In [1]:
```
#importing libraries
import numpy as np
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout, BatchNormaliza
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

In [3]:
```
train_df = pd.read_csv('C:/Users/Priya/Documents/Deep Learning/archive 1/sign_mnist_te
test_df = pd.read_csv('C:/Users/Priya/Documents/Deep Learning/archive 1/sign_mnist_tra
```

In [4]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7172 entries, 0 to 7171
Columns: 785 entries, label to pixel784
dtypes: int64(785)
memory usage: 43.0 MB
```

In [5]:
```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27455 entries, 0 to 27454
Columns: 785 entries, label to pixel784
dtypes: int64(785)
memory usage: 164.4 MB
```

In [6]:
```
train_df.head()
```

Out[6]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776
0	6	149	149	150	150	150	151	151	150	151	...	138	148
1	5	126	128	131	132	133	134	135	135	136	...	47	104
2	10	85	88	92	96	105	123	135	143	147	...	68	166
3	0	203	205	207	206	207	209	210	209	210	...	154	248
4	3	188	191	193	195	199	201	202	203	203	...	26	40

5 rows × 785 columns

In [7]:
```
train_label = train_df['label']  #sign language gesture classes,he 'label' column is e
train_label.head(20)
```

Out[7]:
```
0      6
1      5
2     10
3      0
4      3
5     21
6     10
7     14
8      3
9      7
10     8
11     8
12    21
13    12
14     7
15     4
16    22
17     0
18     7
19     7
Name: label, dtype: int64
```

In [8]:
```
trainset=train_df.drop(['label'],axis=1)
trainset.head()
```

Out[8]:

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	...	pixel775	pixel776
0	149	149	150	150	150	151	151	150	151	152	...	138	148
1	126	128	131	132	133	134	135	135	136	138	...	47	104
2	85	88	92	96	105	123	135	143	147	152	...	68	166
3	203	205	207	206	207	209	210	209	210	209	...	154	248
4	188	191	193	195	199	201	202	203	203	203	...	26	40

5 rows × 784 columns

In [9]:
```
#Converting the dataframe to numpy array type to be used while
#training the CNN. The array is converted from 1-D to 3-D which is
#the required input to the first layer of the CNN. Similar #preprocessing is done to t
```

In [10]:
```
X_train = trainset.values
X_train = trainset.values.reshape(-1,28,28,1)
print(X_train.shape)
```

```
(7172, 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()
```

```
(27455, 784)
```

Out[11]:

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	...	pixel775	pixel776
0	107	118	127	134	139	143	146	150	153	156	...	207	207
1	155	157	156	156	156	157	156	158	158	157	...	69	149
2	187	188	188	187	187	186	187	188	187	186	...	202	201
3	211	211	212	212	211	210	211	210	210	211	...	235	234
4	164	167	170	172	176	179	180	184	185	186	...	92	105

5 rows × 784 columns

In [12]:
```
#Converting the integer labels to binary form
#The label dataframe consist of single values from 1 to 24 for each individual picture
#The CNN output layer will be of 24 nodes since it has 24 different labels as a multi
#Hence each integer is encoded in a binary array of size 24 with the corresponding lab
#Such as if y=4 the the array is [0 0 0 1 0 0.....0]. The LabelBinarizer package from
#preprocessing is used for that
```

In [13]:
```
from sklearn.preprocessing import LabelBinarizer
lb=LabelBinarizer()
y_train=lb.fit_transform(train_label)
y_test=lb.fit_transform(test_label)
```

In [14]:
```
y_train
```

Out[14]:
```
array([[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, 1, ..., 0, 0, 0]])
```

In [15]:
```
X_test=X_test.values.reshape(-1,28,28,1)
X_test.shape
```

Out[15]:
```
(27455, 28, 28, 1)
```

In [16]:
```
print(X_train.shape,y_train.shape,X_test.shape,y_test.shape)
```

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

In [17]:
```
#Augmenting the image dataset to generate new data
#ImageDataGenerator package from keras.preprocessing.image allows
#to add different distortions to image dataset by providing random
#rotation, zoom in/out , height or width scaling etc to images
#pixel by pixel.
```

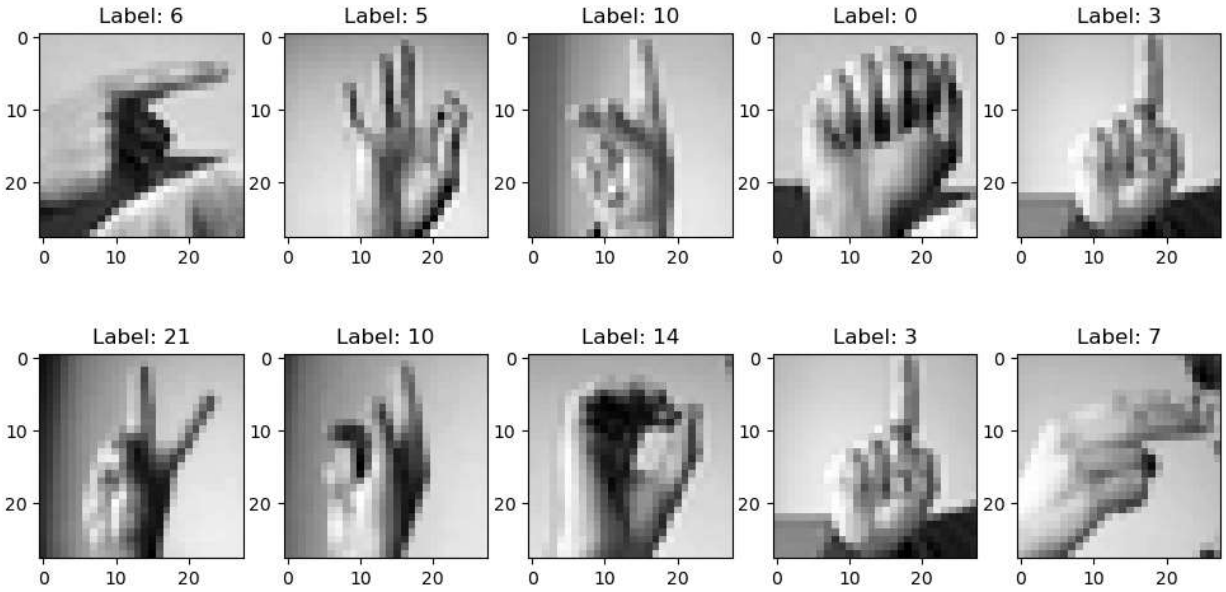
In [18]:
```
#The image dataset in also normalised here using the rescale
#parameter which divides each pixel by 255 such that the pixel
#values range between 0 to 1.
```

In [19]:
```
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
```

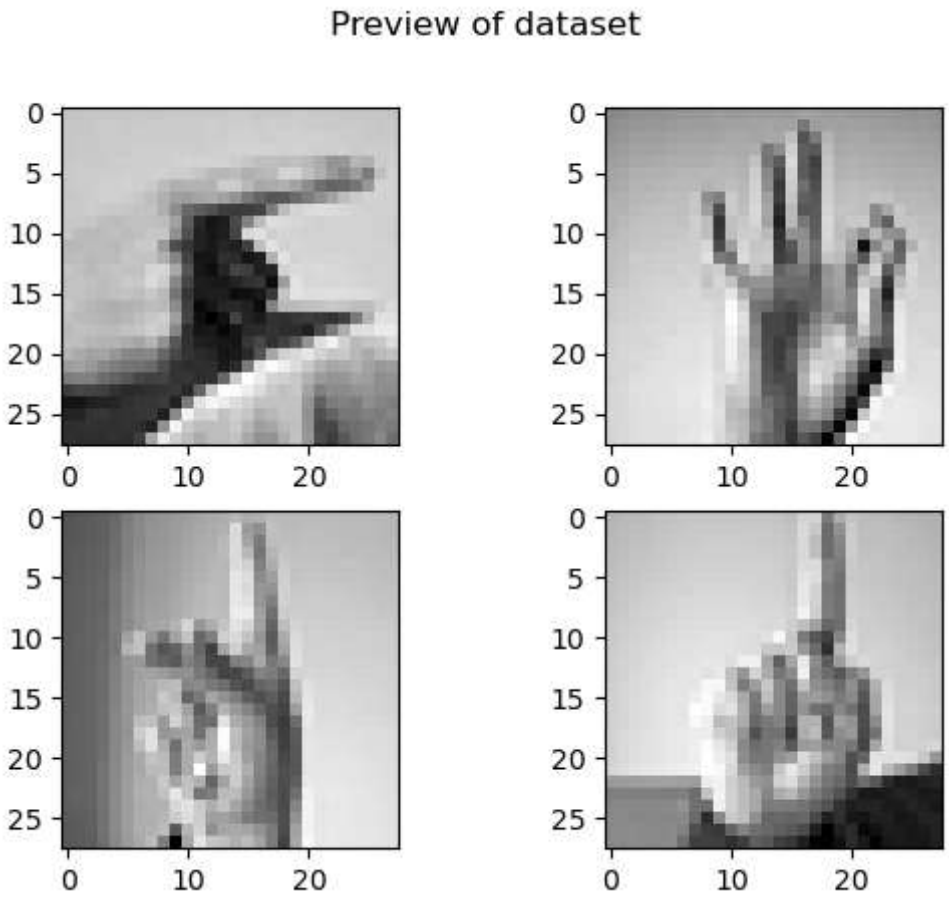
In [21]:
```
# Visualize a few samples from the dataset
fig, axes = plt.subplots(2, 5, figsize=(12, 6))
for i in range(10):
    ax = axes[i // 5, i % 5]
    ax.imshow(X_train[i].reshape(28, 28), cmap='gray')
    ax.set_title(f"Label: {train_label[i]}")
plt.show()
```



In [20]:
```
#Visualization

fig,axe=plt.subplots(2,2)
fig.suptitle('Preview of dataset')
axe[0,0].imshow(X_train[0].reshape(28,28),cmap='gray')
axe[0,1].imshow(X_train[1].reshape(28,28),cmap='gray')
axe[1,0].imshow(X_train[2].reshape(28,28),cmap='gray')
axe[1,1].imshow(X_train[4].reshape(28,28),cmap='gray')
```

Out[20]:
```
<matplotlib.image.AxesImage at 0x19dec93c430>
```



Building A CNN Model

The model consist of : Three convolution layer each followed bt MaxPooling for better feature capture A dense layer of 512 units The output layer with 24 units for 24 different classes

Convolution layers 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

In [22]:
```
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 [23]:
```
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 [24]:
```
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

In [30]:
```
history = 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
36/36 [==============================] - 51s 1s/step - loss: 0.4109 - accuracy: 0.864
6 - val_loss: 0.7312 - val_accuracy: 0.8075
Epoch 2/35
36/36 [==============================] - 59s 2s/step - loss: 0.4072 - accuracy: 0.863
6 - val_loss: 0.7628 - val_accuracy: 0.8072
Epoch 3/35
36/36 [==============================] - 53s 1s/step - loss: 0.3564 - accuracy: 0.881
5 - val_loss: 0.7058 - val_accuracy: 0.8359
Epoch 4/35
36/36 [==============================] - 59s 2s/step - loss: 0.3640 - accuracy: 0.879
1 - val_loss: 0.8187 - val_accuracy: 0.7897
Epoch 5/35
36/36 [==============================] - 57s 2s/step - loss: 0.3495 - accuracy: 0.886
9 - val_loss: 0.7745 - val_accuracy: 0.8227
Epoch 6/35
36/36 [==============================] - 55s 2s/step - loss: 0.3409 - accuracy: 0.886
9 - val_loss: 0.6893 - val_accuracy: 0.8234
Epoch 7/35
36/36 [==============================] - 55s 2s/step - loss: 0.3234 - accuracy: 0.890
7 - val_loss: 0.7532 - val_accuracy: 0.8160
Epoch 8/35
36/36 [==============================] - 54s 2s/step - loss: 0.3244 - accuracy: 0.894
9 - val_loss: 0.7271 - val_accuracy: 0.8232
Epoch 9/35
36/36 [==============================] - 54s 2s/step - loss: 0.3320 - accuracy: 0.887
8 - val_loss: 0.7235 - val_accuracy: 0.8298
Epoch 10/35
36/36 [==============================] - 55s 2s/step - loss: 0.2805 - accuracy: 0.908
5 - val_loss: 0.7555 - val_accuracy: 0.8188
Epoch 11/35
36/36 [==============================] - 56s 2s/step - loss: 0.2934 - accuracy: 0.900
7 - val_loss: 0.7649 - val_accuracy: 0.8254
Epoch 12/35
36/36 [==============================] - 55s 2s/step - loss: 0.2793 - accuracy: 0.904
4 - val_loss: 0.7802 - val_accuracy: 0.8178
Epoch 13/35
36/36 [==============================] - 55s 2s/step - loss: 0.2991 - accuracy: 0.901
1 - val_loss: 0.7073 - val_accuracy: 0.8312
Epoch 14/35
36/36 [==============================] - 57s 2s/step - loss: 0.2756 - accuracy: 0.908
5 - val_loss: 0.7675 - val_accuracy: 0.8040
Epoch 15/35
36/36 [==============================] - 57s 2s/step - loss: 0.3308 - accuracy: 0.887
8 - val_loss: 0.8024 - val_accuracy: 0.8182
Epoch 16/35
36/36 [==============================] - 60s 2s/step - loss: 0.2944 - accuracy: 0.902
1 - val_loss: 0.7077 - val_accuracy: 0.8256
Epoch 17/35
36/36 [==============================] - 55s 2s/step - loss: 0.2673 - accuracy: 0.911
2 - val_loss: 0.6757 - val_accuracy: 0.8349
Epoch 18/35
36/36 [==============================] - 56s 2s/step - loss: 0.2535 - accuracy: 0.916
2 - val_loss: 0.7296 - val_accuracy: 0.8390
Epoch 19/35
36/36 [==============================] - 54s 2s/step - loss: 0.2364 - accuracy: 0.920
5 - val_loss: 0.7294 - val_accuracy: 0.8317
Epoch 20/35
36/36 [==============================] - 58s 2s/step - loss: 0.2262 - accuracy: 0.923
3 - val_loss: 0.8070 - val_accuracy: 0.8307
```

```
Epoch 21/35
36/36 [==============================] - 53s 1s/step - loss: 0.2285 - accuracy: 0.921
1 - val_loss: 0.7427 - val_accuracy: 0.8397
Epoch 22/35
36/36 [==============================] - 54s 2s/step - loss: 0.2302 - accuracy: 0.923
5 - val_loss: 0.6475 - val_accuracy: 0.8444
Epoch 23/35
36/36 [==============================] - 52s 1s/step - loss: 0.2159 - accuracy: 0.928
2 - val_loss: 0.7196 - val_accuracy: 0.8378
Epoch 24/35
36/36 [==============================] - 54s 2s/step - loss: 0.2110 - accuracy: 0.932
5 - val_loss: 0.7977 - val_accuracy: 0.8366
Epoch 25/35
36/36 [==============================] - 53s 1s/step - loss: 0.2092 - accuracy: 0.933
2 - val_loss: 0.6847 - val_accuracy: 0.8505
Epoch 26/35
36/36 [==============================] - 55s 2s/step - loss: 0.2160 - accuracy: 0.926
9 - val_loss: 0.7515 - val_accuracy: 0.8331
Epoch 27/35
36/36 [==============================] - 54s 2s/step - loss: 0.1970 - accuracy: 0.936
7 - val_loss: 0.6950 - val_accuracy: 0.8469
Epoch 28/35
36/36 [==============================] - 55s 2s/step - loss: 0.1940 - accuracy: 0.935
9 - val_loss: 0.8237 - val_accuracy: 0.8219
Epoch 29/35
36/36 [==============================] - 53s 2s/step - loss: 0.1960 - accuracy: 0.933
5 - val_loss: 0.6896 - val_accuracy: 0.8466
Epoch 30/35
36/36 [==============================] - 54s 2s/step - loss: 0.1999 - accuracy: 0.930
1 - val_loss: 0.7534 - val_accuracy: 0.8465
Epoch 31/35
36/36 [==============================] - 53s 1s/step - loss: 0.1755 - accuracy: 0.941
0 - val_loss: 0.7842 - val_accuracy: 0.8409
Epoch 32/35
36/36 [==============================] - 54s 2s/step - loss: 0.1825 - accuracy: 0.943
0 - val_loss: 0.7910 - val_accuracy: 0.8485
Epoch 33/35
36/36 [==============================] - 51s 1s/step - loss: 0.1799 - accuracy: 0.938
9 - val_loss: 0.7741 - val_accuracy: 0.8349
Epoch 34/35
36/36 [==============================] - 55s 2s/step - loss: 0.1661 - accuracy: 0.943
5 - val_loss: 0.7005 - val_accuracy: 0.8542
Epoch 35/35
36/36 [==============================] - 54s 2s/step - loss: 0.1654 - accuracy: 0.945
1 - val_loss: 0.8078 - val_accuracy: 0.8421
```

Evaluate the Model

In [31]:
```
(loss,acc)=model.evaluate(x=X_test,y=y_test)
```

```
858/858 [==============================] - 29s 34ms/step - loss: 0.8078 - accuracy:
0.8421
```

In [32]:
```
print('The accuracy of the model for testing data is:',acc*100)
print('The Loss of the model for testing data is:',loss)
```

```
The accuracy of the model for testing data is: 84.20688509941101
The Loss of the model for testing data is: 0.8078429698944092
```

In [33]:
```
# Visualize training history
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```