

Name: Akashdeep Singh Multani

Roll No- 201980002

Class-MCA

# Lab Evaluation-1

## (Challenge-Kannada MNIST)

### Code Snippet 1:

The screenshot shows a Kaggle notebook interface. On the left is a sidebar with navigation links: Home, Compete, Data, Notebooks (selected), Discuss, Courses, Jobs, and More. Below these are 'Recently Viewed' items including 'FinalCall(201980002)-...', 'TheOne(201980002)', 'Kannada MNIST', and two instances of 'TheOne-3(201980002)...'. The main area displays the notebook 'TheOne-3(201980002)-ASM-1' by Akashdeep Singh Multani, submitted 2 hours ago. It shows a 'Best Submission' status as 'Successful' with a Private Score of 0.98520 and a Public Score of 0.98540. The notebook code is visible in a cell, showing imports for numpy and pandas, and comments about the environment and data directory.

```
In [1]:  
# 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
```

- This snippet is the default importing which is done after we make a notebook for a particular challenge, in my case it is Kannada MNIST.

The screenshot shows a Kaggle notebook page for the user 'akashdeepmultani/theone-3-201980002-asm-1'. The left sidebar contains navigation links: Home, Compete, Data, Notebooks (selected), Discuss, Courses, Jobs, and More. Below these are 'Recently Viewed' items. The main area displays two code snippets. The first snippet is a Python function that walks through the '/kaggle/input' directory and prints the paths of files. The second snippet is a code cell starting with 'In [2]:' that imports various libraries including pandas, numpy, tensorflow, sklearn, and keras. The right sidebar shows the notebook version (3 of 3) and a list of tabs: Input (1), Output, Execution Info, Log, and Comments (0).

```
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 5GB 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

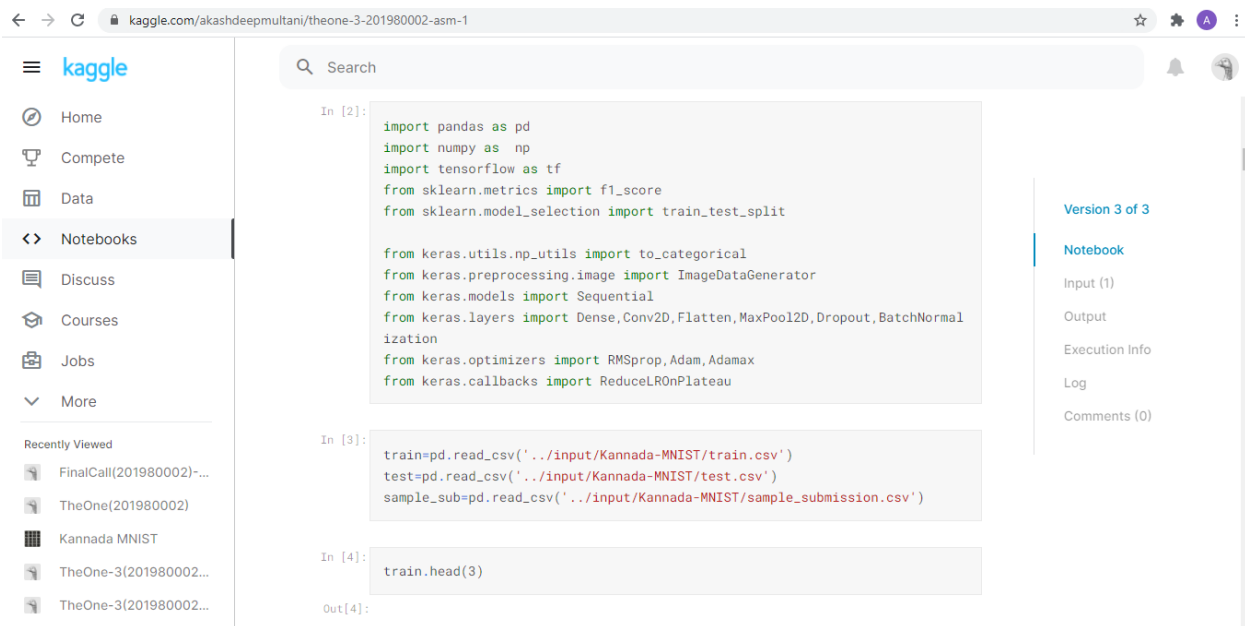
/kaggle/input/Kannada-MNIST/test.csv
/kaggle/input/Kannada-MNIST/Dig-MNIST.csv
/kaggle/input/Kannada-MNIST/sample_submission.csv
/kaggle/input/Kannada-MNIST/train.csv

In [2]:
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split

from keras.utils.np_utils import to_categorical
from keras.preprocessing.image import ImageDataGenerator
```

- In this snippet necessary files for the challenge are imported by default.

## Code Snippet 2:



The screenshot shows a Kaggle notebook titled 'theone-3-201980002-asm-1'. The left sidebar contains navigation links: Home, Compete, Data, Notebooks (selected), Discuss, Courses, Jobs, and More. Below these are 'Recently Viewed' items. The main area displays a Jupyter notebook with three input cells. The first cell (In [2]:) contains import statements for pandas, numpy, tensorflow, sklearn metrics, and sklearn model selection. The second cell (In [3]:) contains code to read CSV files for training, testing, and sample submission. The third cell (In [4]:) contains a single line of code to view the first three rows of the training data. The output of the third cell (Out[4]:) is currently empty. On the right side, there is a sidebar with 'Version 3 of 3', 'Notebook', and links for 'Input (1)', 'Output', 'Execution Info', 'Log', and 'Comments (0)'.

```
In [2]:
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split

from keras.utils.np_utils import to_categorical
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, MaxPool2D, Dropout, BatchNormalization
from keras.optimizers import RMSprop, Adam, Adamax
from keras.callbacks import ReduceLROnPlateau

In [3]:
train=pd.read_csv('../input/Kannada-MNIST/train.csv')
test=pd.read_csv('../input/Kannada-MNIST/test.csv')
sample_sub=pd.read_csv('../input/Kannada-MNIST/sample_submission.csv')

In [4]:
train.head(3)

Out[4]:
```

- In this snippet I have imported pandas as pd means we will use the pandas as the name pd for the reference and this is needed for reading the '.csv' files.
- Then I have imported numpy as np for the tasks related to arrays .
- Then I have imported train\_test\_split which will be used afterwards for the splitting of training and testing set.
- Then I have imported to\_categorical to convert the dataset values to 'one-hot vector'.
- ImageDataGenerator is imported for the preprocessing on the dataset.
- Sequential is imported for the model that will have one layer after another and so on for the rest of layers in the model.

- Conv2D is imported for the 2-dimensional convolution layers.
  - Dense is the fully connected neural network layer, each input node is connected to each output node.
  - Maxpooling2D is used for selecting the maximum value out of 2\*2 matrix which applied on the input.
  - Dropout is used for discontinuing some links according to values provided to dropout for eliminating over-fitting.
  - BatchNormalization is the normalization which is applied on the batches which will normalize the values of the dataset such that no particular value in the dataset can counter-part another on the basis of more values.
- 
- Adam optimizer is used for minimizing the loss such that difference between actual and predicted values is small.
- 
- Then I have imported 'ReduceLROnPlateau' which will adjust the learning rate according to the height or slope.

The screenshot shows a Kaggle notebook titled 'theone-3-201980002-asm-1'. The left sidebar contains navigation links: Home, Compete, Data, Notebooks (selected), Discuss, Courses, Jobs, and More. Below these are 'Recently Viewed' items including 'FinalCall(201980002)-...', 'TheOne(201980002)', 'Kannada MNIST', and two instances of 'TheOne-3(201980002)...'. The main area displays a code cell with the following imports:

```
from keras.preprocessing.image import image_data_generator
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten, MaxPool2D, Dropout, BatchNormalization
from keras.optimizers import RMSprop, Adam, Adamax
from keras.callbacks import ReduceLROnPlateau
```

Below the imports, two input cells are shown:

```
In [3]: train=pd.read_csv('../input/Kannada-MNIST/train.csv')
test=pd.read_csv('../input/Kannada-MNIST/test.csv')
sample_sub=pd.read_csv('../input/Kannada-MNIST/sample_submission.csv')
```

```
In [4]: train.head(3)
```

The output of the second cell is a table showing the first three rows of the training data:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	...	pixel784
0	0	0	0	0	0	0	0	0	0	0	...	0	...	0
1	1	0	0	0	0	0	0	0	0	0	...	0	...	0
2	2	0	0	0	0	0	0	0	0	0	...	0	...	0

Below the table, it indicates '3 rows x 785 columns'.

On the right side of the notebook, there is a sidebar with 'Version 3 of 3', 'Notebook', 'Input (1)', 'Output', 'Execution Info', 'Log', and 'Comments (0)'.

- In this first of all I have looked upon the training set about the values it contains by head function which will return only that row values which I specify in the brackets. Here I have specified 3 so first three rows is fetched and shown.

### Code Snippet 3:

```
In [5]: test.head(3)
test=test.drop('id',axis=1)

In [6]: X_train=train.drop('label',axis=1)
Y_train=train.label

In [7]: X_train=X_train/255
test=test/255

In [8]: X_train=X_train.values.reshape(-1,28,28,1)
test=test.values.reshape(-1,28,28,1)

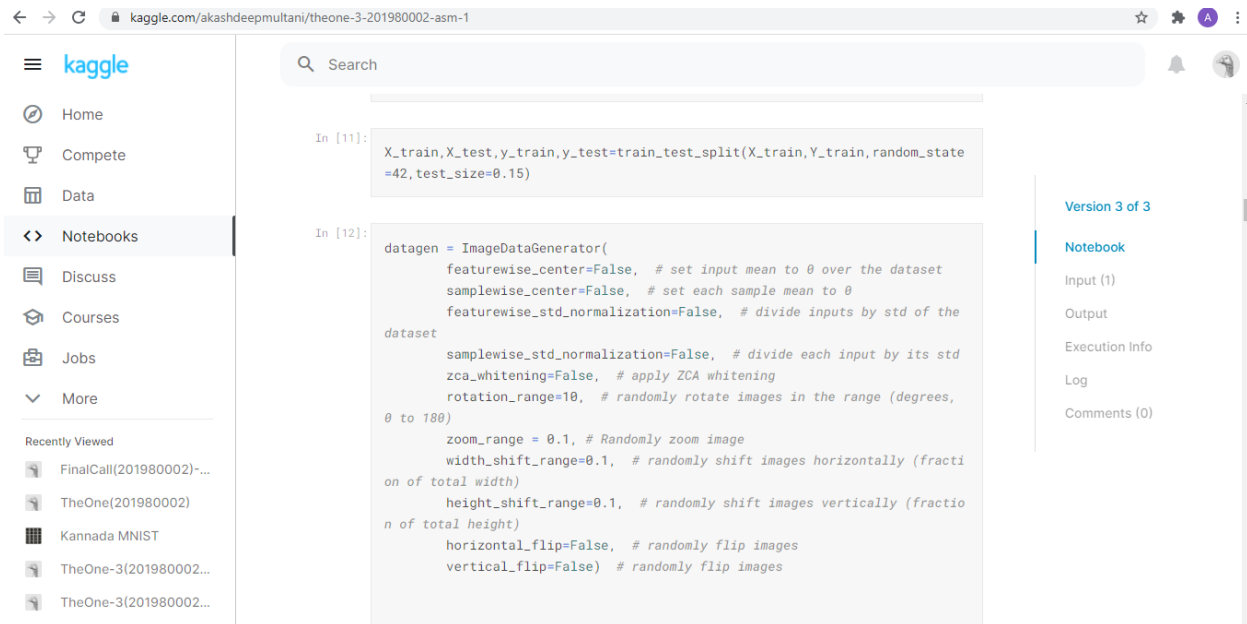
In [9]: X_train.shape,test.shape

Out[9]: ((60000, 28, 28, 1), (5000, 28, 28, 1))

In [10]:
```

- Next I have dropped the 'Id' column which is not needed or contributing to the task assigned or challenge assigned. Here only 'label' is the column which will be predicted according to the challenge whether the image is related to that particular Kannada character or not.
- As according to the challenge we need 'label' column for the testing purpose but not for training .So I have dropped the column 'label' here from the train set.
- Next I have normalized the X\_train and test values.
- Here first the dimensionality is changed for X\_train and test for the convenience that images should focus more on the data in the image that is character in the image not on whole image.

## Code Snippet 4:



The screenshot shows a Kaggle notebook interface. On the left is a sidebar with navigation links: Home, Compete, Data, Notebooks (selected), Discuss, Courses, Jobs, and More. Below these are 'Recently Viewed' items. The main area displays two code cells. The first cell, labeled 'In [11]:', contains the code to split the data into training and testing sets using `train_test_split`. The second cell, labeled 'In [12]:', contains the code to create an `ImageDataGenerator` with various preprocessing parameters. On the right side of the notebook, there is a sidebar with 'Version 3 of 3', 'Notebook', 'Input (1)', 'Output', 'Execution Info', 'Log', and 'Comments (0)'.

```
In [11]: X_train,X_test,y_train,y_test=train_test_split(X_train,Y_train,random_state=42,test_size=0.15)
```

```
In [12]: 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
```

- Then training set and testing set is splitted from the original train set and 15% of data values are given to the test set and remaining to train set.
- Then here data preprocessing is done such that the number of images on which model is to be trained increases which subsequently increases the more precise prediction for the dataset.

## Code Snippet 5:

The screenshot shows a Kaggle notebook interface. On the left is a sidebar with navigation links: Home, Compete, Data, Notebooks (selected), Discuss, Courses, Jobs, and More. Below these are 'Recently Viewed' items. The main area displays two code cells. Cell [13] imports 'regularizers' from 'tensorflow.keras'. Cell [14] defines a 'Sequential' model with multiple layers: two Conv2D layers (32 and 64 filters, 5x5 and 3x3 kernels), a BatchNormalization layer, a MaxPool2D layer, and Dropout layers (0.25). The model uses 'relu' activation and 'Same' padding. The right sidebar shows 'Version 3 of 3', 'Notebook', and a table of contents with links to Input, Output, Execution Info, Log, and Comments.

```
In [13]: from tensorflow.keras import regularizers

In [14]: model = Sequential()

model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation = 'relu', input_shape = (28,28,1)))
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation = 'relu'))
model.add(BatchNormalization(momentum=0.15))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation = 'relu'))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation = 'relu'))
model.add(BatchNormalization(momentum=0.15))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation = 'relu', input_shape = (28,28,1)))
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same'
```

- Here I have imported regularizers for preventing overfitting when the model is trained.
- Then here sequential function is used for layer by layer arrangement in sequential manner.
- In Conv2D I have used initially filters=32, it means it learns 32 filters first.
- Then model learns 64 filters.
- Kernel\_size means how much size of convolution or kernel should be applied on image.
- Here I have use height=5 and width=5.
- Padding="same" means original dimension of the image will remain conserve after applying filter.



- Here batch normalization is used for standardizing the inputs and momentum is the “lag” in learning mean and variance, so that noise due to mini-batch can be ignored.
- MaxPool2D is used to select one maximum value out of 2\*2 matrix when filter is applied to image.
- Dropout Regularization is used to reduce over fitting.

The screenshot shows a Kaggle notebook interface. On the left is a sidebar with navigation links: Home, Compete, Data, Notebooks (selected), Discuss, Courses, Jobs, and More. Below these are 'Recently Viewed' items. The main area contains a code editor with the following Python code:

```
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation = 'relu'))
model.add(BatchNormalization(momentum=.15))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256,kernel_regularizer=regularizers.l2(0.05), activation =
"relu"))
model.add(Dropout(0.4))
model.add(Dense(10, activation = "softmax"))
```

Below the code editor, the command `model.summary()` is executed. The output is a table showing the model's architecture:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 32)	832
conv2d_1 (Conv2D)	(None, 28, 28, 32)	25632

On the right side of the notebook, there are links for 'Version 3 of 3', 'Notebook', 'Input (1)', 'Output', 'Execution Info', 'Log', and 'Comments (0)'.

- Flatten function will convert multidimensional tensor to single 1D tensor.

## Code Snippet 6:

← → ↻ kaggle.com/akashdeepmultani/theone-3-201980002-asm-1 ☆ ⚙️ A ⋮

≡ kaggle

- 🏠 Home
- 🏆 Compete
- 📊 Data
- 📁 Notebooks
- 💬 Discuss
- 🎓 Courses
- 💼 Jobs
- ⌵ More

Recently Viewed

- 📄 FinalCall(201980002)-...
- 📄 TheOne(201980002)
- 📄 Kannada MNIST
- 📄 TheOne-3(201980002...
- 📄 TheOne-3(201980002...

🔍 Search

```
max_pooling2d (MaxPooling2D) (None, 14, 14, 32) 0
-----
dropout (Dropout) (None, 14, 14, 32) 0
-----
conv2d_2 (Conv2D) (None, 14, 14, 64) 18496
conv2d_3 (Conv2D) (None, 14, 14, 64) 36928
-----
batch_normalization_1 (Batch Normalization) (None, 14, 14, 64) 256
max_pooling2d_1 (MaxPooling2D) (None, 7, 7, 64) 0
-----
dropout_1 (Dropout) (None, 7, 7, 64) 0
-----
conv2d_4 (Conv2D) (None, 7, 7, 32) 51232
conv2d_5 (Conv2D) (None, 7, 7, 32) 25632
-----
batch_normalization_2 (Batch Normalization) (None, 7, 7, 32) 128
max_pooling2d_2 (MaxPooling2D) (None, 3, 3, 32) 0
-----
dropout_2 (Dropout) (None, 3, 3, 32) 0
-----
```

Version 3 of 3

Notebook

Input (1)

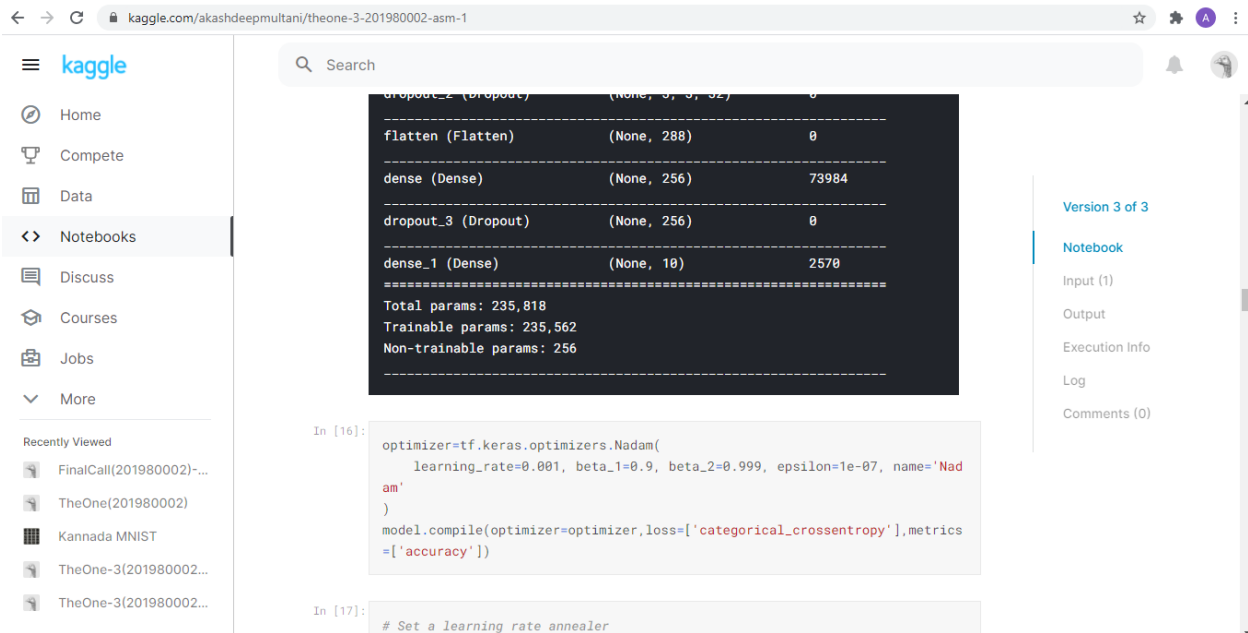
Output

Execution Info

Log

Comments (0)

## Code Snippet 7:



The screenshot shows a Kaggle notebook interface. On the left is a sidebar with navigation links: Home, Compete, Data, Notebooks (selected), Discuss, Courses, Jobs, and More. Below these are 'Recently Viewed' items. The main area displays a Keras model summary for a neural network. The summary table is as follows:

Layer	Output Shape	Param Count
Flatten (Flatten)	(None, 288)	0
dense (Dense)	(None, 256)	73984
dropout_3 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 10)	2570

Below the table, the summary states: Total params: 235,818; Trainable params: 235,562; Non-trainable params: 256.

The code cell below the summary shows the configuration of the Nadam optimizer and the compilation of the model:

```
In [16]: optimizer=tf.keras.optimizers.Nadam(
        learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, name='Nadam')
model.compile(optimizer=optimizer, loss=['categorical_crossentropy'], metrics=['accuracy'])
```

The next code cell is partially visible:

```
In [17]: # Set a learning rate annealer
```

On the right side of the notebook, there are links for 'Version 3 of 3', 'Notebook', 'Input (1)', 'Output', 'Execution Info', 'Log', and 'Comments (0)'.

- Dense layer will multiply input tensor that is 256 in my case is multiplied to weight of dense layer that is 10 in my case plus bias which leads to  $(2560+10)=2570$ .
- It connects neurons in one layer to neurons in another layer. It is used to classify images between different category by training.
- In this snippet I have used '**Nadam**' as the optimizer to reduce the loss i.e difference between actual and predicted values because it converges to the global minima with the fastest speed as compared to other and also efficient.
- **Categorical Crossentropy** is used as the loss because of multiclass classification as in my case there are 10 classes.

The screenshot shows a Kaggle notebook interface. The left sidebar contains navigation links: Home, Compete, Data, Notebooks (selected), Discuss, Courses, Jobs, and More. Below these are 'Recently Viewed' items: FinalCall(201980002)-..., TheOne(201980002), Kannada MNIST, TheOne-3(201980002)..., and TheOne-3(201980002).... The main area displays three code cells:

```
In [17]:  
# Set a learning rate annealer  
learning_rate_reduction = ReduceLRonPlateau(monitor='val_acc',  
                                             patience=3,  
                                             verbose=1,  
                                             factor=0.5,  
                                             min_lr=0.00001)
```

```
In [18]:  
epochs=30  
batch_size=64
```

```
In [19]:  
# Fit the model  
history = model.fit_generator(datagen.flow(X_train,y_train, batch_size=batch_size),  
                             epochs = epochs, validation_data = (X_test,y_test),  
                             verbose = 2, steps_per_epoch=X_train.shape[0]  
                             // batch_size,  
                             , callbacks=[learning_rate_reduction])
```

Below the code cells, the output of the last cell is visible:

```
Epoch 1/30  
796/796 - 231s - loss: 1.4593 - accuracy: 0.9240 - val_loss: 0.1163 - va
```

On the right side, there is a 'Version 3 of 3' indicator and a 'Notebook' tab. Below the tab are links for 'Input (1)', 'Output', 'Execution Info', 'Log', and 'Comments (0)'.

- Then I have fitted the model with my preprocessed data and with epochs=30 and batch\_size=64.

kaggle

Home

Compete

Data

Notebooks

Discuss

Courses

Jobs

More

Recently Viewed

FinalCall(201980002)-...

TheOne(201980002)

Kannada MNIST

TheOne-3(201980002...

TheOne-3(201980002...

Search

Epoch 2/30  
796/796 - 229s - loss: 0.1346 - accuracy: 0.9818 - val\_loss: 0.0951 - val\_accuracy: 0.9920  
Epoch 3/30  
796/796 - 231s - loss: 0.1177 - accuracy: 0.9852 - val\_loss: 0.0883 - val\_accuracy: 0.9931  
Epoch 4/30  
796/796 - 228s - loss: 0.1000 - accuracy: 0.9884 - val\_loss: 0.0815 - val\_accuracy: 0.9944  
Epoch 5/30  
796/796 - 232s - loss: 0.1041 - accuracy: 0.9893 - val\_loss: 0.0861 - val\_accuracy: 0.9947  
Epoch 6/30  
796/796 - 232s - loss: 0.0987 - accuracy: 0.9893 - val\_loss: 0.0792 - val\_accuracy: 0.9906  
Epoch 7/30  
796/796 - 228s - loss: 0.0943 - accuracy: 0.9900 - val\_loss: 0.0854 - val\_accuracy: 0.9934  
Epoch 8/30  
796/796 - 230s - loss: 0.0892 - accuracy: 0.9913 - val\_loss: 0.0699 - val\_accuracy: 0.9957  
Epoch 9/30  
796/796 - 227s - loss: 0.0932 - accuracy: 0.9915 - val\_loss: 0.0814 - val\_accuracy: 0.9959  
Epoch 10/30  
796/796 - 227s - loss: 0.0880 - accuracy: 0.9914 - val\_loss: 0.0726 - val\_accuracy: 0.9958

Version 3 of 3

Notebook

Input (1)

Output

Execution Info

Log

Comments (0)

kaggle

Home

Compete

Data

Notebooks

Discuss

Courses

Jobs

More

Recently Viewed

FinalCall(201980002)-...

TheOne(201980002)

Kannada MNIST

TheOne-3(201980002...

TheOne-3(201980002...

Search

1\_accuracy: 0.9958  
Epoch 11/30  
796/796 - 230s - loss: 0.0811 - accuracy: 0.9923 - val\_loss: 0.0741 - val\_accuracy: 0.9953  
Epoch 12/30  
796/796 - 229s - loss: 0.0826 - accuracy: 0.9928 - val\_loss: 0.0695 - val\_accuracy: 0.9957  
Epoch 13/30  
796/796 - 230s - loss: 0.0825 - accuracy: 0.9927 - val\_loss: 0.0688 - val\_accuracy: 0.9956  
Epoch 14/30  
796/796 - 225s - loss: 0.0901 - accuracy: 0.9921 - val\_loss: 0.0751 - val\_accuracy: 0.9953  
Epoch 15/30  
796/796 - 226s - loss: 0.0724 - accuracy: 0.9938 - val\_loss: 0.0599 - val\_accuracy: 0.9966  
Epoch 16/30  
796/796 - 229s - loss: 0.0747 - accuracy: 0.9934 - val\_loss: 0.0694 - val\_accuracy: 0.9963  
Epoch 17/30  
796/796 - 227s - loss: 0.0881 - accuracy: 0.9931 - val\_loss: 0.0469 - val\_accuracy: 0.9959  
Epoch 18/30  
796/796 - 228s - loss: 0.0673 - accuracy: 0.9944 - val\_loss: 0.0517 - val\_accuracy: 0.9960  
Epoch 19/30

Version 3 of 3

Notebook

Input (1)


Output

Execution Info

Log

Comments (0)

← → ↻ kaggle.com/akashdeepmultani/theone-3-201980002-asm-1 ☆ ⚙️ 👤



- Home
- Compete
- Data
- Notebooks**
- Discuss
- Courses
- Jobs
- More

Recently Viewed

- FinalCall(201980002)-...
- TheOne(201980002)
- Kannada MNIST
- TheOne-3(201980002)...
- TheOne-3(201980002)...

🔍 Search

```
Epoch 19/30
796/796 - 238s - loss: 0.0764 - accuracy: 0.9944 - val_loss: 0.0633 - va
l_accuracy: 0.9958
Epoch 20/30
796/796 - 226s - loss: 0.0743 - accuracy: 0.9942 - val_loss: 0.0494 - va
l_accuracy: 0.9968
Epoch 21/30
796/796 - 229s - loss: 0.0901 - accuracy: 0.9937 - val_loss: 0.0678 - va
l_accuracy: 0.9976
Epoch 22/30
796/796 - 227s - loss: 0.0718 - accuracy: 0.9945 - val_loss: 0.0791 - va
l_accuracy: 0.9953
Epoch 23/30
796/796 - 228s - loss: 0.0701 - accuracy: 0.9948 - val_loss: 0.1150 - va
l_accuracy: 0.9952
Epoch 24/30
796/796 - 231s - loss: 0.0743 - accuracy: 0.9943 - val_loss: 0.0824 - va
l_accuracy: 0.9962
Epoch 25/30
796/796 - 227s - loss: 0.0688 - accuracy: 0.9952 - val_loss: 0.0526 - va
l_accuracy: 0.9950
Epoch 26/30
796/796 - 230s - loss: 0.0703 - accuracy: 0.9949 - val_loss: 0.0900 - va
l_accuracy: 0.9949
Epoch 27/30
```

Version 3 of 3

Notebook

Input (1)

Output

Execution Info

Log

Comments (0)

## Code Snippet 8:

The screenshot displays a Kaggle notebook page. On the left is a sidebar with navigation links: Home, Compete, Data, Notebooks (selected), Discuss, Courses, Jobs, and More. Below these is a 'Recently Viewed' section listing several notebooks. The main area contains a search bar and a list of training logs for a model. The logs show metrics like loss, accuracy, and validation loss across different epochs. Below the logs are two code cells. The first cell, labeled 'In [20]:', contains Python code using pandas to read a CSV file, drop the 'id' column, and reshape the data. The second cell, labeled 'In [21]:', contains code for making predictions using a model and converting the results to labels.

```
796/796 - 233s - loss: 0.0644 - accuracy: 0.9954 - val_loss: 0.0605 - val_accuracy: 0.9964
Epoch 28/30
796/796 - 229s - loss: 0.0716 - accuracy: 0.9951 - val_loss: 0.0439 - val_accuracy: 0.9967
Epoch 29/30
796/796 - 231s - loss: 0.0734 - accuracy: 0.9952 - val_loss: 0.0662 - val_accuracy: 0.9959
Epoch 30/30
796/796 - 233s - loss: 0.0704 - accuracy: 0.9951 - val_loss: 0.0550 - val_accuracy: 0.9967
```

```
In [20]: test=pd.read_csv('../input/Kannada-MNIST/test.csv')
        test_id=test.id

        test=test.drop('id',axis=1)
        test=test/255
        test=test.values.reshape(-1,28,28,1)
```

```
In [21]: y_pre=model.predict(test)      ##making prediction
        y_pre=np.argmax(y_pre,axis=1) ##changing the prediction into labels
```

- Here I have used pandas for reading the csv file i.e test.csv.
- Here test.csv contains column 'id' which is removed from test.csv file.
- And then reshaping the test.csv file values for making the dimensionality same for submitting to Kaggle.

## Code Snippet 9:



- Here predict function is used for making the prediction.
- np.argmax is used here for getting the maximum value from the prediction.
- Then submission is done for the evaluation on Kaggle.