In [1]:

```
#Importing our required packages Packages
import numpy as np
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Dense,Flatten,Conv2D,MaxPool2D,Dropout
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.preprocessing.image import ImageDataGenerator

```

In [2]:

```
#Loading our datasets,
train_df=pd.read_csv('datasets/sign_mnist_train.csv')
test_df=pd.read_csv('datasets/sign_mnist_test.csv')
```

In [5]:

```
#Looking at the basic statistics of our dataset.
train_df.describe()
```

Out[5]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5
count	27455.000000	27455.000000	27455.000000	27455.000000	27455.000000	27455.000000
mean	12.318813	145.419377	148.500273	151.247714	153.546531	156.210891
std	7.287552	41.358555	39.942152	39.056286	38.595247	37.111165
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	6.000000	121.000000	126.000000	130.000000	133.000000	137.000000
50%	13.000000	150.000000	153.000000	156.000000	158.000000	160.000000
75%	19.000000	174.000000	176.000000	178.000000	179.000000	181.000000
max	24.000000	255.000000	255.000000	255.000000	255.000000	255.000000

8 rows × 785 columns

In [25]:

```
#taking a look at our data
train_df.head()
```

Out[25]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel77
0	3	107	118	127	134	139	143	146	150	153	...	207	20
1	6	155	157	156	156	156	157	156	158	158	...	69	14
2	2	187	188	188	187	187	186	187	188	187	...	202	20
3	2	211	211	212	212	211	210	211	210	210	...	235	23
4	13	164	167	170	172	176	179	180	184	185	...	92	10

5 rows × 785 columns

In [9]:

```
train_label=train_df['label']
train_label.head()
trainset=train_df.drop(['label'],axis=1)
trainset.head()
```

Out[9]:

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	...	pixel775	pixel
0	107	118	127	134	139	143	146	150	153	156	...	207	
1	155	157	156	156	156	157	156	158	158	157	...	69	
2	187	188	188	187	187	186	187	188	187	186	...	202	
3	211	211	212	212	211	210	211	210	210	211	...	235	
4	164	167	170	172	176	179	180	184	185	186	...	92	

5 rows × 784 columns

In [10]:

```
X_train = trainset.values
X_train = trainset.values.reshape(-1,28,28,1)
print(X_train.shape)
```

```
(27455, 28, 28, 1)
```

In [11]:

```
test_label=test_df['label']
X_test=test_df.drop(['label'],axis=1)
print(X_test.shape)
X_test.head()
```

```
(7172, 784)
```

Out[11]:

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	...	pixel775	pixel
0	149	149	150	150	150	151	151	150	151	152	...	138	
1	126	128	131	132	133	134	135	135	136	138	...	47	
2	85	88	92	96	105	123	135	143	147	152	...	68	
3	203	205	207	206	207	209	210	209	210	209	...	154	
4	188	191	193	195	199	201	202	203	203	203	...	26	

5 rows × 784 columns

In [12]:

```
#Converting our integers to binary form. With the help of LabelBinarizer
from sklearn.preprocessing import LabelBinarizer
lb=LabelBinarizer()
y_train=lb.fit_transform(train_label)
y_test=lb.fit_transform(test_label)
```

In [13]:

```
y_train
```

Out[13]:

```
array([[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, 1, 0]])
```

In [14]:

```
X_test=X_test.values.reshape(-1,28,28,1)
```

In [15]:

```
print(X_train.shape,y_train.shape,X_test.shape,y_test.shape)
```

```
(27455, 28, 28, 1) (27455, 24) (7172, 28, 28, 1) (7172, 24)
```

In [16]:

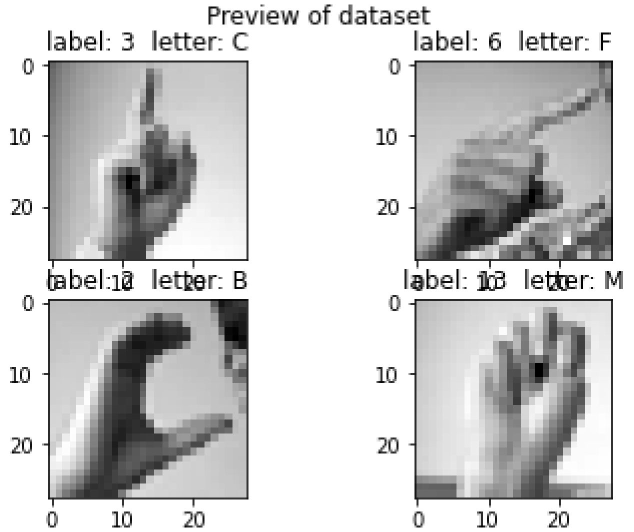
```
'''
Augmenting the image dataset to generate new data

ImageDataGenerator package from keras.preprocessing.image allows to add different d

Here is the package details https://keras.io/preprocessing/image/

The image dataset in also normalised here using the rescale parameter which divides
'''

train_datagen = ImageDataGenerator(rescale = 1./255,
                                  rotation_range = 0,
                                  height_shift_range=0.2,
                                  width_shift_range=0.2,
                                  shear_range=0,
                                  zoom_range=0.2,
                                  horizontal_flip=True,
                                  fill_mode='nearest')

X_test=X_test/255
```

In [17]:

```
'''
Visualization of the Dataset
Preview of the images in the training dataset
'''
fig,axe=plt.subplots(2,2)
fig.suptitle('Preview of dataset')
axe[0,0].imshow(X_train[0].reshape(28,28),cmap='gray')
axe[0,0].set_title('label: 3  letter: C')
axe[0,1].imshow(X_train[1].reshape(28,28),cmap='gray')
axe[0,1].set_title('label: 6  letter: F')
axe[1,0].imshow(X_train[2].reshape(28,28),cmap='gray')
axe[1,0].set_title('label: 2  letter: B')
axe[1,1].imshow(X_train[4].reshape(28,28),cmap='gray')
axe[1,1].set_title('label: 13  letter: M')
```

Out[17]:

```
Text(0.5, 1.0, 'label: 13  letter: M')
```



In [19]:

```
'''
Building our CNN Model
This model consist of-

1. Three convolution layer which uses  MaxPooling for better feature capture.
2. A dense layer with 512 units
3. The output layer consist of 24 units for 24 different classes



Some information about the Convolution layers
The activation fucntion which we have used is ReLu.

Conv layer 1 -- UNITS - 128 KERNEL SIZE - 5 * 5 STRIDE LENGTH - 1 ACTIVATION - ReLu
Conv layer 2 -- UNITS - 64 KERNEL SIZE - 3 * 3 STRIDE LENGTH - 1 ACTIVATION - ReLu
Conv layer 3 -- UNITS - 32 KERNEL SIZE - 2 * 2 STRIDE LENGTH - 1 ACTIVATION - ReLu

MaxPool layer 1 -- MAX POOL WINDOW - 3 * 3 STRIDE - 2
MaxPool layer 2 -- MAX POOL WINDOW - 2 * 2 STRIDE - 2
MaxPool layer 3 -- MAX POOL WINDOW - 2 * 2 STRIDE - 2
'''

model=Sequential()
model.add(Conv2D(128,kernel_size=(5,5),
                 strides=1,padding='same',activation='relu',input_shape=(28,28,1)))
model.add(MaxPool2D(pool_size=(3,3),strides=2,padding='same'))
model.add(Conv2D(64,kernel_size=(2,2),
                strides=1,activation='relu',padding='same'))
model.add(MaxPool2D((2,2),2,padding='same'))
model.add(Conv2D(32,kernel_size=(2,2),
                strides=1,activation='relu',padding='same'))
model.add(MaxPool2D((2,2),2,padding='same'))

model.add(Flatten())
```

In [20]:

```
#Dense and output layers
model.add(Dense(units=512,activation='relu'))
model.add(Dropout(rate=0.25))
model.add(Dense(units=24,activation='softmax'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 128)       3328      
                                                                 
 max_pooling2d (MaxPooling2D  (None, 14, 14, 128)      0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 14, 14, 64)        32832     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 7, 7, 64)         0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 7, 7, 32)          8224      
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 4, 4, 32)         0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (None, 512)               0         
                                                                 
 dense (Dense)               (None, 512)               262656    
                                                                 
 dropout (Dropout)           (None, 512)               0         
                                                                 
 dense_1 (Dense)             (None, 24)                12312     
                                                                 
=================================================================
Total params: 319,352
Trainable params: 319,352
Non-trainable params: 0
_________________________________________________________________
```

In [21]:

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

In [22]:

```
#Training the model
model.fit(train_datagen.flow(X_train,y_train,batch_size=200),
         epochs = 35,
          validation_data=(X_test,y_test),
          shuffle=1
         )
```

```
Epoch 1/35
138/138 [==============================] - 89s 620ms/step - loss: 2.992
8 - accuracy: 0.1007 - val_loss: 2.3336 - val_accuracy: 0.2716
Epoch 2/35
138/138 [==============================] - 85s 616ms/step - loss: 2.265
2 - accuracy: 0.2861 - val_loss: 1.5806 - val_accuracy: 0.4558
Epoch 3/35
138/138 [==============================] - 97s 702ms/step - loss: 1.660
4 - accuracy: 0.4520 - val_loss: 1.1958 - val_accuracy: 0.5471
Epoch 4/35
138/138 [==============================] - 111s 803ms/step - loss: 1.28
77 - accuracy: 0.5664 - val_loss: 0.9396 - val_accuracy: 0.6528
Epoch 5/35
138/138 [==============================] - 108s 786ms/step - loss: 1.06
20 - accuracy: 0.6426 - val_loss: 0.7139 - val_accuracy: 0.7400
Epoch 6/35
138/138 [==============================] - 106s 770ms/step - loss: 0.88
63 - accuracy: 0.6988 - val_loss: 0.5081 - val_accuracy: 0.8260
Epoch 7/35
138/138 [==============================] - 98s 711ms/step - loss: 0.766
9 - accuracy: 0.7396 - val_loss: 0.4069 - val_accuracy: 0.8639
Epoch 8/35
138/138 [==============================] - 103s 747ms/step - loss: 0.65
12 - accuracy: 0.7803 - val_loss: 0.3731 - val_accuracy: 0.8681
Epoch 9/35
138/138 [==============================] - 103s 750ms/step - loss: 0.57
92 - accuracy: 0.8017 - val_loss: 0.2901 - val_accuracy: 0.9091
Epoch 10/35
138/138 [==============================] - 112s 811ms/step - loss: 0.50
62 - accuracy: 0.8271 - val_loss: 0.2422 - val_accuracy: 0.9285
Epoch 11/35
138/138 [==============================] - 104s 752ms/step - loss: 0.45
52 - accuracy: 0.8467 - val_loss: 0.1882 - val_accuracy: 0.9472
Epoch 12/35
138/138 [==============================] - 111s 807ms/step - loss: 0.41
73 - accuracy: 0.8578 - val_loss: 0.1622 - val_accuracy: 0.9505
Epoch 13/35
138/138 [==============================] - 109s 792ms/step - loss: 0.38
06 - accuracy: 0.8715 - val_loss: 0.1261 - val_accuracy: 0.9672
Epoch 14/35
138/138 [==============================] - 111s 803ms/step - loss: 0.34
53 - accuracy: 0.8836 - val_loss: 0.1193 - val_accuracy: 0.9670
Epoch 15/35
138/138 [==============================] - 102s 738ms/step - loss: 0.31
59 - accuracy: 0.8937 - val_loss: 0.0935 - val_accuracy: 0.9742
Epoch 16/35
138/138 [==============================] - 107s 774ms/step - loss: 0.31
54 - accuracy: 0.8920 - val_loss: 0.1538 - val_accuracy: 0.9543
Epoch 17/35
138/138 [==============================] - 105s 759ms/step - loss: 0.28
14 - accuracy: 0.9061 - val_loss: 0.0865 - val_accuracy: 0.9785
```

```
Epoch 18/35
138/138 [==============================] - 111s 805ms/step - loss: 0.26
05 - accuracy: 0.9114 - val_loss: 0.0787 - val_accuracy: 0.9763
Epoch 19/35
138/138 [==============================] - 112s 807ms/step - loss: 0.24
16 - accuracy: 0.9185 - val_loss: 0.0570 - val_accuracy: 0.9831
Epoch 20/35
138/138 [==============================] - 109s 786ms/step - loss: 0.23
36 - accuracy: 0.9219 - val_loss: 0.0634 - val_accuracy: 0.9826
Epoch 21/35
138/138 [==============================] - 78s 559ms/step - loss: 0.206
8 - accuracy: 0.9286 - val_loss: 0.0500 - val_accuracy: 0.9894
Epoch 22/35
138/138 [==============================] - 76s 549ms/step - loss: 0.201
9 - accuracy: 0.9301 - val_loss: 0.0622 - val_accuracy: 0.9808
Epoch 23/35
138/138 [==============================] - 76s 552ms/step - loss: 0.198
4 - accuracy: 0.9346 - val_loss: 0.0323 - val_accuracy: 0.9948
Epoch 24/35
138/138 [==============================] - 78s 562ms/step - loss: 0.190
8 - accuracy: 0.9363 - val_loss: 0.0232 - val_accuracy: 0.9967
Epoch 25/35
138/138 [==============================] - 76s 550ms/step - loss: 0.193
7 - accuracy: 0.9348 - val_loss: 0.0653 - val_accuracy: 0.9802
Epoch 26/35
138/138 [==============================] - 76s 548ms/step - loss: 0.173
8 - accuracy: 0.9415 - val_loss: 0.0278 - val_accuracy: 0.9957
Epoch 27/35
138/138 [==============================] - 76s 549ms/step - loss: 0.156
8 - accuracy: 0.9485 - val_loss: 0.0217 - val_accuracy: 0.9974
Epoch 28/35
138/138 [==============================] - 76s 552ms/step - loss: 0.162
4 - accuracy: 0.9456 - val_loss: 0.0291 - val_accuracy: 0.9915
Epoch 29/35
138/138 [==============================] - 77s 555ms/step - loss: 0.153
5 - accuracy: 0.9490 - val_loss: 0.0246 - val_accuracy: 0.9927
Epoch 30/35
138/138 [==============================] - 76s 550ms/step - loss: 0.148
4 - accuracy: 0.9488 - val_loss: 0.0221 - val_accuracy: 0.9932
Epoch 31/35
138/138 [==============================] - 76s 550ms/step - loss: 0.154
0 - accuracy: 0.9484 - val_loss: 0.0375 - val_accuracy: 0.9880
Epoch 32/35
138/138 [==============================] - 76s 553ms/step - loss: 0.137
4 - accuracy: 0.9546 - val_loss: 0.0314 - val_accuracy: 0.9883
Epoch 33/35
138/138 [==============================] - 77s 554ms/step - loss: 0.132
1 - accuracy: 0.9565 - val_loss: 0.0443 - val_accuracy: 0.9861
Epoch 34/35
138/138 [==============================] - 76s 552ms/step - loss: 0.129
5 - accuracy: 0.9567 - val_loss: 0.0226 - val_accuracy: 0.9886
Epoch 35/35
138/138 [==============================] - 76s 552ms/step - loss: 0.120
8 - accuracy: 0.9585 - val_loss: 0.0217 - val_accuracy: 0.9954
```

Out[22]:

```
<keras.callbacks.History at 0x21c4fb80550>
```

In [23]:

```
#Evaluating the model
(ls,acc)=model.evaluate(x=X_test,y=y_test)

```

```
225/225 [==============================] - 6s 27ms/step - loss: 0.0217 - a
ccuracy: 0.9954
```

In [24]:

```
print('Model Accuracy is {}%'.format(acc*100))

```

```
Model Accuracy is 99.53987598419193%
```

In [ ]:

```

```