**Data Description**

The data files train.csv and test.csv contain gray-scale images of hand-drawn digits, from zero through nine.

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

The training data set, (train.csv), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.

Each pixel column in the training set has a name like pixelx, where x is an integer between 0 and 783, inclusive. To locate this pixel on the image, suppose that we have decomposed x as x = i \* 28 + j, where i and j are integers between 0 and 27, inclusive. Then pixelx is located on row i and column j of a 28 x 28 matrix, (indexing by zero).

For example, pixel31 indicates the pixel that is in the fourth column from the left, and the second row from the top, as in the ascii-diagram below.

Visually, if we omit the "pixel" prefix, the pixels make up the image like this:

000 001 002 003 ... 026 027

028 029 030 031 ... 054 055

056 057 058 059 ... 082 083

| | | | ... | |

728 729 730 731 ... 754 755

756 757 758 759 ... 782 783

The test data set, (test.csv), is the same as the training set, except that it does not contain the "label" column.

Your submission file should be in the following format: For each of the 28000 images in the test set, output a single line containing the ImageId and the digit you predict. For example, if you predict that the first image is of a 3, the second image is of a 7, and the third image is of a 8, then your submission file would look like:

ImageId,Label  
1,3  
2,7  
3,8

(27997 more lines)

The evaluation metric for this contest is the categorization accuracy, or the proportion of test images that are correctly classified. For example, a categorization accuracy of 0.97 indicates that you have correctly classified all but 3% of the images.

*# This Python 3 environment comes with many helpful analytics libraries installed*

*# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python*

*# For example, here's several helpful packages to load*

​

**import** numpy **as** np *# linear algebra*

**import** pandas **as** pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*

​

*# Input data files are available in the read-only "../input/" directory*

*# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory*

​

**import** os

**for** dirname, \_, filenames **in** os.walk('/kaggle/input'):

**for** filename **in** filenames:

print(os.path.join(dirname, filename))

​

*# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"*

*# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session*

/kaggle/input/digit-recognizer/sample\_submission.csv

/kaggle/input/digit-recognizer/train.csv

/kaggle/input/digit-recognizer/test.csv

df**=**pd.read\_csv("../input/digit-recognizer/train.csv")

df.head()

*#Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total.*

[113]:

|  | **label** | **pixel0** | **pixel1** | **pixel2** | **pixel3** | **pixel4** | **pixel5** | **pixel6** | **pixel7** | **pixel8** | **...** | **pixel774** | **pixel775** | **pixel776** | **pixel777** | **pixel778** | **pixel779** | **pixel780** | **pixel781** | **pixel782** | **pixel783** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

5 rows × 785 columns

test**=**pd.read\_csv("../input/digit-recognizer/test.csv")

test.head()

[114]:

|  | **pixel0** | **pixel1** | **pixel2** | **pixel3** | **pixel4** | **pixel5** | **pixel6** | **pixel7** | **pixel8** | **pixel9** | **...** | **pixel774** | **pixel775** | **pixel776** | **pixel777** | **pixel778** | **pixel779** | **pixel780** | **pixel781** | **pixel782** | **pixel783** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

5 rows × 784 columns

print(df.shape)

print(test.shape)

(42000, 785)

(28000, 784)

*#By split training data into features and target*

X**=**df.drop("label",axis**=**1).values

y**=**df["label"].values

*#by normalize the data*

print(X.shape)

print(y.shape)

print(test.shape)

(42000, 784)

(42000,)

(28000, 784)

import matplotlib.pyplot as plt

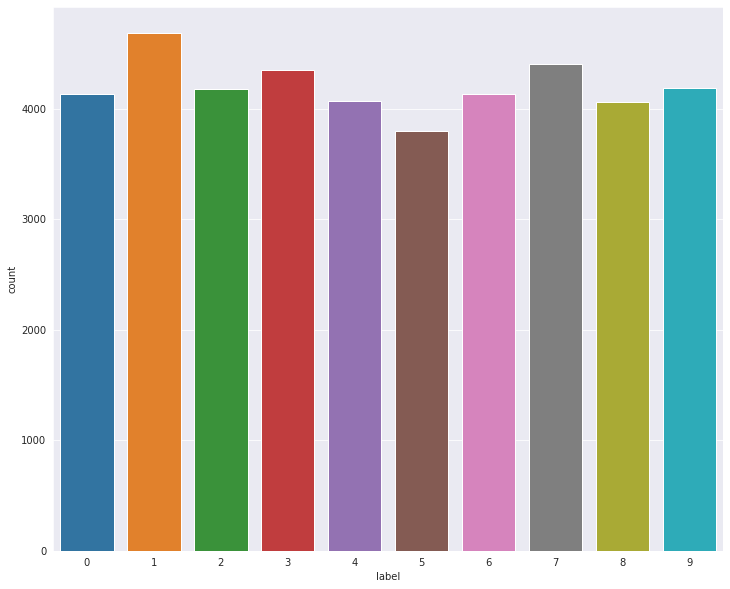
import seaborn as sns

plt.figure(figsize=(12,10))

sns.set\_style("darkgrid")

sns.countplot(x="label",data=df)

<AxesSubplot:xlabel='label', ylabel='count'>



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

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X,y, test\_size**=**0.05, random\_state**=**42)

print(X\_train.shape)

print(X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

(39900, 784)

(2100, 784)

(39900,)

(2100,)

from sklearn.neighbors import KNeighborsClassifier

error\_rate=list()

#here by iterate meny different k values and plot their error rates

#and discover which one is better than others and has the lowest error rate

for i in range(1,10):

knn=KNeighborsClassifier(n\_neighbors=i)

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

prediction\_i=knn.predict(X\_test)

error\_rate.append(np.mean(prediction\_i != y\_test))

# Now we will plot the prediction error rates of different k values

plt.figure(figsize=(15,10))

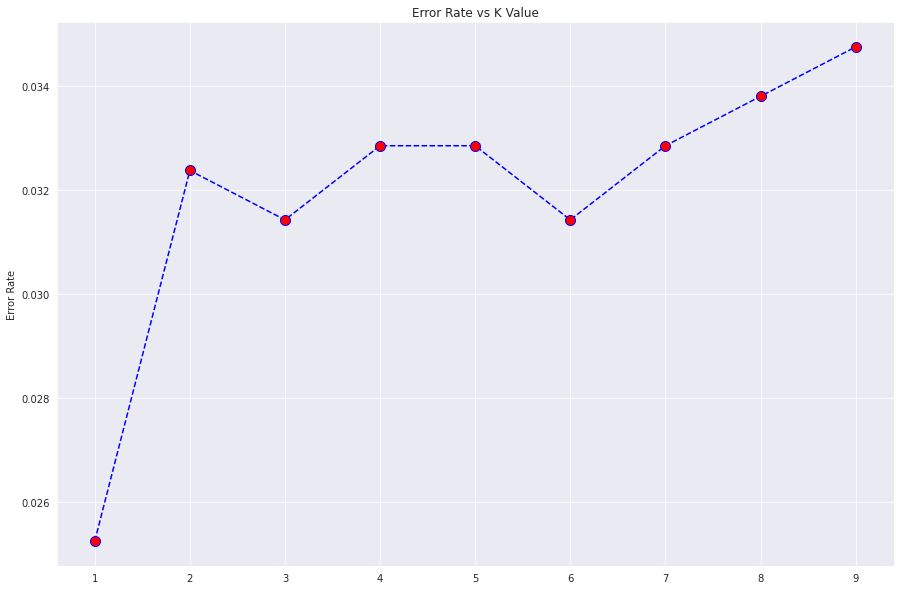
plt.plot(range(1,10),error\_rate, color="blue", linestyle="--",marker="o",markerfacecolor="red",markersize=10)

plt.title("Error Rate vs K Value")

plt.xlabel="K Value"

plt.ylabel("Error Rate")

Text(0, 0.5, 'Error Rate')



knn**=**KNeighborsClassifier(n\_neighbors**=**1) *# we get the minimum error when n=1*

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

knn\_predictions **=** knn.predict(X\_test)

from sklearn.metrics import classification\_report, confusion\_matrix,accuracy\_score

print(confusion\_matrix(y\_test,knn\_predictions))

[[215 0 0 0 0 0 1 0 0 0] [ 0 234 0 0 0 0 0 0 0 0] [ 1 4 211 0 2 0 1 2 0 1] [ 0 1 1 252 0 2 0 2 1 1] [ 0 1 0 0 191 0 0 0 0 4] [ 0 0 0 1 0 155 1 0 2 1] [ 2 0 0 0 0 0 197 0 0 0] [ 0 4 0 0 0 0 0 223 0 3] [ 0 2 2 1 0 4 0 0 181 1] [ 1 0 0 0 1 0 0 2 0 188]]

print(classification\_report(y\_test, knn\_predictions))

precision recall f1-score support

0 0.98 1.00 0.99 216

1 0.95 1.00 0.97 234

2 0.99 0.95 0.97 222

3 0.99 0.97 0.98 260

4 0.98 0.97 0.98 196

5 0.96 0.97 0.97 160

6 0.98 0.99 0.99 199

7 0.97 0.97 0.97 230

8 0.98 0.95 0.97 191

9 0.94 0.98 0.96 192

accuracy 0.97 2100

macro avg 0.97 0.97 0.97 2100

weighted avg 0.98 0.97 0.97 2100

print(accuracy\_score(y\_test, knn\_predictions))

0.9747619047619047

from sklearn.ensemble import RandomForestClassifier

random=RandomForestClassifier()

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

random\_predictions= random.predict(X\_test)

print(accuracy\_score(y\_test, random\_predictions)) #Random forest performs better than KNN

0.9623809523809523

**from** sklearn.naive\_bayes **import** GaussianNB

bayes**=**GaussianNB()

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

bayes\_predictions**=**bayes.predict(X\_test)

​

print(accuracy\_score(y\_test, bayes\_predictions)) *#The predictions are not good*

0.5552380952380952

print(X.shape,y.shape)

X\_train,X\_test,y\_train,y\_test **=** train\_test\_split(X,y, test\_size**=**0.1, random\_state**=**8)

print(X\_train.shape)

print(X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

(42000, 784) (42000,)

(37800, 784)

(4200, 784)

(37800,)

(4200,)

# Reshape image in 3 dimensions (height = 8px, width =8px , canal = 1)

# canal = 1 => For gray scale

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

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

print(type(X))

print(type(test))

test=test.to\_numpy()

test = test.reshape(-1,28,28,1)

# we encode labels to one hot vectors (like [0,0,1,0,0,0,0,0,0,0])

from keras.utils.np\_utils import to\_categorical

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

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

# print(f"Label size {y.shape}")

<class 'numpy.ndarray'>

<class 'pandas.core.frame.DataFrame'>

X\_visualization = X\_train.reshape(X\_train.shape[0], 28, 28)

fig, axis = plt.subplots(1, 4, figsize=(20, 10))

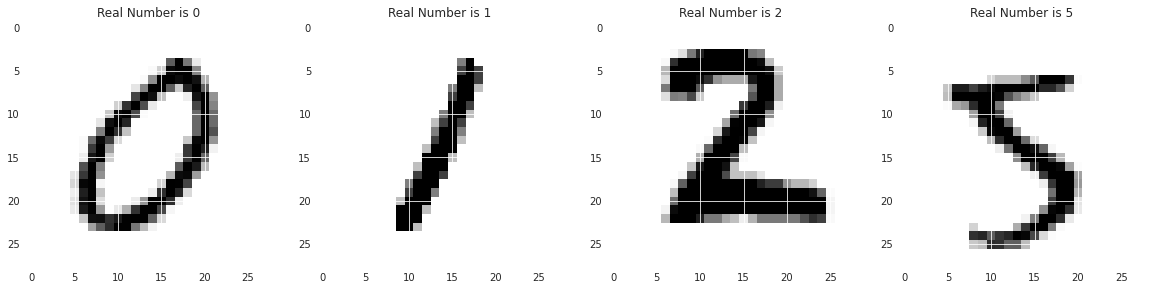
for i, ax **in** enumerate(axis.flat):

ax.imshow(X\_visualization[i], cmap='binary')

digit = y\_train[i].argmax()

ax.set(title = f"Real Number is **{**digit**}**");

*# we see how our data look like.*



from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,Dense, Dropout, BatchNormalization

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

cnn=Sequential()

print(X\_train.shape[0])

print(X\_test.shape[0])

#model.add(Lambda(standardize,input\_shape=(28,28,1)))

cnn.add(Conv2D(filters=64, kernel\_size = (3,3), activation="relu", input\_shape=(28,28,1)))

cnn.add(Conv2D(filters=64, kernel\_size = (3,3), activation="relu"))

cnn.add(MaxPooling2D(pool\_size=(2,2)))

cnn.add(BatchNormalization())

cnn.add(Conv2D(filters=128, kernel\_size = (3,3), activation="relu"))

cnn.add(Conv2D(filters=128, kernel\_size = (3,3), activation="relu"))

cnn.add(MaxPooling2D(pool\_size=(2,2)))

cnn.add(BatchNormalization())

cnn.add(Conv2D(filters=256, kernel\_size = (3,3), activation="relu"))

cnn.add(MaxPooling2D(pool\_size=(2,2)))

cnn.add(Flatten())

cnn.add(BatchNormalization())

cnn.add(Dense(512,activation="relu"))

cnn.add(Dense(10,activation="softmax"))

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

# With data augmentation to prevent overfitting

from keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(

featurewise\_center=False, # set input mean to 0 over the dataset

samplewise\_center=False, # set each sample mean to 0

featurewise\_std\_normalization=False, # divide inputs by std of the dataset

samplewise\_std\_normalization=False, # divide each input by its std

zca\_whitening=False, # apply ZCA whitening

rotation\_range=10, # randomly rotate images in the range (degrees, 0 to 180)

zoom\_range = 0.1, # Randomly zoom image

width\_shift\_range=0.1, # randomly shift images horizontally (fraction of total width)

height\_shift\_range=0.1, # randomly shift images vertically (fraction of total height)

horizontal\_flip=False, # randomly flip images

vertical\_flip=False) # randomly flip images

print(X\_train.shape,y\_train.shape)

print(X\_train.shape,y\_test.shape)

#datagen.fit(X\_train)

train\_generator = datagen.flow(X\_train, y\_train, batch\_size=32)

test\_generator = datagen.flow(X\_test, y\_test, batch\_size=32)

, ,

# Fit the model

history = cnn.fit(train\_generator,

epochs = 10,

steps\_per\_epoch = X\_train.shape[0] // 32,

validation\_steps = X\_test.shape[0] // 32,

validation\_data = test\_generator)

37800

4200

(37800, 28, 28, 1) (37800, 10)

(37800, 28, 28, 1) (4200, 10)

Epoch 1/10

1181/1181 [==============================] - 148s 125ms/step - loss: 0.2877 - accuracy: 0.9075 - val\_loss: 0.1116 - val\_accuracy: 0.9673

Epoch 2/10

1181/1181 [==============================] - 147s 124ms/step - loss: 0.0908 - accuracy: 0.9729 - val\_loss: 0.0799 - val\_accuracy: 0.9776

Epoch 3/10

1181/1181 [==============================] - 147s 125ms/step - loss: 0.0709 - accuracy: 0.9797 - val\_loss: 0.0685 - val\_accuracy: 0.9800

Epoch 4/10

1181/1181 [==============================] - 148s 126ms/step - loss: 0.0594 - accuracy: 0.9821 - val\_loss: 0.0810 - val\_accuracy: 0.9797

Epoch 5/10

1181/1181 [==============================] - 149s 126ms/step - loss: 0.0536 - accuracy: 0.9846 - val\_loss: 0.0719 - val\_accuracy: 0.9816

Epoch 6/10

1181/1181 [==============================] - 147s 125ms/step - loss: 0.0482 - accuracy: 0.9851 - val\_loss: 0.0567 - val\_accuracy: 0.9831

Epoch 7/10

1181/1181 [==============================] - 147s 124ms/step - loss: 0.0450 - accuracy: 0.9875 - val\_loss: 0.0414 - val\_accuracy: 0.9874

Epoch 8/10

1181/1181 [==============================] - 148s 125ms/step - loss: 0.0354 - accuracy: 0.9886 - val\_loss: 0.0723 - val\_accuracy: 0.9781

Epoch 9/10

1181/1181 [==============================] - 147s 124ms/step - loss: 0.0363 - accuracy: 0.9892 - val\_loss: 0.0411 - val\_accuracy: 0.9893

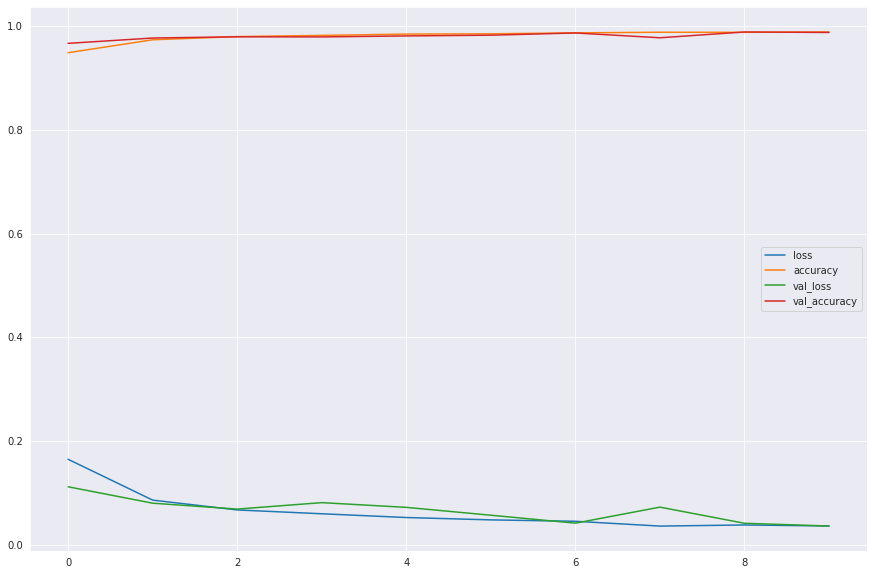
Epoch 10/10

1181/1181 [==============================] - 150s 127ms/step - loss: 0.0316 - accuracy: 0.9905 - val\_loss: 0.0359 - val\_accuracy: 0.9883

sns.set\_style("darkgrid")

pd.DataFrame(cnn.history.history).plot(figsize=(15,10))

<AxesSubplot:>



y\_pred **=** cnn.predict(X\_test)

X\_new **=** X\_test.reshape(X\_test.shape[0], 28, 28)

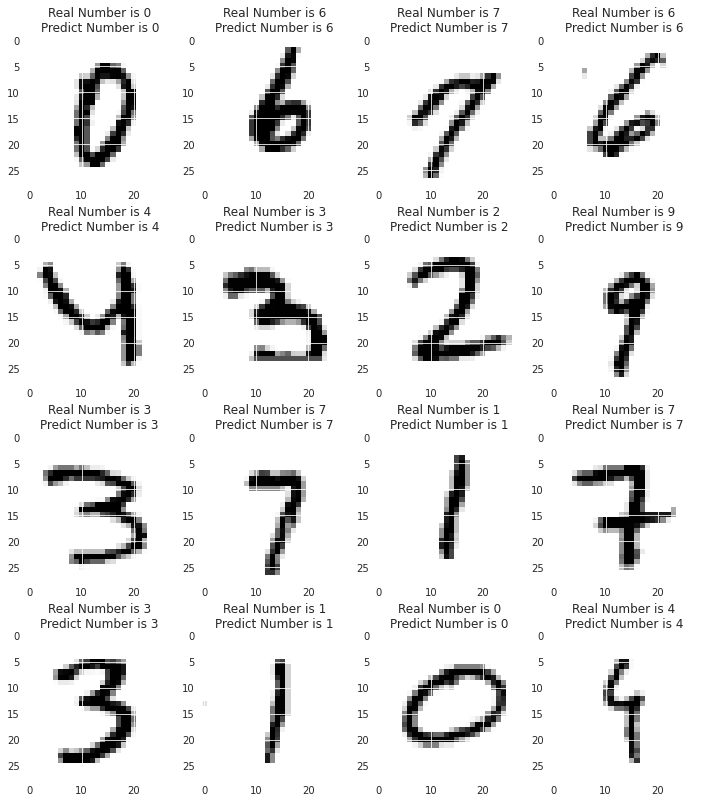
​

fig, axis **=** plt.subplots(4, 4, figsize**=**(12, 14))

**for** i, ax **in** enumerate(axis.flat):

ax.imshow(X\_new[i], cmap**=**'binary')

ax.set(title **=** f"Real Number is {y\_test[i].argmax()}\nPredict Number is {y\_pred[i].argmax()}");



predictions **=** cnn.predict(test, verbose**=**2)

predictions

875/875 - 24s

[229]:

array([[2.5971504e-14, 4.1495401e-13, 1.0000000e+00, ..., 8.1139220e-09,

4.2064414e-11, 1.4805727e-12],

[9.9953377e-01, 1.2438196e-10, 7.2780581e-06, ..., 4.1009003e-05,

4.3436189e-06, 4.1342725e-04],

[9.4930715e-07, 9.3450030e-09, 3.5956219e-07, ..., 8.8415487e-05,

1.4028854e-05, 9.9988580e-01],

...,

[4.0526236e-15, 1.4629605e-16, 1.1151520e-13, ..., 3.3732931e-14,

3.1584902e-13, 1.7327924e-11],

[1.0209483e-08, 7.6529103e-09, 1.2202731e-08, ..., 8.8057723e-06,

7.7707813e-07, 9.9986196e-01],

[4.8995825e-14, 6.4629939e-15, 1.0000000e+00, ..., 8.4178509e-10,

5.6637905e-10, 1.0630967e-11]], dtype=float32)

new\_predictions **=**np.argmax(predictions, axis**=**1) *# we get the original values instead of one hot coded version*

new\_predictions

array([2, 0, 9, ..., 3, 9, 2])

submission **=** pd.read\_csv('../input/digit-recognizer/sample\_submission.csv')

submission

|  | **ImageId** | **Label** |
| --- | --- | --- |
| **0** | 1 | 0 |
| **1** | 2 | 0 |
| **2** | 3 | 0 |
| **3** | 4 | 0 |
| **4** | 5 | 0 |
| **...** | ... | ... |
| **27995** | 27996 | 0 |
| **27996** | 27997 | 0 |
| **27997** | 27998 | 0 |
| **27998** | 27999 | 0 |
| **27999** | 28000 | 0 |

28000 rows × 2 columns

submission['Label'] **=** new\_predictions

submission.to\_csv("my\_submission3.csv", index**=False**)

submission.head()

|  | **ImageId** | **Label** |
| --- | --- | --- |
| **0** | 1 | 2 |
| **1** | 2 | 0 |
| **2** | 3 | 9 |
| **3** | 4 | 0 |
| **4** | 5 | 3 |