Import Libraries

```
import os
import numpy as np
import matplotlib.pyplot as plt

from pathlib import Path
from collections import Counter

import tensorflow as tf
from tensorflow import keras
from keras import layers
```

Loading Data

```
!curl -LO https://github.com/AakashKumarNain/CaptchaCracker/raw/m
!unzip -qq captcha_images_v2.zip
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 8863k  100 8863k    0     0  13.5M      0 --:--:-- --:--:-- --:--:-- 13.5M
```

```
# Path to the data directory
data_dir = Path("./captcha_images_v2/")

# Get list of all the images
images = sorted(list(map(str, list(data_dir.glob("*.png")))))
labels = [img.split(os.path.sep)[-1].split(".png")[0] for img in
characters = set(char for label in labels for char in label)
characters = sorted(list(characters))

print("Number of images found: ", len(images))
print("Number of labels found: ", len(labels))
print("Number of unique characters: ", len(characters))
```

```
print("Characters present: ", characters)

# Batch size for training and validation
batch_size = 16

# Desired image dimensions
img_width = 200
img_height = 50

# Factor by which the image is going to be downsampled
# by the convolutional blocks. We will be using two
# convolution blocks and each block will have
# a pooling layer which downsample the features by a factor of 2.
# Hence total downsampling factor would be 4.
downsample_factor = 4

# Maximum length of any captcha in the dataset
max_length = max([len(label) for label in labels])
```

```
Number of images found:  1040
Number of labels found:  1040
Number of unique characters:  19
Characters present:  ['2', '3', '4', '5', '6', '7', '8', 'b', 'c', 'd', 'e', 'f', 'g',
```

Preprocessing

```
# Mapping characters to integers
char_to_num = layers.StringLookup(
    vocabulary=list(characters), mask_token=None
)

# Mapping integers back to original characters
num_to_char = layers.StringLookup(
    vocabulary=char_to_num.get_vocabulary(), mask_token=None, inv
)
```

```
def split_data(images, labels, train_size=0.9, shuffle=True):
    # 1. Get the total size of the dataset
    size = len(images)
    # 2. Make an indices array and shuffle it, if required
    indices = np.arange(size)
    if shuffle:
        np.random.shuffle(indices)
    # 3. Get the size of training samples
    train_samples = int(size * train_size)
    # 4. Split data into training and validation sets
    x_train, y_train = images[indices[:train_samples]], labels[in
    x_valid, y_valid = images[indices[train_samples:]], labels[in
    return x_train, x_valid, y_train, y_valid


# Splitting data into training and validation sets
x_train, x_valid, y_train, y_valid = split_data(np.array(images),


def encode_single_sample(img_path, label):
    # 1. Read image
    img = tf.io.read_file(img_path)
    # 2. Decode and convert to grayscale
    img = tf.io.decode_png(img, channels=1)
    # 3. Convert to float32 in [0, 1] range
    img = tf.image.convert_image_dtype(img, tf.float32)
    # 4. Resize to the desired size
    img = tf.image.resize(img, [img_height, img_width])
    # 5. Transpose the image because we want the time
    # dimension to correspond to the width of the image.
    img = tf.transpose(img, perm=[1, 0, 2])
    # 6. Map the characters in label to numbers
    label = char_to_num(tf.strings.unicode_split(label, input_enc
    # 7. Return a dict as our model is expecting two inputs
    return {"image": img, "label": label}
```

Create Dataset Objects

```
train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_tr
train_dataset = (
    train_dataset.map(
        encode_single_sample, num_parallel_calls=tf.data.AUTOTUNE
    )
    .batch(batch_size)
    .prefetch(buffer_size=tf.data.AUTOTUNE)
)

validation_dataset = tf.data.Dataset.from_tensor_slices((x_valid,
validation_dataset = (
    validation_dataset.map(
        encode_single_sample, num_parallel_calls=tf.data.AUTOTUNE
    )
    .batch(batch_size)
    .prefetch(buffer_size=tf.data.AUTOTUNE)
)
```

Visualization

```
_, ax = plt.subplots(4, 4, figsize=(12, 8))
for batch in train_dataset.take(1):
    images = batch["image"]
    labels = batch["label"]
    for i in range(16):
        img = (images[i] * 255).numpy().astype("uint8")
        label = tf.strings.reduce_join(num_to_char(labels[i])).nu
        ax[i // 4, i % 4].imshow(img[:, :, 0].T, cmap="gray")
        ax[i // 4, i % 4].set_title(label)
        ax[i // 4, i % 4].axis("off")
plt.show()
```

Model

```
class CTCLayer(layers.Layer):
    def __init__(self, name=None):
        super().__init__(name=name)
        self.loss_fn = keras.backend.ctc_batch_cost

    def call(self, y_true, y_pred):
        # Compute the training-time loss value and add it
        # to the layer using `self.add_loss()`.
        batch_len = tf.cast(tf.shape(y_true)[0], dtype="int64")
        input_length = tf.cast(tf.shape(y_pred)[1], dtype="int64"
        label_length = tf.cast(tf.shape(y_true)[1], dtype="int64"

        input_length = input_length * tf.ones(shape=(batch_len, 1
        label_length = label_length * tf.ones(shape=(batch_len, 1

        loss = self.loss_fn(y_true, y_pred, input_length, label_l
        self.add_loss(loss)

        # At test time, just return the computed predictions
        return y_pred


def build_model():
```

```
    # Inputs to the model
    input_img = layers.Input(
        shape=(img_width, img_height, 1), name="image", dtype="fl
    )
    labels = layers.Input(name="label", shape=(None,), dtype="flo

    # First conv block
    x = layers.Conv2D(
        32,
        (3, 3),
        activation="relu",
        kernel_initializer="he_normal",
        padding="same",
        name="Conv1",
    )(input_img)
    x = layers.MaxPooling2D((2, 2), name="pool1")(x)

    # Second conv block
    x = layers.Conv2D(
        64,
        (3, 3),
        activation="relu",
        kernel_initializer="he_normal",
        padding="same",
        name="Conv2",
    )(x)
    x = layers.MaxPooling2D((2, 2), name="pool2")(x)

    # We have used two max pool with pool size and strides 2.
    # Hence, downsampled feature maps are 4x smaller. The number
    # filters in the last layer is 64. Reshape accordingly before
    # passing the output to the RNN part of the model
    new_shape = ((img_width // 4), (img_height // 4) * 64)
    x = layers.Reshape(target_shape=new_shape, name="reshape")(x)
    x = layers.Dense(64, activation="relu", name="dense1")(x)
    x = layers.Dropout(0.2)(x)

    # RNNs
    x = layers.Bidirectional(layers.LSTM(128, return_sequences=Tr
```

```
    x = layers.Bidirectional(layers.LSTM(64, return_sequences=Tru

    # Output layer
    x = layers.Dense(
        len(char_to_num.get_vocabulary()) + 1, activation="softma
    )(x)

    # Add CTC layer for calculating CTC loss at each step
    output = CTCLayer(name="ctc_loss")(labels, x)

    # Define the model
    model = keras.models.Model(
        inputs=[input_img, labels], outputs=output, name="ocr_mod
    )
    # Optimizer
    opt = keras.optimizers.Adam()
    # Compile the model and return
    model.compile(optimizer=opt)
    return model


# Get the model
model = build_model()
model.summary()
```

```
Model: "ocr_model_v1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 image (InputLayer)             [(None, 200, 50, 1)  0           []
                                ]

 Conv1 (Conv2D)                 (None, 200, 50, 32)  320         ['image[0][0]']

 pool1 (MaxPooling2D)           (None, 100, 25, 32)  0           ['Conv1[0][0]']

 Conv2 (Conv2D)                 (None, 100, 25, 64)  18496       ['pool1[0][0]']

 pool2 (MaxPooling2D)           (None, 50, 12, 64)   0           ['Conv2[0][0]']

 reshape (Reshape)              (None, 50, 768)      0           ['pool2[0][0]']

 dense1 (Dense)                 (None, 50, 64)       49216       ['reshape[0][0]']

 dropout_1 (Dropout)            (None, 50, 64)       0           ['dense1[0][0]']
```

```
 bidirectional_2 (Bidirectional  (None, 50, 256)     197632      ['dropout_1[0][0]']
 )

 bidirectional_3 (Bidirectional  (None, 50, 128)     164352      ['bidirectional_2[0][0
 )

 label (InputLayer)             [(None, None)]       0           []

 dense2 (Dense)                 (None, 50, 21)       2709        ['bidirectional_3[0][0

 ctc_loss (CTCLayer)            (None, 50, 21)       0           ['label[0][0]',
                                                                  'dense2[0][0]']

=========================================================================================
Total params: 432,725
Trainable params: 432,725
Non-trainable params: 0
_________________________________________________________________________________________
```

Training

```
epochs = 120
early_stopping_patience = 8
# Add early stopping
early_stopping = keras.callbacks.EarlyStopping(
    monitor="val_loss", patience=early_stopping_patience, restore
)

# Train the model
history = model.fit(
    train_dataset,
    validation_data=validation_dataset,
    epochs=epochs,
    callbacks=[early_stopping],
)
```

```
59/59 [==============================] - 18s 303ms/step - loss: 0.0170 - val_loss: 0.
Epoch 58/120
59/59 [==============================] - 19s 316ms/step - loss: 0.0176 - val_loss: 0.
Epoch 59/120
59/59 [==============================] - 18s 300ms/step - loss: 0.0157 - val_loss: 0.
Epoch 60/120
59/59 [==============================] - 17s 292ms/step - loss: 0.0132 - val_loss: 0.
Epoch 61/120
59/59 [==============================] - 17s 292ms/step - loss: 0.0396 - val_loss: 0.
```

```
Epoch 62/120
59/59 [==============================] - 17s 296ms/step - loss: 0.0239 - val_loss: 0.
Epoch 63/120
59/59 [==============================] - 17s 291ms/step - loss: 0.0116 - val_loss: 0.
Epoch 64/120
59/59 [==============================] - 17s 293ms/step - loss: 0.0338 - val_loss: 0.
Epoch 65/120
59/59 [==============================] - 17s 293ms/step - loss: 0.0440 - val_loss: 0.
Epoch 66/120
59/59 [==============================] - 19s 314ms/step - loss: 0.0308 - val_loss: 0.
Epoch 67/120
59/59 [==============================] - 17s 294ms/step - loss: 0.0360 - val_loss: 0.
Epoch 68/120
59/59 [==============================] - 17s 291ms/step - loss: 0.0246 - val_loss: 0.
Epoch 69/120
59/59 [==============================] - 17s 289ms/step - loss: 0.0290 - val_loss: 0.
Epoch 70/120
59/59 [==============================] - 17s 291ms/step - loss: 0.0348 - val_loss: 0.
Epoch 71/120
59/59 [==============================] - 17s 290ms/step - loss: 0.0883 - val_loss: 0.
Epoch 72/120
59/59 [==============================] - 17s 288ms/step - loss: 0.0758 - val_loss: 0.
Epoch 73/120
59/59 [==============================] - 17s 290ms/step - loss: 0.0207 - val_loss: 0.
Epoch 74/120
59/59 [==============================] - 18s 312ms/step - loss: 0.0141 - val_loss: 0.
Epoch 75/120
59/59 [==============================] - 17s 290ms/step - loss: 0.0258 - val_loss: 0.
Epoch 76/120
59/59 [==============================] - 17s 291ms/step - loss: 0.0257 - val_loss: 0.
Epoch 77/120
59/59 [==============================] - 17s 289ms/step - loss: 0.0463 - val_loss: 0.
Epoch 78/120
59/59 [==============================] - 17s 291ms/step - loss: 0.0564 - val_loss: 0.
Epoch 79/120
59/59 [==============================] - 17s 291ms/step - loss: 0.0225 - val_loss: 0.
Epoch 80/120
59/59 [==============================] - 17s 293ms/step - loss: 0.0608 - val_loss: 0.
Epoch 81/120
59/59 [==============================] - 17s 287ms/step - loss: 0.0897 - val_loss: 0.
Epoch 82/120
59/59 [==============================] - 18s 311ms/step - loss: 0.0291 - val_loss: 0.
Epoch 83/120
59/59 [==============================] - 17s 287ms/step - loss: 0.0267 - val_loss: 0.
Epoch 84/120
59/59 [==============================] - 17s 290ms/step - loss: 0.0240 - val_loss: 0.
Epoch 85/120
59/59 [==============================] - 17s 287ms/step - loss: 0.0215 - val_loss: 0.
```

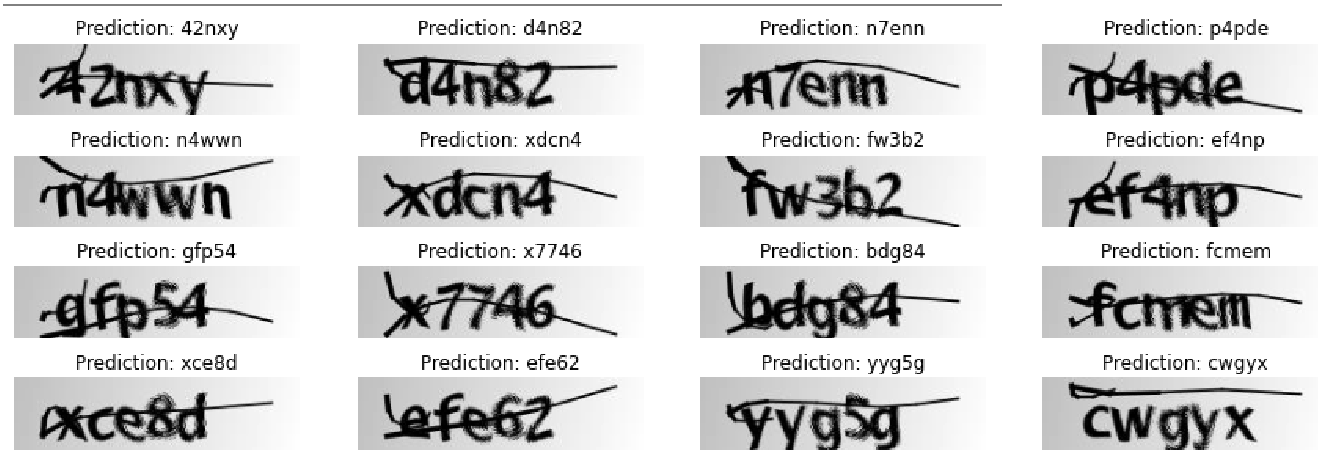
Inference

```
# Get the prediction model by extracting layers till the output l
```

```
prediction_model = keras.models.Model(
    model.get_layer(name="image").input, model.get_layer(name="de
)
prediction_model.summary()

# A utility function to decode the output of the network
def decode_batch_predictions(pred):
    input_len = np.ones(pred.shape[0]) * pred.shape[1]
    # Use greedy search. For complex tasks, you can use beam sear
    results = keras.backend.ctc_decode(pred, input_length=input_l
        :, :max_length
    ]
    # Iterate over the results and get back the text
    output_text = []
    for res in results:
        res = tf.strings.reduce_join(num_to_char(res)).numpy().de
        output_text.append(res)
    return output_text


#  Let's check results on some validation samples
for batch in validation_dataset.take(1):
    batch_images = batch["image"]
    batch_labels = batch["label"]

    preds = prediction_model.predict(batch_images)
    pred_texts = decode_batch_predictions(preds)

    orig_texts = []
    for label in batch_labels:
        label = tf.strings.reduce_join(num_to_char(label)).numpy(
        orig_texts.append(label)

    _, ax = plt.subplots(4, 4, figsize=(15, 5))
    for i in range(len(pred_texts)):
        img = (batch_images[i, :, :, 0] * 255).numpy().astype(np.
        img = img.T
        title = f"Prediction: {pred_texts[i]}"
        ax[i // 4, i % 4].imshow(img, cmap="gray")
```

```
        ax[i // 4, i % 4].set_title(title)
        ax[i // 4, i % 4].axis("off")
plt.show()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 image (InputLayer)          [(None, 200, 50, 1)]      0         
                                                                 
 Conv1 (Conv2D)              (None, 200, 50, 32)       320       
                                                                 
 pool1 (MaxPooling2D)        (None, 100, 25, 32)       0         
                                                                 
 Conv2 (Conv2D)              (None, 100, 25, 64)       18496     
                                                                 
 pool2 (MaxPooling2D)        (None, 50, 12, 64)        0         
                                                                 
 reshape (Reshape)           (None, 50, 768)           0         
                                                                 
 dense1 (Dense)              (None, 50, 64)            49216     
                                                                 
 dropout_1 (Dropout)         (None, 50, 64)            0         
                                                                 
 bidirectional_2 (Bidirectio  (None, 50, 256)          197632    
 nal)                                                            
                                                                 
 bidirectional_3 (Bidirectio  (None, 50, 128)          164352    
 nal)                                                            
                                                                 
 dense2 (Dense)              (None, 50, 21)            2709      
                                                                 
=================================================================
Total params: 432,725
Trainable params: 432,725
Non-trainable params: 0
_________________________________________________________________
```



3s completed at 14:49