چکیده:

در این پروژه، با استفاده از کتابخانه‌ی tensorflow و نیز ماژول keras که به صورت گسترده در حوزه‌ی یادگیری ماشین، مورد استفاده قرار می‌گیرند. بهره گرفته شده است.
با یک binary image classification روبه‌رو هستیم.

از داده های ارائه شده در فایل فشرده، مورد 1 و مورد 8 را به عنوان 2 شخص مساله انتخاب کرده‌ام.

طبق خواسته‌ی پروژه، زیرمجموعه 2 برای ارزیابی یا test_set و باقی تصاویر برای train_set استفاده شدند.

یک شبکه کانولوشنی چند لایه ایجاد و نهایتا خروجی توسط یک لایه Dense یا Full Connection layer مشخص می‌شود.

از Adam به عنوان optimizer استفاده و learning_rate برابر 0.001 قرار داده شده است.

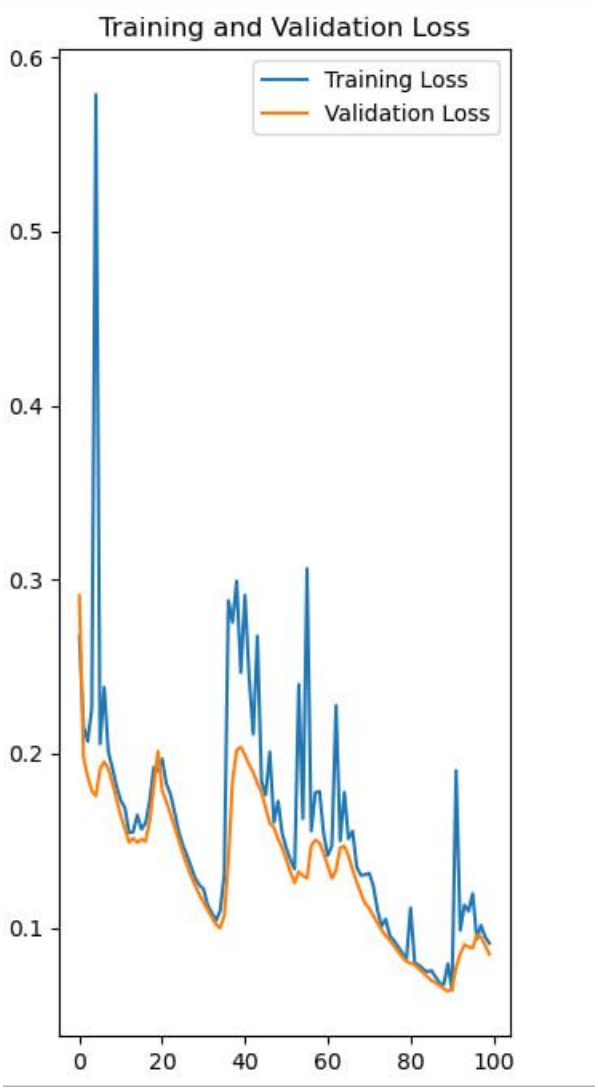
L2Regularizer و سایر ثابت ها نیز طبق فایل توضیحات پروژه، ترتیب اثر داده شده اند.

کد پیاده‌سازی نیز ضمیمه شده است.

*نکته:

فایل convertor.py موجود در ضمیمه‌ها، به‌منظور تبدیل فرمت تصاویر از tif. به jpg. می‌باشد.

نمودار Loss شبکه (train & test)



شبکه‌ی پیاده‌سازی شده

```
In [183]: model.summary()
```

```
Model: "sequential_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_14 (Sequential)  (None, 160, 160, 3)       0

 rescaling_17 (Rescaling)    (None, 160, 160, 3)       0

 conv2d_45 (Conv2D)          (None, 158, 158, 16)      448

 max_pooling2d_30 (MaxPooli  (None, 39, 39, 16)        0
 ng2D)

 batch_normalization_45 (Ba  (None, 39, 39, 16)        64
 tchNormalization)

 conv2d_46 (Conv2D)          (None, 37, 37, 32)        4640

 max_pooling2d_31 (MaxPooli  (None, 18, 18, 32)        0
 ng2D)

 batch_normalization_46 (Ba  (None, 18, 18, 32)        128
 tchNormalization)

 conv2d_47 (Conv2D)          (None, 16, 16, 64)        18496

 average_pooling2d_15 (Aver  (None, 8, 8, 64)          0
 agePooling2D)

 batch_normalization_47 (Ba  (None, 8, 8, 64)          256
 tchNormalization)

 dropout_14 (Dropout)        (None, 8, 8, 64)          0

 flatten_15 (Flatten)        (None, 4096)              0

 dense_31 (Dense)            (None, 256)               1048832

 dense_32 (Dense)            (None, 148)               38036

 output (Dense)              (None, 1)                 149

=================================================================
Total params: 1111049 (4.24 MB)
Trainable params: 1110825 (4.24 MB)
Non-trainable params: 224 (896.00 Byte)
```

همانطور که قابل مشاهده است، روی لایه ها، BatchNormalization اعمال شده، در دو لایه‌ی اول MaxPooling و در لایه‌ی سوم از AveragePooling استفاده شده است.

Dropout نیز برای جلوگیری از overfit اعمال شد.

تصویری از کد مربوط به افزودن لایه های مختلف نیز، در ادامه آمده است.

re-create model (augmented and dropout)

```
In [181]:
num_classes = len(class_names)
model = Sequential([
  data_augmentation,

  layers.Rescaling(1./255, input_shape=(img_h, img_w, 3)),

  layers.Conv2D(16, 3, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(L2regularization), bias_regularizer=tf.kera
  layers.MaxPooling2D(pool_size = 4),
  layers.BatchNormalization(),

  layers.Conv2D(32, 3, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(L2regularization), bias_regularizer=tf.kera
  layers.MaxPooling2D(pool_size = 2),
  layers.BatchNormalization(),

  layers.Conv2D(64, 3, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(L2regularization), bias_regularizer=tf.kera
  layers.AveragePooling2D(pool_size = 2),
  layers.BatchNormalization(),

  layers.Dropout(0.1),

  layers.Flatten(),

  layers.Dense(256, kernel_regularizer=tf.keras.regularizers.l2(L2regularization), activation='relu'),
  layers.Dense(148, kernel_regularizer=tf.keras.regularizers.l2(L2regularization), activation='relu'),
  layers.Dense(1, activation='sigmoid', name="output")
])
```

تصویری از خروجی برای عکسی از مورد 8 که با توجه به نحوه پیاده‌سازی شبکه، به person_2 کلاس‌بندی شده است:

Part 4 - Making a single prediction

```
In [40]:
import numpy as np
from sklearn.metrics import f1_score

img = tf.keras.utils.load_img(
    'dataset/checking/p1_or_p2_2.jpg', target_size=(img_h, img_w)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

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

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

```
1/1 [==============================] - 0s 38ms/step
This image most likely belongs to person_2 with a 72.93 percent confidence.
```

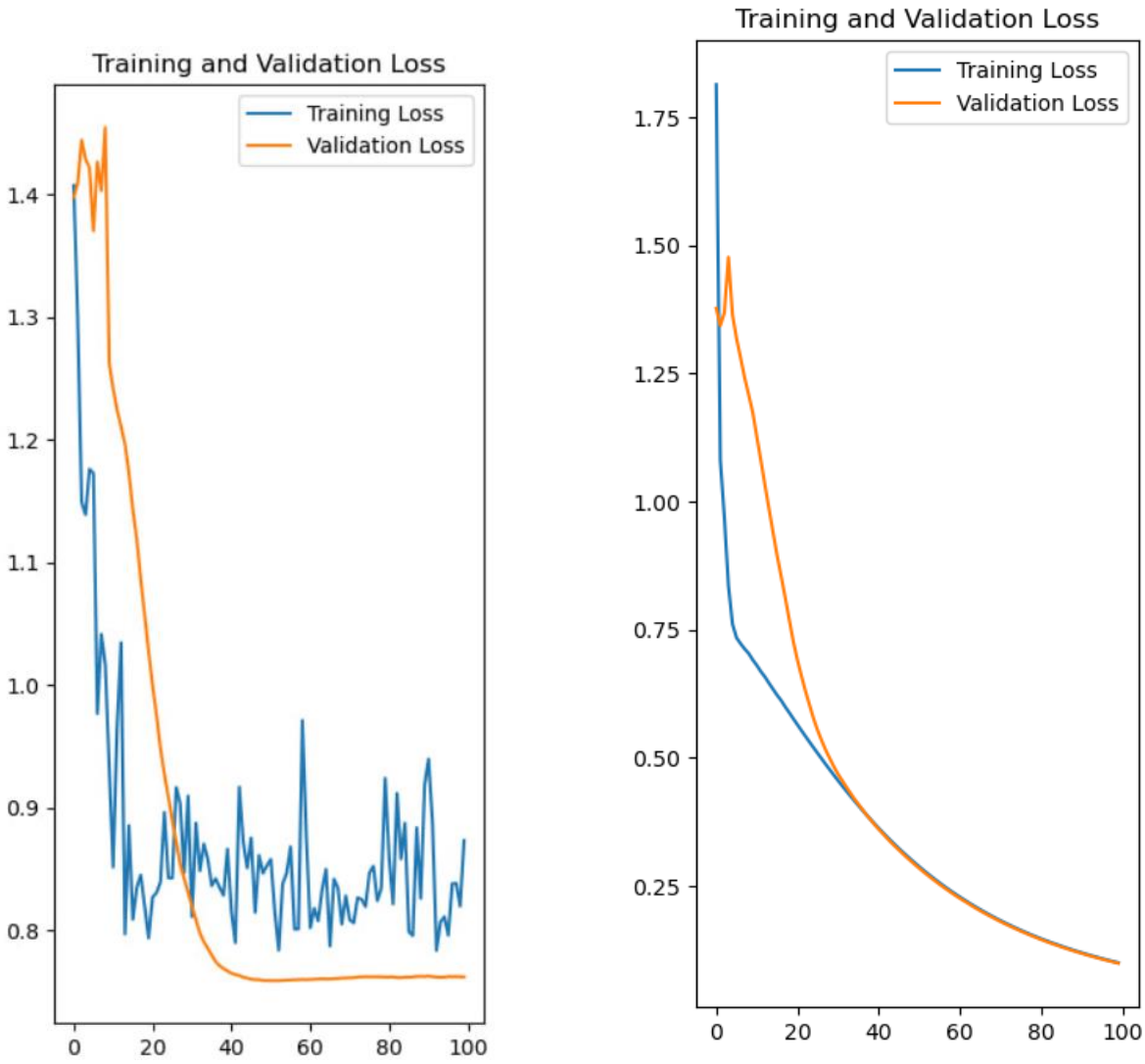
گزارش F1:

```
# Calculate F1 score
f1 = f1_score([true_label], [binary_prediction])

print(
    "Predicted class: {}, True class: {}, F1 Score: {:.2f}%"
    .format(class_names[np.argmax(score)], class_names[true_label], f1 * 100)
)
```

```
1/1 [==============================] - 0s 25ms/step
Predicted class: person_2, True class: person_2, F1 Score: 100.00%
```

همچنین نتیجه دو نوع پیاده‌سازی دیگر برای همین مساله:



با کم و زیاد کردن درصد dropout یا استفاده از توابع loss مختلف و نیز لایه بندی متفاوت، نمودار نیز واکنش داده و متناسبا تغییر خواهد کرد. اما کلیات کار، حفظ خواهد شد.

منابع:

https://keras.io/api/

https://www.tensorflow.org/api_docs/python/tf

https://docs.python.org/3.11/