Personal Project_04_v10_test1_5conv-layer-Copy1

April 30, 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 = 0.001
dropout_value = 0.3
# n-conv_layers = 5
n_units_last_layer = 2048
n_filters_l1 = 16
n_filters_l2 = 32
```

[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']
epochs = 100
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 < 20:
        if epoch < 15:
            if epoch % 10 == 0 and epoch > 0:
                #return lr * 0.1
                return lr / 2
            return lr
        elif epoch % 10 == 0 and epoch > 0:
            #return lr * 0.1
            return lr / 5
        return lr
    elif epoch % 10 == 0 and epoch > 0:
        #return lr * 0.1
        return lr / 10
    return lr

lr_callback = LearningRateScheduler(scheduler)
```

[12]:
```python
# 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]:
```python
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),
        #####    CONV_LAYER_5:     #####
        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, 16)	784
max_pooling2d (MaxPooling2D)	(None, 63, 63, 16)	0
conv2d_1 (Conv2D)	(None, 61, 61, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 32)	0
conv2d_2 (Conv2D)	(None, 28, 28, 64)	18,496
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

conv2d_4 (Conv2D)                (None, 4, 4, 64)           36,928

max_pooling2d_4 (MaxPooling2D)   (None, 2, 2, 64)                0

flatten (Flatten)                (None, 256)                     0

dropout (Dropout)                (None, 256)                     0

dense (Dense)                    (None, 2048)              526,336

dense_1 (Dense)                  (None, 1)                   2,049
```

Total params: 626,161 (2.39 MB)

Trainable params: 626,161 (2.39 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 1/100
27/27 - 3s - 116ms/step - accuracy: 0.4888 - loss: 0.6975 - val_accuracy: 0.5638
- val_loss: 0.6900 - learning_rate: 0.0010
Epoch 2/100
27/27 - 1s - 42ms/step - accuracy: 0.5406 - loss: 0.6868 - val_accuracy: 0.4362
- val_loss: 0.7054 - learning_rate: 0.0010
Epoch 3/100
27/27 - 1s - 43ms/step - accuracy: 0.6631 - loss: 0.6344 - val_accuracy: 0.6702
- val_loss: 0.6337 - learning_rate: 0.0010
Epoch 4/100
27/27 - 1s - 45ms/step - accuracy: 0.6949 - loss: 0.5823 - val_accuracy: 0.6596
- val_loss: 0.6103 - learning_rate: 0.0010
Epoch 5/100
27/27 - 1s - 44ms/step - accuracy: 0.7244 - loss: 0.5699 - val_accuracy: 0.6702
- val_loss: 0.6025 - learning_rate: 0.0010
Epoch 6/100
27/27 - 1s - 43ms/step - accuracy: 0.7550 - loss: 0.5296 - val_accuracy: 0.7234
- val_loss: 0.5198 - learning_rate: 0.0010
Epoch 7/100
27/27 - 1s - 43ms/step - accuracy: 0.7644 - loss: 0.4999 - val_accuracy: 0.7660
- val_loss: 0.5145 - learning_rate: 0.0010
Epoch 8/100
27/27 - 1s - 46ms/step - accuracy: 0.7585 - loss: 0.5229 - val_accuracy: 0.7660
- val_loss: 0.5214 - learning_rate: 0.0010
Epoch 9/100
27/27 - 1s - 46ms/step - accuracy: 0.7538 - loss: 0.4912 - val_accuracy: 0.7872
- val_loss: 0.5634 - learning_rate: 0.0010
Epoch 10/100
27/27 - 1s - 47ms/step - accuracy: 0.7927 - loss: 0.4641 - val_accuracy: 0.8298
- val_loss: 0.4318 - learning_rate: 0.0010
Epoch 11/100
27/27 - 1s - 45ms/step - accuracy: 0.8269 - loss: 0.4014 - val_accuracy: 0.8511
- val_loss: 0.3682 - learning_rate: 5.0000e-04
Epoch 12/100
```

```
27/27 - 1s - 46ms/step - accuracy: 0.8163 - loss: 0.4049 - val_accuracy: 0.8404
- val_loss: 0.4428 - learning_rate: 5.0000e-04
Epoch 13/100
27/27 - 1s - 43ms/step - accuracy: 0.8363 - loss: 0.3854 - val_accuracy: 0.8511
- val_loss: 0.3520 - learning_rate: 5.0000e-04
Epoch 14/100
27/27 - 1s - 43ms/step - accuracy: 0.8492 - loss: 0.3628 - val_accuracy: 0.8511
- val_loss: 0.3256 - learning_rate: 5.0000e-04
Epoch 15/100
27/27 - 1s - 43ms/step - accuracy: 0.8304 - loss: 0.3717 - val_accuracy: 0.8936
- val_loss: 0.3001 - learning_rate: 5.0000e-04
Epoch 16/100
27/27 - 1s - 43ms/step - accuracy: 0.8516 - loss: 0.3747 - val_accuracy: 0.8511
- val_loss: 0.3405 - learning_rate: 5.0000e-04
Epoch 17/100
27/27 - 1s - 43ms/step - accuracy: 0.8481 - loss: 0.3418 - val_accuracy: 0.8617
- val_loss: 0.4021 - learning_rate: 5.0000e-04
Epoch 18/100
27/27 - 1s - 45ms/step - accuracy: 0.8634 - loss: 0.3230 - val_accuracy: 0.8617
- val_loss: 0.3437 - learning_rate: 5.0000e-04
Epoch 19/100
27/27 - 1s - 43ms/step - accuracy: 0.8433 - loss: 0.3539 - val_accuracy: 0.8830
- val_loss: 0.2870 - learning_rate: 5.0000e-04
Epoch 20/100
27/27 - 1s - 42ms/step - accuracy: 0.8728 - loss: 0.3065 - val_accuracy: 0.9043
- val_loss: 0.2947 - learning_rate: 5.0000e-04
Epoch 21/100
27/27 - 1s - 46ms/step - accuracy: 0.8751 - loss: 0.2799 - val_accuracy: 0.8723
- val_loss: 0.3039 - learning_rate: 5.0000e-05
Epoch 22/100
27/27 - 1s - 46ms/step - accuracy: 0.8881 - loss: 0.2736 - val_accuracy: 0.8617
- val_loss: 0.2974 - learning_rate: 5.0000e-05
Epoch 23/100
27/27 - 1s - 49ms/step - accuracy: 0.8693 - loss: 0.2917 - val_accuracy: 0.8830
- val_loss: 0.2915 - learning_rate: 5.0000e-05
Epoch 24/100
27/27 - 1s - 46ms/step - accuracy: 0.8857 - loss: 0.2651 - val_accuracy: 0.8936
- val_loss: 0.2894 - learning_rate: 5.0000e-05
Epoch 25/100
27/27 - 1s - 43ms/step - accuracy: 0.8940 - loss: 0.2575 - val_accuracy: 0.8617
- val_loss: 0.2972 - learning_rate: 5.0000e-05
Epoch 26/100
27/27 - 1s - 43ms/step - accuracy: 0.8881 - loss: 0.2862 - val_accuracy: 0.8936
- val_loss: 0.2774 - learning_rate: 5.0000e-05
Epoch 27/100
27/27 - 1s - 44ms/step - accuracy: 0.8963 - loss: 0.2640 - val_accuracy: 0.8830
- val_loss: 0.2864 - learning_rate: 5.0000e-05
Epoch 28/100
```

```
27/27 - 1s - 43ms/step - accuracy: 0.8905 - loss: 0.2519 - val_accuracy: 0.8830
- val_loss: 0.2833 - learning_rate: 5.0000e-05
Epoch 29/100
27/27 - 1s - 43ms/step - accuracy: 0.9034 - loss: 0.2560 - val_accuracy: 0.8936
- val_loss: 0.2816 - learning_rate: 5.0000e-05
Epoch 30/100
27/27 - 1s - 43ms/step - accuracy: 0.9034 - loss: 0.2554 - val_accuracy: 0.8723
- val_loss: 0.2932 - learning_rate: 5.0000e-05
Epoch 31/100
27/27 - 1s - 43ms/step - accuracy: 0.8987 - loss: 0.2553 - val_accuracy: 0.8830
- val_loss: 0.2910 - learning_rate: 5.0000e-06
Epoch 32/100
27/27 - 1s - 43ms/step - accuracy: 0.8857 - loss: 0.2610 - val_accuracy: 0.8830
- val_loss: 0.2905 - learning_rate: 5.0000e-06
Epoch 33/100
27/27 - 1s - 47ms/step - accuracy: 0.8940 - loss: 0.2570 - val_accuracy: 0.8830
- val_loss: 0.2884 - learning_rate: 5.0000e-06
Epoch 34/100
27/27 - 1s - 51ms/step - accuracy: 0.8893 - loss: 0.2586 - val_accuracy: 0.8830
- val_loss: 0.2862 - learning_rate: 5.0000e-06
Epoch 35/100
27/27 - 1s - 50ms/step - accuracy: 0.8893 - loss: 0.2669 - val_accuracy: 0.8936
- val_loss: 0.2882 - learning_rate: 5.0000e-06
Epoch 36/100
27/27 - 1s - 47ms/step - accuracy: 0.9011 - loss: 0.2416 - val_accuracy: 0.8936
- val_loss: 0.2863 - learning_rate: 5.0000e-06
Epoch 37/100
27/27 - 1s - 46ms/step - accuracy: 0.9011 - loss: 0.2408 - val_accuracy: 0.8830
- val_loss: 0.2847 - learning_rate: 5.0000e-06
Epoch 38/100
27/27 - 1s - 44ms/step - accuracy: 0.8975 - loss: 0.2581 - val_accuracy: 0.8830
- val_loss: 0.2836 - learning_rate: 5.0000e-06
Epoch 39/100
27/27 - 1s - 44ms/step - accuracy: 0.8822 - loss: 0.2641 - val_accuracy: 0.8936
- val_loss: 0.2846 - learning_rate: 5.0000e-06
Epoch 40/100
27/27 - 1s - 46ms/step - accuracy: 0.8869 - loss: 0.2693 - val_accuracy: 0.8936
- val_loss: 0.2853 - learning_rate: 5.0000e-06
Epoch 41/100
27/27 - 1s - 48ms/step - accuracy: 0.8822 - loss: 0.2707 - val_accuracy: 0.8936
- val_loss: 0.2850 - learning_rate: 5.0000e-07
Epoch 42/100
27/27 - 1s - 46ms/step - accuracy: 0.8893 - loss: 0.2517 - val_accuracy: 0.8936
- val_loss: 0.2851 - learning_rate: 5.0000e-07
Epoch 43/100
27/27 - 1s - 46ms/step - accuracy: 0.8822 - loss: 0.2510 - val_accuracy: 0.8936
- val_loss: 0.2850 - learning_rate: 5.0000e-07
Epoch 44/100
```

```
27/27 - 1s - 49ms/step - accuracy: 0.8999 - loss: 0.2591 - val_accuracy: 0.8936
- val_loss: 0.2848 - learning_rate: 5.0000e-07
Epoch 45/100
27/27 - 1s - 48ms/step - accuracy: 0.8905 - loss: 0.2581 - val_accuracy: 0.8936
- val_loss: 0.2847 - learning_rate: 5.0000e-07
Epoch 46/100
27/27 - 1s - 46ms/step - accuracy: 0.8963 - loss: 0.2666 - val_accuracy: 0.8936
- val_loss: 0.2846 - learning_rate: 5.0000e-07
Epoch 47/100
27/27 - 1s - 48ms/step - accuracy: 0.8999 - loss: 0.2491 - val_accuracy: 0.8936
- val_loss: 0.2846 - learning_rate: 5.0000e-07
Epoch 48/100
27/27 - 1s - 51ms/step - accuracy: 0.8975 - loss: 0.2605 - val_accuracy: 0.8936
- val_loss: 0.2844 - learning_rate: 5.0000e-07
Epoch 49/100
27/27 - 1s - 50ms/step - accuracy: 0.8857 - loss: 0.2674 - val_accuracy: 0.8936
- val_loss: 0.2842 - learning_rate: 5.0000e-07
Epoch 50/100
27/27 - 1s - 48ms/step - accuracy: 0.9069 - loss: 0.2446 - val_accuracy: 0.8936
- val_loss: 0.2842 - learning_rate: 5.0000e-07
Epoch 51/100
27/27 - 1s - 45ms/step - accuracy: 0.8963 - loss: 0.2694 - val_accuracy: 0.8936
- val_loss: 0.2841 - learning_rate: 5.0000e-08
Epoch 52/100
27/27 - 1s - 46ms/step - accuracy: 0.8940 - loss: 0.2564 - val_accuracy: 0.8936
- val_loss: 0.2841 - learning_rate: 5.0000e-08
Epoch 53/100
27/27 - 1s - 45ms/step - accuracy: 0.8916 - loss: 0.2504 - val_accuracy: 0.8936
- val_loss: 0.2841 - learning_rate: 5.0000e-08
Epoch 54/100
27/27 - 1s - 46ms/step - accuracy: 0.8916 - loss: 0.2503 - val_accuracy: 0.8936
- val_loss: 0.2841 - learning_rate: 5.0000e-08
Epoch 55/100
27/27 - 1s - 46ms/step - accuracy: 0.9011 - loss: 0.2466 - val_accuracy: 0.8936
- val_loss: 0.2841 - learning_rate: 5.0000e-08
Epoch 56/100
27/27 - 1s - 45ms/step - accuracy: 0.8999 - loss: 0.2585 - val_accuracy: 0.8936
- val_loss: 0.2841 - learning_rate: 5.0000e-08
Epoch 57/100
27/27 - 1s - 45ms/step - accuracy: 0.9011 - loss: 0.2379 - val_accuracy: 0.8936
- val_loss: 0.2841 - learning_rate: 5.0000e-08
Epoch 58/100
27/27 - 1s - 46ms/step - accuracy: 0.8928 - loss: 0.2587 - val_accuracy: 0.8936
- val_loss: 0.2841 - learning_rate: 5.0000e-08
Epoch 59/100
27/27 - 1s - 46ms/step - accuracy: 0.8963 - loss: 0.2505 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-08
Epoch 60/100
```

```
27/27 - 1s - 49ms/step - accuracy: 0.8905 - loss: 0.2524 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-08
Epoch 61/100
27/27 - 1s - 49ms/step - accuracy: 0.9022 - loss: 0.2532 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-09
Epoch 62/100
27/27 - 1s - 50ms/step - accuracy: 0.9022 - loss: 0.2580 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-09
Epoch 63/100
27/27 - 1s - 47ms/step - accuracy: 0.8905 - loss: 0.2535 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-09
Epoch 64/100
27/27 - 1s - 45ms/step - accuracy: 0.8928 - loss: 0.2421 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-09
Epoch 65/100
27/27 - 1s - 46ms/step - accuracy: 0.8940 - loss: 0.2467 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-09
Epoch 66/100
27/27 - 1s - 46ms/step - accuracy: 0.9022 - loss: 0.2371 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-09
Epoch 67/100
27/27 - 1s - 45ms/step - accuracy: 0.8975 - loss: 0.2512 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-09
Epoch 68/100
27/27 - 1s - 46ms/step - accuracy: 0.8963 - loss: 0.2506 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-09
Epoch 69/100
27/27 - 1s - 46ms/step - accuracy: 0.9022 - loss: 0.2312 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-09
Epoch 70/100
27/27 - 1s - 48ms/step - accuracy: 0.9128 - loss: 0.2462 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-09
Epoch 71/100
27/27 - 1s - 47ms/step - accuracy: 0.9081 - loss: 0.2426 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-10
Epoch 72/100
27/27 - 1s - 47ms/step - accuracy: 0.8940 - loss: 0.2706 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-10
Epoch 73/100
27/27 - 1s - 51ms/step - accuracy: 0.9058 - loss: 0.2472 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-10
Epoch 74/100
27/27 - 1s - 54ms/step - accuracy: 0.8940 - loss: 0.2527 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-10
Epoch 75/100
27/27 - 1s - 51ms/step - accuracy: 0.8893 - loss: 0.2606 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-10
Epoch 76/100
```

```
27/27 - 1s - 46ms/step - accuracy: 0.9058 - loss: 0.2279 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-10
Epoch 77/100
27/27 - 1s - 46ms/step - accuracy: 0.8928 - loss: 0.2612 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-10
Epoch 78/100
27/27 - 1s - 48ms/step - accuracy: 0.8775 - loss: 0.2768 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-10
Epoch 79/100
27/27 - 1s - 46ms/step - accuracy: 0.8893 - loss: 0.2516 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-10
Epoch 80/100
27/27 - 1s - 46ms/step - accuracy: 0.9046 - loss: 0.2451 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-10
Epoch 81/100
27/27 - 1s - 46ms/step - accuracy: 0.8893 - loss: 0.2491 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-11
Epoch 82/100
27/27 - 1s - 46ms/step - accuracy: 0.8834 - loss: 0.2630 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-11
Epoch 83/100
27/27 - 1s - 46ms/step - accuracy: 0.8963 - loss: 0.2604 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-11
Epoch 84/100
27/27 - 1s - 45ms/step - accuracy: 0.8940 - loss: 0.2529 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-11
Epoch 85/100
27/27 - 1s - 50ms/step - accuracy: 0.8881 - loss: 0.2530 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-11
Epoch 86/100
27/27 - 1s - 49ms/step - accuracy: 0.8846 - loss: 0.2639 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-11
Epoch 87/100
27/27 - 1s - 48ms/step - accuracy: 0.8963 - loss: 0.2574 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-11
Epoch 88/100
27/27 - 1s - 48ms/step - accuracy: 0.8940 - loss: 0.2453 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-11
Epoch 89/100
27/27 - 1s - 45ms/step - accuracy: 0.8810 - loss: 0.2711 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-11
Epoch 90/100
27/27 - 1s - 45ms/step - accuracy: 0.8822 - loss: 0.2577 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-11
Epoch 91/100
27/27 - 1s - 45ms/step - accuracy: 0.9022 - loss: 0.2483 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-12
Epoch 92/100
```

```
27/27 - 1s - 45ms/step - accuracy: 0.8881 - loss: 0.2500 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-12
Epoch 93/100
27/27 - 1s - 45ms/step - accuracy: 0.8940 - loss: 0.2529 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-12
Epoch 94/100
27/27 - 1s - 46ms/step - accuracy: 0.8940 - loss: 0.2391 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-12
Epoch 95/100
27/27 - 1s - 47ms/step - accuracy: 0.9022 - loss: 0.2502 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-12
Epoch 96/100
27/27 - 1s - 47ms/step - accuracy: 0.8928 - loss: 0.2543 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-12
Epoch 97/100
27/27 - 1s - 47ms/step - accuracy: 0.8928 - loss: 0.2577 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-12
Epoch 98/100
27/27 - 1s - 49ms/step - accuracy: 0.8810 - loss: 0.2772 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-12
Epoch 99/100
27/27 - 1s - 50ms/step - accuracy: 0.8881 - loss: 0.2635 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-12
Epoch 100/100
27/27 - 1s - 49ms/step - accuracy: 0.8893 - loss: 0.2637 - val_accuracy: 0.8936
- val_loss: 0.2840 - learning_rate: 5.0000e-12
```

[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(15)
```

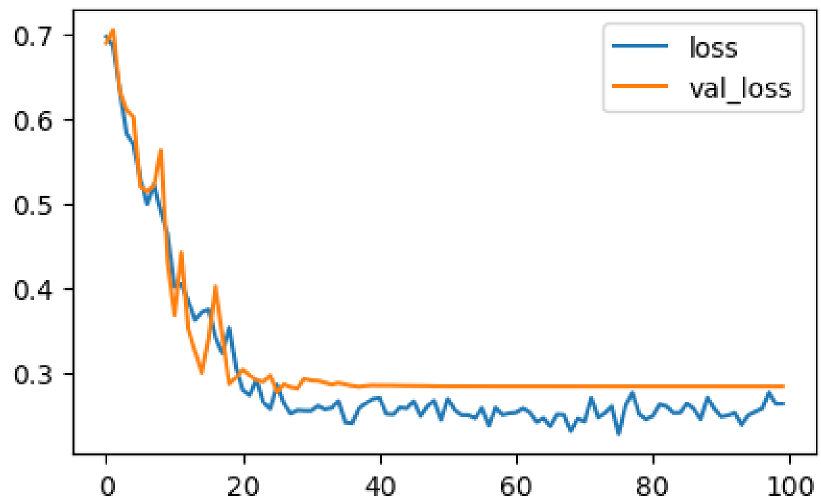
[20]:

	accuracy	loss	val_accuracy	val_loss	learning_rate
0	0.488810	0.697535	0.563830	0.689982	0.0010
1	0.540636	0.686790	0.436170	0.705353	0.0010
2	0.663133	0.634382	0.670213	0.633700	0.0010
3	0.694935	0.582258	0.659574	0.610273	0.0010
4	0.724382	0.569885	0.670213	0.602472	0.0010
5	0.755006	0.529599	0.723404	0.519779	0.0010
6	0.764429	0.499851	0.765957	0.514465	0.0010
7	0.758539	0.522868	0.765957	0.521371	0.0010
8	0.753828	0.491202	0.787234	0.563364	0.0010
9	0.792697	0.464055	0.829787	0.431788	0.0010

```
10  0.826855  0.401430      0.851064  0.368193          0.0005
11  0.816254  0.404856      0.840426  0.442838          0.0005
12  0.836278  0.385424      0.851064  0.352016          0.0005
13  0.849234  0.362829      0.851064  0.325566          0.0005
14  0.830389  0.371654      0.893617  0.300061          0.0005
```

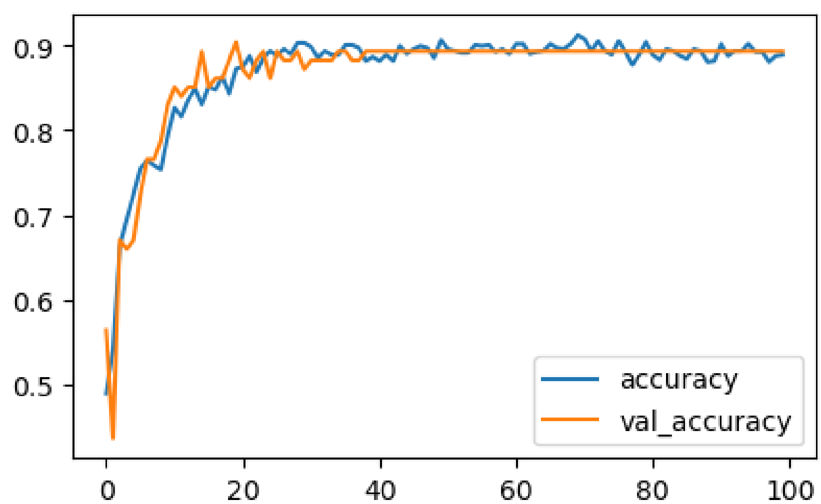
[21]:
```
result_history[['loss', 'val_loss']].plot(figsize=(5, 3))
```

[21]: <Axes: >



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

[22]: <Axes: >



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

```
['loss', 'compile_metrics']
3/3              0s 17ms/step -
accuracy: 0.8843 - loss: 0.2957
[0.2840394079685211, 0.8936170339584351]
```

[24]:
```python
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 47ms/step
Classification Report:
               precision    recall  f1-score   support

      Female       0.83      0.95      0.89        41
        Male       0.96      0.85      0.90        53

    accuracy                           0.89        94
   macro avg       0.89      0.90      0.89        94
weighted avg       0.90      0.89      0.89        94
```

[25]:
```python
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:
```

```python
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} %)")

# 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 88ms/step
1/1              0s 46ms/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
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```