Image classification using deep learning:

Image classification is a fascinating deep learning project. Specifically, image classification comes under the computer vision project category.

In this project, we will build a convolution neural network in Keras with python on a CIFAR-10 dataset. First, we will explore our dataset, and then we will train our neural network using python and Keras.

What is Image Classification

- The classification problem is to categorize all the pixels of a digital image into one of the defined classes.
- Image classification is the most critical use case in digital image analysis.
- Image classification is an application of both supervised classification and unsupervised classification.
  - In supervised classification, we select samples for each target class. We train our neural network on these target class samples and then classify new samples.
  - In unsupervised classification, we group the sample images into clusters of images having similar properties. Then, we classify each cluster into our intended classes.

About Image Classification Dataset

CIFAR-10 is a very popular computer vision dataset. This dataset is well studied in many types of deep learning research for object recognition.

This dataset consists of 60,000 images divided into 10 target classes, with each category containing 6000 images of shape 32*32. This dataset contains images of low resolution (32*32), which allows researchers to try new algorithms. The 10 different classes of this dataset are:

1. Airplane
2. Car
3. Bird
4. Cat
5. Deer
6. Dog
7. Frog
8. Horse
9. Ship
10. Truck

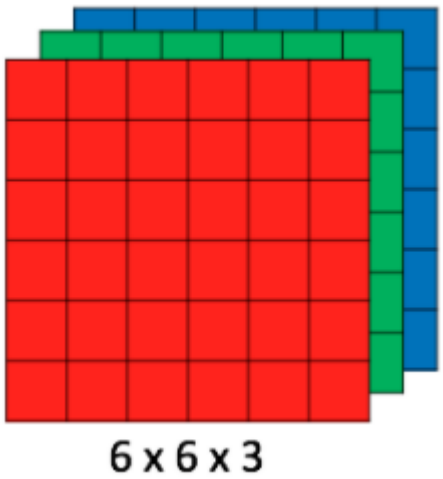
CIFAR-10 dataset is already available in the datasets module of Keras. We do not need to download it; we can directly import it from keras.datasets.

CNN ALGORITHM

Convolutional neural network:

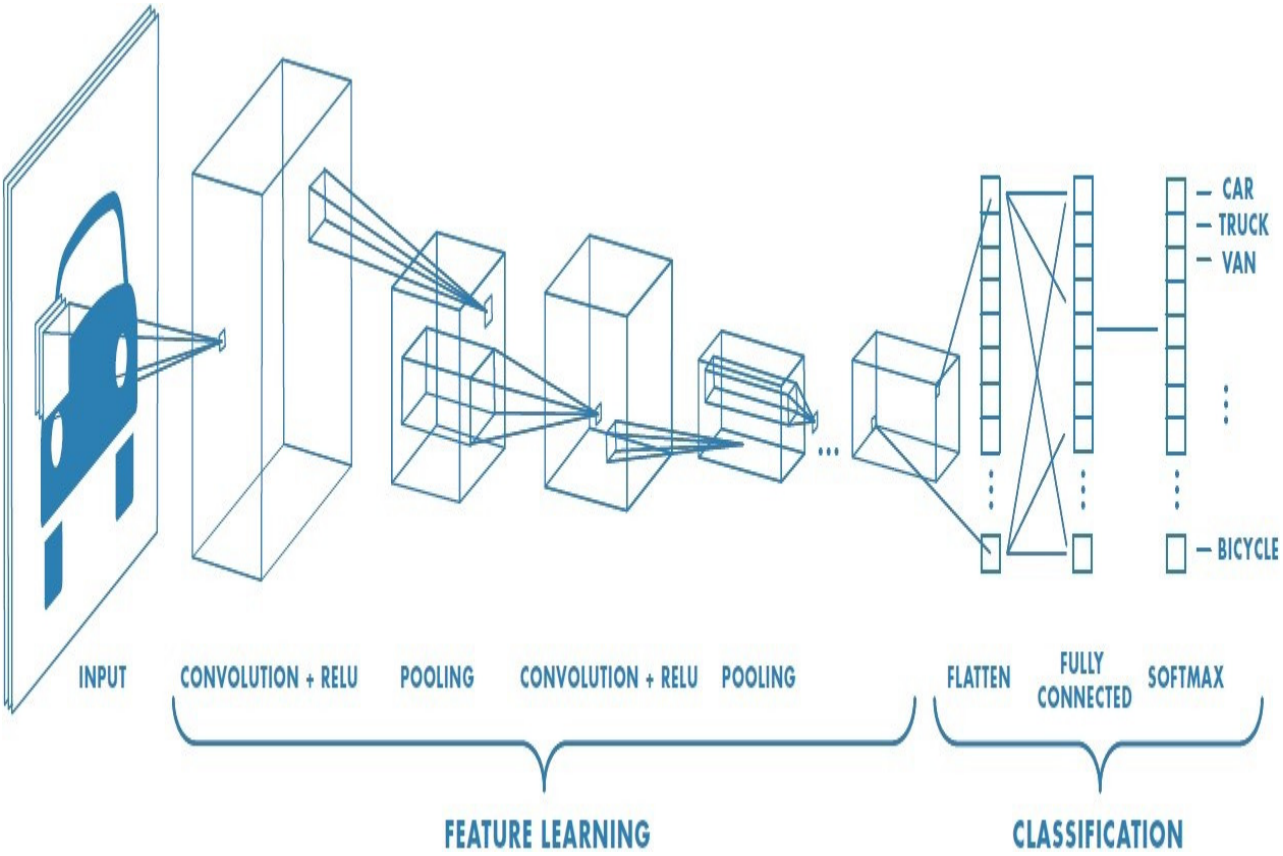
Convolutional neural network is one of the main categories to do images recognition, images classifications. Objects detections, recognition faces etc., are some of the areas where CNNs are widely used.

CNN image classifications takes an input image, process it and classify it under certain categories (Eg., Dog, Cat, Tiger, Lion). Computers sees an input image as array of pixels and it depends on the image resolution. Based on the image resolution, it will see h x w x d(h = Height, w = Width, d = Dimension). Eg., An image of 6 x 6 x 3 array of matrix of RGB (3 refers to RGB values) and an image of 4 x 4 x 1 array of matrix of grayscale image.



Array of RGB Matrix

Technically, deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernals), Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1. The below figure is a complete flow of CNN to process an input image and classifies the objects based on values.



Neural network with many convolutional layers

Convolution Layer

Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernal

- An image matrix (volume) of dimension **(h x w x d)**
- A filter **(f_h x f_w x d)**
- Outputs a volume dimension **(h - f_h + 1) x (w - f_w + 1) x 1**

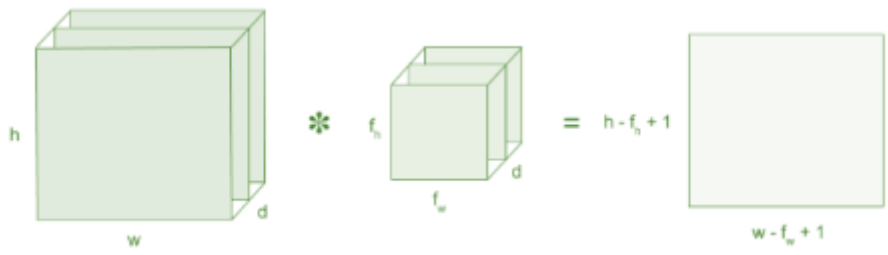
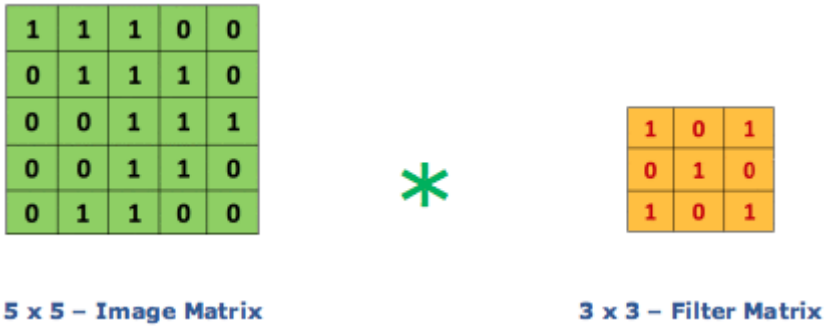


Image matrix multiplies kernel or filter matrix

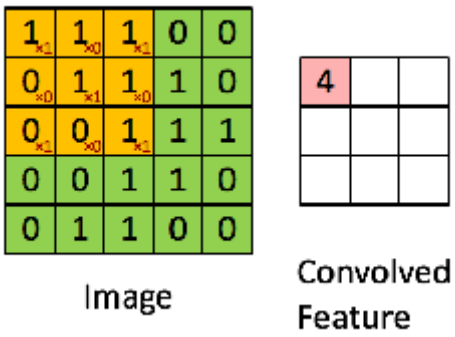
Consider a 5 x 5 whose image pixel values are 0, 1 and filter matrix 3 x 3 as shown in below



5 x 5 – Image Matrix 3 x 3 – Filter Matrix

Image matrix multiplies kernel or filter matrix

Then the convolution of 5 x 5 image matrix multiplies with 3 x 3 filter matrix which is called **"Feature Map"** as output shown in below



3 x 3 Output matrix

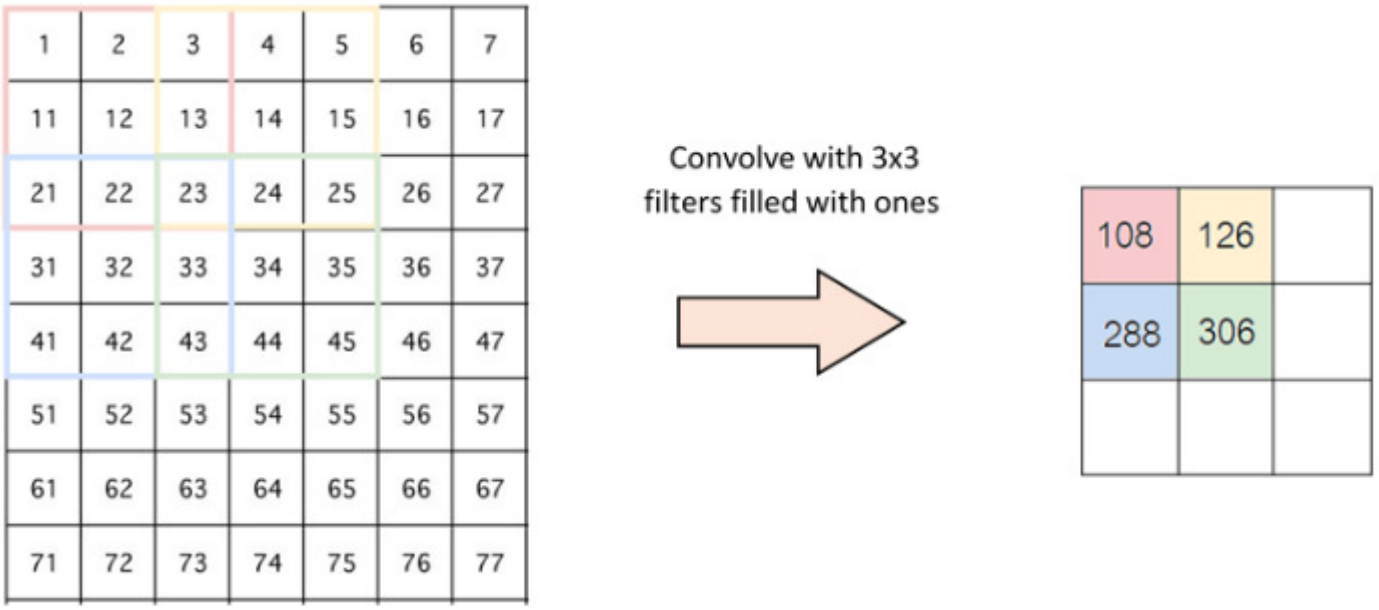
Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters. The below example shows various convolution image after applying different types of filters (Kernels).

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Some common filters

Strides

Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on. The below figure shows convolution would work with a stride of 2.



Stride of 2 pixels

Padding

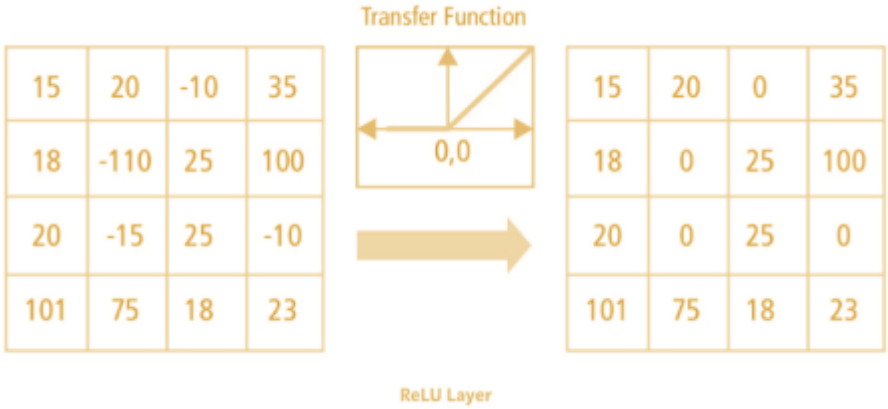
Sometimes filter does not fit perfectly fit the input image. We have two options:

- Pad the picture with zeros (zero-padding) so that it fits
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.

Non Linearity (ReLU)

ReLU stands for Rectified Linear Unit for a non-linear operation. The output is ***f(x) = max(0,x).***

Why ReLU is important :ReLU's purpose is to introduce non-linearity in our ConvNet. Since, the real world data would want our ConvNet to learn would be non-negative linear values.



ReLU operation

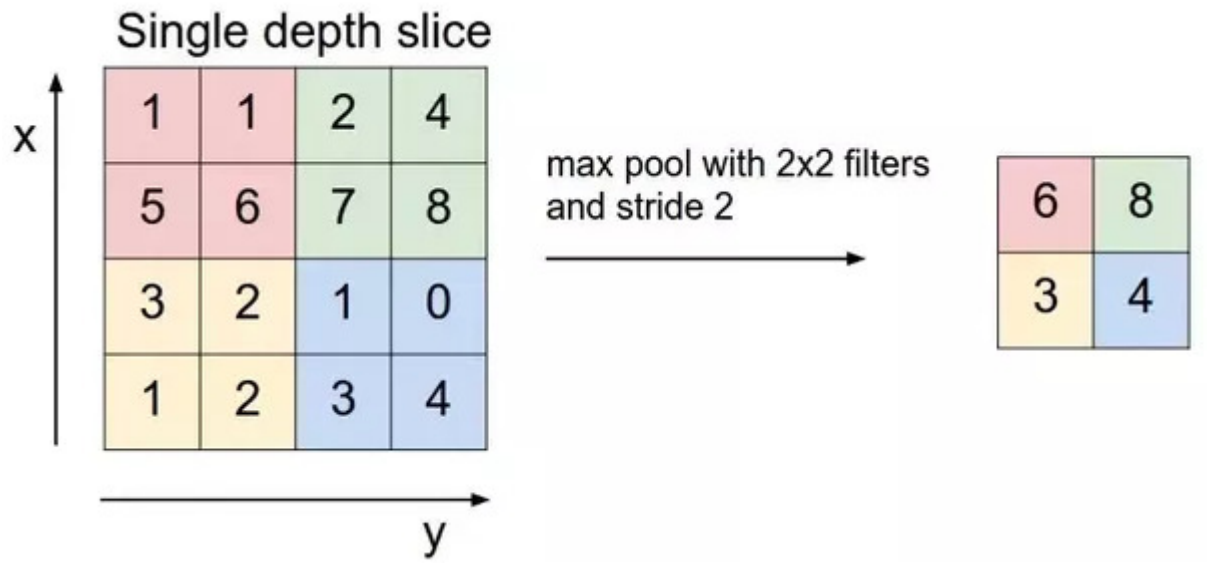
There are other nonlinear functions such as tanh or sigmoid can also be used instead of ReLU. Most of the data scientists uses ReLU since performance wise ReLU is better than other two.

Pooling Layer

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or downsampling which reduces the dimensionality of each map but retains the important information. Spatial pooling can be of different types:

- Max Pooling
- Average Pooling
- Sum Pooling

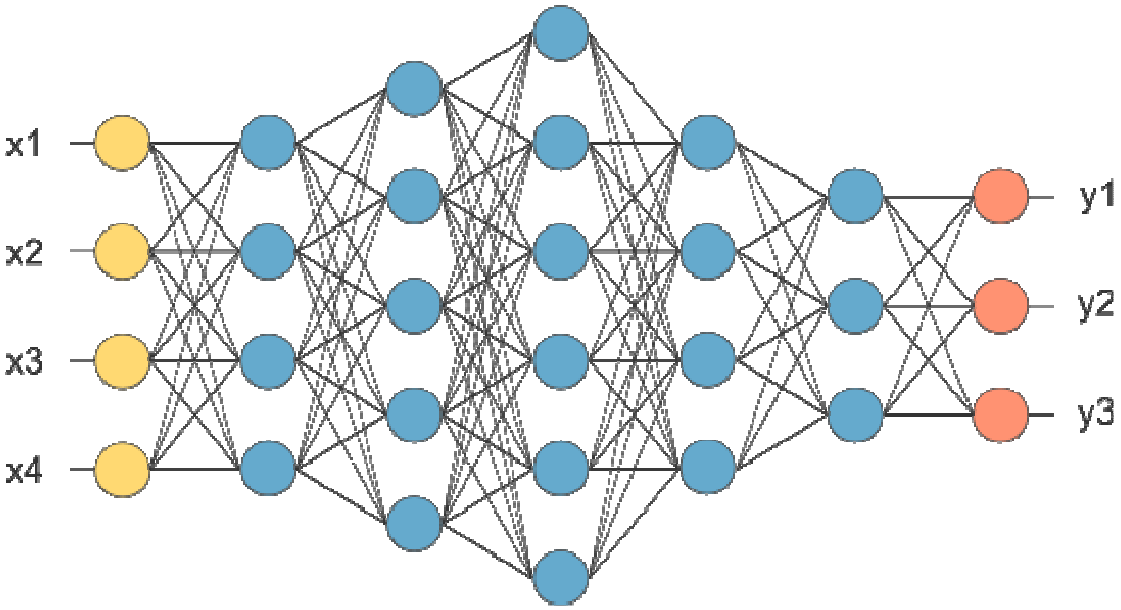
Max pooling take the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.



Max Pooling

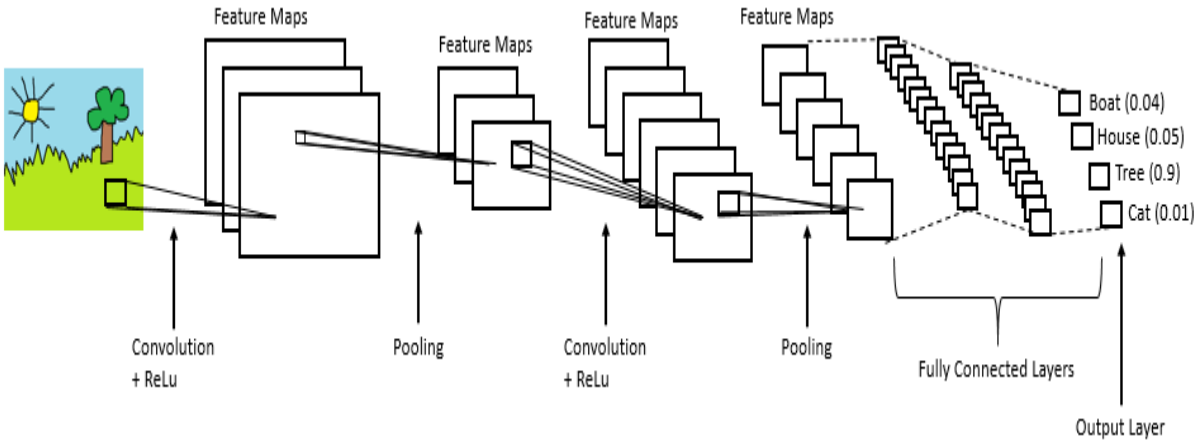
Fully Connected Layer

The layer we call as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like neural network.



After pooling layer, flattened as FC layer

In the above diagram, feature map matrix will be converted as vector (x1, x2, x3, …). With the fully connected layers, we combined these features together to create a model. Finally, we have an activation function such as softmax or sigmoid to classify the outputs as cat, dog, car, truck etc.,



Important points

- Provide input image into convolution layer
- Choose parameters, apply filters with strides, padding if requires. Perform convolution on the image and apply ReLU activation to the matrix.
- Perform pooling to reduce dimensionality size
- Add as many convolutional layers until satisfied
- Flatten the output and feed into a fully connected layer (FC Layer)
- Output the class using an activation function (Logistic Regression with cost functions) and classifies images.

Load the dataset from keras datasets module

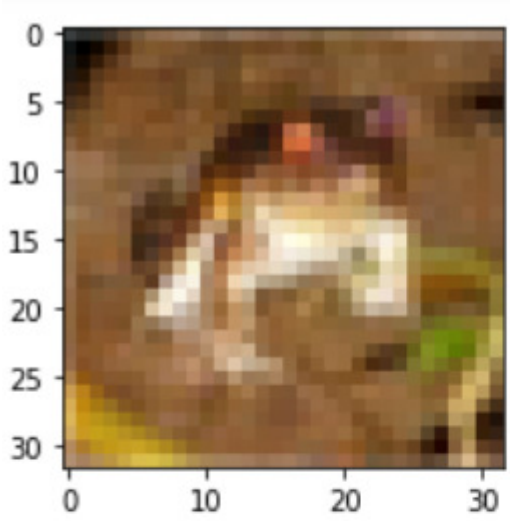
```
#importing libraries
from keras.datasets import cifar10
import matplotlib.pyplot as plt

(train_X,train_Y),(test_X,test_Y)=cifar10.load_data()
```

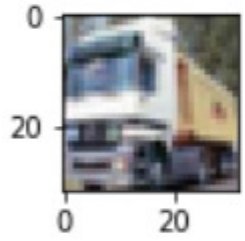
```
Using TensorFlow backend.
```

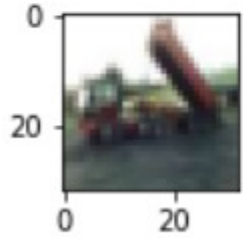
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 31s 0us/step
```

Plot some images from the dataset to visualize the dataset

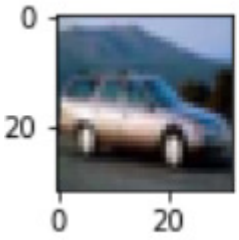
```
#display images
n=6
plt.figure(figsize=(20,10))
for i in range(n):
    plt.subplot(330+1+i)
    plt.imshow(train_X[i])
    plt.show()
```

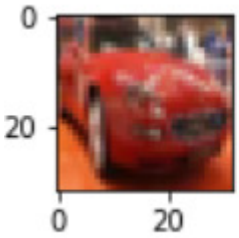

0
5
10
15
20
25
30
0
10
20
30


0
20
0
20


0
20
0
20


0
20





Import the required layers and modules to create our convolution neural net architecture

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
```

Convert the pixel values of the dataset to float type and then normalize the dataset

```
#tarin and test
train_x=train_X.astype('float32')
test_X=test_X.astype('float32')

train_X=train_X/255.0
test_X=test_X/255.0
```

Now perform the one-hot encoding for target classes

```
train_Y=np_utils.to_categorical(train_Y)
test_Y=np_utils.to_categorical(test_Y)

num_classes=test_Y.shape[1]
```

```
model=Sequential()
model.add(Conv2D(32,(3,3),input_shape=(32,32,3),
    padding='same',activation='relu',
    kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(32,(3,3),activation='relu',padding='same',kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(512,activation='relu',kernel_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

Create the sequential model and add the layers

```
sgd=SGD(lr=0.01,momentum=0.9,decay=(0.01/25),nesterov=False)

model.compile(loss='categorical_crossentropy',
  optimizer=sgd,
  metrics=['accuracy'])
```

Configure the optimizer and compile the model

```
model.fit(train_X,train_Y,
    validation_data=(test_X,test_Y),
    epochs=10,batch_size=32)
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 411s 8ms/step - loss: 1.7157 - accuracy: 0.3736 - val_loss: 1.3603 - val_accurac
y: 0.5186
Epoch 2/10
50000/50000 [==============================] - 371s 7ms/step - loss: 1.3283 - accuracy: 0.5241 - val_loss: 1.2004 - val_accurac
y: 0.5652
Epoch 3/10
50000/50000 [==============================] - 361s 7ms/step - loss: 1.1548 - accuracy: 0.5874 - val_loss: 1.0941 - val_accurac
y: 0.6116
Epoch 4/10
50000/50000 [==============================] - 364s 7ms/step - loss: 1.0303 - accuracy: 0.6341 - val_loss: 1.0088 - val_accurac
y: 0.6396
Epoch 5/10
50000/50000 [==============================] - 366s 7ms/step - loss: 0.9284 - accuracy: 0.6710 - val_loss: 0.9728 - val_accurac
y: 0.6533
Epoch 6/10
50000/50000 [==============================] - 434s 9ms/step - loss: 0.8458 - accuracy: 0.6994 - val_loss: 0.9278 - val_accurac
y: 0.6711
Epoch 7/10
50000/50000 [==============================] - 458s 9ms/step - loss: 0.7699 - accuracy: 0.7258 - val_loss: 0.9245 - val_accurac
y: 0.6714
Epoch 8/10
50000/50000 [==============================] - 483s 10ms/step - loss: 0.7096 - accuracy: 0.7466 - val_loss: 0.9065 - val_accura
Epoch 9/10
50000/50000 [==============================] - 504s 10ms/step - loss: 0.6521 - accuracy: 0.7705 - val_loss: 0.9066 - val_accura
cy: 0.6903
Epoch 10/10
50000/50000 [==============================] - 388s 8ms/step - loss: 0.5976 - accuracy: 0.7902 - val_loss: 0.9027 - val_accurac
y: 0.6943
```

Calculate its accuracy on testing data

```
_,acc=model.evaluate(test_X,test_Y)
print(acc*100)
```

```
10000/10000 [==============================] - 12s 1ms/step
69.42999958992004
```

Save the model

```
model.save("model1_cifar_10epoch.h5")
```

Make a dictionary to map to the output classes and make predictions from the model

```
results={
   0:'aeroplane',
   1:'automobile',
   2:'bird',
   3:'cat',
   4:'deer',
   5:'dog',
   6:'frog',
   7:'horse',
   8:'ship',
   9:'truck'
}
from PIL import Image
import numpy as np
im=Image.open("C:/Users/GOPINADH/Pictures/truck.jpg")
# the input image is required to be in the shape of dataset, i.e (32,32,3)

im=im.resize((32,32))
im=np.expand_dims(im,axis=0)
im=np.array(im)
pred=model.predict_classes([im])[0]
print(pred,results[pred])
```

```
9 truck
```