devnagari-handwritten-chars-classification-cnn

November 15, 2021

1 Handwritten Devnagari Character classification

1.0.1 Author: Sanjay Jaras

1.0.2 Bellevue University

1.1 Import Libraries

[ ]:

```
#!pip install git+https://github.com/tensorflow/examples.git
import os
import tensorflow as tf
from tensorflow.keras.layers.experimental import preprocessing
from IPython.display import clear_output
import matplotlib.pyplot as plt
import PIL
from PIL import Image
import numpy as np
from tqdm import tqdm
import random
from keras.preprocessing.image import ImageDataGenerator
```

1.2 Define datasource paths

[ ]:

```
base_path = "DevanagariHandwrittenCharacterDataset"
#base_path = "../input/devnagrihandwrittenchars/
 ↪DevanagariHandwrittenCharacterDataset"
train_path = os.path.join(base_path, "Train")
test_path = os.path.join(base_path, "Test")
```

1.3 Function to scan the folders and load images in array.

[ ]:

```
def load_image_to_array(file_path):
    with open(file_path, "rb") as f:
        img = PIL.Image.open(f)
        nparr = np.asarray(img)
        # plt.imshow(nparr)
        nparr = nparr[:, :, np.newaxis]
        return nparr
```

```python
def read_data_from_folder(folder_path, read_first_record_only=False):
    imgs = []
    labels = []
    for folder in tqdm(os.listdir(folder_path)):
        sub_folder = os.path.join(folder_path, folder)
        for f in os.listdir(sub_folder):
            img = load_image_to_array(os.path.join(sub_folder, f))
            imgs.append(img)
            labels.append(folder)
            if read_first_record_only:
                break
    return np.asarray(imgs), np.asarray(labels)
```

1.4 Sample images from all source folders

[ ]:

```python
sample_imgs, sample_labels = read_data_from_folder(train_path, True)
sample_imgs.shape
```

```
100%|      | 46/46 [00:00<00:00, 223.85it/s]
```

[ ]:

```
(46, 32, 32, 1)
```

1.5 Function to Display Images

[ ]:

```python
def display_image(imgarr):
    plt.figure(figsize=(20, 40))
    for i in range(len(imgarr)):
        plt.subplot(46, 6, i+1)
        img = tf.image.resize(imgarr[i], [100, 100])
        plt.imshow(img)
        plt.axis('off')
    plt.show()
```

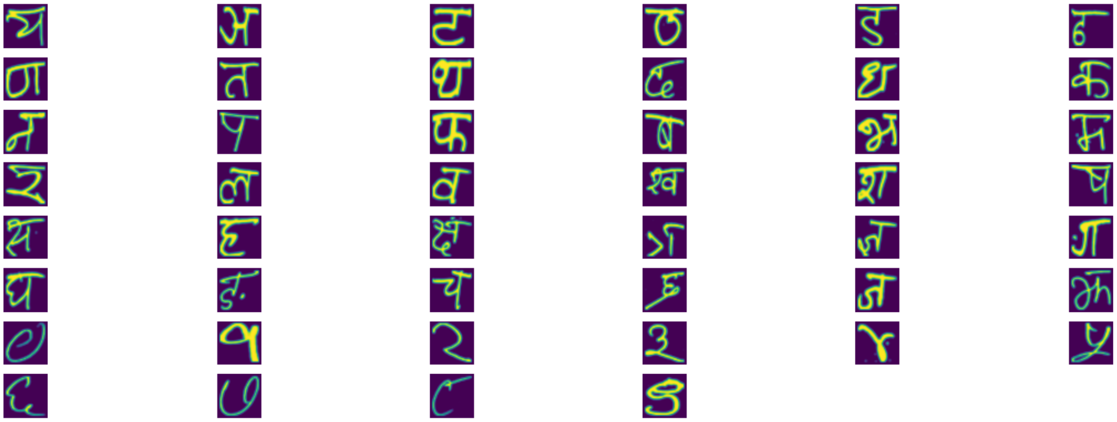
1.6 Show one sample image from each of input training folder

[ ]:

```python
display_image(sample_imgs)
```

```
2021-11-02 20:54:30.953964: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2021-11-02 20:54:30.958900: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2021-11-02 20:54:30.959079: I
```

```
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2021-11-02 20:54:30.959416: I tensorflow/core/platform/cpu_feature_guard.cc:142]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2021-11-02 20:54:30.959986: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2021-11-02 20:54:30.960143: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2021-11-02 20:54:30.960280: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2021-11-02 20:54:31.292360: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2021-11-02 20:54:31.292780: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2021-11-02 20:54:31.293140: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2021-11-02 20:54:31.293471: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 10080 MB memory:  -> device:
0, name: NVIDIA GeForce GTX 1080 Ti, pci bus id: 0000:2d:00.0, compute
capability: 6.1
```



1.7 Load Training and Test Dataset

```
[ ]: print("Loading training data....")
     train_data_img, train_data_labels = read_data_from_folder(train_path)
     print("Loading test data....")
     test_data_imgs, test_data_labels = read_data_from_folder(test_path)
```

```
Loading training data…

100%|      | 46/46 [00:11<00:00,  4.15it/s]

Loading test data…

100%|      | 46/46 [00:01<00:00, 26.13it/s]
```

1.8 Display Dataset shapes

```
[ ]: print("Training data imgs shape", train_data_img.shape)
     print("Training data labels shape", train_data_labels.shape)
     print("Test data imgs shape", test_data_imgs.shape)
     print("Test data labels shape", test_data_labels.shape)
```

```
Training data imgs shape (78200, 32, 32, 1)
Training data labels shape (78200,)
Test data imgs shape (13800, 32, 32, 1)
Test data labels shape (13800,)
```

1.9 Show some sample images from training dataset

```
[ ]: def display_image(imgarr):
         plt.figure(figsize=(20, 20))
         for i in range(len(imgarr)):
             plt.subplot(1, len(imgarr), i+1)
             plt.imshow(imgarr[i])
```

```
        plt.axis('off')
    plt.show()


rand = [random.randrange(1, 78200) for i in range(1, 20)]
display_image(train_data_img[rand])
```

1.10 Add some augmented images in training set

```
[ ]: def augment_data(images, labels):
         imgs = []
         labs = []
         data_gen = ImageDataGenerator(
             rotation_range=10,
             width_shift_range=0.1,
             height_shift_range=0.1,
             shear_range=0.1,
             brightness_range=(0.3, 1.0),
             fill_mode="nearest",
         )

         # generate samples and plot
         for i in range(images.shape[0]):
             # generate batch of images
             it = data_gen.flow(images[i:i+1], batch_size=1)
             batch = it.next()
             # convert to unsigned integers for viewing
             image = batch[0].astype("uint8")
             imgs.append(image)
             labs.append(labels[i])

         return imgs, labs
```

```
[ ]: imgs, labels = augment_data(train_data_img[rand], train_data_labels[rand])
     display_image(imgs)
```

[ ]:

```
imgs, labels = augment_data(train_data_img, train_data_labels)
train_data_img = np.concatenate((train_data_img, imgs))
train_data_labels = np.concatenate((train_data_labels, labels))
```

[ ]:

```
print("Training dataset shape after augmentation:", train_data_img.shape)
print("Training dataset labels shape after augmentation:", train_data_labels.
 ↪shape)
```

```
Training dataset shape after augmentation: (156400, 32, 32, 1)
Training dataset labels shape after augmentation: (156400,)
```

[ ]:

```
TRAIN_LENGTH = train_data_img.shape[0]
```

1.11 Define vocabulary for labels to convert label strings to int

[ ]:

```
vocab = np.unique(train_data_labels)

label_to_int = tf.keras.layers.StringLookup(vocabulary=vocab, invert=False)
train_data_labels = label_to_int(train_data_labels)
test_data_labels = label_to_int(test_data_labels)
```

1.12 Load datasets into TensorSliceDataset

[ ]:

```
train_images_ds = tf.data.Dataset.from_tensor_slices(
    (train_data_img, train_data_labels))

test_val_images_ds = tf.data.Dataset.from_tensor_slices(
    (test_data_imgs, test_data_labels))

#val_images_ds = tf.data.Dataset.from_tensor_slices((val_images, val_masks))
```

1.13 Split Test dataset into Test and validation datasets

[ ]:

```
ds_size = 13800
ds = test_val_images_ds.shuffle(10000, seed=12)

test_size = int(0.5 * ds_size)
val_size = int(0.5 * ds_size)

test_images_ds = ds.take(test_size)
val_images_ds = ds.skip(test_size).take(val_size)
```

1.14 Define Batch size

[ ]:

```
BUFFER_SIZE = TRAIN_LENGTH
BATCH_SIZE = 32
input_shape = (32, 32)
```

1.15 Create Batches for all 3 dataset

[ ]:

```
train_batches = (
    train_images_ds
    .cache()
    .shuffle(BUFFER_SIZE)
    .batch(BATCH_SIZE)
    .repeat()
    # .map(Augment())
    .prefetch(buffer_size=tf.data.experimental.AUTOTUNE))
# tf.data.AUTOTUNE

test_batches = test_images_ds.batch(BATCH_SIZE)
val_batches = val_images_ds.batch(BATCH_SIZE)
```

1.16 Define CNN Model

[ ]:

```
OUTPUT_CLASSES = 47

model = tf.keras.models.Sequential([
    tf.keras.layers.Rescaling(1./255, input_shape=(32, 32, 1)),
    tf.keras.layers.Conv2D(16, 2, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(64, 4, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(OUTPUT_CLASSES)
])
```

1.17 Compile model

[ ]:

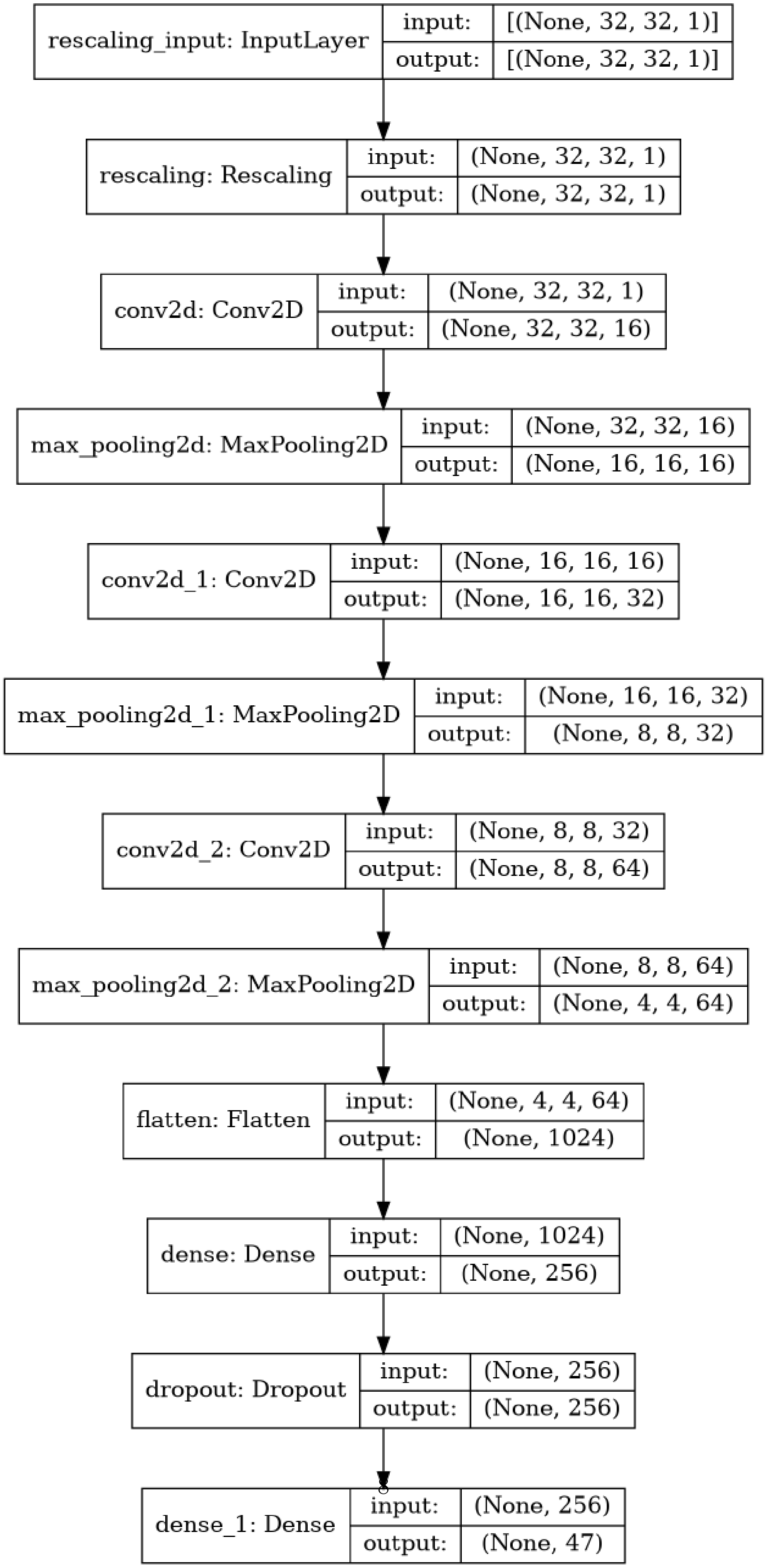
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(
                  from_logits=True),
              metrics=['accuracy'])
```

1.18 Show compiled model

[ ]:

```
tf.keras.utils.plot_model(model, show_shapes=True)
```

[ ]:


rescaling_input: InputLayer
input: [(None, 32, 32, 1)]
output: [(None, 32, 32, 1)]
rescaling: Rescaling
input: (None, 32, 32, 1)
output: (None, 32, 32, 1)
conv2d: Conv2D
input: (None, 32, 32, 1)
output: (None, 32, 32, 16)
max_pooling2d: MaxPooling2D
input: (None, 32, 32, 16)
output: (None, 16, 16, 16)
conv2d_1: Conv2D
input: (None, 16, 16, 16)
output: (None, 16, 16, 32)
max_pooling2d_1: MaxPooling2D
input: (None, 16, 16, 32)
output: (None, 8, 8, 32)
conv2d_2: Conv2D
input: (None, 8, 8, 32)
output: (None, 8, 8, 64)
max_pooling2d_2: MaxPooling2D
input: (None, 8, 8, 64)
output: (None, 4, 4, 64)
flatten: Flatten
input: (None, 4, 4, 64)
output: (None, 1024)
dense: Dense
input: (None, 1024)
output: (None, 256)
dropout: Dropout
input: (None, 256)
output: (None, 256)
dense_1: Dense
input: (None, 256)
output: (None, 47)

1.19 Callback functions for early stopping and Displaying information

[ ]:

```
int_to_label = tf.keras.layers.StringLookup(vocabulary=vocab, invert=True)


def show_images_predictions(imgs, pred):
    plt.figure(figsize=(15, 40))
    for i in range(len(imgs)):
        plt.subplot(32, 2, i+1)
        plt.imshow(imgs[i])
        lab = int_to_label([np.argmax(pred[i])]).numpy()[0]
        conf = np.max(tf.nn.softmax(pred[i])) * 100
        plt.title("Label:{} with confidence:{:.2f}%".format(lab, conf))
        plt.axis('off')
    plt.show()


def show_predictions(dataset=None, num=1, rec=BATCH_SIZE):

    for image_batch, label_batch in dataset.take(num):
        pred_batch = model.predict(image_batch[:rec])
        show_images_predictions(image_batch[:rec], pred_batch)
        # print(np.argmax(pred_batch[0]))
```

[ ]:

```
earlyStopCallback = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss', patience=5, min_delta=0.0001, restore_best_weights=True)


for image_batch, label_batch in val_batches.take(1):
    sample_images = image_batch[:2]


class DisplayCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        # clear_output(wait=True)
        print('\nSample Prediction after epoch {}\n'.format(epoch+1))
        pred_batch = model.predict(sample_images)
        show_images_predictions(sample_images, pred_batch)
        # for key in logs.keys():
        #    print("epoch {}, the {} is {:7.2f}.".format(
        #     (epoch+1), key, logs[key]))
        # print(logs.keys())
```

1.20 Train the model

[ ]:
```
EPOCHS = 30
VAL_SUBSPLITS = 5
VAL_LENGTH = 6900
VALIDATION_STEPS = VAL_LENGTH//BATCH_SIZE//VAL_SUBSPLITS  # 10
STEPS_PER_EPOCH = TRAIN_LENGTH // BATCH_SIZE
model_history = model.fit(train_batches, epochs=EPOCHS,
                          steps_per_epoch=STEPS_PER_EPOCH,
                          validation_steps=VALIDATION_STEPS,
                          validation_data=val_batches,
                          callbacks=[DisplayCallback(), earlyStopCallback])
#
```

```
Epoch 1/30
```

```
2021-11-02 20:55:10.109853: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR
Optimization Passes are enabled (registered 2)
2021-11-02 20:55:10.579195: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369]
Loaded cuDNN version 8204
```

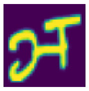
```
4887/4887 [==============================] - 14s 3ms/step - loss: 0.6688 -
accuracy: 0.8071 - val_loss: 0.1024 - val_accuracy: 0.9622

Sample Prediction after epoch 1
```

Label:b'character_10_yna' with confidence:99.99%

Label:b'character_4_gha' with confidence:99.54%

```
Epoch 2/30
4887/4887 [==============================] - 12s 2ms/step - loss: 0.2274 -
accuracy: 0.9295 - val_loss: 0.0719 - val_accuracy: 0.9775

Sample Prediction after epoch 2
```

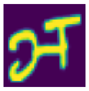
Label:b'character_10_yna' with confidence:100.00%



Label:b'character_4_gha' with confidence:100.00%

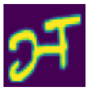
```
Epoch 3/30
```

```
4887/4887 [==============================] - 12s 2ms/step - loss: 0.1635 -
accuracy: 0.9496 - val_loss: 0.0509 - val_accuracy: 0.9847

Sample Prediction after epoch 3
```


Label:b'character_10_yna' with confidence:100.00%

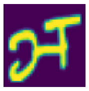
Label:b'character_4_gha' with confidence:100.00%

```
Epoch 4/30
4887/4887 [==============================] - 12s 2ms/step - loss: 0.1288 -
accuracy: 0.9603 - val_loss: 0.0586 - val_accuracy: 0.9840

Sample Prediction after epoch 4
```


Label:b'character_10_yna' with confidence:100.00%

Label:b'character_4_gha' with confidence:100.00%

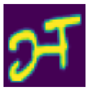
```
Epoch 5/30
4887/4887 [==============================] - 12s 2ms/step - loss: 0.1091 -
accuracy: 0.9657 - val_loss: 0.0492 - val_accuracy: 0.9855

Sample Prediction after epoch 5
```


Label:b'character_10_yna' with confidence:100.00%

Label:b'character_4_gha' with confidence:100.00%

```
Epoch 6/30
4887/4887 [==============================] - 12s 2ms/step - loss: 0.0928 -
accuracy: 0.9715 - val_loss: 0.0587 - val_accuracy: 0.9847

Sample Prediction after epoch 6
```

Label:b'character_10_yna' with confidence:100.00%



Label:b'character_4_gha' with confidence:100.00%

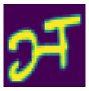
```
Epoch 7/30
4887/4887 [==============================] - 12s 2ms/step - loss: 0.0813 -
accuracy: 0.9747 - val_loss: 0.0562 - val_accuracy: 0.9869

Sample Prediction after epoch 7
```

Label:b'character_10_yna' with confidence:100.00%

Label:b'character_4_gha' with confidence:100.00%

```
Epoch 8/30
4887/4887 [==============================] - 12s 2ms/step - loss: 0.0750 -
accuracy: 0.9762 - val_loss: 0.0670 - val_accuracy: 0.9869

Sample Prediction after epoch 8
```

Label:b'character_10_yna' with confidence:100.00%



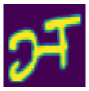
Label:b'character_4_gha' with confidence:99.99%

```
Epoch 9/30
4887/4887 [==============================] - 12s 2ms/step - loss: 0.0691 -
accuracy: 0.9779 - val_loss: 0.0480 - val_accuracy: 0.9862

Sample Prediction after epoch 9
```

Label:b'character_10_yna' with confidence:100.00%

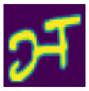
Label:b'character_4_gha' with confidence:100.00%

```
Epoch 10/30
4887/4887 [==============================] - 12s 2ms/step - loss: 0.0637 -
accuracy: 0.9801 - val_loss: 0.0974 - val_accuracy: 0.9869

Sample Prediction after epoch 10
```


Label:b'character_10_yna' with confidence:100.00%

Label:b'character_4_gha' with confidence:100.00%

```
Epoch 11/30
4887/4887 [==============================] - 12s 2ms/step - loss: 0.0608 -
accuracy: 0.9813 - val_loss: 0.0601 - val_accuracy: 0.9876

Sample Prediction after epoch 11
```


Label:b'character_10_yna' with confidence:100.00%

Label:b'character_4_gha' with confidence:100.00%

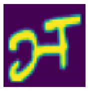
```
Epoch 12/30
4887/4887 [==============================] - 12s 2ms/step - loss: 0.0570 -
accuracy: 0.9823 - val_loss: 0.0345 - val_accuracy: 0.9920

Sample Prediction after epoch 12
```

Label:b'character_10_yna' with confidence:100.00%

Label:b'character_4_gha' with confidence:100.00%

```
Epoch 13/30
4887/4887 [==============================] - 12s 2ms/step - loss: 0.0541 -
accuracy: 0.9830 - val_loss: 0.0547 - val_accuracy: 0.9869

Sample Prediction after epoch 13
```

Label:b'character_10_yna' with confidence:100.00%



Label:b'character_4_gha' with confidence:100.00%

```
Epoch 14/30
4887/4887 [==============================] - 12s 2ms/step - loss: 0.0522 -
accuracy: 0.9839 - val_loss: 0.0657 - val_accuracy: 0.9876

Sample Prediction after epoch 14
```

Label:b'character_10_yna' with confidence:100.00%

Label:b'character_4_gha' with confidence:100.00%

```
Epoch 15/30
4887/4887 [==============================] - 12s 2ms/step - loss: 0.0511 -
accuracy: 0.9844 - val_loss: 0.0773 - val_accuracy: 0.9869

Sample Prediction after epoch 15
```

Label:b'character_10_yna' with confidence:100.00%

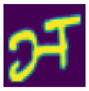
Label:b'character_4_gha' with confidence:100.00%

```
Epoch 16/30
4887/4887 [==============================] - 12s 2ms/step - loss: 0.0481 -
accuracy: 0.9850 - val_loss: 0.0209 - val_accuracy: 0.9949

Sample Prediction after epoch 16
```

Label:b'character_10_yna' with confidence:100.00%

Label:b'character_4_gha' with confidence:100.00%

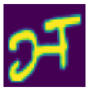
```
Epoch 17/30
4887/4887 [==============================] - 12s 2ms/step - loss: 0.0468 -
accuracy: 0.9855 - val_loss: 0.0878 - val_accuracy: 0.9876

Sample Prediction after epoch 17
```



Label:b'character_10_yna' with confidence:100.00%

Label:b'character_4_gha' with confidence:100.00%

```
Epoch 18/30
4887/4887 [==============================] - 12s 2ms/step - loss: 0.0443 -
accuracy: 0.9863 - val_loss: 0.0603 - val_accuracy: 0.9869

Sample Prediction after epoch 18
```



Label:b'character_10_yna' with confidence:100.00%

Label:b'character_4_gha' with confidence:100.00%

```
Epoch 19/30
4887/4887 [==============================] - 12s 3ms/step - loss: 0.0449 -
accuracy: 0.9864 - val_loss: 0.0574 - val_accuracy: 0.9913

Sample Prediction after epoch 19
```

Label:b'character_10_yna' with confidence:100.00%

Label:b'character_4_gha' with confidence:100.00%

```
Epoch 20/30
4887/4887 [==============================] - 12s 2ms/step - loss: 0.0429 -
accuracy: 0.9867 - val_loss: 0.0646 - val_accuracy: 0.9891

Sample Prediction after epoch 20
```

Label:b'character_10_yna' with confidence:100.00%
Label:b'character_4_gha' with confidence:100.00%

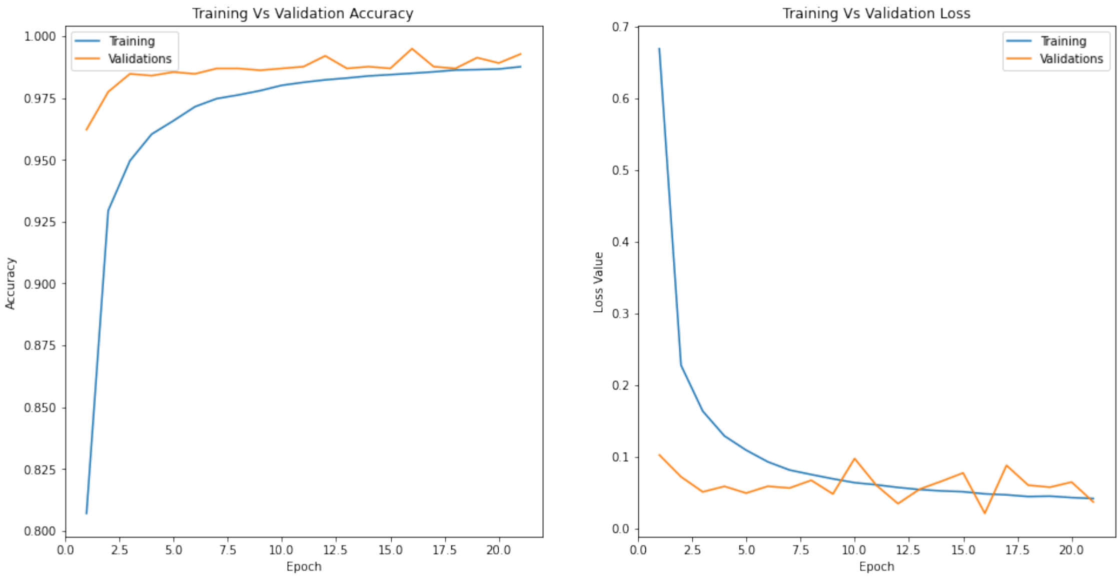
```
Epoch 21/30
4887/4887 [==============================] - 12s 2ms/step - loss: 0.0416 -
accuracy: 0.9876 - val_loss: 0.0369 - val_accuracy: 0.9927

Sample Prediction after epoch 21
```

Label:b'character_10_yna' with confidence:100.00%
Label:b'character_4_gha' with confidence:100.00%

1.21 Plot accuracy and loss for training and validations

[ ]:

```
length = len(model_history.history["accuracy"])+1

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(16, 8))
titles = ['Training Vs Validation Accuracy', 'Training Vs Validation Loss']
ax[0].set_title(titles[0])
ax[0].plot(range(1, length), model_history.history["accuracy"])
ax[0].plot(range(1, length), model_history.history["val_accuracy"])
ax[0].set_xlabel('Epoch')
ax[0].set_ylabel('Accuracy')
ax[0].legend(["Training", "Validations"])


ax[1].set_title(titles[1])
ax[1].plot(range(1, length), model_history.history["loss"])
ax[1].plot(range(1, length), model_history.history["val_loss"])
ax[1].set_xlabel('Epoch')
ax[1].set_ylabel('Loss Value')
ax[1].legend(["Training", "Validations"])
plt.show()
```



1.22 Evalute model against test dataset

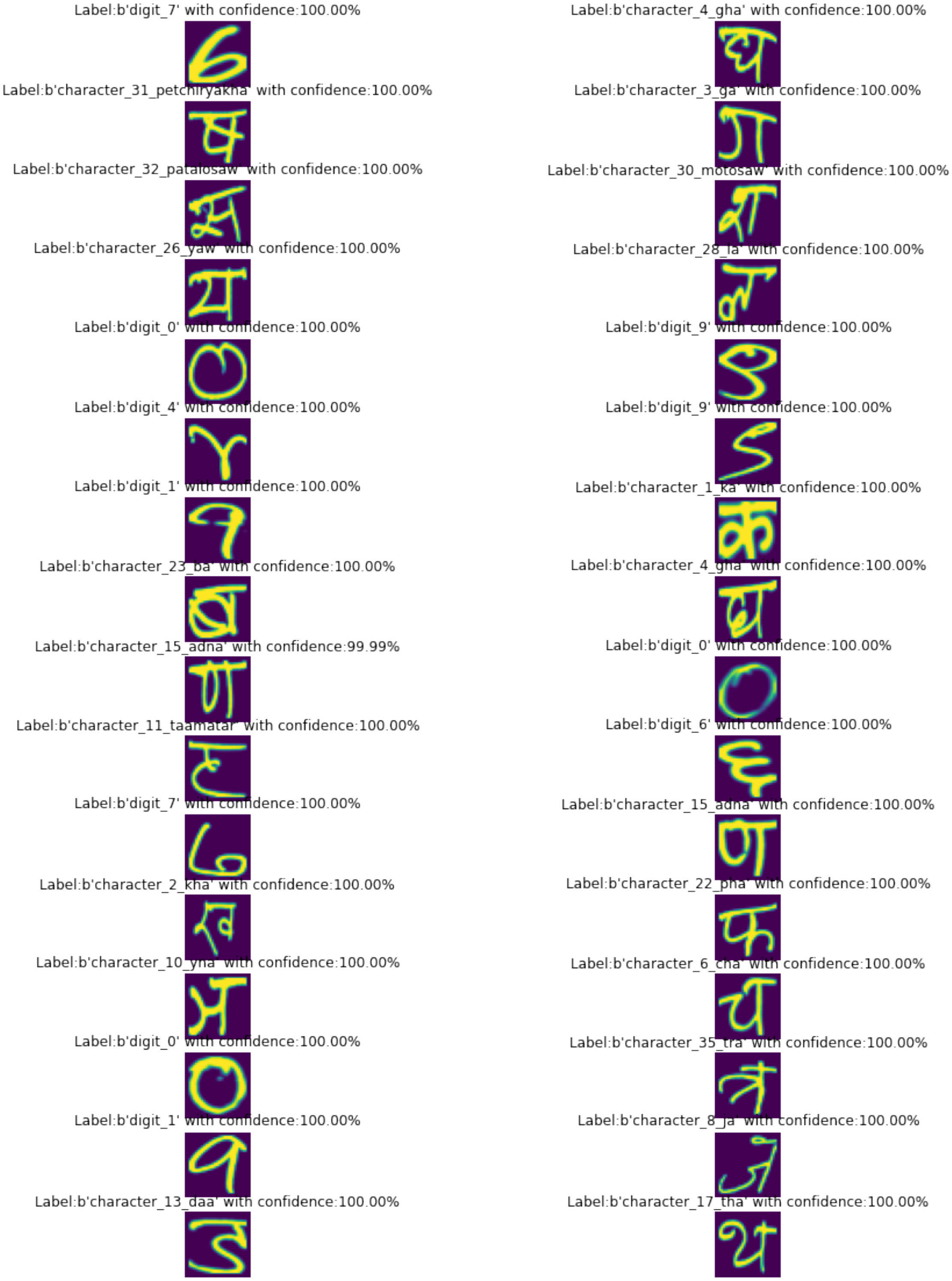
```
[ ]: model.evaluate(test_batches)
```

```
216/216 [==============================] - 0s 1ms/step - loss: 0.0762 -
accuracy: 0.9883
```

```
[ ]: [0.07615387439727783, 0.9882608652114868]
```

1.23 Sample predictions from validation dataset

```
[ ]: show_predictions(val_batches.shuffle(buffer_size=64), num=1)
```



Label:b'digit_7' with confidence:100.00%

Label:b'character_31_petchiryakha' with confidence:100.00%

Label:b'character_32_patalosaw' with confidence:100.00%

Label:b'character_26_yaw' with confidence:100.00%

Label:b'digit_0' with confidence:100.00%

Label:b'digit_4' with confidence:100.00%

Label:b'digit_1' with confidence:100.00%

Label:b'character_23_ba' with confidence:100.00%

Label:b'character_15_adna' with confidence:99.99%

Label:b'character_11_taamatar' with confidence:100.00%

Label:b'digit_7' with confidence:100.00%

Label:b'character_2_kha' with confidence:100.00%

Label:b'character_10_yna' with confidence:100.00%

Label:b'digit_0' with confidence:100.00%

Label:b'digit_1' with confidence:100.00%

Label:b'character_13_daa' with confidence:100.00%

Label:b'character_4_gha' with confidence:100.00%

Label:b'character_3_ga' with confidence:100.00%

Label:b'character_30_motosaw' with confidence:100.00%

Label:b'character_28_la' with confidence:100.00%

Label:b'digit_9' with confidence:100.00%

Label:b'digit_9' with confidence:100.00%

Label:b'character_1_ka' with confidence:100.00%

Label:b'character_4_gha' with confidence:100.00%

Label:b'digit_0' with confidence:100.00%

Label:b'digit_6' with confidence:100.00%

Label:b'character_15_adna' with confidence:100.00%

Label:b'character_22_pha' with confidence:100.00%

Label:b'character_6_cha' with confidence:100.00%

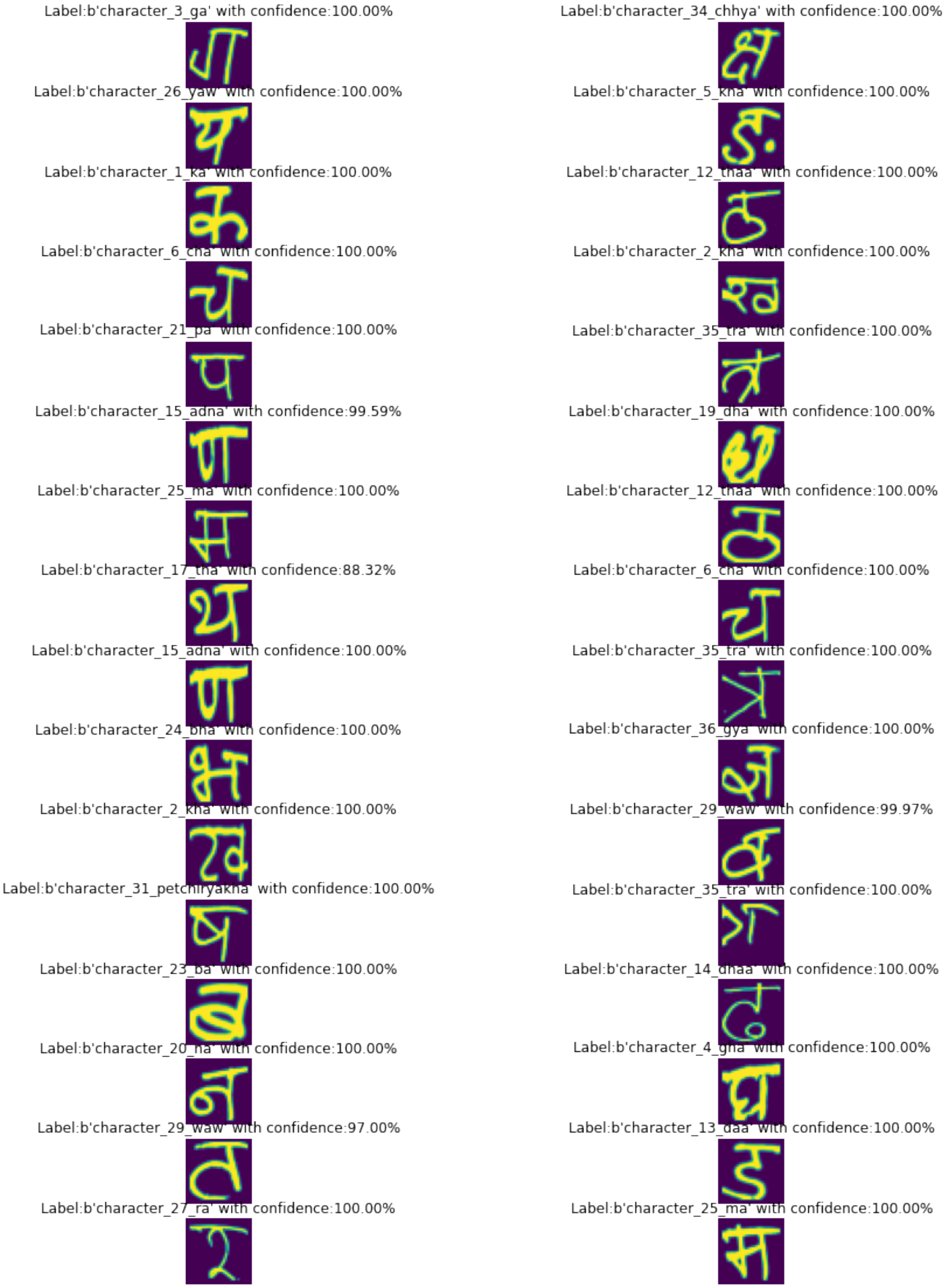
Label:b'character_35_tra' with confidence:100.00%

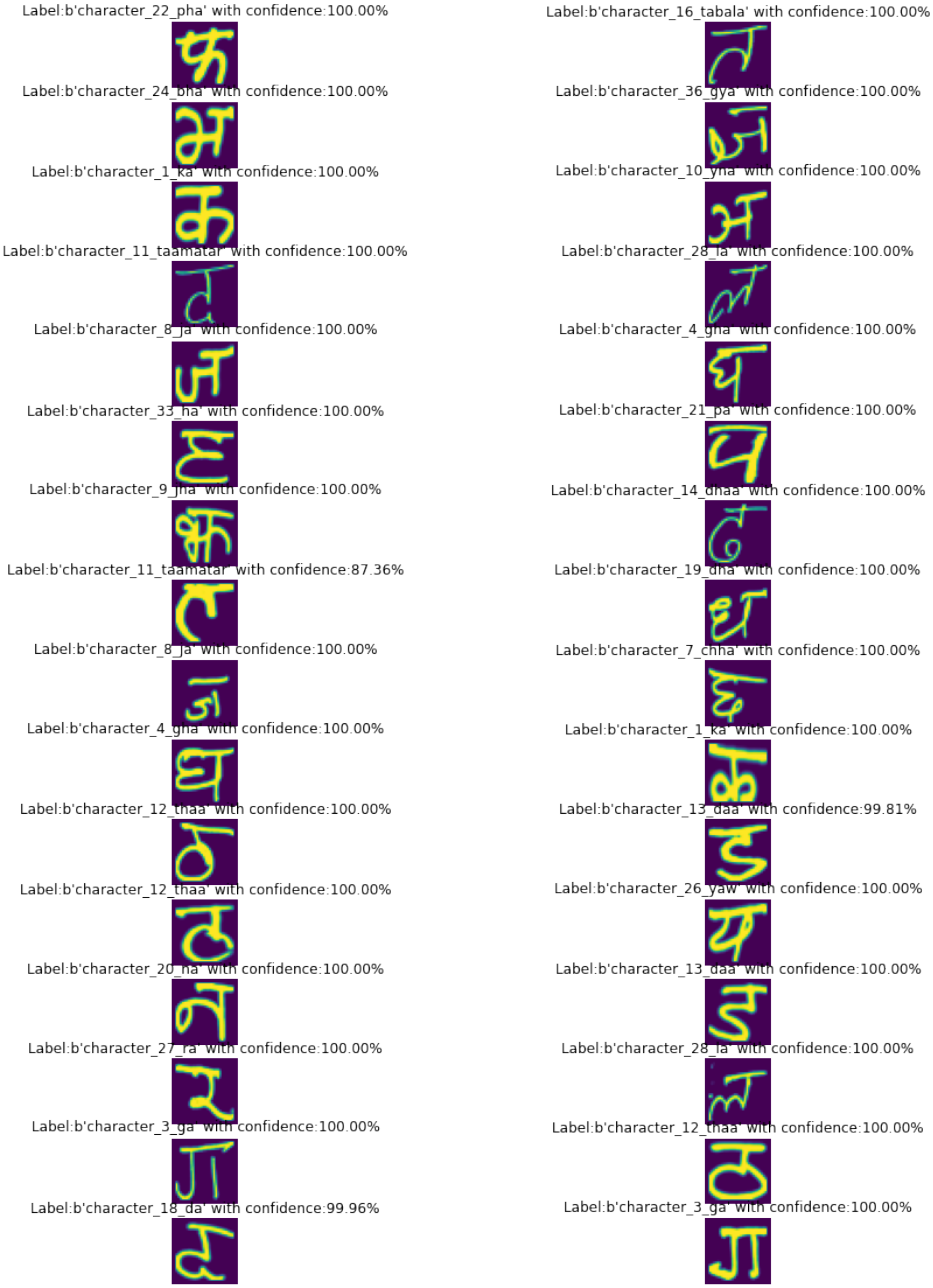
Label:b'character_8_ja' with confidence:100.00%

Label:b'character_17_tha' with confidence:100.00%

1.24 Sample Predictions from Test dataset

```
[ ]: show_predictions(test_batches.shuffle(buffer_size=64), num=2)
```




Label:b'character_22_pha' with confidence:100.00%
Label:b'character_16_tabala' with confidence:100.00%
Label:b'character_24_bha' with confidence:100.00%
Label:b'character_36_gya' with confidence:100.00%
Label:b'character_1_ka' with confidence:100.00%
Label:b'character_10_yna' with confidence:100.00%
Label:b'character_11_taamatar' with confidence:100.00%
Label:b'character_28_la' with confidence:100.00%
Label:b'character_8_ja' with confidence:100.00%
Label:b'character_4_gha' with confidence:100.00%
Label:b'character_33_ha' with confidence:100.00%
Label:b'character_21_pa' with confidence:100.00%
Label:b'character_9_jha' with confidence:100.00%
Label:b'character_14_dhaa' with confidence:100.00%
Label:b'character_11_taamatar' with confidence:87.36%
Label:b'character_19_dha' with confidence:100.00%
Label:b'character_8_ja' with confidence:100.00%
Label:b'character_7_chha' with confidence:100.00%
Label:b'character_4_gha' with confidence:100.00%
Label:b'character_1_ka' with confidence:100.00%
Label:b'character_12_thaa' with confidence:100.00%
Label:b'character_13_daa' with confidence:99.81%
Label:b'character_12_thaa' with confidence:100.00%
Label:b'character_26_yaw' with confidence:100.00%
Label:b'character_20_na' with confidence:100.00%
Label:b'character_13_daa' with confidence:100.00%
Label:b'character_27_ra' with confidence:100.00%
Label:b'character_28_la' with confidence:100.00%
Label:b'character_3_ga' with confidence:100.00%
Label:b'character_12_thaa' with confidence:100.00%
Label:b'character_18_da' with confidence:99.96%
Label:b'character_3_ga' with confidence:100.00%