# CLASIFICADOR DE ROPA

Raúl Tenorio, Ariel Calderón, Leonel Molina, Mateo Cueva
*Escuela de Formación de Tecnólogos*
*Escuela Politécnica Nacional*
Quito, Ecuador
{raul.tenorio, ariel.calderon, leonel.molina,
mateo.cueva}@epn.edu.ec

***Resumen* –. En el presente documento se incluye la información detallada del proyecto final para la materia de Fundamentos de Inteligencia Artificial, la cual consta de un clasificador de imágenes de ropa desarrollado con código Python y librerías específicas, brindando así una manera de poder cargar un conjunto de imágenes y clasificarlas en distintas salidas.**

***Índices – Clasificación, Python, Ropa, Tensorflow*.**

## I. INTRODUCCIÓN

El desarrollo de la inteligencia artificial ha permitido mejorar considerablemente los métodos tradicionales de clasificación de objetos. El aprendizaje automático mediante el uso de neuronas artificiales resulta ser una herramienta útil al instante de realizar predicciones, recibiendo previamente datos de entrenamiento que le permitan al algoritmo reconocer qué es lo que se desea obtener.

Por lo tanto, mediante la codificación en Python e implementación de librerías como Tensorflow, Matplotlib, Numpy y Math, se estableció un modelo que es capaz de clasificar ropa en 10 categorías distintas, el dataset se carga directamente de Tensorflow y contiene 70000 imágenes para su uso, específicamente 60000 para entrenamiento y 10000 datos de prueba, siendo manejados en un mismo tamaño de pixeles y transformación en escala de grises, lo cuál permite obtener un programa eficiente si es que en un futuro se desea realizar una clasificación automática.

## II. DESARROLLO DE LA PRÁCTICA

Para comenzar, se hace la importación de las librerías respectivas, tal y como se puede observar en la siguiente imagen.

```
# CLASIFICADOR DE IMÁGENES DE ROPA

# IMPORTACIÓN DE LIBRERÍAS
# Librería general de tensorflow
import tensorflow as tf

# Librería de tensorflow para traer el dataset con las imágenes de ropa directamente
import tensorflow_datasets as tfds

# Librería de matplotlib para mostrar gráficamente las imágenes del dataset
import matplotlib.pyplot as plt

# Librería de apoyo que permite realizae operaciones matemáticas complejas
import math

# Librería para manejar los pixeles de las imágenes como arrays de dos dimensiones
import numpy as np
```

Figura 1. Importación de librerías

Después, desde Tensorflow se importa el modelo de datos (véase Figura 2) y se hace un debug del resultado obtenido con el fin de visualizar la información con la que se va a trabajar (véase Figura 3).

```
# Se descarga el dataset de la empresa "Fashion MNIST" de Zalando quienes realizaron una recopilación de 70000 imágenes de ropa
datos, metadatos = tfds.load('fashion_mnist', as_supervised=True, with_info=True)

# Se imprime en pantalla la variable "metadatos" para visualizar la información que contiene el mismo
print(metadatos)
```

Figura 2. Obtención del dataset desde Tensorflow

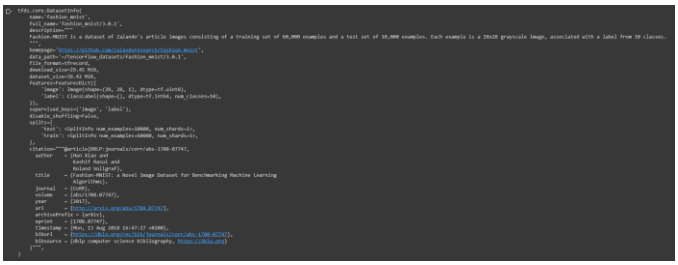


Figura 3. Visualización de la información del dataset obtenido

Ahora, se clasifican los datos de entrenamiento y prueba, además de obtener en un array el nombre de las salidas que tendrá el algoritmo.

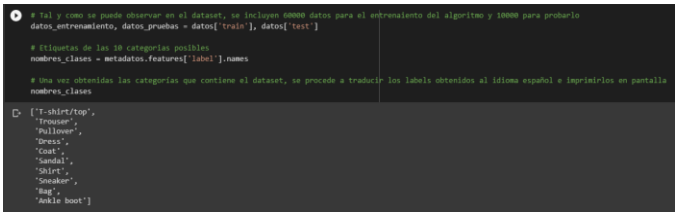



Figura 4. Definición de datos de entrenamiento y prueba

Los labels del algoritmo por defecto vienen en idioma inglés, sin embargo, con fines prácticos se realizó la traducción de estos al lenguaje español.


Figura 5. Traducción de labels de inglés a español

Una de las cuestiones más relevantes que hay que recalcar es que los datos deben ser normalizados, es decir, que no haya valores nulos, vacíos o que sean "corruptos" (véase Figura 6), además de que, se codifica la salida de una imagen de prueba para comprobar el procedimiento (véase Figura 7).

```
# Se procede a definir una función que se encargue de normalizar los datos que se tienen para transformar la gama de colores blanco y negro de 0 a 255 al rango de 0 y 1
def normalizar(imagenes, etiquetas):
  imagenes = tf.cast(imagenes, tf.float32)
  imagenes /= 255 # Aquí se transforma de 0-255 a 0-1
  return imagenes, etiquetas

# Se hace un mapeo de los datos de entrenamiento y de pruebas
datos_entrenamiento = datos_entrenamiento.map(normalizar)
datos_pruebas = datos_pruebas.map(normalizar)

# Ahora, se procede a utilizar los datos en memoria caché en vez del disco, ya que, de esta manera el tiempo de ejecución en entrenamiento tarda menos
datos_entrenamiento = datos_entrenamiento.cache()
datos_pruebas = datos_pruebas.cache()

# Para comprobar que el procedimiento se está realizando de manera correcta, se toma la primera imagen del dataset y utilizando la librería de matplotlib se muestra el resultado
for imagen, etiqueta in datos_entrenamiento.take(1):
  break
imagen = imagen.numpy().reshape((28,28)) # Se aplica el redimensionamiento de 100x100px a 28x28px

# Se dibuja la primera imagen del dataset
plt.figure()
plt.imshow(imagen, cmap=plt.cm.binary)
plt.colorbar()
plt.grid(False)
plt.show()
```

Figura 6. Normalización de datos

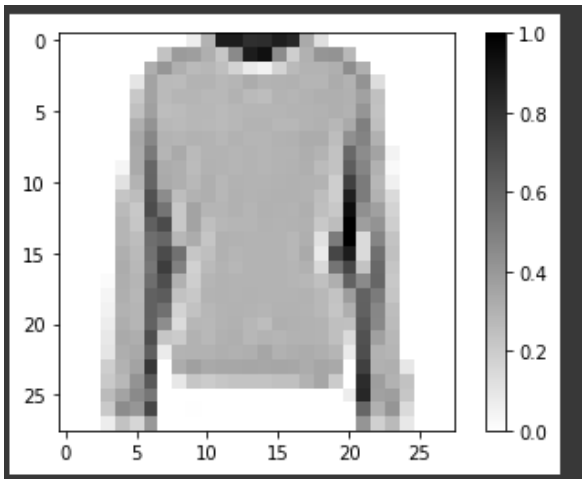


Figura 7. Salida de imagen de prueba

Posteriormente, también se muestran más datos para verificar tanto imágenes como labels.

```
# Ahora que tenemos dicha comprobación, se muestran más imágenes para su última verificación
plt.figure(figsize=(10,10))
for i, (imagen, etiqueta) in enumerate(datos_entrenamiento.take(25)):
  imagen = imagen.numpy().reshape((28,28)) # Se aplica el redimensionamiento de nuevo
  plt.subplot(5,5,i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False) # Se deshabilitan las líneas grid del gráfico
  plt.imshow(imagen, cmap=plt.cm.binary)
  plt.xlabel(nombres_clases[etiqueta])
plt.show()
```

Figura 8. Codificación para obtener varias imágenes


Figura 9. Imágenes de prueba obtenidas

Una vez comprobado que los datos han sido normalizados, se procede a implementar el modelo de clasificación, en donde las imágenes contienen un tamaño de 25 pixeles y se encuentran en escala de grisses, esto permite clasificar a los tonos en un rango de 0 a 255, lo cuál se transforma en una matriz bidimensional que luego será capáz de reconocer formas e identificar a qué tipo de prenda pertenece.

```
# Se crea el modelo
modelo = tf.keras.Sequential([
  tf.keras.layers.Flatten(input_shape=(28,28,1)), # El atributo 1 quiere decir que se renocen como imágenes en escala de grises (blanco y negro)
  tf.keras.layers.Dense(50, activation=tf.nn.relu),
  tf.keras.layers.Dense(50, activation=tf.nn.relu),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax) # Se aplican para las redes de clasificación
])

# Se hace uso del modelo creado
modelo.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy']
)

# Obtenemos el número de datos para entrenamiento y pruebas
num_ej_entrenamiento = metadatos.splits["train"].num_examples
num_ej_pruebas = metadatos.splits["test"].num_examples

# Se imprimen los números obtenidos
print(num_ej_entrenamiento)
print(num_ej_pruebas)

# Para que la carga de los datos no se haga tardada, se trabajarán por lotes de 32 imágenes
TAMANO_LOTE = 32

# Las funciones de "shuffle" y "repeat" mezclan aleatoriamente los datos para que la red no aprenda ningún tipo de orden en la organización de las imágenes
datos_entrenamiento = datos_entrenamiento.repeat().shuffle(num_ej_entrenamiento).batch(TAMANO_LOTE)
datos_pruebas = datos_pruebas.batch(TAMANO_LOTE)

# Se procede a realizar el entrenamiento enviando todas las variables calculadas con anterioridad
historial = modelo.fit(datos_entrenamiento, epochs=5, steps_per_epoch= math.ceil(num_ej_entrenamiento/TAMANO_LOTE))
```

Figura 10. Implementación del modelo de predicción

```
60000
10000
Epoch 1/5
1875/1875 [==============================] - 7s 1ms/step - loss: 0.5235 - accuracy: 0.8146
Epoch 2/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.3861 - accuracy: 0.8594
Epoch 3/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3474 - accuracy: 0.8734
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3281 - accuracy: 0.8786
Epoch 5/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3088 - accuracy: 0.8851
```

Figura 11. Ejecución de los datos de entrenamiento

Por otro lado, también se calcula un margen de error para visualizar la precisión que tendrá el algoritmo al momento de realizar predicciones.

```
# Ahora, se procede a visualizar una gráfica que muestra la magnitud de la pérdida en la información
plt.xlabel("Número De Época")
plt.ylabel("Magnitud De Pérdida")
plt.plot(historial.history["loss"])
```

```
[<matplotlib.lines.Line2D at 0x7f105bae1cd0>]
```

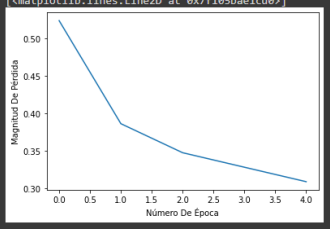
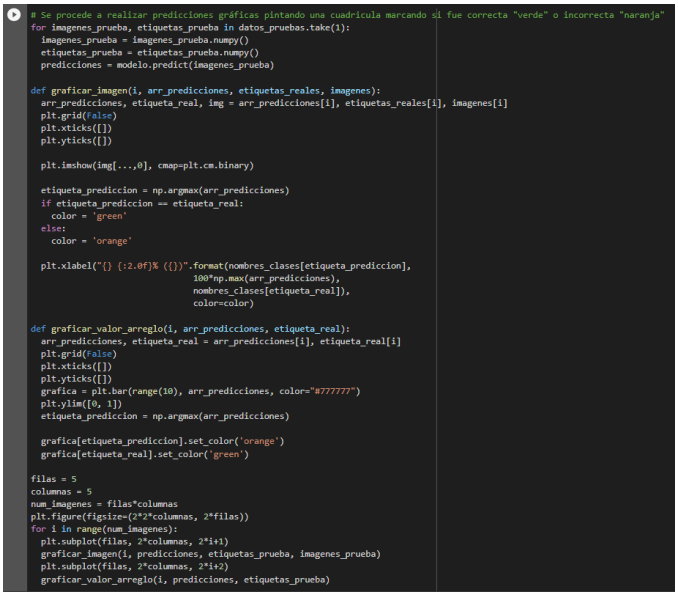



Figura 12. Magnitud de pérdida de información

Así, mediante el uso de la librería de Numpy y Matplotlib, se realiza un esquema gráfico en el cuál la predicción obtenida se identificará como correcta con una barra de color verde y una naranja en caso de que haya fallado.



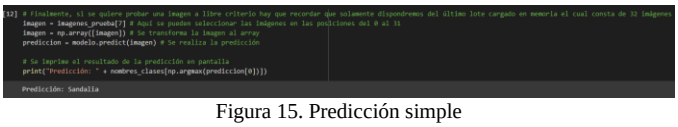
```
# Se procede a realizar predicciones gráficas pintando una cuadrícula marcando si fue correcta "verde" o incorrecta "naranja"
for imagenes_prueba, etiquetas_prueba in datos_pruebas.take(1):
  imagenes_prueba = imagenes_prueba.numpy()
  etiquetas_prueba = etiquetas_prueba.numpy()
  predicciones = modelo.predict(imagenes_prueba)

def graficar_imagen(i, arr_predicciones, etiquetas_reales, imagenes):
  arr_predicciones, etiqueta_real, img = arr_predicciones[i], etiquetas_reales[i], imagenes[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img[...,0], cmap=plt.cm.binary)

  etiqueta_prediccion = np.argmax(arr_predicciones)
  if etiqueta_prediccion == etiqueta_real:
    color = 'green'
  else:
    color = 'orange'

  plt.xlabel("{} {:2.0f}% ({})".format(nombres_clases[etiqueta_prediccion],
                                100*np.max(arr_predicciones),
                                nombres_clases[etiqueta_real]),
                                color=color)

def graficar_valor_arreglo(i, arr_predicciones, etiqueta_real):
  arr_predicciones, etiqueta_real = arr_predicciones[i], etiqueta_real[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])
  grafica = plt.bar(range(10), arr_predicciones, color="#777777")
  plt.ylim([0, 1])
  etiqueta_prediccion = np.argmax(arr_predicciones)

  grafica[etiqueta_prediccion].set_color('orange')
  grafica[etiqueta_real].set_color('green')

filas = 5
columnas = 5
num_imagenes = filas*columnas
plt.figure(figsize=(2*2*columnas, 2*filas))
for i in range(num_imagenes):
  plt.subplot(filas, 2*columnas, 2*i+1)
  graficar_imagen(i, predicciones, etiquetas_prueba, imagenes_prueba)
  plt.subplot(filas, 2*columnas, 2*i+2)
  graficar_valor_arreglo(i, predicciones, etiquetas_prueba)
```

Figura 13. Codificación del modelo de predicción gráfica

Entonces, se puede apreciar el resultado en donde las predicciones después de un rato de entrenamiento han logrado mostrar resultados en su mayoría correctos, en algunos casos aún se pueden presentar fallas, sin embargo, estas se pueden dar por la calidad de las imágenes en la cuál al algoritmo le puede resultar complicado reconocer a qué figura se acopla mejor.

Figura 14. Resultados obtenidos

Finalmente, queda destacar que los datos fueron cargados en memoria en vez del disco para agilizar el proceso, además de que se cargó en lotes de 32 imágenes, el último lote es el que se imprime en pantalla para su predicción, por lo tanto, obteniendo la posición de dicha imagen también se puede hacer una predicción sencilla tal y como se puede observar en la Figura 15.



```
[12] # Finalmente, si se quiere probar una imagen a libre criterio hay que recordar que solamente dispondremos del último lote cargado en memoria el cual consta de 32 imágenes
     imagen = imagenes_prueba[7] # Aquí se pueden seleccionar las imágenes en las posiciones del 0 al 31
     imagen = np.array([imagen]) # Se transforma la imagen al array
     prediccion = modelo.predict(imagen) # Se realiza la predicción

     # Se imprime el resultado de la predicción en pantalla
     print("Predicción: " + nombres_clases[np.argmax(prediccion[0])])

     Predicción: Sandalia
```

Figura 15. Predicción simple

**Link (Google Colaboratory):**
https://colab.research.google.com/drive/1YLigAc0daS6QJcGQou1C9i62bFZ8TiJ5?usp=sharing

**Link (Github):**
https://github.com/raul-tenorio/clasificador-de-ropa

## III. Agradecimientos

Los autores agradecen la colaboración prestada al ingeniero J.P. Zaldumbide por brindarnos los archivos base para el desarrollo del proyecto.

## IV. Biografías

**Raúl Tenorio,** nació en Quito-Ecuador el 9 de marzo de 2002. Realizó sus estudios secundarios en la Institución Educativa Fiscal "Miguel de Santiago". Actualmente, estudia en la Escuela Politécnica Nacional la carrera de Tecnología Superior en Desarrollo de Software. Áreas de interés: desarrollo de aplicaciones web y móviles, electrónica de consumo, programación y diseño de interfaces. (raul.tenorio@epn.edu.ec)

**Leonel Molina,** nació en Mérida-Venezuela el 18 de junio de 1988. Realizó sus estudios secundarios en el colegio San Martín de Porres, luego cursó la carrera de ingeniería en sistemas en la Universidad de los Andes de Venezuela y actualmente se encuentra estudiando Desarrollo de Software en la Escuela Politécnica Nacional del Ecuador. (leonel.molina@epn.edu.ec)

**Ariel Calderón,** nació en Quito-Ecuador el 26 de junio de 1998. Realizó sus estudios secundarios en la unidad educativa William Caxton. Actualmente realiza sus estudios en la Escuela Politécnica Nacional en Tecnología Superior en Desarrollo de Software. Trabaja en la empresa Soft Warehouse como desarrollador junior desde el 15 de marzo de 2022.
Áreas de interés: desarrollo web, desarrollo de videojuegos, deportes y actividades recreacionales.
(ariel.calderon@epn.edu.ec)

**Mateo Cueva**, nació en Sangolquí- Ecuador el 3 de octubre del 2001. Realizo sus estudios en la Unidad Educativa "San Rafael". Actualmente estudiante de cuarto semestre de la carrera de Desarrollo de Software en la Escuela Politécnica Nacional. Impartió cursos relevantes a la carrera en varias escuelas fiscales del Ecuador, además participo en el desarrollo de varios proyectos de software en el ámbito estudiantil. Áreas de interés: Hardware, desarrollo frontend, ciberseguridad, IoT. (mateo.cueva@epn.edu.ec)