Face mask detection

Contents

Step 0: Import libraries

In [2]:

```
import PIL.Image
import matplotlib.pyplot as plt
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

import kagglehub
import pathlib

import numpy as np
```

Step 1: Load and explore data

In [3]:

```
# Download latest version
#path = kagglehub.dataset_download("ashishjangra27/face-mask-12k-images-dataset")

#print("Path to dataset files:", path)
```

In [4]:

```
path = 'C:/Users/ronal/.cache/kagglehub/datasets/ashishjangra27/face-mask-12k-images-d
```

In [5]:

```
path = pathlib.Path(path).with_suffix('')/'Face Mask Dataset'
```

In [7]:

```
len(list(path.glob("*/*/*.png")))
```

Out[7]:

```
11792
```

In [8]:

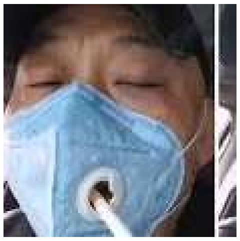
```
mask = list(path.glob("Train/WithMask/*"))

PIL.Image.open(str(mask[0]))
```

Out[8]:

In [9]:

```
PIL.Image.open(str(mask[100]))
```

Out[9]:



In [10]:

```
no_mask = list(path.glob("Train/WithoutMask/*"))

PIL.Image.open(str(no_mask[0]))
```

Out[10]:

In [11]:

```
PIL.Image.open(str(no_mask[100]))
```

Out[11]:

Step 2: Load data using Keras function

In [12]:

```
batch_size = 32
img_height = 128
img_width = 128
```

In [13]:

```
train_ds = tf.keras.utils.image_dataset_from_directory(
  path/'Train',
  seed=9,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 10000 files belonging to 2 classes.
```

In [14]:

```
val_ds = tf.keras.utils.image_dataset_from_directory(
  path/'Validation',
  seed=9,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

```
Found 800 files belonging to 2 classes.
```

In [15]:

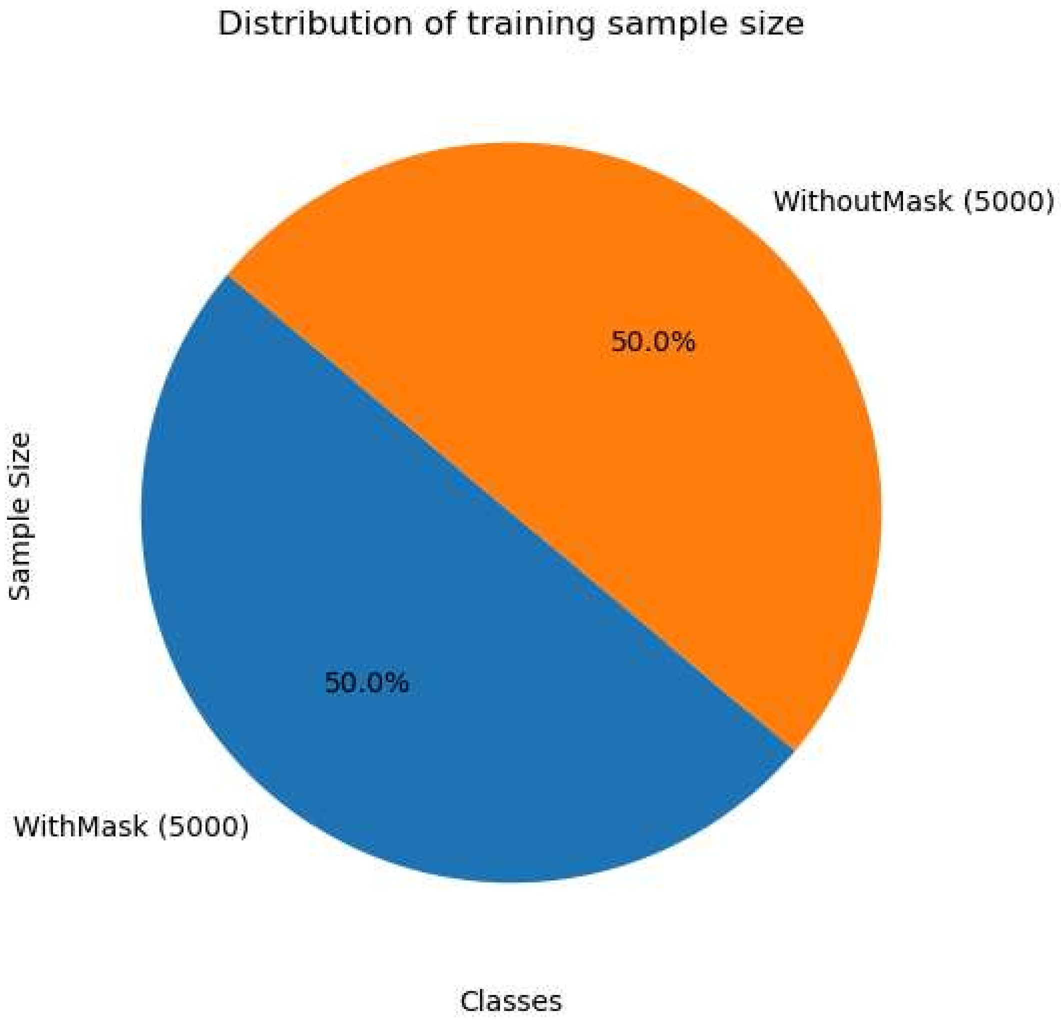
```
test_ds = tf.keras.utils.image_dataset_from_directory(
  path/'Test',
  seed=9,
  batch_size=32,
  image_size=(img_height, img_width))
```

```
Found 992 files belonging to 2 classes.
```

Step 3: Analize training data

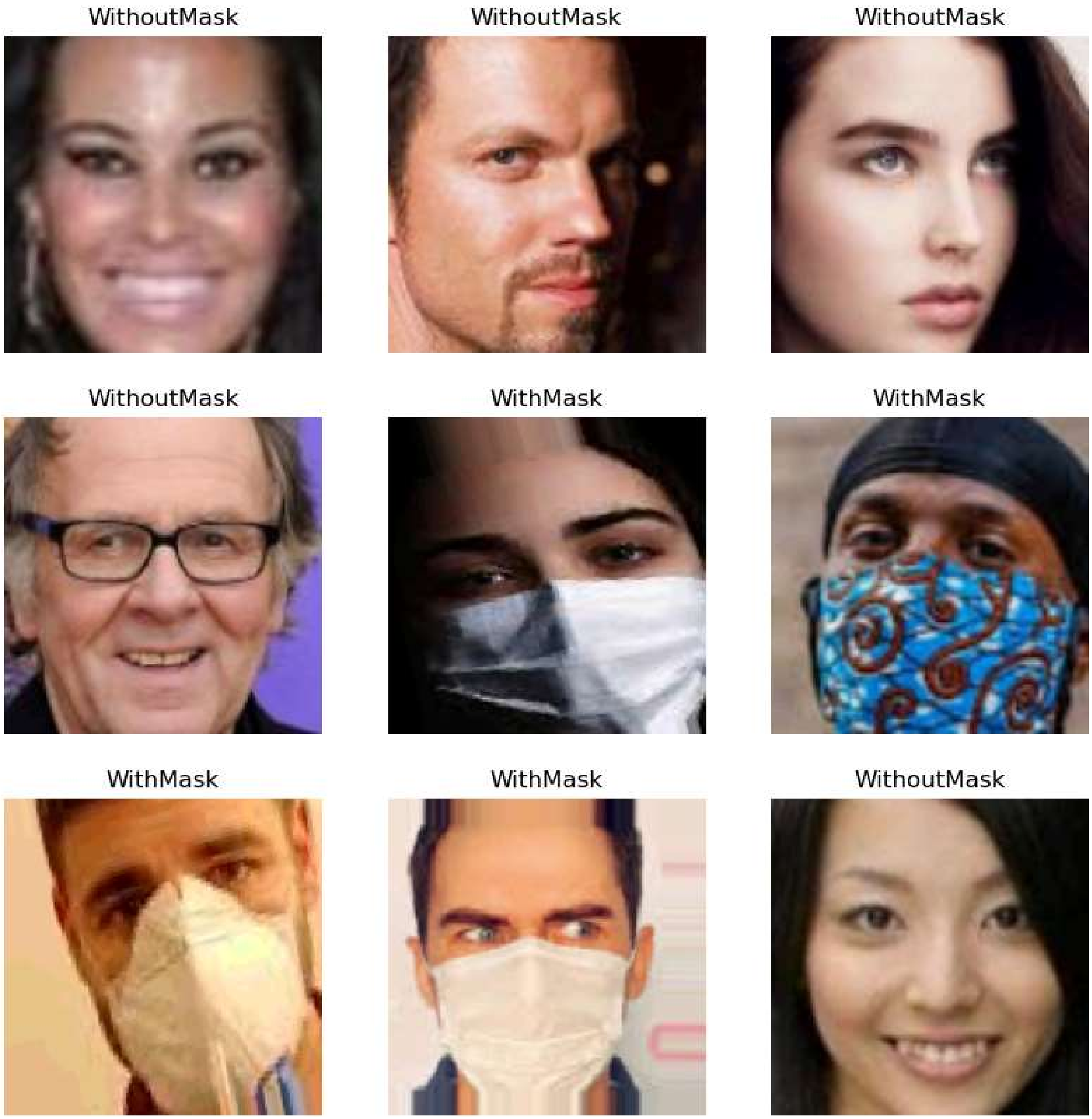
In [16]:

```
class_labels = train_ds.class_names
class_counts = [len(mask), len(no_mask)]

labels = [f"{name} ({count})" for name, count in zip(class_labels, class_counts)]


plt.figure(figsize=(8, 6))
plt.pie( class_counts,labels=labels,autopct='%1.1f%%', startangle=140)
plt.xlabel('Classes')
plt.ylabel('Sample Size')
plt.title('Distribution of training sample size')
plt.show()
```



In [17]:

```python
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_labels[labels[i]])
    plt.axis("off")
```



Step 4: Build the model

In [18]:

```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

In [19]:

```
num_classes = len(class_labels)

model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes)
])
```

```
c:\Users\ronal\anaconda3\Lib\site-packages\keras\src\layers\preprocessing\tf_data_lay
er.py:19: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the first laye
r in the model instead.
  super().__init__(**kwargs)
```

In [20]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

In [21]:
```
model.summary()
```

```
Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                     ┃ Output Shape            ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ rescaling (Rescaling)            │ (None, 128, 128, 3)     │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ conv2d (Conv2D)                  │ (None, 128, 128, 16)    │           448 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ max_pooling2d (MaxPooling2D)     │ (None, 64, 64, 16)      │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ conv2d_1 (Conv2D)                │ (None, 64, 64, 32)      │         4,640 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ max_pooling2d_1 (MaxPooling2D)   │ (None, 32, 32, 32)      │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ conv2d_2 (Conv2D)                │ (None, 32, 32, 64)      │        18,496 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ max_pooling2d_2 (MaxPooling2D)   │ (None, 16, 16, 64)      │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ flatten (Flatten)                │ (None, 16384)           │             0 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ dense (Dense)                    │ (None, 128)             │     2,097,280 │
├──────────────────────────────────┼─────────────────────────┼───────────────┤
│ dense_1 (Dense)                  │ (None, 2)               │           258 │
└──────────────────────────────────┴─────────────────────────┴───────────────┘

 Total params: 2,121,122 (8.09 MB)

 Trainable params: 2,121,122 (8.09 MB)

 Non-trainable params: 0 (0.00 B)
```

In [22]:
```
from tensorflow.keras.utils import plot_model

plot_model(model, to_file='model_architecture.png', show_shapes=True, show_layer_names
```

```
You must install graphviz (see instructions at https://graphviz.gitlab.io/download/)
for `plot_model` to work.
```

Step 5: Fit the model

In [23]:

```
epochs=10
history = model.fit(
                    train_ds,
                    validation_data=val_ds,
                    epochs=epochs
                    )
```

```
Epoch 1/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 46s 121ms/step - accuracy: 0.8780 - loss: 0.2711 - val_a
ccuracy: 0.9875 - val_loss: 0.0470
Epoch 2/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 35s 112ms/step - accuracy: 0.9847 - loss: 0.0385 - val_a
ccuracy: 0.9875 - val_loss: 0.0324
Epoch 3/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 36s 115ms/step - accuracy: 0.9873 - loss: 0.0351 - val_a
ccuracy: 0.9925 - val_loss: 0.0255
Epoch 4/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 34s 108ms/step - accuracy: 0.9915 - loss: 0.0230 - val_a
ccuracy: 0.9950 - val_loss: 0.0211
Epoch 5/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 33s 106ms/step - accuracy: 0.9930 - loss: 0.0170 - val_a
ccuracy: 0.9912 - val_loss: 0.0204
Epoch 6/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 33s 104ms/step - accuracy: 0.9921 - loss: 0.0178 - val_a
ccuracy: 0.9912 - val_loss: 0.0219
Epoch 7/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 38s 122ms/step - accuracy: 0.9964 - loss: 0.0109 - val_a
ccuracy: 0.9950 - val_loss: 0.0246
Epoch 8/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 38s 121ms/step - accuracy: 0.9972 - loss: 0.0084 - val_a
ccuracy: 0.9887 - val_loss: 0.0379
Epoch 9/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 38s 121ms/step - accuracy: 0.9924 - loss: 0.0244 - val_a
ccuracy: 0.9912 - val_loss: 0.0183
Epoch 10/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 40s 128ms/step - accuracy: 0.9970 - loss: 0.0077 - val_a
ccuracy: 0.9900 - val_loss: 0.0282
```

In [24]:

```
model.save('facemask_model_simple.keras')
```

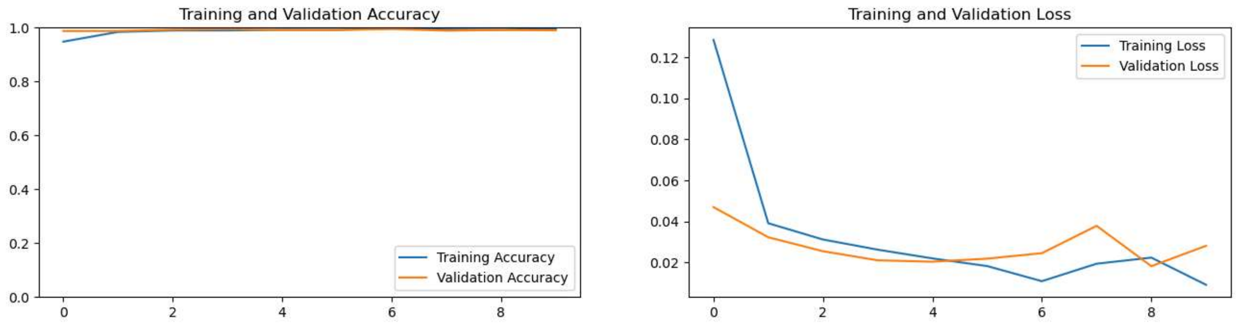
In [25]:

```
#from tensorflow.keras.models import load_model

#model = load_model('facemask_model_simple.keras')
#predictions = model.predict(test_ds)
```

Step 6: Evaluate the model

In [26]:

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(10)

plt.figure(figsize=(16, 8))
plt.subplot(2, 2, 1)
plt.ylim(0,1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



In [36]:

```
# Evaluate the model on the test dataset
test_loss, test_accuracy = model.evaluate(test_ds)

# Print the test loss and accuracy
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.9858 - loss: 0.0492
Test Loss: 0.051358141005039215
Test Accuracy: 0.9868951439857483
```

Step 7: Results

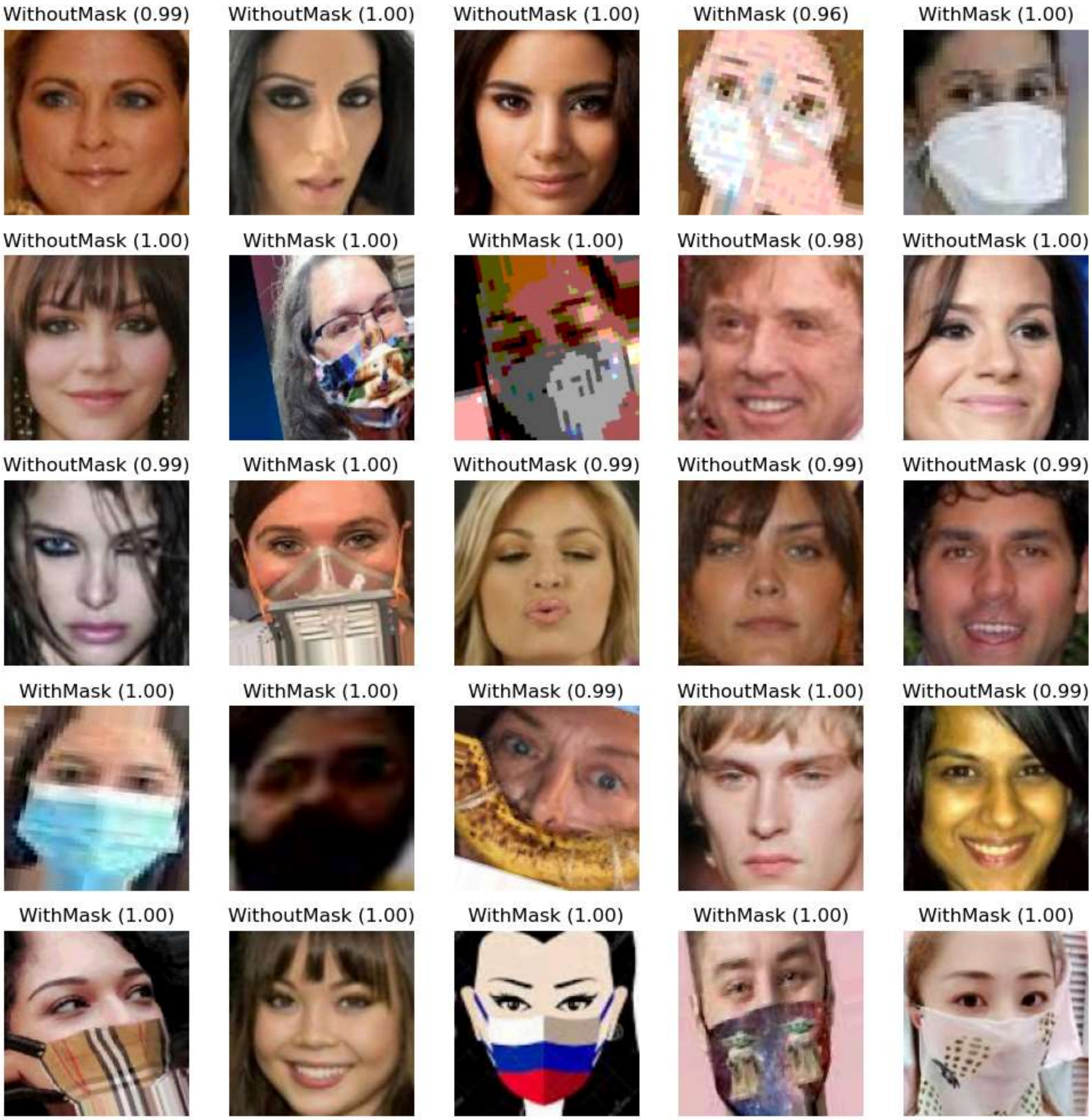
In [28]:

```
predictions = model.predict(test_ds)
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 2s 44ms/step
```

In [37]:

```
plt.figure(figsize=(10, 10))

num_images_to_plot = 25  # Limit of images to display
current_plot = 0  # Counter for images displayed

for images, labels in test_ds.take(19):
    if current_plot >= num_images_to_plot:
        break  # Stop when the plot limit is reached

    # Get predictions for the current batch
    batch_predictions = model(images, training=False)

    # Loop over each image in the batch
    for image, prediction in zip(images, batch_predictions):
        if current_plot >= num_images_to_plot:
            break  # Stop if we have reached the plot limit
        # Prepare image and prediction details
        image = image.numpy().astype("uint8")
        prediction_score = tf.nn.softmax(prediction)
        predicted_label = class_labels[tf.argmax(prediction_score).numpy()]
        confidence = tf.reduce_max(prediction_score).numpy()

        # Plot image with label and confidence
        plt.subplot(5, 5, current_plot + 1)
        plt.imshow(image)
        plt.axis("off")
        plt.title(f"{predicted_label} ({confidence:.2f})")

        current_plot += 1  # Increment plot count

plt.tight_layout()
plt.show()
```



Step 8: Improve model (optional)

In [30]:

```
data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal",
                      input_shape=(img_height,
                                   img_width,
                                   3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
  ]
)
```

In [ ]:

```
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
```

In [32]:

```
model = Sequential([
  data_augmentation,
  layers.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes, name="outputs")
])
```

In [33]:

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

In [ ]:

```
model.summary()
```

In [ ]:

```
epochs = 10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

In [ ]:

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)
```

```
plt.figure(figsize=(16, 8))
plt.subplot(2, 2, 1)
plt.ylim(0,1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

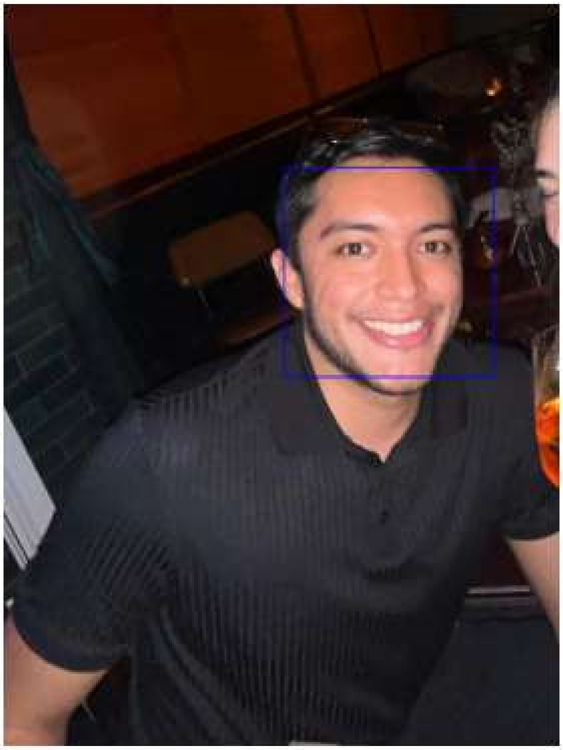
In [ ]:

```
plt.figure(figsize=(10, 10))

# Keep track of how many images are plotted
num_images_to_plot = 25
current_plot = 0

for batch_index, (images, labels) in enumerate(test_ds):
    # Perform predictions on the current batch of images
    batch_predictions = model(images, training=False)  # Get predictions directly from

    for image_index in range(images.shape[0]):
        if current_plot >= num_images_to_plot:
            break  # Stop if we have plotted 25 images

        plt.subplot(5, 5, current_plot + 1)

        # Extract the image
        image = images[image_index].numpy().astype("uint8")

        # Get the prediction and score for the current image
        prediction_score = tf.nn.softmax(batch_predictions[image_index])
        predicted_label = class_labels[np.argmax(prediction_score)]
        confidence = tf.reduce_max(prediction_score).numpy()  # Extract the highest co

        # Plot the image
        plt.imshow(image)
        plt.axis("off")

        # Set the title with the predicted label and confidence
        plt.title(f"{predicted_label} ({confidence:.2f})")

        current_plot += 1

    if current_plot >= num_images_to_plot:
        break  # Stop after plotting 25 images

plt.tight_layout()
plt.show()
```

Additional: Test own photos

In [45]:

```
import cv2
import matplotlib.pyplot as plt

img_path = pathlib.Path('C:/Users/ronal/Downloads/ronald.jpg') #Change path

# Cargar el modelo Haar Cascade para detección de rostros
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_

# Cargar la imagen
image = cv2.imread(img_path)
# Convertir la imagen a escala de grises, ya que Haar Cascade funciona mejor en grises
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Detectar rostros en la imagen
faces = face_cascade.detectMultiScale(gray_image, scaleFactor=1.1, minNeighbors=5, mir

# Dibujar un rectángulo alrededor de cada rostro detectado
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x+w, y+h), (255, 0, 0), 2)  # Rectángulo azul alrede

# Mostrar la imagen con los rostros detectados
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis('off')  # Quita los ejes
plt.show()
```



In [ ]:

```
# Check if any faces were detected
if len(faces) > 0:
    for (x, y, w, h) in faces:
        # Crop the face portion from the original image
        face = cv2.cvtColor(image[y:y+h, x:x+w], cv2.COLOR_BGR2RGB)
        face = cv2.resize(face,(128,128))
        # Display the cropped face
        plt.imshow(face)
        plt.axis('off')  # Hide axes
        plt.show()


else:
    print("No faces detected.")
```

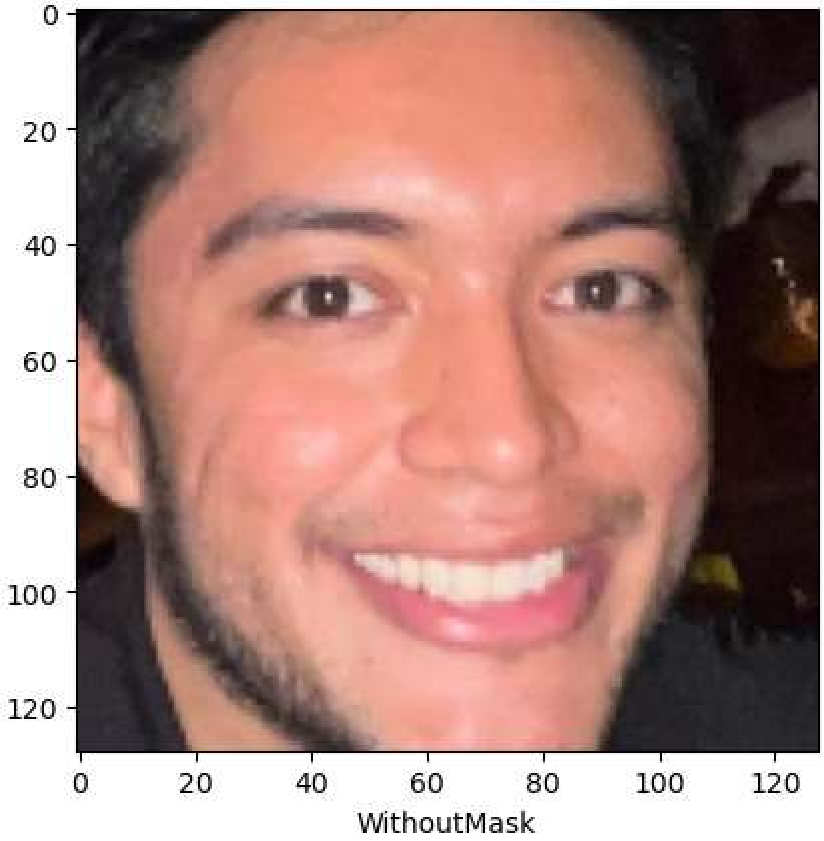
In [ ]:

```
index = 0  #Change index
face = cv2.cvtColor(image[faces[index][1]:faces[index][1]+faces[index][3], faces[index
face = cv2.resize(face,(128,128))
plt.imshow(face)
plt.axis('off')
plt.show()
```

In [42]:

```
img_array = tf.expand_dims(face, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

plt.figure()
plt.imshow(face)
plt.grid(False)
plt.xlabel(class_labels[np.argmax(score)])
plt.show()
print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_labels[np.argmax(score)], 100 * np.max(score))
)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 146ms/step
```



```
This image most likely belongs to WithoutMask with a 99.63 percent confidence.
```