In [2]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from keras import layers, models
from keras.preprocessing.image import ImageDataGenerator
from keras_preprocessing.image import load_img
from sklearn.model_selection import train_test_split
import matplotlib.image as mpimg
import zipfile
import os
import re
```

In [3]:
```
IMAGE_SHAPE = (120, 120, 1)
```

1. Collect images of handwritten letters of one of the Indian languages

In [4]:
```
import os
for dirname, _, filenames in os.walk('D:\EDUCATION\handwritten recog\shuffled'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
D:\EDUCATION\handwritten recog\shuffled\u16_000t01.tiff
D:\EDUCATION\handwritten recog\shuffled\u16_000t02.tiff
D:\EDUCATION\handwritten recog\shuffled\u16_000t03.tiff
D:\EDUCATION\handwritten recog\shuffled\u16_000t04.tiff
D:\EDUCATION\handwritten recog\shuffled\u16_000t05.tiff
D:\EDUCATION\handwritten recog\shuffled\u16_000t06.tiff
D:\EDUCATION\handwritten recog\shuffled\u16_000t07.tiff
D:\EDUCATION\handwritten recog\shuffled\u16_000t08.tiff
D:\EDUCATION\handwritten recog\shuffled\u16_000t09.tiff
D:\EDUCATION\handwritten recog\shuffled\u16_000t10.tiff
D:\EDUCATION\handwritten recog\shuffled\u16_001t01.tiff
D:\EDUCATION\handwritten recog\shuffled\u16_001t02.tiff
D:\EDUCATION\handwritten recog\shuffled\u16_001t03.tiff
D:\EDUCATION\handwritten recog\shuffled\u16_001t04.tiff
D:\EDUCATION\handwritten recog\shuffled\u16_001t05.tiff
D:\EDUCATION\handwritten recog\shuffled\u16_001t06.tiff
D:\EDUCATION\handwritten recog\shuffled\u16_001t07.tiff
D:\EDUCATION\handwritten recog\shuffled\u16_001t08.tiff
D:\EDUCATION\handwritten recog\shuffled\u16_001t09.tiff
```

In [5]:
```
TRAIN_PATH = 'D:\EDUCATION\handwritten recog\shuffled'
print('Number of images in the dataset: ', len(os.listdir(TRAIN_PATH)))
```

```
Number of images in the dataset:  3000
```

In [6]:
```
files = os.listdir(TRAIN_PATH)
target = []
category = []
for filename in os.listdir(TRAIN_PATH):
  substr = re.search('_(.+?)t', filename)
  if(substr):
    category = substr.group(1)
    target.append(category)
```

In [7]:
```
df = pd.DataFrame({
    'filename': files,
    'category': target
})
```

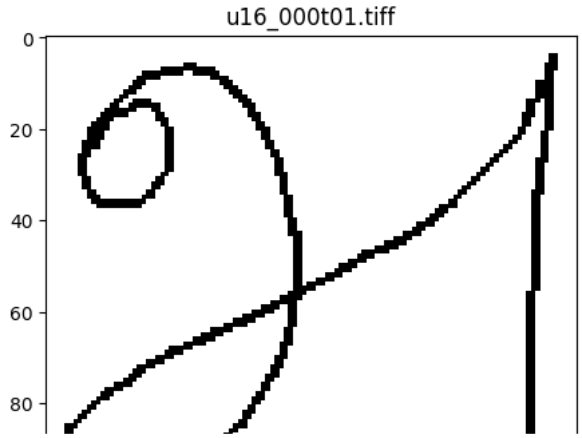
In [9]:
```
print(df)
```

```
             filename category
0     u16_000t01.tiff      000
1     u16_000t02.tiff      000
2     u16_000t03.tiff      000
3     u16_000t04.tiff      000
4     u16_000t05.tiff      000
...               ...      ...
2995  u54_009t01.tiff      009
2996  u54_009t02.tiff      009
2997  u54_010t01.tiff      010
2998  u54_010t02.tiff      010
2999  u55_000t01.tiff      000

[3000 rows x 2 columns]
```

In [10]:
```
print('Number of unique characters: {}'.format(df['category'].unique()))
```

```
Number of unique characters: ['000' '001' '002' '003' '004' '005' '006' '007' '008' '009' '010' '155']
```

In [11]:
```
for each in df['category'].unique():
  filename = df[df['category'] == each]['filename'].iloc[0]
  plt.figure()
  img = mpimg.imread(os.path.join(TRAIN_PATH, filename))
  plt.imshow(img)
  plt.title(filename)
  plt.show()
```



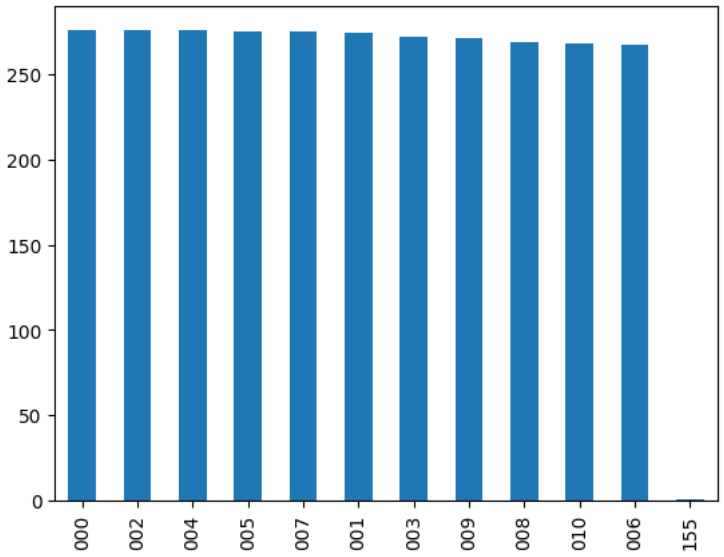
In [12]:
```
MAP = {
    '000':u'\u0B85',
    '001':u'\u0B86',
    '002':u'\u0B87',
    '003':u'\u0B88',
    '004':u'\u0B89',
    '005':u'\u0B8A',
    '006':u'\u0B8E',
    '007':u'\u0B8F',
    '008':u'\u0B90',
    '009':u'\u0B92',
    '010':u'\u0B93',
    '155':u'\u0B94'
    }
```

In [13]:
```
MAP.items()
```

Out[13]:
```
dict_items([('000', 'அ'), ('001', 'ஆ'), ('002', 'இ'), ('003', 'ஈ'), ('004', 'உ'), ('005', 'ஊ'), ('006', 'எ'), ('007', 'ஏ'), ('008', 'ஐ'), ('009', 'ஒ'), ('010', 'ஓ'), ('155', 'ஔ')])
```

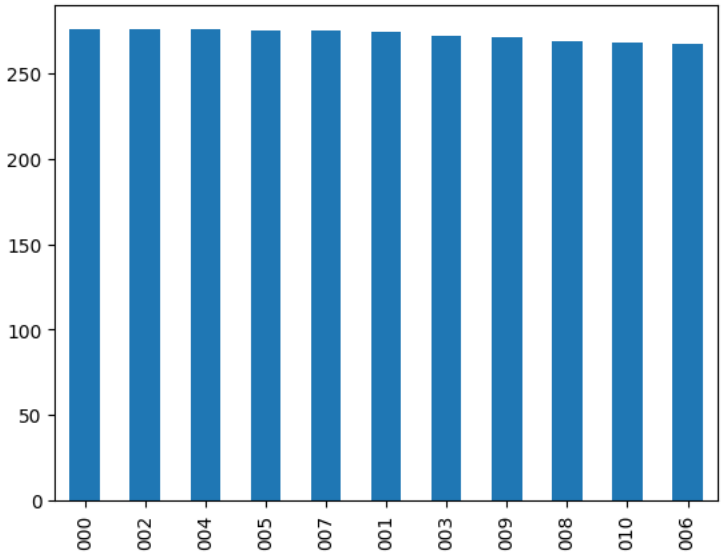
In [14]:
```
df['category'].value_counts().plot.bar()
```

Out[14]:
```
<AxesSubplot: >
```



In [15]:
```
df.drop(df[df['category'] == '155'].index, inplace = True)
df['category'].value_counts().plot.bar()
```

Out[15]: <AxesSubplot: >



In [16]:
```
for filename in df[df['category']=='006']['filename']:
  plt.figure()
  img = mpimg.imread(os.path.join(TRAIN_PATH, filename))
  plt.imshow(img)
  plt.title(filename)
  plt.show()
```

In [17]:
```
height, width, depth = [], [], []
for filename in df['filename']:
  img = mpimg.imread(os.path.join(TRAIN_PATH, filename))
  h, w, d = img.shape
  height.append(h)
  width.append(w)
  depth.append(d)

dim_df = pd.DataFrame({
    'height': height,
    'width': width,
    'depth': depth
})
```

In [18]:
```
dim_df.describe()
```

Out[18]:

	height	width	depth
count	2999.000000	2999.000000	2999.0
mean	107.107036	136.556185	4.0
std	27.767347	44.495558	0.0
min	42.000000	40.000000	4.0
25%	87.000000	105.000000	4.0
50%	106.000000	128.000000	4.0
75%	125.000000	161.000000	4.0
max	234.000000	406.000000	4.0

2. Split the data into train and test tests (80% train and 20% test)

In [46]:
```
df1 = pd.get_dummies(df, columns = ['filename', 'category'])
```

In [47]:
```
df1
```

Out[47]:

ff	filename_u16_000t10.tiff	...	category_001	category_002	category_003	category_004	category_005	category_006	category_007	category_008	category_009	category_010
0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	...	0	0	0	0	0	0	0	0	0	0
0	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	1	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	0	0	0	0	0	0	0	0	0

In [70]:
```
X = df['filename']
y = df['category']
```

In [73]:
```
y1 =  df1.iloc[:, -10:].values
X1 = df1.iloc[ 1: , :-10].values
```

In [68]:
```
X
```

Out[68]:
```
array([[0, 1, 0, ..., 0, 0, 1],
       [0, 0, 1, ..., 0, 0, 1],
       [0, 0, 0, ..., 0, 0, 1],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 1, 0, 0],
       [0, 0, 0, ..., 0, 1, 1]], dtype=uint8)
```

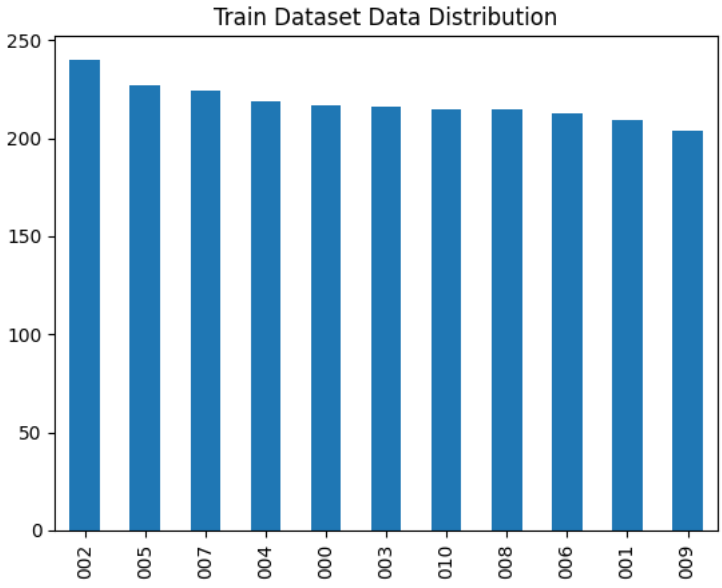
In [71]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, train_size = .80)
```

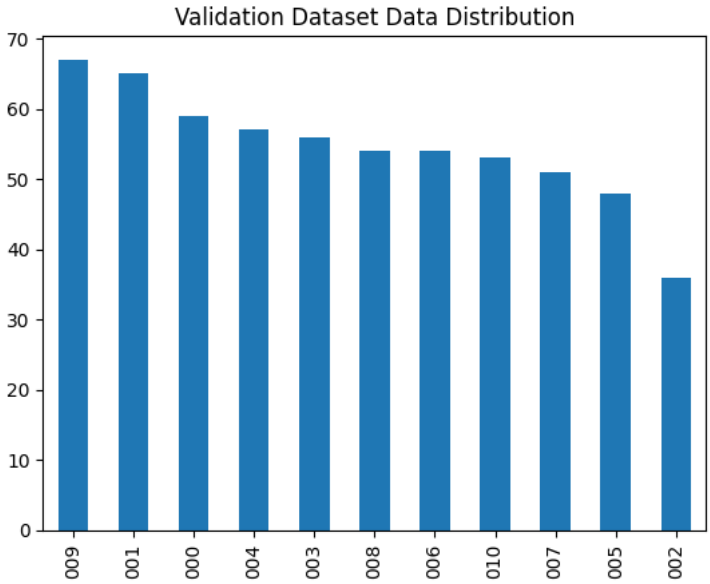
In [72]:
```
print('Train Dataset Size: ', len(X_train))
print('Validation Dataset Size: ', len(X_test))

print('Train Dataset Size: ', len(y_train))
print('Validation Dataset Size: ', len(y_test))
```

```
Train Dataset Size:  2399
Validation Dataset Size:  600
Train Dataset Size:  2399
Validation Dataset Size:  600
```

In [27]:
```
train_df, test_df = train_test_split(df, test_size=0.2, random_state=28)
train_df = train_df.reset_index(drop=True)
test_df = test_df.reset_index(drop=True)
```

In [28]:
```
train_df['category'].value_counts().plot.bar()
plt.title('Train Dataset Data Distribution')
plt.show()

plt.figure()

test_df['category'].value_counts().plot.bar()
plt.title('Validation Dataset Data Distribution')
plt.show()
```





3. Use dimension reduction techniques, PCA and t-SNE to reduce the dimensions of the data

In [29]:
```
from sklearn.decomposition import PCA
import seaborn as sns
from sklearn.preprocessing import StandardScaler
```

In [31]:
```
scaler = StandardScaler()
scaler.fit(df1)
```

Out[31]:
```
StandardScaler
StandardScaler()
```

In [54]:
```
scaled_data = scaler.transform(df1)
```

PCA - Dimension Reduction

In [55]:
```
pca = PCA(n_components=2)
```

In [56]:
```
pca.fit(scaled_data)
```

Out[56]:
```
PCA
PCA(n_components=2)
```

In [57]:
```
x_pca = pca.transform(scaled_data)
```

In [58]:
```
scaled_data.shape
```

Out[58]:
```
(2999, 3010)
```

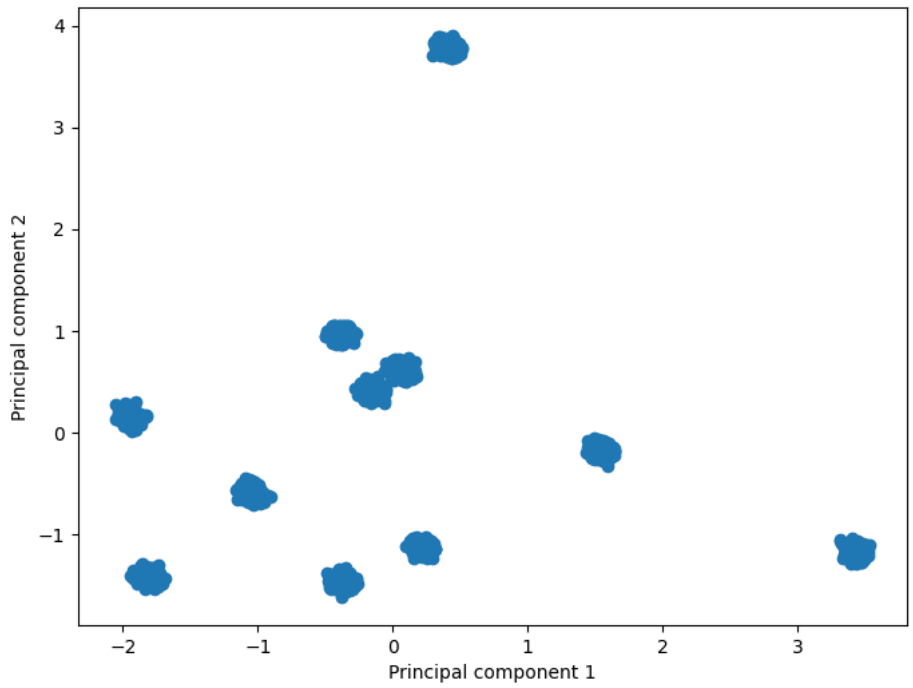
In [59]:
```
x_pca.shape
```

Out[59]:
```
(2999, 2)
```

In [40]:
```
plt.figure(figsize=(8,6))
plt.scatter(x_pca[:,0],x_pca[:,1], c=None, cmap='plasma')
plt.xlabel('Principal component 1')
plt.ylabel('Principal component 2')
```

```
C:\Users\yokes\AppData\Local\Temp\ipykernel_32576\898624224.py:2: UserWarning: No data for colormapping provided via 'c'. Parameters 'cma
p' will be ignored
  plt.scatter(x_pca[:,0],x_pca[:,1], c=None, cmap='plasma')
```

Out[40]:
```
Text(0, 0.5, 'Principal component 2')
```



t-SNE -Dimension Reduction

In [90]:
```
import time
from sklearn.manifold import TSNE
time_start = time.time()
tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300)
tsne_results = tsne.fit_transform(df1)

print('t-SNE done! Time elapsed: {} seconds'.format(time.time()-time_start))
```

```
[t-SNE] Computing 121 nearest neighbors...
[t-SNE] Indexed 2999 samples in 0.159s...
[t-SNE] Computed neighbors for 2999 samples in 1.940s...
[t-SNE] Computed conditional probabilities for sample 1000 / 2999
[t-SNE] Computed conditional probabilities for sample 2000 / 2999
[t-SNE] Computed conditional probabilities for sample 2999 / 2999
[t-SNE] Mean sigma: 0.051846
[t-SNE] KL divergence after 250 iterations with early exaggeration: 33.144661
[t-SNE] KL divergence after 300 iterations: 0.117815
t-SNE done! Time elapsed: 19.456538200378418 seconds
```

4. Train a classification model using neural networks and support vector machines on the training data

In [93]:
```
batch_size = 5
epoch = 50

train_count = train_df.shape[0]
test_count = test_df.shape[0]

train_datagen = ImageDataGenerator(
    rescale = 1./255,
    # rotation_range = 10,
    # width_shift_range = 0.2,
    # height_shift_range = 0.2,
    # shear_range = 0.2,
    horizontal_flip=False,
    fill_mode='nearest',
    )

test_datagen = ImageDataGenerator(
    rescale = 1./255,
    # rotation_range = 10,
    # width_shift_range = 0.2,
    # height_shift_range = 0.2,
    # shear_range = 0.2,
    horizontal_flip=False,
    fill_mode='nearest',
    )

train_gen = train_datagen.flow_from_dataframe(
    train_df,
    directory = TRAIN_PATH,
    x_col = 'filename',
    y_col = 'category',
    class_mode = 'categorical',
    target_size = IMAGE_SHAPE[:2],
    batch_size = batch_size,
    color_mode='grayscale'
)

test_gen = train_datagen.flow_from_dataframe(
    test_df,
    directory = TRAIN_PATH,
    x_col = 'filename',
    y_col = 'category',
    class_mode = 'categorical',
    target_size = IMAGE_SHAPE[:2],
    batch_size = batch_size,
    color_mode='grayscale'
)
```

```
Found 2399 validated image filenames belonging to 11 classes.
Found 600 validated image filenames belonging to 11 classes.
```

In [42]:
```
def build_model():
  model = models.Sequential()

  model.add(layers.Conv2D(32, (5, 5), activation='relu', input_shape=IMAGE_SHAPE))
  model.add(layers.BatchNormalization())
  model.add(layers.MaxPool2D(pool_size=(2, 2)))
  model.add(layers.Dropout(0.2))

  model.add(layers.Conv2D(32, (5, 5), activation='relu'))
  model.add(layers.BatchNormalization())
  model.add(layers.MaxPool2D(pool_size=(2, 2)))
  model.add(layers.Dropout(0.2))

  model.add(layers.Conv2D(32, (5, 5), activation='relu'))
  model.add(layers.BatchNormalization())
  model.add(layers.MaxPool2D(pool_size=(2, 2)))
  model.add(layers.Dropout(0.2))

  model.add(layers.Conv2D(64, (6, 6), activation='relu'))
  model.add(layers.BatchNormalization())
  model.add(layers.MaxPool2D(pool_size=(2, 2)))
  model.add(layers.Dropout(0.2))

  model.add(layers.Flatten())
  model.add(layers.Dense(256, activation='relu'))
  model.add(layers.BatchNormalization())
  model.add(layers.Dense(11, activation='softmax'))

  model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
  return model

model = build_model()
```

In [43]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 116, 116, 32)      832

 batch_normalization (BatchN  (None, 116, 116, 32)     128
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 58, 58, 32)       0
 )

 dropout (Dropout)           (None, 58, 58, 32)        0

 conv2d_1 (Conv2D)           (None, 54, 54, 32)        25632

 batch_normalization_1 (Batc  (None, 54, 54, 32)       128
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 27, 27, 32)       0
 2D)

 dropout_1 (Dropout)         (None, 27, 27, 32)        0

 conv2d_2 (Conv2D)           (None, 23, 23, 32)        25632

 batch_normalization_2 (Batc  (None, 23, 23, 32)       128
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 11, 11, 32)       0
 2D)

 dropout_2 (Dropout)         (None, 11, 11, 32)        0

 conv2d_3 (Conv2D)           (None, 6, 6, 64)          73792

 batch_normalization_3 (Batc  (None, 6, 6, 64)         256
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 3, 3, 64)         0
 2D)

 dropout_3 (Dropout)         (None, 3, 3, 64)          0

 flatten (Flatten)           (None, 576)               0

 dense (Dense)               (None, 256)               147712

 batch_normalization_4 (Batc  (None, 256)              1024
 hNormalization)

 dense_1 (Dense)             (None, 11)                2827

=================================================================
Total params: 278,091
Trainable params: 277,259
Non-trainable params: 832
_________________________________________________________________
```

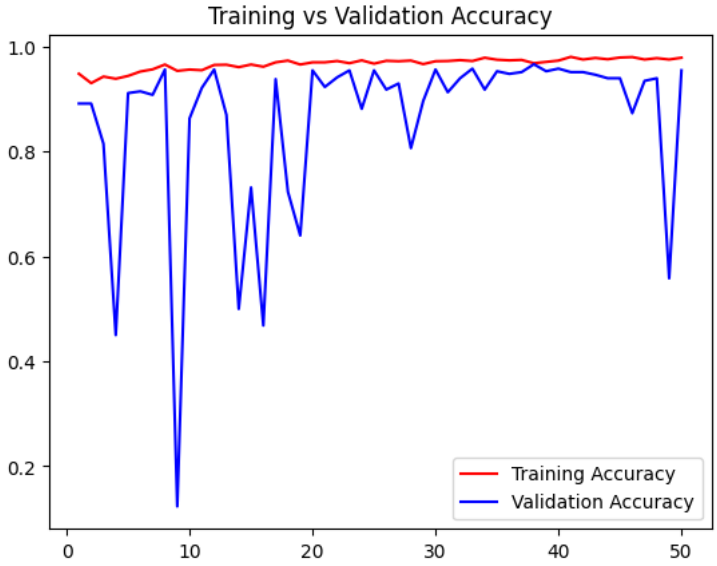
In [91]:
```
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau

earlystop = EarlyStopping(patience=10)
lrreducuction = ReduceLROnPlateau(monitor='val_loss', factor=0.05, patience=2, verbose=1, min_lr=0.000005)
filepath = "checkpoint.h5"
checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1, save_best_only=True, mode='min')

callbacks = [earlystop, lrreducuction, checkpoint]
```

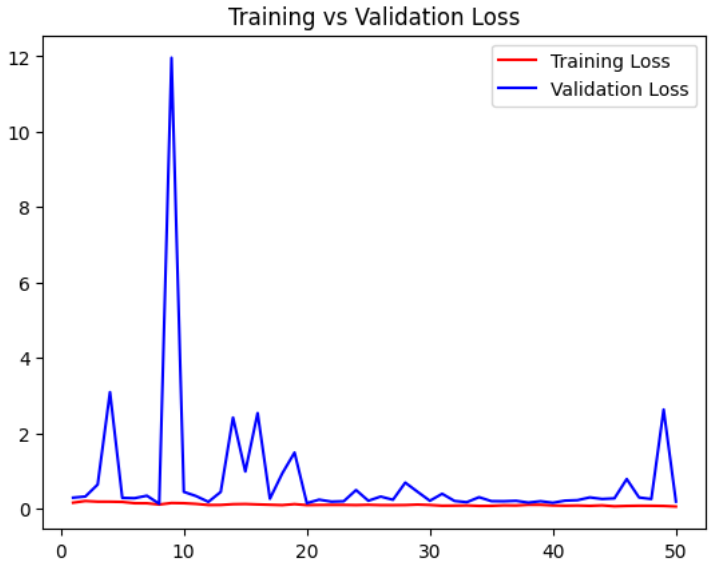
5. Validate the model on the test data

In [100]:
```
history = model.fit(
    train_gen,
    epochs=50,
    steps_per_epoch = train_count // batch_size,
    validation_data = test_gen,
    validation_steps = test_count // batch_size,
    #callbacks = callbacks
)
```

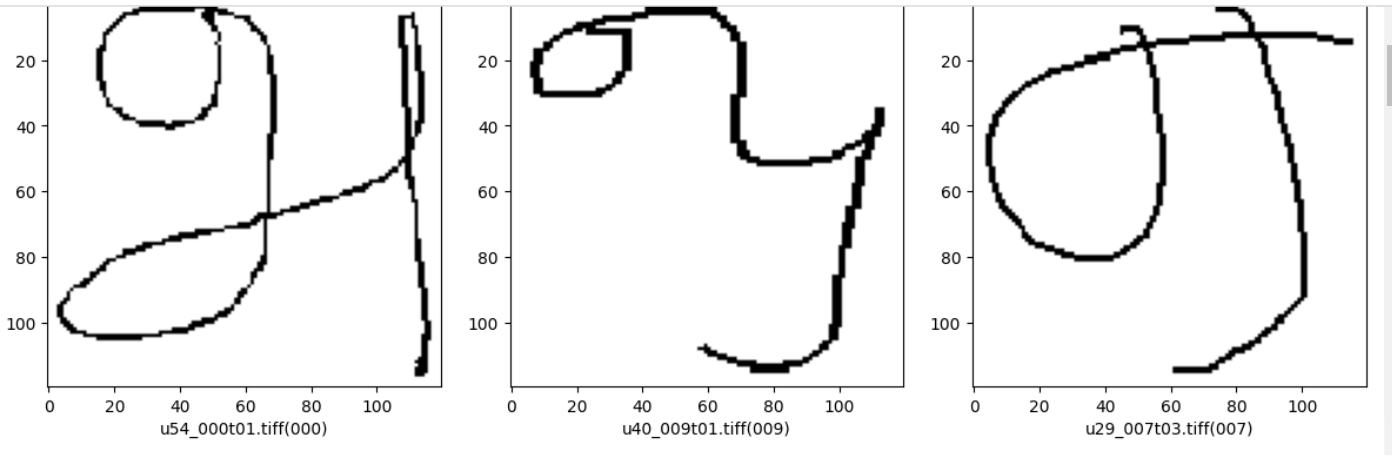
```
Epoch 26/50
479/479 [==============================] - 56s 118ms/step - loss: 0.0961 - accuracy: 0.9733 - val_loss: 0.3231 - val_accuracy: 0.9183
Epoch 27/50
479/479 [==============================] - 51s 107ms/step - loss: 0.0953 - accuracy: 0.9724 - val_loss: 0.2383 - val_accuracy: 0.9300
Epoch 28/50
479/479 [==============================] - 54s 112ms/step - loss: 0.0979 - accuracy: 0.9737 - val_loss: 0.6932 - val_accuracy: 0.8067
Epoch 29/50
479/479 [==============================] - 1118s 2s/step - loss: 0.1105 - accuracy: 0.9670 - val_loss: 0.4498 - val_accuracy: 0.8967
Epoch 30/50
479/479 [==============================] - 63s 132ms/step - loss: 0.0994 - accuracy: 0.9724 - val_loss: 0.2117 - val_accuracy: 0.9567
Epoch 31/50
479/479 [==============================] - 55s 114ms/step - loss: 0.0800 - accuracy: 0.9728 - val_loss: 0.3971 - val_accuracy: 0.9133
Epoch 32/50
479/479 [==============================] - 763s 2s/step - loss: 0.0827 - accuracy: 0.9745 - val_loss: 0.2071 - val_accuracy: 0.9400
Epoch 33/50
479/479 [==============================] - 50s 104ms/step - loss: 0.0879 - accuracy: 0.9728 - val_loss: 0.1741 - val_accuracy: 0.9583
Epoch 34/50
479/479 [==============================] - 46s 97ms/step - loss: 0.0750 - accuracy: 0.9791 - val_loss: 0.3065 - val_accuracy: 0.9183
Epoch 35/50
479/479 [==============================] - 55s 115ms/step - loss: 0.0769 - accuracy: 0.9754 - val_loss: 0.2018 - val_accuracy: 0.9533
```

In [102]:
```
epoch_xaxis = range(1, len(history.history['accuracy'])+1)

plt.plot(epoch_xaxis, history.history['accuracy'], 'r', label='Training Accuracy')
plt.plot(epoch_xaxis, history.history['val_accuracy'], 'b', label='Validation Accuracy')
plt.title('Training vs Validation Accuracy')
plt.legend()
plt.show()

plt.figure()

plt.plot(epoch_xaxis, history.history['loss'], 'r', label='Training Loss')
plt.plot(epoch_xaxis, history.history['val_loss'], 'b', label='Validation Loss')
plt.title('Training vs Validation Loss')
plt.legend()
plt.show()
```





In [103]:
```
sample_test = test_df.head(18)
sample_test.head()
plt.figure(figsize=(12, 24))
for index, row in sample_test.iterrows():
    filename = row['filename']
    category = row['category']
    img = load_img(os.path.join(TRAIN_PATH,filename), target_size=IMAGE_SHAPE)
    plt.subplot(6, 3, index+1)
    plt.imshow(img)
    plt.xlabel(filename + '(' + "{}".format(category) + ')' )
plt.tight_layout()
plt.show()
```



6. Fine tune the parameters to increase the classification accuracies of the model on training and test data.

In [104]:
```
for index, row in sample_test.iterrows():
    filename = row['filename']
    category = row['category']
    print(filename + ' ==> ' + MAP[category])
```

```
u54_000t01.tiff ==> அ
u40_009t01.tiff ==> ஓ
u29_007t03.tiff ==> ஏ
u45_003t01.tiff ==> ஈ
u19_007t06.tiff ==> ஏ
u22_008t08.tiff ==> ஐ
u30_005t04.tiff ==> உ
u34_001t04.tiff ==> ஆ
u33_005t04.tiff ==> உ
u16_005t02.tiff ==> உ
u18_008t05.tiff ==> ஐ
u33_005t10.tiff ==> உ
u26_010t02.tiff ==> ஓ
u19_010t08.tiff ==> ஓ
u33_010t08.tiff ==> ஓ
u22_007t04.tiff ==> ஏ
u18_010t08.tiff ==> ஓ
u16_005t03.tiff ==> உ
```

In [105]:
```
scores = model.evaluate(test_gen)
```

```
120/120 [==============================] - 3s 22ms/step - loss: 0.1903 - accuracy: 0.9550
```

In [ ]: