UNIVERSIDAD POLITÉCNICA DE YUCATÁN

Machine Learning

Final Project

Supervised Learning solution

Team:

- Ari Isaías Quintal Várguez
- Amaury Castellanos Palomo
- Jorge Carlos Canul Cal
- Pablo Iván Martin Enríquez

16/11/2023

Introduction.

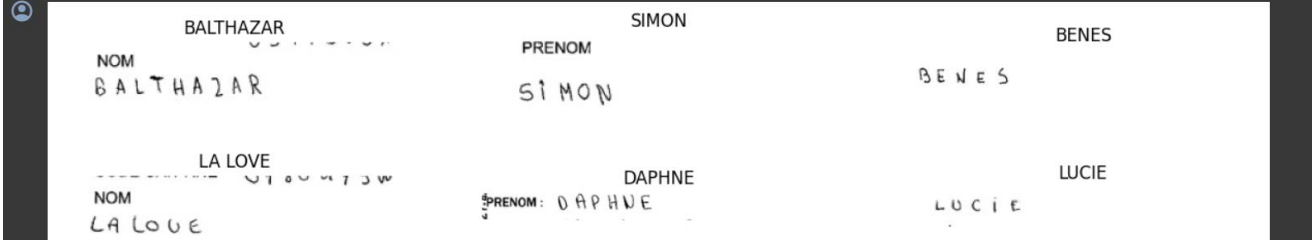
In the initial phase of this project, the focus is on presenting a significant advancement, specifically a supervised learning model, as part of the first deliverable. The success of this progress will be assessed based on the completeness and thoroughness of the documentation. The documentation is pivotal as it is expected to provide a comprehensive understanding of the developed model and its training process.

The supervised learning model implies a structured approach where the algorithm learns patterns from labeled data. The main expectation is that this model will contribute significantly to the project's overarching goal of optimizing baggage transport, a critical aspect affecting both passengers and airline operations.

- **Documentation of the code created to train the model.**

Initially the problem statement was to reach efficiency at airports for the better experience of passengers and airline operations since one of the most significant challenges in the context is baggage transport which often can be a slow and error-prone process so by providing machine learning techniques it is expected to improve the process.

```
plt.figure(figsize=(15, 10))
for i in range(6):
    ax = plt.subplot(2, 3, i+1)
    img_dir = '/content/train_v2/train/' + train.loc[i, 'FILENAME']
    image = cv2.imread(img_dir, cv2.IMREAD_GRAYSCALE)
    plt.imshow(image, cmap='gray')
    plt.title(train.loc[i, 'IDENTITY'], fontsize=12)
    plt.axis('off')
plt.subplots_adjust(wspace=0.2, hspace=-0.8)
```



```
[ ]  train.dropna(axis=0, inplace=True)
     valid.dropna(axis=0, inplace=True)

     train_csv = train[train['IDENTITY'] != 'UNREADABLE']
     val_csv = valid[valid['IDENTITY'] != 'UNREADABLE']
     train = train[train['IDENTITY'] != 'UNREADABLE']
     valid = valid[valid['IDENTITY'] != 'UNREADABLE']
     train['IDENTITY'] = train['IDENTITY'].str.upper()
     valid['IDENTITY'] = valid['IDENTITY'].str.upper()
     train.reset_index(inplace = True, drop=True)
     valid.reset_index(inplace = True, drop=True)


     print(train_csv.shape[0], val_csv.shape[0])
```

```
Number of NaNs in train set      :  565
Number of NaNs in validation set :  78
330294 41280
<ipython-input-10-ace5b7f81ff7>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  train['IDENTITY'] = train['IDENTITY'].str.upper()
<ipython-input-10-ace5b7f81ff7>:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  valid['IDENTITY'] = valid['IDENTITY'].str.upper()
```

The first solution proposal recommended to search for datasets that contain country tag identification yet during the implementation was decided to change the country tag identification for handwritten airports ID that essentially contains the purpose of the project but with a different perspective since working with images with colors results to be complicated to train and test considering the limited time at hand to finish the model besides the other two learning solutions (unsupervised and reinforcement learning).

Before training the model, it was necessary to pre-process the information in the dataset retrieved from Kaggle with the name of *Worlds Airports and Airlines Datasets*, that in simple words compiles IDs of airlines, airlines ID, destination airport, destination airport ID, etc. handwritten instead of color images that become complicated the training and validation of the model. Once that explained, the actual code import libraries for additional computer task such as Keras-CV or TensorFlow Datasets libraries to simplify the process of downloading and preparing the data necessary for the project. Then, as shown in the picture above, there is tested a random input to validate the model and observe the initial behavior for further adjustments. This is a visual exploration of the handwriting images in the training dataset. It displays a grid of 6 images along with their corresponding identities.

The pre-process shows how many NaN values in the *IDENTITY* colum are for both the training and validation sets, but it only removes rows with NaN values, yet the code

creates new DataFrames by excluding rows where the identity is labeled as *UNREADABLE*. Additionally, it updates the original DataFrames to remove these entries.

The model defines a convolutional neural network (CNN) followed by a bidirectional long short-term memory network (Bi-LSTM) for handwriting recognition. Additionally, it sets up the CTC (Connectionist Temporal Classification) loss function. The model is designed for end-to-end handwriting recognition, where the model learns to recognize sequences of characters from input images using the CTC loss for sequence labeling tasks. The Bidirectional LSTM layers are crucial for capturing both past and future context in the input sequences. The CTC loss helps handle variable-length output sequences during training.

- **Documentation of the code used to evaluate the model.**

```
# Third Convolutional Layer
inner = Conv2D(128, (3, 3), padding='same', name='conv3', kernel_initializer='he_normal')(inner)
inner = BatchNormalization()(inner)
inner = Activation('relu')(inner)
inner = MaxPooling2D(pool_size=(1, 2), name='max3')(inner)
inner = Dropout(0.3)(inner)

# Reshape the output for CNN-to-RNN transition
inner = Reshape(target_shape=((64, 1024)), name='reshape')(inner)
inner = Dense(64, activation='relu', kernel_initializer='he_normal', name='dense1')(inner)

# Bidirectional LSTM Layers
inner = Bidirectional(LSTM(256, return_sequences=True), name='lstm1')(inner)
inner = Bidirectional(LSTM(256, return_sequences=True), name='lstm2')(inner)

# Output Layer
inner = Dense(num_of_characters, kernel_initializer='he_normal', name='dense2')(inner)
y_pred = Activation('softmax', name='softmax')(inner)

model = Model(inputs=input_data, outputs=y_pred)
```

```
def ctc_lambda_func(args):
    y_pred, labels, input_length, label_length = args
    y_pred = y_pred[:, 2:, :]
    loss = tf.keras.backend.ctc_batch_cost(labels, y_pred, input_length, label_length)

    return loss
```

Training the model.

The model is trained on a labeled dataset. Labeled data means that for each input, the corresponding correct output is provided, allowing the model to learn the mapping between inputs and outputs.

```
[ ] textReader.compile(loss={'ctc': lambda y_true, y_pred: y_pred}, optimizer=Adam(lr=0.0001))

    textReader.fit(
        x=[train_x, train_y, train_input_len, train_label_len],
        y=train_output,
        validation_data=([valid_x, valid_y, valid_input_len, valid_label_len], valid_output),
        epochs=60,
        batch_size=128
    )
    model.save("handwriting_recognition_model")
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
Epoch 1/60
24/24 [==============================] - 246s 10s/step - loss: 34.3736 - val_loss: 21.6412
Epoch 2/60
24/24 [==============================] - 228s 9s/step - loss: 20.9872 - val_loss: 21.2514
Epoch 3/60
24/24 [==============================] - 228s 9s/step - loss: 20.7146 - val_loss: 21.0826
Epoch 4/60
24/24 [==============================] - 240s 10s/step - loss: 20.4520 - val_loss: 21.0547
Epoch 5/60
24/24 [==============================] - 230s 10s/step - loss: 20.2855 - val_loss: 20.9093
Epoch 6/60
24/24 [==============================] - 230s 10s/step - loss: 20.1728 - val_loss: 20.6252
Epoch 7/60
24/24 [==============================] - 235s 10s/step - loss: 20.0443 - val_loss: 20.5860
Epoch 8/60
24/24 [==============================] - 229s 10s/step - loss: 19.9312 - val_loss: 20.5010
Epoch 9/60
24/24 [==============================] - 237s 10s/step - loss: 19.8029 - val_loss: 20.6698
Epoch 10/60
```

- **Document the steps followed to generate the solution and how it fits into the final solution.**

This code implements a Convolutional Recurrent Neural Network (CRNN) for handwriting recognition, utilizing the Connectionist Temporal Classification (CTC) loss. The process begins with image preprocessing, including reading, decoding, and resizing to a standardized format. Data is loaded from a CSV file, where text labels are encoded for training. A TensorFlow dataset is created to efficiently handle the input data. The pre-trained CRNN model is loaded, compiled with the CTC loss, and then applied to an unseen test dataset. The results are visualized through a function displaying predicted texts alongside ground truth for a subset of images. Overall, the code provides a comprehensive solution for handwriting recognition, combining effective data preprocessing, model training, and insightful result visualization. The code follows a handwriting recognition workflow, starting with preprocessing images by reading, decoding, and resizing to a standardized format. Data, including image filenames and corresponding labels, is loaded from a CSV file, and text labels are encoded. A TensorFlow dataset is then created to efficiently handle the input data. The pre-trained CRNN model is loaded, compiled with CTC loss, and applied to an unseen test dataset. The code concludes by visualizing the model's predictions alongside ground truth for a subset of images, providing a comprehensive solution for handwriting recognition.

The loss function In this code measures how far the model’s guesses are from the correct answers. The main goal during training is to make these guesses as accurate as possible.

Here, the code deals with recognizing handwriting using a method called Connectionist Temporal Classification (CTC). This method is useful when we don’t know the exact alignment between what the model sees and what it should predict.

The code sets up a model, a kind of computer brain, for this task. It uses a mix of different types of neural networks to process images and figure out which letters or characters are in them.

The CTC loss part calculates how much the model’s guesses differ from the real answers. The final “TextReader” model is then trained to get better at predicting by adjusting its

guesses based on the differences between its predictions and the correct answers. The aim is to minimize these differences over multiple rounds of training.

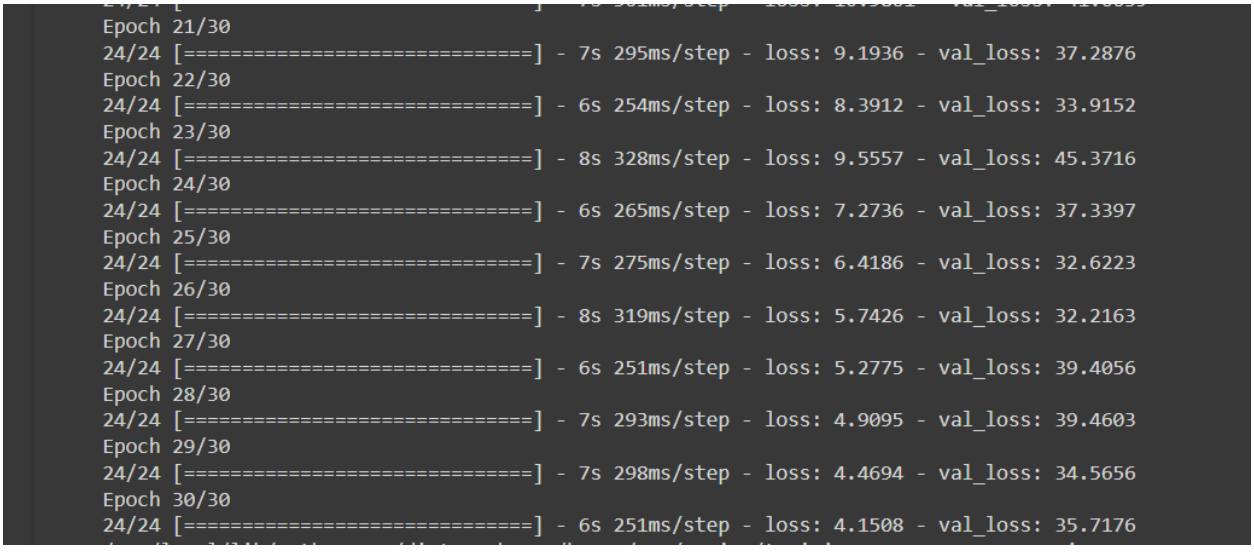
- **Deviations between planned actions and those actually executed.**

During the training process on Google Colab, challenges arose due to a notable deficiency in available RAM. This deficiency became particularly evident as the training progressed, leading to recurrent memory exhaustion issues that disrupted the continuity of the training workflow. Furthermore, it was observed that the Colab environment consistently terminated after surpassing the predefined four-hour training duration limit.

To counteract these persistent challenges, a strategic approach was employed to alleviate the strain on system resources. A key adjustment involved the resizing of images within the training dataset. By reducing the dimensions of these images, the aim was to mitigate the memory demands incurred during training sessions, with the expectation that this would foster a more stable and sustained training process.

This optimization initiative was part of a systematic effort to strike a balance between effective model training and the inherent constraints posed by the Colab environment. The goal was to ensure a smoother and uninterrupted process for experimentation and development.

- **Evidence and obtained results**



- **Conclusions**

As of right now, the has been a good amount of progress, solid results and we have learned a lot on how these techniques work, there is no doubt about the complexity, it takes a considerable amount of time, especially when the platforms show their limits, such as long waiting queues because of some complex procedures or finding errors.

During the training process on Google Colab, challenges emerged due to RAM limitations and duration constraints. To address these issues, a strategic resizing of images was implemented to mitigate memory demands and ensure a stable training process. Despite these deviations from the initial plan, the project demonstrated adaptability and a systematic approach to overcoming obstacles.

As of right now, we have finished one third of the project as a whole, but this first part has reassured us that we have the ability to finish everything as long as time is in our favor, otherwise it will be a hassle to finish up everything on time.