Report TP4

Transfer Learning with MobileNetV2

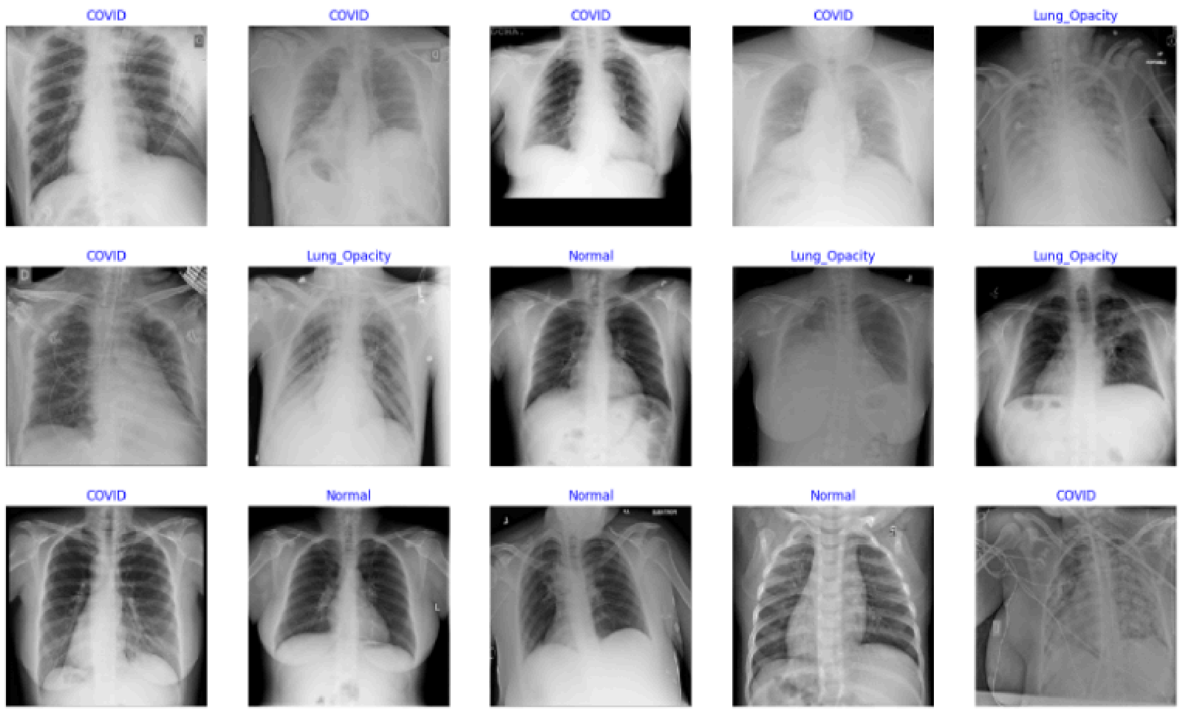
Ghandouz Amina G2

1. About the dataset: COVID-19 radiography dataset

Based on the url which we can find on *tp4 file*, COVID-19 Radiography dataset contains 33,920 chest X-ray images including:

- 11,956 covid-19
- 11,263 non-covid infections (viral or bacterial pneumonia)
- 10,701 normal

It contains also ground-truth lung segmentation masks for the entire dataset
The dataset we worked with has three main classes/categories: COVID-19, Normal, and non-covid infections (Viral Pneumonia and Lung_Opacity).In total we have ~21165 images for all the categories, we're using this dataset for multi-classification task.



2. Preprocessing

Before starting the training process, we preprocessed the dataset where we did:

- normalization (by dividing the pixel values by 255.0)
- resizing (resize all images to 244*244 pixels to match the input size expected by the MobileNet architecture)
- data splitting (we splitted the dataset into three parts: 70% for training, 15% for testing to evaluate the model during development and 15% for validation)

3. Loading and pre-trained architecture:

We imported the MobileNetV2 architecture and initialized it with pretrained weights from ImageNet and removed the original classification head to add our own classifier later.

Using the pretrained MobileNetV2 model, we’re leveraging transfer learning and this allows us to:

- avoid the training process from scratch.

- use a model that already learned useful visual features.

- speed up convergence and often achieve higher accuracy with less data.

4. Loading and pre-trained the mobileNetV2 model:

The original top layers used for ImageNet classification have 1000 classes sp we excluded them to add a new sequence of fully connected layers:

```
model = keras.Sequential([
    base_model,
    layers.Flatten(),
    layers.Dense(1024, activation="relu"),
    layers.Dense(512, activation="relu"),
    layers.Dense(3, activation="softmax")
])
```

- *Flatten*(): Converts the 4D output into a 1D vector.
- *Dense*(1024, activation='relu')
- *Dense*(512, activation='relu')

- *Dense*(3, activation='softmax').

For the total parameters we got 67,011,140 parameters where 66,977,028 of them are trainable and the rest (34,112) are non-trainable parameters

Here's a summary of the model layers with their parameter counts:

Layer	output shape	params
MobileNetV2	(None, 7, 7, 1280)	2,257,984
Flatten	(None, 62720)	0
Dense (1024)	(None, 1024)	64,226,304
Dense (512)	(None, 512)	524,800
Dense (number of classes)	(None, 4)	2,052
total params		67,011,140

5. Transfer learning strategies:

To train the mobileNetV2 model, we used transfer learning with three different strategies and fixed hyper-parameters:

- number of epochs = 20
- batch size = 64
- optimizer = adam
- learning rate = 0.001

The strategies we have are:

a- Training only the last 3 fully connected layers:

Using this strategy, all the layers in mobileNetV2 are frozen except the last 3 fully connected layers (the ones we added before on step 5).

b- Training the last convolution layer and 3 fully connected layers:

The trainable layers are the last convolution layer and 3 fully connected layers we used on the previous strategy.

c- Training the last 2 convolution and the last 3 fully connected layers:

In this strategy we added an additional layer so we have trained the last 2 convolution layers and 3 fully connected ones.

6. The obtained results:

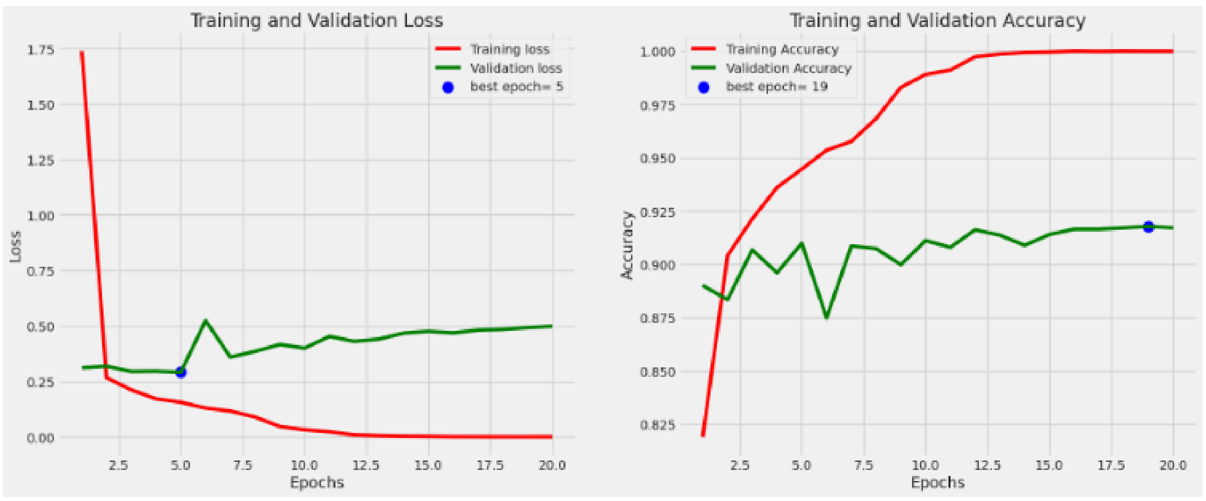
Training time:

For all the three model we have **total params = 67,011,140**

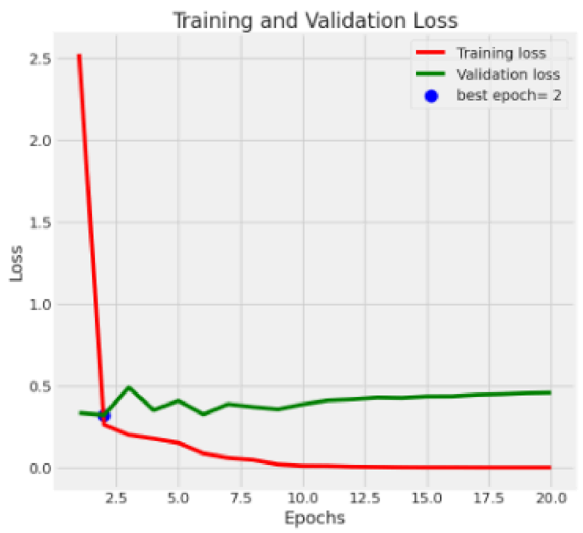
model	layers	trainable params	non-trainable params	training time
model1	3 FC	64,753,156	2,257,984	25 min 2 s
model2	1 Conv + 3 FC	65,165,316	1,845,824	**24 min 18 s**
model3	2 Conv + 3 FC	65,473,156	1,537,984	26 min 59 s

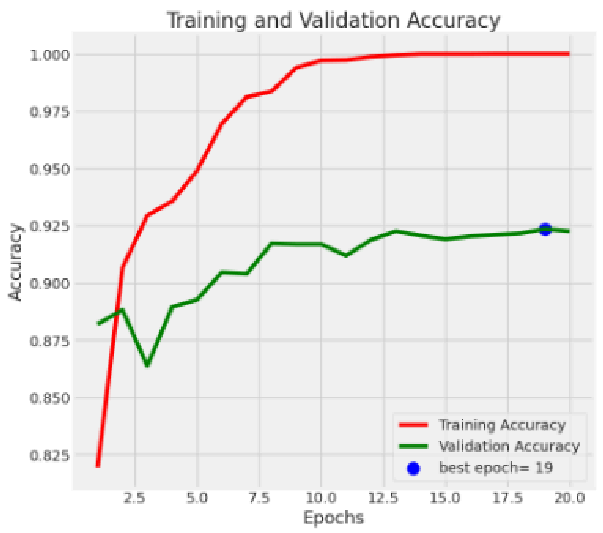
The convergence curve:

Model1: 3 FC

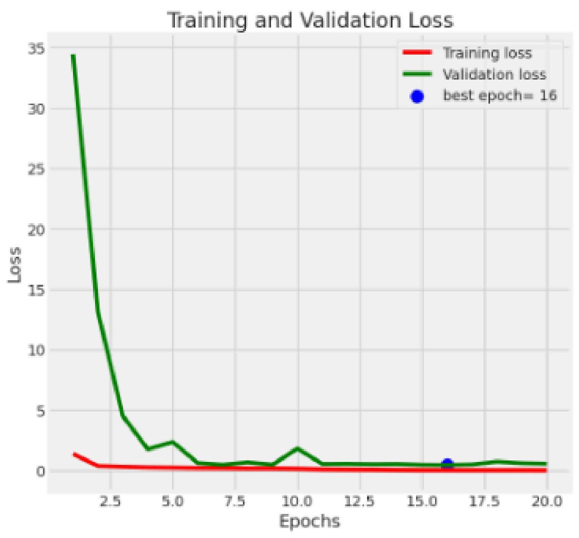


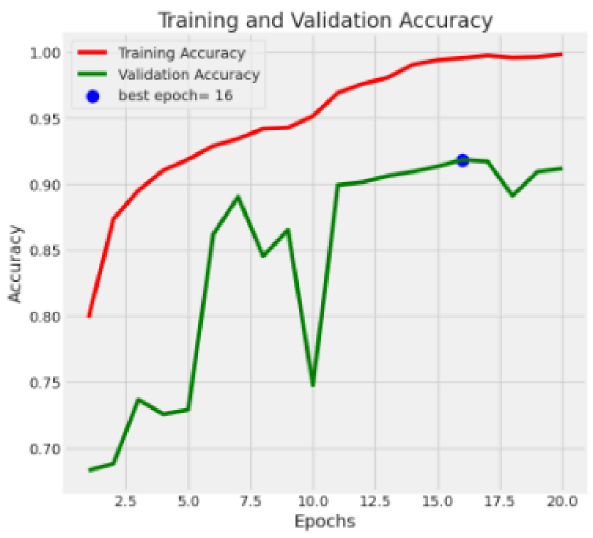
Model2: 1 Conv + 3 FC





Model3: 2 Conv + 3 FC





model	loss best epoch	accuracy best epoch	best training accuracy
model1	5	19	0.92
model2	**2**	19	0.925
model3	16	**16**	**0.93**

Accuracy and loss for the three sets:

model	loss/accuracy	training set	test set	validation set
model1	accuracy	0.96	0.908	0.909
	loss	0.101	**0.27**	**0.29**
model2	accuracy	0.91	0.89	0.88
	loss	0.22	0.29	0.31
model3	accuracy	**0.99**	**0.91**	**0.91**
	loss	**0.02**	0.46	0.44

Model 1:

```
                 precision    recall  f1-score   support


          COVID       0.95      0.92      0.94       542
   Lung_Opacity       0.91      0.81      0.86       902
         Normal       0.89      0.95      0.92      1529
Viral Pneumonia       0.98      0.96      0.97       202


       accuracy                           0.91      3175
      macro avg       0.93      0.91      0.92      3175
   weighted avg       0.91      0.91      0.91      3175
```

Model 2:

```
                 precision    recall  f1-score   support

           COVID       0.89      0.91      0.90       542
    Lung_Opacity       0.91      0.79      0.85       902
          Normal       0.88      0.95      0.91      1529
 Viral Pneumonia       0.98      0.92      0.95       202

        accuracy                           0.89      3175
       macro avg       0.92      0.89      0.90      3175
    weighted avg       0.90      0.89      0.89      3175
```

Model 3:

```
                 precision    recall  f1-score   support

           COVID       0.94      0.92      0.93       542
    Lung_Opacity       0.90      0.84      0.87       902
          Normal       0.90      0.95      0.92      1529
 Viral Pneumonia       0.99      0.92      0.96       202

        accuracy                           0.91      3175
       macro avg       0.93      0.91      0.92      3175
    weighted avg       0.91      0.91      0.91      3175
```

We notice that model2 is the fastest maybe due to the fewer convolution layers compared to model3 while model3 converges slightly faster than model1 and model2 and has the highest accuracy (99% for training and 91% for training/testing sets), so adding more convolutional layers in the learning phase will lead to faster convergence but it may lead to overfitting (loss on training set is 0.02 but for validation/test sets is more than 0.4)

Model1 has the highest precision for all classes and a good balance between precision and recall, resulting in a good F1-Score and model3 is close to model1 in terms of performance, especially for "COVID" and "Viral Pneumonia," but has slightly lower performance for "Lung_Opacity."

So, **model1** (with only 3 FC layers) achieves the **best overall performance** in terms of accuracy, F1-Score, and precision across the test and validation sets **model3** (with 2 convolutional layers + 3 FC layers) performs **almost as well as model1**, with **similar accuracy** and F1-Score. However, it takes slightly longer to converge and has an overfitting problem, adding an extra convolutional layer seems to help the model detect more complex features, leading to a slight improvement in accuracy for certain classes, but the performance boost is marginal compared to model1, and the increase in training time is a trade-off. **Model2** (with 1 convolutional layer + 3 FC layers) is **slightly slower** and yields **lower overall accuracy**.

So the best model is **Model1**

7. Part two: U-Net segmentation model on masks:

U-Net is a deep learning model designed for image segmentation, it has a U-shaped architecture with two main parts:

- encoder: it extracts the features from input images using convolution and max pooling layers to reduce the image size while increasing the depth
- decoder: it upsamples the features maps to the original image size using Conv2DTranspose and skip connections which bring back info from the encoder

We applied the U-net model only on the COVID images/masks to avoid applying the model on the whole dataset (it needs a lot of time)

We used the data generator class to pair correctly image-mask then generate data in batches with shuffling after each epoch.

On the U-net model architecture, we could distinguish:

- convolution block: 2 Conv2D layers with ReLU activation function.

- encoder block: extract features via conv_block and reduce size using max pooling layer.
- decoder block: Conv2DTranspose to upsample, concatenate layer and conv_block

We build the model to have 4 encoder blocks/decoders and final output layer (conv2D) with sigmoid activation function to get the image's mask

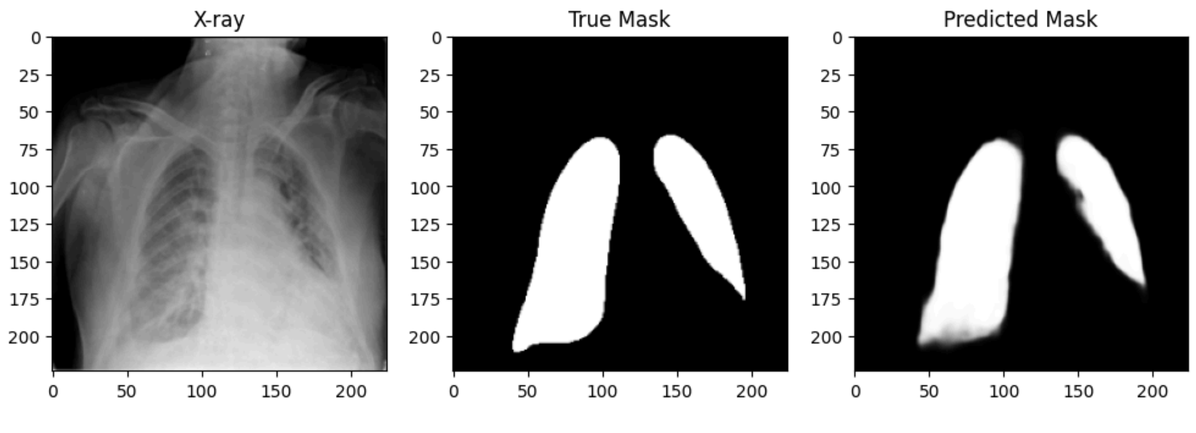
For the training, we splitted the dataset into 80% training and 20% validation sets, the training process was done with adam optimizer

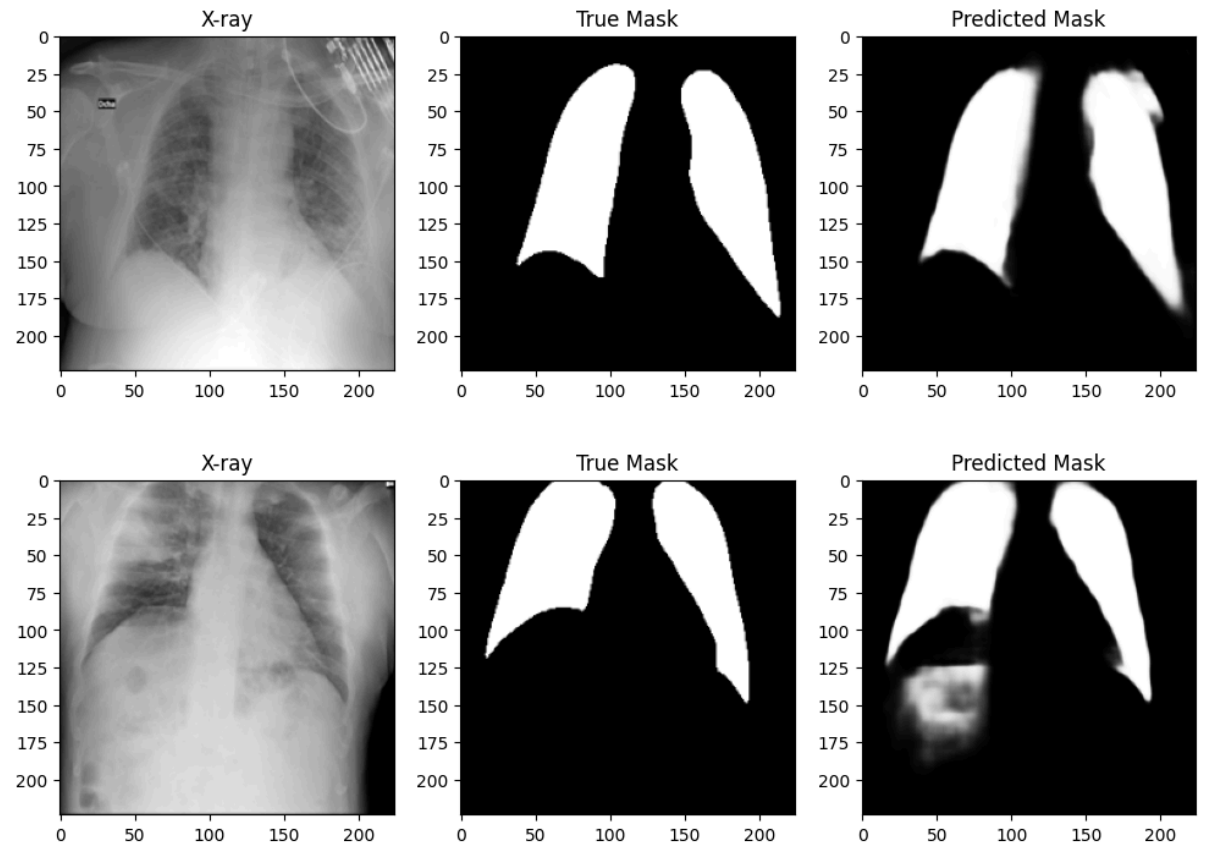
Obtained results:

epoch	**accuracy (train)**	**loss (train)**	**accuracy (val)**	**loss (val)**
1	0.7413	0.5147	0.8563	0.3326
5	0.9671	0.0616	0.9682	0.0580
10	0.9758	0.0363	0.9758	0.0372

We notice that the model improved rapidly across the epochs. with high accuracy on both validation and training set (> 97%) and low loss (0.04) with no overfitting.

Prediction results:




X-ray
True Mask
Predicted Mask
X-ray
True Mask
Predicted Mask