

# Development of a Machine Learning Robot for Baggage Transport Optimization at Airports.

*Ari Isaías Quintal Vázquez - Amaury D. Castellanos Palomo - Jorge Carlos Canul Cal - Pablo Iván Martín Enríquez*

**Mr. Victor Alejandro Ortiz Santiago**

**Abstract** - This final project explores key concepts in machine learning: supervised, unsupervised, and reinforcement learning. We focus on these three learning models, utilizing KMeans clustering and Principal Component Analysis (PCA) to optimize baggage transportation in airports. The project documents the code creation and evaluation processes, highlighting deviations from planned actions. We reflect on the complexity of machine learning models and the valuable insights gained through direct interaction, offering potential for future projects.

**Index Terms:** KMeans, Clustering, Learning Models, SKLearn

## I. INTRODUCTION

IT is explored three core concepts in machine learning: supervised learning, unsupervised learning, and reinforcement learning. We will have detailed information and applications of all three.

These three are very easy to describe, beginning with supervised learning, which involves training models on labeled data for making predictions, unsupervised learning focuses on finding patterns in unlabeled data, and finishing with reinforcement learning, which guides models in making sequential decisions based on feedback from the environment.

Supervised learning, a subset of machine learning and artificial intelligence, relies on labeled datasets to train algorithms for accurate data classification or outcome prediction. The primary goal of supervised learning is to enable the algorithm to learn the mapping between inputs and outputs, allowing it to make accurate predictions or classifications when presented with new, unseen data. [1]

Unsupervised machine learning is considered a type of learning that learns from the data without human supervision as compared to supervised machine learning. For that reason, unsupervised machine learning models are given unlabeled data and allowed to discover patterns and insights without any explicit guidance or instruction [2].

Reinforcement learning is a type of machine learning where an agent learns to make decisions by interacting with an environment. The agent receives feedback in the form of rewards or punishments, and its goal is to learn a policy that maximizes cumulative rewards over time. This approach is

useful in scenarios where explicit training data is not available, and the agent must learn through trial and error. Key components include the agent, environment, state, action, reward, and policy. If "reinforced machine learning" refers to a specific term or concept, it's advisable to check more recent sources for the latest information.[3]

The best situations that unsupervised models can solve are complex processing task such as organizing large datasets into clusters, identifying undetected patterns in the process. Considering the approach and the complicated baggage movement in airports it is proposed a model capable of clustering three main bags found: luggage, handbag, and backpacks, tending the first one to be put as documented baggage. That way can not only be optimized transportation routes but also passengers' movements across the place.

## II. DEVELOPMENT

### A. Problem and solution proposal

In an increasingly connected and globalized world, efficiency at airports plays a crucial role in the travel experience of passengers and airline operations. One of the most significant challenges in this context is baggage transport, which often can be a slow and error-prone process.

The loss or delay in baggage delivery can have a negative impact on both customer satisfaction and airport operations. This proposal aims to address this challenge by developing a machine learning robot that optimizes baggage transport at airports. This robot will be capable of using advanced machine learning techniques such as country tag identification, baggage detection, and route optimization to enhance efficiency and accuracy in baggage delivery.

#### 1) Objectives

The primary objectives of this project are as follows: Supervised, Unsupervised and Transformative.

1. Develop a machine learning system capable of accurately and swiftly identifying and reading country tags on baggage.
2. Implement a baggage detection algorithm that efficiently locates and tracks each piece of luggage.
3. Design a route optimization system that determines the most efficient route for transporting baggage from the aircraft to the baggage claim area.

### III. METHODOLOGY

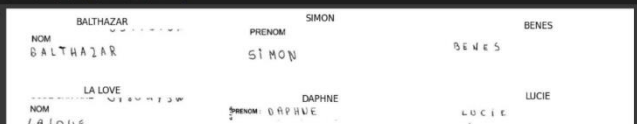
To achieve the proposed objectives, the following methodology will be followed:

1. Country Tag Identification: Supervised machine learning techniques will be employed to train a model capable of recognizing country tags on baggage through image processing.
2. Baggage Detection: An unsupervised machine learning approach will be used for baggage detection and tracking based on video analysis and sensors.
3. Route Optimization: Machine learning and reinforcement learning algorithms will be applied to calculate optimal baggage transport routes, considering factors such as airport load and baggage location.

#### A. Supervised model

The CTC algorithm can calculate the probability of any Y given an X. The crucial aspect in computing this probability lies in how CTC conceptualizes alignments between input and output sequences. We will begin by examining these alignments and subsequently demonstrate how to utilize them for computing the loss function and conducting inference. [4] The Convolutional Recurrent Neural Network (CRNN) is a fusion of two highly influential neural network architectures. It comprises a Convolutional Neural Network followed by a Recurrent Neural Network. Although the suggested network shares similarities with the CRNN, it produces superior or more optimal outcomes, particularly in the realm of audio signal processing. [5] 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(5):
    ax = plt.subplot(2, 3, i+1)
    img_dir = os.path.join(train_dir, 'train_imgs', f'FILENAME')
    image = cv2.imread(img_dir, cv2.IMREAD_GRAYSCALE)
    plt.imshow(image, cmap=gray)
    plt.title(train_imgs[i], f'{IDENTITY}', fontsize=22)
    plt.axis('off')
plt.subplots_adjust(wspace=0.2, hspace=0.2)
```



```
train.dropna(inplace=True)
valid.dropna(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  
Train set size  
python input-to-ascii(ffff):12: Setting the comparing:  
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/10min/reshaping.html#reshaping-a-view-versus-a-copy>  
train['IDENTITY'] = train['IDENTITY'].str.upper()  
python input-to-ascii(ffff):13: Setting the comparing:  
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/10min/reshaping.html#reshaping-a-view-versus-a-copy>  
valid['IDENTITY'] = valid['IDENTITY'].str.upper()

Figure 1. Supervised model

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 is explained, the actual code import libraries for additional computer tasks 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 column 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.

#### a) Documentation of the code used to evaluate the model

```
Model
```

```
input_data = Input(shape=(256, 64, 1), name='input')

# First convolutional layer
inner = Conv2D(32, (3, 3), padding='same', name='conv1', kernel_initializer='he_normal')(input_data)
inner = BatchNormalization()(inner)
inner = Activation('relu')(inner)
inner = MaxPooling2D(pool_size=(2, 2), name='max1')(inner)

# Second convolutional layer
inner = Conv2D(64, (3, 3), padding='same', name='conv2', kernel_initializer='he_normal')(inner)
inner = BatchNormalization()(inner)
inner = Activation('relu')(inner)
inner = MaxPooling2D(pool_size=(2, 2), name='max2')(inner)
inner = Dropout(0.3)(inner)

# 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=(2, 2), name='max3')(inner)
inner = Dropout(0.3)(inner)

# Flatten the output for CNN to RNN transition
inner = Flatten(target_shape=(64, 64), name='reshape')(inner)
inner = Dense(64, activation='relu', kernel_initializer='he_normal', name='dense')(inner)
```

Figure 2. Definition of 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

```

Figure 3. Convolutional layers

#### b) 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.

```

testreader.compile(loss=[ctc_lambda_func], optimizer=Adam(lr=0.0001))

testreader.fit(
    x=train_x, train_y=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: `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 180s/step - loss: 34.3736 - val_loss: 21.6412
Epoch 2/60
24/24 [====] - 228s 9s/step - loss: 20.9072 - val_loss: 21.2514
Epoch 3/60
24/24 [====] - 228s 9s/step - loss: 20.7146 - val_loss: 21.0626
Epoch 4/60
24/24 [====] - 240s 180s/step - loss: 20.4528 - val_loss: 21.0547
Epoch 5/60
24/24 [====] - 238s 180s/step - loss: 20.2855 - val_loss: 20.9093
Epoch 6/60
24/24 [====] - 238s 180s/step - loss: 20.1728 - val_loss: 20.6252
Epoch 7/60
24/24 [====] - 235s 180s/step - loss: 20.0643 - val_loss: 20.5808
Epoch 8/60
24/24 [====] - 229s 180s/step - loss: 19.9312 - val_loss: 20.5010
Epoch 9/60
24/24 [====] - 237s 180s/step - loss: 19.8029 - val_loss: 20.6698
Epoch 10/60

```

Figure 4. Training of the model

#### c) 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 pretrained 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.

#### d) 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.

### e) Evidence and obtained results

```
Epoch 21/30
24/24 [=====] - 7s 295ms/step - loss: 9.1936
Epoch 22/30
24/24 [=====] - 6s 254ms/step - loss: 8.3912
Epoch 23/30
24/24 [=====] - 8s 328ms/step - loss: 9.5557
Epoch 24/30
24/24 [=====] - 6s 265ms/step - loss: 7.2736
Epoch 25/30
24/24 [=====] - 7s 275ms/step - loss: 6.4186
Epoch 26/30
24/24 [=====] - 8s 319ms/step - loss: 5.7426
Epoch 27/30
24/24 [=====] - 6s 251ms/step - loss: 5.2775
Epoch 28/30
24/24 [=====] - 7s 293ms/step - loss: 4.9095
Epoch 29/30
24/24 [=====] - 7s 298ms/step - loss: 4.4694
Epoch 30/30
24/24 [=====] - 6s 251ms/step - loss: 4.1508
```

Figure 5. Behavior of the supervised model

```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import os
from tensorflow.keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt
from google.colab import drive

# Google Drive
drive.mount('/content/drive')

# Folder in Google Drive
drive_path = '/content/drive/MyDrive/Datasets'

# Files in the training and test folder
training_path = os.path.join(drive_path, 'train')
test_path = os.path.join(drive_path, 'test')
```

Figure 6 Requisites for the model.

As of right now, there 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.

### B. Unsupervised model

The model requires libraries from sklearn in order to work correctly such as KMeans and PCA. KMeans clustering is one of the simplest unsupervised machine learning algorithms that makes inferences from the dataset using only input vectors without referring to known, or labelled, outcomes [6]. KMeans uses vector quantization and aims to assign each observation to the cluster with the nearest mean or centroid, meaning that in this specific case it is needed to split the data into three groups to optimize baggage transportation in airports. On the other hand, it is mentioned Principal Component Analysis (PCA), that is a dimensionality reduction method to minimize the dimensionality of large datasets, by transforming variables into a smaller one that still contains relevant information [7]. Then it is imported the remaining important libraries.

For this case, the dataset was imported into Google Drive as shown in the figure below, dividing the images in test and train, so it becomes easy to understand what is being worked with.

Then, it is only loaded the images from the Drive folder to the Colab with a specified image size not to overload the system, then it is displayed the clusters that takes three main inputs, selecting randomly from the identified cluster indices without replacement. However, it is not done there yet, because the following code lines create a subplot for each image in a horizontal layout. The display\_cluster\_images function takes a cluster number, cluster labels, and image data as inputs. Its purpose is to visually present a random selection of images from the specified cluster. The function identifies the indices of images in the given cluster, randomly selects a subset of these indices, and displays the corresponding images in a row of subplots. Each subplot includes the image and its index, with axis labels turned off for clarity. The function provides a convenient way to inspect the contents of a particular cluster within a dataset.

```
# Function to load images from a folder
def load_images_from_folder(folder):
    images = []
    for filename in os.listdir(folder):
        img_path = os.path.join(folder, filename)
        img = image.load_img(img_path, target_size=(224, 224))
        img_array = image.img_to_array(img)
        images.append(img_array)
    return np.array(images)

# Function for displaying images of the cluster
def display_cluster_images(cluster_number, cluster_labels, images, num_display=5):
    cluster_indices = np.where(cluster_labels == cluster_number)[0]
    selected_indices = np.random.choice(cluster_indices, num_display, replace=False)

    plt.figure(figsize=(15, 3))
    for i, idx in enumerate(selected_indices):
        plt.subplot(1, num_display, i + 1)
        plt.imshow(images[idx].astype(np.uint8))
        plt.title(f'Imagen {idx}')
        plt.axis('off')
    plt.show()
```

Figure 7. Definition of the model

In this part, the training and test image data are loaded, and the image arrays are flattened. First, load the training and test images from the specified folders, converting each image into a NumPy matrix. The resulting matrices are stored in X\_train and X\_test, respectively. The data are then flattened. The training and test data matrices (X\_train and X\_test) are converted into one-dimensional matrices (X\_train\_flat and X\_test\_flat). This



flattening process is a common preprocessing step in machine learning, often performed before feeding data into certain algorithms, such as neural networks. It transforms multidimensional image matrices into one-dimensional matrices while maintaining the number of samples.

```
# Load training data
X_train = load_images_from_folder(training_path)

# Load test data
X_test = load_images_from_folder(test_path)

# Flatten the data
X_train_flat = X_train.reshape(X_train.shape[0], -1)
X_test_flat = X_test.reshape(X_test.shape[0], -1)
```

Figure 8. Parameters for training

In this section of the code, the dimensionality of the image data is reduced using PCA followed by the training of a K-Means clustering model. First, PCA is applied to the flattened training data to transform it into a lower-dimensional space while retaining 90 principal components. The same transformation is applied to the flattened test data. Subsequently, a K-Means clustering model is initialized with three clusters. The model is trained using the reduced features obtained from PCA on the training data. This step involves the algorithm learning to partition the data into three distinct clusters based on the reduced feature set. Finally, the trained K-Means model is used to predict cluster assignments for the test data. Each data point in the test set is assigned to one of the three clusters determined during training.

```
# Dimensionality reduction with PCA
pca = PCA(n_components=90)
X_train_pca = pca.fit_transform(X_train_flat)
X_test_pca = pca.transform(X_test_flat)

# Training the K-Means model
num_clusters = 3
kmeans = KMeans(n_clusters=num_clusters, random_state=42)
kmeans.fit(X_train_pca)

# Predicting clusters for test data
test_clusters = kmeans.predict(X_test_pca)
```

Figure 9. Dimensionality reduction (PCA)

In this section of the code, the script aims to visually inspect and showcase a selection of images from each of the three clusters identified through K-Means clustering. The cluster labels are explicitly printed, providing a clear indication of the clusters being examined. Subsequently, the `display_cluster_images` function is invoked three times, each time with a different cluster number. This function randomly selects a subset of images from the specified cluster and presents them in a row of subplots for visual examination. The chosen images are based on their indices within the original test data, and the subplot titles include

the respective indices, offering a convenient way to identify and assess the content of each cluster.

```
# Show cluster images
print('Cluster 0')
cluster_to_display = 0
display_cluster_images(cluster_to_display, test_clusters, X_test)

print('Cluster 1')
cluster_to_display = 1
display_cluster_images(cluster_to_display, test_clusters, X_test)

print('Cluster 2')
cluster_to_display = 2
display_cluster_images(cluster_to_display, test_clusters, X_test)
```

Figure 10. Deployment of clusters

#### a) Evidence and obtained results

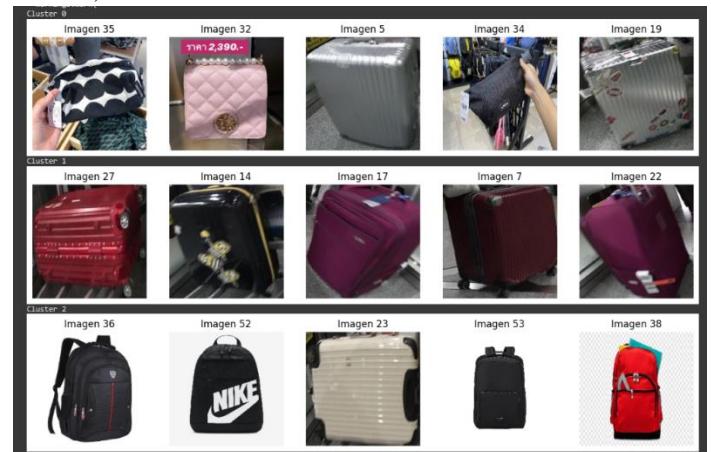


Figure 11. Behavior of the model

As observed in the generated results, the data was successfully divided into three clusters as specified, corresponding to the desired classifications: handbag, backpack, and suitcase. In Cluster 0, the code appears to face some challenges in accurately identifying handbags. There is a slight confusion with certain suitcases; however, due to shifts in image resolutions, it becomes evident that the sizes of these items resemble small handbags. It is important to note that, as demonstrated later, the code is capable of correctly identifying handbags without difficulty. For Cluster 1, it is evident that all suitcases are identified without any issues, even when they vary in color. Subsequent demonstrations will further illustrate the code's capability to accurately identify these objects. For Cluster 2, the code seems to have an easier time identifying backpacks compared to handbags. Although there is an instance of a different item, it can be assumed that the code achieves a high accuracy in this category. Further details and demonstrations will support the understanding of the code's proficiency in accurately classifying these objects.

```

from google.colab import files

# Function to upload images
def load_image():
    uploaded = files.upload()
    file_path = list(uploaded.keys())[0]
    img = image.load_img(file_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    return img_array

# Function to predict the cluster of a new image
def predict_cluster(new_img_array, pca_model, kmeans_model):
    new_img_flat = new_img_array.reshape(1, -1)

    if pca_model is not None:
        new_img_pca = pca_model.transform(new_img_flat)
    else:
        new_img_pca = new_img_flat

    # Predicting the cluster using the K-Means model
    cluster = kmeans_model.predict(new_img_pca)

    return cluster[0]

# Load new image
new_img_array = load_image()

# Predicting the cluster of the new image
predicted_cluster = predict_cluster(new_img_array, pca, kmeans)

# Show results
print(f'La nueva imagen pertenece al clúster {predicted_cluster}')

# Show image
plt.imshow(new_img_array.astype(np.uint8))
plt.title(f'Imagen a clasificar en el clúster {predicted_cluster}')
plt.axis('off')
plt.show()

```

Figure 12. Code proficiency



Figure 13. Cluster 1, identification of luggage

This code introduces functionality to dynamically upload a new image and predict its cluster assignment based on pre-trained models. The `load_image` function facilitates interactive image uploading, converting the uploaded image into a NumPy array using standard preprocessing steps. The `predict_cluster` function takes the new image array, along with the PCA and K-Means models, to predict the cluster of the image. The uploaded image is then loaded, and its cluster is predicted using the models. The results, including the predicted cluster, are printed, and the image is displayed alongside its predicted cluster label. As observed, the prediction made about the image aligns with the expectations established in the previous implementation. The model correctly identifies the image as belonging to Cluster 1, where suitcases are clustered. This alignment between the predicted cluster and the content of the image underscores the effectiveness of the model in accurately classifying new, unseen

data, reinforcing its capability to generalize well beyond the training set.



Figure 14. identification of backpack

Similarly, testing was conducted with a different image to evaluate the implementation. In this case, a distinct image was provided for the model to identify its corresponding cluster. The model successfully recognized that the image belongs to Cluster 2, which aligns accurately with the actual content of the image. This successful prediction reinforces the robustness of the implementation in correctly classifying diverse images into their respective clusters.



Figure 15. identification of handbag

For the final test, an image featuring a handbag was utilized, using the same code as in the previous evaluations. The classification was accurate, as the model correctly identified the image as belonging to Cluster 0. This aligns with the expectation, as Cluster 0 predominantly contains handbags. However, as observed in the model's performance, it exhibits

minor challenges in achieving perfect identification of handbags. While the model generally succeeds, it faces slight difficulties in achieving a 100% accuracy in the recognition of this category.

### C. Reinforcement learning model.

Reinforcement learning is a machine learning training method based on rewarding desired behaviors and punishing undesired ones. In general, a reinforcement learning agent -- the entity being trained -- is able to perceive and interpret its environment, take actions, and learn through trial and error. Reinforcement learning deals with artificial intelligence that allows machines to work independently without any manual interpretation to reach their goals and to interact with dynamic environment. [8]

Q-learning, a reinforcement learning algorithm in machine learning, operates on a model-free basis, implying it doesn't necessitate understanding the intricacies of the underlying system dynamics. Rather, it derives knowledge through interactions with the environment to formulate decisions aimed at maximizing cumulative rewards. The Q-value, represents the expected cumulative reward of taking a particular action in a specific state and following a particular policy thereafter. [9]

In this particular case, we faced many setbacks, just as in the previous models, we experienced errors in the code that took hours to fix, this delayed us until the very last moments, now we cannot fully deploy this model, for we require more time.

terminate (False for the done flag), and a new box is randomly chosen for the next iteration. This simple environment serves as a learning environment for the agent to associate box labels with the correct containers, with success measured by the frequency of correct classifications.

```
env = BoxSortingEnv()

# Define the Q-network model
model = tf.keras.Sequential([
    layers.Dense(32, activation='relu', input_shape=(5,)), # Update input shape here
    layers.Dense(5, activation='linear')
])

# Define the Q-learning agent
class QLearningAgent:
    def __init__(self, model, learning_rate=0.001, gamma=0.99):
        self.model = model
        self.optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
        self.gamma = gamma

    def train(self, state, action, reward, next_state):
        # Convert the state and next_state to integers
        state = ord(state) - ord('a')
        next_state = ord(next_state) - ord('a')

        # Convert the state and next_state to one-hot encoded vectors
        state_one_hot = tf.one_hot([state], depth=5)
        next_state_one_hot = tf.one_hot([next_state], depth=5)

        # Calculate the target Q-value
        target = reward + self.gamma * tf.reduce_max(self.model(next_state_one_hot))

        with tf.GradientTape() as tape:
            # Get the predicted Q-value for the current state
            predicted_values = self.model(state_one_hot, training=True)
            predicted_q_value = predicted_values[0, action]

            # Calculate the loss
            loss = tf.reduce_mean(tf.square(target - predicted_q_value))

        # Update the model weights
        gradients = tape.gradient(loss, self.model.trainable_variables)
        self.optimizer.apply_gradients(zip(gradients, self.model.trainable_variables))
```

Figure 117. Defining Q-network model and agent

```
# Define the environment
class BoxSortingEnv(gym.Env):
    def __init__(self):
        super(BoxSortingEnv, self).__init__()
        self.observation_space = gym.spaces.Discrete(5) # 5 labels (a, b, c, d, e)
        self.action_space = gym.spaces.Discrete(5) # 5 containers (A, B, C, D, E)

    def reset(self):
        self.current_box = np.random.choice(['a', 'b', 'c', 'd', 'e'])
        self.current_container = np.random.choice(['A', 'B', 'C', 'D', 'E'])
        return self.current_box

    def step(self, action):
        # Update the container based on the chosen action
        self.current_container = ['A', 'B', 'C', 'D', 'E'][action]

        # Reward the agent for correct classification
        reward = 1 if self.current_box == self.current_container.lower() else 0

        # Generate a new box
        self.current_box = np.random.choice(['a', 'b', 'c', 'd', 'e'])

        # Return the next state, reward, and done flag
        return self.current_box, reward, False, {}
```

Figure 116. Define environment

The BoxSortingEnv is a reinforcement learning environment designed for box sorting tasks within the OpenAI Gym framework. The observation space consists of five discrete labels ('a', 'b', 'c', 'd', 'e'), while the action space represents five containers ('A', 'B', 'C', 'D', 'E'). Upon initialization, the environment randomly selects a box and a container. The reset method returns the initial state, which is the label of the selected box. In each step, the agent chooses a container, and the environment updates the container accordingly. The agent receives a reward of 1 if the box label matches the container label (case-insensitive), and 0 otherwise. The episode does not

In part, the setup for the box sorting environment is extended by introducing a Q-learning agent. First, an instance of the box sorting environment (BoxSortingEnv) is created. Following that, a Q-network model is defined using TensorFlow's Keras Sequential API, comprising two dense layers. The Q-learning agent, encapsulated within the QLearningAgent class, is constructed with this Q-network model, a specified learning rate, and a discount factor. The agent's training logic is implemented in the train method, where the current state, chosen action, received reward, and next state are processed. State and next state are transformed into integers and then one-hot encoded to represent the state vectors. The Q-learning update rule is employed to calculate the target Q-value, and the model is trained to minimize the mean squared error between predicted and target Q-values. This Q-learning agent, powered by the defined Q-network, endeavors to learn an optimal policy for the box sorting task by iteratively adjusting its Q-values based on observed rewards and state-action transitions.



```

# Create the Q-learning agent instance
agent = QLearningAgent(model)

# Training loop
num_episodes = 10

for episode in range(num_episodes):
    state = env.reset()
    total_reward = 0

    while True:
        # Choose action based on epsilon-greedy strategy
        epsilon = max(0.1, 1 - episode / 800) # Exploration-exploitation trade-off
        if np.random.rand() < epsilon:
            action = env.action_space.sample() # Exploration
        else:
            action = np.argmax(agent.model.predict(tf.one_hot([state], depth=5)))

        # Take action and observe the next state and reward
        next_state, reward, done, _ = env.step(action)

        # Train the Q-learning agent
        agent.train(state, action, reward, next_state)

        total_reward += reward
        state = next_state

    if done:
        break

    if episode % 50 == 0:
        print(f"Episode {episode}, Total Reward: {total_reward}")

```

Figure 18. Training

In this segment of the code, a Q-learning agent is instantiated with a pre-defined Q-network model, and a training loop is established to facilitate the agent's learning in the box sorting environment. The loop runs for a specified number of episodes, and for each episode, the environment is reset, and the total reward is initialized. Within the episode, the agent employs an epsilon-greedy strategy to determine its actions, balancing exploration and exploitation. The exploration-exploitation trade-off is controlled by the parameter epsilon, which decreases linearly with the episode number. Actions are selected either randomly for exploration or based on the agent's current knowledge for exploitation. The chosen action is executed in the environment, and the agent's Q-network is updated through the training process using the observed state, action, reward, and next state. Progress is periodically printed, displaying the episode number and the total reward obtained, providing insights into the agent's learning performance. This training loop enables the Q-learning agent to iteratively refine its strategy and improve its ability to sort boxes into containers effectively.

```

# Testing loop
num_test_episodes = 10

for _ in range(num_test_episodes):
    state = env.reset()

    while True:
        # Choose the best action based on learned Q-values
        action = np.argmax(agent.model.predict(tf.one_hot([state], depth=5)))

        # Take action and observe the next state
        next_state, _, done, _ = env.step(action)

        # Print the movement of the boxes
        print(f"Move box {state.upper()} to container {[ 'A', 'B', 'C', 'D', 'E' ]}")

        state = next_state

    if done:
        break

```

Figure 118. Testing

In the testing phase of the code, a loop is implemented to assess the Q-learning agent's performance in the box sorting environment over a set number of test episodes. For each episode, the environment is reset, and the agent's actions are determined based on its learned Q-values. The agent selects actions by choosing the one with the highest predicted Q-value for the current state. As the agent interacts with the environment, the movement of boxes is visually displayed through print statements, indicating the selected actions and corresponding containers. This testing loop provides a clear representation of the Q-learning agent's decision-making process, allowing for an insightful examination of its sorting strategy and overall effectiveness in accurately placing boxes into containers.

#### IV. CHALLENGES

Most issues could be considered as hardware limitations, many of the issues had to be thoroughly analyzed and, in some cases, we did not manage to find a concrete solution.

By far, the saddest case was that of the reinforcement model, there were way too many hardware limitations that stopped us from successfully testing.

Nonetheless, we have found that in order to get the adequate results we would need to have more powerful hardware, this is of course in our case, this may not be the same situation for others.

It could have gotten results by having more time for training testing. With that being said, the purpose of the reinforcement model, as stated in the initial project proposal, is to optimize route transportation once identified luggage and airport tag, but reinforcement learning itself demands quite much time only for training because of the environment and yet, it is mentioned that the model must work with previous datasets used in the other models. Especially the airport tag that directly impacts the behavior of the model. Another challenge at hand was computational resources that could not significantly reduce the training model, carrying with it another problem that stopped the process was the hyperparameter tuning that could not be modified since the very first trainings took a lot of time, way more than expected, so it could not be fitted correctly. Finally, the inherent complexity of the problem being solved influences training time. Some problems are inherently more challenging and may require longer training periods.

On the other hand, the initial proposal for supervised learning model was to develop a country tag identification, yet the essence is the same by identifying airlines, making the luggage to reach its destination, the process and methodology changes that organizes baggage in airlines, which results in better outcome for both airport personnel and passengers.

#### V. CONCLUSION

The project successfully tackled the optimization of baggage transport at airports through the implementation of supervised, unsupervised and reinforcement machine learning models. The supervised model, employing a Convolutional Recurrent Neural Network (CRNN), overcomes challenges such as RAM limitations during training. The unsupervised model utilizes



KMeans clustering and Principal Component Analysis (PCA) to categorize baggage into clusters, demonstrating success in predicting cluster assignments for new images.

This project proved to be quite the challenge for us, mainly because of the time we had, there were many complications that stopped us from making certain improvements or finding clear solutions to issues such as the RAM limitations and such.

Overall, the project provides valuable insights into the complexity of machine learning models and proved that these technologies can and eventually will be widely used for processes such as the one we mention and study in this project.

## REFERENCES

- [1] IBM, "What is supervised learning?," IBM. [Online]. Available: <https://www.ibm.com/topics/supervised-learning>. Accessed on: Dec. 4, 2023.
- [2] Google, "What is unsupervised learning?," Google Cloud. [Online]. Available: <https://cloud.google.com/discover/what-is-unsupervised-learning>. Accessed on: Dec. 4, 2023.
- [3] C. Hashemi-Pour and J. M. Carew, "What is reinforcement learning?," Enterprise AI, Aug. 16, 2023. [Online]. Available: <https://www.techtarget.com/searchenterpriseai/definition/reinforcement-learning>. Accessed on: Dec. 6, 2023.
- [4] A. Hannun, "Sequence Modeling With CTC," Distill, Nov. 27, 2017. [Online]. Available: <https://distill.pub/2017/ctc/>. Accessed on: Dec. 4, 2023.
- [5] C. C. Chatterjee, "An Approach Towards Convolutional Recurrent Neural Networks," Towards Data Science, Nov. 27, 2019. [Online]. Available: <https://towardsdatascience.com/an-approach-towards-convolutional-recurrent-neural-networks-a2>
- [6] Education Ecosystem (LEDU), "Understanding K-means Clustering in Machine Learning," Towards Data Science. [Online]. Available: <https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1>. Accessed on: Dec. 4, 2023.
- [7] Z. Jaadi, "A Step-by-Step Explanation of Principal Component Analysis (PCA)," builtin. [Online]. Available: <https://builtin.com/data-science/step-step-explanation-principal-component-analysis>. Accessed on: Dec. 4, 2023.
- [8] K. Unger y D. Badre, "Hierarchical reinforcement learning", en Brain Mapping, Elsevier, 2015, pp. 367–373.
- [9] S. Paul, "An Introduction to Q-Learning: Reinforcement Learning," Floydhub, May 15, 2019. [Online]. Available: <https://blog.floydhub.com/an-introduction-to-q-learning-reinforcement>.