Handwritten Character Recognition using EMNIST Dataset

Import the main libraries needed for the project
emnist -- dataset
tensorflow -- libraries useful for machine learning
matplotlib -- data visualization
PIL -- image grabbing and manipulation
keras -- api for machine learning

In [1]:
```
from emnist import list_datasets
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from PIL import ImageGrab, Image
from tensorflow import keras
```

List the datasets available for the EMNIST

Balanced -- Alphanumeric database containing 814,255 units, 697,932 train, 116,323 test. 47 different classes to identify characters. Certain alphabets have been combined who have similar-looking UPPER and LOWER case image. Each class contains an equal amount of data to be used for training/testing
ByClass -- Alphanumeric database containing 814,255 units, 697,932 train, 116,323 test. 62 different classes to identify characters, 10 for digits, 26 for upper-case, 26 for lower-case. Certain classes have more units due to frequeuncy of usage in the english language
ByMerge -- Alphanumeric database containing 814,255 units, 697,932 train, 116,323 test. 47 different classes to identify characters. Certain alphabets have been combined who have similar-looking UPPER and LOWER case image. Certain classes have more units due to frequeunct of usage in the english language
Digits -- Larger database for digits. 280,000 units, 240,000 train, 40,000 test
Letters -- Database for only alphabetical characters. 103,600 units, 88,800 train, 14,800 test
MNIST -- Default Database for digits. 70,000 units, 60000 train, 10,000 test

In [2]:
```
list_datasets()
```

Out[2]:
```
['balanced', 'byclass', 'bymerge', 'digits', 'letters', 'mnist']
```

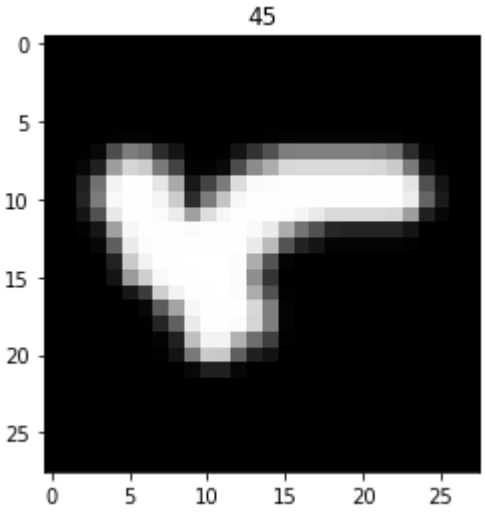
In [3]:
```
from emnist import extract_training_samples
from emnist import extract_test_samples
```

In [4]:
```
x_train, y_train = extract_training_samples('balanced')  # split the train/test data x_
x_test, y_test = extract_test_samples('balanced')
```

In [5]:
```
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

```
(112800, 28, 28) (112800,) (18800, 28, 28) (18800,)
```

In [6]:
```
plt.imshow(x_train[0], cmap=plt.get_cmap('gray'))
plt.title(y_train[0])
plt.show()
# index 0 of y_train represents the 45th value of the 47 classes, in this case lowercas
```



In [7]:
```
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)  # reshape data to 28x28 in 1 di
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
x_train = x_train.astype('float32')  # make the data include float values, and constrai
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

Complex model with several layers to filter and classiy the data

In [13]:
```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(filters=6, kernel_size=(3, 3), activation='relu', input_shape=
  tf.keras.layers.AveragePooling2D(),
  tf.keras.layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'),
  tf.keras.layers.AveragePooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(188, activation='relu'),
  tf.keras.layers.Dense(94, activation='relu'),
  tf.keras.layers.Dense(47, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=10)
loss, accuracy = model.evaluate(x_test, y_test)
model.save('sample.model')
model.summary()
```

```
Epoch 1/10
3525/3525 [==============================] - 13s 4ms/step - loss: 0.8059 - accuracy: 0.7
481
Epoch 2/10
3525/3525 [==============================] - 13s 4ms/step - loss: 0.4682 - accuracy: 0.8
380
```

```
Epoch 3/10
3525/3525 [==============================] - 13s 4ms/step - loss: 0.4022 - accuracy: 0.8
566
Epoch 4/10
3525/3525 [==============================] - 13s 4ms/step - loss: 0.3622 - accuracy: 0.8
690
Epoch 5/10
3525/3525 [==============================] - 13s 4ms/step - loss: 0.3354 - accuracy: 0.8
764
Epoch 6/10
3525/3525 [==============================] - 13s 4ms/step - loss: 0.3131 - accuracy: 0.8
838
Epoch 7/10
3525/3525 [==============================] - 13s 4ms/step - loss: 0.2976 - accuracy: 0.8
871
Epoch 8/10
3525/3525 [==============================] - 13s 4ms/step - loss: 0.2790 - accuracy: 0.8
929
Epoch 9/10
3525/3525 [==============================] - 13s 4ms/step - loss: 0.2655 - accuracy: 0.8
967
Epoch 10/10
3525/3525 [==============================] - 13s 4ms/step - loss: 0.2549 - accuracy: 0.9
003
588/588 [==============================] - 1s 2ms/step - loss: 0.3977 - accuracy: 0.8695
INFO:tensorflow:Assets written to: sample.model\assets
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 26, 26, 6)         60
_________________________________________________________________
average_pooling2d_2 (Average (None, 13, 13, 6)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 11, 11, 16)        880
_________________________________________________________________
average_pooling2d_3 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_3 (Dense)              (None, 188)               75388
_________________________________________________________________
dense_4 (Dense)              (None, 94)                17766
_________________________________________________________________
dense_5 (Dense)              (None, 47)                4465
=================================================================
Total params: 98,559
Trainable params: 98,559
Non-trainable params: 0
_________________________________________________________________
```

Basic model using only dense layers to filter the data and classify the data. Each layer categorizes the data, aiming for more accuracy each time

In [9]:
```
# model = tf.keras.models.Sequential([
#   tf.keras.layers.Flatten(input_shape=(28, 28, 1)),
#   tf.keras.layers.Dense(512, activation='relu'),
#   tf.keras.layers.Dense(256, activation='relu'),
#   tf.keras.layers.Dense(188, activation='relu'),
#   tf.keras.layers.Dense(94, activation='relu'),
#   tf.keras.layers.Dense(47, activation='softmax')
# ])

# model.compile(optimizer='adam',
```

```
#               loss='sparse_categorical_crossentropy',
#               metrics=['accuracy'])

# model.fit(x_train, y_train, epochs=10)
# loss, accuracy = model.evaluate(x_test, y_test)
# model.save('sample.model')
```

Using the mapping text file given with the dataset to classify the images based on their index.
Each mapping index corresponds to an ASCII value (0-9, A-Z, a, b, d, e, f, g, h, n, q, r, t)

In [10]:
```
mapping = np.loadtxt('emnist-balanced-mapping.txt',dtype=int, usecols=(1), unpack=True)
print(mapping)
char_labels={}
for i in range(47):
    char_labels[i] = chr(mapping[i])
print(char_labels)
```

```
[ 48  49  50  51  52  53  54  55  56  57  65  66  67  68  69  70  71  72
  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
  97  98 100 101 102 103 104 110 113 114 116]
{0: '0', 1: '1', 2: '2', 3: '3', 4: '4', 5: '5', 6: '6', 7: '7', 8: '8', 9: '9', 10:
'A', 11: 'B', 12: 'C', 13: 'D', 14: 'E', 15: 'F', 16: 'G', 17: 'H', 18: 'I', 19: 'J', 2
0: 'K', 21: 'L', 22: 'M', 23: 'N', 24: 'O', 25: 'P', 26: 'Q', 27: 'R', 28: 'S', 29: 'T',
30: 'U', 31: 'V', 32: 'W', 33: 'X', 34: 'Y', 35: 'Z', 36: 'a', 37: 'b', 38: 'd', 39:
'e', 40: 'f', 41: 'g', 42: 'h', 43: 'n', 44: 'q', 45: 'r', 46: 't'}
```

Only need to run bottom line inorder to load the model. Do not need to train the model each time

In [11]:
```
from tensorflow import keras
from PIL import ImageGrab, Image
import numpy as np
model = keras.models.load_model('emnist.model')
```

Image testing with drawn images from tkinter gui. Each image is unique, 0-9, A-Z, then lower case letters

In [12]:
```
for x in range(0,47):
    img = Image.open(f'images/{x}.png')
    img = np.array(img)
    img = np.invert(np.array([img]))
    img = img.reshape(1,28,28,1)
    img = img/255
    predict = model.predict(img)  # identify the class which resonates highest with the
    print(char_labels[np.argmax(predict)])
#     Testing with 10 pixel lines
```

```
0
L
2
3
4
5
G
7
8
9
A
8
C
D
E
F
```

G
H
I
J
K
L
M
N
0
P
Q
R
S
T
U
V
W
X
Y
Z
a
b
d
E
F
g
h
M
q
t
T