, 13, 256)	0
conv2d_2 (Conv2D)	(None, 13, 13, 384)	885120
batch normalization_2 (BatchN	(None, 13, 13, 384)	1536
conv2d_3 (Conv2D)	(None, 13, 13, 384)	1327488
batch normalization_3 (BatchN	(None, 13, 13, 384)	1536
conv2d_4 (Conv2D)	(None, 13, 13, 256)	884992
batch normalization_4 (BatchN	(None, 13, 13, 256)	1024
max pooling2d_2 (MaxPooling2D)	(None, 6, 6, 256)	0
flatten (Flatten)	(None, 9216)	0
dense (Dense)	(None, 4096)	37752832
dropout (Dropout)	(None, 4096)	0
dense_1 (Dense)	(None, 4096)	16781312

Now here we are **training the AlexNet** on CIFAR-10 dataset for 5 epochs only we haven't taken much epochs as it was taking more time and also GPU access of colab is limited:-



The screenshot shows a Google Colab notebook titled "VR_mini_project_1st_a_relu.ipynb". The code cell contains Python code for training a neural network using the SGD optimizer with ReLU activation. It includes a note about tensorboard callbacks and a log output showing training progress for 5 epochs. The output pane shows the command to reload tensorboard and a note about graphical representation.

```
super(SGD, self).__init__(name, **kwargs)

##### now here we are After executing this cell of code the network will begin to train and validate ag
##### we will start to see training and validation logs such as the one shown below for 5 epochs :-

model.fit(train_ds,
            epochs=5,
            validation_data=validation_ds,
            validation_freq=1,
            callbacks=[tensorboard_cb])

Epoch 1/5
156/156 [=====] - 25s 71ms/step - loss: 3.2645 - accuracy: 0.1959 - val_loss: 2.0073 - val_accuracy: 0.3236
Epoch 2/5
156/156 [=====] - 9s 51ms/step - loss: 2.3561 - accuracy: 0.3005 - val_loss: 1.6596 - val_accuracy: 0.3921
Epoch 3/5
156/156 [=====] - 10s 54ms/step - loss: 2.0397 - accuracy: 0.3389 - val_loss: 1.5554 - val_accuracy: 0.4355
Epoch 4/5
156/156 [=====] - 10s 52ms/step - loss: 1.8465 - accuracy: 0.3836 - val_loss: 1.4799 - val_accuracy: 0.4728
Epoch 5/5
156/156 [=====] - 10s 52ms/step - loss: 1.6724 - accuracy: 0.4263 - val_loss: 1.4319 - val_accuracy: 0.4889
<keras.callbacks.History at 0x7fac58337850>

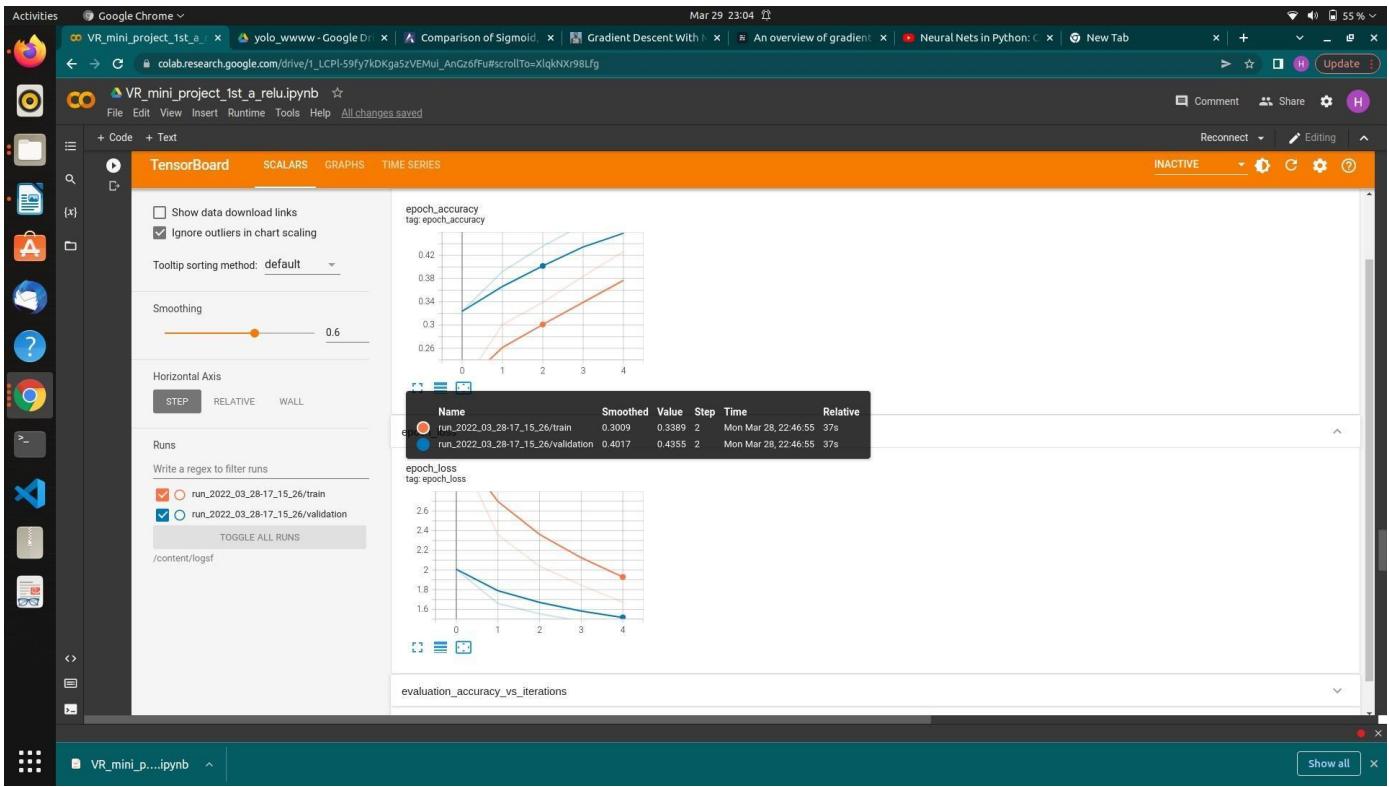
[1]: %reload_ext tensorboard
      tensorboard --logdir /content/logsfs

##### graphical representation of accuracy and losses for validation and train dataset:-
```

Here we have used **ReLU** as activation function and **SGD** optimization algorithm and using **Learning rate =0.001** for calculating metrics that is loss and accuracy.

Now here we are using **TENSORBOARD** which is a tool that provides a suite of visualization and monitoring the metrics on

train and validation datasets. For the work we'll be utilizing TensorBoard to monitor the progress of the training of the model on the datasets.



Now we are evaluating our trained model on test datasets:-

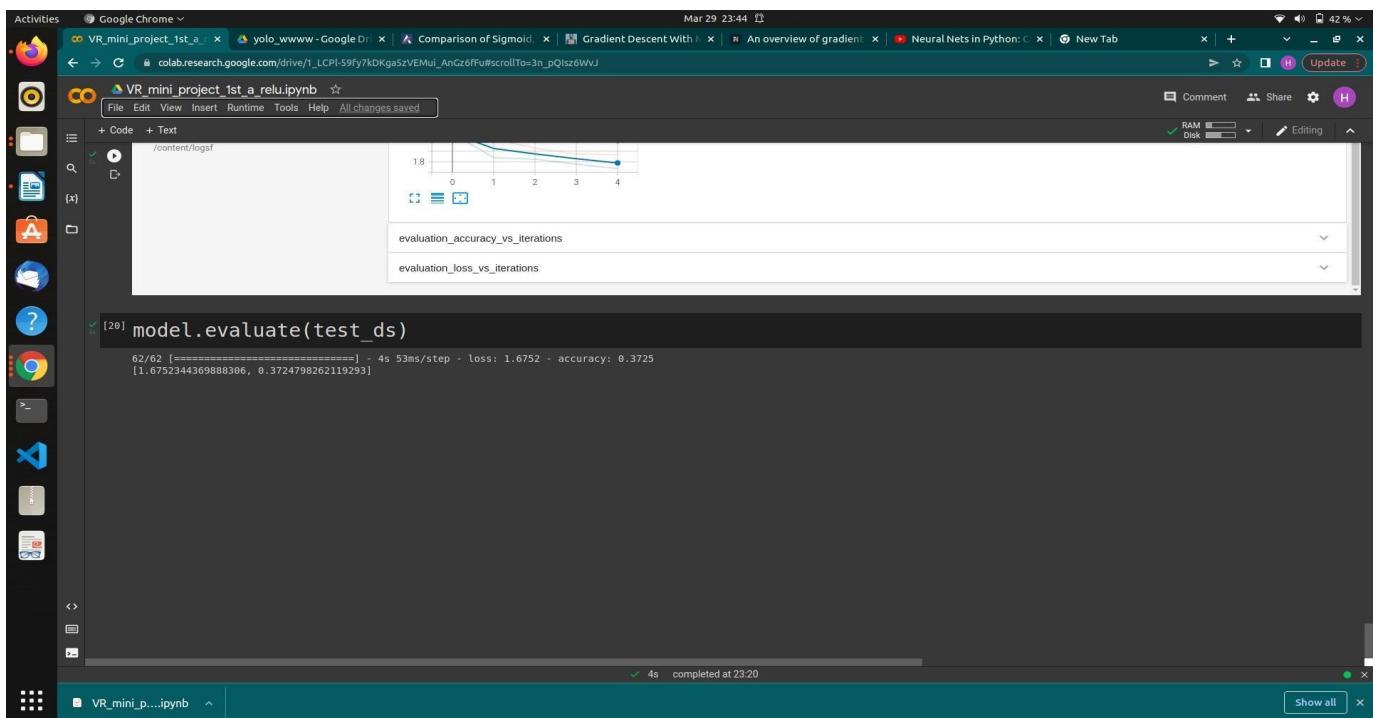
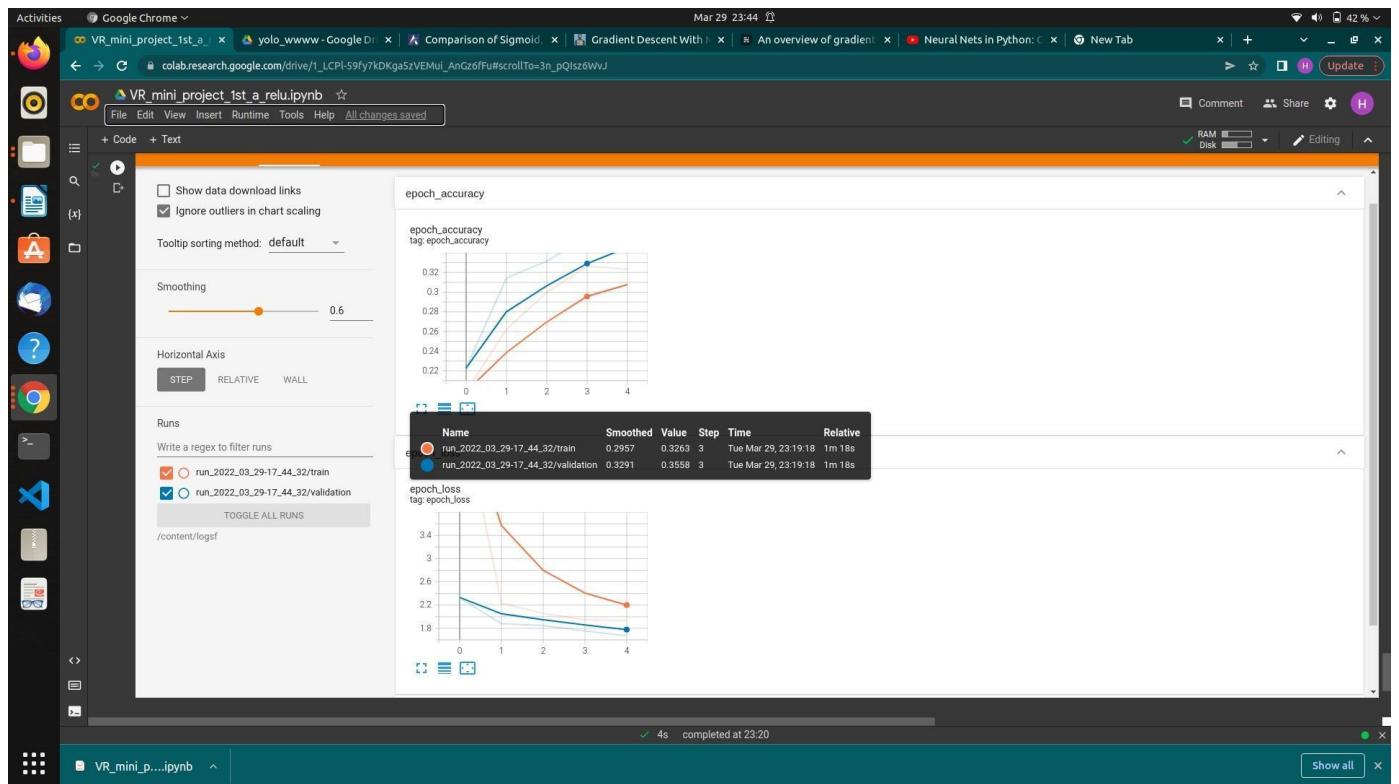
```
model.evaluate(test_ds) ##### evaluating the performance on test dataset
62/62 [=====] - 3s 28ms/step - loss: 1.4737 - accuracy: 0.4909
[1.4736825227737427, 0.49092742800712585]

BY using ADAM optimizer:
here we have measuring all the metrics and showcasing through TENSORBOARD

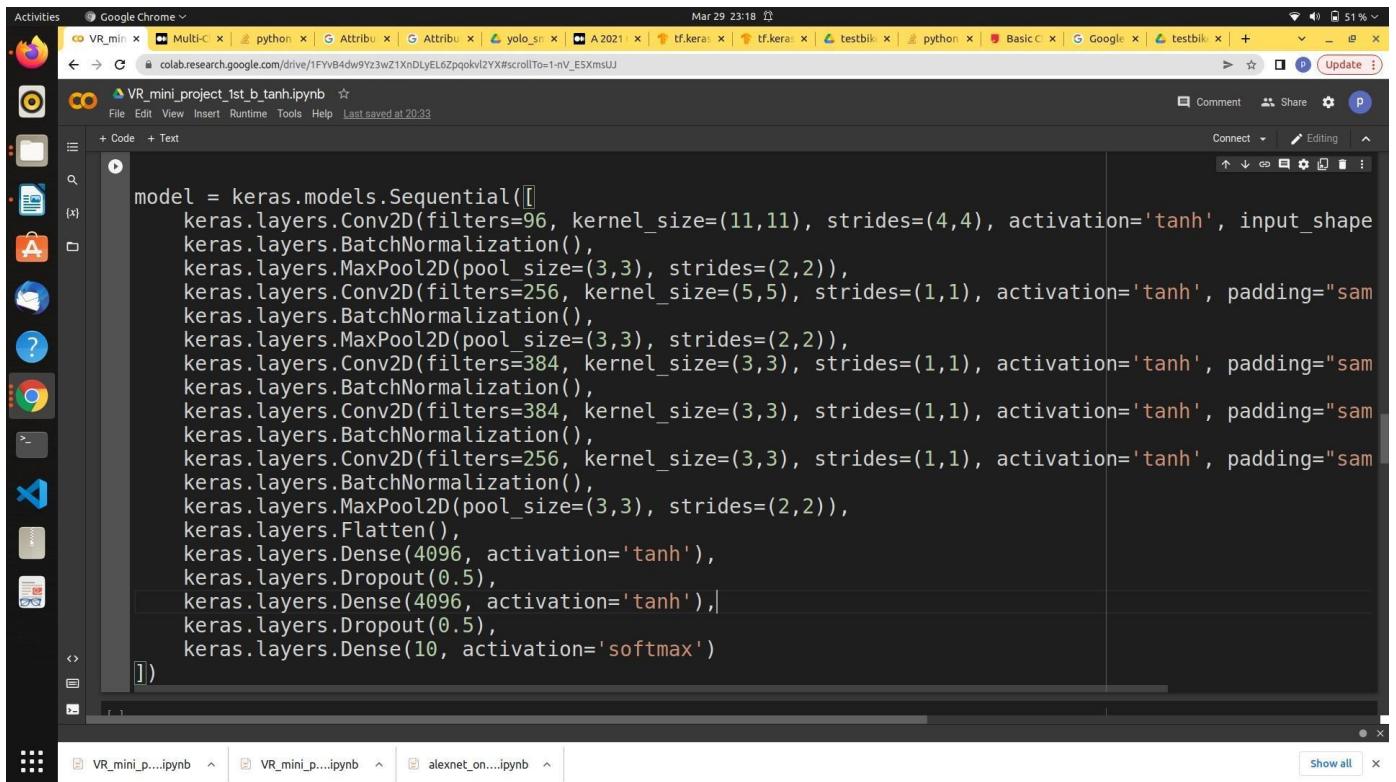
model.compile(loss='sparse_categorical_crossentropy',
              optimizer=tf.optimizers.Adam(lr=0.001), metrics=['accuracy'])
model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_15 (Conv2D)	(None, 55, 55, 96)	34944
batch_normalization_15 (BatchNormalization)	(None, 55, 55, 96)	384
max_pooling2d_9 (MaxPooling)	(None, 27, 27, 96)	0

Now metrics visualization and prediction of loss and accuracy on test dataset by using Adam optimizer and with same learning rate as 0.001:-



By using tanh as activation function and we are using softmax activation function for last layer

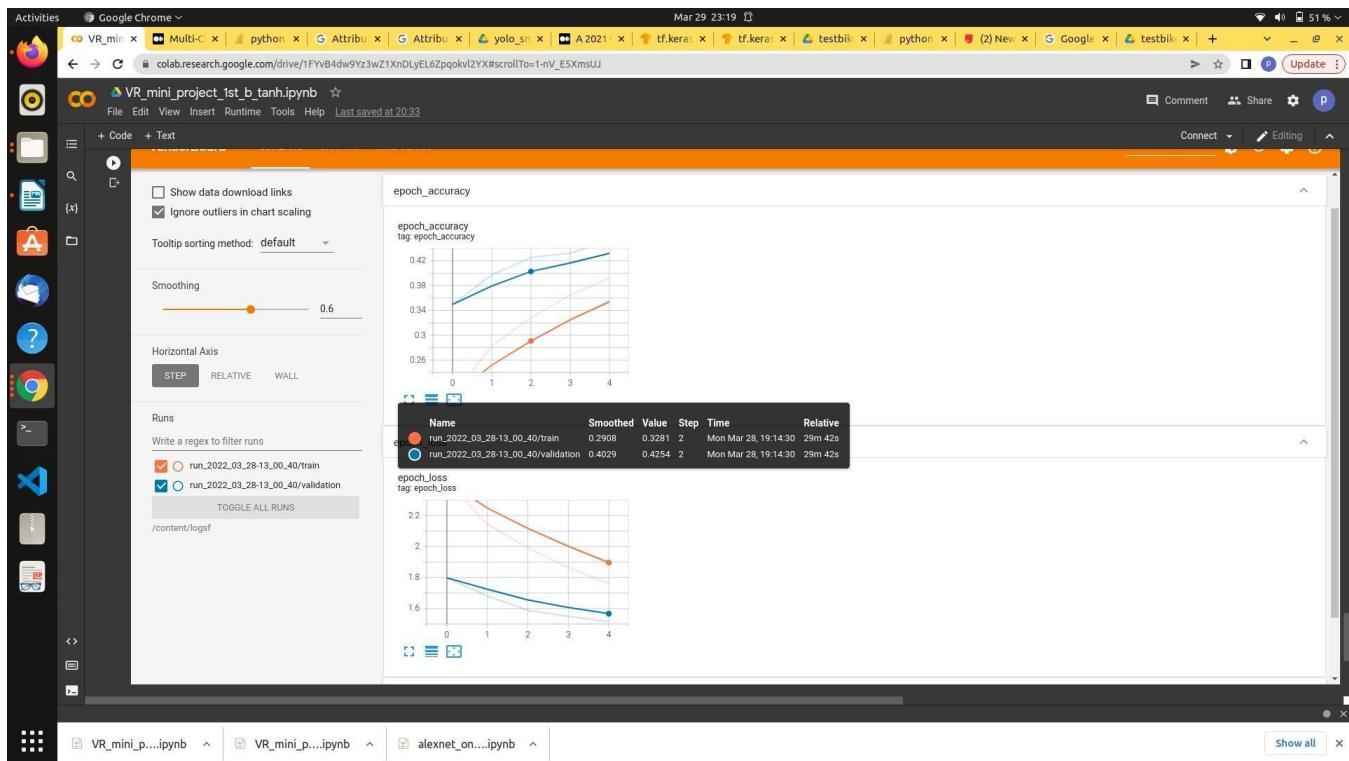


The screenshot shows a Google Colab notebook titled "VR_mini_project_1st_b_tanh.ipynb". The code defines a Sequential model with the following layers:

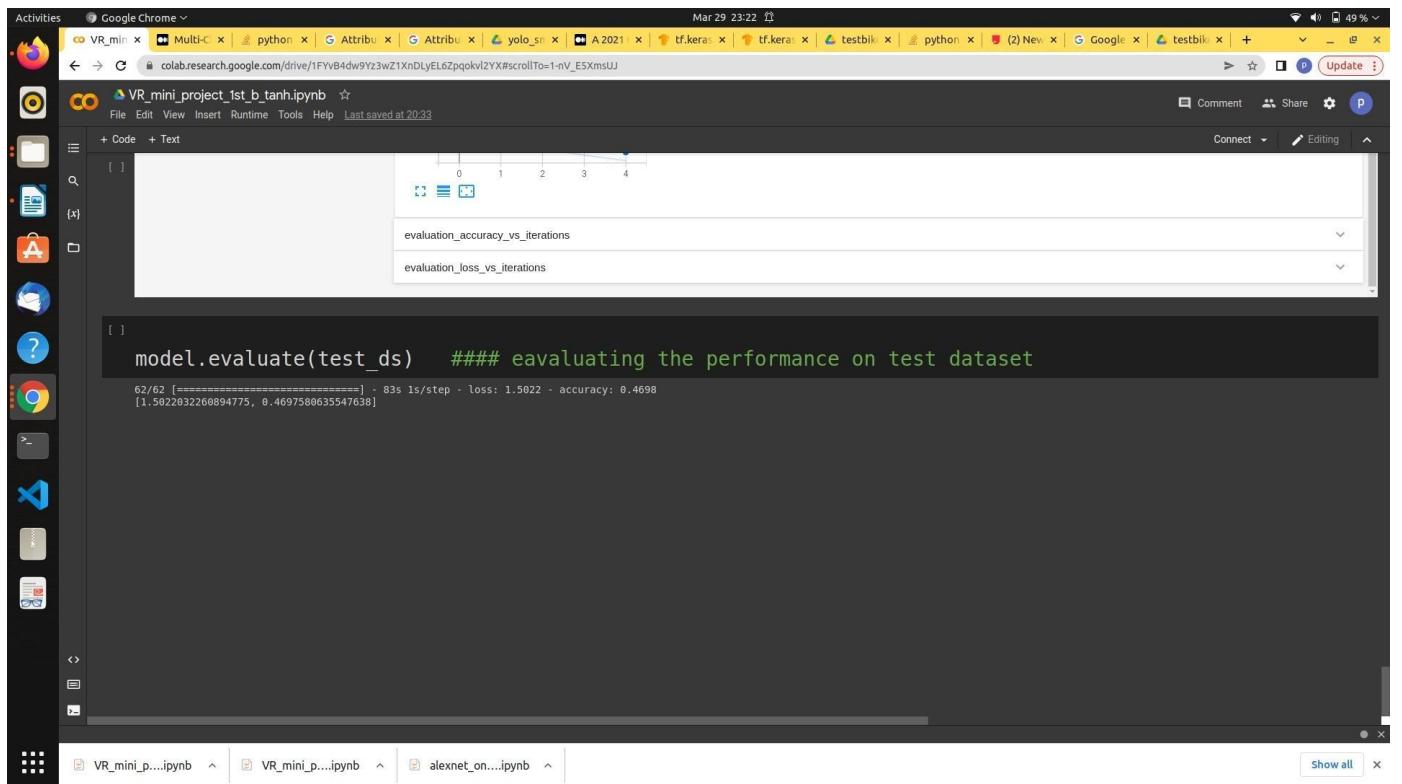
```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='tanh', input_shape=),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='tanh', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='tanh', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='tanh', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), activation='tanh', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Flatten(),
    keras.layers.Dense(4096, activation='tanh'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(4096, activation='tanh'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation='softmax')
])
```

The code uses the tanh activation function for most layers and softmax for the final layer. The model is defined with an input shape and various layers including convolutional, pooling, normalization, and fully connected layers.

Loss and Accuracy graph using Tensorboard:-



Evaluating the performance on test datasets:-



The screenshot shows a Google Colab notebook titled "VR_mini_project_1st_b_tanh.ipynb". The code cell contains the following Python code:

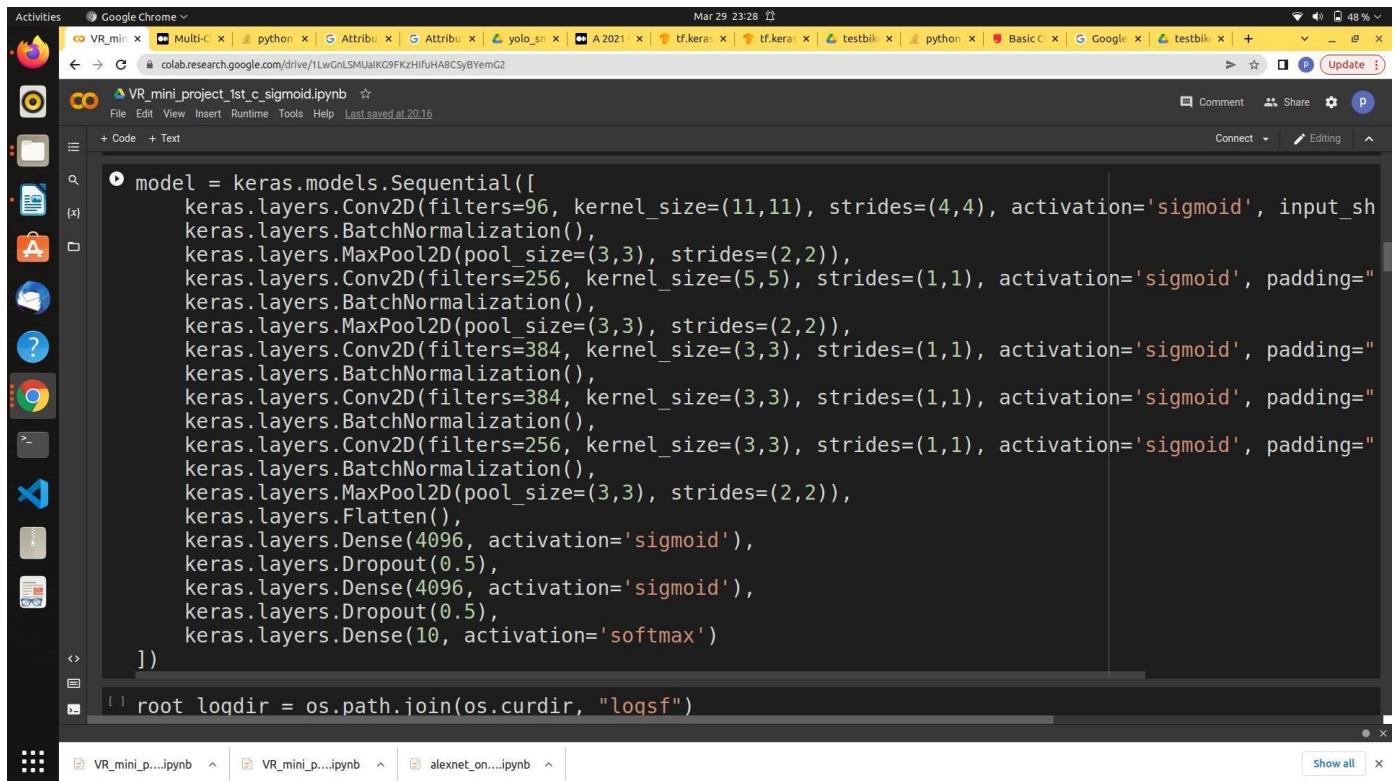
```
model.evaluate(test_ds) ##### evaluating the performance on test dataset
```

The output of the code cell is:

```
62/62 [=====] - 83s 1s/step - loss: 1.5022 - accuracy: 0.4698  
[1.5022032266894775, 0.4697580635547638]
```

The notebook sidebar shows other open files: "VR_mini_p...ipynb", "VR_mini_p...ipynb", and "alexnet_on...ipynb". The bottom right corner has a "Show all" button.

By using Sigmoid as activation function and we are using softmax activation function for last layer:-



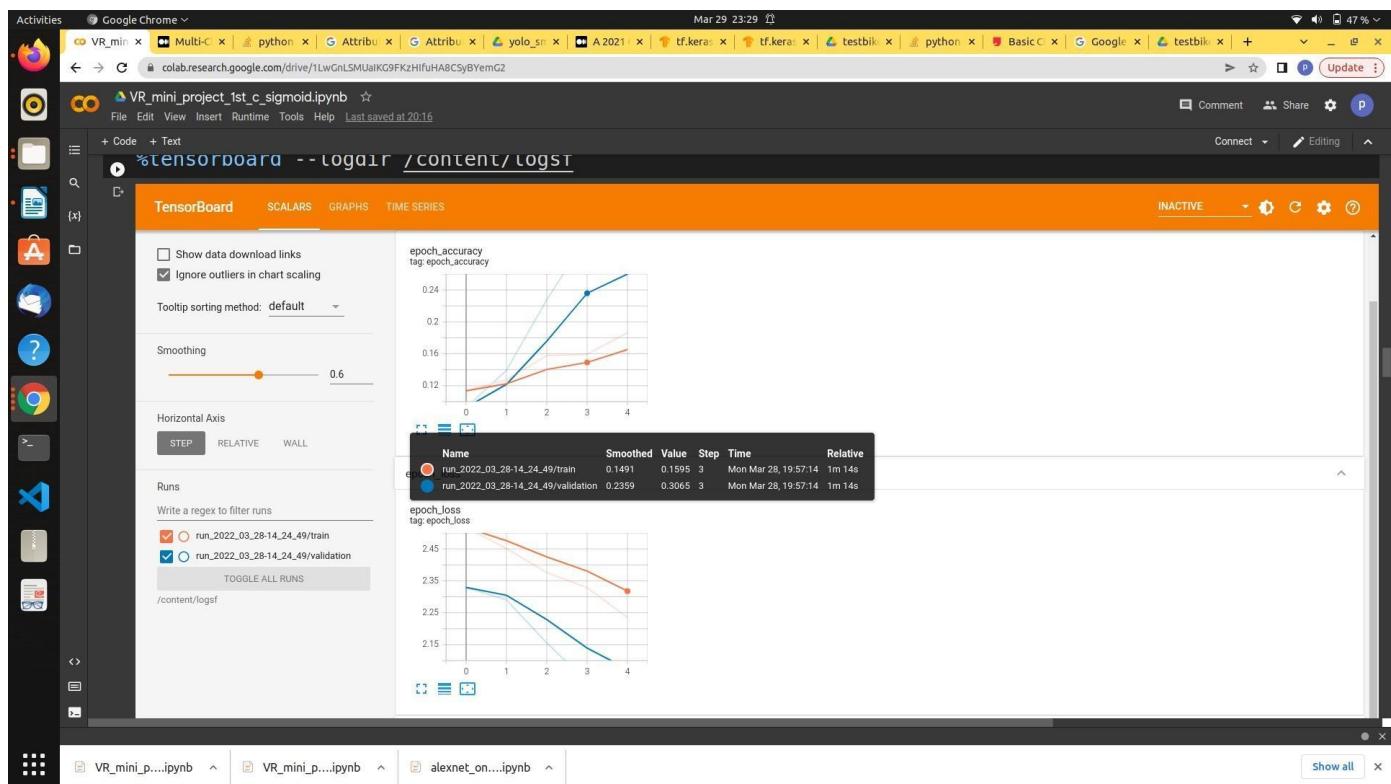
The screenshot shows a Google Colab notebook titled "VR_mini_project_1st_c_sigmoid.ipynb". The code defines a Sequential model from Keras:

```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='sigmoid', input_shape=(224,224,3)),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='sigmoid', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='sigmoid', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='sigmoid', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), activation='sigmoid', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Flatten(),
    keras.layers.Dense(4096, activation='sigmoid'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(4096, activation='sigmoid'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation='softmax')
])
```

Below the model definition, there is a cell with the following code:

```
root_logdir = os.path.join(os.curdir, "logsf")
```

Loss and Accuracy graph using Tensorboard:-



Evaluating the performance on test datasets:-

Activities Mar 29 23:30 47 %

Google Chrome VR_mini.ipynb Multi-C python x G Attribu x G Attribu x yolo_sn x A 2021 x tf.keras x tf.keras x testbilk x python x Basic C x Google x testbilk x + Update

File Edit View Insert Runtime Tools Help Last saved at 20:16

Comment Share Connect Editing

+ Code + Text

```
[ ] evaluation_accuracy_vs_iterations
[ ] evaluation_loss_vs_iterations
```

[] model.evaluate(test_ds)

```
62/62 [=====] - 4s 56ms/step - loss: 1.9966 - accuracy: 0.3004
[1.9966472387313843, 0.3004032373428345]
```

- lets do by different optimizer and by not assigning any learning rates:-

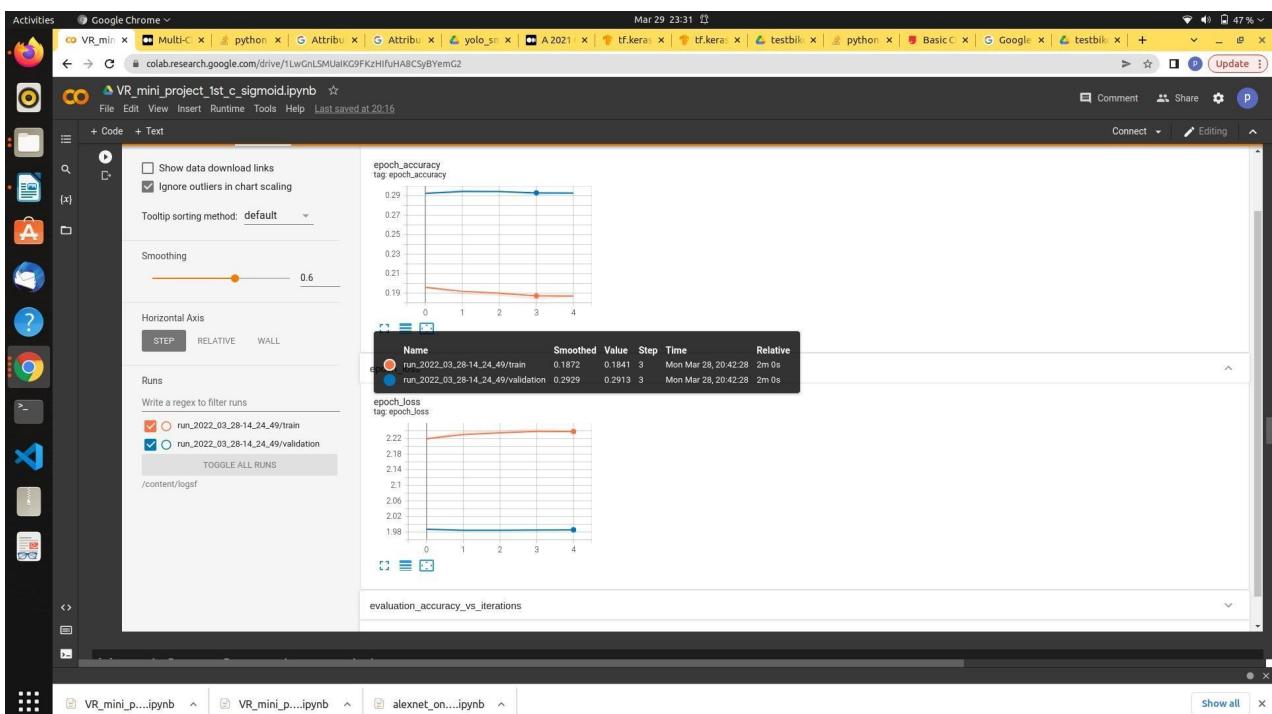
```
[ ] model.compile(loss='sparse_categorical_crossentropy',
                 optimizer=tf.optimizers.SGD(lr=0.00), metrics=['accuracy'])
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 55, 55, 96)	34944
batch normalization (BatchN ormalization)	(None, 55, 55, 96)	384
max_pooling2d (MaxPooling2D)	(None, 27, 27, 96)	0
conv2d_1 (Conv2D)	(None, 27, 27, 256)	614656
batch normalization_1 (BatchN ormalization)	(None, 27, 27, 256)	1024
max_pooling2d_1 (MaxPooling2D)	(None, 13, 13, 256)	0

VR_mini_p...ipynb VR_mini_p...ipynb alexnet_on...ipynb Show all

Using SGD optimizer without any learning rates the graph of loss and accuracy on train and validation datasets :-



Visualization of accuracy and loss on test datasets:-

The screenshot shows a Google Colab notebook titled "VR_mini_project_1st_c_sigmoid.ipynb". The code cell contains the following commands:

```
model.evaluate(test_ds)
model.compile(loss='sparse_categorical_crossentropy',
              optimizer=tf.optimizers.Adam(lr=0.001), metrics=['accuracy'])
model.summary()
```

Output from the first command:

```
62/62 [=====] - 5s 61ms/step - loss: 1.9945 - accuracy: 0.3039
[1.9944742918014526, 0.30393144488334656]
```

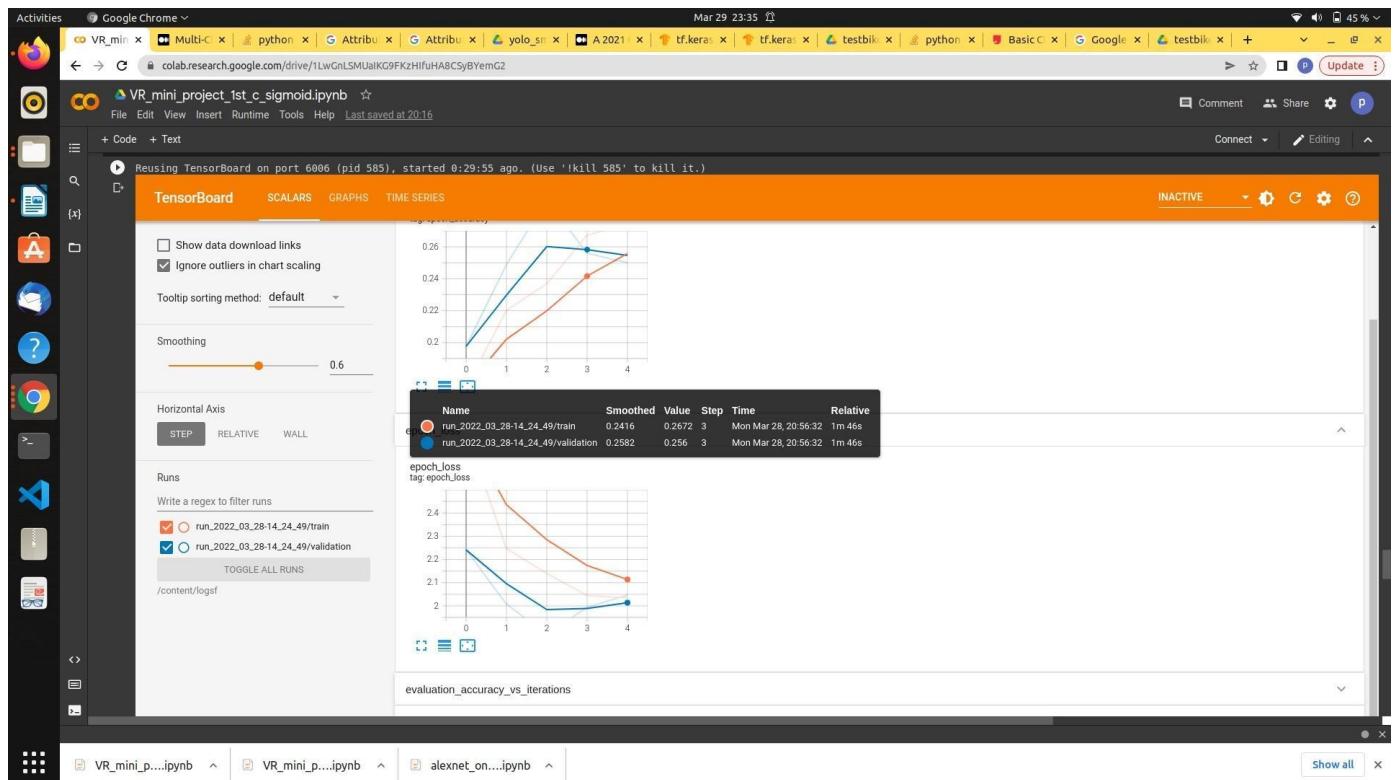
Double-click (or enter) to edit

Output from the second command:

```
Model: "sequential"
Layer (type)      Output Shape        Param #
conv2d (Conv2D)    (None, 55, 55, 96)   34944
batch normalization (BatchN (None, 55, 55, 96)   384
ormalization)
max_pooling2d (MaxPooling2D (None, 27, 27, 96)   0
```

At the top, there are two plots: "evaluation_accuracy_vs_iterations" and "evaluation_loss_vs_iterations".

By using Adam optimizer and Learning rate as .001 without momentum:-



Visualization of accuracy and loss on test datasets:-

The screenshot shows a Google Colab notebook titled "VR_mini_project_1st_c_sigmoid.ipynb". The code cell contains the following Python code:

```
[1]: model.evaluate(test_ds)
62/62 [=====] - 6s 69ms/step - loss: 2.0974 - accuracy: 0.2505
[2.0974364280700684, 0.2505940466785431]

● model.compile(loss='sparse_categorical_crossentropy',
                 optimizer=tf.keras.optimizers.SGD(
                     learning_rate=0.01,
                     momentum=0.9,
                     nesterov=False,
                     name='SGD',


                ))
model.summary()
```

Output from the first cell:

```
Model: "sequential"
```

Annotations on the right side of the code cell:

here we have used with momentum and ab
by varying the learning rate and putti

The Colab interface includes a sidebar with various icons and a bottom navigation bar with tabs for other notebooks.

By using SGD optimizer and Learning rate as .001 with momentum:-

Activities Google Chrome Mar 29 23:38 44% ▾

VR_min x Multi-C x python x G Attribu x G Attribu x yolo_sr x A 2021! x tf.keras x tf.keras x testbik x python x (2) New x Google x testbik x +

colab.research.google.com/drive/1LwGnLSMuAlK9FKzHifuHABCsyByemG2

VR_mini_project_1st_c_sigmoid.ipynb

File Edit View Insert Runtime Tools Help Last saved at 20:16

Connect Editing

+ Code + Text

```
[ ] model.evaluate(test_ds)
62/62 [=====] - 6s 69ms/step - loss: 2.0974 - accuracy: 0.2505
[2.0974364280709684, 0.2505040466785431]

● model.compile(loss='sparse_categorical_crossentropy',
                 optimizer=tf.keras.optimizers.SGD(
                     learning_rate=0.01,
                     momentum=0.9,
                     nesterov=False,
                     name='SGD',
                 ))
```

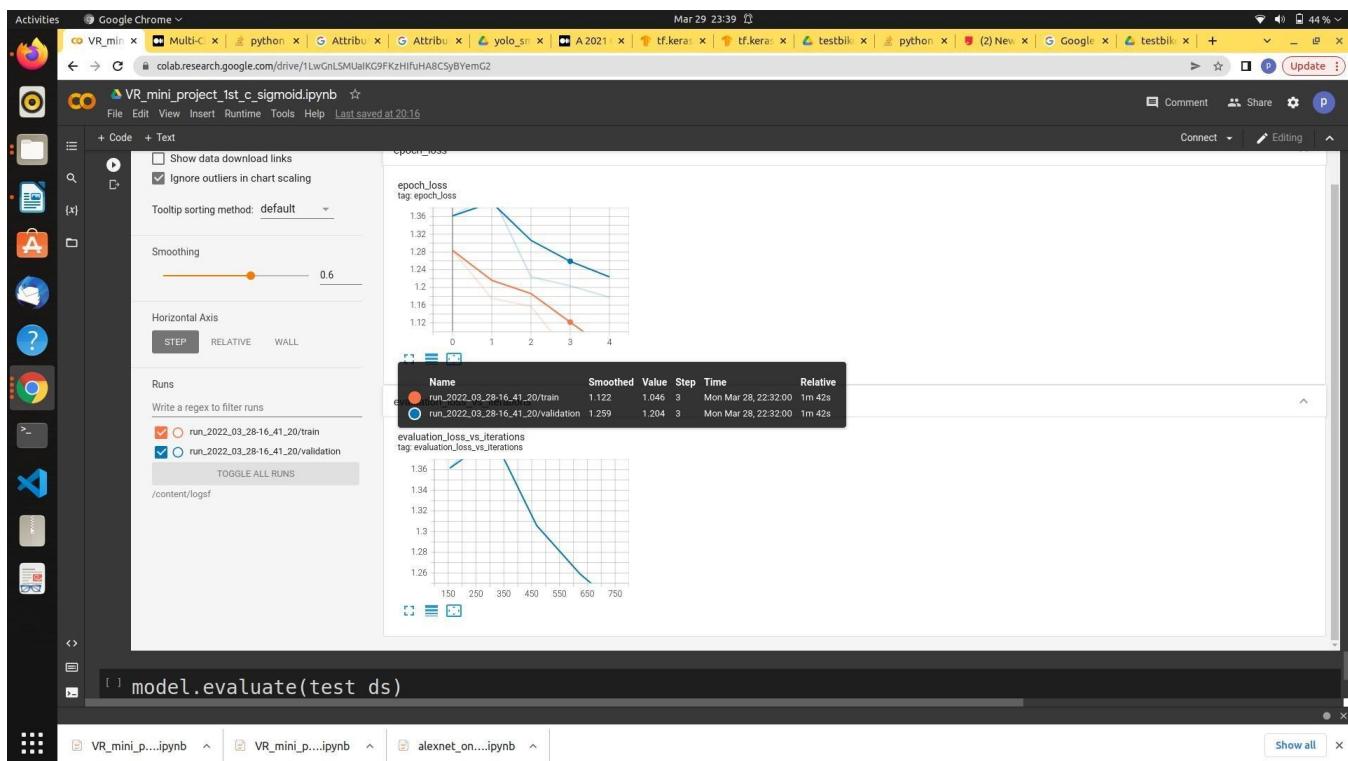
model.summary()

Model: "sequential"

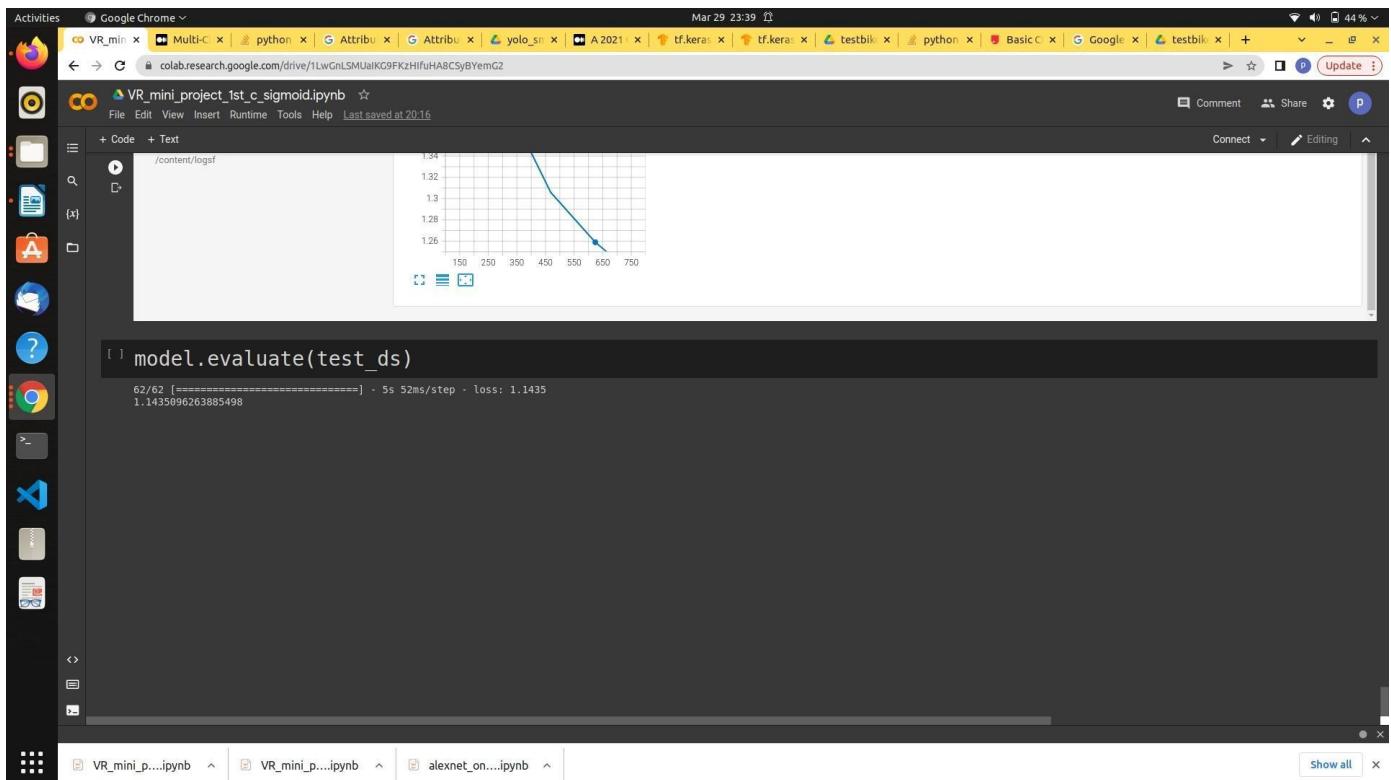
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 55, 55, 96)	34944
batch normalization (BatchN	(None, 55, 55, 96)	384
ormalization)		
max_pooling2d (MaxPooling2D	(None, 27, 27, 96)	0
)		
conv2d_1 (Conv2D)	(None, 27, 27, 256)	614656

here we have used with momentum and ab
by varying the learning rate and putti

VR_mini_p...ipynb ^ VR_mini_p...ipynb ^ alexnet_on...ipynb ^ Show all x



Visualization of accuracy and loss on test datasets:-



Report of Assignment 3b

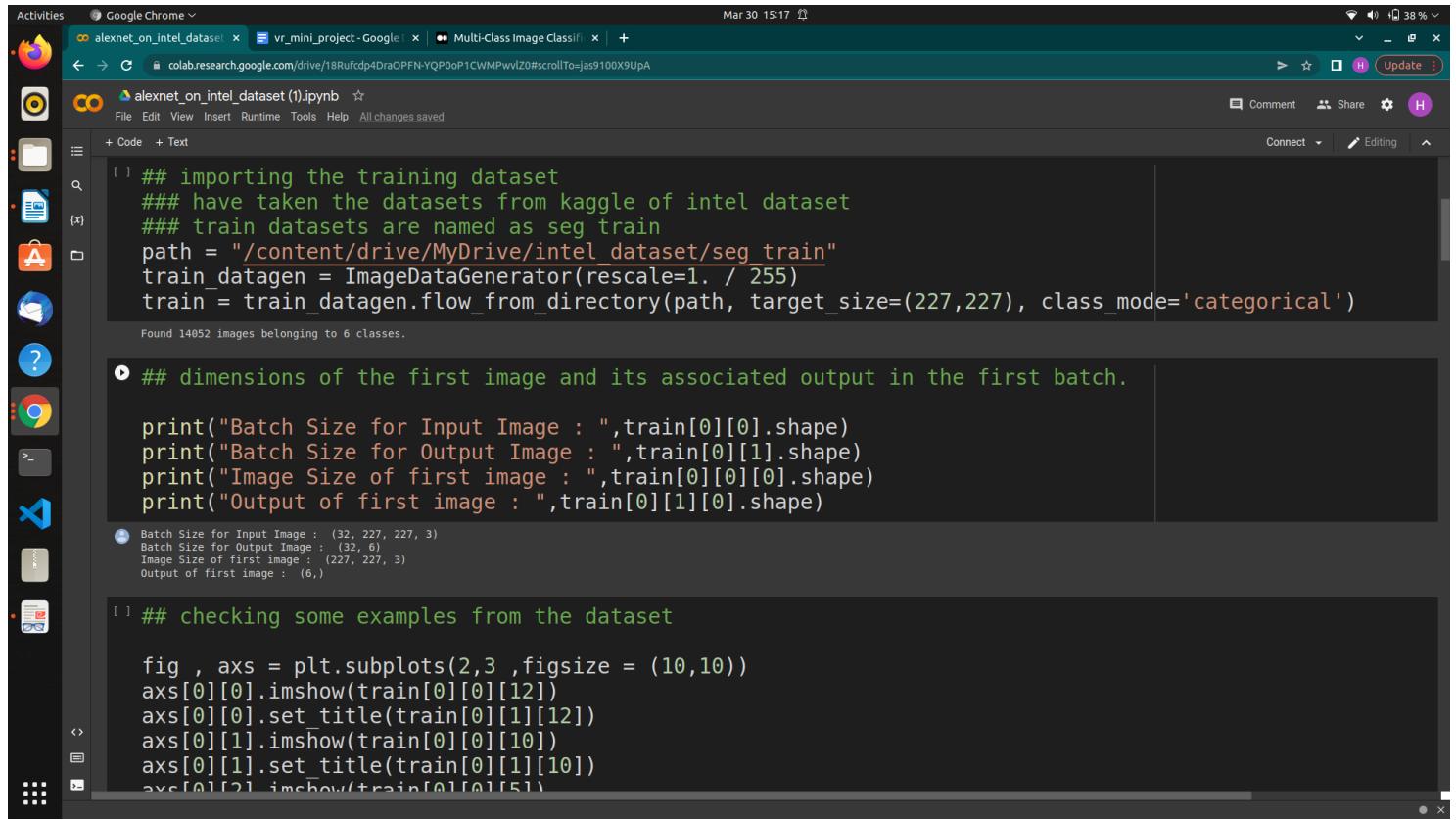
AlexNet implementation on Intel Datasets using Tensorflow and keras:-

Keras is an API for python, built over Tensorflow 2.0, which is scalable and adapt to deployment capabilities of Tensorflow.

we are using Intel Datasets taken from kaggle:-

This Data contains around 25k images of size 150x150 distributed under 6 categories, namely : ‘**buildings**’ , ‘**forest**’ , ‘**glacier**’ , ‘**mountain**’ , ‘**sea**’ , ‘**street**’ . There are 14K images in training set, 3K in validation set and 7K in the Prediction set.

we will import the dataset that is training dataset named as seg_train now as shown below :-



A screenshot of a Google Colab notebook titled "alexnet_on_intel_dataset (1).ipynb". The code cell contains Python code for importing a training dataset from a local directory. It prints the dimensions of the first image and its associated output in the first batch. The output shows the shapes: (32, 227, 227, 3) for the input image, (32, 6) for the output image, and (227, 227, 3) for the image size. Another cell is shown below, starting with "# checking some examples from the dataset".

```
[ ] ## importing the training dataset
    ### have taken the datasets from kaggle of intel dataset
    ### train datasets are named as seg_train
path = "/content/drive/MyDrive/intel_dataset/seg_train"
train_datagen = ImageDataGenerator(rescale=1. / 255)
train = train_datagen.flow_from_directory(path, target_size=(227,227), class_mode='categorical')

Found 14052 images belonging to 6 classes.

● ## dimensions of the first image and its associated output in the first batch.

print("Batch Size for Input Image : ",train[0][0].shape)
print("Batch Size for Output Image : ",train[0][1].shape)
print("Image Size of first image : ",train[0][0][0].shape)
print("Output of first image : ",train[0][1][0].shape)

Batch Size for Input Image : (32, 227, 227, 3)
Batch Size for Output Image : (32, 6)
Image Size of first image : (227, 227, 3)
Output of first image : (6,)

[ ] ## checking some examples from the dataset

fig , axs = plt.subplots(2,3 ,figsize = (10,10))
axs[0][0].imshow(train[0][0][12])
axs[0][0].set_title(train[0][1][12])
axs[0][1].imshow(train[0][0][10])
axs[0][1].set_title(train[0][1][10])
axs[0][2].imshow(train[0][0][5])

```

Now as a part of preprocessing , the input size for AlexNet is 227x227x3 and so we will change the target size to (227,227). The by default Batch Size is 32. Lets see the type of train and train_datagen.

Next let us check the dimensions of the first image and its associated output in the first batch which is shown above in the screenshots.

check out some Examples from our Dataset :

The screenshot shows a Jupyter Notebook interface in Google Colab. The notebook is titled "alexnet_on_intel_dataset (1).ipynb". The code cell contains the following Python code:

```
axs[1][0].imsave('train[0][0][20]')
axs[1][0].set_title('train[0][1][20]')
axs[1][1].imsave('train[0][0][25]')
axs[1][1].set_title('train[0][1][25]')
axs[1][2].imsave('train[0][0][3]')
axs[1][2].set_title('train[0][1][3]')
```

Below the code, there is a warning message:

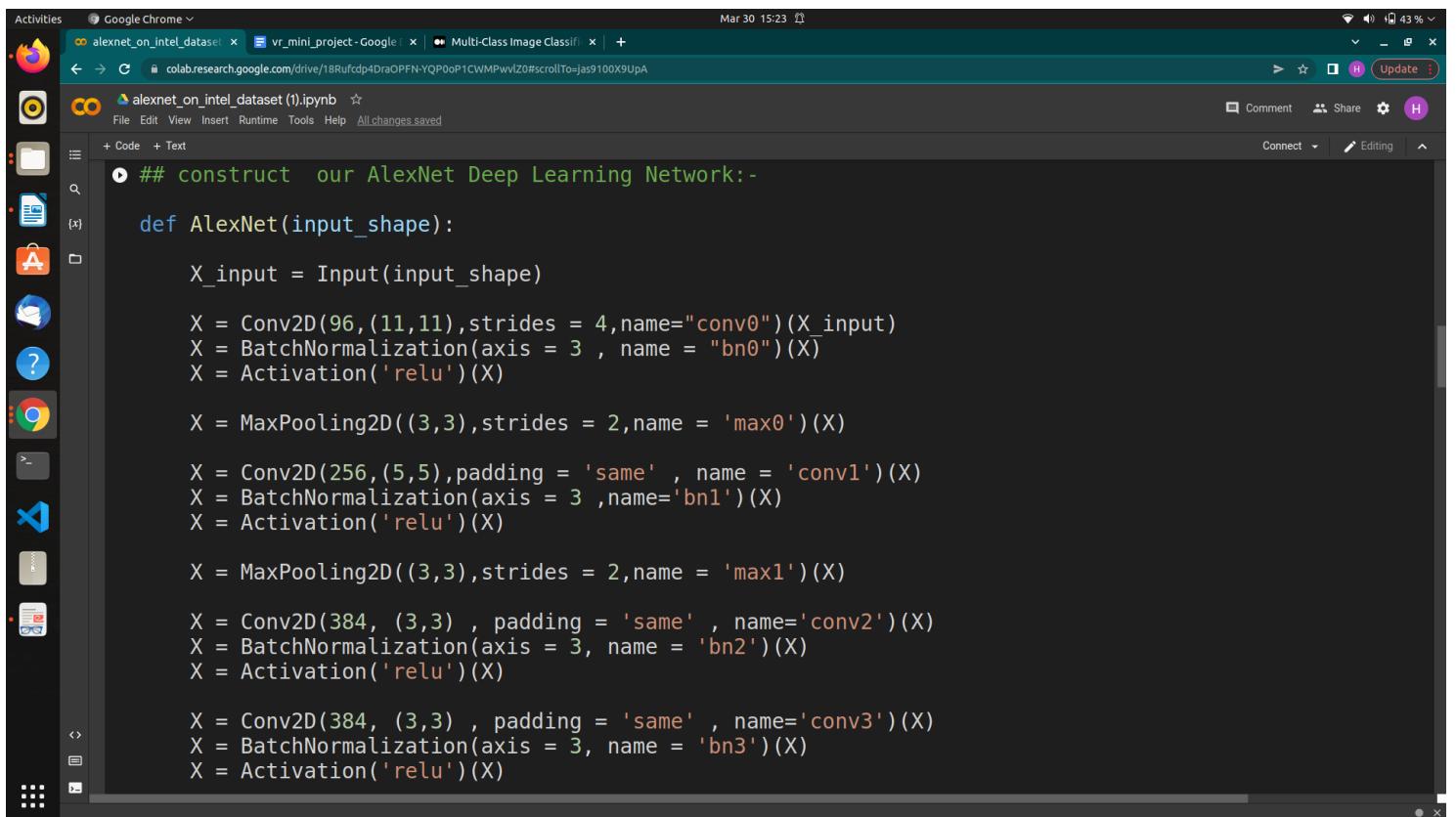
```
/usr/local/lib/python3.7/dist-packages/matplotlib/text.py:1165: FutureWarning: elementwise comparison failed; returning scalar instead, but in the future will perform elementwise comparison
```

Following the warning, there are six subplots arranged in a 2x3 grid. Each subplot has a title and a corresponding numerical vector below it:

- Top-left: [1. 0. 0. 0. 0. 0.] (Buildings)
- Top-middle: [0. 0. 0. 0. 0. 1.] (Buildings)
- Top-right: [0. 0. 1. 0. 0. 0.] (Desert)
- Bottom-left: [0. 0. 0. 1. 0. 0.] (Landscape)
- Bottom-middle: [0. 0. 1. 0. 0. 0.] (Snowy mountain)
- Bottom-right: [1. 0. 0. 0. 0. 0.] (Dark textured surface)

The notebook interface includes a sidebar with various icons for file operations, a toolbar at the top, and a status bar at the bottom indicating the date and time.

Now let's start the construction of the Model. The following code will construct our AlexNet Deep Learning Network :



The screenshot shows a Google Colab notebook titled "alexnet_on_intel_dataset (1).ipynb". The code in the notebook constructs an AlexNet Deep Learning Network. The code uses TensorFlow's Keras API to define layers like Conv2D, MaxPooling2D, and Activation (ReLU), along with BatchNormalization. The notebook is running in a browser-based environment with various icons and tools visible on the left sidebar.

```
## construct our AlexNet Deep Learning Network:-  
  
def AlexNet(input_shape):  
    X_input = Input(input_shape)  
  
    X = Conv2D(96,(11,11),strides = 4,name="conv0")(X_input)  
    X = BatchNormalization(axis = 3 , name = "bn0")(X)  
    X = Activation('relu')(X)  
  
    X = MaxPooling2D((3,3),strides = 2,name = 'max0')(X)  
  
    X = Conv2D(256,(5,5),padding = 'same' , name = 'conv1')(X)  
    X = BatchNormalization(axis = 3 ,name='bn1')(X)  
    X = Activation('relu')(X)  
  
    X = MaxPooling2D((3,3),strides = 2,name = 'max1')(X)  
  
    X = Conv2D(384, (3,3) , padding = 'same' , name='conv2')(X)  
    X = BatchNormalization(axis = 3, name = 'bn2')(X)  
    X = Activation('relu')(X)  
  
    X = Conv2D(384, (3,3) , padding = 'same' , name='conv3')(X)  
    X = BatchNormalization(axis = 3, name = 'bn3')(X)  
    X = Activation('relu')(X)
```

we are showing the model summary here:-

```

## printing the summary of the ALEXNET model
alex.summary()

Model: "AlexNet"
+-----+-----+-----+
| Layer (type) | Output Shape | Param # |
+-----+-----+-----+
| input_1 (InputLayer) | [(None, 227, 227, 3)] | 0 |
| conv0 (Conv2D) | (None, 55, 55, 96) | 34944 |
| bn0 (BatchNormalization) | (None, 55, 55, 96) | 384 |
| activation (Activation) | (None, 55, 55, 96) | 0 |
| max0 (MaxPooling2D) | (None, 27, 27, 96) | 0 |
| conv1 (Conv2D) | (None, 27, 27, 256) | 614656 |
| bn1 (BatchNormalization) | (None, 27, 27, 256) | 1024 |
| activation_1 (Activation) | (None, 27, 27, 256) | 0 |
| max1 (MaxPooling2D) | (None, 13, 13, 256) | 0 |
| conv2 (Conv2D) | (None, 13, 13, 384) | 885120 |
| bn2 (BatchNormalization) | (None, 13, 13, 384) | 1536 |
| activation_2 (Activation) | (None, 13, 13, 384) | 0 |
| conv3 (Conv2D) | (None, 13, 13, 384) | 1327488 |
| bn3 (BatchNormalization) | (None, 13, 13, 384) | 1536 |
| activation_3 (Activation) | (None, 13, 13, 384) | 0 |
| conv4 (Conv2D) | (None, 13, 13, 256) | 884992 |
| bn4 (BatchNormalization) | (None, 13, 13, 256) | 1024 |
| activation_4 (Activation) | (None, 13, 13, 256) | 0 |
| max2 (MaxPooling2D) | (None, 6, 6, 256) | 0 |
| flatten (Flatten) | (None, 9216) | 0 |

```

Now we will compile the model using “adam” optimizer and choosing loss as categorical_crossentropy , with accuracy metrics.

```

[ ] fc0 (Dense) (None, 4096) 37752832
[ ] fc1 (Dense) (None, 4096) 16781312
[ ] fc2 (Dense) (None, 6) 24582
=====
Total params: 58,311,430
Trainable params: 58,308,678
Non-trainable params: 2,752

[ ] alex.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics=['accuracy'])

▼ training the model using fit_generator()

[ ] alex.fit_generator(train,epochs=10) ### we are using here 10 epochs
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning: 'Model.fit_generator' is deprecated and will be removed in a future version. Please use 'Model.fit', which supports generators.
...Entry point for launching an IPython kernel.
Epoch 1/10
440/440 [=====] - 2945s 7s/step - loss: 2.1825 - accuracy: 0.5186
Epoch 2/10
440/440 [=====] - 56s 126ms/step - loss: 0.9483 - accuracy: 0.6387
Epoch 3/10
440/440 [=====] - 65s 148ms/step - loss: 0.7608 - accuracy: 0.7242
Epoch 4/10
440/440 [=====] - 62s 140ms/step - loss: 0.6567 - accuracy: 0.7663
Epoch 5/10
440/440 [=====] - 61s 138ms/step - loss: 0.5830 - accuracy: 0.7943
Epoch 6/10
440/440 [=====] - 55s 124ms/step - loss: 0.5534 - accuracy: 0.8044
Epoch 7/10
440/440 [=====] - 56s 127ms/step - loss: 0.5017 - accuracy: 0.8237
Epoch 8/10
440/440 [=====] - 55s 126ms/step - loss: 0.4703 - accuracy: 0.8338
Epoch 9/10
440/440 [=====] - 55s 125ms/step - loss: 0.4568 - accuracy: 0.8397
Epoch 10/10
440/440 [=====] - 55s 126ms/step - loss: 0.4299 - accuracy: 0.8527
<keras.callbacks.History at 0x7f957009ab90>

```

Now we will train the model using fit_generator with the following command with 10 epochs :

```
alex.fit_generator(train,epochs=10) ### we are using here 10 epochs
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  """/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
Epoch 1/10
440/440 [=====] - 2945s 7s/step - loss: 2.1825 - accuracy: 0.5186
Epoch 2/10
440/440 [=====] - 56s 126ms/step - loss: 0.9483 - accuracy: 0.6387
Epoch 3/10
440/440 [=====] - 65s 148ms/step - loss: 0.7608 - accuracy: 0.7242
Epoch 4/10
440/440 [=====] - 62s 140ms/step - loss: 0.6567 - accuracy: 0.7663
Epoch 5/10
440/440 [=====] - 61s 138ms/step - loss: 0.5830 - accuracy: 0.7943
Epoch 6/10
440/440 [=====] - 55s 124ms/step - loss: 0.5534 - accuracy: 0.8044
Epoch 7/10
440/440 [=====] - 56s 127ms/step - loss: 0.5017 - accuracy: 0.8237
Epoch 8/10
440/440 [=====] - 55s 126ms/step - loss: 0.4703 - accuracy: 0.8338
Epoch 9/10
440/440 [=====] - 55s 125ms/step - loss: 0.4568 - accuracy: 0.8397
Epoch 10/10
440/440 [=====] - 55s 126ms/step - loss: 0.4299 - accuracy: 0.8527
<keras.callbacks.History at 0x7f957009ab90>

[1]: ### seg test are our validation datasets

path_test = "/content/drive/MyDrive/intel_dataset/seg_test"
test_datagen = ImageDataGenerator(rescale=1. / 255)
test = test_datagen.flow_from_directory(path_test, target_size=(227,227), class_mode='categorical')
```

and got final accuracy of 85% and losses of 42%

Now we will load validation data to get test accuracy :-

```
alexnet_on_intel_dataset | Multi-Class Image Classification | +  
File Edit View Insert Runtime Tools Help All changes saved  
Comment Share Editing  
+ Code + Text  
<keras.callbacks.History at 0x7f957009ab90>  
load test data to get test accuracy  
### seg test are our validation datasets  
path_test = "/content/drive/MyDrive/intel_dataset/seg_test"  
test_datagen = ImageDataGenerator(rescale=1. / 255)  
test = test_datagen.flow_from_directory(path_test, target_size=(227,227), class_mode='categorical')  
Found 3010 images belonging to 6 classes.  
## displaying accuracy and loss on the test data  
preds = alex.evaluate_generator(test)  
print ("Loss = " + str(preds[0]))  
print ("Test Accuracy = " + str(preds[1]))  
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning: 'Model.evaluate_generator' is deprecated and will be removed in a future version. Please use 'Model.evaluate', which supports ge  
""Entry point for launching an IPython kernel.  
Loss = 1.2943999767303467  
Test Accuracy = 0.6139534711837769  
## loading the prediction data that is "seg_pred" on which we have to predict :-  
path_pred = "/content/drive/MyDrive/intel_dataset/seg_pred"  
predict_datagen = ImageDataGenerator(rescale=1. / 255)  
predict = predict_datagen.flow_from_directory(path_pred, target_size=(227,227), batch_size = 1, class_m  
Found 7301 images belonging to 1 classes.
```

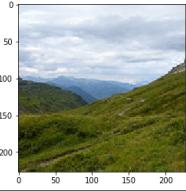
We got a test accuracy of 61.3% Now we will run the model over prediction Images to predict the image.

```
[ ] path_pred = "/content/drive/MyDrive/intel_dataset/seg_pred"
predict_datagen = ImageDataGenerator(rescale=1. / 255)
predict = predict_datagen.flow_from_directory(path_pred, target_size=(227,227), batch_size = 1, class_m
Found 7301 images belonging to 1 classes.

[ ]

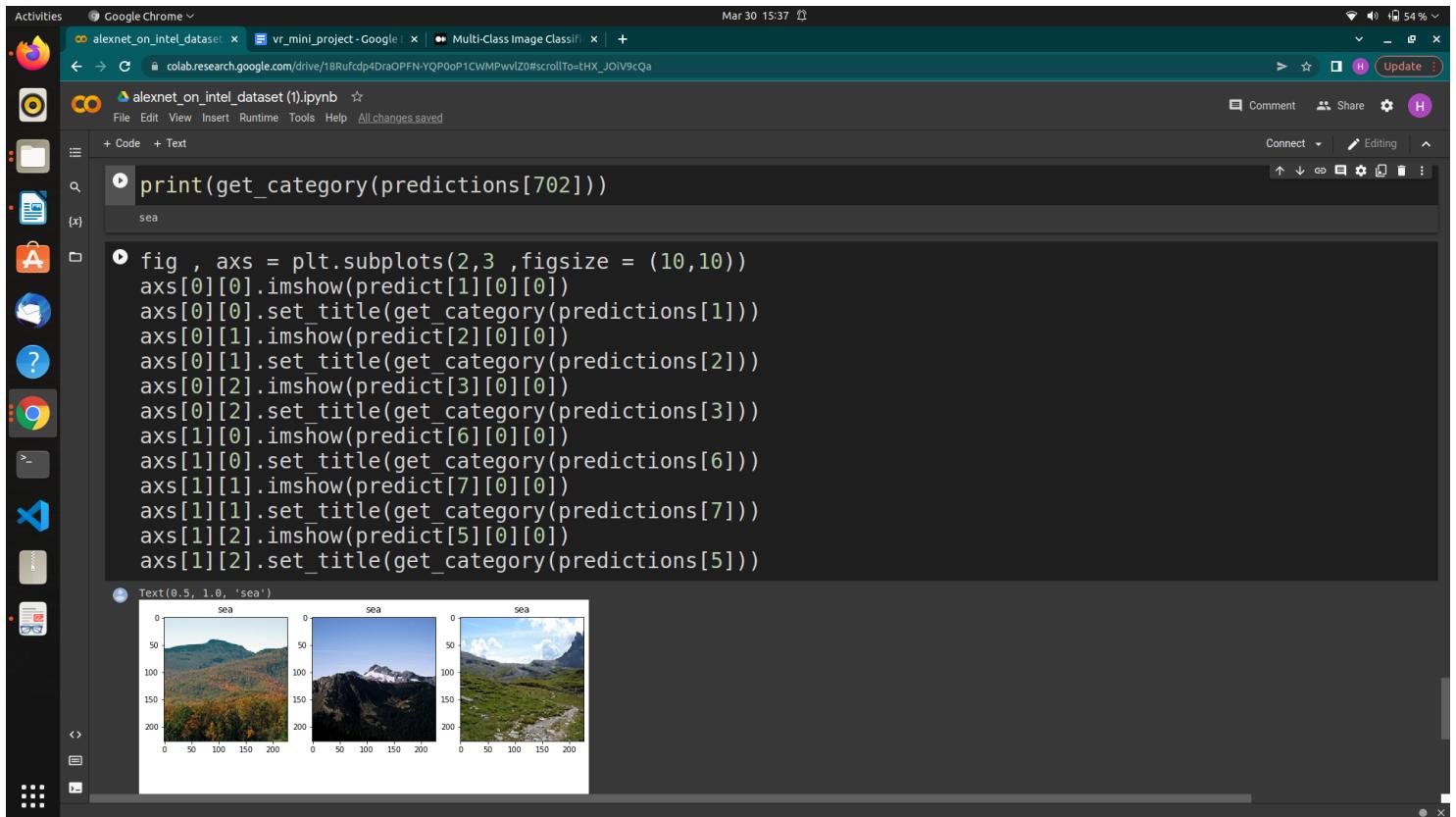
▶ ## using predict_generator() function to predict
predictions = alex.predict_generator(predict)

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: UserWarning: 'Model.predict_generator' is deprecated and will be removed in a future version. Please use 'Model.predict', which supports gene

[ ] imshow(predictions[702][0][0])
<matplotlib.image.AxesImage at 0x7f94f7c1f290>

[ ] print(predictions[702]) ### this is giving the probabilities of every class for 702th image
[2.6123534e-04 2.7370428e-05 9.3322390e-01 6.0048345e-02 3.7787892e-03
 2.6604233e-03]
```

This is the output of our model, since we used softmax at last layer , the model is returning the probabilities for each category for this particular image input. As seen above, the model is predicting the image as ‘sea’ with a probability of 0.933223

Now we don't want to have this to be our output format, so we will make a function that will give us the category to which the Input Image, predicted by the model will belong to.



A screenshot of a Google Colab notebook titled "alexnet_on_intel_dataset (1).ipynb". The code cell contains the following Python code:

```
print(get_category(predictions[702]))
```

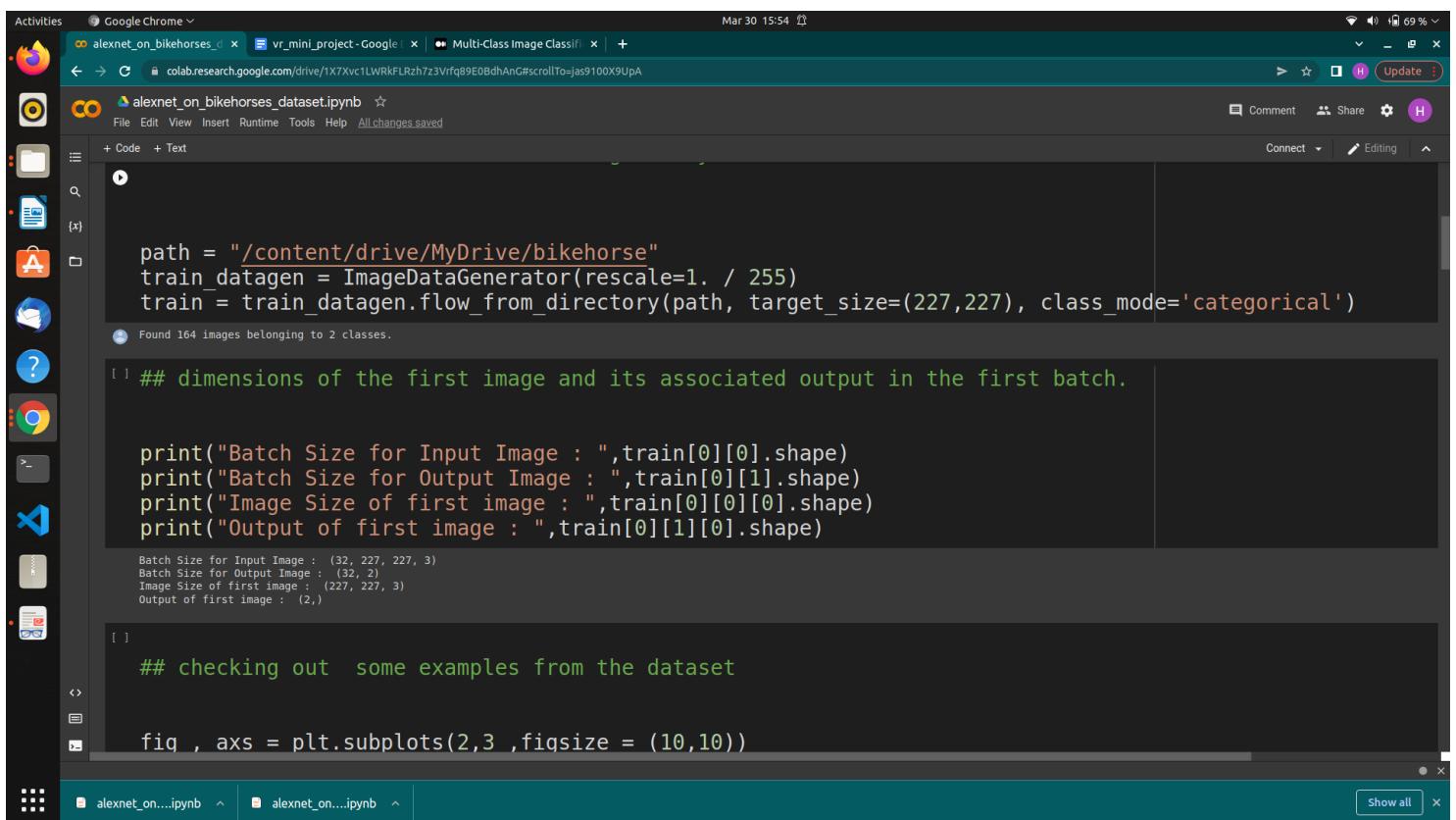
The code uses plt.subplots to create a 2x3 grid of subplots. Each subplot displays a different image of a landscape, all labeled "sea". The first subplot shows a mountain range with autumn foliage. The second subplot shows a mountain range with snow-capped peaks. The third subplot shows a green hillside under a blue sky with clouds. The x-axis for each plot ranges from 0 to 200, and the y-axis ranges from 0 to 200.

Since we have limited access of GPU on collab so can't able to run more epochs on such a huge dataset, so training accuracy is not that much and also some prediction is going wrong.

ALEXNET ON BIKE VS HORSE DATASET

Now we have implemented same AlexNet CNN on 2 classes datasets given by sir that's on Horse and Bike datasets:-

Uploading the datasets to train the model upon and checking the batch size after changing the target size which are by default:-



The screenshot shows a Google Colab notebook titled "alexnet_on_bikehorses.ipynb". The code cell contains Python code for generating training data from a directory:

```
path = "/content/drive/MyDrive/bikehorse"
train_datagen = ImageDataGenerator(rescale=1. / 255)
train = train_datagen.flow_from_directory(path, target_size=(227,227), class_mode='categorical')
```

Output of the first cell:

```
[ ] ## dimensions of the first image and its associated output in the first batch.

print("Batch Size for Input Image : ",train[0][0].shape)
print("Batch Size for Output Image : ",train[0][1].shape)
print("Image Size of first image : ",train[0][0][0].shape)
print("Output of first image : ",train[0][1][0].shape)

Batch Size for Input Image : (32, 227, 227, 3)
Batch Size for Output Image : (32, 2)
Image Size of first image : (227, 227, 3)
Output of first image : (2,)
```

Second code cell:

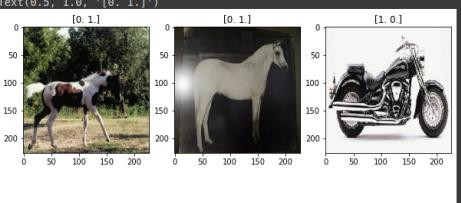
```
[ ] ## checking out some examples from the dataset

fig , axs = plt.subplots(2,3 ,figsize = (10,10))
```

Bottom navigation bar:

- alexnet_on...ipynb ^
- alexnet_on...ipynb ^
- Show all

checking out some examples here from our train datasets of bikes and horses:-



A screenshot of a Google Colab notebook titled "alexnet_on_bikehorses.ipynb". The code cell contains Python code using plt.subplots to display three images side-by-side. The first image is a horse, the second is a white horse, and the third is a motorcycle. Each image has a title above it indicating its class: [0, 1] for the first two and [1, 0] for the motorcycle.

```
fig, axs = plt.subplots(2,3, figsize = (10,10))
axs[0][0].imshow(train[0][0][12])
axs[0][0].set_title(train[0][1][12])
axs[0][1].imshow(train[0][0][10])
axs[0][1].set_title(train[0][1][10])
axs[0][2].imshow(train[0][0][5])
axs[0][2].set_title(train[0][1][5])
axs[1][0].imshow(train[0][0][20])
axs[1][0].set_title(train[0][1][20])
axs[1][1].imshow(train[0][0][25])
axs[1][1].set_title(train[0][1][25])
axs[1][2].imshow(train[0][0][3])
axs[1][2].set_title(train[0][1][3])
```

Training the model using fit_generator with the following command with 10 epochs :

Activities

Google Chrome

alexnet_on_bikehorses.ipynb vr_mini_project-Google Colab Multi-Class Image Classifi

Mar 30 15:57

File Edit View Insert Runtime Tools Help All changes saved

Comment Share Connect Editing

```
[ ] alex.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics=['accuracy'])

[ ] erator(train,epochss=10) ## we are using here 10 epochs have checked through increasing epochs too
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning: 'Model.fit_generator' is deprecated and will be removed in a future version. Please use 'Model.fit', which supports generators.
  """Entry point for launching an IPython kernel.

Epoch 1/10
6/6 [=====] - 1s 228ms/step - loss: 0.0209 - accuracy: 0.9939
Epoch 2/10
6/6 [=====] - 1s 228ms/step - loss: 0.0216 - accuracy: 0.9939
Epoch 3/10
6/6 [=====] - 1s 189ms/step - loss: 0.0158 - accuracy: 0.9939
Epoch 4/10
6/6 [=====] - 1s 188ms/step - loss: 0.0330 - accuracy: 0.9878
Epoch 5/10
6/6 [=====] - 1s 184ms/step - loss: 0.0682 - accuracy: 0.9695
Epoch 6/10
6/6 [=====] - 1s 202ms/step - loss: 0.0971 - accuracy: 0.9634
Epoch 7/10
6/6 [=====] - 1s 185ms/step - loss: 0.0505 - accuracy: 0.9756
Epoch 8/10
6/6 [=====] - 1s 200ms/step - loss: 0.0371 - accuracy: 0.9878
Epoch 9/10
6/6 [=====] - 1s 187ms/step - loss: 0.0387 - accuracy: 0.9878
Epoch 10/10
6/6 [=====] - 1s 193ms/step - loss: 0.0376 - accuracy: 0.9817
<keras.callbacks.History at 0x7f748ef4bdd0>

[ ] # path_test = '/content/drive/MyDrive/test'
# test_datagen = ImageDataGenerator(rescale=1. / 255)
# test = test_datagen.flow_from_directory(path_test, target_size=(227,227), class_mode='categorical')
Found 0 images belonging to 0 classes.

[ ] # preds = alex.evaluate_generator(test)

alexnet_on_bikehorses.ipynb alexnet_on_bikehorses.ipynb Show all
```

This is the output of our model, since we used softmax at last layer , the model is returning the probabilities for each category for this particular image input. As seen above, the model is predicting the image as 'Horse' with a probability of 0.552

Activities

Google Chrome

alexnet_on_bikehorses.ipynb vr_mini_project-Google Colab Multi-Class Image Classifi

Mar 30 15:59

File Edit View Insert Runtime Tools Help All changes saved

Comment Share Connect Editing

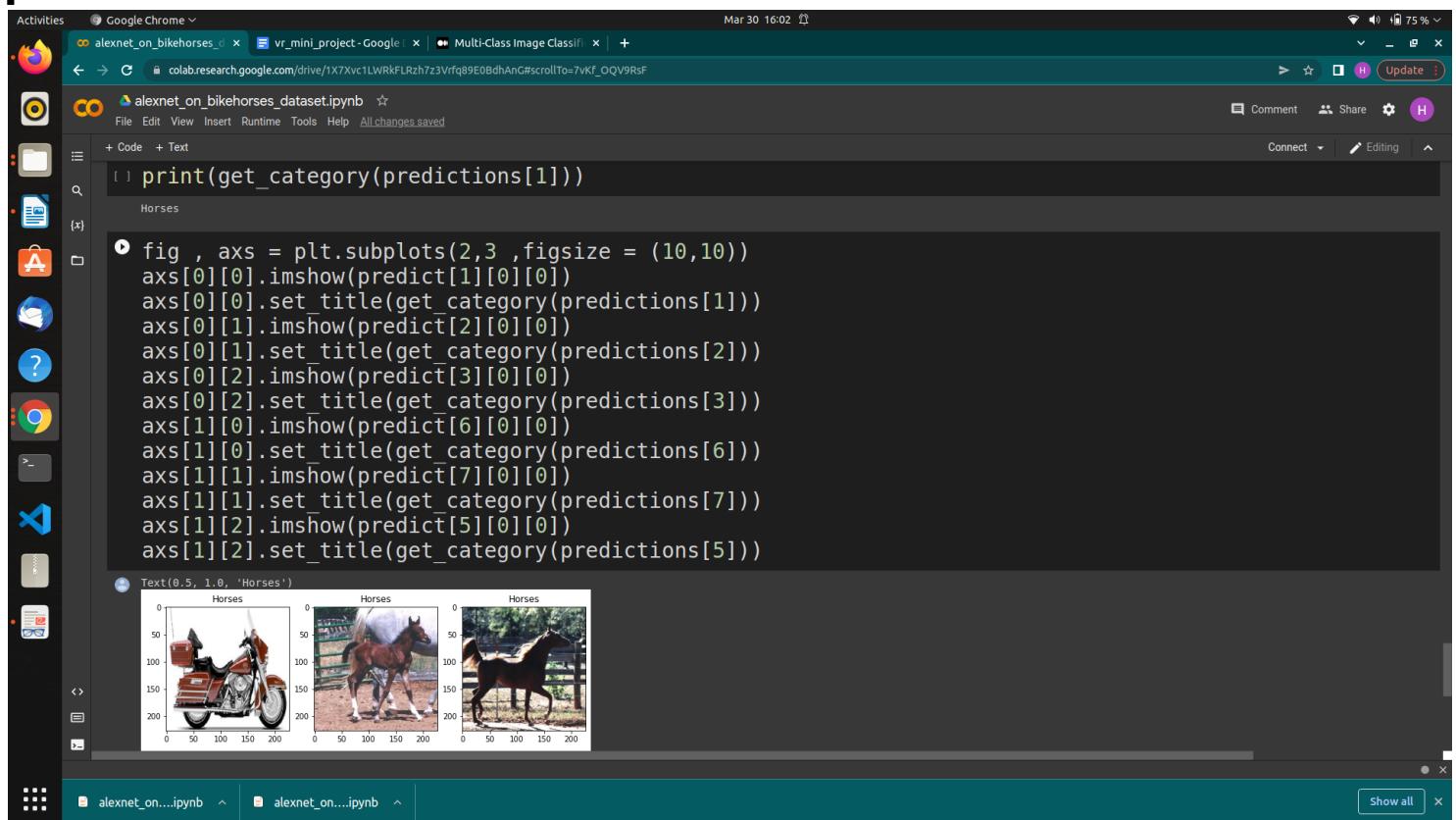
```
[ ] # PRINT ( TEST ACCURACY - + str(preds[1]))
[ ] ## loading the prediction data that is "seg_pred" on which we have to predict :
path_test = "/content/drive/MyDrive/test"
predict_datagen = ImageDataGenerator(rescale=1. / 255)
predict = predict_datagen.flow_from_directory(path_test, target_size=(227,227), batch_size = 1,class_mode='categorical')
Found 15 images belonging to 1 classes.

[ ] ## using predict_generator() function to predict
predictions = alex.predict_generator(predict)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning: 'Model.predict_generator' is deprecated and will be removed in a future version. Please use 'Model.predict', which supports generators.
  """Entry point for launching an IPython kernel.

[ ] imshow(predict[4][0][0])
<matplotlib.image.AxesImage at 0x7f748e42bf50>
<img alt="A white horse standing in a field, viewed from the side and slightly from behind, looking towards the right. The image is a 227x227 pixel square." data-bbox="120 815 245 905>

alexnet_on_bikehorses.ipynb alexnet_on_bikehorses.ipynb Show all
```

so here are the some prediction of our model on prediction datasets:-



A screenshot of a Google Colab notebook titled "alexnet_on_bikehorses.ipynb". The code cell contains Python code to print the category of the first prediction and to display six images of horses in a 2x3 grid. The images are labeled "Horses" and show different breeds of horses in various settings. The notebook interface includes a sidebar with various icons and a toolbar at the top.

```
print(get_category(predictions[1]))  
Horses  
fig, axs = plt.subplots(2,3, figsize = (10,10))  
axs[0][0].imshow(predict[1][0][0])  
axs[0][0].set_title(get_category(predictions[1]))  
axs[0][1].imshow(predict[2][0][0])  
axs[0][1].set_title(get_category(predictions[2]))  
axs[0][2].imshow(predict[3][0][0])  
axs[0][2].set_title(get_category(predictions[3]))  
axs[1][0].imshow(predict[6][0][0])  
axs[1][0].set_title(get_category(predictions[6]))  
axs[1][1].imshow(predict[7][0][0])  
axs[1][1].set_title(get_category(predictions[7]))  
axs[1][2].imshow(predict[5][0][0])  
axs[1][2].set_title(get_category(predictions[5]))
```

Conclusion:-

since we have very small dataset given by sir so i have trained my model by varying the epochs from 300 epochs to 50 epochs but prediction accuracy was coming very good o train dataset but not able to predict all data of test datasets properly .

so lastly concluded my prediction after training by giving 100 epochs on training datasets.

Report of Assignment 3c

AUTO DETECTION USING YOLO

What is YOLO?

YOLO is an abbreviation for the term ‘You Only Look Once’. This is an algorithm that detects and recognizes various objects in a picture (in real-time). Object detection in YOLO is done as a regression problem and provides the class probabilities of the detected images.

It takes the entire image in a single instance and predicts the bounding box coordinates and class probabilities for these boxes. The biggest advantage of using YOLO is its superb speed, it's incredibly fast and can process 45 frames per second. YOLO also understands generalized object representation.

It is based on regression, instead of selecting the interesting part of an Image, it predicts classes and bounding boxes for the whole image in one run of the Algorithm.

Object detection consists of various approaches such as fast R-CNN, Retina-Net, and Single-Shot MultiBox Detector (SSD). Although these approaches have solved the challenges of data limitation and modeling in object detection, they are not able to detect objects in a single algorithm run. YOLO algorithm has gained popularity because of its superior performance over the aforementioned object detection techniques.

The YOLO algorithm employs convolutional neural networks (CNN) to detect objects in real-time. As the name suggests, the algorithm requires only a single forward propagation through a neural network to detect objects.

This means that prediction in the entire image is done in a single algorithm run. The CNN is used to predict various class probabilities and bounding boxes simultaneously.

The YOLO algorithm consists of various variants. Some of the common ones include tiny YOLO and YOLOv3.

Why is the YOLO algorithm important?

YOLO algorithm is important because of the following reasons:

- **Speed:** This algorithm improves the speed of detection because it can predict objects in real-time.
- **High accuracy:** YOLO is a predictive technique that provides accurate results with minimal background errors.
- **Learning capabilities:** The algorithm has excellent learning capabilities that enable it to learn the representations of objects and apply them in object detection.

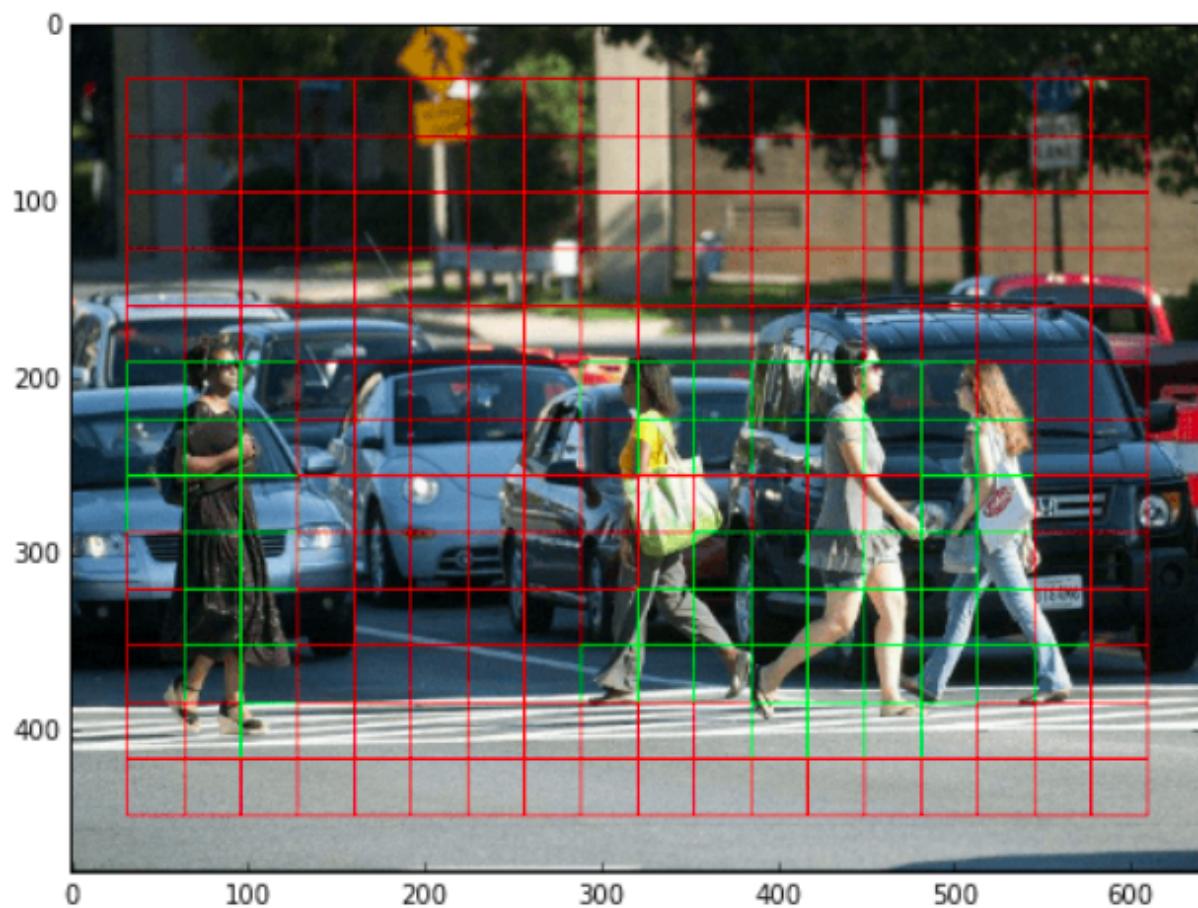
How the YOLO algorithm works

YOLO algorithm works using the following three techniques:

- Residual blocks
- Bounding box regression
- Intersection Over Union (IOU)

Residual blocks

First, the image is divided into various grids. Each grid has a dimension of $S \times S$. The following image shows how an input image is divided into grids.



In the image above, there are many grid cells of equal dimension. Every grid cell will detect objects that appear within them. For example, if an object center appears within a certain grid cell, then this cell will be responsible for detecting it.

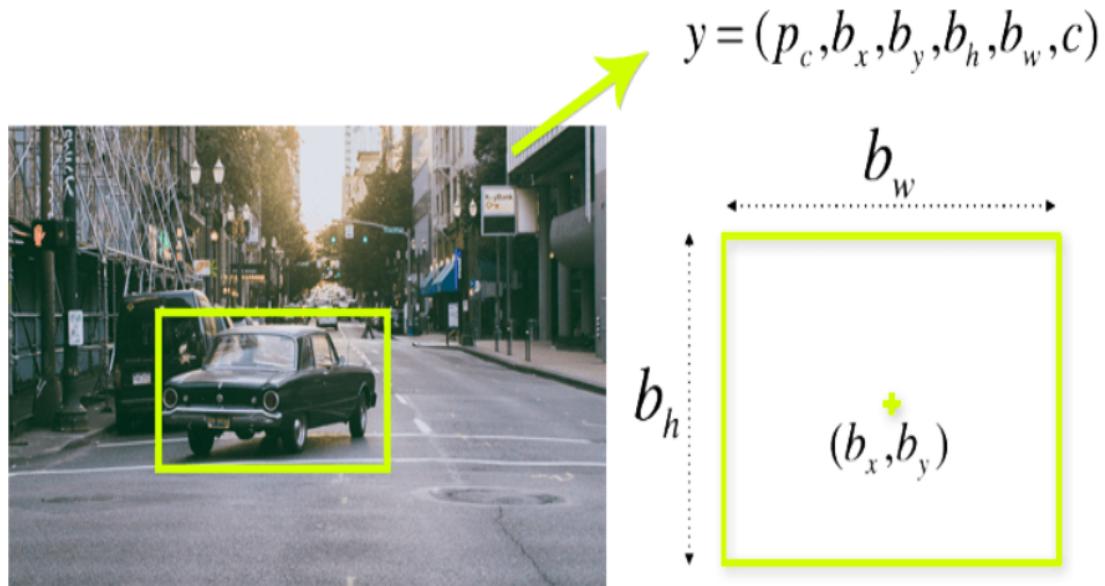
Bounding box regression

A bounding box is an outline that highlights an object in an image.

Every bounding box in the image consists of the following attributes:

- Width (bw)
- Height (bh)
- Class (for example, person, car, traffic light, etc.)- This is represented by the letter c.
- Bounding box center (bx,by)

The following image shows an example of a bounding box. The bounding box has been represented by a yellow outline.



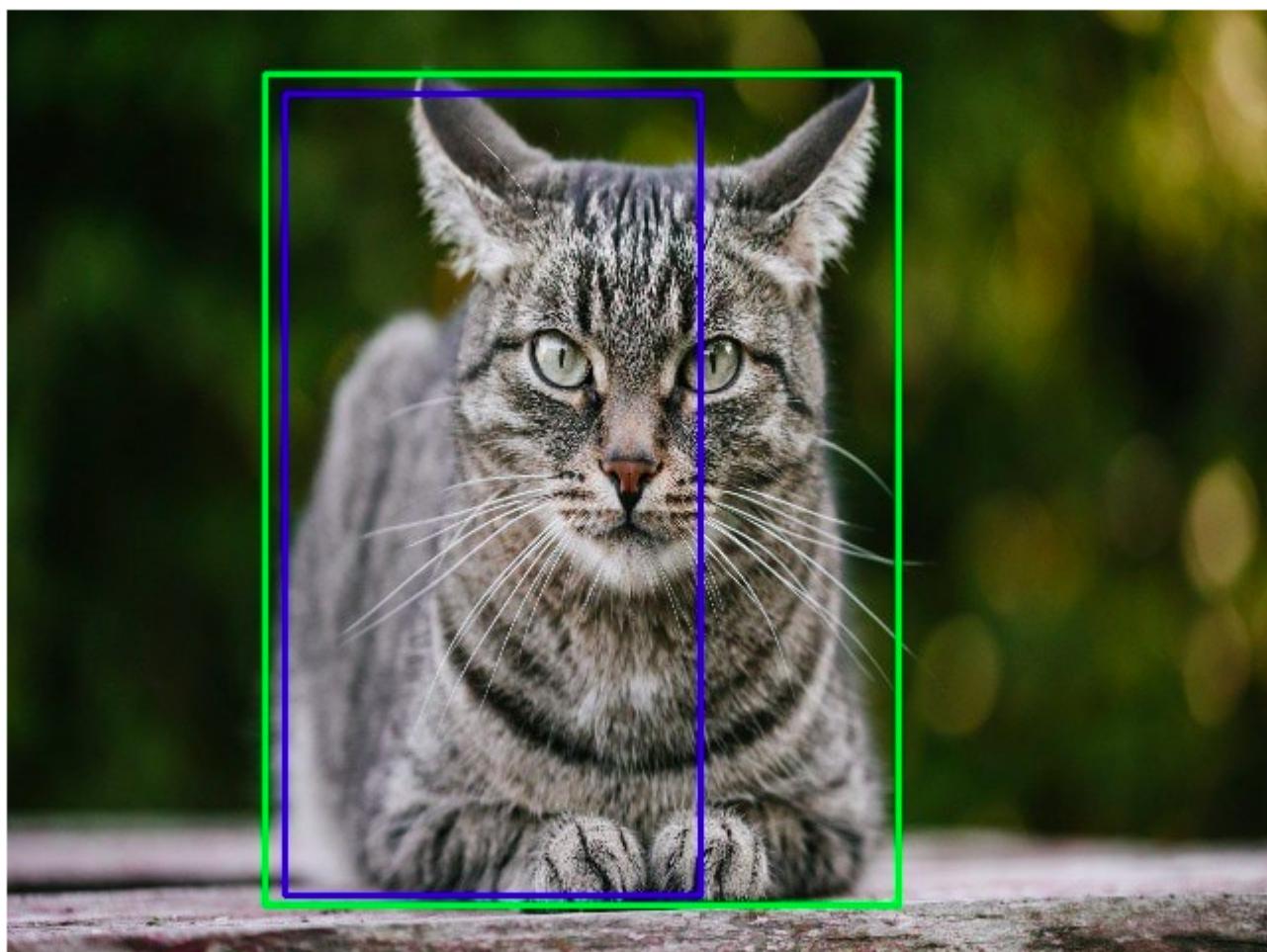
YOLO uses a single bounding box regression to predict the height, width, center, and class of objects. In the image above, represents the probability of an object appearing in the bounding box.

Intersection over union (IOU)

Intersection over union (IOU) is a phenomenon in object detection that describes how boxes overlap. YOLO uses IOU to provide an output box that surrounds the objects perfectly.

Each grid cell is responsible for predicting the bounding boxes and their confidence scores. The IOU is equal to 1 if the predicted bounding box is the same as the real box. This mechanism eliminates bounding boxes that are not equal to the real box.

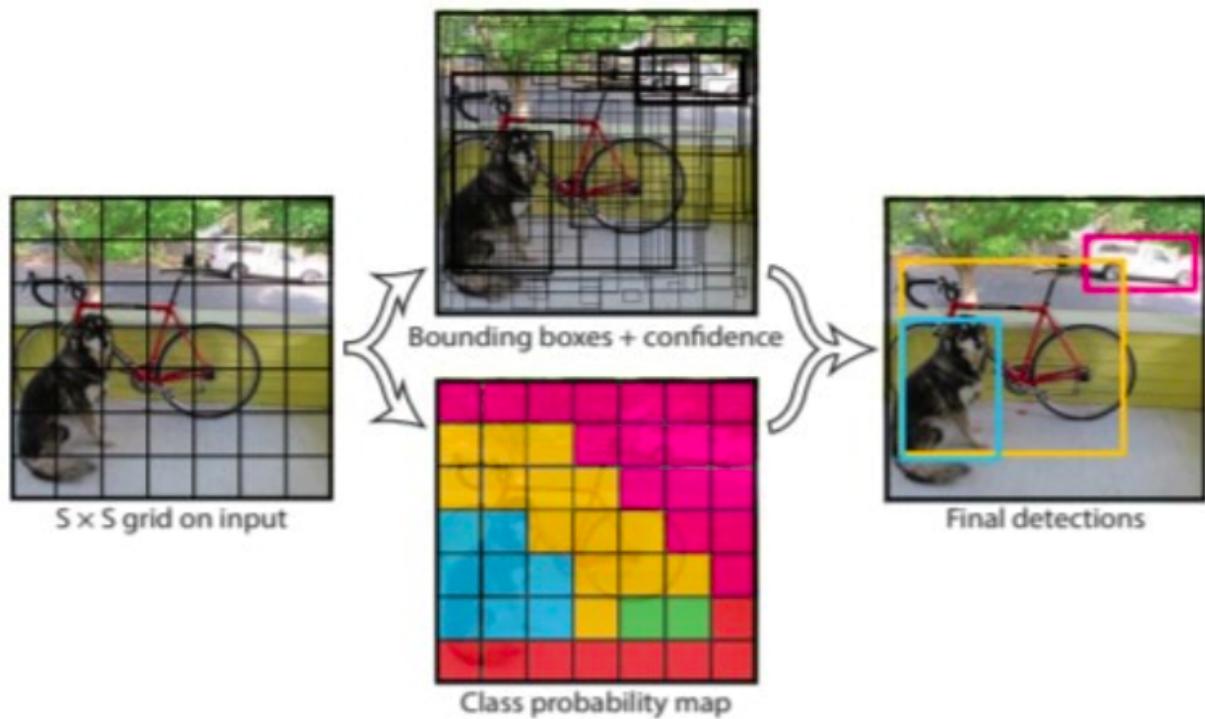
The following image provides a simple example of how IOU works.



In the image above, there are two bounding boxes, one in green and the other one in blue. The blue box is the predicted box while the green box is the real box. YOLO ensures that the two bounding boxes are equal.

Combination of the three techniques

The following image shows how the three techniques are applied to produce the final detection results.

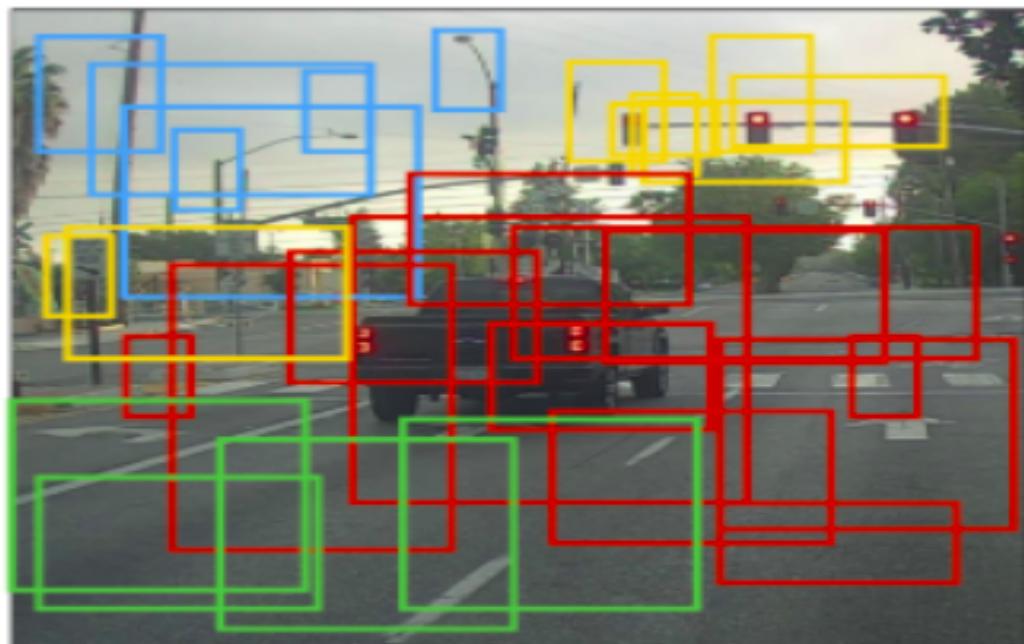


First, the image is divided into grid cells. Each grid cell forecasts B bounding boxes and provides their confidence scores. The cells predict the class probabilities to establish the class of each object.

For example, we can notice at least three classes of objects: a car, a dog, and a bicycle. All the predictions are made simultaneously using a single

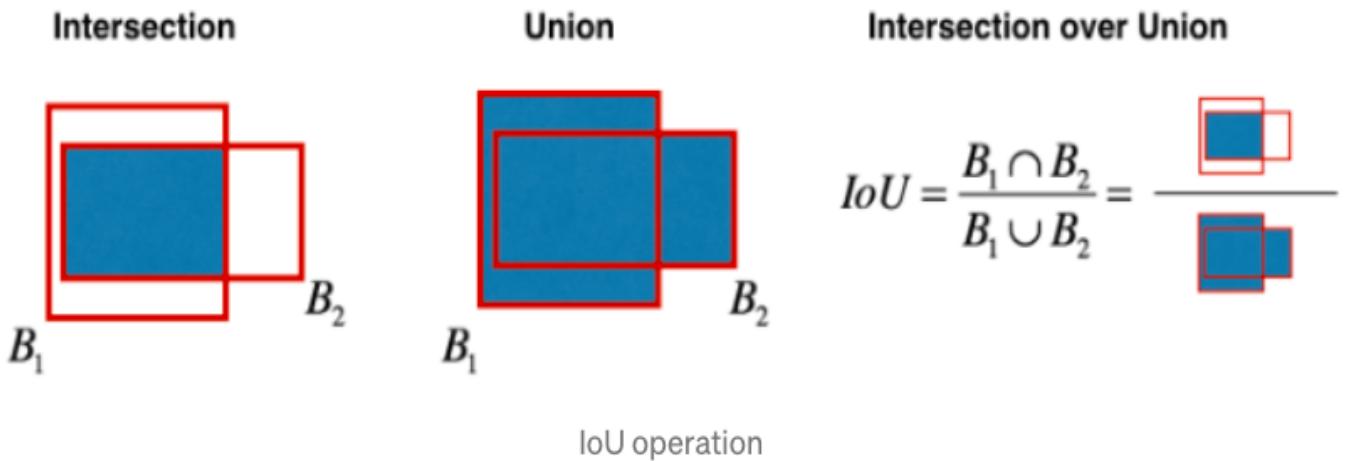
convolutional neural network.

This shows the before and after of predicting the class probabilities for each grid cell. After predicting the class probabilities, the next step is Non-max suppression. It helps the algorithm to get rid of the unnecessary anchor boxes, like you can see in the figure below, there are numerous anchor boxes calculated based on the class probabilities.



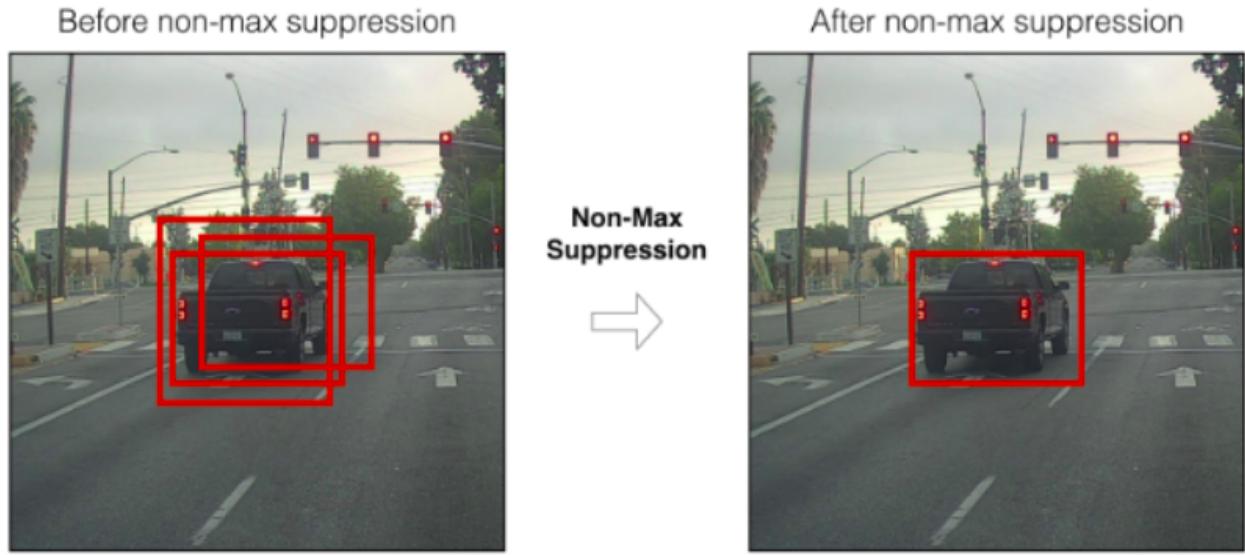
Anchor boxes

To resolve this problem Non-max suppression eliminates the bounding boxes that are very close by performing the IoU (Intersection over Union) with the one having the highest class probability among them.



It calculates the value of IoU for all the bounding boxes respective to the one having the highest class probability, it then rejects the bounding boxes whose value of IoU is greater than a threshold. It signifies that those two bounding boxes are covering the same object but the other one has a low probability for the same, thus it is eliminated.

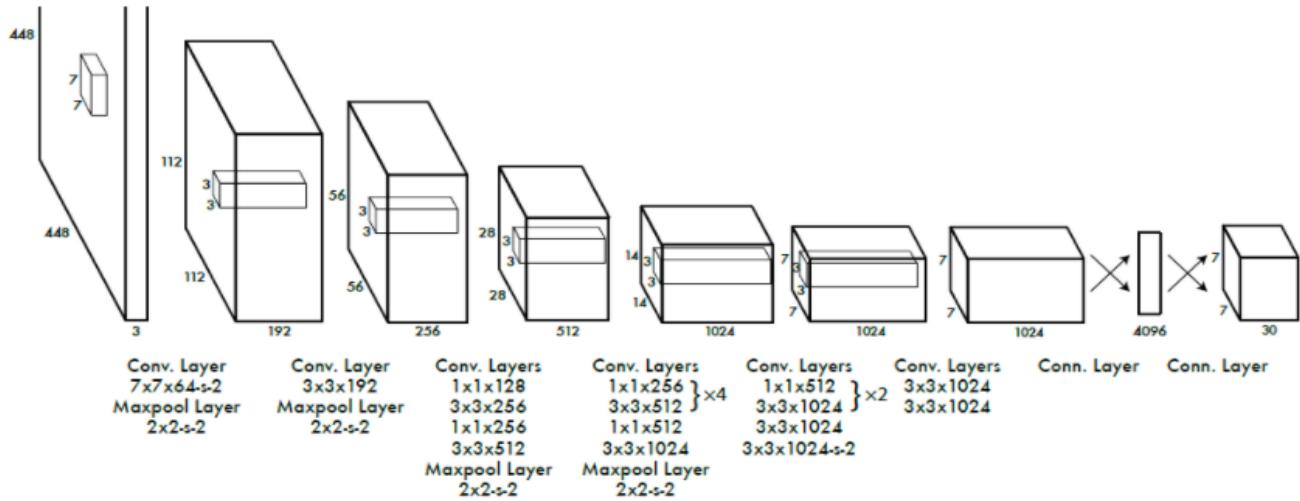
Once done, the algorithm finds the bounding box with next highest class probabilities and does the same process, it is done until we are left with all the different bounding boxes.



Before and After of Non-max suppression

After this, almost all of our work is done, the algorithm finally outputs the required vector showing the details of the bounding box of the respective class. The overall architecture of the algorithm can be viewed below :

Architecture of YOLOV3



YOLO Architecture, source: [You Only Look Once: Unified, Real-Time Object detection](#)

Still improvements are being made in the algorithm. We currently have four generations of the YOLO Algorithm from v1 to v4, along with a slightly small version of it YOLO-tiny, it is specifically designed to achieve a incredibly high speed of 220fps.

Training YOLOv3 object detection on AUTO dataset

To train a YOLOv3 object detection model we'll training an auto detection model on our own dataset of auto using Colab and darknet

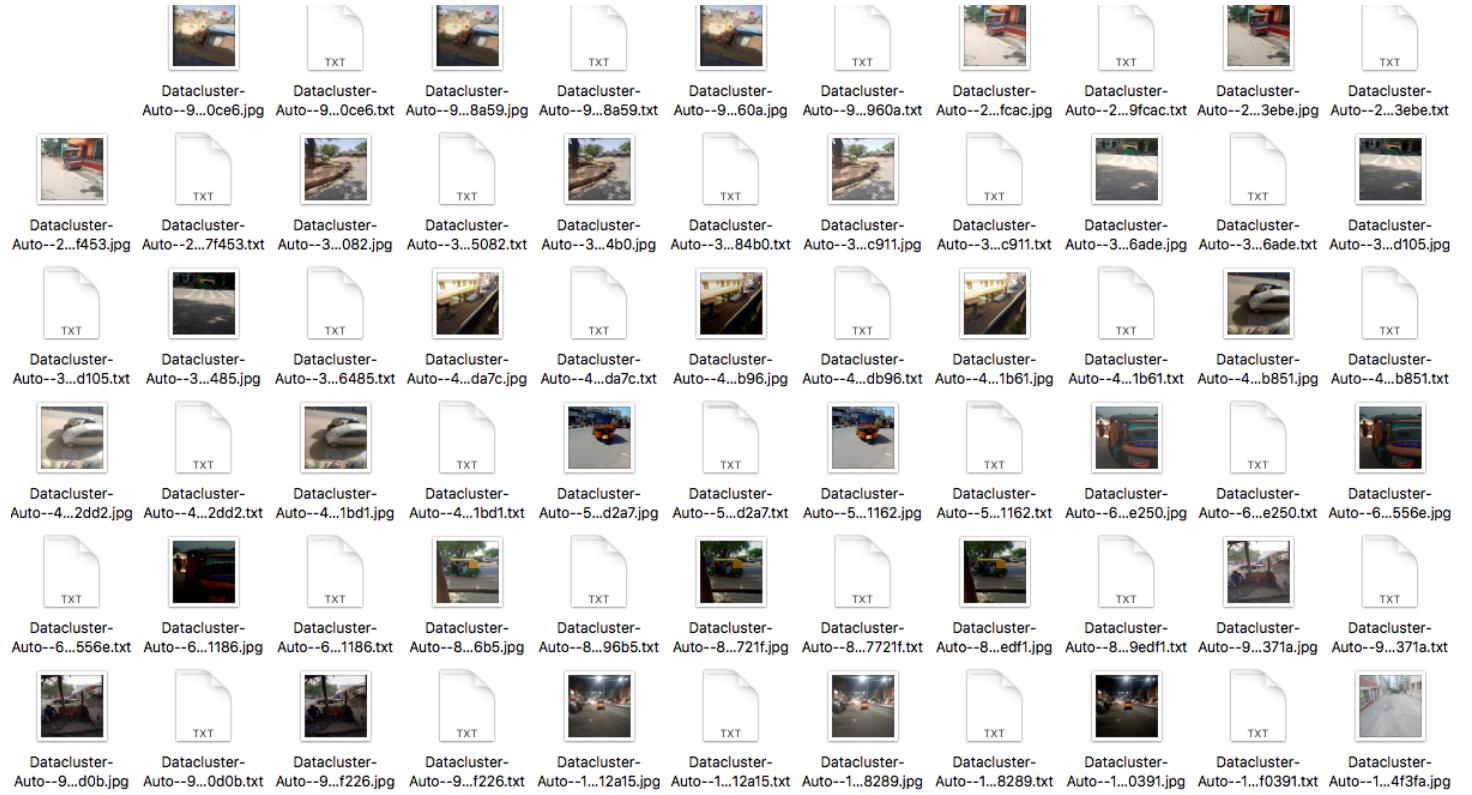
Our Data:

given Our dataset of 150 unannotated auto images.

ANNOTATING ALL THE IMAGES

Firstly, we have annotated all the images using roboflow software and converted the annotated file into the yolov3 format as shown below:

the image and its annotated txt file is organized at the same place as shown below:

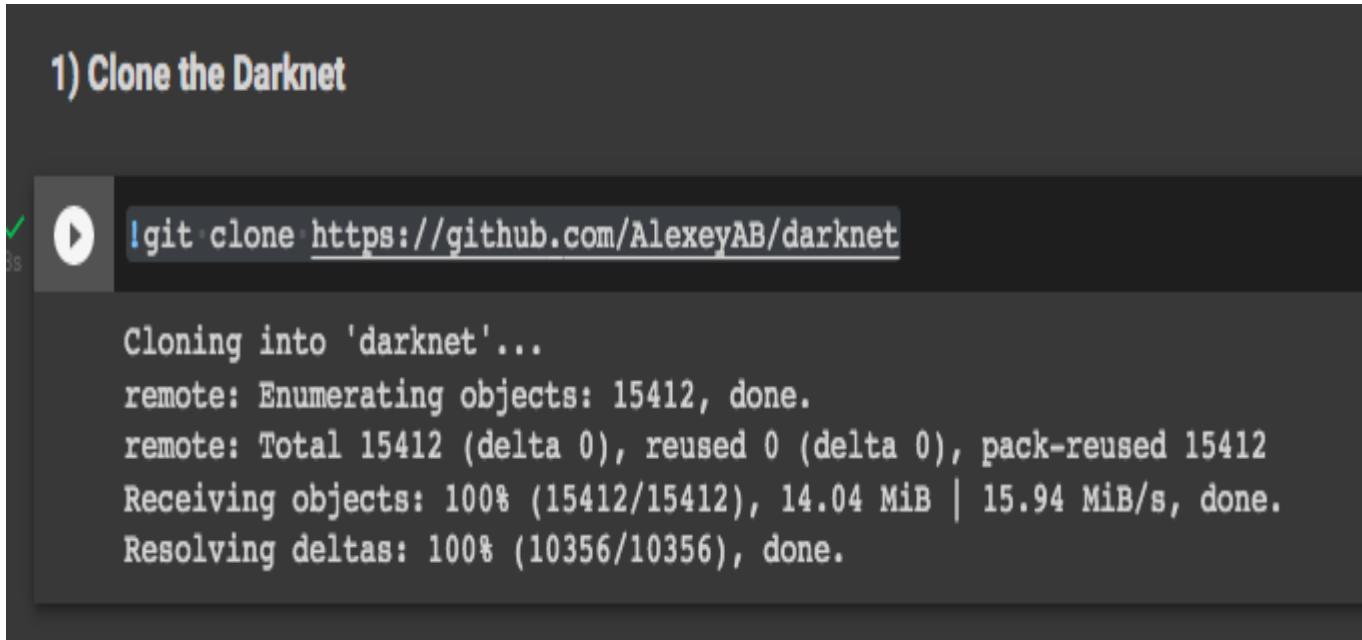


a typical .txt annotated file for an image :

```
 Datacluster-Auto--25-.jpg.rf.8676a9e11a7d81002f6d71ccc9434287.txt Open withTextEdit
0 0.5123697916666666 0.24169921875 0.08919270833333333 0.0654296875
0 0.2708333333333333 0.3115234375 0.1614583333333334 0.10888671875
0 0.37890625 0.43408203125 0.1569010416666666 0.083984375
0 0.7493489583333334 0.4755859375 0.10611979166666667 0.1328125
0 0.4791666666666667 0.49755859375 0.18619791666666666 0.10791015625
0 0.8274739583333334 0.17822265625 0.05533854166666664 0.04541015625
```

cloning the darknet repository

```
!git clone https://github.com/AlexeyAB/darknet
```



The screenshot shows a terminal window with a dark background. At the top, the text "1) Clone the Darknet" is displayed in a light-colored font. Below this, a command line interface is shown with the following text:
!git clone https://github.com/AlexeyAB/darknet
Cloning into 'darknet'...
remote: Enumerating objects: 15412, done.
remote: Total 15412 (delta 0), reused 0 (delta 0), pack-reused 15412
Receiving objects: 100% (15412/15412), 14.04 MiB | 15.94 MiB/s, done.
Resolving deltas: 100% (10356/10356), done.

Darknet

Darknet: An open source platform for neural networks in C

Darknet is a high performance open source framework for the implementation of neural networks. Written in C and CUDA, it can be integrated with CPUs and GPUs.

Advanced implementations of deep neural networks can be done using Darknet. These implementations include You Only Look Once (YOLO) for real-time object detection, ImageNet classification, recurrent neural networks (RNNs), and many others

configuring the darknet network:

3) Configure Darknet network for training YOLO V3

```
[ ] !cp cfg/yolov3.cfg cfg/yolov3_training.cfg

[ ] !sed -i 's/batch=1/batch=64/' cfg/yolov3_training.cfg
!sed -i 's/subdivisions=1/subdivisions=16/' cfg/yolov3_training.cfg
!sed -i 's/max_batches = 500200/max_batches = 6000/' cfg/yolov3_training.cfg
!sed -i '610 s@classes=80@classes=1@' cfg/yolov3_training.cfg
!sed -i '696 s@classes=80@classes=1@' cfg/yolov3_training.cfg
!sed -i '783 s@classes=80@classes=1@' cfg/yolov3_training.cfg
!sed -i '603 s@filters=255@filters=18@' cfg/yolov3_training.cfg
!sed -i '689 s@filters=255@filters=18@' cfg/yolov3_training.cfg
!sed -i '776 s@filters=255@filters=18@' cfg/yolov3_training.cfg

[ ] # Create folder on google drive so that we can save there the weights
!mkdir "/content/drive/MyDrive/yolo_v3"

mkdir: cannot create directory '/content/drive/MyDrive/yolo_v3': File exists

[ ] !echo "auto" > data/obj.names
!echo -e 'classes= 1\ntrain  = data/train.txt\nvalid   = data/test.txt\nnames = data/obj.names\nbackup = /content/drive/MyDrive/yolo_v3' > data/obj.data
!mkdir data/obj

mkdir: cannot create directory 'data/obj': File exists
```

Downloading the pretrained weights of the YOLO :-

```
[ ] # Download weights darknet model 53
!wget https://pjreddie.com/media/files/darknet53.conv.74

[ ] --2022-03-28 13:26:39-- https://pjreddie.com/media/files/darknet53.conv.74
Resolving pjreddie.com (pjreddie.com)... 128.208.4.108
Connecting to pjreddie.com (pjreddie.com)|128.208.4.108|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 162482580 (155M) [application/octet-stream]
Saving to: 'darknet53.conv.74.2'

darknet53.conv.74.2 100%[=====] 154.96M 21.0MB/s    in 7.8s

2022-03-28 13:26:47 (19.9 MB/s) - 'darknet53.conv.74.2' saved [162482580/162482580]
```

We're going to convert the class index on the .txt files. As we're working with only one

class, it's supposed to be class 0. If the index is different from 0 then we're going to change it.

```
import glob
import os
import re

txt_file_paths = glob.glob(r"data/obj/train/*.txt")
for i, file_path in enumerate(txt_file_paths):
    # get image size
    with open(file_path, "r") as f_o:
        lines = f_o.readlines()

    text_converted = []
    for line in lines:
        print(line)
        numbers = re.findall("[0-9.]+", line)
        print(numbers)
        if numbers:

            # Define coordinates
            text = "{} {} {} {} {}".format(0, numbers[1], numbers[2], numbers[3], numbers[4])
            text_converted.append(text)
            print(i, file_path)
            print(text)
    # Write file
    with open(file_path, 'w') as fp:
        for item in text_converted:
            fp.writelines("{}\n".format(item))

92 data/obj/train/Datacluster-Auto--65-_jpg.rf.33a7b2023cc41d77c9d6ac79a30b8b1f.txt
0 0.505859375 0.255859375 0.0911458333333333 0.05078125
0 0.294921875 0.5517578125 0.033203125 0.02099609375

['0', '0.294921875', '0.5517578125', '0.033203125', '0.02099609375']
93 data/obj/train/Datacluster-Auto--43-_jpg.rf.f033008d31f044395a793c367a22e581.txt
0 0.505859375 0.255859375 0.0911458333333333 0.05078125
```

Making the list of names all the images :

```
import glob
images_list = glob.glob("data/obj/train/*.jpg")
print(len(images_list))
print(images_list)

782
['data/obj/train/Datacluster-Auto--33-_jpg.rf.0b4800f8738b5510bbbfa898cb91bb3.jpg', 'data/obj/train/Datacluster-Auto--46-_jpg.rf.c4d40aa79cd458764ec6dce44c
```

creating the train.txt file for training the model:

```
[ ] #Create training.txt file
file = open("data/train.txt", "w")
file.write("\n".join(images_list))
file.close()
```

Training the model for 300 epochs :

```
# Start the training
!./darknet detector train data/obj.data cfg/yolov3_training.cfg darknet53.conv.74 -dont_show -epochs 300

v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.610298), count: 3, class_loss = 1.760342, iou_loss = 0.223981, total_loss =
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.565712), count: 1, class_loss = 1.136141, iou_loss = 0.107346, total_loss =
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.805603, iou_loss = 0.000000, total_loss =
total_bbox = 14268, rewritten_bbox = 0.000000 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.396995), count: 4, class_loss = 2.124478, iou_loss = 0.772548, total_loss =
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.296509), count: 1, class_loss = 1.104689, iou_loss = 0.805412, total_loss =
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.806219, iou_loss = 0.000000, total_loss =
total_bbox = 14273, rewritten_bbox = 0.000000 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.516060), count: 3, class_loss = 1.774929, iou_loss = 0.755288, total_loss =
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.257452), count: 1, class_loss = 1.163162, iou_loss = 0.743407, total_loss =
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.838266, iou_loss = 0.000000, total_loss =
total_bbox = 14277, rewritten_bbox = 0.000000 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.562737), count: 4, class_loss = 2.010544, iou_loss = 0.831665, total_loss =
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.000000), count: 1, class_loss = 0.864711, iou_loss = 0.000000, total_loss =
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.826429, iou_loss = 0.000000, total_loss =
total_bbox = 14281, rewritten_bbox = 0.000000 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.608213), count: 4, class_loss = 1.981104, iou_loss = 0.442960, total_loss =
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.000000), count: 1, class_loss = 0.852525, iou_loss = 0.000000, total_loss =
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.809039, iou_loss = 0.000000, total_loss =
total_bbox = 14285, rewritten_bbox = 0.000000 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.750019), count: 2, class_loss = 1.548289, iou_loss = 0.229079, total_loss =
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.500924), count: 2, class_loss = 1.420185, iou_loss = 0.498734, total_loss =
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.821955, iou_loss = 0.000000, total_loss =
total_bbox = 14289, rewritten_bbox = 0.000000 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.260090), count: 4, class_loss = 2.078038, iou_loss = 1.769656, total_loss =
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.000000), count: 1, class_loss = 0.842986, iou_loss = 0.000000, total_loss =
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.795839, iou_loss = 0.000000, total_loss =
total_bbox = 14293, rewritten_bbox = 0.000000 %
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.369633), count: 4, class_loss = 2.020746, iou_loss = 1.091243, total_loss =
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.000000), count: 1, class_loss = 0.863868, iou_loss = 0.000000, total_loss =
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class_loss = 0.802748, iou_loss = 0.000000, total_loss =
total_bbox = 14297, rewritten_bbox = 0.000000 %
```

Testing the model:

loading the yolo weights:

the YOLO trained weights are saved as `yolov3_training_last.weights` which we have used to test our model

▼ testing

```
[ ] # plot ALL results in test directory (NOTE: adjust figsize as you please)
# for img in np_images:
#     plt.figure(figsize=(8, 6))
#     plt.imshow(img)

[ ] ## testing

▶ import cv2
import numpy as np
import glob
import random
import matplotlib.pyplot as plt

[ ] from google.colab.patches import cv2_imshow
# cv2_imshow(img)

[ ] # Load Yolo
net = cv2.dnn.readNet("/content/drive/MyDrive/yolo_v3/yolov3_training_last.weights", "/content/drive/MyDrive/yolo_v3/yolov3_testing.cfg")

# Name custom object
classes = ["auto"]

# Images path
images_path = glob.glob(r"/content/drive/MyDrive/yolo_v3/testimages/*.jpg")
```

test image paths for testing our model:

```
▶ images_path
[ '/content/drive/MyDrive/yolo_v3/testimages/Datacluster Auto (7).jpg',
  '/content/drive/MyDrive/yolo_v3/testimages/Datacluster Auto (79).jpg',
  '/content/drive/MyDrive/yolo_v3/testimages/Datacluster Auto (28).jpg',
  '/content/drive/MyDrive/yolo_v3/testimages/Datacluster Auto (27).jpg',
  '/content/drive/MyDrive/yolo_v3/testimages/Datacluster Auto (24).jpg',
  '/content/drive/MyDrive/yolo_v3/testimages/Datacluster Auto (23).jpg',
  '/content/drive/MyDrive/yolo_v3/testimages/Datacluster Auto (10).jpg',
  '/content/drive/MyDrive/yolo_v3/testimages/Datacluster Auto (101).jpg']
```

loading images for detection:

```
# Loading image
img = cv2.imread(img_path)
img = cv2.resize(img, None, fx=0.8, fy=0.8)
height, width, channels = img.shape
```

detecting objects and showing information on the screen:

```

# Detecting objects
blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

net.setInput(blob)
outs = net.forward(output_layers)

# Showing informations on the screen
class_ids = []
confidences = []
boxes = []
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.3:
            # Object detected
            print(class_id)
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)

```

inserting the rectangular co-ordinates in the image for each auto detected:

```

h = int(detection[3] * height)

# Rectangle coordinates
x = int(center_x - w / 2)
y = int(center_y - h / 2)

boxes.append([x, y, w, h])
confidences.append(float(confidence))
class_ids.append(class_id)

```

displaying the output:

Activities Firefox Web Browser Mar 30 01:37

Train YOLO to detect... Assignment3.pptx - Google Slides YOLO: Real-Time Object Detection | Object detection with YOLO v3 Train_YoloV3_final.ipynb WhatsApp

yo_lo_detection.ipynb

File Edit View Insert Runtime Tools Help Last saved at March 27

Reconnect Comment Share Editing

Files Connecting to a runtime to enable file browsing.

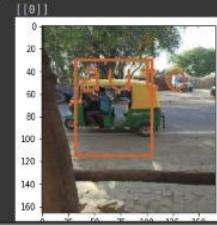
```
boxes.append([x, y, w, h])
confidences.append(float(confidence))
class_ids.append(class_id)

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
print(indexes)
font = cv2.FONT_HERSHEY_PLAIN
for i in range(len(boxes)):
    if i in indexes:
        x, y, w, h = boxes[i]
        label = str(classes[class_ids[i]])
        color = colors[class_ids[i]]
        cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
        cv2.putText(img, label, (x, y + 30), font, 3, color, 2)

# plt.imshow("Image", img)
# plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
# plt.show()
key = cv2.waitKey(0)

cv2.destroyAllWindows()
```

0 ([0])



Differences between YOLOv3, YOLOv4 and YOLOv5

	YOLOv3	YOLOv4	YOLOv5
Neural Network Type	Fully convolution	Fully convolution	Fully convolution
Backbone Feature Extractor	Darknet-53	CSPDarknet53	CSPDarknet53
Loss Function	Binary cross entropy	Binary cross entropy	Binary cross entropy and Logits loss function
Neck	FPN	SSP and PANet	PANet
Head	YOLO layer	YOLO layer	YOLO layer

The main differences between YOLOv3, YOLOv4, and YOLOv5 architecture is that YOLOv3 uses Darknet53 backbone. YOLOv4 architecture uses CSPdarknet53 as a backbone and YOLOv5 uses Focus structure with CSPdarknet53 as a backbone. The Focus layer is first introduced in YOLOv5

Auto detection using YOL0 V5

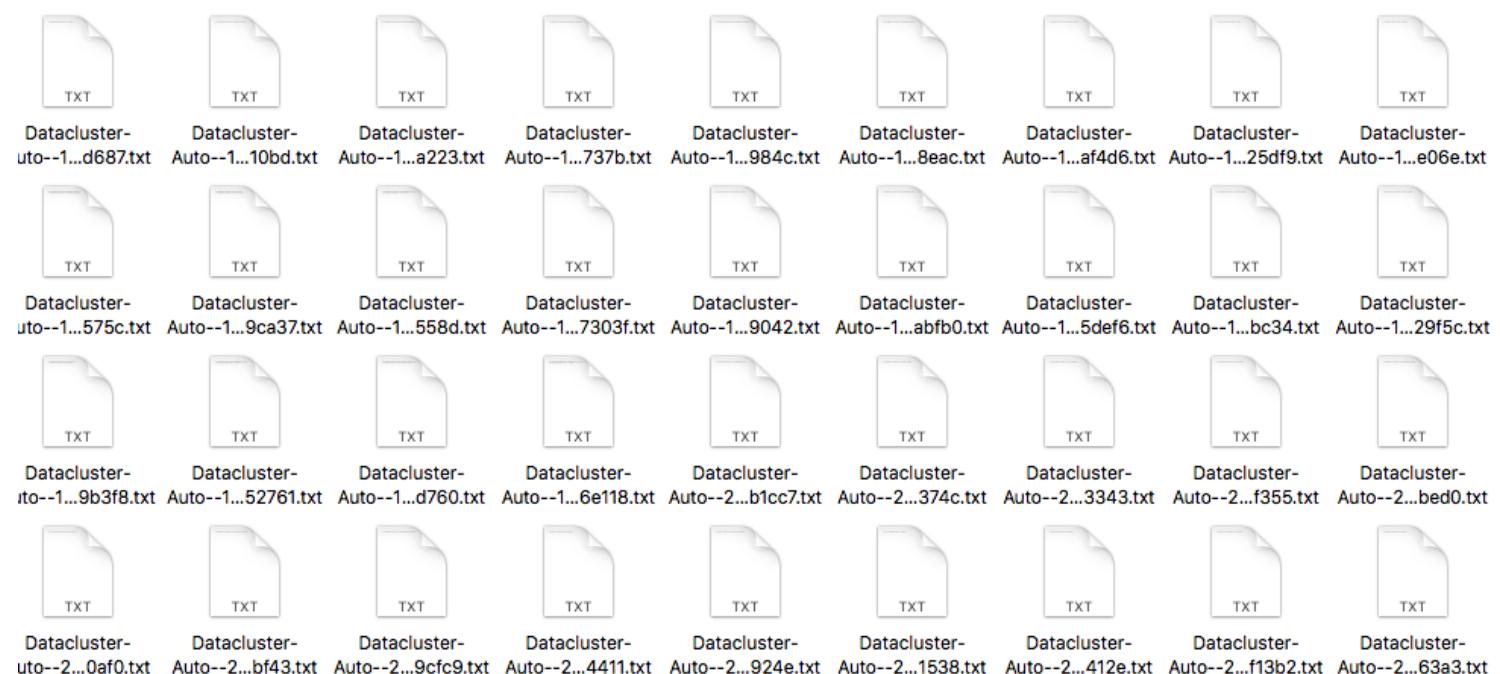
To train a YOLOv5 object detection model we'll training an auto detection model on our own dataset of auto using Colab and darknet

Our Data:

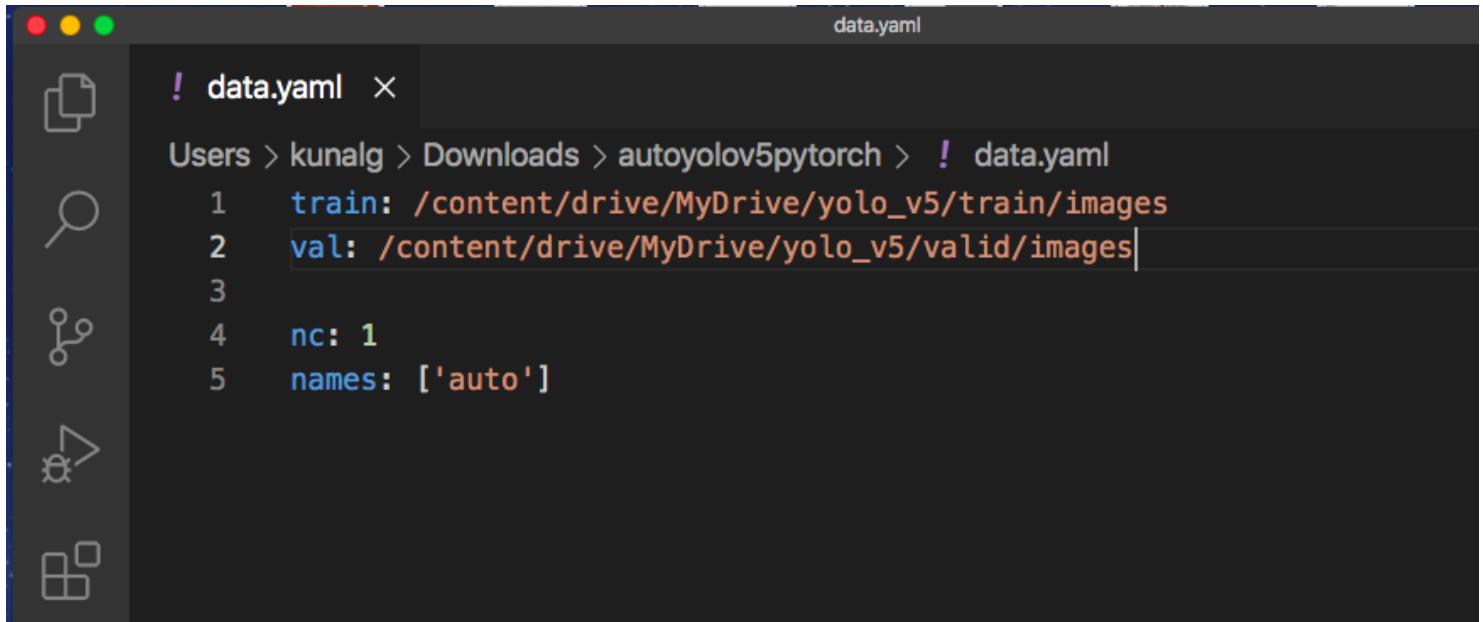
given Our dataset of 150 unannotated auto images.

ANNOTATING THE IMAGES IN YOLOV5 FORMAT

Separating the images annotation .txt file in one place:



giving the path of train and test images in the yaml file :



```
! data.yaml ×  
Users > kunalg > Downloads > autoyolov5pytorch > ! data.yaml  
1   train: /content/drive/MyDrive/yolo_v5/train/images  
2   val: /content/drive/MyDrive/yolo_v5/valid/images  
3  
4   nc: 1  
5   names: ['auto']
```

we have taken the help from roboflow site for annotating our images on yolov5 dataset.

YOLOv5 training

To train our detector we take the following steps:

- * Install YOLOv5 dependencies
- * Download custom YOLOv5 object detection data
- * Write our YOLOv5 Training configuration
- * Run YOLOv5 training
- * Evaluate YOLOv5 performance
- * Visualize YOLOv5 training data
- * Run YOLOv5 inference on test images
- * Export saved YOLOv5 weights for future inference

INSTALLING YOLOV5 DEPENDENCIES

we have cloned the github repository and imported torch

Install Dependencies

(Remember to choose GPU in Runtime if not already selected. Runtime → Change Runtime Type → Hardware accelerator → GPU)

```
[2] # clone YOLOv5 repository
!git clone https://github.com/ultralytics/yolov5  # clone repo
%cd yolov5
!git reset --hard 886f1c03d839575afecb059accf74296fad395b6

Cloning into 'yolov5'...
remote: Enumerating objects: 12310, done.
remote: Total 12310 (delta 0), reused 0 (delta 0), pack-reused 12310
Receiving objects: 100% (12310/12310), 11.51 MiB | 26.02 MiB/s, done.
Resolving deltas: 100% (8553/8553), done.
/content/yolov5
HEAD is now at 886f1c0 DDP after autoanchor reorder (#2421)

▶ # install dependencies as necessary
!pip install -qr requirements.txt  # install dependencies (ignore errors)
import torch

from IPython.display import Image, clear_output  # to display images
from utils.google_utils import gdrive_download  # to download models/datasets

# clear_output()
print('Setup complete. Using torch %s %s' % (torch.__version__, torch.cuda.get_device_properties(0))

[  |██████████| 596 kB 5.1 MB/s
Setup complete. Using torch 1.10.0+cu111 _CudaDeviceProperties(name='Tesla K80', major=3, minor=7,
```

Download custom YOLOv5 object detection data

running a python script to automatically download yolov5 weights:

```
$ download_weights.sh ×

Users > kunalg > Downloads > $ download_weights.sh

1  #!/bin/bash
2  # Download latest models from https://github.com/ultralytics/yolov5/releases
3  # Usage:
4  #     $ bash weights/download_weights.sh
5
6  python - <<EOF
7  from utils.google_utils import attempt_download
8
9  for x in ['s', 'm', 'l', 'x']:
10    attempt_download(f'yolov5{x}.pt')
11
12 EOF
13
```

Defining Model Configuration and Architecture

We will write a yaml script that defines the parameters for our model like the number of classes, anchors, and each layer.

```
[38] # define number of classes based on YAML
import yaml
with open("/content/gdrive/MyDrive/yolo_v5/data.yaml", 'r') as stream:
    num_classes = str(yaml.safe_load(stream)['nc'])

#this is the model configuration we will use for our tutorial
%cat /content/yolov5/models/yolov5s.yaml

# parameters
nc: 80 # number of classes
depth_multiple: 0.33 # model depth multiple
width_multiple: 0.50 # layer channel multiple

# anchors
anchors:
  - [10,13, 16,30, 33,23] # P3/8
  - [30,61, 62,45, 59,119] # P4/16
  - [116,90, 156,198, 373,326] # P5/32

# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]], # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]], # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]] # 2-P3/8
```

customizing the yaml file of yolov5 :

```
[6] #customize iPython writefile so we can write variables
from IPython.core.magic import register_line_cell_magic
```

```
@register_line_cell_magic
def writetemplate(line, cell):
    with open(line, 'w') as f:
        f.write(cell.format(**globals()))
```

```
▶ %%writetemplate /content/yolov5/models/custom_yolov5s.yaml
```

```
# parameters
nc: {num_classes} # number of classes
depth_multiple: 0.33 # model depth multiple
width_multiple: 0.50 # layer channel multiple

# anchors
anchors:
  - [10,13, 16,30, 33,23] # P3/8
  - [30,61, 62,45, 59,119] # P4/16
  - [116,90, 156,198, 373,326] # P5/32

# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]], # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]], # 1-P2/4
   [-1, 3, BottleneckCSP, [128]],
   [-1, 1, Conv, [256, 3, 2]], # 3-P3/8
   [-1, 9, BottleneckCSP, [256]],
   [-1, 1, Conv, [512, 3, 2]], # 4-P4/16
   [-1, 1, Conv, [512, 3, 2]], # 5-P5/32
   [-1, 1, Conv, [512, 3, 2]], # 6-P6/64
   [-1, 1, Conv, [512, 3, 2]]], # 7-P7/128
```

```

# YOLOv5 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]], # cat backbone P4
   [-1, 3, BottleneckCSP, [512, False]], # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]], # cat backbone P3
   [-1, 3, BottleneckCSP, [256, False]], # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]], # cat head P4
   [-1, 3, BottleneckCSP, [512, False]], # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]], # cat head P5
   [-1, 3, BottleneckCSP, [1024, False]], # 23 (P5/32-large)

   [[17, 20, 23], 1, Detect, [nc, anchors]], # Detect(P3, P4, P5)
  ]

```

Training Custom YOLOv5 Detector

Here, we are able to pass a number of arguments:

- ****img:**** define input image size
- ****batch:**** determine batch size
- ****epochs:**** define the number of training epochs.
- ****data:**** set the path to our yaml file
- ****cfg:**** specify our model configuration
- ****weights:**** specify a custom path to weights.
- ****name:**** result names
- ****nosave:**** only save the final checkpoint
- ****cache:**** cache images for faster training

```

▶ from google.colab import drive
drive.mount('/content/drive')

↳ Mounted at /content/drive
+ Code + Text

[23] # train yolov5s on custom data for 100 epochs
# time its performance
%%time
%cd /content/yolov5/
!python train.py --img 516 --batch 16 --epochs 100 --data "/content/drive/MyDrive/yolo_v5/data.yaml" --cfg ./models/c

```

/content/yolov5
github: ⚠ WARNING: code is out of date by 934 commits. Use 'git pull' to update or 'git clone <https://github.com/ul>
YOLOv5 v4.0-126-g886f1c0 torch 1.10.0+cu111 CUDA:0 (Tesla K80, 11441.1875MB)

Namespace(adam=False, batch_size=16, bucket='', cache_images=True, cfg='./models/custom_yolov5s.yaml', data='/content/drive/MyDrive/yolo_v5/data.yaml', epochs=100, img=516, lr=0.01, lr0=0.01, lr0f=0.2, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1)

	from	n	params	module	arguments
0	-1	1	3520	models.common.Focus	[3, 32, 3]
1	-1	1	18560	models.common.Conv	[32, 64, 3, 2]
2	-1	1	19904	models.common.BottleneckCSP	[64, 64, 1]
3	-1	1	73984	models.common.Conv	[64, 128, 3, 2]
4	-1	1	161152	models.common.BottleneckCSP	[128, 128, 3]
5	-1	1	295424	models.common.Conv	[128, 256, 3, 2]
6	-1	1	641792	models.common.BottleneckCSP	[256, 256, 3]
7	-1	1	1180672	models.common.Conv	[256, 512, 3, 2]

WARNING: --img-size 516 must be multiple of max stride 32, updating to 544
train: Scanning '/content/drive/MyDrive/yolo_v5/train/labels.cache' for images and labels... 428 found, 0 missing,
train: Caching images (0.3GB): 100% 428/428 [00:16<00:00, 26.65it/s]
val: Scanning '/content/drive/MyDrive/yolo_v5/valid/labels.cache' for images and labels... 42 found, 0 missing, 0 errors
val: Caching images (0.0GB): 100% 42/42 [00:02<00:00, 14.92it/s]
Plotting labels...

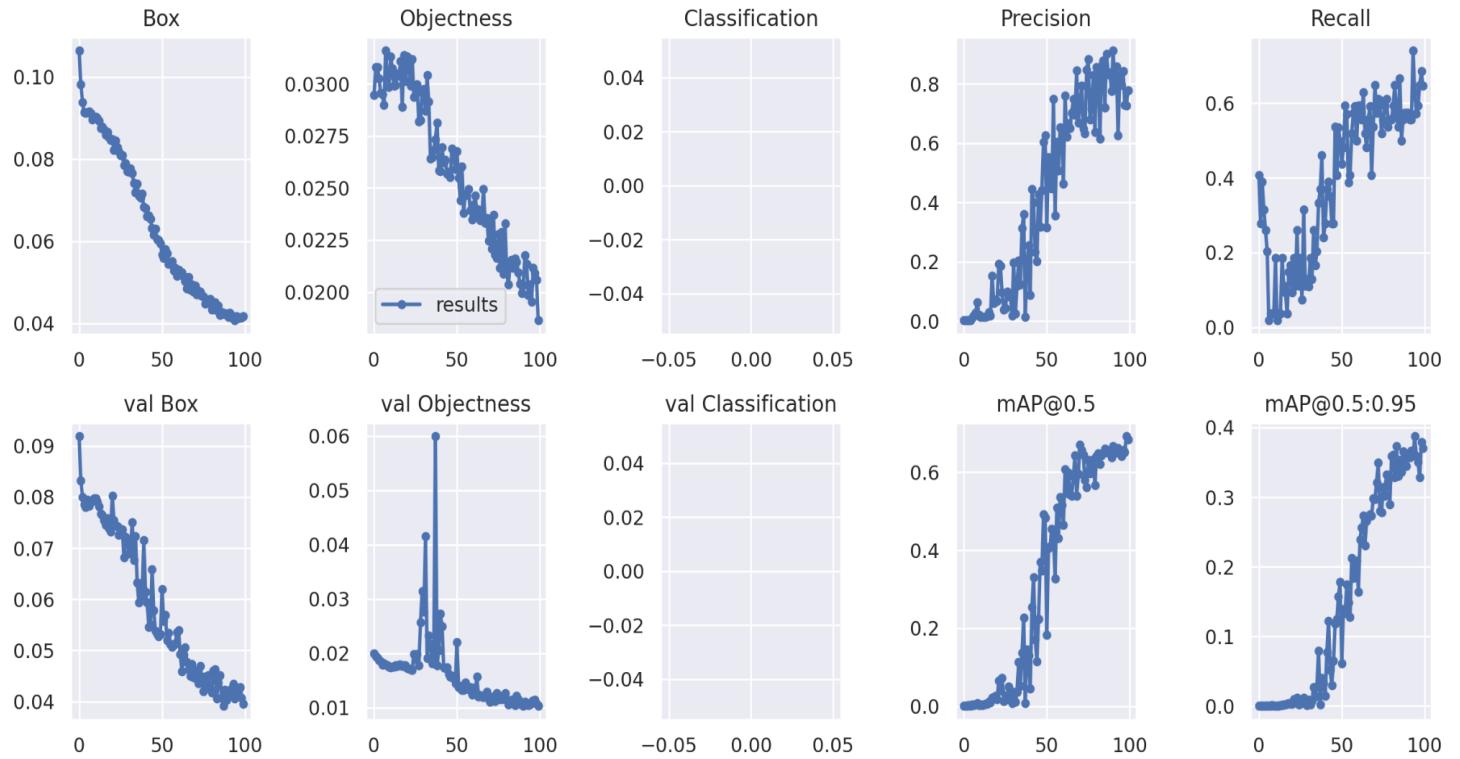
autoanchor: Analyzing anchors... anchors/target = 5.04, Best Possible Recall (BPR) = 1.0000
Image sizes 544 train, 544 test
Using 2 dataloader workers
Logging results to runs/train/yolov5s_results4
Starting training for 100 epochs...

Epoch	gpu_mem	box	obj	cls	total	targets	img_size
0/99	1.82G	0.1064	0.02947	0	0.1358	41	544: 100% 27/27 [00:43<00:00, 1.60s/it]
	Class	Images	Targets		P	R	mAP@.5: .95: 100% 2/2 [00:09<00:00]
	all		42	54	0.00175	0.407	0.00073 0.000114

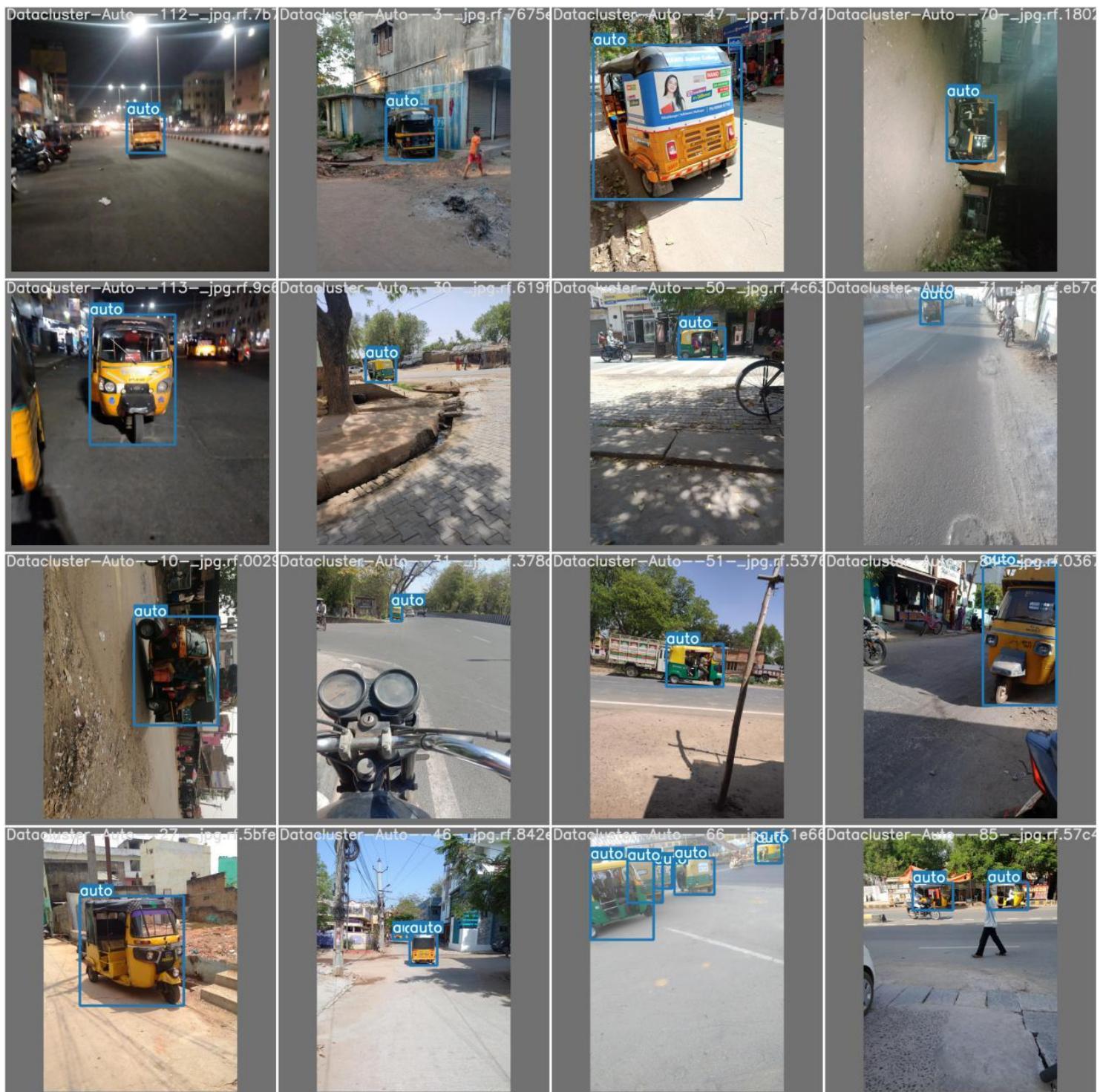
Epoch	gpu_mem	box	obj	cls	total	targets	img_size
1/99	2.25G	0.09821	0.03081	0	0.129	29	544: 100% 27/27 [00:23<00:00, 1.14it/it]
	Class	Images	Targets		P	R	mAP@.5: .95: 100% 2/2 [00:01<00:00]
	all		42	54	0.00239	0.278	0.00151 0.000239

Epoch	gpu_mem	box	obj	cls	total	targets	img_size
2/99	2.25G	0.09386	0.03081	0	0.1247	31	544: 100% 27/27 [00:23<00:00, 1.14it/it]
	Class	Images	Targets		P	R	mAP@.5: .95: 100% 2/2 [00:01<00:00]
	all		42	54	0.00235	0.389	0.00171 0.000301

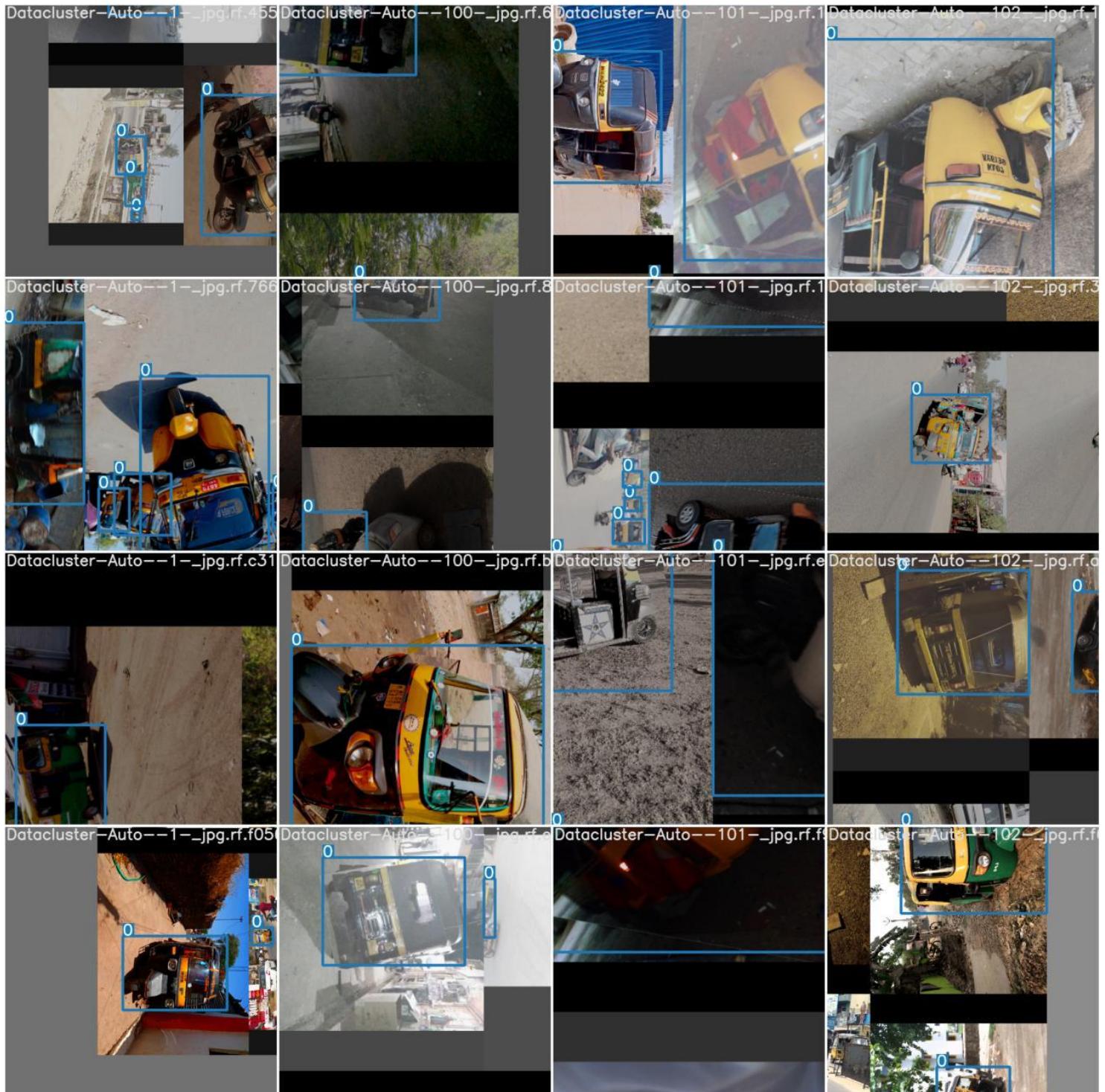
Evaluate Custom YOLOv5 Detector Performance through tensorboard



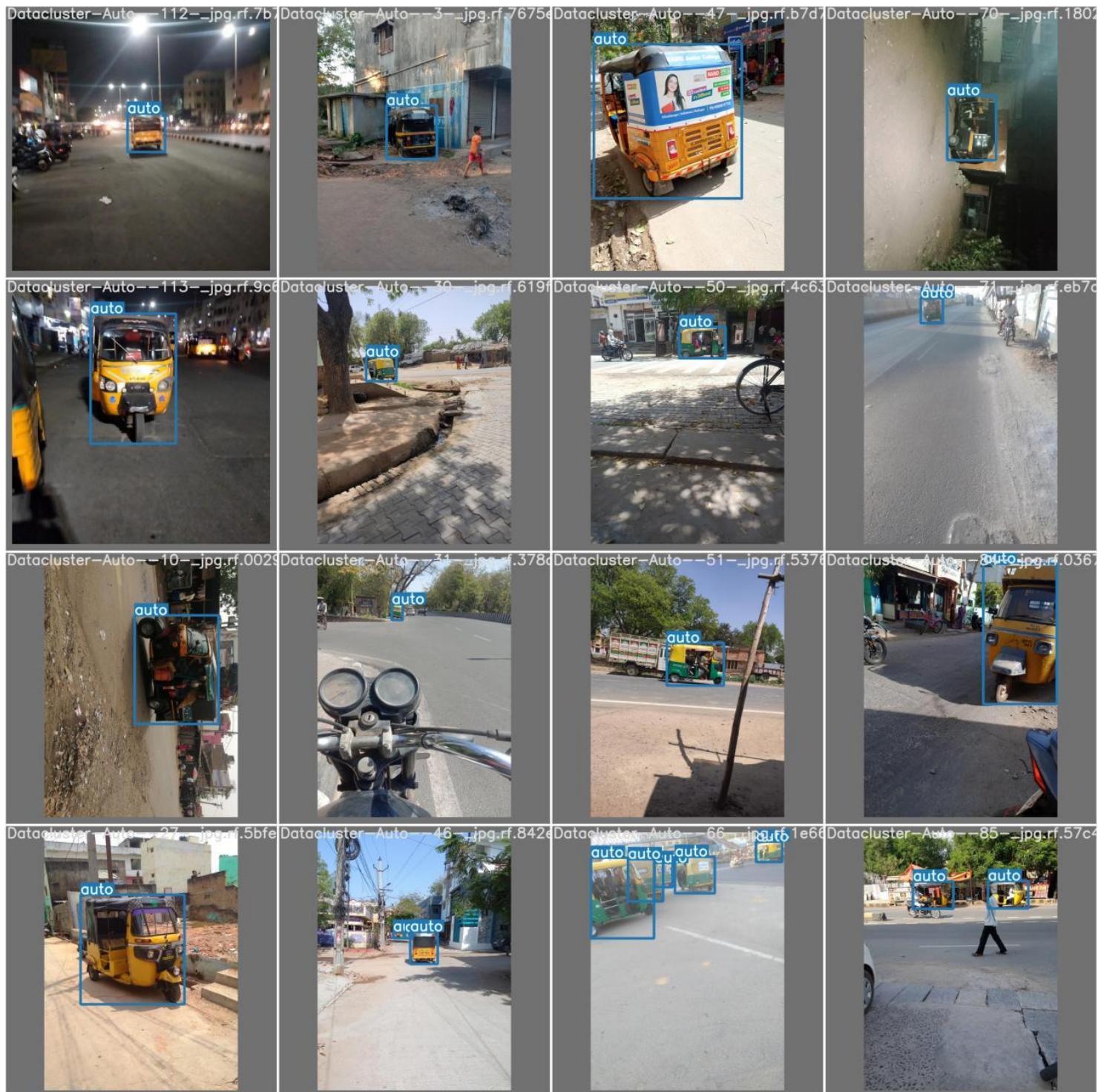
Visualizing our training model data with labels and augmentation



training images augmentation by the yolov5 algorithm



testing data :





Running inferences with trained weights

Run inference with a pretrained checkpoint on contents of `test/images` folder downloaded from Roboflow.

```
[28] # trained weights are saved by default in our weights folder
%ls runs/
detect/ train/

▶ %ls /content/yolov5/runs/train/yolov5s_results4/weights
└ best.pt last.pt

[31] 0.07 second inference time. That is 140 FPS on a TESLA P100!

/content/yolov5/runs/train/yolov5s_results4/weights/best.pt --img 84 --conf 0.4 --source "/content/drive/MyDrive/yolo

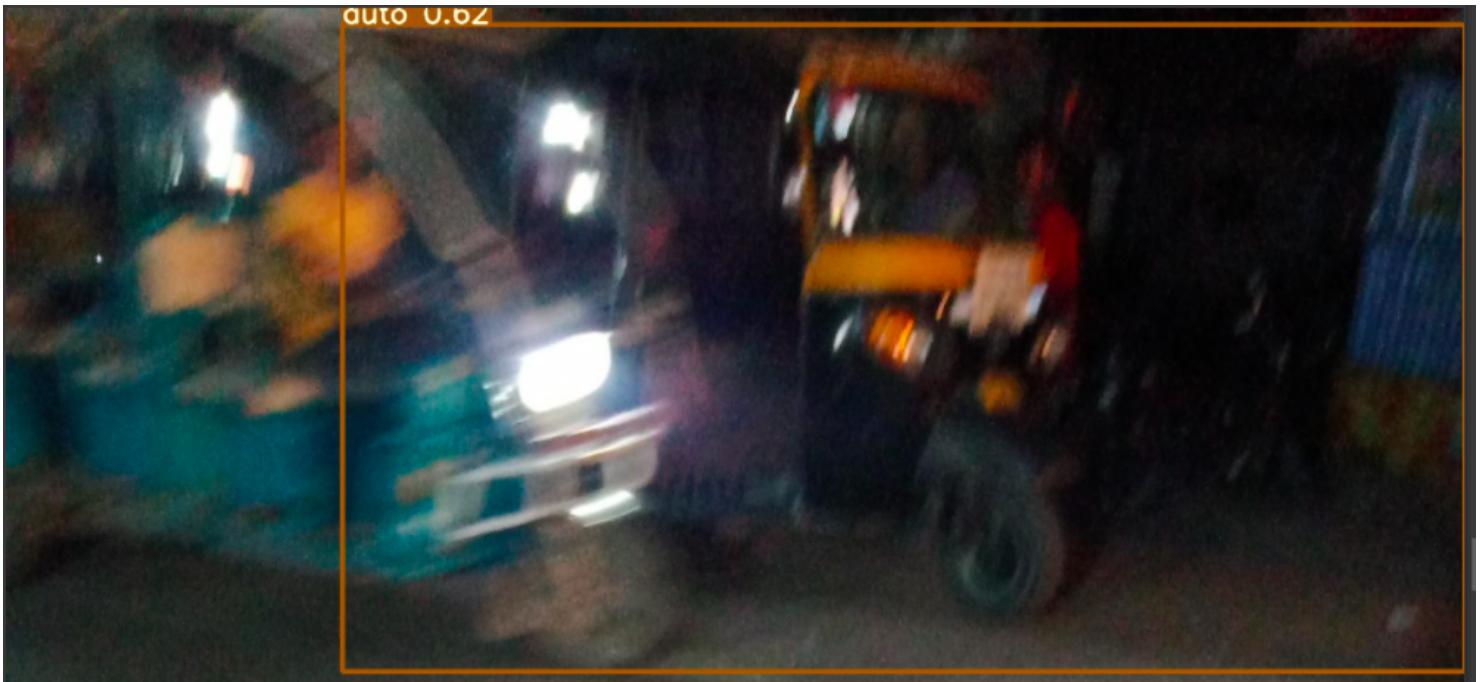
/content/yolov5
Namespace(agnostic_nms=False, augment=False, classes=None, conf_thres=0.4, device='', exist_ok=False, img_size=84, i
YOLOv5 v4.0-126-g886f1c0 torch 1.10.0+cu111 CUDA:0 (Tesla K80, 11441.1875MB)

Fusing layers...
/usr/local/lib/python3.7/dist-packages/torch/functional.py:445: UserWarning: torch.meshgrid: in an upcoming release,
    return _VF.meshgrid(tensors, **kwargs) # type: ignore[attr-defined]
Model Summary: 232 layers, 7246518 parameters, 0 gradients, 16.8 GFLOPS
WARNING: --img-size 84 must be multiple of max stride 32, updating to 96
  
```

Displaying inferences on the test data:







Exporting trained weights for future references

Export Trained Weights for Future Inference

Now we have trained your custom detector, you can export the trained weights you have made here for inference on your device elsewhere.

```
[34] from google.colab import drive
drive.mount('/content/gdrive')

Mounted at /content/gdrive

[35] %cp /content/yolov5/runs/train/yolov5s_results4/weights/best.pt /content/drive/MyDrive/yolo_v5
cp: failed to access '/content/drive/MyDrive/yolo_v5': Transport endpoint is not connected

[ ] /content/yolov5/runs/detect/exp3
```

Applications of YOLO

YOLO algorithm can be applied in the following fields:

- Autonomous driving: YOLO algorithm can be used in autonomous cars to detect objects around cars such as vehicles, people, and parking signals. Object detection in autonomous cars is done to avoid collision since no human driver is controlling the car.
- Wildlife: This algorithm is used to detect various types of animals in forests. This type of detection is used by wildlife rangers and journalists to identify animals in videos (both recorded and real-time) and images. Some of the animals that can be detected include giraffes, elephants, and bears.

Security: YOLO can also be used in security systems to enforce security in an area. Let's assume that people have been restricted from passing through a certain area for security reasons. If someone passes through the restricted area, the YOLO algorithm will detect him/her, which will require the security personnel to take further action.

Where YOLO detection can not be used

Disadvantages of YOLO:

- Comparatively low recall and more localization error compared to Faster R_CNN.
- Struggles to detect close objects because each grid can propose only 2 bounding boxes.
- Struggles to detect small objects.