**13.Design a convolutional neural network for classification of any dataset in Keras**

**SOURCE CODE:**

from keras.datasets import mnist

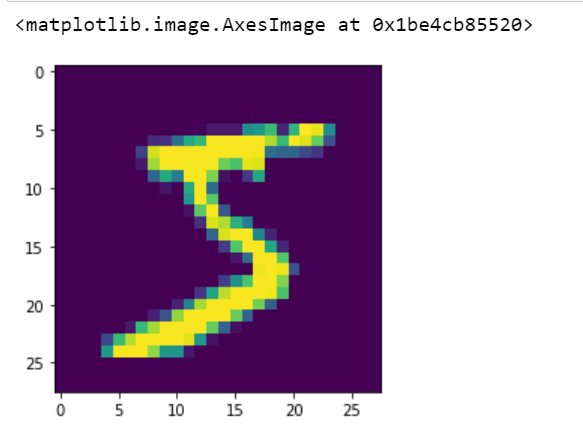
**#download mnist data and split into train and test sets**

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

import matplotlib.pyplot as plt

#plot the first image in the dataset

plt.imshow(X\_train[0])



**#reshape data to fit model**

X\_train = X\_train.reshape(60000,28,28,1)

X\_test = X\_test.reshape(10000,28,28,1)

from keras.utils import to\_categorical

**#one-hot encode target column**

y\_train = to\_categorical(y\_train)

y\_test = to\_categorical(y\_test)

y\_train[0]

from keras.models import Sequential

from keras.layers import Dense, Conv2D, Flatten

**#create model**

model = Sequential()

**#add model layers**

model.add(Conv2D(64, kernel\_size=3, activation='relu', input\_shape=(28,28,1)))

model.add(Conv2D(32, kernel\_size=3, activation='relu'))

model.add(Flatten())

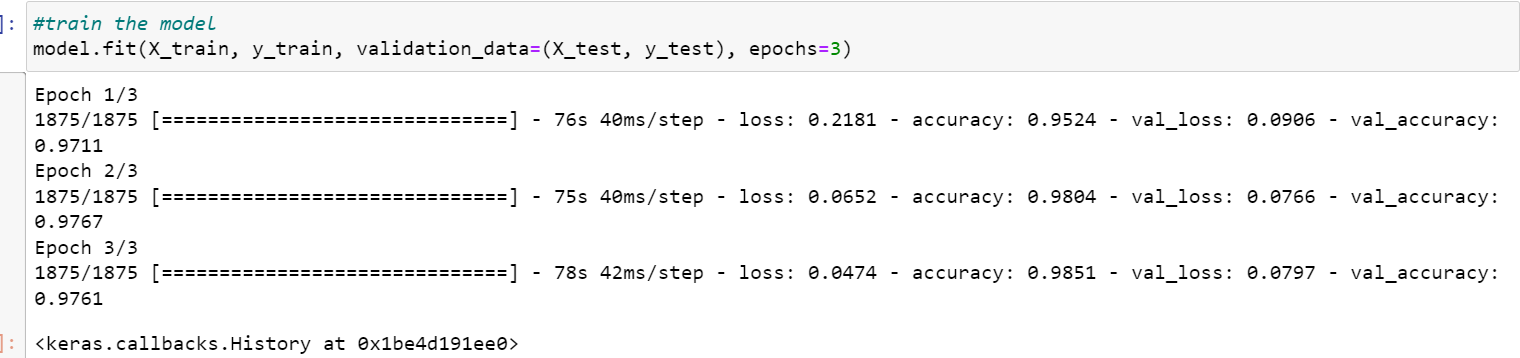
model.add(Dense(10, activation='softmax'))

**#compile model using accuracy to measure model performance**

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

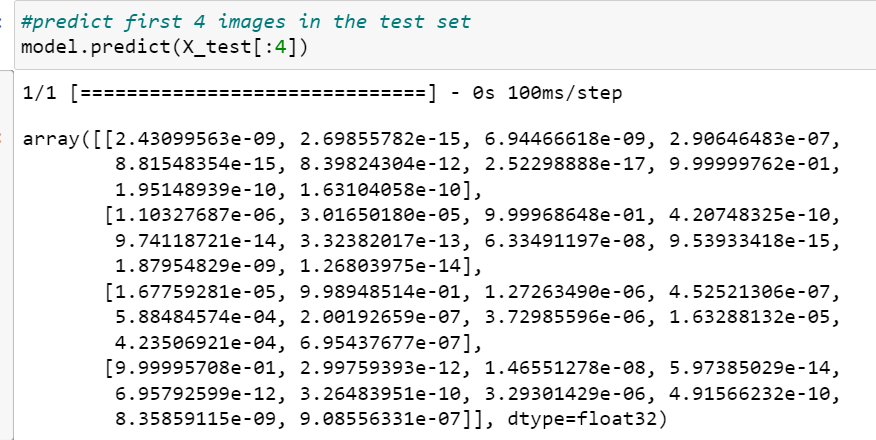
#train the model

model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=3)



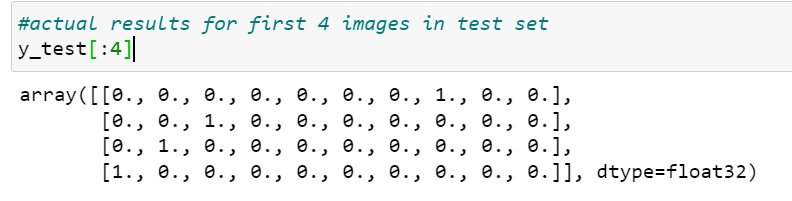
**#predict first 4 images in the test set**

model.predict(X\_test[:4])



**#actual results for first 4 images in test set**

y\_test[:4]



**OBSERVATION:**

* The actual results show that the first four images are also 7, 2,1 and 0. Our model predicted correctly.

**14. Implement Stochastic Gradient Descent optimizations on a linear regression model any dataset.**

**SOURCE CODE:**

**# Load the dataset**

import pandas as pd

from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import SGDRegressor

from sklearn.metrics import mean\_squared\_error

**# Load the dataset**

df = pd.read\_csv('C:\\Users\\GAYATHRI\\Documents\\SRET-I YR- MSC\\TERM III\\Bigdata analytics\\housing (1).csv')

X = df.iloc[:, 1:].values

y = df.iloc[:, 0].values

**# Encode the categorical variables**

labelencoder = LabelEncoder()

X[:, 5] = labelencoder.fit\_transform(X[:, 5])

X[:, 6] = labelencoder.fit\_transform(X[:, 6])

X[:, 7] = labelencoder.fit\_transform(X[:, 7])

X[:, 8] = labelencoder.fit\_transform(X[:, 8])

X[:, 9] = labelencoder.fit\_transform(X[:, 9])

onehotencoder = OneHotEncoder()

X = onehotencoder.fit\_transform(X).toarray()

**# Split the dataset into training and testing sets**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

**# Standardize the features**

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

**# Train the model using SGD**

model = SGDRegressor(max\_iter=1000, eta0=0.01, alpha=0.0001, penalty='l2')

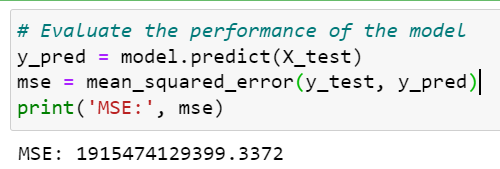
model.fit(X\_train, y\_train)

**# Evaluate the performance of the model**

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred)

print('MSE:', mse)



**OBSERVATION:**

* Stochastic Gradient Descent (SGD) Regressor model is created with an initial learning rate of 0.01, regularization parameter of alpha=0.0001, and L2 penalty. This model is fit to the training data.
* The model is then used to predict the target variable for the test data using the predict method. The mean squared error (MSE) between the predicted and actual target variables is then calculated using the mean\_squared\_error function from the scikit-learn library.
* **MSE value is quite high, MSE: 1915474129399.3372** which indicates that the model may not be performing well on the Housing dataset.