ADD CNN LAYER

NEURAL NETWORK PROGRAM TO ADD CNN LAYER :

```
from keras.models import Sequential
from keras.layers.convolutional import Conv2D
from keras.layers import MaxPooling2D
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dense

class CNN:
    @staticmethod
    def build(width, height, depth, total_classes, Saved_Weights_Path=None):
        # Initialize the Model
        model = Sequential()


        # First CONV => RELU => POOL Layer
        model.add(Conv2D(20, 5, 5, border_mode="same", input_shape=(depth, height, width)))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), dim_ordering="th"))


        # Second CONV => RELU => POOL Layer
        model.add(Conv2D(50, 5, 5, border_mode="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), dim_ordering="th"))
        # Third CONV => RELU => POOL Layer
        # Convolution -> ReLU Activation Function -> Pooling Layer
        model.add(Conv2D(100, 5, 5, border_mode="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), dim_ordering="th"))


        # FC => RELU layers
        #  Fully Connected Layer -> ReLU Activation Function
        model.add(Flatten())
        model.add(Dense(500))
        model.add(Activation("relu"))


        # Using Softmax Classifier for Linear Classification
        model.add(Dense(total_classes))
        model.add(Activation("softmax"))


        # If the saved_weights file is already present i.e model is pre-trained, load that weights
        if Saved_Weights_Path is not None:
            model.load_weights(Saved_Weights_Path)
        return model
# --------------------------------- EOC ------------------------------------
```

CNN LAYER PROGRAM :

```
import numpy as np
import argparse
import cv2
from cnn.neural_network import CNN
from keras.utils import np_utils
from keras.optimizers import SGD
# from sklearn.datasets import fetch_mldata
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split


# Parse the Arguments
ap = argparse.ArgumentParser()
ap.add_argument("-s", "--save_model", type=int, default=-1)
ap.add_argument("-l", "--load_model", type=int, default=-1)
ap.add_argument("-w", "--save_weights", type=str)
args = vars(ap.parse_args())


# Read/Download MNIST Dataset
print('Loading MNIST Dataset...')
# dataset = fetch_mldata('MNIST Original')
dataset = fetch_openml('mnist_784')


# Read the MNIST data as array of 784 pixels and convert to 28x28 image matrix
mnist_data = dataset.data.reshape((dataset.data.shape[0], 28, 28))
mnist_data = mnist_data[:, np.newaxis, :, :]
```

```
# Divide data into testing and training sets.
train_img, test_img, train_labels, test_labels = train_test_split(mnist_data/255.0, dataset.target.astype("int"), test_size=0.1)


# Now each image rows and columns are of 28x28 matrix type.
img_rows, img_columns = 28, 28


# Transform training and testing data to 10 classes in range [0,classes] ; num. of classes = 0 to 9 = 10 classes
total_classes = 10        # 0 to 9 labels
train_labels = np_utils.to_categorical(train_labels, 10)
test_labels = np_utils.to_categorical(test_labels, 10)


# Defing and compile the SGD optimizer and CNN model
print('\n Compiling model...')
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
clf = CNN.build(width=28, height=28, depth=1, total_classes=10, Saved_Weights_Path=args["save_weights"] if args["load_model"] > 0 else None)
clf.compile(loss="categorical_crossentropy", optimizer=sgd, metrics=["accuracy"])


# Initially train and test the model; If weight saved already, load the weights using arguments.
b_size = 128      # Batch size
num_epoch = 20    # Number of epochs
verb = 1          # Verbose
```

```
# If weights saved and argument load_model; Load the pre-trained model.
if args["load_model"] < 0:
    print('\nTraining the Model...')
    clf.fit(train_img, train_labels, batch_size=b_size, epochs=num_epoch,verbose=verb)

    # Evaluate accuracy and loss function of test data
    print('Evaluating Accuracy and Loss Function...')
    loss, accuracy = clf.evaluate(test_img, test_labels, batch_size=128, verbose=1)
    print('Accuracy of Model: {:.2f}%'.format(accuracy * 100))



# Save the pre-trained model.
if args["save_model"] > 0:
    print('Saving weights to file...')
    clf.save_weights(args["save_weights"], overwrite=True)



# Show the images using OpenCV and making random selections.
for num in np.random.choice(np.arange(0, len(test_labels)), size=(5,)):
    # Predict the label of digit using CNN.
    probs = clf.predict(test_img[np.newaxis, num])
    prediction = probs.argmax(axis=1)
    # Resize the Image to 100x100 from 28x28 for better view.
    image = (test_img[num][0] * 255).astype("uint8")
    image = cv2.merge([image] * 3)
    image = cv2.resize(image, (100, 100), interpolation=cv2.INTER_LINEAR)
    cv2.putText(image, str(prediction[0]), (5, 20),cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 255, 0), 2)


    # Show and print the Actual Image and Predicted Label Value
    print('Predicted Label: {}, Actual Value: {}'.format(prediction[0],np.argmax(test_labels[num])))
#   cv2.imshow('Digits', image)
#   cv2.waitKey(0)


#---------------------- EOC ---------------------
```