Personal Project_04_v10_test1_3conv-layer_run09_advanced control

May 1, 2025

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib.image as mpimg
import tensorflow as tf
```

[2]:
```
# default initial values of DOE factors:
# learning_rate = 0.001
# dropout_value = 0.3
# #n-conv_layers = 3
# n_units_last_layer = 2048
# n_filters_l1 = 32
# n_filters_l2 = 16
```

[3]:
```
# DOE factors:
learning_rate = 0.0005
dropout_value = 0.2
# n-conv_layers = 4
n_units_last_layer = 1024
n_filters_l1 = 8
n_filters_l2 = 16
```

[4]:
```
# other factors:
img_size = 130
batch_size = 32
validation_split = 0.1  # 10% for validation
test_split = 0.00  # 0% for testing
shuffle_buffer_size = 1000
seed_num = 101
desired_accuracy = 0.99  # it should be active if EarlyStoppingCallback is␣
 ↪activated
loss = 'binary_crossentropy'
#optimizer = tf.keras.optimizers.RMSprop(learning_rate=learning_rate)
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
metrics = ['accuracy']
```

```python
epochs = 30
f_mode = 'nearest'  # fill_mode in image augmentation
```

```
My dataset_root/
   woman/
       woman_1.jpg
       woman_2.jpg
       ...
   man/
       man_1.jpg
       man_2.jpg
       ...
```

[6]:
```python
import os

DATA_DIR = "D:\\CS online courses\\Free DataSets\\Free Images\\Easier portrait␣
↪images_GPU_03"

# Subdirectories for each class
data_dir_woman = os.path.join(DATA_DIR, 'woman')
data_dir_man = os.path.join(DATA_DIR, 'man')

# os.listdir returns a list containing all files under the given dir
print(f"There are {len(os.listdir(data_dir_woman))} images of woman.")
print(f"There are {len(os.listdir(data_dir_man))} images of man.")
```

```
There are 471 images of woman.
There are 472 images of man.
```

[7]:
```python
image_size = (img_size, img_size)  # Resize images to this size

# Load train dataset (excluding validation & test set):
train_dataset = tf.keras.utils.image_dataset_from_directory(
    directory = DATA_DIR,
    image_size = image_size,
    batch_size = batch_size,
    label_mode='binary',
    validation_split = validation_split + test_split,  # Total split for val +␣
↪test
    subset = "training",
    seed = seed_num
)

# Load validation dataset
val_dataset = tf.keras.utils.image_dataset_from_directory(
    directory = DATA_DIR,
    image_size = image_size,
    batch_size = batch_size,
```

```
    label_mode='binary',
    validation_split = validation_split + test_split,
    subset = "validation",
    seed = seed_num
)

# Further manually split validation dataset to extract test dataset
val_batches = tf.data.experimental.cardinality(val_dataset)
# Compute test dataset size (number of batches)
test_size = round(val_batches.numpy() * (test_split / (validation_split +␣
 ↪test_split)))
# Split validation dataset into validation and test subsets
test_dataset = val_dataset.take(test_size)
val_dataset = val_dataset.skip(test_size)


print(f"Train batches: {tf.data.experimental.cardinality(train_dataset).
 ↪numpy()}")
print(f"Validation batches: {tf.data.experimental.cardinality(val_dataset).
 ↪numpy()}")
print(f"Test batches: {tf.data.experimental.cardinality(test_dataset).numpy()}")

# Optimize for performance
AUTOTUNE = tf.data.AUTOTUNE
training_dataset = train_dataset.cache().shuffle(shuffle_buffer_size).
 ↪prefetch(buffer_size = AUTOTUNE)
validation_dataset = val_dataset.cache().prefetch(buffer_size = AUTOTUNE)
test_dataset = test_dataset.cache().prefetch(buffer_size = AUTOTUNE)
```

```
Found 943 files belonging to 2 classes.
Using 849 files for training.
Found 943 files belonging to 2 classes.
Using 94 files for validation.
Train batches: 27
Validation batches: 3
Test batches: 0
```

[8]:
```
# Get the first batch of images and labels
for images, labels in training_dataset.take(1):
        example_batch_images = images
        example_batch_labels = labels

max_pixel = np.max(example_batch_images)
print(f"Maximum pixel value of images: {max_pixel}\n")
print(f"Shape of batch of images: {example_batch_images.shape}")
print(f"Shape of batch of labels: {example_batch_labels.shape}")
```

```
Maximum pixel value of images: 255.0
```

```
Shape of batch of images: (32, 130, 130, 3)
Shape of batch of labels: (32, 1)
```

[9]:
```
'''
class EarlyStoppingCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        train_accuracy = logs.get('accuracy')
        val_accuracy = logs.get('val_accuracy')
        if train_accuracy >= desired_accuracy and val_accuracy >=␣
↪desired_accuracy:
            self.model.stop_training = True
            print(f"\nReached {desired_accuracy}% accuracy so cancelling␣
↪training!")
'''
```

[9]:
```
'\nclass EarlyStoppingCallback(tf.keras.callbacks.Callback):\n    def
on_epoch_end(self, epoch, logs=None):\n        train_accuracy =
logs.get(\'accuracy\')\n        val_accuracy = logs.get(\'val_accuracy\')\n
if train_accuracy >= desired_accuracy and val_accuracy >= desired_accuracy:\n
self.model.stop_training = True\n            print(f"\nReached
{desired_accuracy}% accuracy so cancelling training!")\n'
```

[10]:
```
'''
from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(monitor='val_loss', patience=3)
'''
```

[10]:
```
"\nfrom tensorflow.keras.callbacks import EarlyStopping\nearly_stop =
EarlyStopping(monitor='val_loss', patience=3)\n"
```

[11]:
```
from tensorflow.keras.callbacks import LearningRateScheduler

# Reduce LR every 10 epochs (Learning rate decay factor)
def scheduler(epoch, lr):
    if epoch < 14:
        if epoch % 5 == 0 and epoch > 0:
            return lr / 1
        return lr
    elif epoch < 18:
        if epoch % 1 == 0 and epoch > 0:
            return lr / 1.5
        return lr
    else:
        if epoch % 2 == 0 and epoch > 0:
            return lr / 1
        return lr
```

```
lr_callback = LearningRateScheduler(scheduler)
```

[12]:
```
# augmentation_model
def augment_model():
    """Creates a model (layers stacked on top of each other) for augmenting␣
↪images of woman and man.

    Returns:
        tf.keras.Model: The model made up of the layers that will be used to␣
↪augment the images of woman and man.
    """

    augmentation_model = tf.keras.Sequential([
        # Specify the input shape.
        tf.keras.Input(shape = (img_size, img_size, 3)),

        tf.keras.layers.RandomFlip("horizontal"),
        tf.keras.layers.RandomRotation(0.1, fill_mode = f_mode),
        #tf.keras.layers.RandomTranslation(0.1, 0.1, fill_mode = f_mode),
        #tf.keras.layers.RandomZoom(0.1, fill_mode=f_mode)
        ])

    return augmentation_model
```

[13]:
```
def create_and_compile_model():
    """Creates, compiles and trains the model to predict woman and man images.

    Returns:
        tf.keras.Model: The model that will be trained to predict woman and man␣
↪images.
    """

    augmentation_layers = augment_model()

    model = tf.keras.Sequential([
        # Note: the input shape is the desired size of the image: 150x150 with␣
↪3 bytes for color
        tf.keras.layers.InputLayer(shape = (img_size, img_size, 3)),
        augmentation_layers,
        tf.keras.layers.Rescaling(1./255),
        #####    CONV_LAYER_1:     #####
        tf.keras.layers.Conv2D(n_filters_l1, (4, 4), activation = 'linear'),
        tf.keras.layers.MaxPooling2D(2, 2),
        #####    CONV_LAYER_2:     #####
        tf.keras.layers.Conv2D(n_filters_l2, (3, 3), activation = 'relu'),
        tf.keras.layers.MaxPooling2D(2, 2),
        #####    CONV_LAYER_3:     #####
```

```
        tf.keras.layers.Conv2D(64, (3, 3), activation = 'relu'),
        tf.keras.layers.MaxPooling2D(2, 2),
        #####    CONV_LAYER_4:     #####
        tf.keras.layers.Conv2D(64, (3, 3), activation = 'relu'),
        tf.keras.layers.MaxPooling2D(2, 2),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dropout(dropout_value),
        #####    BEFORE_LAST_LAYER:     #####
        tf.keras.layers.Dense(n_units_last_layer, activation = 'relu'),
        # It will contain a value from 0-1 where 0 for the class 'female' and 1␣
↪for the 'male'
        tf.keras.layers.Dense(1, activation = 'sigmoid')])

    model.compile(
        loss = loss,
        optimizer = optimizer,
        metrics = metrics
    )

    return model
```

[14]:

```
# Create the compiled but untrained model
model = create_and_compile_model()
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 130, 130, 3)	0
rescaling (Rescaling)	(None, 130, 130, 3)	0
conv2d (Conv2D)	(None, 127, 127, 8)	392
max_pooling2d (MaxPooling2D)	(None, 63, 63, 8)	0
conv2d_1 (Conv2D)	(None, 61, 61, 16)	1,168
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 16)	0
conv2d_2 (Conv2D)	(None, 28, 28, 64)	9,280
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_3 (Conv2D)	(None, 12, 12, 64)	36,928

max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten (Flatten)	(None, 2304)	0
dropout (Dropout)	(None, 2304)	0
dense (Dense)	(None, 1024)	2,360,320
dense_1 (Dense)	(None, 1)	1,025

Total params: 2,409,113 (9.19 MB)

Trainable params: 2,409,113 (9.19 MB)

Non-trainable params: 0 (0.00 B)

[15]:
```
'''
training_history = model.fit(
    training_dataset,
    epochs = epochs,
    validation_data = validation_dataset,
    callbacks = [EarlyStoppingCallback()],
    verbose = 2
)
'''
```

[15]:
```
'\ntraining_history = model.fit(\n    training_dataset,\n    epochs = epochs,\n
validation_data = validation_dataset,\n    callbacks =
[EarlyStoppingCallback()],\n    verbose = 2\n)\n'
```

[16]:
```
'''
training_history = model.fit(
    training_dataset,
    epochs = epochs,
    validation_data = validation_dataset,
    callbacks=[early_stop],
    verbose = 2
)
'''
```

[16]:
```
'\ntraining_history = model.fit(\n    training_dataset,\n    epochs = epochs,\n
validation_data = validation_dataset,\n    callbacks=[early_stop],\n    verbose
= 2\n)\n'
```

[17]:
```
training_history = model.fit(
    training_dataset,
    epochs = epochs,
    validation_data = validation_dataset,
    callbacks = [lr_callback],
    verbose = 2
)
```

```
Epoch 1/30
27/27 - 5s - 176ms/step - accuracy: 0.6137 - loss: 0.6667 - val_accuracy: 0.6170
- val_loss: 0.6887 - learning_rate: 5.0000e-04
Epoch 2/30
27/27 - 1s - 42ms/step - accuracy: 0.6820 - loss: 0.5860 - val_accuracy: 0.7021
- val_loss: 0.5557 - learning_rate: 5.0000e-04
Epoch 3/30
27/27 - 1s - 40ms/step - accuracy: 0.7468 - loss: 0.5223 - val_accuracy: 0.7234
- val_loss: 0.5557 - learning_rate: 5.0000e-04
Epoch 4/30
27/27 - 1s - 42ms/step - accuracy: 0.7397 - loss: 0.5315 - val_accuracy: 0.7766
- val_loss: 0.5035 - learning_rate: 5.0000e-04
Epoch 5/30
27/27 - 1s - 42ms/step - accuracy: 0.7750 - loss: 0.4777 - val_accuracy: 0.7872
- val_loss: 0.4600 - learning_rate: 5.0000e-04
Epoch 6/30
27/27 - 1s - 41ms/step - accuracy: 0.8009 - loss: 0.4457 - val_accuracy: 0.7979
- val_loss: 0.4652 - learning_rate: 5.0000e-04
Epoch 7/30
27/27 - 1s - 42ms/step - accuracy: 0.8045 - loss: 0.4380 - val_accuracy: 0.8085
- val_loss: 0.5899 - learning_rate: 5.0000e-04
Epoch 8/30
27/27 - 1s - 41ms/step - accuracy: 0.8163 - loss: 0.4022 - val_accuracy: 0.8404
- val_loss: 0.4322 - learning_rate: 5.0000e-04
Epoch 9/30
27/27 - 1s - 41ms/step - accuracy: 0.8269 - loss: 0.3897 - val_accuracy: 0.8085
- val_loss: 0.3945 - learning_rate: 5.0000e-04
Epoch 10/30
27/27 - 1s - 40ms/step - accuracy: 0.8445 - loss: 0.3609 - val_accuracy: 0.7979
- val_loss: 0.5708 - learning_rate: 5.0000e-04
Epoch 11/30
27/27 - 1s - 41ms/step - accuracy: 0.8221 - loss: 0.4036 - val_accuracy: 0.8404
- val_loss: 0.3594 - learning_rate: 5.0000e-04
Epoch 12/30
27/27 - 1s - 41ms/step - accuracy: 0.8504 - loss: 0.3433 - val_accuracy: 0.8191
- val_loss: 0.5030 - learning_rate: 5.0000e-04
Epoch 13/30
27/27 - 1s - 41ms/step - accuracy: 0.8645 - loss: 0.3021 - val_accuracy: 0.8191
- val_loss: 0.4028 - learning_rate: 5.0000e-04
Epoch 14/30
```

```
27/27 - 1s - 41ms/step - accuracy: 0.8481 - loss: 0.3103 - val_accuracy: 0.8404
- val_loss: 0.3303 - learning_rate: 5.0000e-04
Epoch 15/30
27/27 - 1s - 41ms/step - accuracy: 0.8810 - loss: 0.2803 - val_accuracy: 0.8298
- val_loss: 0.3563 - learning_rate: 3.3333e-04
Epoch 16/30
27/27 - 1s - 41ms/step - accuracy: 0.8810 - loss: 0.2671 - val_accuracy: 0.8511
- val_loss: 0.4314 - learning_rate: 2.2222e-04
Epoch 17/30
27/27 - 1s - 40ms/step - accuracy: 0.8869 - loss: 0.2470 - val_accuracy: 0.8723
- val_loss: 0.3237 - learning_rate: 1.4815e-04
Epoch 18/30
27/27 - 1s - 40ms/step - accuracy: 0.8975 - loss: 0.2423 - val_accuracy: 0.8511
- val_loss: 0.3578 - learning_rate: 9.8765e-05
Epoch 19/30
27/27 - 1s - 42ms/step - accuracy: 0.9105 - loss: 0.2197 - val_accuracy: 0.8511
- val_loss: 0.3564 - learning_rate: 9.8765e-05
Epoch 20/30
27/27 - 1s - 40ms/step - accuracy: 0.8975 - loss: 0.2503 - val_accuracy: 0.8298
- val_loss: 0.3416 - learning_rate: 9.8765e-05
Epoch 21/30
27/27 - 1s - 40ms/step - accuracy: 0.9176 - loss: 0.2337 - val_accuracy: 0.8404
- val_loss: 0.3284 - learning_rate: 9.8765e-05
Epoch 22/30
27/27 - 1s - 40ms/step - accuracy: 0.9022 - loss: 0.2429 - val_accuracy: 0.8511
- val_loss: 0.3060 - learning_rate: 9.8765e-05
Epoch 23/30
27/27 - 1s - 41ms/step - accuracy: 0.9164 - loss: 0.2284 - val_accuracy: 0.8723
- val_loss: 0.3959 - learning_rate: 9.8765e-05
Epoch 24/30
27/27 - 1s - 41ms/step - accuracy: 0.9011 - loss: 0.2229 - val_accuracy: 0.8723
- val_loss: 0.3413 - learning_rate: 9.8765e-05
Epoch 25/30
27/27 - 1s - 41ms/step - accuracy: 0.9117 - loss: 0.2291 - val_accuracy: 0.8617
- val_loss: 0.3129 - learning_rate: 9.8765e-05
Epoch 26/30
27/27 - 1s - 45ms/step - accuracy: 0.9140 - loss: 0.2164 - val_accuracy: 0.8617
- val_loss: 0.3721 - learning_rate: 9.8765e-05
Epoch 27/30
27/27 - 1s - 40ms/step - accuracy: 0.9140 - loss: 0.2239 - val_accuracy: 0.8617
- val_loss: 0.3253 - learning_rate: 9.8765e-05
Epoch 28/30
27/27 - 1s - 40ms/step - accuracy: 0.9081 - loss: 0.2298 - val_accuracy: 0.8617
- val_loss: 0.3794 - learning_rate: 9.8765e-05
Epoch 29/30
27/27 - 1s - 39ms/step - accuracy: 0.9105 - loss: 0.2231 - val_accuracy: 0.8936
- val_loss: 0.3838 - learning_rate: 9.8765e-05
Epoch 30/30
```

27/27 - 1s - 40ms/step - accuracy: 0.9117 - loss: 0.2160 - val_accuracy: 0.8723
- val_loss: 0.3404 - learning_rate: 9.8765e-05

[18]:
```
#from tensorflow.keras.models import load_model
#model.save('gender_recognition_project04_v10.h5')
```

[19]:
```
model.metrics_names
```

[19]: ['loss', 'compile_metrics']

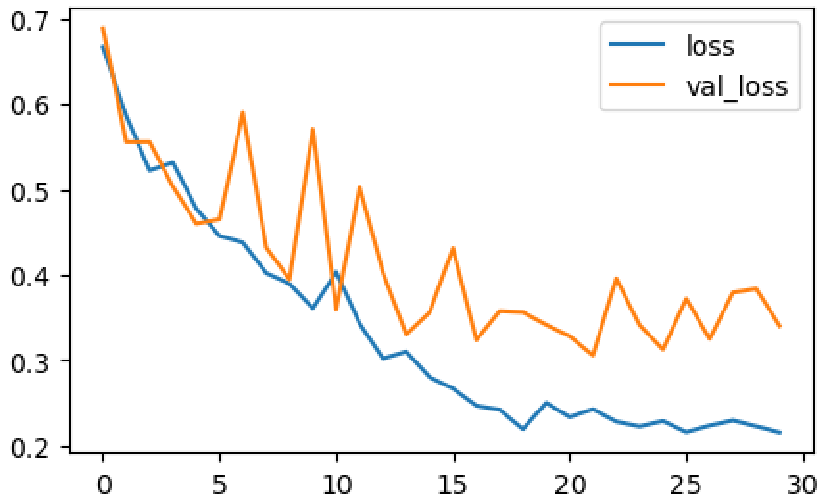
[20]:
```
result_history = pd.DataFrame(model.history.history)
result_history.head(25)
```

[20]:

	accuracy	loss	val_accuracy	val_loss	learning_rate
0	0.613663	0.666729	0.617021	0.688670	0.000500
1	0.681979	0.585953	0.702128	0.555670	0.000500
2	0.746761	0.522301	0.723404	0.555732	0.000500
3	0.739694	0.531549	0.776596	0.503451	0.000500
4	0.775029	0.477651	0.787234	0.459999	0.000500
5	0.800942	0.445673	0.797872	0.465195	0.000500
6	0.804476	0.438019	0.808511	0.589931	0.000500
7	0.816254	0.402218	0.840426	0.432239	0.000500
8	0.826855	0.389733	0.808511	0.394526	0.000500
9	0.844523	0.360938	0.797872	0.570776	0.000500
10	0.822144	0.403582	0.840426	0.359353	0.000500
11	0.850412	0.343250	0.819149	0.503042	0.000500
12	0.864547	0.302090	0.819149	0.402827	0.000500
13	0.848057	0.310331	0.840426	0.330251	0.000500
14	0.881037	0.280341	0.829787	0.356283	0.000333
15	0.881037	0.267092	0.851064	0.431377	0.000222
16	0.886926	0.246998	0.872340	0.323730	0.000148
17	0.897527	0.242314	0.851064	0.357756	0.000099
18	0.910483	0.219678	0.851064	0.356389	0.000099
19	0.897527	0.250350	0.829787	0.341629	0.000099
20	0.917550	0.233693	0.840426	0.328380	0.000099
21	0.902238	0.242896	0.851064	0.306003	0.000099
22	0.916372	0.228408	0.872340	0.395886	0.000099
23	0.901060	0.222949	0.872340	0.341313	0.000099
24	0.911661	0.229107	0.861702	0.312863	0.000099

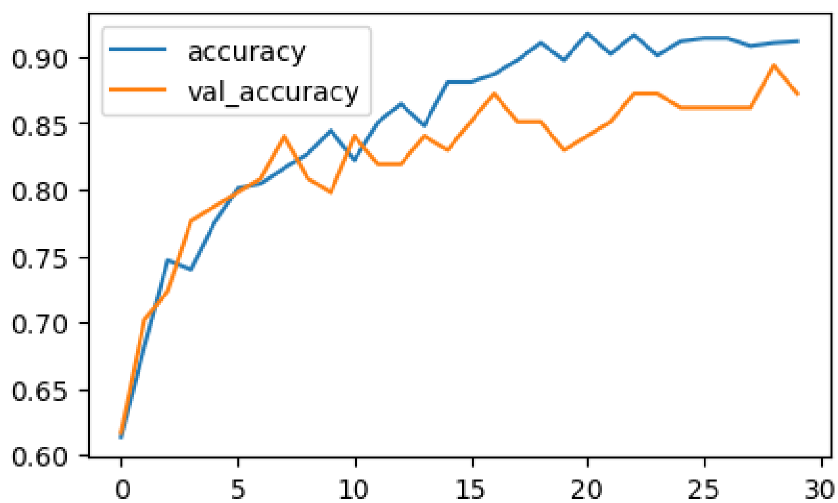
[21]:
```
result_history[['loss', 'val_loss']].plot(figsize=(5, 3))
```

[21]: <Axes: >



[22]:
```
result_history[['accuracy', 'val_accuracy']].plot(figsize=(5, 3))
```

[22]: <Axes: >



[23]:
```
print(model.metrics_names)
print(model.evaluate(validation_dataset))
```

```
['loss', 'compile_metrics']
3/3              0s 14ms/step -
accuracy: 0.8698 - loss: 0.3677
[0.3404359221458435, 0.8723404407501221]
```

[24]:
```
from sklearn.metrics import classification_report, confusion_matrix

y_true = np.concatenate([y.numpy() for _, y in validation_dataset])
y_pred_prob = model.predict(validation_dataset)
# Convert probabilities to class labels (0:Female or 1:Male)
y_pred = (y_pred_prob > 0.5).astype(int).flatten()

print("Classification Report:\n", classification_report(y_true, y_pred,␣
↪target_names=['Female', 'Male']))
```

```
3/3           0s 63ms/step
Classification Report:
               precision    recall  f1-score   support

      Female       0.82      0.90      0.86        41
        Male       0.92      0.85      0.88        53

    accuracy                           0.87        94
   macro avg       0.87      0.88      0.87        94
weighted avg       0.88      0.87      0.87        94
```

[25]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Model
from tensorflow.keras.utils import load_img, img_to_array

img_size = img_size
model = tf.keras.models.load_model("gender_recognition_project04_v10.h5")

# Load your personal image if you are interested to predict:
your_image_path = "D:\\Hossein's desktop files in Microsoft Studio␣
↪Laptop\\Personal Photos\\Hossein_10.jpg"

img = load_img(your_image_path, target_size=(img_size, img_size))
final_img = img_to_array(img)
# Adding a batch dimension:
final_img = np.expand_dims(final_img, axis=0)
prediction = model.predict(final_img)
result = "Female" if prediction > 0.5 else "Male"
if result=="Female":
    confidence = (model.predict(final_img)[0][0])*100
else:
    confidence = (1-model.predict(final_img)[0][0])*100
print(f"Prediction result: {result} (confidence= {confidence:.2f} %)")
```

```python
# Visualize CNN Layers
successive_feature_maps = visualization_model.predict(final_img)
layer_names = [layer.name for layer in model.layers]

for layer_name, feature_map in zip(layer_names, successive_feature_maps):
    if len(feature_map.shape) == 4:  # Only visualize conv/maxpool layers
        n_features = feature_map.shape[-1]  # Number of filters
        size = feature_map.shape[1]  # Feature map size
        display_grid = np.zeros((size, size * n_features))

        for i in range(n_features):
            x = feature_map[0, :, :, i]
            x -= x.mean()
            x /= (x.std() + 1e-8)  # Normalize
            x *= 64
            x += 128
            x = np.clip(x, 0, 255).astype('uint8')  # Convert to image format
            display_grid[:, i * size: (i + 1) * size] = x

        scale = 20. / n_features
        plt.figure(figsize=(scale * n_features, scale))
        plt.title(layer_name)
        plt.grid(False)
        plt.imshow(display_grid, aspect='auto', cmap='cividis')
        plt.show()
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.

1/1              0s 124ms/step
1/1              0s 69ms/step
Prediction result: Male (confidence= 94.19 %)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[25], line 26
     23 print(f"Prediction result: {result} (confidence= {confidence:.2f} %)")
     25 # Visualize CNN Layers
---> 26 successive_feature_maps = visualization_model.predict(final_img)
     27 layer_names = [layer.name for layer in model.layers]
     29 for layer_name, feature_map in zip(layer_names, successive_feature_maps):

NameError: name 'visualization_model' is not defined
```

[ ]:

```
[ ]:
```

```
[ ]:
```