

CIFAR-10 Image Classification

```

In [1]: #importing the libraries required
        ↗ used for data manipulation.

In [120]: import pandas as pd
          import numpy as np
          import matplotlib.pyplot as plt
          import seaborn
          ↗ used for plotting.

In [121]: #importing the dataset
          from keras.datasets import cifar10
          (X_train, y_train), (X_test, y_test) = cifar10.load_data()
          ↗ 3
          ↗ CIFAR-10 dataset is
          ↗ contained in the keras
          ↗ datasets along with other.

In [122]: #getting the shape of the training and test set variables
          X_train.shape
          ↗ divides the dataset
          ↗ into training subset and
          ↗ test subset.
          ↗ CIFAR-10 dataset contains
          ↗ 60,000 images of 10 different
          ↗ classes.

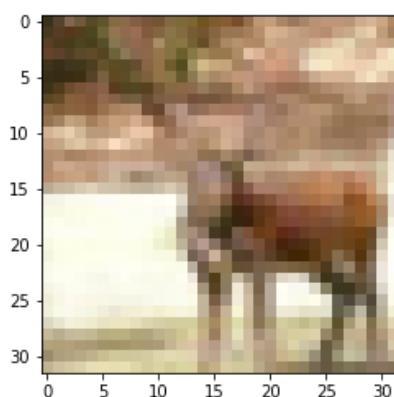
In [123]: X_test.shape
Out[123]: (10000, 32, 32, 3)

In [124]: y_train.shape
Out[124]: (50000, 1)

In [125]: y_test.shape
Out[125]: (10000, 1)

In [126]: #step 2 visualising the data and checking
          i = 1001
          plt.imshow(X_train[i])
          print(y_train[i])
          ↗ 3
          ↗ Checking the dataset by
          ↗ plotting an image randomly

```



In [127]:

```
#creating a matrix of pictures
W_grid = 15 → the size of the width of matrix.
L_grid = 15 → the size of the length of matrix.

fig, axes = plt.subplots(L_grid, W_grid, figsize = (25, 25))
axes = axes.ravel()
n_training = len(X_train) → input size. → Variable consisting of values of the

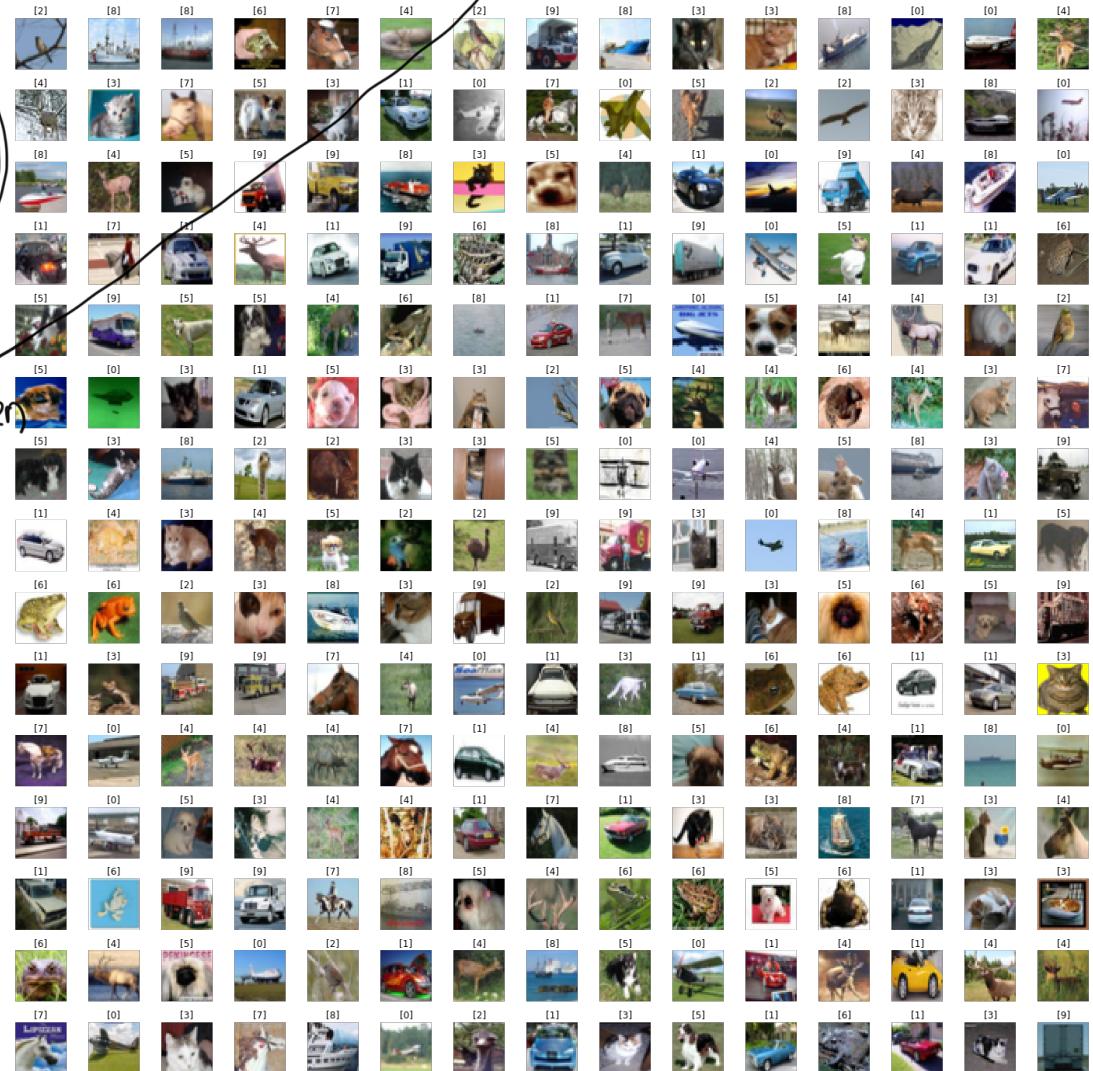
#creating a for loop to plot an image
for i in np.arange(0, L_grid * W_grid):
    index = np.random.randint(0, n_training)
    axes[i].imshow(X_train[index])
    axes[i].set_title(y_train[index])
    axes[i].axis('off')
plt.subplots_adjust(hspace = 0.4) → Prints out the matrix.
→ for loop to training set.
→ Plot the pictures.
→ Shows the index of the image to be plotted.
```

here we build a matrix which will contain the specified number of images.

turns off axis

take any random image from training Subdataset.

Set the category index as title.



In [128]: #step 3 data preparation

```
In [129]: #convert actual images into float type
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
number_cat = 10 #number of categories
```

```
In [130]: y_train
```

```
Out[130]: array([[6],
```

```
[9],
```

```
[9],
```

```
...,
```

```
[9],
```

```
[1],
```

```
[1]], dtype=uint8)
```

```
In [131]: import keras
```

```
#converting y decimal values into binaries
```

```
y_train = keras.utils.to_categorical(y_train, number_cat)
```

```
In [132]: y_train
```

```
Out[132]: array([[0., 0., 0., ..., 0., 0., 0.],
```

```
[0., 0., 0., ..., 0., 0., 1.],
```

```
[0., 0., 0., ..., 0., 0., 1.],
```

```
...,
```

```
[0., 0., 0., ..., 0., 0., 1.],
```

```
[0., 1., 0., ..., 0., 0., 0.],
```

```
[0., 1., 0., ..., 0., 0., 0.]], dtype=float32)
```

```
In [133]: y_test
```

```
Out[133]: array([[3],
```

```
[8],
```

```
[8],
```

```
...,
```

```
[5],
```

```
[1],
```

```
[7]])
```

```
In [134]: #converting y_test into binary values
```

```
y_test = keras.utils.to_categorical(y_test, number_cat)
```

```
In [135]: y_test
```

```
Out[135]: array([[0., 0., 0., ..., 0., 0., 0.],
```

```
[0., 0., 0., ..., 0., 1., 0.],
```

```
[0., 0., 0., ..., 0., 1., 0.],
```

```
...,
```

```
[0., 0., 0., ..., 0., 0., 0.],
```

```
[0., 1., 0., ..., 0., 0., 0.],
```

```
[0., 0., 0., ..., 1., 0., 0.]], dtype=float32)
```

```
In [136]: #normalising the values
```

```
X_train = X_train/255
```

```
X_test = X_test/255
```

```
In [137]: X_train
```

```
Out[137]: array([[[[0.23137255, 0.24313726, 0.24705882],  
[0.16862746, 0.18039216, 0.1764706 ],  
[0.19607843, 0.1882353 , 0.16862746],  
...,  
[0.61960787, 0.5176471 , 0.42352942],  
[0.59607846, 0.49019608, 0.4       ],  
[0.5803922 , 0.4862745 , 0.40392157]],  
[[[0.0627451 , 0.07843138, 0.07843138],  
[0.        , 0.        , 0.        ],  
[0.07058824, 0.03137255, 0.        ],  
...,  
[0.48235294, 0.34509805, 0.21568628],  
[0.46666667, 0.3254902 , 0.19607843],  
[0.47843137, 0.34117648, 0.22352941]],  
[[[0.09803922, 0.09411765, 0.08235294],  
[0.0627451 , 0.02745098, 0.        ],  
[0.19215687, 0.10588235, 0.03137255],  
...,  
[0.4627451 , 0.32941177, 0.19607843],  
[0.47058824, 0.32941177, 0.19607843],  
[0.42745098, 0.28627452, 0.16470589]],  
...,  
[[0.8156863 , 0.66666667 , 0.3764706 ],  
[0.7882353 , 0.6       , 0.13333334],  
[0.7764706 , 0.6313726 , 0.10196079],  
...,  
[0.627451 , 0.52156866, 0.27450982],  
[0.21960784, 0.12156863, 0.02745098],  
[0.20784314, 0.13333334, 0.07843138]],  
[[[0.7058824 , 0.54509807, 0.3764706 ],  
[0.6784314 , 0.48235294, 0.16470589],  
[0.7294118 , 0.5647059 , 0.11764706],  
...,  
[0.72156864, 0.5803922 , 0.36862746],  
[0.38039216, 0.24313726, 0.13333334],  
[0.3254902 , 0.20784314, 0.13333334]],  
[[[0.69411767, 0.5647059 , 0.45490196],  
[0.65882355, 0.5058824 , 0.36862746],  
[0.7019608 , 0.5568628 , 0.34117648],  
...,  
[0.84705883, 0.72156864, 0.54901963],  
[0.5921569 , 0.4627451 , 0.32941177],  
[0.48235294, 0.36078432, 0.28235295]]],  
[[[0.6039216 , 0.69411767, 0.73333335],  
[0.49411765, 0.5372549 , 0.53333336],  
[0.4117647 , 0.40784314, 0.37254903],  
...,  
[0.35686275, 0.37254903, 0.2784314 ],  
[0.34117648, 0.3529412 , 0.2784314 ],  
[0.30980393, 0.31764707, 0.27450982]],  
[[[0.54901963, 0.627451 , 0.6627451 ],  
[0.5686275 , 0.6       , 0.6039216 ],  
[0.49019608, 0.49019608, 0.4627451 ],  
...,
```

This prints out the total number of images contained and the dimensions of the image.

```
In [138]: X_train.shape  
Out[138]: (50000, 32, 32, 3)
```

Step for training the model.

```
In [139]: #getting the dimension of the image  
Input_shape = X_train.shape[1:]  
Input_shape  
Out[139]: (32, 32, 3)
```

Through this step we remove the number of images, it is filtering and only shows the dimensions of the image itself.

```
In [140]: #step 4 training the model  
from keras.models import Sequential  
from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D, Dense, Flatten, Dropout  
from keras.optimizers import Adam  
from keras.callbacks import TensorBoard
```

↳ required to create the neural network in a sequence.

```
In [141]: #building the model
```

```
In [142]: cnn_model = Sequential()  
cnn_model.add(Conv2D(filters = 32, kernel_size = (3,3), activation = 'relu', input_shape = Input_shape))  
cnn_model.add(Conv2D(filters = 32, kernel_size = (3, 3), activation = 'relu'))  
cnn_model.add(MaxPooling2D(2, 2))  
cnn_model.add(Dropout(0.3))
```

↳ to flatten out the image in 1D. Dropping the neurons.

↳ required for layer of convolution network, Pooling layer.

↳ the dimensions of the image. 3x3 matrix for computation.

```
In [143]: #adding extra convolution layers  
cnn_model.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))  
cnn_model.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))  
cnn_model.add(MaxPooling2D(2, 2))  
cnn_model.add(Dropout(0.2))
```

↳ Inputs the var containing no of filters. Activation function is relu.

↳ adding two more layers of neurons.

```
In [144]: #creating fully connected neurons  
cnn_model.add(Flatten())  
cnn_model.add(Dense(units = 512, activation = 'relu'))  
cnn_model.add(Dense(units = 512, activation = 'relu'))
```

↳ flattens the output image.

```
In [145]: #building the output layer
```

↳ neurons.

↳ activation

↳ number of neurons

```
In [146]: cnn_model.add(Dense(units = 10, activation = 'softmax'))
```

↳ activation function.

```
In [147]: cnn_model.compile(loss = 'categorical_crossentropy', optimizer = keras.optimizers.rmsprop(lr = 0.001), metrics = ['accuracy'])
```

Output part. It should give an output as classifying or predicting the class of the input image by recognising and analysing it.

↳ learning rate of the model.

↳ the parameter on which the model will be judged.

Compiling the completed model.

give compilation history of the model.

the no of images which will be fed at once.

In [149]: `history = cnn_model.fit(X_train, y_train, batch_size = 32, epochs = 2, shuffle = True)`

Epoch 1/2

50000/50000 [=====] - 176s 4ms/step - loss: 1.5488

- acc: 0.4400

Epoch 2/2

50000/50000 [=====] - 179s 4ms/step - loss: 1.1269

- acc: 0.6048

In [150]: `#step 5 evaluating the model created`

In [151]: `evaluation = cnn_model.evaluate(X_test, y_test)`
`print('Test Accuracy: {}'.format(evaluation[1]))`

10000/10000 [=====] - 10s 951us/step

Test Accuracy: 0.6648

the input parameters doesn't make a pattern.

Evaluation of the model.

print) the test accuracy .

In [152]: `predicted_classes = cnn_model.predict_classes(X_test)`
`predicted_classes`

Out[152]: `array([3, 8, 8, ..., 5, 0, 7])`

In [153]: `y_test`

Out[153]: `array([[0., 0., 0., ..., 0., 0., 0.],`
`[0., 0., 0., ..., 0., 1., 0.],`
`[0., 0., 0., ..., 0., 1., 0.],`
`...,`
`[0., 0., 0., ..., 0., 0., 0.],`
`[0., 1., 0., ..., 0., 0., 0.],`
`[0., 0., 0., ..., 1., 0., 0.]], dtype=float32)`

In [154]: `y_test = y_test.argmax(1)`

In [155]: `y_test`

Out[155]: `array([3, 8, 8, ..., 5, 1, 7])`

Checking the results are compiled successfully by predicting the class of the input test subdataset.

Creating a matrix plotting few images.

```
In [156]: L = 7
W = 7
fig, axes = plt.subplots(L, W, figsize = (12, 12))
axes = axes.ravel()

for i in np.arange(0, L*W):
    axes[i].imshow(X_test[i])
    axes[i].set_title('Prediction = {} \n True = {}'.format(predicted_classes[i], y_test[i]))
    axes[i].axis('off')

plt.subplots_adjust(wspace = 1)
```

Length of the matrix

Width of the matrix.

for loop which will plot the image in the matrix

Shows the test images in the test subdataset .

plots the axes of the matrix.

Creates the matrix .

range

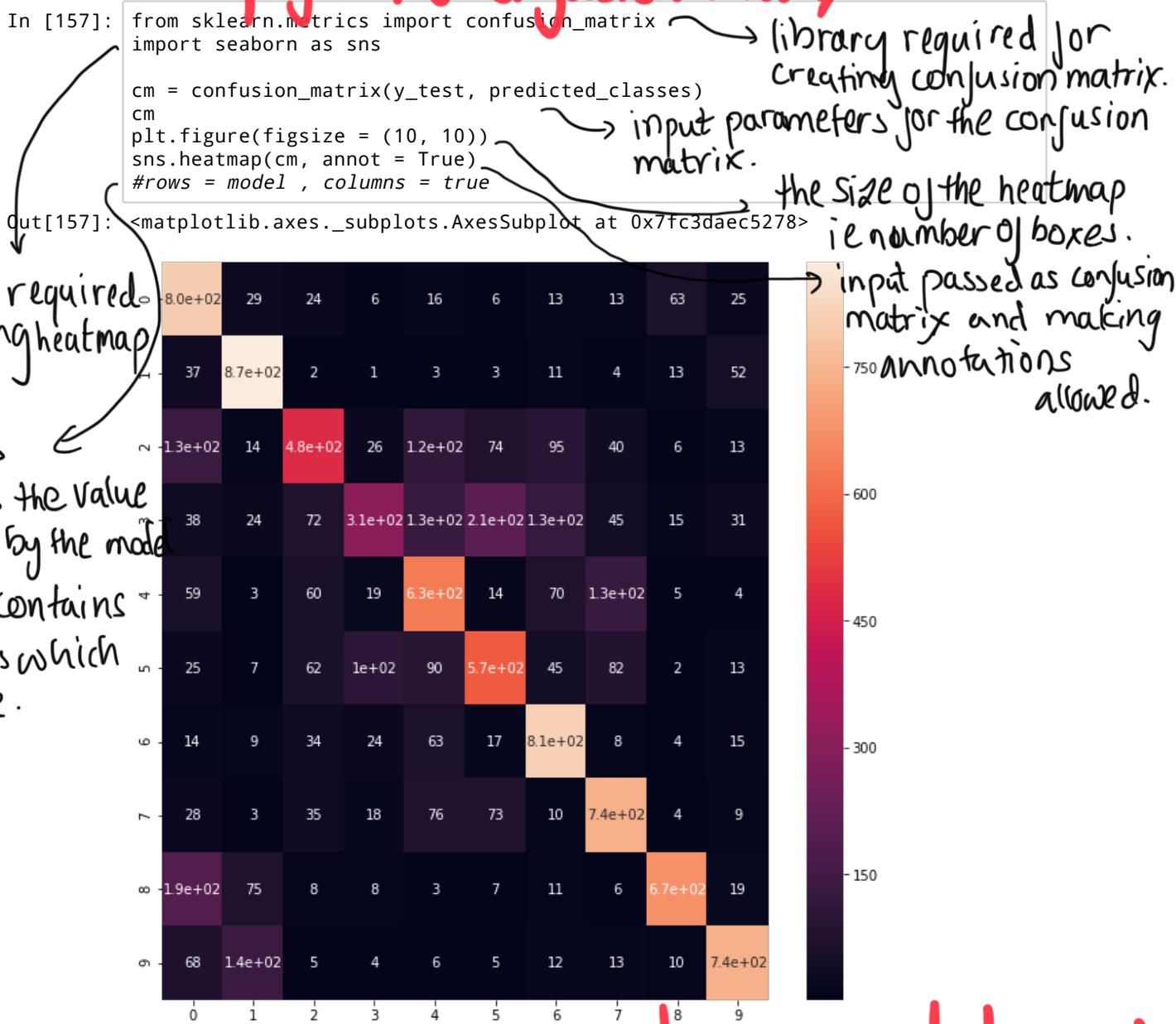
Sets a title which shows the class predicted by the model and the true class .

will remove the axis.

will provide spacing between pictures.

Prediction = 3 True = 3	Prediction = 8 True = 8	Prediction = 8 True = 8	Prediction = 0 True = 0	Prediction = 6 True = 6	Prediction = 6 True = 6	Prediction = 9 True = 1
						
Prediction = 6 True = 6	Prediction = 3 True = 3	Prediction = 1 True = 1	Prediction = 4 True = 0	Prediction = 9 True = 9	Prediction = 4 True = 5	Prediction = 7 True = 7
						
Prediction = 9 True = 9	Prediction = 6 True = 8	Prediction = 5 True = 5	Prediction = 7 True = 7	Prediction = 8 True = 8	Prediction = 6 True = 6	Prediction = 7 True = 7
						
Prediction = 0 True = 0	Prediction = 0 True = 4	Prediction = 9 True = 9	Prediction = 4 True = 5	Prediction = 5 True = 2	Prediction = 4 True = 4	Prediction = 0 True = 0
						
Prediction = 9 True = 9	Prediction = 6 True = 6	Prediction = 6 True = 6	Prediction = 5 True = 5	Prediction = 2 True = 4	Prediction = 3 True = 5	Prediction = 9 True = 9
						
Prediction = 3 True = 2	Prediction = 7 True = 4	Prediction = 1 True = 1	Prediction = 9 True = 9	Prediction = 5 True = 5	Prediction = 0 True = 4	Prediction = 6 True = 6
						
Prediction = 5 True = 5	Prediction = 6 True = 6	Prediction = 0 True = 0	Prediction = 9 True = 9	Prediction = 3 True = 3	Prediction = 7 True = 9	Prediction = 7 True = 7
						

Heatmap for the Confusion matrix.



In [158]: #step 6 saving the model

In [159]: import os

```
directory = os.path.join(os.getcwd(), 'saved_models')
if not os.path.isdir(directory):
    os.makedirs(directory)
model_path = os.path.join(directory, 'keras_CIFAR-10_trained_model.h5')
cnn_model.save(model_path)
```

In [113]: #using augmentation to increase the accuracy of the model
#augmentation means we are increasing the variations in the images by enlarging them rotating them etc

Saving the Completed model as h5 ext.

→ creates a directory

→ checks if directory exists or not.

→ creates a path for the directory.

→ Final step Saving the model -

Getting more accuracy by augmentation of images

```
In [114]: import keras
from keras.datasets import cifar10
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
```

} importing the dataset .
} Converting into decimal.

```
In [115]: X_train.shape
```

```
Out[115]: (50000, 32, 32, 3)
```

```
In [116]: n = 8
```

```
X_train_samples = X_train[:n]
X_train_samples.shape
```

```
Out[116]: (8, 32, 32, 3)
```

```
In [70]:
```

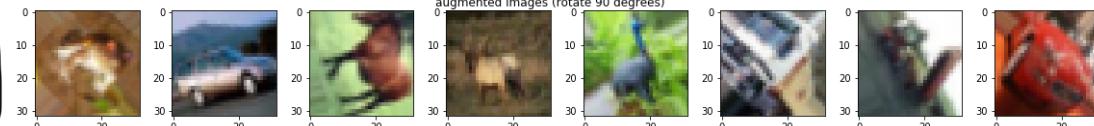
```
from keras.preprocessing.image import ImageDataGenerator
datagen_train = ImageDataGenerator(rotation_range = 90)
#datagen_train = ImageDataGenerator(vertical_flip = True)
#datagen_train = ImageDataGenerator(height_shift_range = 0.5)
#datagen_train = ImageDataGenerator(brightness_range = (1, 5))
datagen_train.fit(X_train)
```

} fitting the dataset with new changes.

```
from PIL import Image
```

```
fig = plt.figure(figsize = (20,2))
for x_batch in datagen_train.flow(X_train_samples, batch_size = n):
    for i in range(0, n):
        ax = fig.add_subplot(1, n, i+1)
        ax.imshow(Image.fromarray(np.uint8(x_batch[i])))
fig.suptitle('augmented images (rotate 90 degrees)')
plt.show()
break;
```

} Size of the figure } number of images takes a input at a time.
} training samples } for loop
} gives a subtitle } for plotting image and changing aspects -



pillow library required for image manipulation and plotting.

plotting and incrementing

Inputs the image

```
In [68]: # step 7.2 model training using augmented dataset
```

```
In [168]: from keras.preprocessing.image import ImageDataGenerator
```

```
datagen = ImageDataGenerator(
```

```
{ rotation_range = 90,
width_shift_range = 0.1,
horizontal_flip = True,
vertical_flip = True
})
```

```
In [169]: datagen.fit(X_train)
```

} fits the dataset.

Parameters on which the image will be changed or altered.

} When executing this cell only till step 6 and not step 7.1

number of epochs.

In [172]: `cnn_model.fit_generator(datagen.flow(X_train, y_train, batch_size = 32), epochs = 2, steps_per_epoch = 10000)`

```
Epoch 1/2
10000/10000 [=====] - 1029s 103ms/step - loss: 1.53
34 - acc: 0.4553
Epoch 2/2
10000/10000 [=====] - 1013s 101ms/step - loss: 1.46
35 - acc: 0.4869
```

Out[172]: <keras.callbacks.History at 0x7fc3dae42940>

In [176]: `score = cnn_model.evaluate(X_test, y_test)
print('Test accuracy', score[10])`

```
ValueError
<ipython-input-176-0540f0fd44ad> in <module>
----> 1 score = cnn_model.evaluate(X_test, y_test)
      2 print('Test accuracy', score[10])
```

→ Minibatch out the accuracy of the model.

```
/usr/local/lib/python3.5/dist-packages/keras/engine/training.py in evaluate(self, x, y, batch_size, verbose, sample_weight, steps)
    1100         x, y,
    1101         sample_weight=sample_weight,
-> 1102             batch_size=batch_size)
    1103     # Prepare inputs, delegate logic to `test_loop`.
    1104     if self._uses_dynamic_learning_phase():
```

```
/usr/local/lib/python3.5/dist-packages/keras/engine/training.py in _standardize_user_data(self, x, y, sample_weight, class_weight, check_array_lengths, batch_size)
    787         feed_output_shapes,
    788         check_batch_axis=False, # Don't enforce the batch size.
-> 789         exception_prefix='target')
    790
    791     # Generate sample-wise weight values given the `sample_w
```

```
/usr/local/lib/python3.5/dist-packages/keras/engine/training_utils.py in standardize_input_data(data, names, shapes, check_batch_axis, exception_prefix)
    136             ': expected ' + names[i] + ' to have shape '
    137             str(shape) + ' but got array with shape '
-> 138             str(data_shape))
    139     return data
    140
```

ValueError: Error when checking target: expected dense_9 to have shape (10,)
but got array with shape (1,)

In [174]: `# save the model
directory = os.path.join(os.getcwd(), 'saved_models')

if not os.path.isdir(directory):
 os.makedirs(directory)
model_path = os.path.join(directory, 'keras_cifar10_trained_model_Augmentation.h5')
cnn_model.save(model_path)`

Saving the created
model as the previous
model was saved.

In []: