Internship Project Title	PRESI S PREMACHANDRAN
Project Title	RIO-125-Automate extraction of handwritten text from an image
Name of the Company	TCS iON
Name of the Industry Mentor	Debashis Roy
Name of the Institute	ICT Academy of Kerala

Start Date	End Date	Total Effort (hrs.)	Project Environment	Tools used
10 February 2023	19 March 2023	125	VSC,Google Colab – Jupyter Notebook, Anaconda Navigator (Jupyter Notebook).	IAM Dataset , Tensorflow 2.0,mltu, Google Colab,VSC

Project Synopsis:

- **Title of the Project -** Automate extraction of handwritten text from an image
- **Introduction** - This is TCS iON-RIO 210-Industry Project. This project is basically focused on identification and recognition of handwritten text from an image. The handwritten text detection model implemented in the project is based on TensorFlow, a popular deep learning framework. It is a convolutional neural network that takes as input an image containing handwritten text and outputs the corresponding text. Overall, this project provides an example of how to implement a handwritten text detection model using TensorFlow and various data preprocessing and augmentation techniques to improve model performance.

- **Objective and Aim -** The objective of automatic extraction of handwritten text using deep learning is to develop accurate and efficient algorithms that can recognize and transcribe text from images of handwritten documents. This technology aims to automate the process of digitizing handwritten text, which is a time-consuming and error-prone task when done manually.

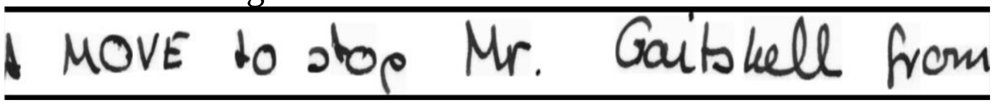
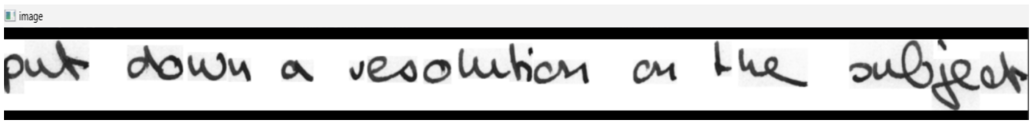
  The aim of automatic extraction of handwritten text using deep learning is to improve the efficiency and accuracy of text recognition and transcription. By developing robust deep learning models that can handle different handwriting styles and languages, this technology aims to make the process of digitizing handwritten text faster, more accurate, and more accessible. The ultimate goal is to enable efficient and automated analysis of large volumes of handwritten text data for various applications, including historical research, language translation, and data entry.

Solution Approach:

The solution approach for this handwritten text detection from image using TensorFlow can be summarized as follows:

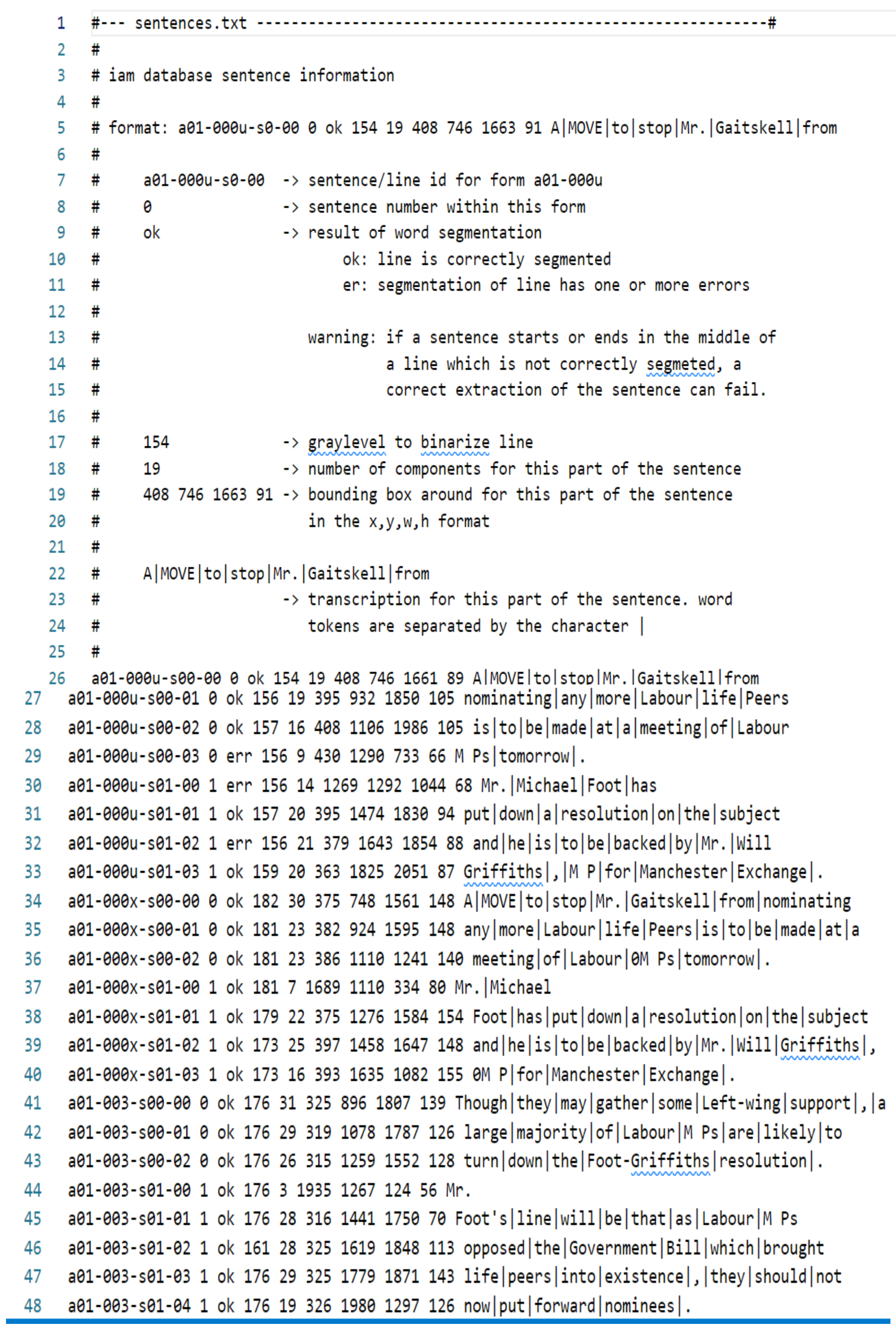
1. Collecting the dataset and preprocessing data: The first step is to collect the dataset containing handwritten text images and process it by resizing, normalizing and augmenting the images.
2. Building the model: The next step is to build the model using TensorFlow. The model architecture usually consists of a convolutional neural network (CNN) that extracts features from the input image, followed by a fully connected layer that maps the extracted features to the output classes. The model is trained using backpropagation to optimize the weights and biases of the network.
3. Training the model: The model is trained on the preprocessed dataset by minimizing the loss function using an optimization algorithm such as stochastic gradient descent (SGD).
4. Evaluating the model: The trained model is evaluated on a separate test set to measure its performance in terms of accuracy, precision, recall, F1 score, and other metrics.
5. Testing the model: Once the model is trained and evaluated, the tested result will display.

1. **Collecting the dataset and preprocessing data:**
   The first step in using TensorFlow for handwritten sentence recognition is to collect a dataset of handwritten sentences. This dataset should include a variety of handwriting styles and should be large enough to train a machine-learning model.
   we want to get a robust and accurate machine learning model, the more data we have better it is, but for simplicity reasons, I chose only the IAM dataset. IAM: The IAM dataset is a collection of handwritten text images provided by the Institute of Document Analysis and Pattern Recognition at the University of Basel. The Dataset consists of 1,539 pages of handwritten text in English, German, and French and contains over 6,000 lines of text. It has been widely used in machine learning and computer vision research
   It contain a sentence.text and a sentence folder. The code first specifies the location of the text file and the folder where the images are stored. Then, it creates three empty variables, Dataset, vocab, and max_len. The code then opens the text file and reads all the lines. For each line in the text file, the code checks if the line starts with "#" or if the third word in the line is "err"; if so, it skips the line. If not, it takes the first three letters and the first eight letters of the first word in the line and uses them to create the folder path where the image is stored. It also creates a file name by adding ".png" to the first word. It also takes the last word in the line, which is the text written in the image, and removes the newline character at the end. Finally, the code adds the image path and the label to the Dataset and adds all the characters in the label to the vocab. It also keeps track of the most extended label in the Dataset so far.
   Next, the Dataset must be preprocessed and prepared for use in the model. This can involve tasks such as reading images, converting the images of handwritten sentences into a suitable format, indexing labels, padding labels, and dividing the Dataset into training and testing sets. Precisely in the same way as before, I am doing these steps with a custom DataProvider object.
   Some handwritten images in dataset:

   - 
     A MOVE to stop Mr. Gaitskell from
   - 

     put down a resolution on the subject

Sentence.text:



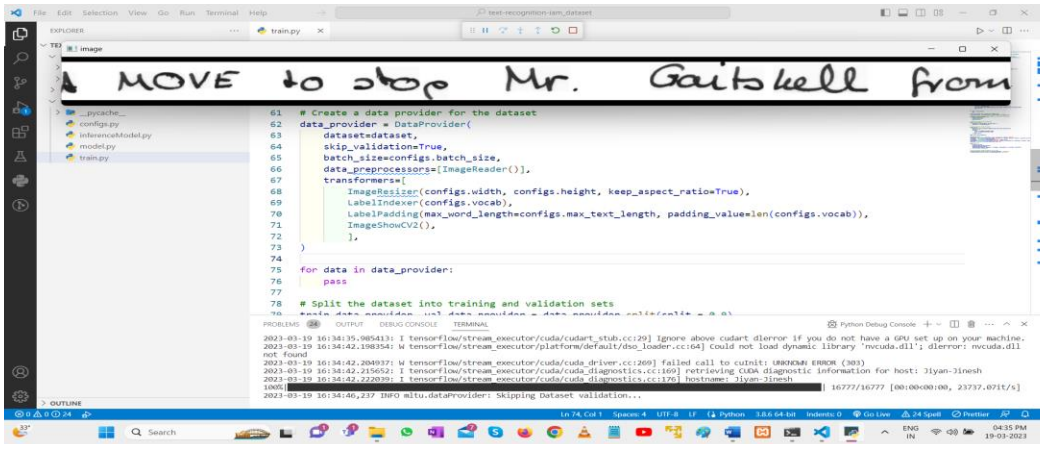
```
#--- sentences.txt -----------------------------------------------------------#
#
# iam database sentence information
#
# format: a01-000u-s0-00 0 ok 154 19 408 746 1663 91 A|MOVE|to|stop|Mr.|Gaitskell|from
#
#     a01-000u-s0-00  -> sentence/line id for form a01-000u
#     0               -> sentence number within this form
#     ok              -> result of word segmentation
#                           ok: line is correctly segmented
#                           er: segmentation of line has one or more errors
#
#                        warning: if a sentence starts or ends in the middle of
#                                 a line which is not correctly segmeted, a
#                                 correct extraction of the sentence can fail.
#
#     154             -> graylevel to binarize line
#     19              -> number of components for this part of the sentence
#     408 746 1663 91 -> bounding box around for this part of the sentence
#                        in the x,y,w,h format
#
#     A|MOVE|to|stop|Mr.|Gaitskell|from
#                     -> transcription for this part of the sentence. word
#                        tokens are separated by the character |
#
a01-000u-s00-00 0 ok 154 19 408 746 1661 89 A|MOVE|to|stop|Mr.|Gaitskell|from
a01-000u-s00-01 0 ok 156 19 395 932 1850 105 nominating|any|more|Labour|life|Peers
a01-000u-s00-02 0 ok 157 16 408 1106 1986 105 is|to|be|made|at|a|meeting|of|Labour
a01-000u-s00-03 0 err 156 9 430 1290 733 66 M Ps|tomorrow|.
a01-000u-s01-00 1 err 156 14 1269 1292 1044 68 Mr.|Michael|Foot|has
a01-000u-s01-01 1 ok 157 20 395 1474 1830 94 put|down|a|resolution|on|the|subject
a01-000u-s01-02 1 err 156 21 379 1643 1854 88 and|he|is|to|be|backed|by|Mr.|Will
a01-000u-s01-03 1 ok 159 20 363 1825 2051 87 Griffiths|,|M P|for|Manchester|Exchange|.
a01-000x-s00-00 0 ok 182 30 375 748 1561 148 A|MOVE|to|stop|Mr.|Gaitskell|from|nominating
a01-000x-s00-01 0 ok 181 23 382 924 1595 148 any|more|Labour|life|Peers|is|to|be|made|at|a
a01-000x-s00-02 0 ok 181 23 386 1110 1241 140 meeting|of|Labour|0M Ps|tomorrow|.
a01-000x-s01-00 1 ok 181 7 1689 1110 334 80 Mr.|Michael
a01-000x-s01-01 1 ok 179 22 375 1276 1584 154 Foot|has|put|down|a|resolution|on|the|subject
a01-000x-s01-02 1 ok 173 25 397 1458 1647 148 and|he|is|to|be|backed|by|Mr.|Will|Griffiths|,
a01-000x-s01-03 1 ok 173 16 393 1635 1082 155 0M P|for|Manchester|Exchange|.
a01-003-s00-00 0 ok 176 31 325 896 1807 139 Though|they|may|gather|some|Left-wing|support|,|a
a01-003-s00-01 0 ok 176 29 319 1078 1787 126 large|majority|of|Labour|M Ps|are|likely|to
a01-003-s00-02 0 ok 176 26 315 1259 1552 128 turn|down|the|Foot-Griffiths|resolution|.
a01-003-s01-00 1 ok 176 3 1935 1267 124 56 Mr.
a01-003-s01-01 1 ok 176 28 316 1441 1750 70 Foot's|line|will|be|that|as|Labour|M Ps
a01-003-s01-02 1 ok 161 28 325 1619 1848 113 opposed|the|Government|Bill|which|brought
a01-003-s01-03 1 ok 176 29 325 1779 1871 143 life|peers|into|existence|,|they|should|not
a01-003-s01-04 1 ok 176 19 326 1980 1297 126 now|put|forward|nominees|.
```

2. Building the model:

The next step is to design and train a machine-learning model using TensorFlow and CTC loss. This involves selecting an appropriate model architecture and choosing hyperparameters such as the learning rate and batch size. Here are a few recommendations when creating the model:

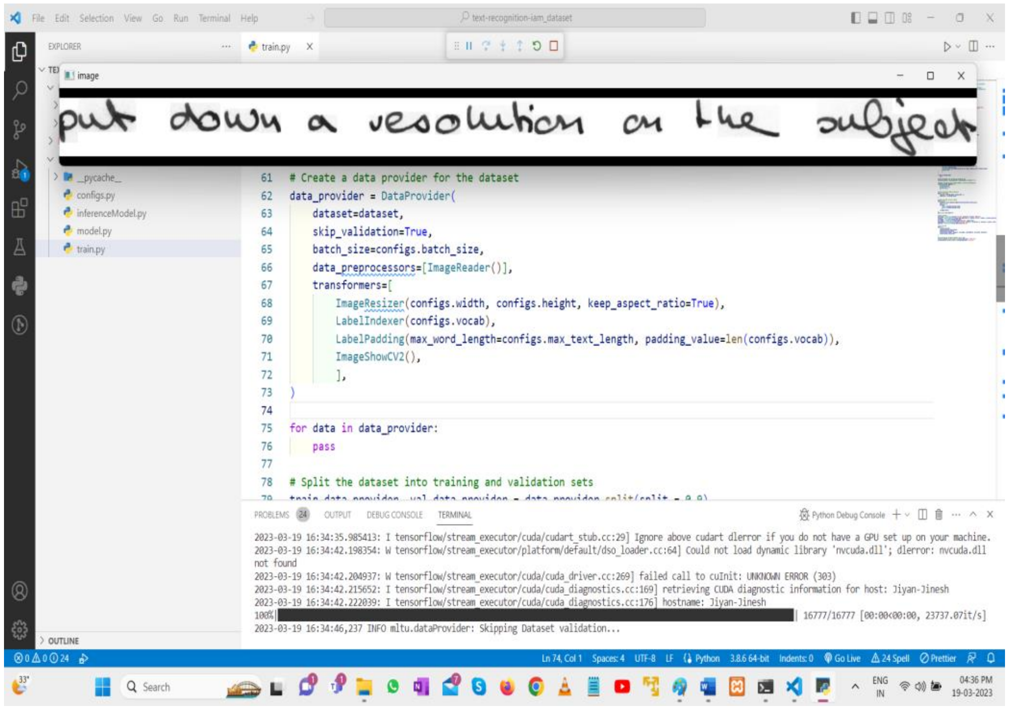
1. We define the input and output layers: The input layer of the model should be a 4D tensor with dimensions [batch size, width, height, channels], where the batch size is the number of images in a batch, width and height are the dimensions of the images, and channels is the number of color channels in the images (1 for grayscale, 3 for RGB);
2. Add the CNN layers: The CNN layers of the model should be responsible for extracting features from the images. A typical architecture for the CNN layers is to use a combination of convolutional, pooling, and fully-connected (dense) layers;
3. Add the RNN layers: The RNN layers of the model should be responsible for processing the sequence of features and predicting the characters in the text. A common type of RNN to use for this task is a long short-term memory (LSTM) network;
4. Finally, we compile the model: Once the CNN and RNN layers have been defined, the model can be compiled using the CTC loss function and an optimizer such as Adam.

some of the input displayed shown below:



```python
# Create a data provider for the dataset
data_provider = DataProvider(
    dataset=dataset,
    skip_validation=True,
    batch_size=configs.batch_size,
    data_preprocessors=[ImageReader()],
    transformers=[
        ImageResizer(configs.width, configs.height, keep_aspect_ratio=True),
        LabelIndexer(configs.vocab),
        LabelPadding(max_word_length=configs.max_text_length, padding_value=len(configs.vocab)),
        ImageShowCV2(),
        ],
)

for data in data_provider:
    pass

# Split the dataset into training and validation sets
```

```
2023-03-19 16:34:35.985413: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
2023-03-19 16:34:42.198354: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'nvcuda.dll'; dlerror: nvcuda.dll
not found
2023-03-19 16:34:42.204937: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to cuInit: UNKNOWN ERROR (303)
2023-03-19 16:34:42.215652: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diagnostic information for host: Jiyan-Jinesh
2023-03-19 16:34:42.222039: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: Jiyan-Jinesh
100%|████████████████████████████████████████| 16777/16777 [00:00<00:00, 23737.07it/s]
2023-03-19 16:34:46,237 INFO mltu.dataProvider: Skipping Dataset validation...
```



```python
# Create a data provider for the dataset
data_provider = DataProvider(
    dataset=dataset,
    skip_validation=True,
    batch_size=configs.batch_size,
    data_preprocessors=[ImageReader()],
    transformers=[
        ImageResizer(configs.width, configs.height, keep_aspect_ratio=True),
        LabelIndexer(configs.vocab),
        LabelPadding(max_word_length=configs.max_text_length, padding_value=len(configs.vocab)),
        ImageShowCV2(),
        ],
)

for data in data_provider:
    pass

# Split the dataset into training and validation sets
```

```
2023-03-19 16:34:35.985413: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
2023-03-19 16:34:42.198354: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'nvcuda.dll'; dlerror: nvcuda.dll
not found
2023-03-19 16:34:42.204937: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to cuInit: UNKNOWN ERROR (303)
2023-03-19 16:34:42.215652: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:169] retrieving CUDA diagnostic information for host: Jiyan-Jinesh
2023-03-19 16:34:42.222039: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:176] hostname: Jiyan-Jinesh
100%|████████████████████████████████████████| 16777/16777 [00:00<00:00, 23737.07it/s]
2023-03-19 16:34:46,237 INFO mltu.dataProvider: Skipping Dataset validation...
```

3. Training the model:

When we have our model, it can then be trained on the training set using an optimization algorithm, such as Adam. During training, the model will make predictions on the input data. Utilizing the CTC loss function, the model will update to reduce the disparity between the estimated values and the actual labels.

When training machine learning models, it's crucial to apply data augmentation techniques to achieve better training results.I am using RandomBrightness, RandomErodeDilate, and RandomSharpen augmentors to the training data provider object.

4. Evaluating the model:

The CER, or Character Error Rate, is a metric used to evaluate the performance of a handwritten sentence recognition model. It is calculated as the number of characters that are incorrectly recognized by the model, divided by the total number of characters in the ground truth text.

The WER, or Word Error Rate, is another metric used to evaluate the performance of a handwritten sentence recognition model. It is calculated as the number of words incorrectly recognized by the model divided by the total number of words in the ground truth text.

CER **and** WER metrics are primarily used metrics in OCR systems and help evaluate the accuracy of a model.

5.Testing the model

Once the model has been trained, it is ready to be tested on new data. To test the model,

1. Preprocess the test data: The test data should be preprocessed in the same way as the training data.
2. Evaluate the model on the test data: The model's performance on the test data can be evaluated using the CTC loss and any additional metrics that defined during training.

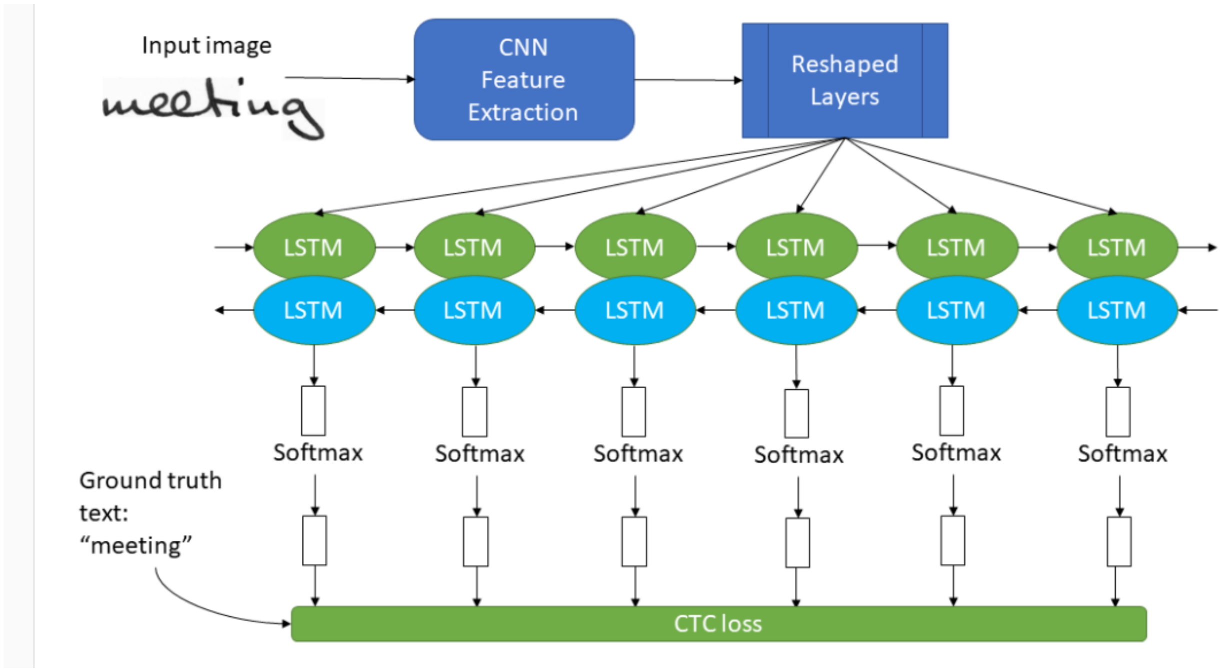
But there is no point testing it out while we already received CER and WER graphs above from training and evaluation processes. So let's run inference on an already trained model that was converter to `.ONNX` model.

Assumptions:

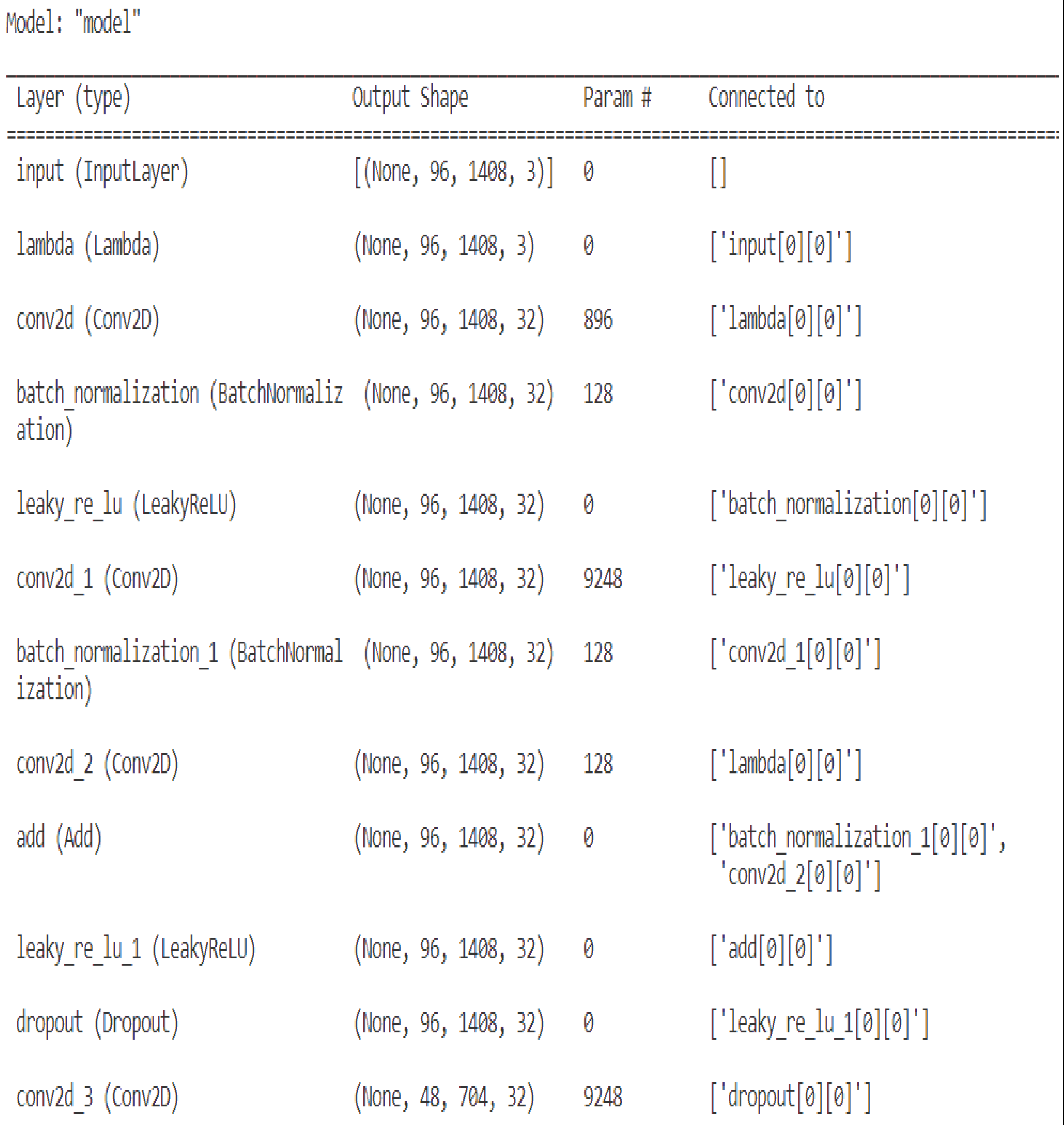
The assumptions considered as follows:

- All machine dependencies must be installed properly
- The handwritten text across the image must be in English.
- Only image is provided for text recognition.
- Clarity of image depends the accuracy.
- More the number of images for training, leads to better prediction

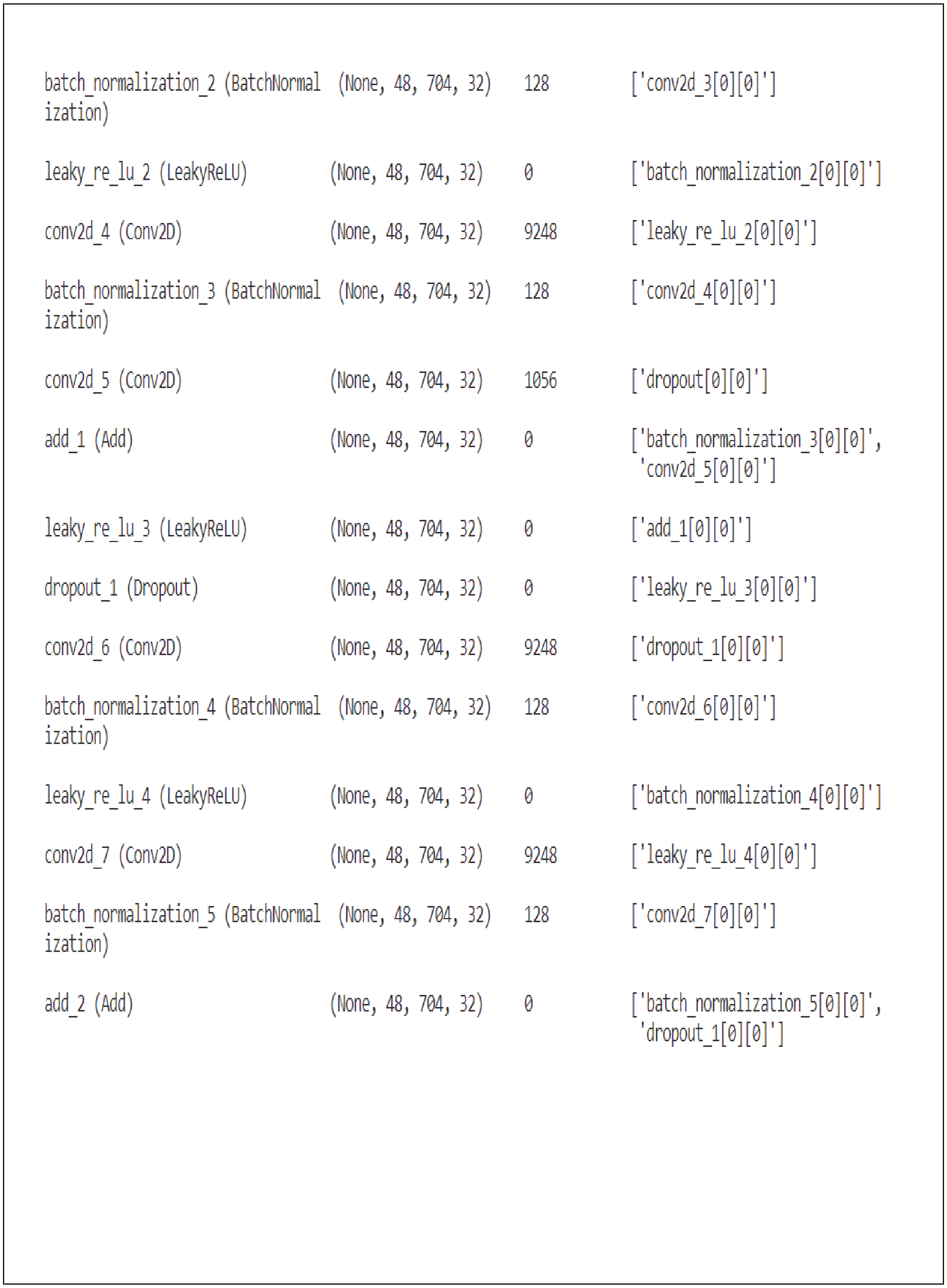
Project Diagrams:



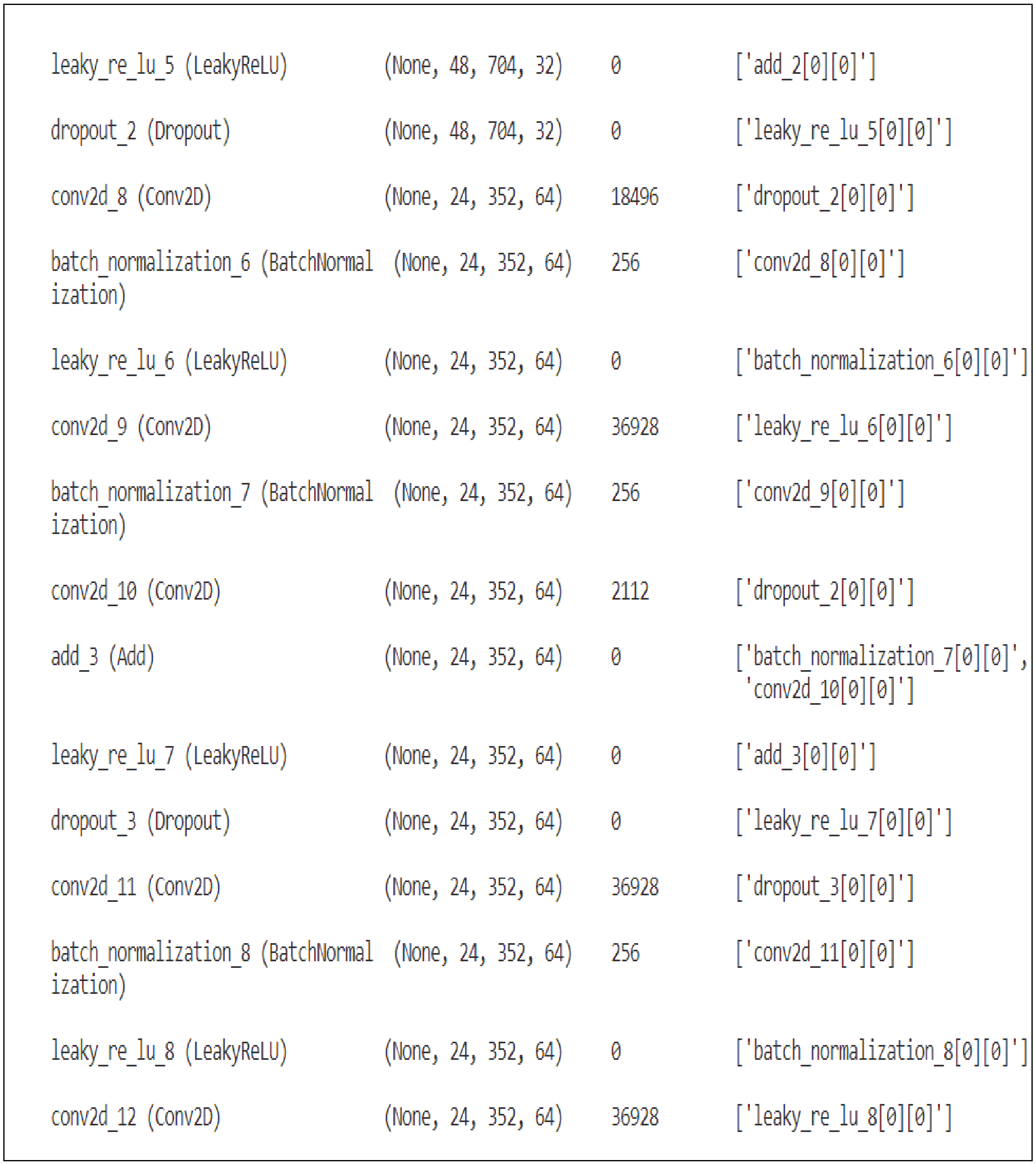
Algorithms:



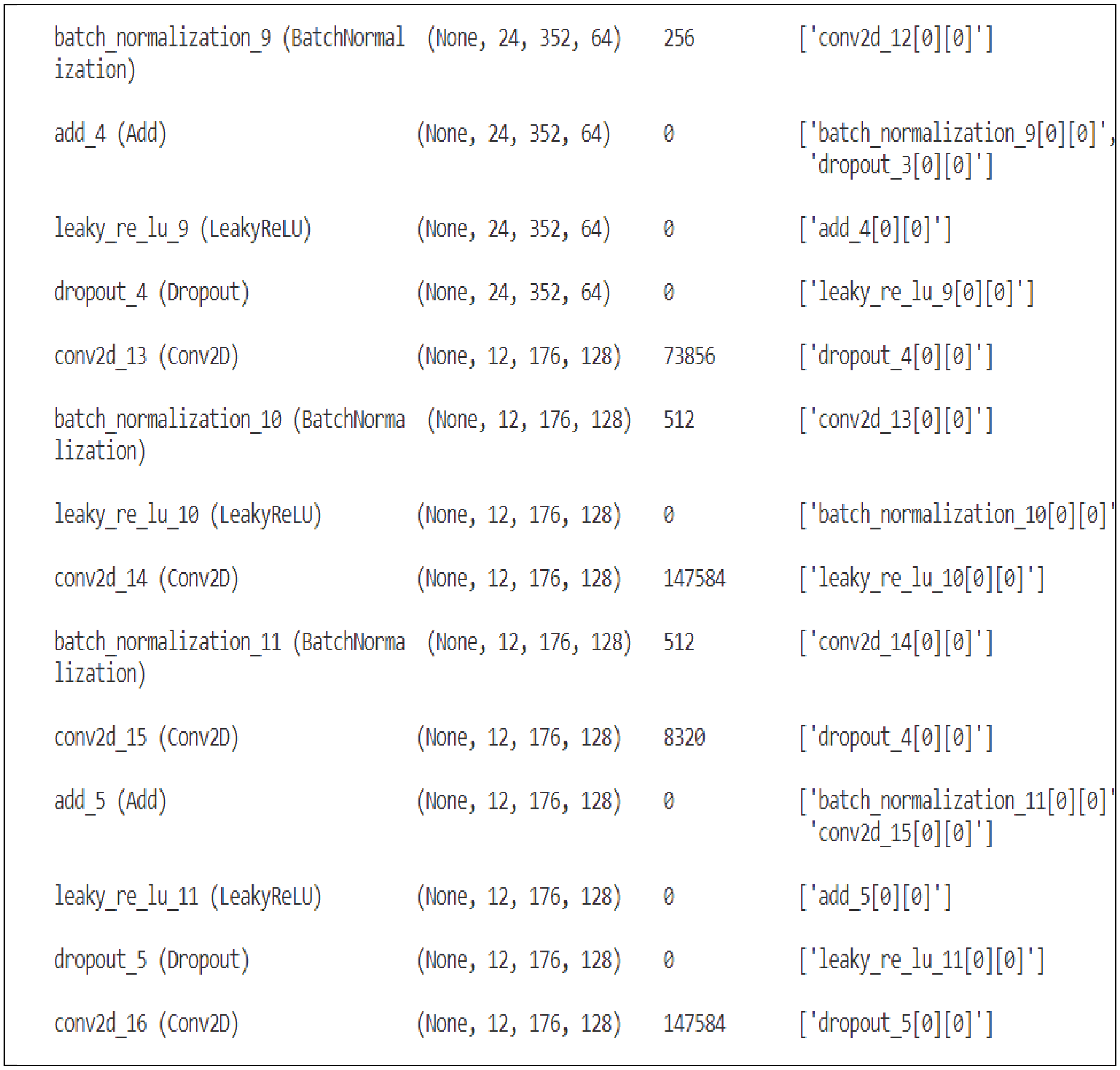
```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape          Param #     Connected to
==================================================================================================
 input (InputLayer)             [(None, 96, 1408, 3)]  0          []

 lambda (Lambda)                (None, 96, 1408, 3)   0           ['input[0][0]']

 conv2d (Conv2D)                (None, 96, 1408, 32)  896         ['lambda[0][0]']

 batch_normalization (BatchNormaliz  (None, 96, 1408, 32)  128    ['conv2d[0][0]']
 ation)

 leaky_re_lu (LeakyReLU)        (None, 96, 1408, 32)  0           ['batch_normalization[0][0]']

 conv2d_1 (Conv2D)              (None, 96, 1408, 32)  9248        ['leaky_re_lu[0][0]']

 batch_normalization_1 (BatchNormal  (None, 96, 1408, 32)  128    ['conv2d_1[0][0]']
 ization)

 conv2d_2 (Conv2D)              (None, 96, 1408, 32)  128         ['lambda[0][0]']

 add (Add)                      (None, 96, 1408, 32)  0           ['batch_normalization_1[0][0]',
                                                                   'conv2d_2[0][0]']

 leaky_re_lu_1 (LeakyReLU)      (None, 96, 1408, 32)  0           ['add[0][0]']

 dropout (Dropout)              (None, 96, 1408, 32)  0           ['leaky_re_lu_1[0][0]']

 conv2d_3 (Conv2D)              (None, 48, 704, 32)   9248        ['dropout[0][0]']
```



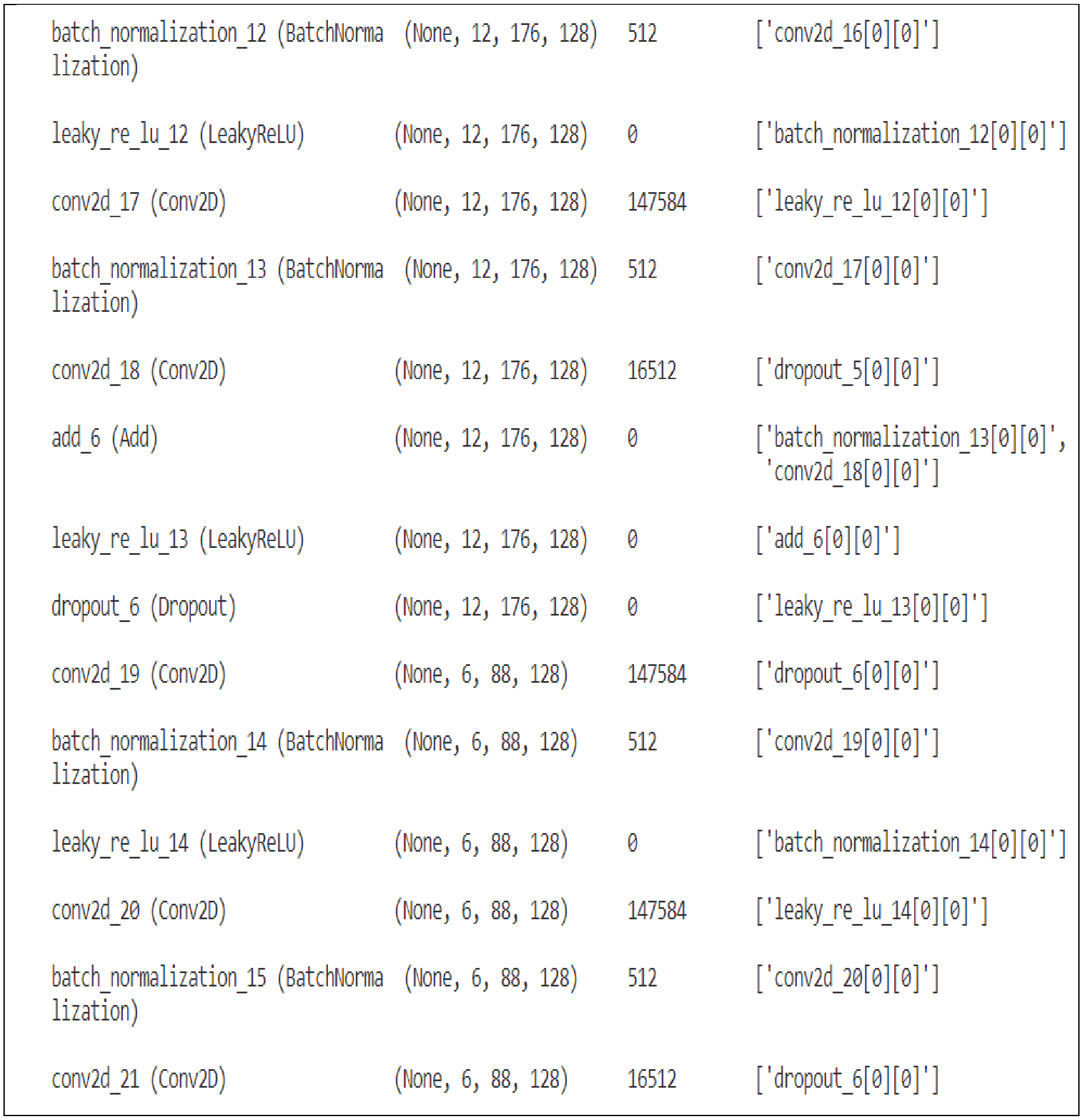
```
batch_normalization_2 (BatchNormal  (None, 48, 704, 32)   128       ['conv2d_3[0][0]']
ization)

leaky_re_lu_2 (LeakyReLU)          (None, 48, 704, 32)   0         ['batch_normalization_2[0][0]']

conv2d_4 (Conv2D)                  (None, 48, 704, 32)   9248      ['leaky_re_lu_2[0][0]']

batch_normalization_3 (BatchNormal  (None, 48, 704, 32)   128       ['conv2d_4[0][0]']
ization)

conv2d_5 (Conv2D)                  (None, 48, 704, 32)   1056      ['dropout[0][0]']

add_1 (Add)                        (None, 48, 704, 32)   0         ['batch_normalization_3[0][0]',
                                                                    'conv2d_5[0][0]']

leaky_re_lu_3 (LeakyReLU)          (None, 48, 704, 32)   0         ['add_1[0][0]']

dropout_1 (Dropout)                (None, 48, 704, 32)   0         ['leaky_re_lu_3[0][0]']

conv2d_6 (Conv2D)                  (None, 48, 704, 32)   9248      ['dropout_1[0][0]']

batch_normalization_4 (BatchNormal  (None, 48, 704, 32)   128       ['conv2d_6[0][0]']
ization)

leaky_re_lu_4 (LeakyReLU)          (None, 48, 704, 32)   0         ['batch_normalization_4[0][0]']

conv2d_7 (Conv2D)                  (None, 48, 704, 32)   9248      ['leaky_re_lu_4[0][0]']

batch_normalization_5 (BatchNormal  (None, 48, 704, 32)   128       ['conv2d_7[0][0]']
ization)

add_2 (Add)                        (None, 48, 704, 32)   0         ['batch_normalization_5[0][0]',
                                                                    'dropout_1[0][0]']
```



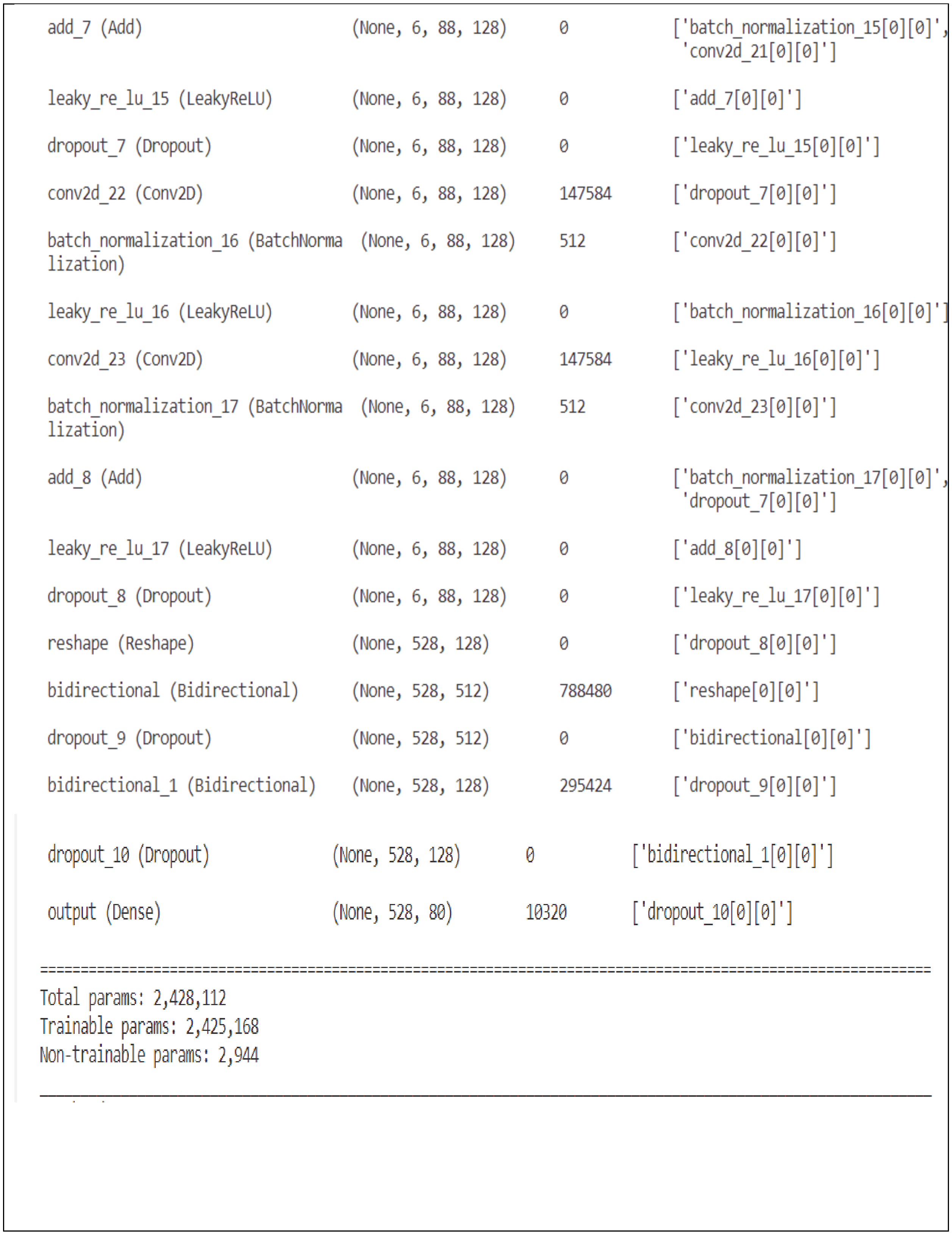
```
leaky_re_lu_5 (LeakyReLU)         (None, 48, 704, 32)   0        ['add_2[0][0]']

dropout_2 (Dropout)               (None, 48, 704, 32)   0        ['leaky_re_lu_5[0][0]']

conv2d_8 (Conv2D)                 (None, 24, 352, 64)   18496    ['dropout_2[0][0]']

batch_normalization_6 (BatchNormal  (None, 24, 352, 64)  256     ['conv2d_8[0][0]']
ization)

leaky_re_lu_6 (LeakyReLU)         (None, 24, 352, 64)   0        ['batch_normalization_6[0][0]']

conv2d_9 (Conv2D)                 (None, 24, 352, 64)   36928    ['leaky_re_lu_6[0][0]']

batch_normalization_7 (BatchNormal  (None, 24, 352, 64)  256     ['conv2d_9[0][0]']
ization)

conv2d_10 (Conv2D)                (None, 24, 352, 64)   2112     ['dropout_2[0][0]']

add_3 (Add)                       (None, 24, 352, 64)   0        ['batch_normalization_7[0][0]',
                                                                  'conv2d_10[0][0]']

leaky_re_lu_7 (LeakyReLU)         (None, 24, 352, 64)   0        ['add_3[0][0]']

dropout_3 (Dropout)               (None, 24, 352, 64)   0        ['leaky_re_lu_7[0][0]']

conv2d_11 (Conv2D)                (None, 24, 352, 64)   36928    ['dropout_3[0][0]']

batch_normalization_8 (BatchNormal  (None, 24, 352, 64)  256     ['conv2d_11[0][0]']
ization)

leaky_re_lu_8 (LeakyReLU)         (None, 24, 352, 64)   0        ['batch_normalization_8[0][0]']

conv2d_12 (Conv2D)                (None, 24, 352, 64)   36928    ['leaky_re_lu_8[0][0]']
```



```
 batch_normalization_9 (BatchNormal  (None, 24, 352, 64)   256         ['conv2d_12[0][0]']
 ization)

 add_4 (Add)                        (None, 24, 352, 64)   0           ['batch_normalization_9[0][0]',
                                                                        'dropout_3[0][0]']

 leaky_re_lu_9 (LeakyReLU)          (None, 24, 352, 64)   0           ['add_4[0][0]']

 dropout_4 (Dropout)                (None, 24, 352, 64)   0           ['leaky_re_lu_9[0][0]']

 conv2d_13 (Conv2D)                 (None, 12, 176, 128)  73856       ['dropout_4[0][0]']

 batch_normalization_10 (BatchNorma  (None, 12, 176, 128)  512         ['conv2d_13[0][0]']
 lization)

 leaky_re_lu_10 (LeakyReLU)         (None, 12, 176, 128)  0           ['batch_normalization_10[0][0]'

 conv2d_14 (Conv2D)                 (None, 12, 176, 128)  147584      ['leaky_re_lu_10[0][0]']

 batch_normalization_11 (BatchNorma  (None, 12, 176, 128)  512         ['conv2d_14[0][0]']
 lization)

 conv2d_15 (Conv2D)                 (None, 12, 176, 128)  8320        ['dropout_4[0][0]']

 add_5 (Add)                        (None, 12, 176, 128)  0           ['batch_normalization_11[0][0]'
                                                                        'conv2d_15[0][0]']

 leaky_re_lu_11 (LeakyReLU)         (None, 12, 176, 128)  0           ['add_5[0][0]']

 dropout_5 (Dropout)                (None, 12, 176, 128)  0           ['leaky_re_lu_11[0][0]']

 conv2d_16 (Conv2D)                 (None, 12, 176, 128)  147584      ['dropout_5[0][0]']
```



```
 batch_normalization_12 (BatchNorma  (None, 12, 176, 128)   512         ['conv2d_16[0][0]']
 lization)

 leaky_re_lu_12 (LeakyReLU)         (None, 12, 176, 128)   0           ['batch_normalization_12[0][0]']

 conv2d_17 (Conv2D)                 (None, 12, 176, 128)   147584      ['leaky_re_lu_12[0][0]']

 batch_normalization_13 (BatchNorma  (None, 12, 176, 128)   512         ['conv2d_17[0][0]']
 lization)

 conv2d_18 (Conv2D)                 (None, 12, 176, 128)   16512       ['dropout_5[0][0]']

 add_6 (Add)                        (None, 12, 176, 128)   0           ['batch_normalization_13[0][0]',
                                                                        'conv2d_18[0][0]']

 leaky_re_lu_13 (LeakyReLU)         (None, 12, 176, 128)   0           ['add_6[0][0]']

 dropout_6 (Dropout)                (None, 12, 176, 128)   0           ['leaky_re_lu_13[0][0]']

 conv2d_19 (Conv2D)                 (None, 6, 88, 128)     147584      ['dropout_6[0][0]']

 batch_normalization_14 (BatchNorma  (None, 6, 88, 128)     512         ['conv2d_19[0][0]']
 lization)

 leaky_re_lu_14 (LeakyReLU)         (None, 6, 88, 128)     0           ['batch_normalization_14[0][0]']

 conv2d_20 (Conv2D)                 (None, 6, 88, 128)     147584      ['leaky_re_lu_14[0][0]']

 batch_normalization_15 (BatchNorma  (None, 6, 88, 128)     512         ['conv2d_20[0][0]']
 lization)

 conv2d_21 (Conv2D)                 (None, 6, 88, 128)     16512       ['dropout_6[0][0]']
```



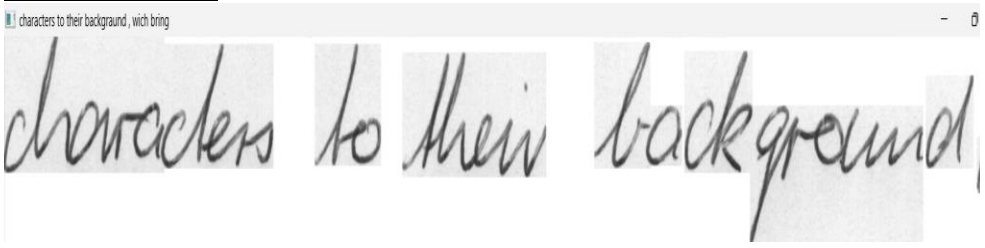
```
 add_7 (Add)                     (None, 6, 88, 128)    0           ['batch_normalization_15[0][0]',
                                                                    'conv2d_21[0][0]']

 leaky_re_lu_15 (LeakyReLU)      (None, 6, 88, 128)    0           ['add_7[0][0]']

 dropout_7 (Dropout)             (None, 6, 88, 128)    0           ['leaky_re_lu_15[0][0]']

 conv2d_22 (Conv2D)              (None, 6, 88, 128)    147584      ['dropout_7[0][0]']

 batch_normalization_16 (BatchNorma  (None, 6, 88, 128)  512       ['conv2d_22[0][0]']
 lization)

 leaky_re_lu_16 (LeakyReLU)      (None, 6, 88, 128)    0           ['batch_normalization_16[0][0]']

 conv2d_23 (Conv2D)              (None, 6, 88, 128)    147584      ['leaky_re_lu_16[0][0]']

 batch_normalization_17 (BatchNorma  (None, 6, 88, 128)  512       ['conv2d_23[0][0]']
 lization)

 add_8 (Add)                     (None, 6, 88, 128)    0           ['batch_normalization_17[0][0]',
                                                                    'dropout_7[0][0]']

 leaky_re_lu_17 (LeakyReLU)      (None, 6, 88, 128)    0           ['add_8[0][0]']

 dropout_8 (Dropout)             (None, 6, 88, 128)    0           ['leaky_re_lu_17[0][0]']

 reshape (Reshape)               (None, 528, 128)      0           ['dropout_8[0][0]']

 bidirectional (Bidirectional)   (None, 528, 512)      788480      ['reshape[0][0]']

 dropout_9 (Dropout)             (None, 528, 512)      0           ['bidirectional[0][0]']

 bidirectional_1 (Bidirectional) (None, 528, 128)      295424      ['dropout_9[0][0]']

 dropout_10 (Dropout)            (None, 528, 128)      0           ['bidirectional_1[0][0]']

 output (Dense)                  (None, 528, 80)       10320       ['dropout_10[0][0]']

==================================================================================================
Total params: 2,428,112
Trainable params: 2,425,168
Non-trainable params: 2,944
__________________________________________________________________________________________________
```

1. We define the input and output layers: The input layer of the model should be a 4D tensor with dimensions [batch size, width, height, channels], where the batch size is the number of images in a batch, width and height are the dimensions of the images, and channels is the number of color channels in the images (1 for grayscale, 3 for RGB);
2. Add the CNN layers: The CNN layers of the model should be responsible for extracting features from the images. A typical architecture for the CNN layers is to use a combination of convolutional, pooling, and fully-connected (dense) layers;
3. Add the RNN layers: The RNN layers of the model should be responsible for processing the sequence of features and predicting the characters in the text. A common type of RNN to use for this task is a long short-term memory (LSTM) network;
4. Finally, we compile the model: Once the CNN and RNN layers have been defined, the model can be compiled using the CTC loss function and an optimizer such as Adam.

Outcome:

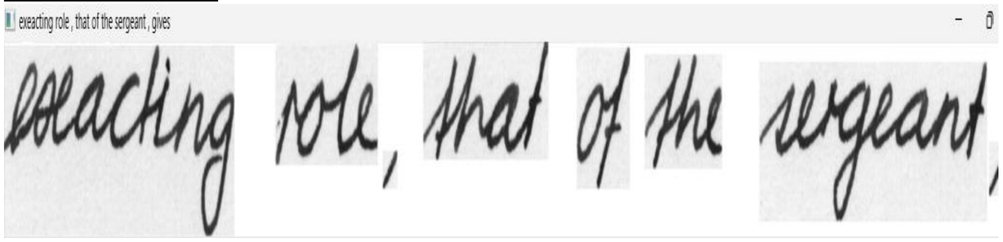
The algorithm is able to detect and segment handwritten text from an image

Extracted output:


characters to their backgraund , wich bring
characters to their background,

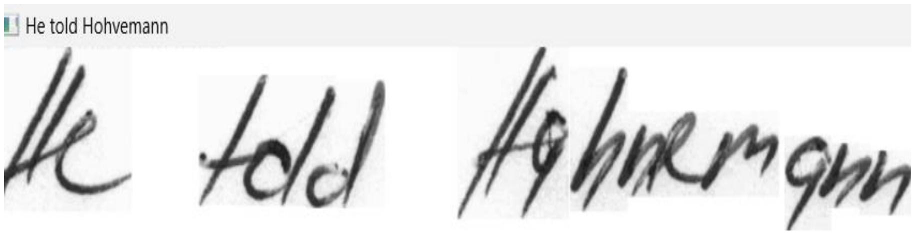
Label: characters to their background , which bring
Prediction: characters to their backgraund , wich bring
CER: 0.045454545454545456; WER: 0.2857142857142857

Extracted output:



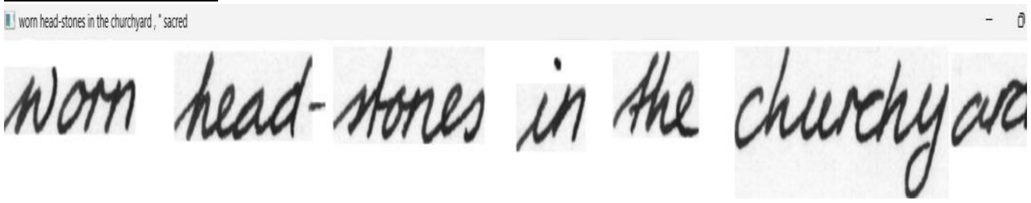
Label: exacting role , that of the sergeant , gives
Prediction: exeacting role , that of the sergeant , gives
CER: 0.022727272727272728; WER: 0.1111111111111111

Extracted output:



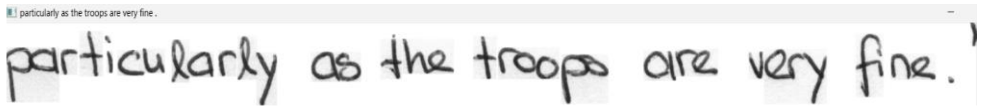
Label: He told Hahnemann
Prediction: He told Hohvemann
CER: 0.11764705882352941; WER: 0.3333333333333333

Extracted output:



Label: worn head-stones in the churchyard , " sacred
Prediction: worn head-stones in the churchyard , " sacred
CER: 0.0; WER: 0.0

Extracted output:



Label: particularly as the troops are very fine . '
Prediction: particularly as the troops are very fine .
CER: 0.022727272727272728; WER: 0.1111111111111111

The model successfully extracted the handwritten text from the given image.

Exceptions considered:

1) The text across the input image must be of the same color not multicolor handwritten text.
2) The image doesn't have any kind's objects in the background across the text of the image.

Enhancement Scope:

The enhancement scope of this project are follows:

1) The accuracy of the model can increase with predefined models and powerful machine learning GPU processors can be used to attain a good percentage of accuracy.
2) In future we can use this algorithm with more than one particular language.
3) This Model can be used in extraction of text from video

References:

[1] https://arxiv.org/pdf/1507.05717.pdf

[2] https://software.intel.com/content/www/us/en/develop/training/course-artificial-intelligence.html

[3] https://software.intel.com/content/www/us/en/develop/training/course-machine-learning.html [4] https://www.python-course.eu/machine_learning.php

[5] https://numpy.org/doc/

[6] https://software.intel.com/en-us/ai/courses/deep-learning

[7] https://www.tensorflow.org/tutorials/images/classification
https://www.tensorflow.org/tutorials/text/text_classification_rnn

Link to Code and executable file:

- I coded it in VSC.All code files are uploaded in the following github link.Dataset and trained Model folder are large in size.So these are included in the following google drive

GitHub link: https://github.com/PRESISPREMACHANDRAN/ml-ai_internship_tcsion

Google drive link: **https://drive.google.com/drive/folders/10-hHC5BVxU17Ejo-WCc5K9yIG2b-AlMD?usp=sharing**

- **All other successful codes are also included in the above github link**