DSC680-Handwritten Character Recognition – Devnagari -cnn

February 23, 2025

1 DSC680-Devnagari-Handwritten-Chars-Classification-CNN

1.0.1 Author: Sheetal Munjewar

1.0.2 Course: DSC 680 - Appliend Data Science

1.0.3 Bellevue University

1.0.4 Introduction -

This handwritten Devanagari character classification project by Sheetal Munjewar utilizes TensorFlow and Keras to build a model capable of recognizing handwritten characters. The dataset, sourced from the "DevanagariHandwrittenCharacterDataset," contains training and testing data that is loaded and preprocessed using Python libraries such as TensorFlow, Keras, and PIL. Data augmentation techniques, including rotations, shifts, and changes in brightness, are applied to enhance the dataset and improve model generalization. A convolutional neural network (CNN) model is built with multiple convolutional layers, max-pooling layers, and fully connected layers. The model is compiled using Adam optimizer and sparse categorical cross-entropy as the loss function. It is then trained using batches of images and labels, with an additional validation set created from the test dataset. The model's architecture and performance are visualized using `plot_model`. Data is preprocessed with TensorFlow's `Rescaling` and `StringLookup` layers, and the datasets are loaded into TensorFlow's Dataset API for efficient training. The project demonstrates a common pipeline for image classification tasks using deep learning techniques.

1.1 Import Libraries

[2]:
```python
# !pip install git+https://github.com/tensorflow/examples.git
# !pip install keras
import os
import tensorflow as tf
# from tensorflow.keras.layers.experimental import preprocessing
# from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Rescaling, Normalization
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 tensorflow.keras.preprocessing.image import ImageDataGenerator
# from keras.preprocessing.image import ImageDataGenerator
```

1.2 Specify data source paths.

[3]:
```
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 A function to traverse directories and load images into an array.

[4]:
```
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


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 Select images from each source folder.

[5]:
```
sample_imgs, sample_labels = read_data_from_folder(train_path, True)
sample_imgs.shape
```

```
100%|

        | 46/46 [00:00<00:00, 104.10it/s]
```

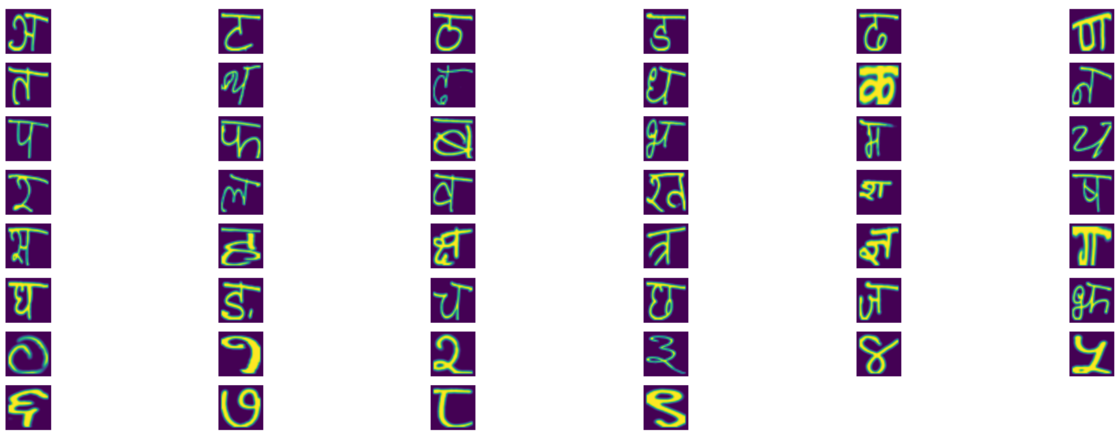
[5]:
```
(46, 32, 32, 1)
```

1.5 Function to showcase images.

[6]:
```
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 Display a sample image from each input training folder.

[7]:
```
display_image(sample_imgs)
```



1.7 Load the training and testing datasets.

[8]:
```
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 [00:35<00:00,  1.29it/s]

Loading test data…

100%|

        | 46/46 [00:06<00:00,  6.84it/s]
```

1.8 Show the shapes of the datasets.

[9]:
```
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 Display a few sample images from the training dataset.

[10]:
```
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 Include additional augmented images in the training set.

[11]:
```
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()
        batch = it.__next__()

        # convert to unsigned integers for viewing
        image = batch[0].astype("uint8")
        imgs.append(image)
        labs.append(labels[i])

    return imgs, labs
```

[12]:
```
imgs, labels = augment_data(train_data_img[rand], train_data_labels[rand])
display_image(imgs)
```

[13]:
```
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))
```

[14]:
```
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,)
```

[15]:
```
TRAIN_LENGTH = train_data_img.shape[0]
```

1.11 Create a vocabulary for labels to map label strings to integers.

[16]:
```
# 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)


import tensorflow as tf
import numpy as np

# Ensure labels are NumPy arrays and strings
train_data_labels = np.array(train_data_labels, dtype=str)
test_data_labels = np.array(test_data_labels, dtype=str)
```

```
# Create vocabulary and StringLookup layer
vocab = np.unique(train_data_labels)
label_to_int = tf.keras.layers.StringLookup(vocabulary=vocab, invert=False)

# Convert labels to integer representation
train_data_labels = label_to_int(tf.convert_to_tensor(train_data_labels))
test_data_labels = label_to_int(tf.convert_to_tensor(test_data_labels))
```

1.12 Load the datasets into a TensorSliceDataset.

[17]:
```
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 Divide the test dataset into test and validation datasets.

[18]:
```
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 Specify the batch size.

[19]:
```
BUFFER_SIZE = TRAIN_LENGTH
BATCH_SIZE = 32
input_shape = (32, 32)
```

1.15 Generate batches for all three datasets.

[20]:
```
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 the CNN model architecture.

[21]:
```
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)
])
```

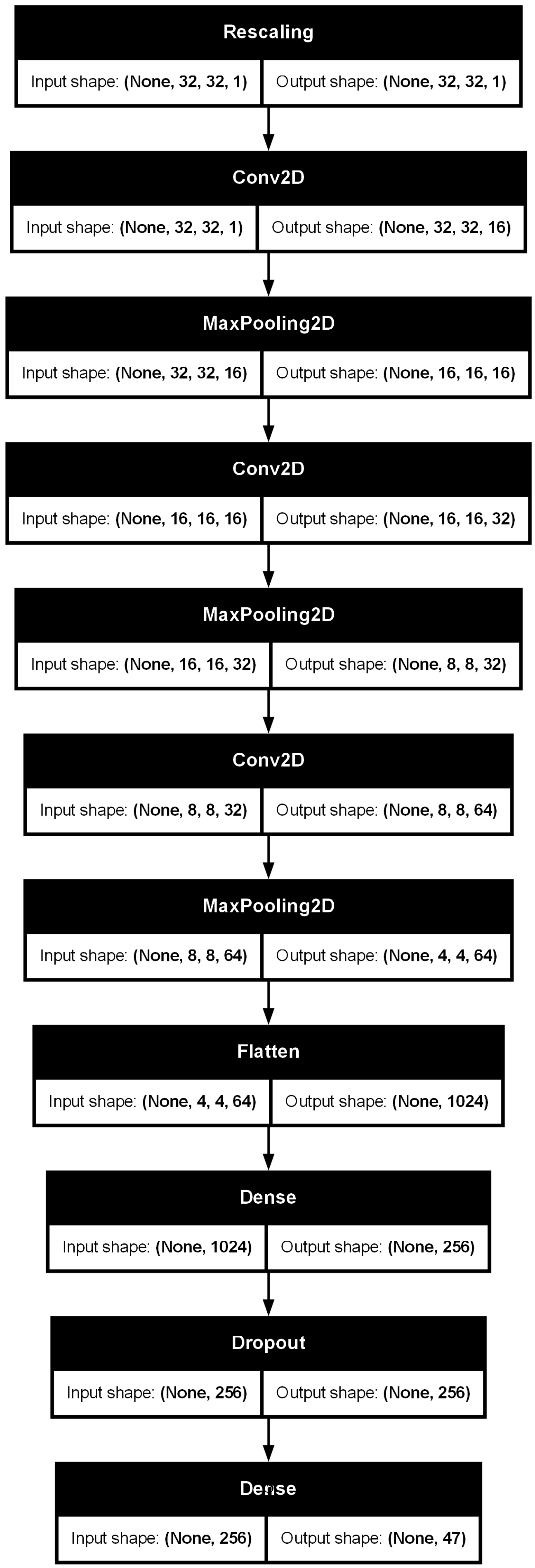
```
C:\Users\munje\anaconda3\lib\site-
packages\keras\src\layers\preprocessing\tf_data_layer.py:19: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(**kwargs)
```

1.17 Compile model

[22]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(
                  from_logits=True),
              metrics=['accuracy'])
```

[23]:
```
tf.keras.utils.plot_model(model, show_shapes=True)
```

[23]:


Rescaling
Input shape: (None, 32, 32, 1)
Output shape: (None, 32, 32, 1)
Conv2D
Input shape: (None, 32, 32, 1)
Output shape: (None, 32, 32, 16)
MaxPooling2D
Input shape: (None, 32, 32, 16)
Output shape: (None, 16, 16, 16)
Conv2D
Input shape: (None, 16, 16, 16)
Output shape: (None, 16, 16, 32)
MaxPooling2D
Input shape: (None, 16, 16, 32)
Output shape: (None, 8, 8, 32)
Conv2D
Input shape: (None, 8, 8, 32)
Output shape: (None, 8, 8, 64)
MaxPooling2D
Input shape: (None, 8, 8, 64)
Output shape: (None, 4, 4, 64)
Flatten
Input shape: (None, 4, 4, 64)
Output shape: (None, 1024)
Dense
Input shape: (None, 1024)
Output shape: (None, 256)
Dropout
Input shape: (None, 256)
Output shape: (None, 256)
Dense
Input shape: (None, 256)
Output shape: (None, 47)

1.18 Callback functions for early stopping and displaying information.

[24]:

```
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]))
```

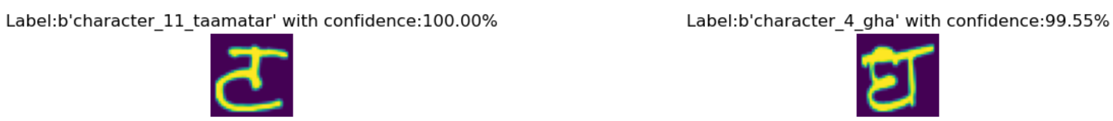
[25]:

```
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())
```

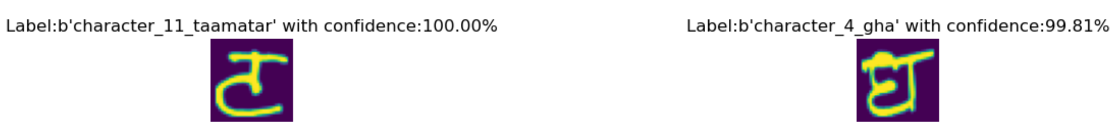
1.19 Train the model

[26]:
```
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
4883/4887            0s 8ms/step -
accuracy: 0.6653 - loss: 1.2184
Sample Prediction after epoch 1

1/1            0s 129ms/step
```




```
4887/4887            41s 8ms/step -
accuracy: 0.6654 - loss: 1.2178 - val_accuracy: 0.9680 - val_loss: 0.0977
Epoch 2/30
4883/4887            0s 8ms/step -
accuracy: 0.9291 - loss: 0.2314
Sample Prediction after epoch 2

1/1            0s 38ms/step
```




```
4887/4887            39s 8ms/step -
accuracy: 0.9291 - loss: 0.2314 - val_accuracy: 0.9789 - val_loss: 0.0668
Epoch 3/30
4883/4887            0s 8ms/step -
accuracy: 0.9511 - loss: 0.1609
Sample Prediction after epoch 3
```

```
1/1              0s 37ms/step
```

Label:b'character_11_taamatar' with confidence:100.00%

Label:b'character_4_gha' with confidence:99.89%

```
4887/4887             40s 8ms/step -
accuracy: 0.9511 - loss: 0.1609 - val_accuracy: 0.9862 - val_loss: 0.0342
Epoch 4/30
4886/4887             0s 8ms/step -
accuracy: 0.9618 - loss: 0.1204
Sample Prediction after epoch 4

1/1              0s 39ms/step
```

Label:b'character_11_taamatar' with confidence:99.99%

Label:b'character_4_gha' with confidence:97.21%

```
4887/4887             39s 8ms/step -
accuracy: 0.9618 - loss: 0.1204 - val_accuracy: 0.9811 - val_loss: 0.0792
Epoch 5/30
4884/4887             0s 8ms/step -
accuracy: 0.9675 - loss: 0.1018
Sample Prediction after epoch 5

1/1              0s 24ms/step
```

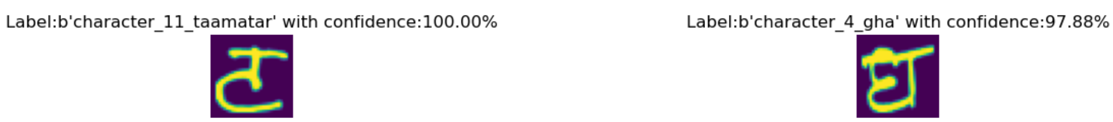
Label:b'character_11_taamatar' with confidence:100.00%

Label:b'character_4_gha' with confidence:99.99%

```
4887/4887             39s 8ms/step -
accuracy: 0.9675 - loss: 0.1018 - val_accuracy: 0.9855 - val_loss: 0.0470
Epoch 6/30
4887/4887             0s 8ms/step -
accuracy: 0.9728 - loss: 0.0859
Sample Prediction after epoch 6

1/1              0s 42ms/step
```


```
4887/4887              39s 8ms/step -
accuracy: 0.9728 - loss: 0.0859 - val_accuracy: 0.9898 - val_loss: 0.0362
Epoch 7/30
4883/4887              0s 8ms/step -
accuracy: 0.9758 - loss: 0.0760
Sample Prediction after epoch 7

1/1            0s 37ms/step
```


```
4887/4887              40s 8ms/step -
accuracy: 0.9758 - loss: 0.0760 - val_accuracy: 0.9876 - val_loss: 0.0535
Epoch 8/30
4885/4887              0s 8ms/step -
accuracy: 0.9783 - loss: 0.0690
Sample Prediction after epoch 8

1/1            0s 39ms/step
```

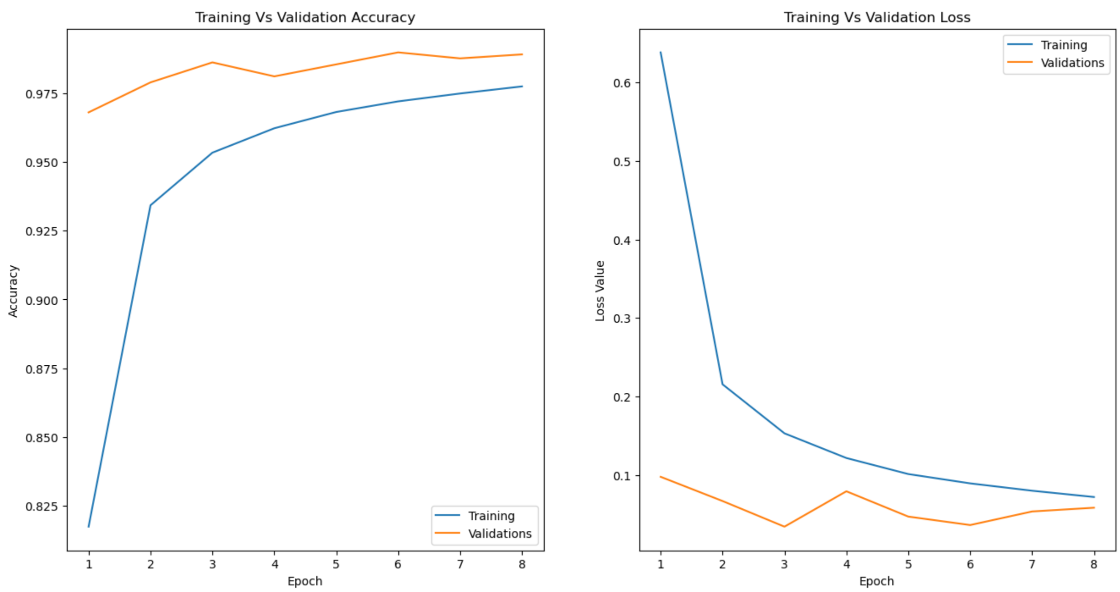



```
4887/4887              39s 8ms/step -
accuracy: 0.9783 - loss: 0.0690 - val_accuracy: 0.9891 - val_loss: 0.0583
```

1.20 Plot the accuracy and loss for both training and validation.

[27]:
```python
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()
```



1.21 Evaluate the model on the test dataset.

[28]:
```
model.evaluate(test_batches)
```

```
216/216                 1s 4ms/step -
accuracy: 0.9809 - loss: 0.0647
```

[28]: [0.06005474552512169, 0.9824637770652771]

1.22 Select sample predictions from the validation dataset.

[29]:
```
show_predictions(val_batches.shuffle(buffer_size=64), num=1)
```

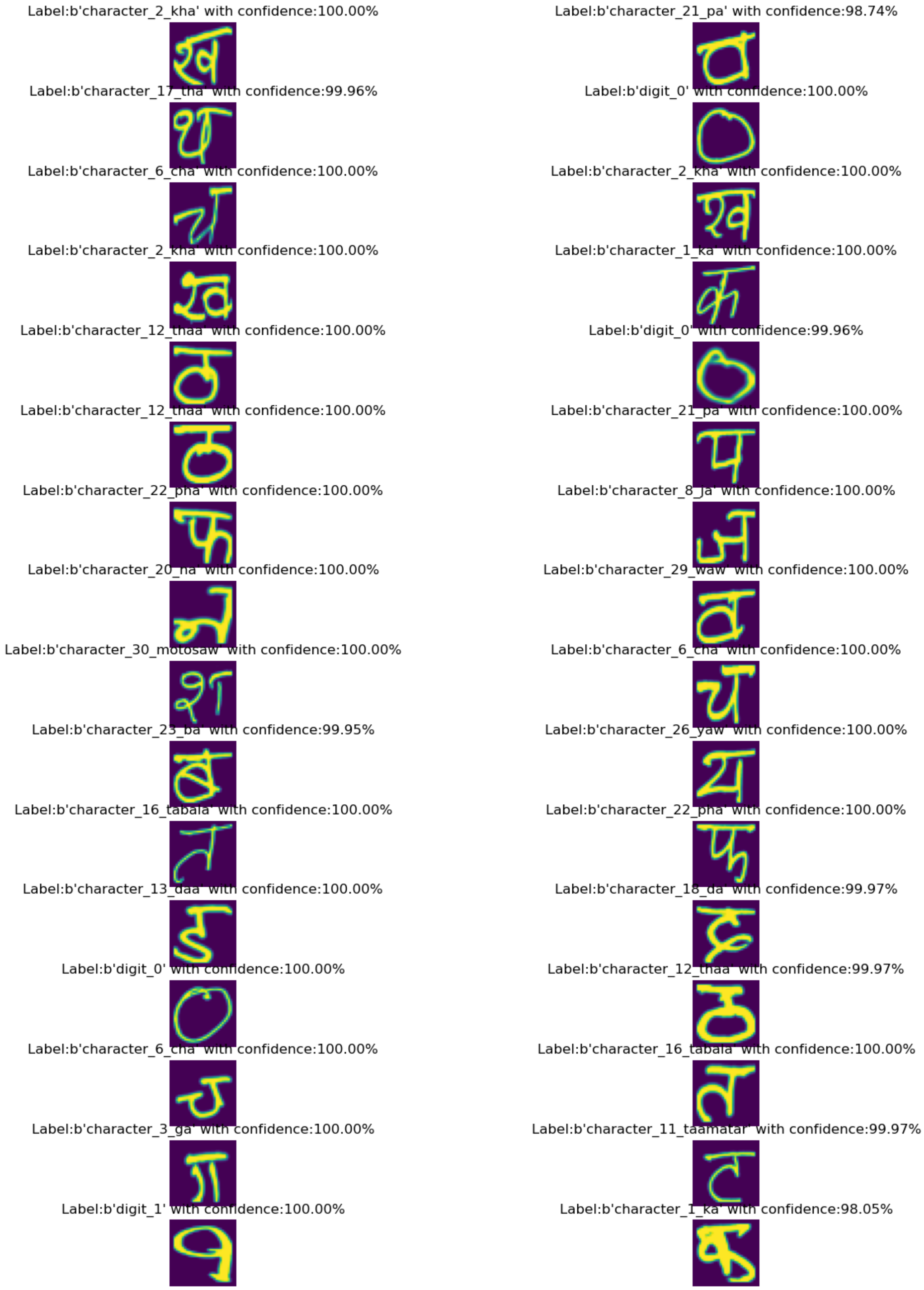
```
1/1              0s 93ms/step
```


Label:b'character_31_petchiryakha' with confidence:99.70%
Label:b'character_5_kna' with confidence:90.84%
Label:b'character_23_ba' with confidence:99.93%
Label:b'character_26_yaw' with confidence:99.98%
Label:b'digit_6' with confidence:99.99%
Label:b'character_9_jha' with confidence:99.38%
Label:b'character_6_cha' with confidence:100.00%
Label:b'character_36_gya' with confidence:100.00%
Label:b'digit_6' with confidence:100.00%
Label:b'digit_4' with confidence:100.00%
Label:b'character_5_kna' with confidence:99.99%
Label:b'digit_1' with confidence:100.00%
Label:b'digit_9' with confidence:100.00%
Label:b'character_13_daa' with confidence:99.93%
Label:b'character_34_chhya' with confidence:100.00%
Label:b'digit_9' with confidence:99.45%
Label:b'digit_6' with confidence:100.00%
Label:b'character_11_taamatar' with confidence:99.97%
Label:b'digit_9' with confidence:100.00%
Label:b'character_3_ga' with confidence:99.90%
Label:b'character_24_bha' with confidence:100.00%
Label:b'digit_1' with confidence:99.95%
Label:b'digit_3' with confidence:100.00%
Label:b'character_11_taamatar' with confidence:99.93%
Label:b'digit_8' with confidence:100.00%
Label:b'character_1_ka' with confidence:99.72%
Label:b'character_18_da' with confidence:100.00%
Label:b'character_33_ha' with confidence:99.99%
Label:b'character_9_jha' with confidence:100.00%
Label:b'digit_9' with confidence:100.00%
Label:b'character_32_patalosaw' with confidence:100.00%
Label:b'character_21_pa' with confidence:99.56%

1.23 Select sample predictions from the test dataset.

[30]:
```
show_predictions(test_batches.shuffle(buffer_size=64), num=2)
```

```
1/1              0s 48ms/step
```




```
1/1                 0s 19ms/step
```


Label:b'character_9_jha' with confidence:100.00%
Label:b'character_36_gya' with confidence:100.00%
Label:b'character_33_ha' with confidence:97.95%
Label:b'character_11_taamatar' with confidence:99.99%
Label:b'character_1_ka' with confidence:100.00%
Label:b'character_25_ma' 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_12_thaa' with confidence:99.98%
Label:b'character_36_gya' with confidence:100.00%
Label:b'character_35_tra' with confidence:99.97%
Label:b'character_32_patalosaw' with confidence:99.97%
Label:b'character_29_waw' with confidence:99.98%
Label:b'character_36_gya' with confidence:99.99%
Label:b'character_31_petchiryakha' with confidence:100.00%
Label:b'character_4_gha' with confidence:100.00%
Label:b'character_19_dha' with confidence:100.00%
Label:b'character_29_waw' with confidence:99.96%
Label:b'character_13_daa' with confidence:100.00%
Label:b'character_29_waw' with confidence:99.89%
Label:b'character_36_gya' with confidence:100.00%
Label:b'character_34_chhya' with confidence:99.99%
Label:b'character_35_tra' with confidence:100.00%
Label:b'character_3_ga' with confidence:100.00%
Label:b'character_18_da' with confidence:100.00%
Label:b'character_8_ja' with confidence:100.00%
Label:b'character_20_na' with confidence:100.00%
Label:b'character_15_adna' with confidence:99.45%
Label:b'character_24_bha' with confidence:100.00%
Label:b'digit_0' with confidence:100.00%
Label:b'character_33_ha' with confidence:100.00%
Label:b'character_23_ba' with confidence:84.05%

```
[ ]:
```