Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Eléctrica

Tarea 3

Reporte

Integrantes: Bastián Garcés G.
Profesor: Claudio A. Perez
Auxiliar: Jorge Zambrano I.
Ayudantes: Gabriel Cubillos F.
Jhon Pilataxi
Ayudante de laboratorio: Juan Pérez C.

Fecha de entrega: 30 de mayo de 2022
Santiago de Chile

Índice de Contenidos

Índice de Figuras

Índice de Códigos

1. Objetivo

Durante las secciones posteriores se expondrán los resultados obtenidos durante la realización de la tarea número 3 del curso EL7007 Introducción al Procesamiento Digital de Imágenes, tarea que tenía como finalidad la extracción de características y clasificación de imágenes de acuerdo a la posición de la mascarilla, es decir, se debía entrenar un clasificador capaz de discernir y clasificar aquellas fotos donde se tenían a personas con mascarillas bien puestas, mal puesta y sin mascarilla. Finalmente se debía emplear el clasificador obtenido sobre 3 fotos que debían ser del estudiante en cuestión, para así corroborar la calidad del clasificador utilizando fotos nuevas.

2. Extractores de característica

Para la realización de esta tarea se tuvieron que implementar 4 métodos de extracción de característica, el método *Local Binnary Pattern* o LBP, el método *Histograma de Gradientes Orientados* o HOG y dos modelos convolucionales, uno de la familia DenseNet y otro de la familia ResNet.

A continuación se expondrán el código utilizado para la obtención de los vectores de característica según los diferentes métodos.

2.1. LBP

En el código 1 se logra observar una función encargada de transformar una imagen a LBP y luego obtener el vector de característica sobre dicha imagen utilizando la función del código 2, donde se extrae un vector de característica a partir de histogramas extraídos de secciones de un tamaño determinado sin traslape.

Código 1: Función `LBP`

```
def LBP(image_path):
  '''
  Función encargada de entregar el vector de característica correspondiente a una imagen
  mediante el uso del método LBP sin traslape.

  Inputs:
    image_path: path de la imagen a trabajar

  Outputs:
    features: vector de característica
  '''
  photo = cv2.imread(image_path, 0)
  photo = cv2.resize(photo, dsize=(photo.shape[0]//2, photo.shape[1]//2))
  photo = local_binary_pattern(photo, P = 8, R = 1)
  features = LBP_sintraslape(photo, photo.shape[0]//16, photo.shape[1]//16)
  return features
```

Código 2: Función `LBP_sintraslape`

```
def LBP_sintraslape(foto_LBP, ancho, alto):
```

```
  '''
  La función LBP_sintraslape extrae un vector de características a partir
  de los histogramas extraidos de la división de la imagen en regiones
  de tamaño anchoxalto sin traslape.

  Inputs:
    foto_LBP: imagen con LBP aplicado
    ancho: ancho de la región que divide la imagen de entrada
    alto: alto de la región que divide la imagen de entrada

  Outputs:
    histogramas: concatenación de los histogramas de cada región
  '''
  histogramas = np.array([])
  for i in range(foto_LBP.shape[0]//ancho): #filas
    for j in range(foto_LBP.shape[1]//alto): #columnas
      histogram, bin_edges = np.histogram(foto_LBP[i*alto:(i+1)*alto][j*ancho:(j+1)*ancho], bins
      ↪ =59)
      histogramas = np.concatenate((histogramas, histogram), axis=0)
  return histogramas
```

2.2. Extractor de características completo

Para el resto de métodos de extracción de característica se utilizaron librerías que ya permitían el uso inmediato de los algoritmos correspondientes. En el código 3 se observa la función *feature_extractor*, cuya misión es retornar el vector de característica deseado ante un método determinado. En dicho código se logra observar que los modelos convolucionales utilizados correspondieron al uso de la red *DenseNet201* y *ResNet101*, donde solo se consideraron las capas convolucionales del modelo (parámetro include_top = False).

Código 3: Función feature_extractor

```
DenseNet_model = DenseNet201(include_top=False, weights='imagenet', input_shape=(224, 224,
    ↪ 3), pooling='avg')
ResNet_model = ResNet101(include_top=False, weights='imagenet', input_shape=(224, 224, 3),
    ↪ pooling='avg')
def feature_extractor(method, image_path):
  '''
  Función encargada de retornar un vector de característica de acuerdo al método
  deseado.

  Inputs:
    method: Método con el cual se quiere realizar la extracción de características,
    puede ser 'LBP', 'HOG', 'DenseNet' o 'ResNet'
    image_path: Path de la imagen a trabajar

  Outputs:
    features: Vector de características correspondiente
  '''
  if method == 'LBP':
```

```
    photo = cv2.imread(image_path, 0)
    photo = cv2.resize(photo, dsize=(photo.shape[0]//2, photo.shape[1]//2))
    photo = local_binary_pattern(photo, P = 8, R = 1)
    features = LBP_sintraslape(photo, photo.shape[0]//16, photo.shape[1]//16)

  elif method == 'HOG':
    photo = cv2.imread(image_path)
    photo = cv2.resize(photo, dsize=(1024,1024))
    features = hog(photo, orientations=8, pixels_per_cell=(32, 32),
                   cells_per_block=(1, 1), visualize=False, multichannel=True)

  elif method == 'DenseNet':
    img = image.load_img(image_path, target_size=(224,224))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)

    features = DenseNet_model.predict(x)
    features = features.flatten()
    features = features.tolist()

  elif method == 'ResNet':
    img = image.load_img(image_path, target_size=(224,224))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input_resnet(x)

    features = ResNet_model.predict(x)
    features = features.flatten()
    features = features.tolist()

  return features
```

2.3. Extracción de características

Una vez implementados los extractores de características a utilizar se procedió a obtener los vectores de características, usando los diferentes métodos sobre todas las imágenes de la base de datos. Es importante mencionar que el proceso de extracción de característica era un tanto lento, motivo por el cual los vectores obtenidos de los diferentes métodos solo se obtuvieron una vez, se transformaron a un archivo *csv* y se guardó dicho archivo en *Google Drive* de tal forma que no se deba incurrir en mucho tiempo de procesamiento cada vez que se iterase el código.

Lo anterior también se realizó sobre las imágenes del estudiante.

3. Resultados

Durante esta sección se expondrán los resultados obtenidos tras implementar un clasificador *Random Forest* y aplicarlo sobre los vectores de característica correspondientes a cada uno de los métodos.

Además, las clases a utilizar corresponderán a las siguientes:

- **Clase 0:** Mascarilla bien puesta.
- **Clase 1:** Mascarilla mal puesta.
- **Clase 2:** Sin mascarilla.

3.1. Función Evaluation2

En el código 4 se presenta la función encargada de implementar el clasificador a utilizar y aplicarlos sobre el método que se desea.

Código 4: Función `Evaluation2`

```
def Evaluation2(method):
  '''
  Función con la que se evalua el conjunto de prueba usando un método de extracción
  de característica determinado.
  Inputs:
    method: Método de extracción de característica a utilizar. Puede ser 'LBP', 'HOG',
    'DenseNet' y 'ResNet'

  Outpust:
    Vector[0][0]: predicción sobre imágenes propias
    Vector[0][1]: predicción sobre el conjunto de prueba
    Vector[1]: labels del conjunto de prueba
    Vector[2]: labels sobre imágenes propias
  '''
  #Lectura de csv dependiendo del método que se quiere testear
  if method == 'LBP':
    features_LBP = pd.read_csv('features_LBP.csv')
    features_LBP = features_LBP.drop(['Unnamed: 0'], axis=1)

    features = features_LBP.iloc[:, 2:]
    clases = features_LBP.iloc[:, 1]

    features_LBP_propio = pd.read_csv('features_LBP_propia.csv')
    features_LBP_propio = features_LBP_propio.drop(['Unnamed: 0'], axis=1)

    features_propio = features_LBP_propio.iloc[:, 2:]
    clases_propio = features_LBP_propio.iloc[:, 1]

  elif method == 'HOG':
    features_HOG = pd.read_csv('features_HOG_v2.csv')
    features_HOG = features_HOG.drop(['Unnamed: 0'], axis=1)
```

```
32
33    features = features_HOG.iloc[:, 2:]
34    clases = features_HOG.iloc[:, 1]
35
36    features_HOG_propio = pd.read_csv('features_HOG_propia.csv')
37    features_HOG_propio = features_HOG_propio.drop(['Unnamed: 0'], axis=1)
38
39    features_propio = features_HOG_propio.iloc[:, 2:]
40    clases_propio = features_HOG_propio.iloc[:, 1]
41
42  elif method == 'DenseNet':
43    features_DenseNet = pd.read_csv('features_DenseNet.csv')
44    features_DenseNet = features_DenseNet.drop(['Unnamed: 0'], axis=1)
45
46    features = features_DenseNet.iloc[:, 2:]
47    clases = features_DenseNet.iloc[:, 1]
48
49    features_DenseNet_propio = pd.read_csv('features_DenseNet_propia.csv')
50    features_DenseNet_propio = features_DenseNet_propio.drop(['Unnamed: 0'], axis=1)
51
52    features_propio = features_DenseNet_propio.iloc[:, 2:]
53    clases_propio = features_DenseNet_propio.iloc[:, 1]
54
55  elif method == 'ResNet':
56    features_ResNet = pd.read_csv('features_ResNet.csv')
57    features_ResNet = features_ResNet.drop(['Unnamed: 0'], axis=1)
58
59    features = features_ResNet.iloc[:, 2:]
60    clases = features_ResNet.iloc[:, 1]
61
62    features_ResNet_propio = pd.read_csv('features_ResNet_propia.csv')
63    features_ResNet_propio = features_ResNet_propio.drop(['Unnamed: 0'], axis=1)
64
65    features_propio = features_ResNet_propio.iloc[:, 2:]
66    clases_propio = features_ResNet_propio.iloc[:, 1]
67
68  #Creación de conjuntos de entrenamiento, validación y prueba
69  X_train, X_test, y_train, y_test = train_test_split(features, clases, test_size=0.3, random_state
      ↪ =42)
70  X_train, X_vali, y_train, y_vali = train_test_split(X_train, y_train, test_size=0.3,
      ↪ random_state=42)
71
72  #Normalizacion de caracteristicas
73  scaler = sklearn.preprocessing.StandardScaler()
74
75  scaler.fit(X_train)
76
77  S_Entrena = scaler.transform(X_train)
78  S_Vali = scaler.transform(X_vali)
79  S_Prueba = scaler.transform(X_test)
80  S_propio = scaler.transform(features_propio)
```

```
81
82  #Concatenamos conjunto de entrenamiento y validacion para que lo use el grid
83  Entrena_Vali = np.concatenate([S_Entrena, S_Vali])
84  Entrena_Vali_class = np.concatenate([y_train, y_vali])
85
86  #Vemos que indices del conjunto Entrena_Vali pertenecen al conjunto de entrenamiento
87  index = np.zeros(X_train.shape[0]+X_vali.shape[0])
88  for i in range(X_train.shape[0]):
89    index[i] = -1
90
91  #Definimos el predifinedSplit
92  ps = sklearn.model_selection.PredefinedSplit(index)
93
94  #------------------------------------------------------------------------
95  #Random Forest que trabaja con todas las caracteristicas
96
97  #Mediante un random forest
98  Forest = sklearn.ensemble.RandomForestClassifier()
99  #Definimos una grilla para seleccionar los hiperparametros
100 GridForest = sklearn.model_selection.GridSearchCV(Forest, param_grid = {'n_estimators' : [150,
      ↪ 175, 200, 225, 250],
101                                                                    'max_depth' : [9, 12, 15]}, cv=ps)
102
103 GridForest.fit(Entrena_Vali, Entrena_Vali_class)
104
105 param_Forest = GridForest.best_params_
106
107 #Obtenemos el mejor estimador
108 estim_Forest = GridForest.best_estimator_
109
110 #Vemos la clasificacion del estimador sobre el conjunto de validacion
111 predi_vali_Forest = estim_Forest.predict(S_Vali)
112
113 #Accuracy del clasificador
114 accuracy_Forest = sklearn.metrics.accuracy_score(y_vali, predi_vali_Forest)
115
116 #Predicciones sobre conjunto de prueba y propio sin reducir caracteristicas
117 predi_test_Forest = estim_Forest.predict(S_Prueba)
118 predi_propio_Forest = estim_Forest.predict(S_propio)
119
120 return [[predi_propio_Forest, predi_test_Forest], y_test, clases_propio]
```

En reglas generales el código anterior se encarga de recibir un string que indica el método que se desea utilizar, luego dependiendo del método se recuperan los vectores de característica respectivos, tanto para las imágenes de la base de datos como para las imágenes del estudiante. Luego se crean los conjuntos de entrenamiento, validación y test sobre las imágenes de la base de datos y se normalizan los vectores de características. Continuando se concatenan los conjuntos de entrenamiento y de validación, lo cual se hace con la finalidad de utilizar una grilla para elegir los mejores hiperparámetros del clasificador. Una vez elegidos los mejores hiperparámetros se aplica el clasificador correspondiente sobre los conjuntos de test (imágenes de la base de datos e imágenes del estudiante).

3.2. Método LBP

Tras aplicar el clasificador sobre los vectores de característica correspondientes al método LBP se obtuvieron los resultados observados en la figura 1, lo cual tomó 184.0436 segundos de tiempo de procesamiento.

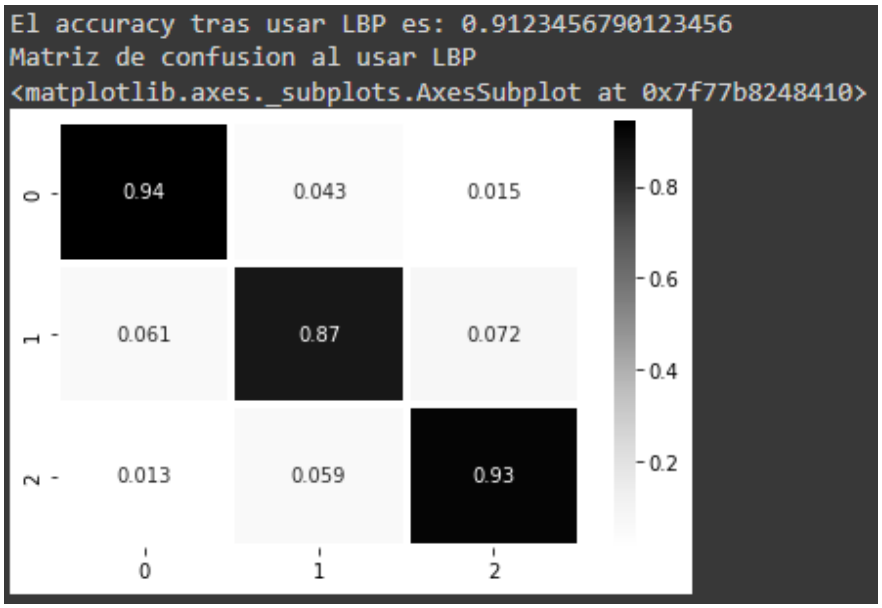

Figura 1: Resultados tras utilizar método LBP

De la matriz de confusión anterior logramos apreciar que utilizando el método LBP para extraer características se tienen mayores dificultades tratando de clasificar aquellos elementos de clase 1 (Mascarilla mal puesta). Pese a lo anterior el accuracy obtenido es bastante elevado.

3.3. Método HOG

Al aplicar el Random Forest sobre los vectores asociados al método HOG se obtienen los resultados de la figura 2, tardando 532.5942 segundos en procesar.

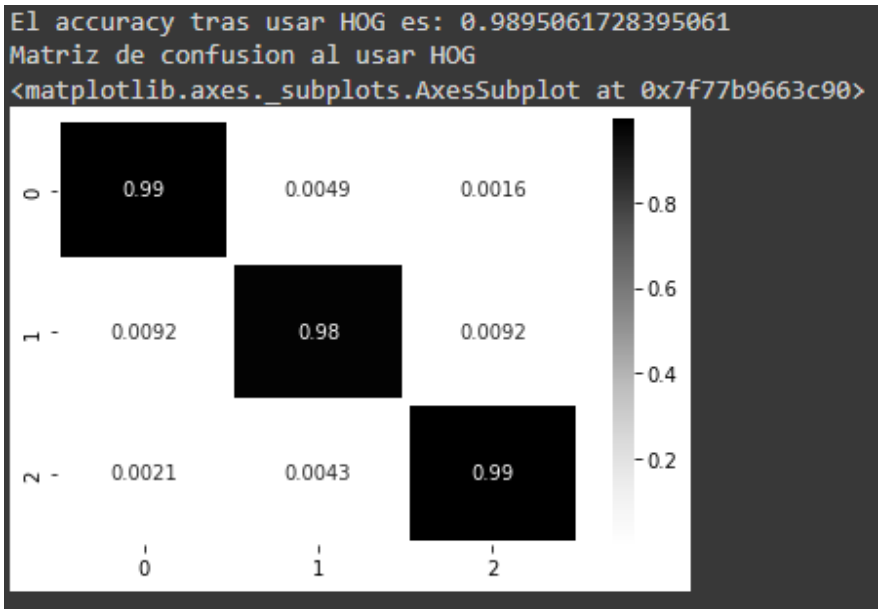

Figura 2: Resultados tras utilizar método HOG

De la figura anterior se logra observar que al utilizar este método de extracción de características se obtiene un método con un accuracy casi perfecto, donde la dificultad al realizar la clasificación se encuentra nuevamente en tratar de identificar aquellos elementos de clase 1, sin embargo esa dificultad es muy poco significativa, pues se tiene un valor muy cercano a la perfección.

Acá es importante notar que este método es muy bueno, sin embargo toma mucho tiempo de procesamiento, cercano a los 9 minutos.

3.4. Método DenseNet

Al aplicar el clasificador sobre los vectores correspondientes a los obtenidos de la red DenseNet201 se obtuvo el accuracy y matriz de confusión observados en la figura 3, tomando 274.9265 segundos.

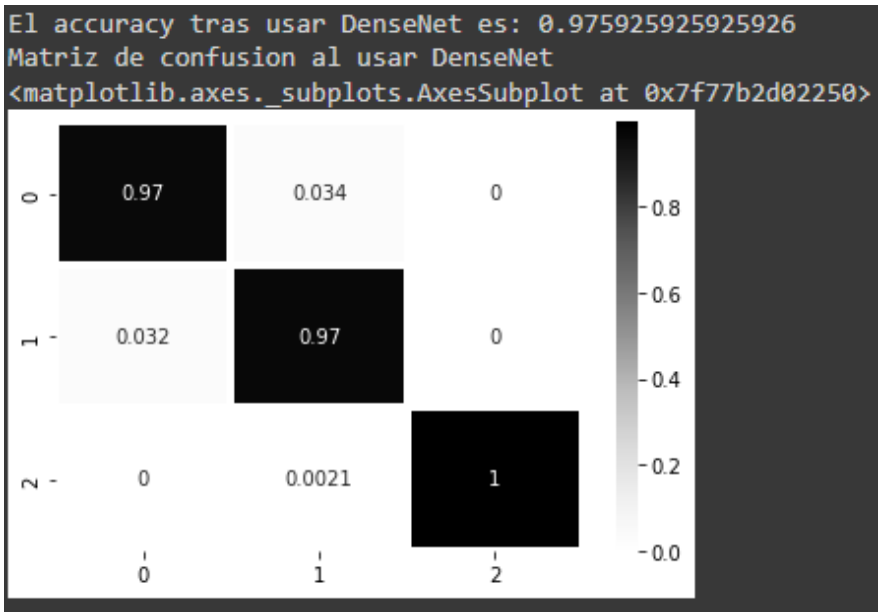

Figura 3: Resultados tras utilizar método DenseNet

De los resultados presentados se logra observar que al utilizar la red DenseNet201 se obtiene un accuracy bastante elevado. Luego, observando la matriz de confusión se identifica que siempre que se predijo clase 2 el valor real era de la misma clase.

Es importante notar que los resultados obtenidos para este caso son muy cercanos a los obtenidos del método HOG, pero el tiempo de procesamiento fue mucho menor motivo por el cual no se podría aseverar que el método actual sea necesariamente peor que el correspondiente al HOG.

3.5. Método ResNet

Finalmente en la figura 4 se observan los resultados asociados al método que utilizaba la red ResNet, tardando 264.8969 segundos en retornar los resultados.

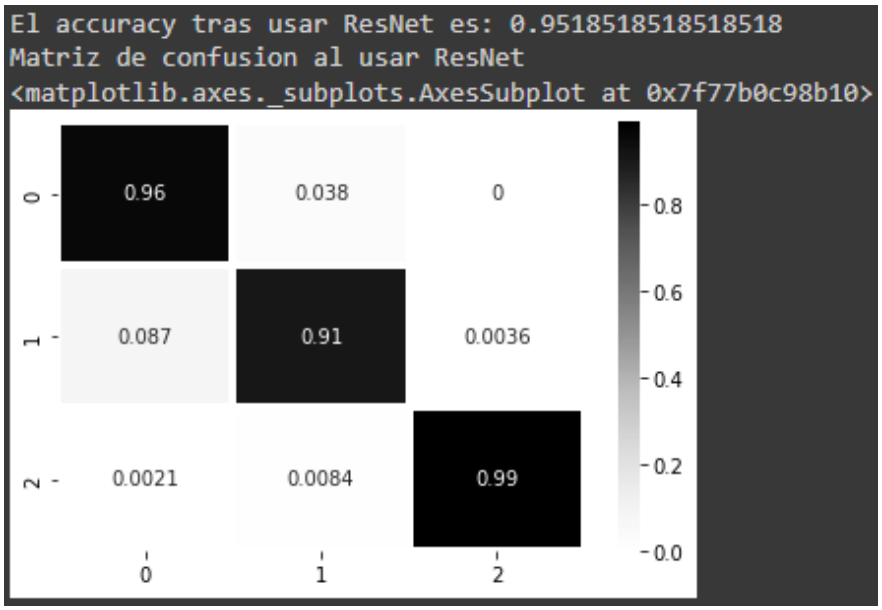

Figura 4: Resultados tras utilizar método ResNet

De los resultados anteriores se logra observar que este método presenta el segundo peor accuracy y de la matriz de confusión se aprecia que, al igual que en el resto de métodos, se tiene mayor dificultad al tratar de identificar aquellos elementos de clase 1.

3.6. Resultados sobre imágenes del estudiante

Con la finalidad de mostrar los resultados sobre las imágenes del estudiante se implementó la función del código 5.

Código 5: Función `Propio`

```
def Propio(index):
  '''
  Función encargada de entregar las predicciones sobre las imagenes propias.

  Inputs:
    index: indica el método a usar. 0 == LBP, 1 == HOG, 2 == DenseNet y 3 == ResNet
  '''
  if index == 0:
    print('El método elegido fue el LBP')

  elif index == 1:
    print('El método elegido fue el HOG')

  elif index == 2:
    print('El método elegido fue el DenseNet')

  elif index == 3:
    print('El método elegido fue el ResNet')

```

```
predi_propio = predic_propia[index]
print('')
print('Las predicciones obtenidas son:', predi_propio)
print('')
print('Las predicciones reales son:', caso_LBP[2].tolist())
print('')
accuracy_mejormetodo = sklearn.metrics.accuracy_score(caso_LBP[2], predi_propio)
print('El accuracy es: '+str(accuracy_mejormetodo))

Confu = sklearn.metrics.confusion_matrix(predi_propio, caso_LBP[2], normalize='true')
print('Matriz de confusion')
sns.heatmap(Confu, linewidths=3, annot=True, cmap="Greys")
```

Antes de presentar los resultados se presentan las imágenes del estudiante, así como la clase a la que corresponden:

Figura 5: Estudiante con mascarilla bien puesta (Clase 0)

Figura 6: Estudiante sin mascarilla (Clase 2)

Figura 7: Estudiante con mascarilla mal puesta (Clase 1)

A continuación se presentan los resultados obtenidos sobre las imágenes anteriores:

- **Método LBP:**

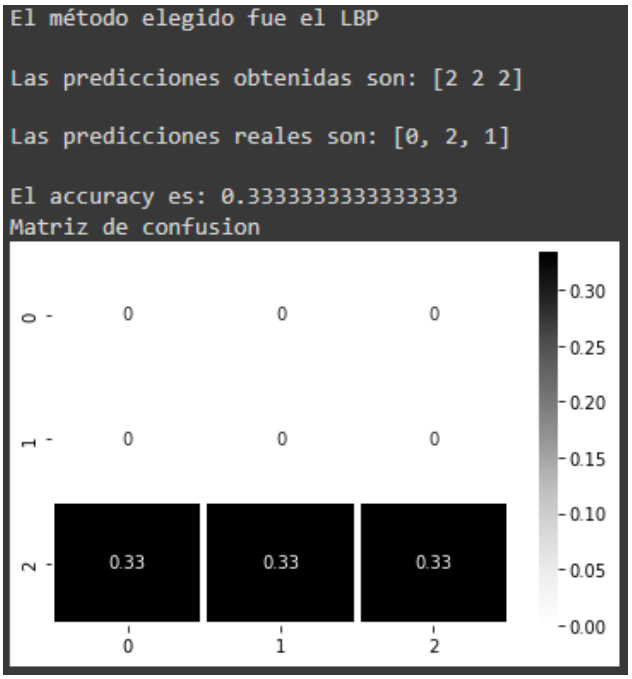

Figura 8: Resultados sobre imágenes del estudiante - LBP

- **Método HOG:**

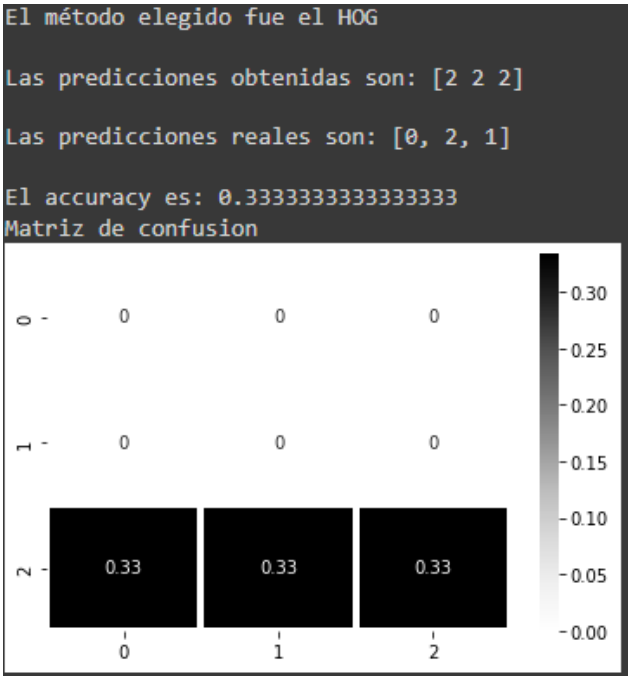

Figura 9: Resultados sobre imágenes del estudiante - HOG

- **Uso de la red DenseNet:**

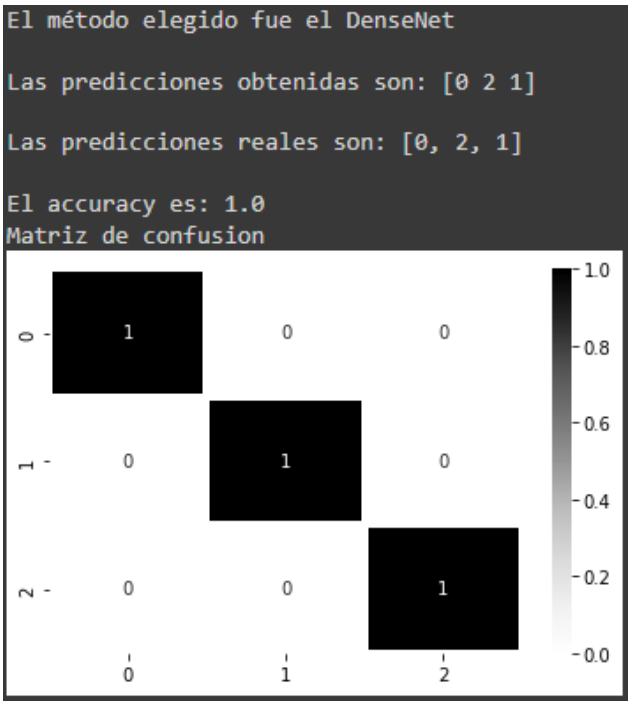

Figura 10: Resultados sobre imágenes del estudiante - DenseNet

- **Uso de la red ResNet:**

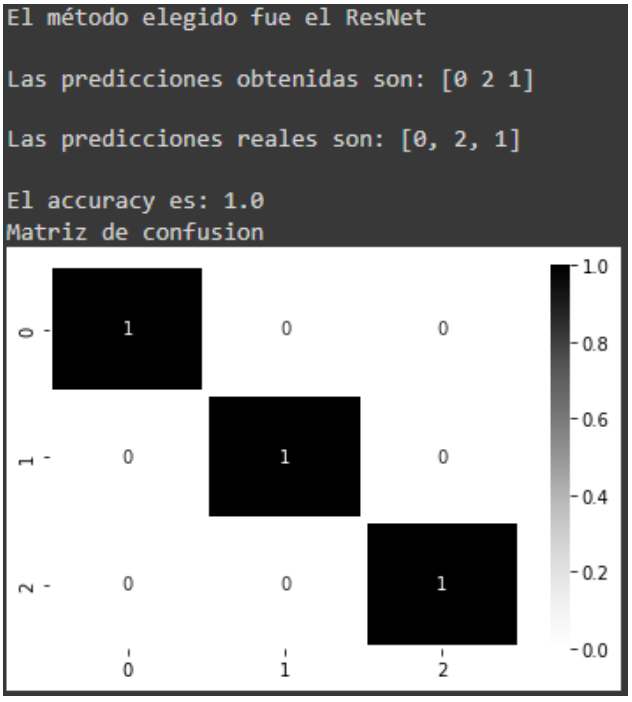

Figura 11: Resultados sobre imágenes del estudiante - ResNet

A partir de los resultados anteriores se logra ver que solo los modelos que utilizan CNN presen-

tan buenos resultados sobre las imágenes del estudiante, mientras que los demás métodos tienden a clasificar las imágenes como clase 2 siempre (Sin mascarilla).

Debido a lo mencionado durante el párrafo anterior es que se puede aseverar el hecho de que pese a que en la base de datos original se obtuvieron mejores resultados, en términos de accuracy y matriz de confusión, usando HOG; el mejor método correspondería al de la red DenseNet201, ya que presenta buenos resultados tanto en las imágenes de la base de datos como en las imágenes del estudiante, además que es un método que se tarda bastante menos tiempo que el método HOG en procesar, lo cual tampoco debería desmerecerse.