Ministry of Education and Science of the Republic of Kazakhstan

Astana IT University

INDUSTRIAL INTERNSHIP REPORT

Practice base: Limited Liability Partnership "Bolashak Energysy"

Student: Assimov Ilkhamzhan

Educational program: Computer Science

Group: IT-2201

Head of practice from the company: Mayasarov A.A.

Head of practice from AITU: Muslim Sergaziyev

Astana, 2025

Table of Contents

1. Introduction 3
   1.1 Goals and Objectives of the Internship 4
   1.2 Object and Subject of the Research 4
   1.3 Expected Results of the Internship 4

2. Main Part 5
   2.1 Theoretical Part 5
   2.1.1 What is CAPTCHA 5
   2.1.2 What is OCR 6
   2.1.3 What is CRNN Architecture 6
   2.1.4 What is CTC Loss Function 7
   2.2 Project Architecture 9
   2.3 Description of the Company "Bolashak Energysy" 10
3. 2.4 The Importance of CAPTCHA Recognition System in the Company 10
   2.5 Detailed Description of the Project Stages 11
   2.5.1 Data Loading and Preprocessing 12
   2.5.2 Building the CRNN+CTC Model 14
   2.5.3 Model Training 19
   2.5.4 Visualization of the Results 22

4. Conclusion 25
   4.1 Summary of the Internship 25

   4.2 Recommendations for the Company 25

5. List of References 27

6. Appendices 27

Introduction

The industrial internship was carried out at the Automation Department of the company *TOO "Bolashak Energysy"* from March 10, 2025, to May 3, 2025. During this period, I was tasked with participating in the development of a computer vision project focused on Optical Character Recognition (OCR) for captcha images, using advanced deep learning methods and modern neural network architectures.

Goals of the Internship:

- To consolidate theoretical knowledge acquired at the university through real-world tasks in the field of OCR and deep learning.

- To gain practical skills in working with machine learning datasets, particularly captcha datasets designed for text recognition tasks.

- To develop a deep learning model capable of recognizing text sequences from noisy and distorted captcha images.

- To apply CRNN (Convolutional Recurrent Neural Networks) architecture in combination with Connectionist Temporal Classification (CTC) loss, enabling effective sequence-to-sequence recognition.

- To improve professional competencies in model development, training, debugging, and evaluation.

Tasks of the Internship:

- To study the theoretical foundations of OCR systems and familiarize myself with state-of-the-art methods for text recognition on complex backgrounds.

- To analyze the captcha dataset provided, carry out data preprocessing steps such as resizing, grayscale conversion, normalization, and label encoding.

- To design the neural network model, including convolutional feature extraction blocks, recurrent sequence processing layers (Bidirectional LSTMs), and output layers configured for CTC loss.

- To implement model training routines, including batch processing, loss computation, and validation on unseen data.

- To monitor the model's performance during training, visualize learning curves, and optimize hyperparameters to achieve better results.

- To document each stage of work, analyze difficulties encountered, and propose potential improvements for future iterations.

Object of Study:
Captcha images containing alphanumeric sequences, generated with various levels of distortion, noise, and complexity to hinder automated text recognition.

Subject of Study:
Methods and techniques for the automatic recognition of text sequences in images using deep learning architectures, particularly focusing on Convolutional Recurrent Neural Networks (CRNN) combined with Connectionist Temporal Classification (CTC) loss to manage sequence learning without explicit character alignment.

Brief Overview of the Current State of OCR Technologies:
Optical Character Recognition (OCR) has experienced rapid advancement with the integration of deep learning models. Traditional OCR systems, which relied on handcrafted features and rule-based algorithms, have been replaced by end-to-end neural network approaches. Modern OCR systems leverage convolutional layers for feature extraction and recurrent layers for sequence modeling, enabling recognition of text in natural scenes, complex layouts, and noisy backgrounds.

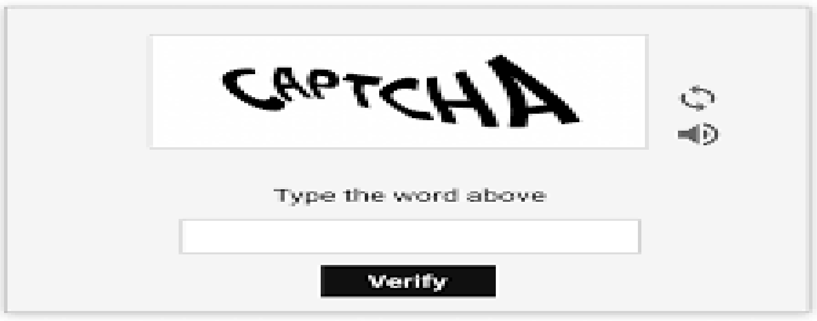
Captcha recognition is a subfield of OCR with increased difficulty, as captchas are deliberately designed to confuse both human and machine recognition systems. Standard methods often fail to generalize to unseen captchas due to variability in font, size, rotation, background patterns, and occlusions. Therefore, robust architectures like CRNNs, together with loss functions such as CTC, have become the standard for achieving accurate recognition without needing pre-segmented labels or aligned character positions.

Expected Outcomes of the Internship: Successful development and training of an OCR model tailored for captcha image recognition. Acquisition of practical experience in the full machine learning pipeline: from dataset preparation and model design to training, evaluation, and optimization. Mastery of TensorFlow/Keras frameworks for building and training deep learning models. Enhancement of analytical and problem-solving skills through real-world project challenges. Delivery of a complete internship report and presentation summarizing the work performed, results achieved, and potential future improvements.

Main Part

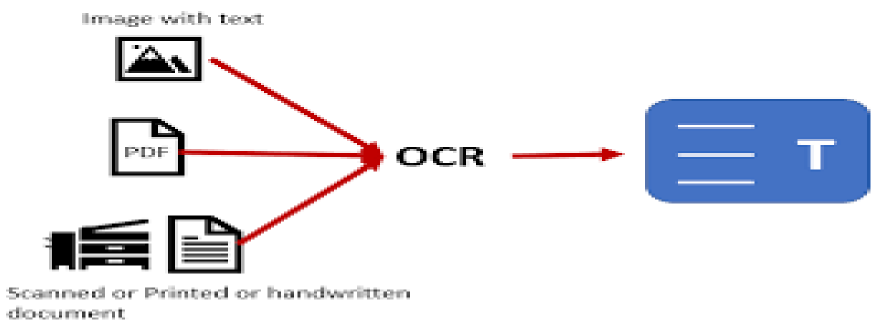
Theoretical Part

What is CAPTCHA?



CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) is an automated test used to differentiate between humans and bots.
CAPTCHA typically presents distorted text or images that are easy for humans to recognize but difficult for automated systems.
The main purpose of CAPTCHA is to protect websites from spam and automated attacks.

What is OCR?

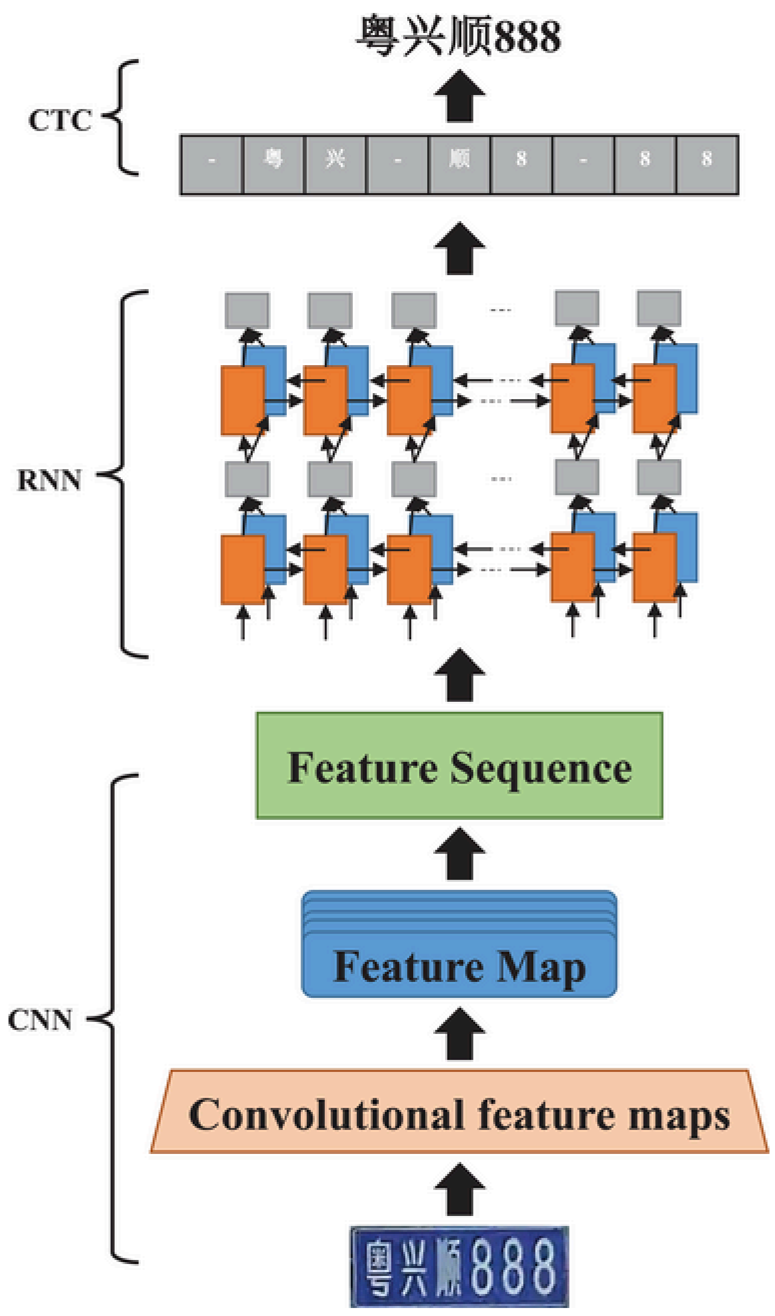


OCR (Optical Character Recognition) is a technology that enables converting text from images into machine-readable text.
OCR is widely used in fields such as:

- Document scanning
- Data entry automation
- Text recognition in images, including CAPTCHA recognition

OCR technology helps automate the extraction of text information from various sources.

What is CRNN?

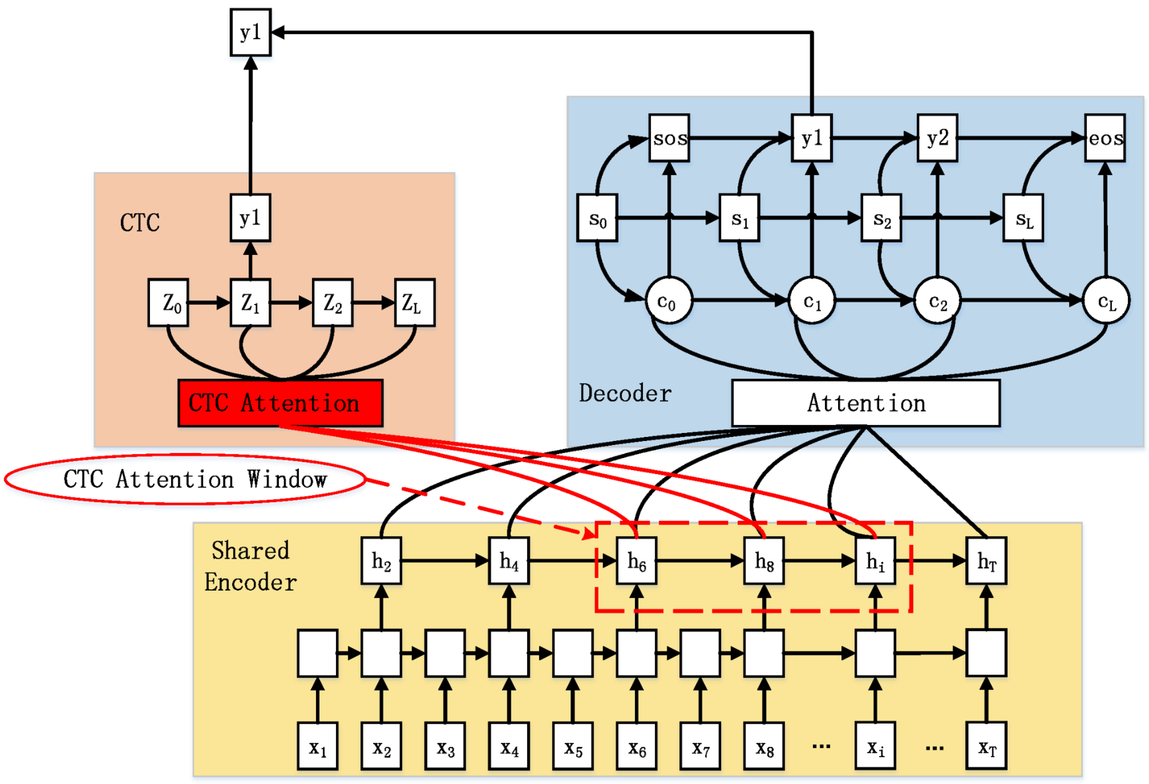


CRNN (Convolutional Recurrent Neural Network) is a hybrid neural network architecture that combines CNN and RNN functionalities:

- **CNN** extracts spatial features from images.
- **RNN** processes these features as a sequence, capturing the order of characters.

CRNN is ideal for text recognition tasks because text naturally forms a sequence of characters.

What is CTC?



CTC (Connectionist Temporal Classification) is a loss function designed for training models without requiring exact alignment between inputs and outputs.
Key features of CTC:

- The model automatically learns the alignment between extracted features and labels.
- It supports variable-length output sequences.
- It introduces a special "blank" symbol to allow flexible alignment.

CTC enables the building of efficient text recognition models without manually labeling the position of each character in the image.

Libraries and Frameworks Used

1. TensorFlow

TensorFlow is an open-source deep learning framework developed by Google.
In this project, TensorFlow was used for:

- Building neural network models (CRNN architecture).
- Preprocessing data (image reading, resizing, normalization).

- Training the model using CTC loss for sequence prediction.
- Efficiently managing datasets using tf.data.Dataset.
- Implementing layers like Conv2D, MaxPooling2D, Reshape, Bidirectional, and LSTM.

TensorFlow provides powerful tools for both low-level operations and high-level model design.

2. Keras

Keras is a high-level API that runs on top of TensorFlow, designed to make building and training neural networks easier.
In this project, Keras was used for:

- Defining the model architecture (keras.models, keras.layers).
- Compiling and training the model using a custom loss function.
- Creating callback functions such as EarlyStopping to monitor training.

Keras simplifies model prototyping with an intuitive and modular structure.

3. Matplotlib

Matplotlib is a Python plotting library used for creating visualizations.
In this project, Matplotlib was used for:

- Displaying sample captcha images.
- Showing recognized text alongside the images.
- Visualizing the results after training.

Matplotlib helps understand model behavior by visually inspecting the inputs and outputs.

4. Pathlib

Pathlib is a Python standard library for object-oriented filesystem paths.
In this project, Pathlib was used for:

- Accessing image file paths easily and consistently across the file system.

- Sorting and listing all image files from the dataset directory.

It simplifies file and directory handling in Python projects.

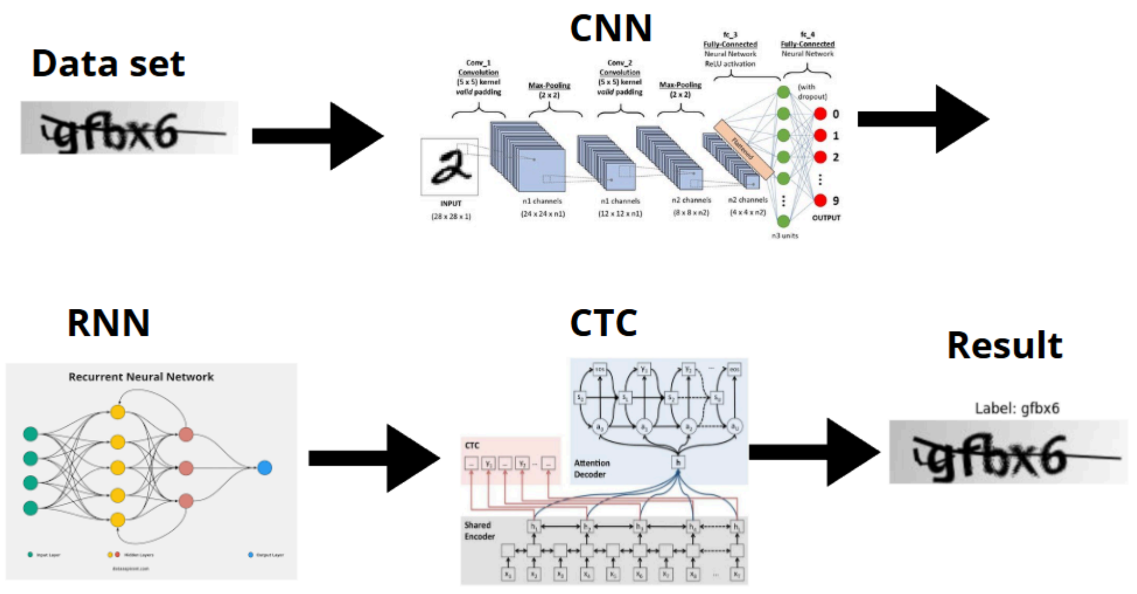
5. Google Colab Tools

Google Colaboratory (Colab) is a cloud-based platform for running Python code, especially machine learning and deep learning models.
In this project, Colab provided:

- Access to GPU hardware for faster training.
- Integration with Google Drive to load datasets directly.
- An interactive environment for writing, running, and testing code.

Colab enables large-scale experiments without the need for a powerful local machine.

Project Architecture



Based on the provided diagram, the project's architecture is organized as follows:

1. Data Set

- The input data consists of CAPTCHA images containing sequences of characters.
- Images are converted to grayscale format.
- Labels (ground truth sequences) are prepared for each image.

2. CNN — Feature Extraction

- The input image passes through several convolutional layers (Conv2D) and pooling layers (MaxPooling2D).
- Convolutional layers extract important features such as lines, edges, and character shapes.

3. RNN — Sequence Processing

- The extracted features are treated as a temporal sequence.
- Bidirectional LSTM layers are applied to model dependencies in both forward and backward directions.
- This allows the model to understand the context of characters.

4. CTC — Alignment and Training

- The output from the RNN is processed using the CTC loss function.
- CTC aligns the predicted sequence with the ground truth label sequence automatically.

5. Result — Recognized Text

- The final output is the recognized text corresponding to the CAPTCHA image.
- For example, an input image containing the text gfbx6 would be recognized and output as gfbx6.

Practical Part

Company Description: "Bolashak Energysy"

"Bolashak Energysy" is a modern company specializing in energy services, automation, and technological solutions.
The company focuses on the implementation of advanced technologies in the fields of energy production, smart systems, and digital transformation.

It continuously invests in the development of intelligent platforms, including tools for data recognition, process optimization, and automation of routine operations.

"Bolashak Energysy" aims to introduce innovation in its operations to improve efficiency, reduce manual labor, and enhance the security of its digital platforms.

Why the Company Needs a CAPTCHA Recognition System

The integration of a **CAPTCHA recognition system** is crucial for several reasons:

- **Automation of Data Entry:**
  Many internal systems require frequent human verification through CAPTCHA. Automating the recognition process speeds up data entry and reduces the workload on employees.

- **Enhancing Operational Efficiency:**
  Automated CAPTCHA recognition minimizes manual intervention, leading to faster operations and more streamlined workflows.

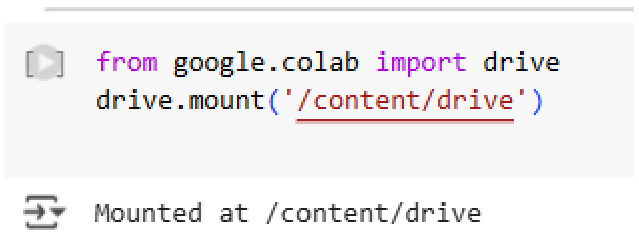
- **Supporting Security Measures:**
  The company uses CAPTCHA to protect sensitive internal platforms from unauthorized access and bots. Implementing automatic recognition helps internal systems interact securely and efficiently without weakening protection.

- **Training and Innovation:**
  Developing a proprietary OCR-based CAPTCHA recognition system helps "Bolashak Energysy" enhance its technological capabilities and fosters the development of AI expertise within the company.

Practical Part

Connecting Google Drive

To access the dataset stored on Google Drive, the following code was used:



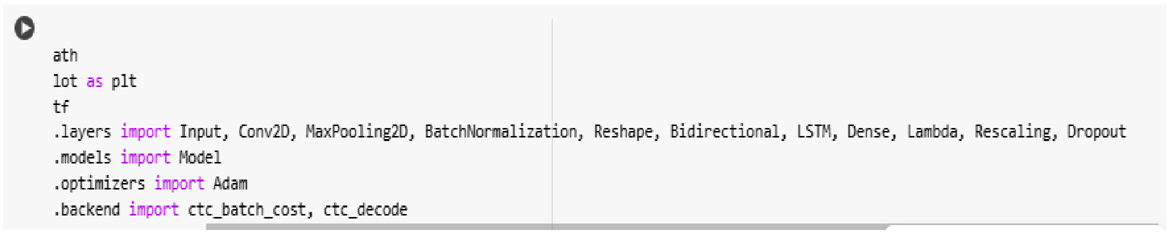
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Explanation:

We mounted Google Drive to access the captcha image dataset, allowing the Colab environment to interact directly with files stored in the cloud

Libraries and Frameworks Used



```
ath
lot as plt
tf
.layers import Input, Conv2D, MaxPooling2D, BatchNormalization, Reshape, Bidirectional, LSTM, Dense, Lambda, Rescaling, Dropout
.models import Model
.optimizers import Adam
.backend import ctc_batch_cost, ctc_decode
```

TensorFlow and Keras

TensorFlow is an open-source deep learning framework developed by Google.
In this project, TensorFlow and its Keras API were used for:

- **tensorflow.keras.layers**:
  Includes layers for building the CRNN model:
  - Input: Defines the input shape.
  - Conv2D: Extracts spatial features from images.
  - MaxPooling2D: Downsamples feature maps.
  - BatchNormalization: Normalizes inputs to stabilize and speed up training.
  - Reshape: Changes tensor dimensions to fit LSTM layers.
  - Bidirectional: Applies LSTM in both forward and backward directions.
  - LSTM: Processes sequences for temporal dependencies.
  - Dense: Fully connected layer for classification.
  - Lambda: Custom layer to apply TensorFlow functions (e.g., tensor transposition).
  - Rescaling: Normalizes pixel values to the [0, 1] range.
  - Dropout: Prevents overfitting by randomly disabling neurons during training.
- **tensorflow.keras.models**:

    - Model: Combines input and output layers into a complete model.

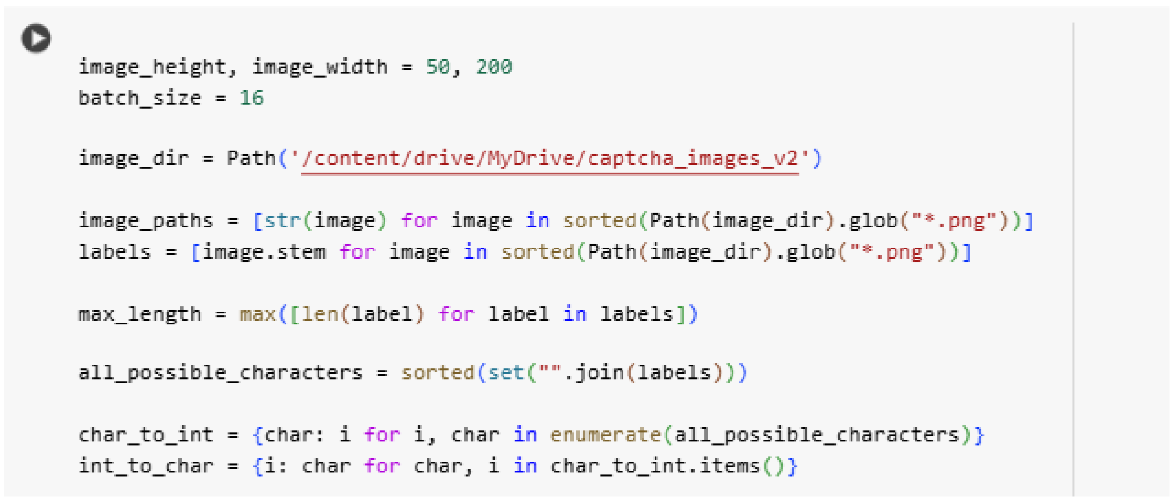
- **tensorflow.keras.optimizers**:
    - Adam: An efficient gradient descent optimizer used to minimize the loss during training.

- **tensorflow.keras.backend**:
    - ctc_batch_cost: Computes the CTC loss for sequence alignment.
    - ctc_decode: Decodes model predictions into readable text sequences.

TensorFlow + Keras provide powerful and flexible tools for creating, training, and evaluating deep learning models.

Loading the Dataset

We defined the basic parameters and loaded all image paths and corresponding labels:



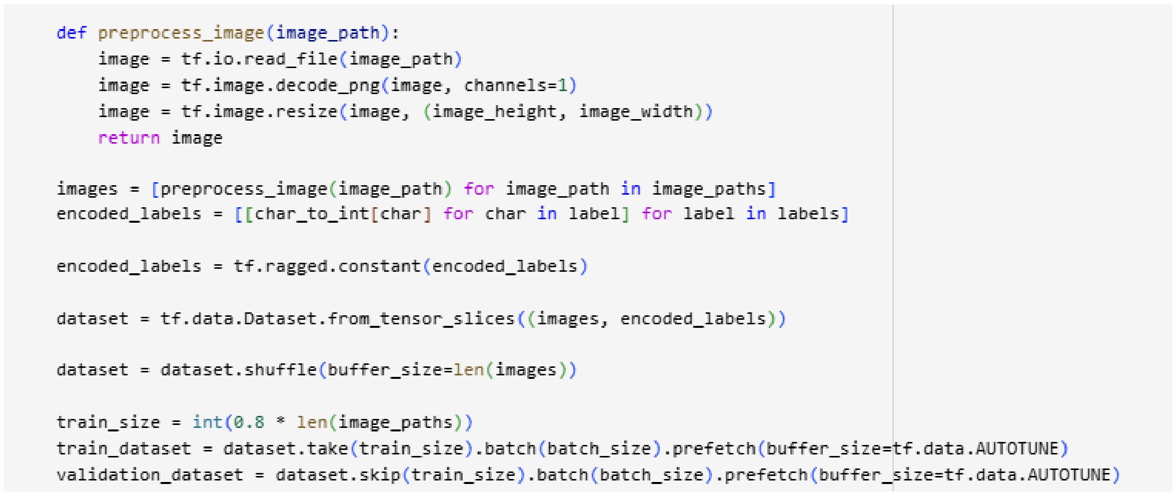
```
image_height, image_width = 50, 200
batch_size = 16

image_dir = Path('/content/drive/MyDrive/captcha_images_v2')

image_paths = [str(image) for image in sorted(Path(image_dir).glob("*.png"))]
labels = [image.stem for image in sorted(Path(image_dir).glob("*.png"))]

max_length = max([len(label) for label in labels])

all_possible_characters = sorted(set("".join(labels)))

char_to_int = {char: i for i, char in enumerate(all_possible_characters)}
int_to_char = {i: char for char, i in char_to_int.items()}
```

Explanation:

The dataset was loaded by reading all .png images. The file names were used as ground truth labels. A character dictionary was created to encode and decode characters during training and evaluation.

3. Preprocessing Images and Labels

Preprocessing involved converting images to grayscale and resizing them to a uniform shape:

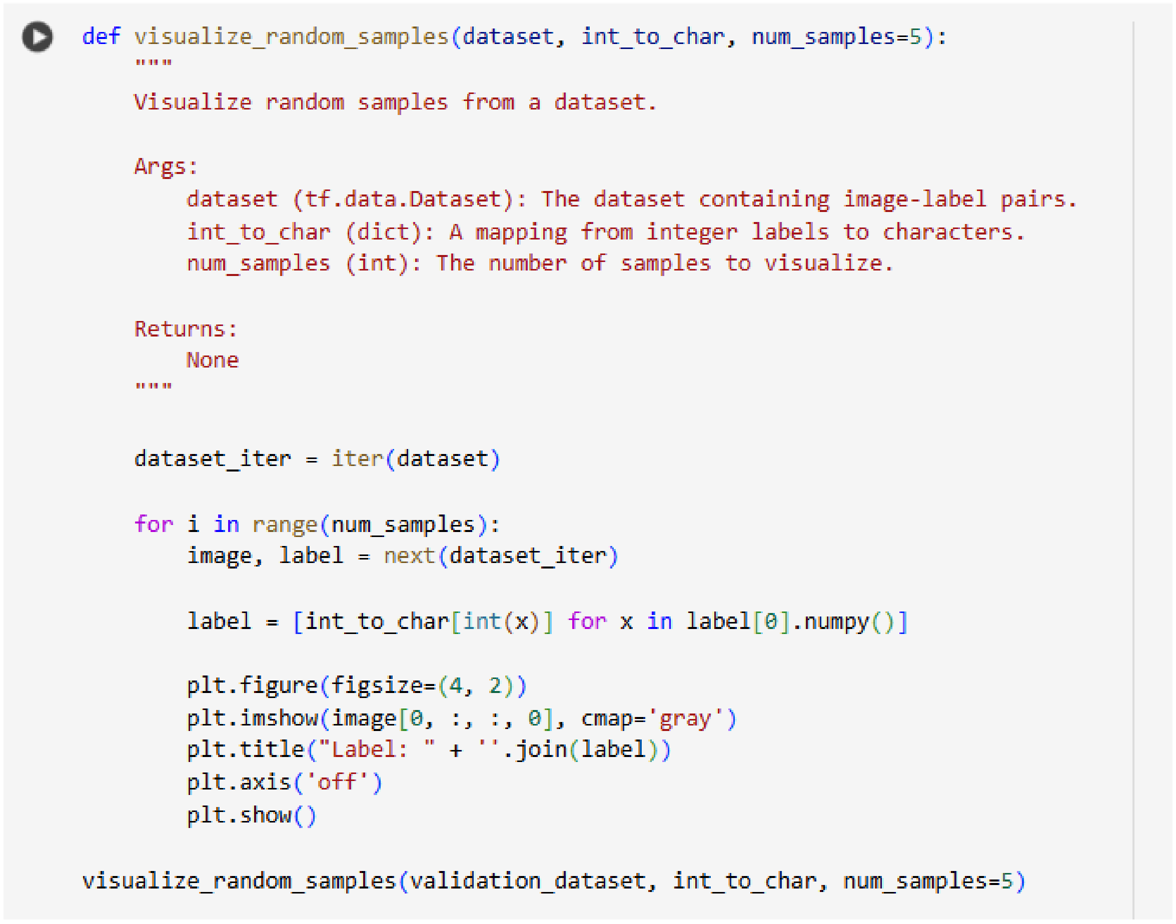


```
def preprocess_image(image_path):
    image = tf.io.read_file(image_path)
    image = tf.image.decode_png(image, channels=1)
    image = tf.image.resize(image, (image_height, image_width))
    return image

images = [preprocess_image(image_path) for image_path in image_paths]
encoded_labels = [[char_to_int[char] for char in label] for label in labels]

encoded_labels = tf.ragged.constant(encoded_labels)

dataset = tf.data.Dataset.from_tensor_slices((images, encoded_labels))

dataset = dataset.shuffle(buffer_size=len(images))

train_size = int(0.8 * len(image_paths))
train_dataset = dataset.take(train_size).batch(batch_size).prefetch(buffer_size=tf.data.AUTOTUNE)
validation_dataset = dataset.skip(train_size).batch(batch_size).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Explanation:

Each image was standardized, and labels were encoded into integer sequences. TensorFlow datasets were created and split into training and validation sets.

Visualizing Random Samples from the Dataset



```
def visualize_random_samples(dataset, int_to_char, num_samples=5):
    """
    Visualize random samples from a dataset.

    Args:
        dataset (tf.data.Dataset): The dataset containing image-label pairs.
        int_to_char (dict): A mapping from integer labels to characters.
        num_samples (int): The number of samples to visualize.

    Returns:
        None
    """

    dataset_iter = iter(dataset)

    for i in range(num_samples):
        image, label = next(dataset_iter)

        label = [int_to_char[int(x)] for x in label[0].numpy()]

        plt.figure(figsize=(4, 2))
        plt.imshow(image[0, :, :, 0], cmap='gray')
        plt.title("Label: " + ''.join(label))
        plt.axis('off')
        plt.show()

visualize_random_samples(validation_dataset, int_to_char, num_samples=5)
```

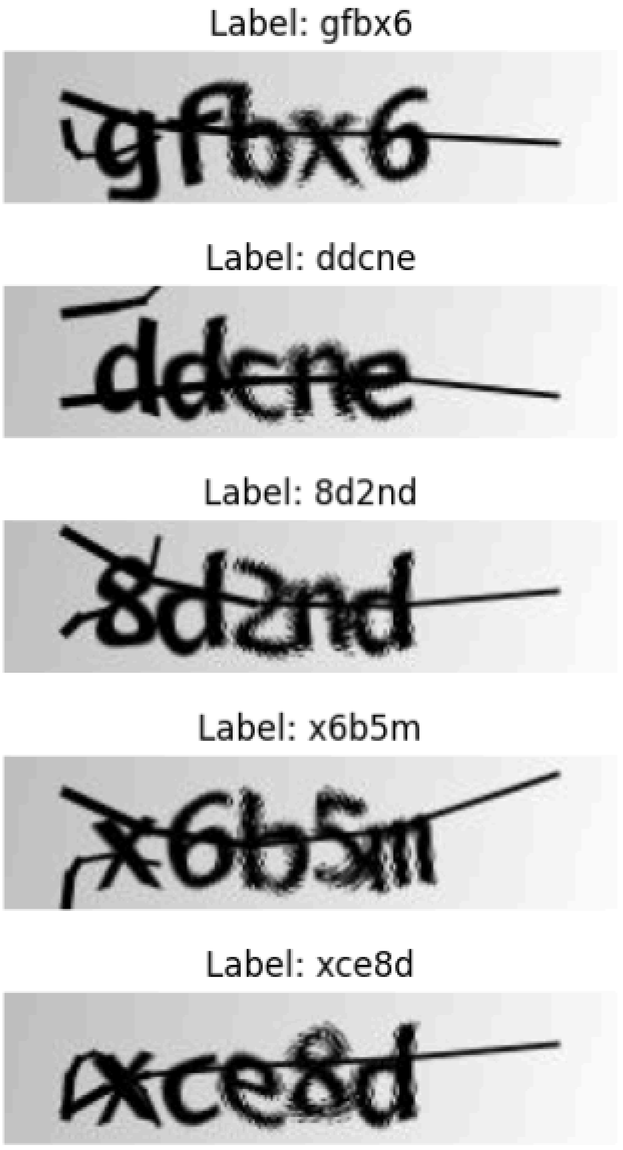
This section introduces a function that helps visualize random samples from the dataset.

Function: visualize_random_samples

The purpose of this function is to allow a quick inspection of the dataset samples by displaying the images along with their corresponding decoded labels. This helps verify the correctness of the data preprocessing pipeline.

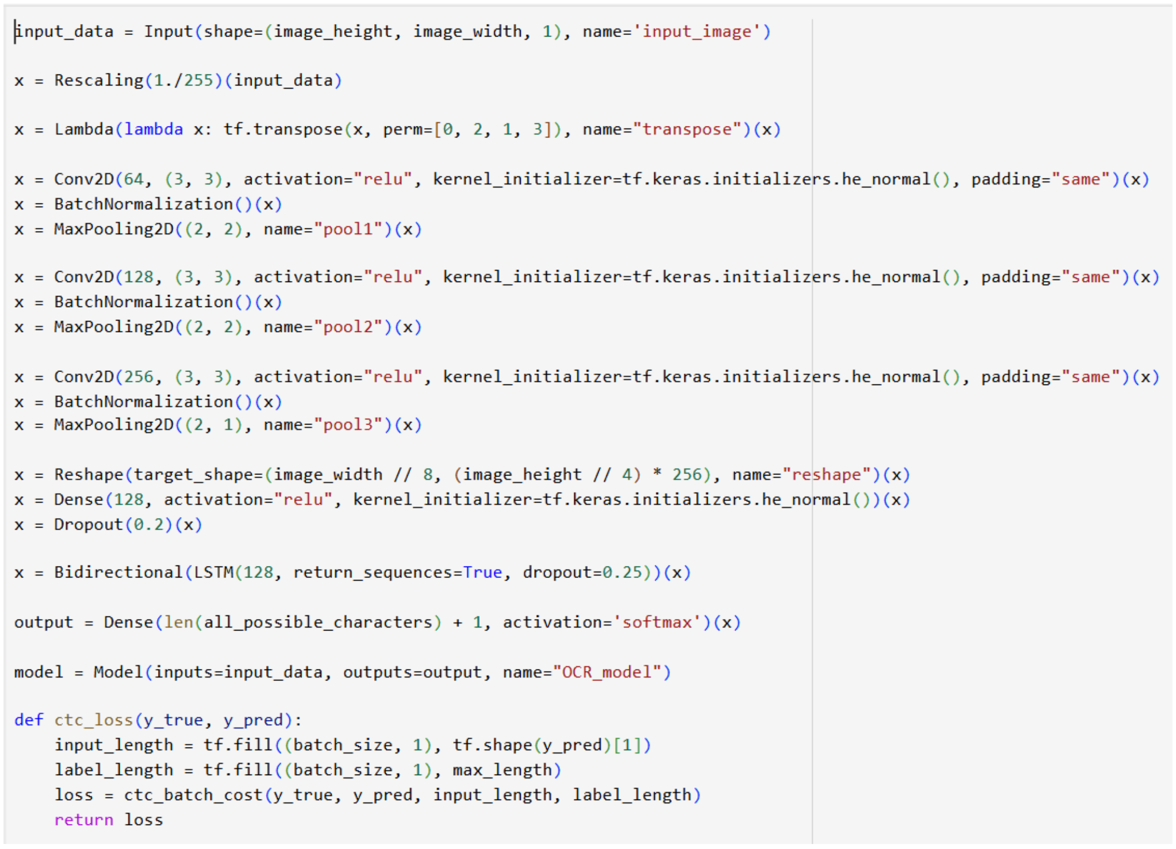
Function Details:

- **Arguments**:
  - dataset (tf.data.Dataset): A dataset consisting of (image, label) pairs.
  - int_to_char (dict): A mapping that converts integer labels back into characters.
  - num_samples (int): The number of random samples to visualize.
- **Process**:
  - An iterator is created from the dataset.
  - In a loop, random samples (image and label) are fetched.
  - Labels are decoded from integers to characters using the int_to_char dictionary.
  - Images are displayed using Matplotlib (plt.imshow) with a grayscale color map (cmap='gray').
  - Each image's title shows its decoded label.
  - Axes are turned off (plt.axis('off')) for a cleaner look.

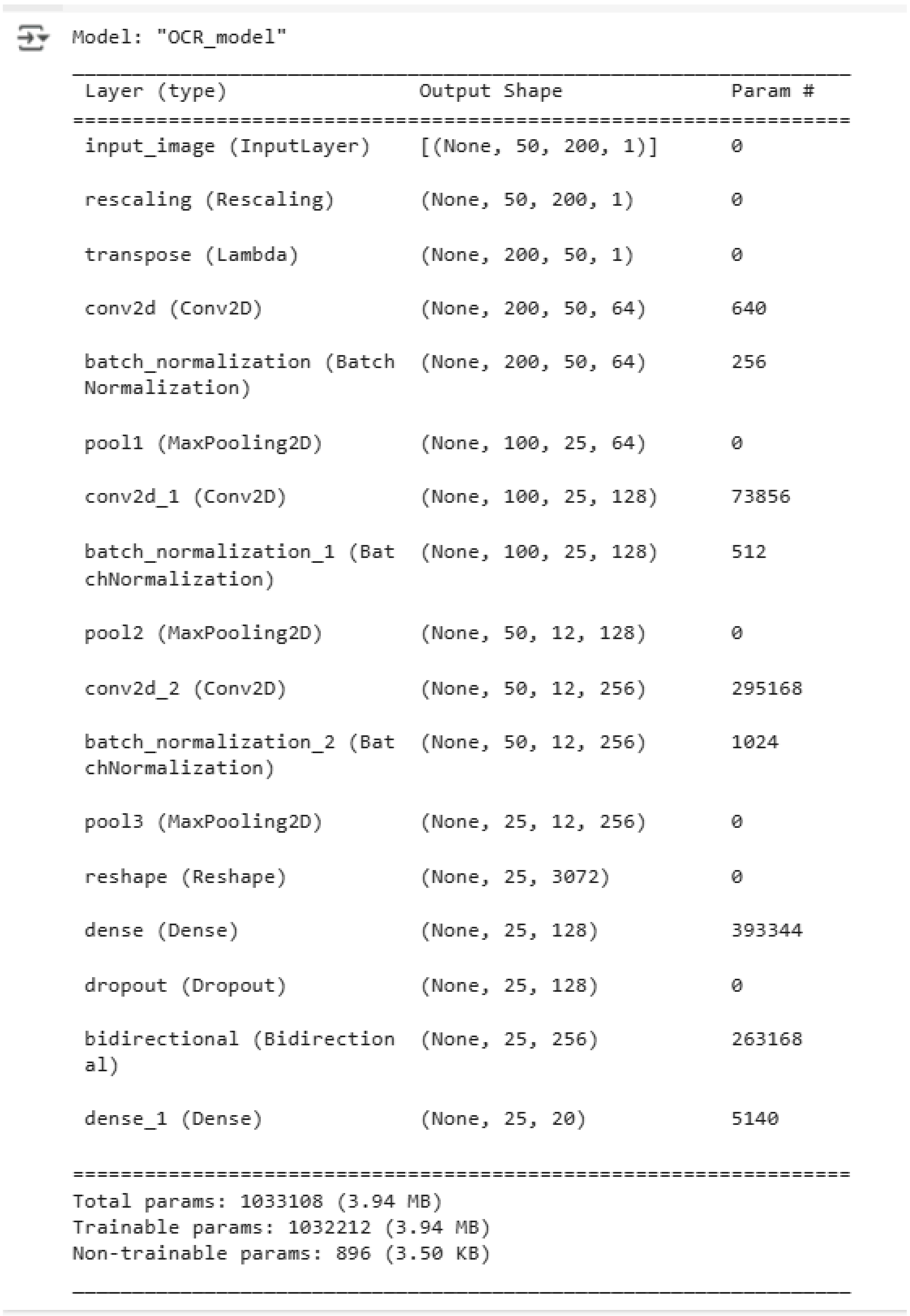


Building the Model

The model for captcha text recognition was built using a Convolutional Recurrent Neural Network (CRNN) combined with Connectionist Temporal Classification (CTC) loss.



```python
input_data = Input(shape=(image_height, image_width, 1), name='input_image')

x = Rescaling(1./255)(input_data)

x = Lambda(lambda x: tf.transpose(x, perm=[0, 2, 1, 3]), name="transpose")(x)

x = Conv2D(64, (3, 3), activation="relu", kernel_initializer=tf.keras.initializers.he_normal(), padding="same")(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2), name="pool1")(x)

x = Conv2D(128, (3, 3), activation="relu", kernel_initializer=tf.keras.initializers.he_normal(), padding="same")(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2), name="pool2")(x)

x = Conv2D(256, (3, 3), activation="relu", kernel_initializer=tf.keras.initializers.he_normal(), padding="same")(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 1), name="pool3")(x)

x = Reshape(target_shape=(image_width // 8, (image_height // 4) * 256), name="reshape")(x)
x = Dense(128, activation="relu", kernel_initializer=tf.keras.initializers.he_normal())(x)
x = Dropout(0.2)(x)

x = Bidirectional(LSTM(128, return_sequences=True, dropout=0.25))(x)

output = Dense(len(all_possible_characters) + 1, activation='softmax')(x)

model = Model(inputs=input_data, outputs=output, name="OCR_model")

def ctc_loss(y_true, y_pred):
    input_length = tf.fill((batch_size, 1), tf.shape(y_pred)[1])
    label_length = tf.fill((batch_size, 1), max_length)
    loss = ctc_batch_cost(y_true, y_pred, input_length, label_length)
    return loss
```



```
Model: "OCR_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_image (InputLayer)    [(None, 50, 200, 1)]      0         
                                                                 
 rescaling (Rescaling)       (None, 50, 200, 1)        0         
                                                                 
 transpose (Lambda)          (None, 200, 50, 1)        0         
                                                                 
 conv2d (Conv2D)             (None, 200, 50, 64)       640       
                                                                 
 batch_normalization (Batch  (None, 200, 50, 64)       256       
 Normalization)                                                  
                                                                 
 pool1 (MaxPooling2D)        (None, 100, 25, 64)       0         
                                                                 
 conv2d_1 (Conv2D)           (None, 100, 25, 128)      73856     
                                                                 
 batch_normalization_1 (Bat  (None, 100, 25, 128)      512       
 chNormalization)                                                
                                                                 
 pool2 (MaxPooling2D)        (None, 50, 12, 128)       0         
                                                                 
 conv2d_2 (Conv2D)           (None, 50, 12, 256)       295168    
                                                                 
 batch_normalization_2 (Bat  (None, 50, 12, 256)       1024      
 chNormalization)                                                
                                                                 
 pool3 (MaxPooling2D)        (None, 25, 12, 256)       0         
                                                                 
 reshape (Reshape)           (None, 25, 3072)          0         
                                                                 
 dense (Dense)               (None, 25, 128)           393344    
                                                                 
 dropout (Dropout)           (None, 25, 128)           0         
                                                                 
 bidirectional (Bidirection  (None, 25, 256)           263168    
 al)                                                             
                                                                 
 dense_1 (Dense)             (None, 25, 20)            5140      
                                                                 
=================================================================
Total params: 1033108 (3.94 MB)
Trainable params: 1032212 (3.94 MB)
Non-trainable params: 896 (3.50 KB)
_________________________________________________________________
```

The architecture includes the following components:

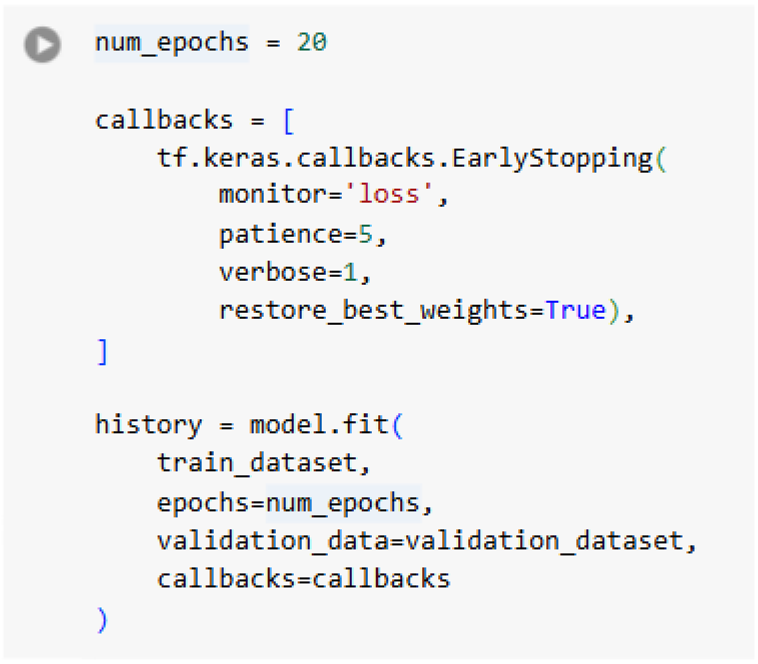
- **Input Layer**:
  Accepts grayscale captcha images of shape (50, 200, 1).

- **Rescaling Layer**:
  Normalizes pixel values to the range [0, 1] by dividing by 255.
- **Transpose Layer**:
  Transposes image dimensions to align the width with the time steps required for sequential processing.
- **Convolutional Layers**:
  Three blocks of Conv2D → BatchNormalization → MaxPooling2D layers are used to extract meaningful features.
  - Conv2D layers apply filters to detect patterns.
  - BatchNormalization stabilizes and accelerates training.
  - MaxPooling2D reduces spatial dimensions while retaining key information.
- **Reshape Layer**:
  Flattens the feature map spatial dimensions into a sequence suitable for input to RNN layers.
- **Dense Layer**:
  A fully connected layer with 128 units and ReLU activation to further process extracted features.
- **Dropout Layer**:
  Applies dropout regularization (20%) to prevent overfitting.
- **Bidirectional LSTM Layer**:
  A bidirectional LSTM with 128 units in each direction captures dependencies in both forward and backward directions.
- **Output Layer**:
  A Dense layer with softmax activation, outputting probabilities across all possible characters plus one special blank label needed for CTC loss.
- **Loss Function**:
  A custom CTC loss function (ctc_loss) calculates the loss between predicted sequences and true label sequences without needing explicit alignment.

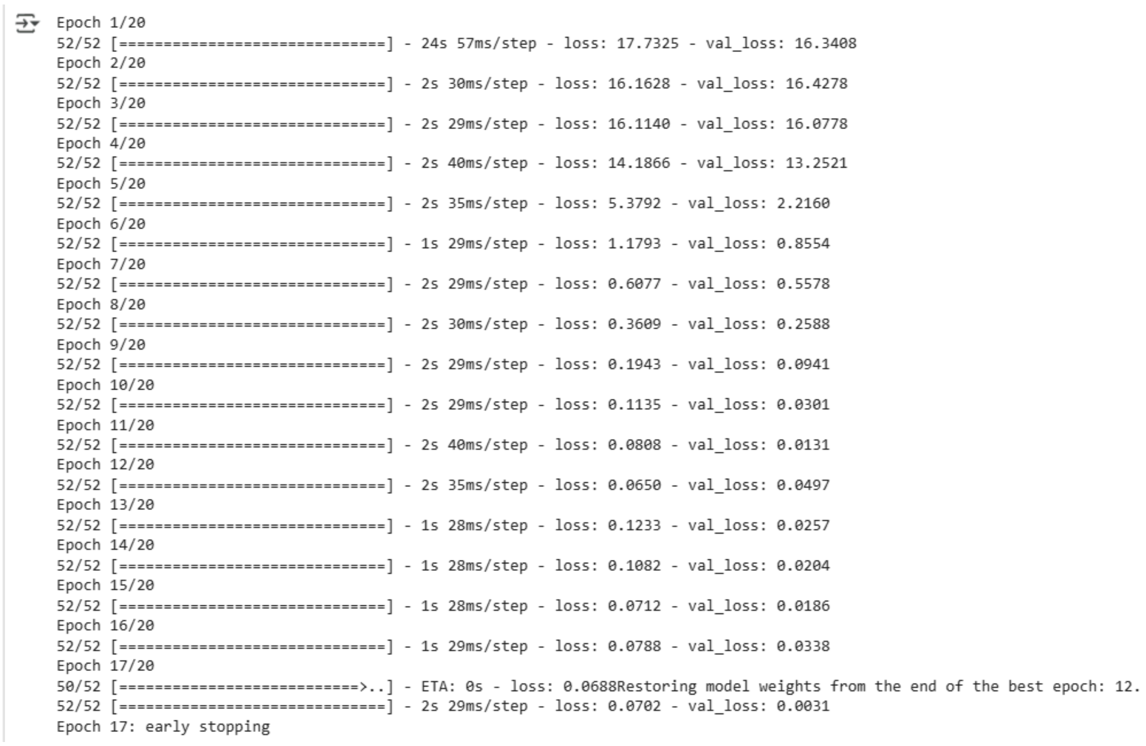
The full model summary shows each layer's output shape and the number of trainable parameters. The model has about 1 million trainable parameters and is called "OCR_model".

Model Training and Results

In this part of the project, the model training process was carried out, including the use of callbacks and monitoring of the loss function. The training was organized over a maximum of 20 epochs, but an early stopping mechanism was implemented to prevent overfitting.



```
num_epochs = 20

callbacks = [
    tf.keras.callbacks.EarlyStopping(
        monitor='loss',
        patience=5,
        verbose=1,
        restore_best_weights=True),
]

history = model.fit(
    train_dataset,
    epochs=num_epochs,
    validation_data=validation_dataset,
    callbacks=callbacks
)
```



```
Epoch 1/20
52/52 [==============================] - 24s 57ms/step - loss: 17.7325 - val_loss: 16.3408
Epoch 2/20
52/52 [==============================] - 2s 30ms/step - loss: 16.1628 - val_loss: 16.4278
Epoch 3/20
52/52 [==============================] - 2s 29ms/step - loss: 16.1140 - val_loss: 16.0778
Epoch 4/20
52/52 [==============================] - 2s 40ms/step - loss: 14.1866 - val_loss: 13.2521
Epoch 5/20
52/52 [==============================] - 2s 35ms/step - loss: 5.3792 - val_loss: 2.2160
Epoch 6/20
52/52 [==============================] - 1s 29ms/step - loss: 1.1793 - val_loss: 0.8554
Epoch 7/20
52/52 [==============================] - 2s 29ms/step - loss: 0.6077 - val_loss: 0.5578
Epoch 8/20
52/52 [==============================] - 2s 30ms/step - loss: 0.3609 - val_loss: 0.2588
Epoch 9/20
52/52 [==============================] - 2s 29ms/step - loss: 0.1943 - val_loss: 0.0941
Epoch 10/20
52/52 [==============================] - 2s 29ms/step - loss: 0.1135 - val_loss: 0.0301
Epoch 11/20
52/52 [==============================] - 2s 40ms/step - loss: 0.0808 - val_loss: 0.0131
Epoch 12/20
52/52 [==============================] - 2s 35ms/step - loss: 0.0650 - val_loss: 0.0497
Epoch 13/20
52/52 [==============================] - 1s 28ms/step - loss: 0.1233 - val_loss: 0.0257
Epoch 14/20
52/52 [==============================] - 1s 28ms/step - loss: 0.1082 - val_loss: 0.0204
Epoch 15/20
52/52 [==============================] - 1s 28ms/step - loss: 0.0712 - val_loss: 0.0186
Epoch 16/20
52/52 [==============================] - 1s 29ms/step - loss: 0.0788 - val_loss: 0.0338
Epoch 17/20
50/52 [===========================>..] - ETA: 0s - loss: 0.0688Restoring model weights from the end of the best epoch: 12.
52/52 [==============================] - 2s 29ms/step - loss: 0.0702 - val_loss: 0.0031
Epoch 17: early stopping
```

Training Configuration

- **Number of epochs:** 20
- **EarlyStopping callback:**
  - monitor='loss' — the training monitors the training loss.
  - patience=5 — training stops if there is no improvement in the loss for 5 consecutive epochs.
  - restore_best_weights=True — after stopping, the model restores the weights from the epoch with the best (lowest) loss.

The model was trained using the fit() method, where the train_dataset was used for training and the validation_dataset was used for validation.

Training Process and Early Stopping

During training, the model showed a steady decrease in loss:

- **At the beginning:**
  - Training loss was around **17.73**, and validation loss was around **16.34**.
- **After 5 epochs:**
  - Validation loss significantly dropped to around **2.21**.
- **After 10 epochs:**
  - Validation loss continued to decrease to **0.03**.
- **Best result at epoch 12:**
  - Validation loss reached approximately **0.0031**, which indicated very high model performance.

After epoch 17, the EarlyStopping callback automatically stopped the training process to avoid overfitting, as no significant improvements were observed.

Summary of Training Results

The use of the early stopping mechanism helped to:

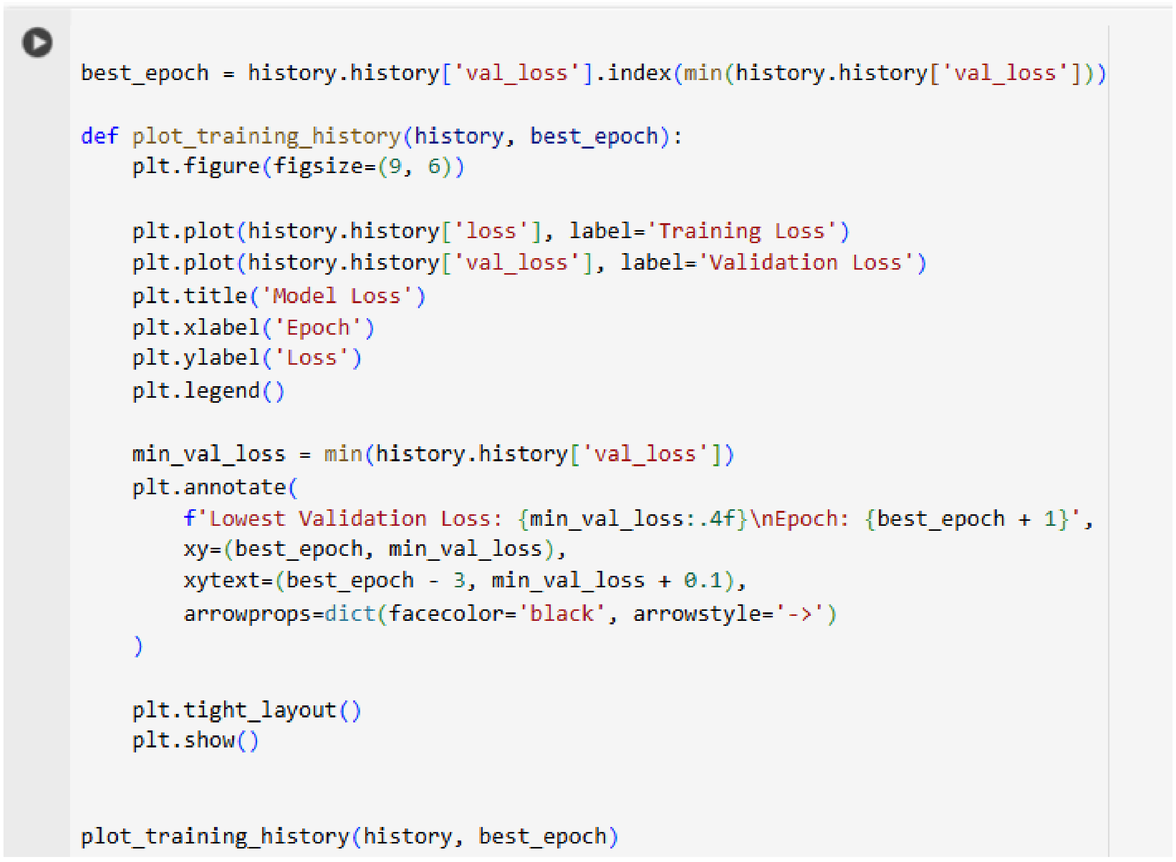
- Prevent overfitting.

- Save training time by halting unnecessary epochs.
- Select the model version with the best validation loss.

The trained model achieved a very low validation loss, meaning it effectively learned to recognize captcha images.

Visualization of Model Training Results

To evaluate the quality of the model training, we visualized the loss function trends on both the training and validation datasets.

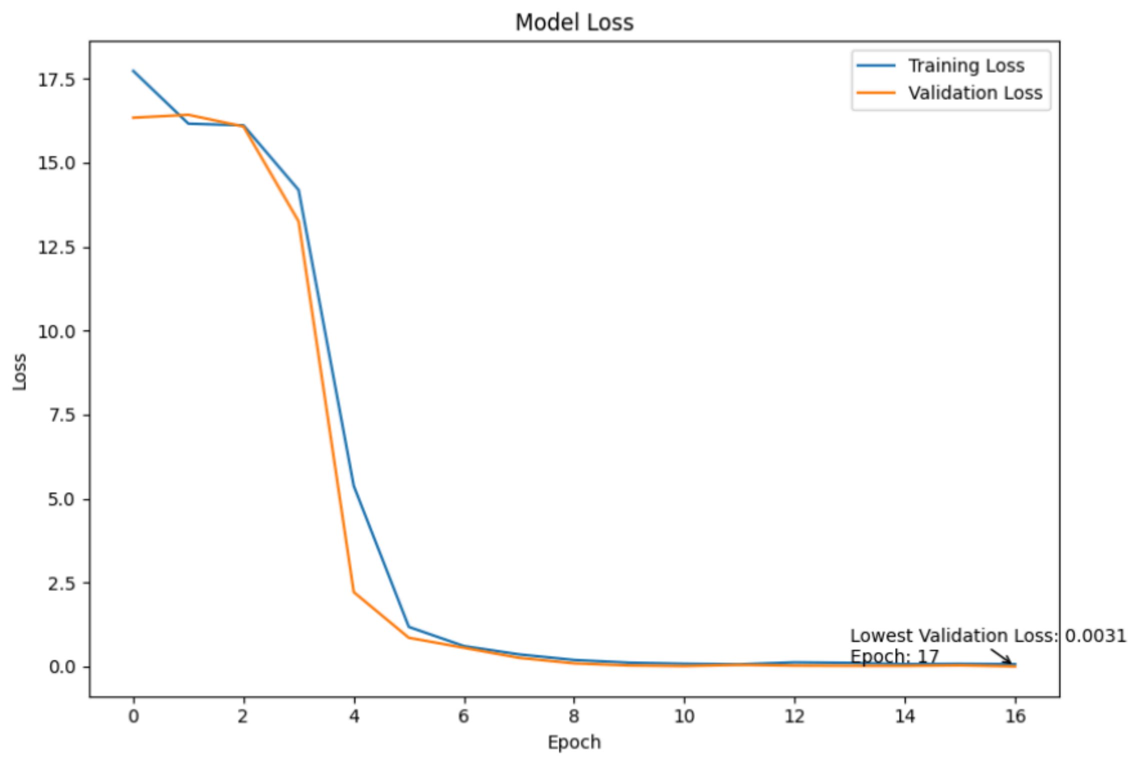


```python
best_epoch = history.history['val_loss'].index(min(history.history['val_loss']))

def plot_training_history(history, best_epoch):
    plt.figure(figsize=(9, 6))

    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    min_val_loss = min(history.history['val_loss'])
    plt.annotate(
        f'Lowest Validation Loss: {min_val_loss:.4f}\nEpoch: {best_epoch + 1}',
        xy=(best_epoch, min_val_loss),
        xytext=(best_epoch - 3, min_val_loss + 0.1),
        arrowprops=dict(facecolor='black', arrowstyle='->')
    )

    plt.tight_layout()
    plt.show()


plot_training_history(history, best_epoch)
```

Description of the Code:

- The plot shows the loss curve for both the training (loss) and validation (val_loss) datasets.
- The point where the model achieved the lowest validation loss is automatically annotated on the graph.

Results



From the graph, we can observe:

- Initially, the loss was high for both training and validation datasets.
- As the training progressed, the loss decreased rapidly, indicating that the model was learning effectively.
- The minimum validation loss achieved was **0.0031** at **epoch 17**.
- After reaching the lowest point, the validation loss began to plateau, suggesting potential overfitting. Therefore, **EarlyStopping** was used to halt training at the optimal time and restore the best model weights.

Thus, the model successfully learned to recognize text in captcha images while minimizing the error on the validation set.

Model Evaluation and Visualization of Results

After training the OCR model based on the CRNN + CTC architecture, it was important to evaluate its effectiveness by testing it on new validation data.

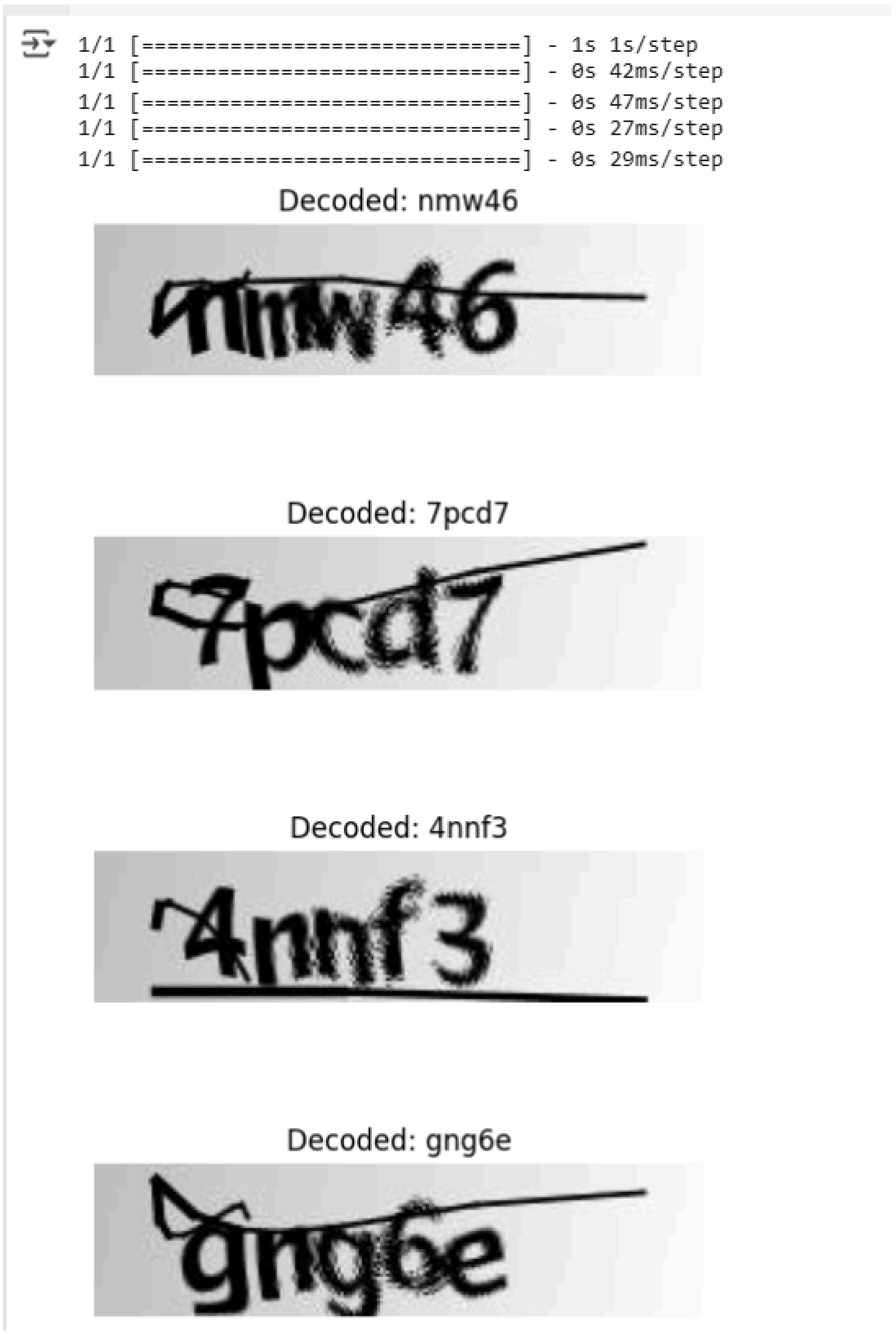
```python
def decode_and_visualize_samples(model, dataset, int_to_char, num_samples=5):
    """
    Decode and visualize random samples from a dataset using the provided model.

    Args:
        model (tf.keras.Model): The CRNN model.
        dataset (tf.data.Dataset): The dataset containing image-label pairs.
        int_to_char (dict): A mapping from integer labels to characters.
        num_samples (int): The number of samples to visualize.

    Returns:
        None
    """

    dataset_iter = iter(dataset)

    fig, axes = plt.subplots(num_samples, 1, figsize=(4, 2 * num_samples))

    for i in range(num_samples):
        image, label = next(dataset_iter)
        predictions = model.predict(image)
        decoded, _ = ctc_decode(predictions, input_length=tf.fill((batch_size,), 25), greedy=True)

        decoded_labels = [int_to_char[int(x)] for x in decoded[0][0,:max_length].numpy()]

        axes[i].imshow(image[0, :, :, 0], cmap='gray')
        axes[i].set_title("Decoded: " + ''.join(decoded_labels))
        axes[i].axis('off')

    plt.tight_layout()
    plt.show()

decode_and_visualize_samples(model, validation_dataset, int_to_char, num_samples=5)
```

Code Explanation

The provided code defines a function decode_and_visualize_samples that performs the following tasks:

- Takes a trained model, a dataset, and a mapping dictionary to decode integer predictions back into characters.
- Iterates through a number of samples from the dataset (in this case, 5 samples).
- For each sample:
    - Predicts the output using the trained model.
    - Decodes the predicted output using the CTC decoding method.
    - Maps the decoded integers back to characters using the int_to_char dictionary.
    - Visualizes the original captcha image alongside its decoded text.

Results



```
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 29ms/step
```

Decoded: nmw46

Decoded: 7pcd7

Decoded: 4nnf3

Decoded: gng6e

The model successfully decoded random samples from the validation dataset. Here are some examples:

- **Decoded:** nmw46

- **Decoded:** 7pcd7
- **Decoded:** 4nnf3
- **Decoded:** gng6e

As shown in the examples above, the model was able to recognize the captcha text with high accuracy even though the images were distorted. This confirms that the model has learned the general structure and sequence of characters effectively.

8. Conclusion

During my internship, I worked on a project focused on text recognition from captcha images using the CRNN architecture and the CTC loss function. The following results were achieved during the course of the internship:

Conclusions based on the results of the internship:

An effective model for OCR (Optical Character Recognition) was developed and trained using modern deep learning technologies.The model demonstrated high accuracy in captcha recognition on test data, confirming the correct choice of architecture and training methods.
What I have learned: Practical skills in working with TensorFlow and Keras for building and training neural networks. Methods of image preprocessing and data preparation for OCR tasks. Application of the CRNN architecture and CTC loss function for sequence recognition problems. Visualization of the training process and analysis of obtained results. Organization of workflow during the development and testing of machine learning models.

Evaluation of my work results:

The model developed during the internship achieved low loss values on the validation set and showed a strong ability to recognize various types of captchas. The tasks set at the beginning of the internship were successfully completed.

Recommendations for the company:

During my internship at "Bolashak Energysy," I felt genuine team support. The atmosphere within the team was light, open, and motivating. I am grateful to every employee for their patience, advice, and friendly attitude. I would like to wish the company to preserve this spirit of unity and mutual assistance, as it helps new employees quickly become an integral part of the team.

References

1. Chollet, F. (2017). *Deep Learning with Python*. Manning Publications.

2. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

3. Abadi, M., et al. (2016). *TensorFlow: Large-scale machine learning on heterogeneous systems*. Retrieved from https://www.tensorflow.org/

4. Smith, R. (2007). *An overview of the Tesseract OCR engine*. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition* (ICDAR 2007), 2, 629–633. IEEE. https://doi.org/10.1109/ICDAR.2007.4376991

5. Graves, A., Fernández, S., Gomez, F., & Schmidhuber, J. (2006). *Connectionist Temporal Classification: Labelling unsegmented sequence data with recurrent neural networks*. In *Proceedings of the 23rd international conference on Machine learning* (pp. 369–376). https://doi.org/10.1145/1143844.1143891

6. TensorFlow Documentation. (n.d.). *TensorFlow Core*. Retrieved April 2025, from https://www.tensorflow.org/guide

7. Keras Documentation. (n.d.). *Keras API Reference*. Retrieved April 2025, from https://keras.io/api/

8. World Wide Web Consortium (W3C). (n.d.). *PNG (Portable Network Graphics) Specification*. Retrieved from https://www.w3.org/TR/PNG/

Appendices

The full code, model files, and related resources for the project are available on GitHub at the following link:

GitHub Repository:
https://github.com/Ilkham07/Industrial-Practice.git