Compulsory Assignment 3: Semantic segmentation

Please fill out the the group name, number, members and optionally the name below.

Group number: 31 **Group member 1**: Asim Rasheed **Group member 2**: Jony karmakar **Group member 3**: Haris Hameed Mian **Group name (optional)**: Group31

Assignment Submission

To complete this assignment answer the relevant questions in this notebook and write the code required to implement the relevant models. This is the biggest assignemnt of the semester, and therefore you get two weeks to work on it. However, we reccomend that **you start early**. This assignment has three semi-big sections, each of which build upon the last. So if you delay the assignment until the day before submission, you will most likely fail. This assignment is completed by doing the following.

- Submit notebook as an .ipynb file to canvas.
- Submit notebook as an .pdf file to canvas.
- Submit the python script you run on ORION to canvas.
- Submit the SLURM script you run on ORION to canvas.
- Submit at least one of your model predictions to the Kaggle leaderboard, and attain a score that is higher than the *BEAT ME* score.

NOTE: Remember to go through the rules given in the lecture "Introduction to compulsory assignments", as there are many do's and dont's with regard to how you should present the work you are going to submit.

Introduction

This assignment will center around semantic segmentation of the dataset in the TGS salt identification challenge. Several of the Earths accumulations of oil and gas **also** have huge deposits of salt, which is easier to detect than the actual hydrocarbons. However, knowing where the salt deposits are precisely is still quite difficult, and segmentation of the seismic images is still something that requires expert interpretation of the images. This leads variable, and highly subjective renderings. To create more accurate, objective segmentations TGS (the worlds leading geoscience data company) have created this challenge to determine if a deep learning model is up to the task.

Dataset

In this assigmnet you will be given 3500 annotated images. The image, and mask dimensions are 128x128 pixels. With each image there follows an annotation mask where each pixel is classified as `1` (salt deposit) or `0` not salt deposit. The test-dataset contains 500 images, where no ground truth masks are given. To evualuate your model on the test dataset, submit your predictions to the Kaggle leaderboard.

Assignment tasks

1. Implement a U-net model, and train it to segment the dataset.
2. Implement a U-net model that uses a pre-trained backbone model of your choice (VGGnet, ResNet, DarkNet, etc.), and train it to segment the dataset.
3. Train one of the models from part 1 or 2 on Orion, and compare the training times and attained performances.
4. Submit the best model prediction on Kaggle learderboard.

Submissions to the Kaggle leaderboard

Link to the Kaggle leaderboard will be posted in the Canvas assignment.

```
y_pred      = model.predict(X_test)                           # Make
prediction
flat_y_pred = y_pred.flatten()                                # Flatten
prediction
flat_y_pred[flat_y_pred >= USER_DETERMINED_THRESHOLD] = 1 # Binarize
prediction (Optional, depends on output activation used)
flat_y_pred[flat_y_pred != 1]   = 0                           # Binarize
prediction (Optional, depends on output activation used)
submissionDF = pd.DataFrame()
submissionDF['ID'] = range(len(flat_y_pred))                  # The
submission csv file must have a column called 'ID'
submissionDF['Prediction'] = flat_y_pred
submissionDF.to_csv('submission.csv', index=False)            # Remember
to store the dataframe to csv without the nameless index column.
```

Library imports

```
import time
from tqdm import tqdm # Cool progress bar
from time import time

import numpy as np
import pandas as pd
import h5py

import matplotlib.pyplot as plt
```

```python
import seaborn as sns

import tensorflow.keras as ks
import tensorflow as tf

SEED = 458 # Feel free to set another seed if you want to
RNG = np.random.default_rng(SEED) # Random number generator
tf.random.set_seed(SEED)

from utilities import *
from visualization import *
```

```
c:\NMBU\Miniconda3\lib\site-packages\scipy\__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this
version of SciPy (detected version 1.24.3
  warnings.warn(f"A NumPy version >={np_minversion} and
<{np_maxversion}"
```

Data loading

Load the data from the HDF5 file `student_TGS_challenge.h5` that is available on Canvas, and Kaggle. The data should be loaded in the same manner as in CA2.

```python
dataset_path = './student_TGS_challenge.h5'

with h5py.File(dataset_path,'r') as f:
    print('Datasets in file:', list(f.keys()))
    X_train = np.asarray(f['X_train'])
    y_train = np.asarray(f['y_train'])
    X_test  = np.asarray(f['X_test'])
    print('Nr. train images: %i'%(X_train.shape[0]))
    print('Nr. test images: %i'%(X_test.shape[0]))
```

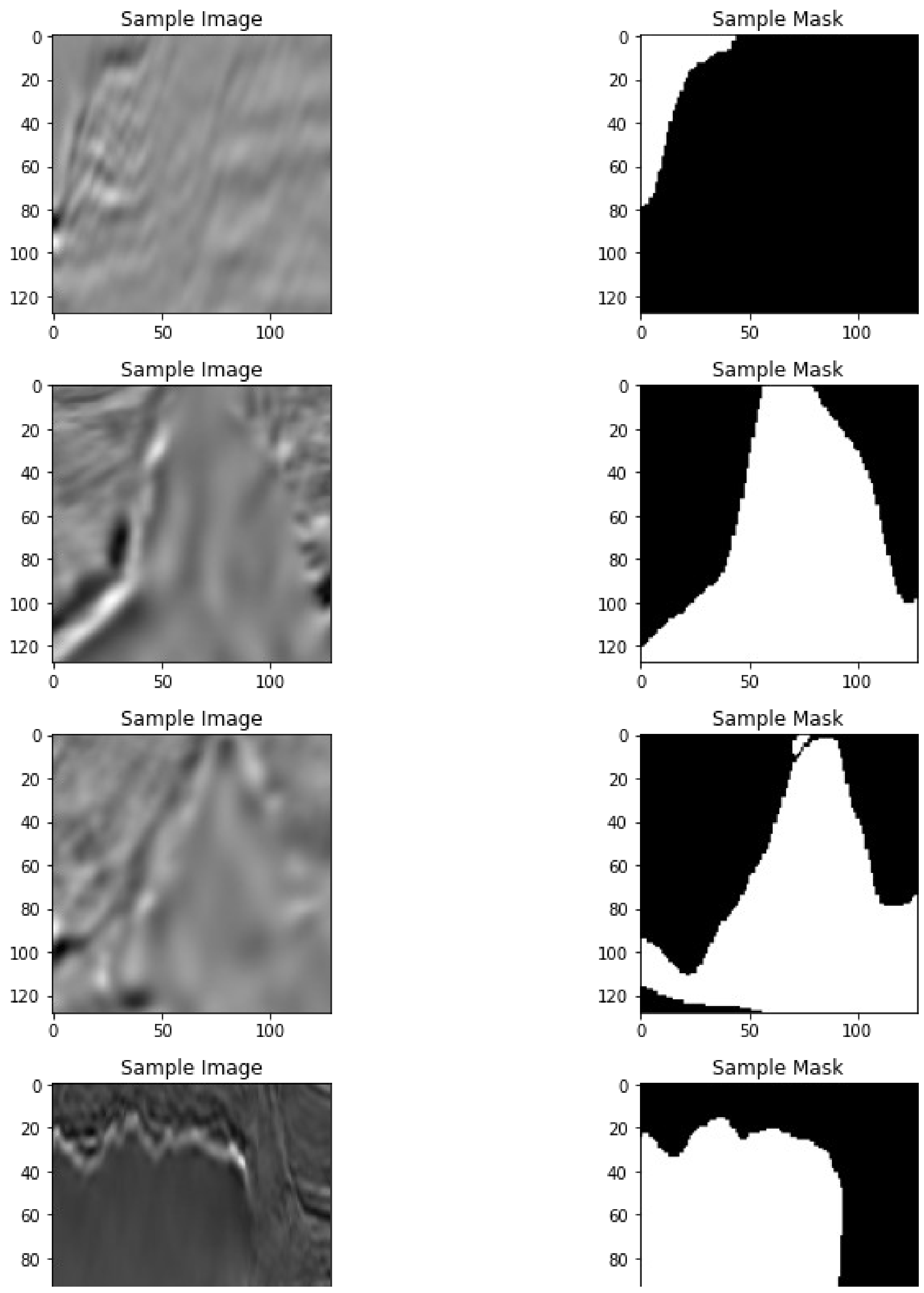
```
Datasets in file: ['X_test', 'X_train', 'y_train']
Nr. train images: 3500
Nr. test images: 500
```

Visualization

Plot a few samples images and masks. Feel free to visualize any other aspects of the dataset that you feel are relevant.

```python
# Number of samples
num_samples = 4
```

```python
# subplot with the specified number of rows and columns
fig, axes = plt.subplots(num_samples, 2, figsize=(12, 12))

# plotting images and masks
for i in range(num_samples):
    # Plot the image
    axes[i, 0].imshow(X_train[i], cmap='gray')
    axes[i, 0].set_title('Sample Image')

    # Plot the mask
    axes[i, 1].imshow(y_train[i], cmap='gray')
    axes[i, 1].set_title('Sample Mask')

# Spacing between subplots
plt.tight_layout()
plt.show()
```


Sample Image
Sample Mask
Sample Image
Sample Mask
Sample Image
Sample Mask
Sample Image
Sample Mask

Preprocessing

Preprocess the dataset in whatever ways you think are helpful.

```
# Normalize the images
X_train_normalized = X_train / 255.0
X_test_normalized = X_test / 255.0
```

Part 1: Implementing U-net

Intersection over Union

The IoU score is a popular metric in both segmentation and object detection problems.

If you want to use the `plot_training_history` function in the `visualization.py` library remember to compile the model with the TP, TN, FP, FN metrics such that you can estimate the *Intersection-over-Union*. **However, it is voluntary to estimate IoU**

See example below:

```
from tensorflow.keras.metrics import FalseNegatives, FalsePositives,
TrueNegatives, TruePositives
from utilities import F1_score,
from visualization import plot_training_history,
...
model.compile(optimizer='Something',
                  loss='Something else',
                  metrics=[FalseNegatives(),
                           FalsePositives(),
                           TrueNegatives(),
                           TruePositives(),
                           F1_score,
                           OtherMetricOfChoice])

training_history = model.fit(X_train, y_train, ...)
plot_training_history(training_history)
```

You have also been provided with a custom F1-score metric in the `utilities.py` library, which is specific for image segmentation. **This is mandatory to use when compiling the model**.

Task 1.1 Model implementation

Implement the classical U-net structure that you have learned about in the lectures. Feel free to experiment with the number of layers, loss-function, batch-normalization, etc. **Remember to compile with the F1-score metric**.

```
def conv2d_block(input_tensor, n_filters, kernel_size=3,
batchnorm=True):
    """Function to add 2 convolutional layers with the parameters
passed to it"""
    # first layer
    x =  ks.layers.Conv2D(filters=n_filters, kernel_size=(kernel_size,
kernel_size), kernel_initializer='he_normal', padding='same')
(input_tensor)
    if batchnorm:
        x =  ks.layers.BatchNormalization()(x)
    x =  ks.layers.Activation('relu')(x)

    # second layer
    x =  ks.layers.Conv2D(filters=n_filters, kernel_size=(kernel_size,
kernel_size), kernel_initializer='he_normal', padding='same')(x)
    if batchnorm:
        x =  ks.layers.BatchNormalization()(x)
    x =  ks.layers.Activation('relu')(x)

    return x

def get_unet(input_img, n_filters=16, dropout=0.1, batchnorm=True):
    # Contracting Path
    c1 = conv2d_block(input_img, n_filters * 1, kernel_size=3,
batchnorm=batchnorm)
    p1 = ks.layers.MaxPooling2D((2, 2))(c1)
    p1 = ks.layers.Dropout(dropout)(p1)

    c2 = conv2d_block(p1, n_filters * 2, kernel_size=3,
batchnorm=batchnorm)
    p2 = ks.layers.MaxPooling2D((2, 2))(c2)
    p2 = ks.layers.Dropout(dropout)(p2)

    c3 = conv2d_block(p2, n_filters * 4, kernel_size=3,
batchnorm=batchnorm)
    p3 = ks.layers.MaxPooling2D((2, 2))(c3)
    p3 = ks.layers.Dropout(dropout)(p3)

    c4 = conv2d_block(p3, n_filters * 8, kernel_size=3,
batchnorm=batchnorm)
    p4 = ks.layers.MaxPooling2D((2, 2))(c4)
    p4 = ks.layers.Dropout(dropout)(p4)

    c5 = conv2d_block(p4, n_filters=n_filters * 16, kernel_size=3,
batchnorm=batchnorm)

    # Expansive Path
    u6 = ks.layers.Conv2DTranspose(n_filters * 8, (3, 3), strides=(2,
2), padding='same')(c5)
    u6 = ks.layers.concatenate([u6, c4])
    u6 =ks.layers. Dropout(dropout)(u6)
    c6 = conv2d_block(u6, n_filters * 8, kernel_size=3,
batchnorm=batchnorm)

    u7 = ks.layers.Conv2DTranspose(n_filters * 4, (3, 3), strides=(2,
2), padding='same')(c6)
    u7 = ks.layers.concatenate([u7, c3])
    u7 = ks.layers.Dropout(dropout)(u7)
    c7 = conv2d_block(u7, n_filters * 4, kernel_size=3,
batchnorm=batchnorm)

    u8 = ks.layers.Conv2DTranspose(n_filters * 2, (3, 3), strides=(2,
2), padding='same')(c7)
    u8 = ks.layers.concatenate([u8, c2])
    u8 = ks.layers.Dropout(dropout)(u8)
    c8 = conv2d_block(u8, n_filters * 2, kernel_size=3,
batchnorm=batchnorm)

    u9 = ks.layers.Conv2DTranspose(n_filters * 1, (3, 3), strides=(2,
2), padding='same')(c8)
    u9 = ks.layers.concatenate([u9, c1])
    u9 = ks.layers.Dropout(dropout)(u9)
    c9 = conv2d_block(u9, n_filters * 1, kernel_size=3,
batchnorm=batchnorm)

    outputs = ks.layers.Conv2D(1, (1, 1), activation='sigmoid')(c9)
    model = ks.Model(inputs=[input_img], outputs=[outputs])
    return model

# Define the U-Net model
input_img = ks.layers.Input((128, 128, 3))
model = get_unet(input_img)

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy',
                       F1_score,
                       tf.keras.metrics.FalseNegatives(),
                       tf.keras.metrics.FalsePositives(),
                       tf.keras.metrics.TrueNegatives(),
                       tf.keras.metrics.TruePositives()])

model.summary()
```

```
Model: "model"
______________________________________________________________________
____________________________
 Layer (type)                Output Shape                 Param #
Connected to
```

```
======================================================================
============================
 input_1 (InputLayer)        [(None, 128, 128, 3)]        0          []



 conv2d (Conv2D)             (None, 128, 128, 16)         448
['input_1[0][0]']


 batch_normalization (Batch  (None, 128, 128, 16)         64
['conv2d[0][0]']
 Normalization)



 activation (Activation)     (None, 128, 128, 16)         0
['batch_normalization[0][0]']


 conv2d_1 (Conv2D)           (None, 128, 128, 16)         2320
['activation[0][0]']


 batch_normalization_1 (Bat  (None, 128, 128, 16)         64
['conv2d_1[0][0]']
 chNormalization)



 activation_1 (Activation)   (None, 128, 128, 16)         0
['batch_normalization_1[0][0]'
                                                                     ]



 max_pooling2d (MaxPooling2  (None, 64, 64, 16)           0
['activation_1[0][0]']
 D)



 dropout (Dropout)           (None, 64, 64, 16)           0
['max_pooling2d[0][0]']


 conv2d_2 (Conv2D)           (None, 64, 64, 32)           4640
['dropout[0][0]']


 batch_normalization_2 (Bat  (None, 64, 64, 32)           128
['conv2d_2[0][0]']
 chNormalization)



 activation_2 (Activation)   (None, 64, 64, 32)           0
['batch_normalization_2[0][0]'
                                                                    ]



 conv2d_3 (Conv2D)           (None, 64, 64, 32)           9248
['activation_2[0][0]']


 batch_normalization_3 (Bat  (None, 64, 64, 32)           128
['conv2d_3[0][0]']
 chNormalization)



 activation_3 (Activation)   (None, 64, 64, 32)           0
['batch_normalization_3[0][0]'
                                                                    ]



 max_pooling2d_1 (MaxPoolin  (None, 32, 32, 32)           0
['activation_3[0][0]']
 g2D)



 dropout_1 (Dropout)         (None, 32, 32, 32)           0
['max_pooling2d_1[0][0]']


 conv2d_4 (Conv2D)           (None, 32, 32, 64)           18496
['dropout_1[0][0]']


 batch_normalization_4 (Bat  (None, 32, 32, 64)           256
['conv2d_4[0][0]']
 chNormalization)



 activation_4 (Activation)   (None, 32, 32, 64)           0
['batch_normalization_4[0][0]'
                                                                    ]
 conv2d_5 (Conv2D)           (None, 32, 32, 64)           36928
['activation_4[0][0]']


 batch_normalization_5 (Bat  (None, 32, 32, 64)           256
['conv2d_5[0][0]']
 chNormalization)



 activation_5 (Activation)   (None, 32, 32, 64)           0
['batch_normalization_5[0][0]'
                                                                        ]



 max_pooling2d_2 (MaxPoolin  (None, 16, 16, 64)           0
['activation_5[0][0]']
 g2D)



 dropout_2 (Dropout)         (None, 16, 16, 64)           0
['max_pooling2d_2[0][0]']


 conv2d_6 (Conv2D)           (None, 16, 16, 128)          73856
['dropout_2[0][0]']


 batch_normalization_6 (Bat  (None, 16, 16, 128)          512
['conv2d_6[0][0]']
 chNormalization)



 activation_6 (Activation)   (None, 16, 16, 128)          0
['batch_normalization_6[0][0]'
                                                                        ]



 conv2d_7 (Conv2D)           (None, 16, 16, 128)          147584
['activation_6[0][0]']


 batch_normalization_7 (Bat  (None, 16, 16, 128)          512
['conv2d_7[0][0]']
 chNormalization)
 activation_7 (Activation)   (None, 16, 16, 128)          0
['batch_normalization_7[0][0]'
                                                                    ]


 max_pooling2d_3 (MaxPoolin  (None, 8, 8, 128)            0
['activation_7[0][0]']
 g2D)


 dropout_3 (Dropout)         (None, 8, 8, 128)            0
['max_pooling2d_3[0][0]']

 conv2d_8 (Conv2D)           (None, 8, 8, 256)            295168
['dropout_3[0][0]']

 batch_normalization_8 (Bat  (None, 8, 8, 256)            1024
['conv2d_8[0][0]']
 chNormalization)


 activation_8 (Activation)   (None, 8, 8, 256)            0
['batch_normalization_8[0][0]'
                                                                    ]


 conv2d_9 (Conv2D)           (None, 8, 8, 256)            590080
['activation_8[0][0]']

 batch_normalization_9 (Bat  (None, 8, 8, 256)            1024
['conv2d_9[0][0]']
 chNormalization)


 activation_9 (Activation)   (None, 8, 8, 256)            0
['batch_normalization_9[0][0]'
                                                                    ]
 conv2d_transpose (Conv2DTr  (None, 16, 16, 128)          295040
['activation_9[0][0]']
 anspose)



 concatenate (Concatenate)   (None, 16, 16, 256)          0
['conv2d_transpose[0][0]',

'activation_7[0][0]']


 dropout_4 (Dropout)         (None, 16, 16, 256)          0
['concatenate[0][0]']


 conv2d_10 (Conv2D)          (None, 16, 16, 128)          295040
['dropout_4[0][0]']


 batch_normalization_10 (Ba  (None, 16, 16, 128)          512
['conv2d_10[0][0]']
 tchNormalization)



 activation_10 (Activation)  (None, 16, 16, 128)          0
['batch_normalization_10[0][0]
                                                                    ']



 conv2d_11 (Conv2D)          (None, 16, 16, 128)          147584
['activation_10[0][0]']


 batch_normalization_11 (Ba  (None, 16, 16, 128)          512
['conv2d_11[0][0]']
 tchNormalization)



 activation_11 (Activation)  (None, 16, 16, 128)          0
['batch_normalization_11[0][0]
                                                                    ']



 conv2d_transpose_1 (Conv2D  (None, 32, 32, 64)           73792
['activation_11[0][0]']
 Transpose)
 concatenate_1 (Concatenate  (None, 32, 32, 128)          0
['conv2d_transpose_1[0][0]',
 )
'activation_5[0][0]']


 dropout_5 (Dropout)         (None, 32, 32, 128)          0
['concatenate_1[0][0]']


 conv2d_12 (Conv2D)          (None, 32, 32, 64)           73792
['dropout_5[0][0]']


 batch_normalization_12 (Ba  (None, 32, 32, 64)           256
['conv2d_12[0][0]']
 tchNormalization)



 activation_12 (Activation)  (None, 32, 32, 64)           0
['batch_normalization_12[0][0]
                                                                    ']



 conv2d_13 (Conv2D)          (None, 32, 32, 64)           36928
['activation_12[0][0]']


 batch_normalization_13 (Ba  (None, 32, 32, 64)           256
['conv2d_13[0][0]']
 tchNormalization)



 activation_13 (Activation)  (None, 32, 32, 64)           0
['batch_normalization_13[0][0]
                                                                    ']



 conv2d_transpose_2 (Conv2D  (None, 64, 64, 32)           18464
['activation_13[0][0]']
 Transpose)
 concatenate_2 (Concatenate  (None, 64, 64, 64)           0
['conv2d_transpose_2[0][0]',
 )
'activation_3[0][0]']


 dropout_6 (Dropout)         (None, 64, 64, 64)           0
['concatenate_2[0][0]']


 conv2d_14 (Conv2D)          (None, 64, 64, 32)           18464
['dropout_6[0][0]']


 batch_normalization_14 (Ba  (None, 64, 64, 32)           128
['conv2d_14[0][0]']
 tchNormalization)



 activation_14 (Activation)  (None, 64, 64, 32)           0
['batch_normalization_14[0][0]
                                                                    ']



 conv2d_15 (Conv2D)          (None, 64, 64, 32)           9248
['activation_14[0][0]']


 batch_normalization_15 (Ba  (None, 64, 64, 32)           128
['conv2d_15[0][0]']
 tchNormalization)



 activation_15 (Activation)  (None, 64, 64, 32)           0
['batch_normalization_15[0][0]
                                                                    ']



 conv2d_transpose_3 (Conv2D  (None, 128, 128, 16)         4624
['activation_15[0][0]']
 Transpose)



 concatenate_3 (Concatenate  (None, 128, 128, 32)         0
['conv2d_transpose_3[0][0]',
 )
'activation_1[0][0]']


 dropout_7 (Dropout)         (None, 128, 128, 32)         0
['concatenate_3[0][0]']


 conv2d_16 (Conv2D)          (None, 128, 128, 16)         4624
['dropout_7[0][0]']


 batch_normalization_16 (Ba  (None, 128, 128, 16)         64
['conv2d_16[0][0]']
 tchNormalization)



 activation_16 (Activation)  (None, 128, 128, 16)         0
['batch_normalization_16[0][0]
                                                                    ']



 conv2d_17 (Conv2D)          (None, 128, 128, 16)         2320
['activation_16[0][0]']


 batch_normalization_17 (Ba  (None, 128, 128, 16)         64
['conv2d_17[0][0]']
 tchNormalization)



 activation_17 (Activation)  (None, 128, 128, 16)         0
['batch_normalization_17[0][0]
                                                                    ']



 conv2d_18 (Conv2D)          (None, 128, 128, 1)          17
['activation_17[0][0]']


======================================================================
============================
Total params: 2164593 (8.26 MB)
Trainable params: 2161649 (8.25 MB)
Non-trainable params: 2944 (11.50 KB)
______________________________________________________________________
____________________________
```

Task 1.2 Train the model, and plot the training history

Feel free to use the `plot_training_history` function from the provided library `utilities.py`

```
training_history = model.fit(X_train_normalized, y_train,
                             batch_size=32,
                             epochs=30,
                             validation_split=0.2,  # Use 10% of the
data for validation
                             shuffle=True)

Epoch 1/30
88/88 [==============================] - 132s 1s/step - loss: 0.4071 -
accuracy: 0.8334 - F1_score: 0.6112 - false_negatives: 5143123.0000 -
false_positives: 2498698.0000 - true_negatives: 31945636.0000 -
true_positives: 6287747.0000 - val_loss: 6.9366 - val_accuracy: 0.3181
- val_F1_score: 0.4319 - val_false_negatives: 17726.0000 -
val_false_positives: 7802529.0000 - val_true_negatives: 638201.0000 -
val_true_positives: 3010344.0000
Epoch 2/30
88/88 [==============================] - 128s 1s/step - loss: 0.3007 -
accuracy: 0.8854 - F1_score: 0.7485 - false_negatives: 3458509.0000 -
false_positives: 1799365.0000 - true_negatives: 32644960.0000 -
true_positives: 7972361.0000 - val_loss: 2.1645 - val_accuracy: 0.6502
- val_F1_score: 0.5729 - val_false_negatives: 274986.0000 -
val_false_positives: 3737211.0000 - val_true_negatives: 4703519.0000 -
val_true_positives: 2753084.0000
Epoch 3/30
88/88 [==============================] - 126s 1s/step - loss: 0.2703 -
accuracy: 0.8965 - F1_score: 0.7722 - false_negatives: 3254561.0000 -
false_positives: 1493214.0000 - true_negatives: 32951120.0000 -
true_positives: 8176309.0000 - val_loss: 0.9258 - val_accuracy: 0.7380
- val_F1_score: 0.6275 - val_false_negatives: 422195.0000 -
val_false_positives: 2582917.0000 - val_true_negatives: 5857813.0000 -
val_true_positives: 2605875.0000
Epoch 4/30
88/88 [==============================] - 130s 1s/step - loss: 0.2556 -
accuracy: 0.9015 - F1_score: 0.7865 - false_negatives: 2972828.0000 -
false_positives: 1543941.0000 - true_negatives: 32900392.0000 -
true_positives: 8458042.0000 - val_loss: 1.1598 - val_accuracy: 0.6838
- val_F1_score: 0.5962 - val_false_negatives: 312004.0000 -
val_false_positives: 3314627.0000 - val_true_negatives: 5126103.0000 -
val_true_positives: 2716066.0000
Epoch 5/30
88/88 [==============================] - 126s 1s/step - loss: 0.2289 -
accuracy: 0.9130 - F1_score: 0.8113 - false_negatives: 2630054.0000 -
false_positives: 1360847.0000 - true_negatives: 33083484.0000 -
true_positives: 8800816.0000 - val_loss: 0.5501 - val_accuracy: 0.7582
```

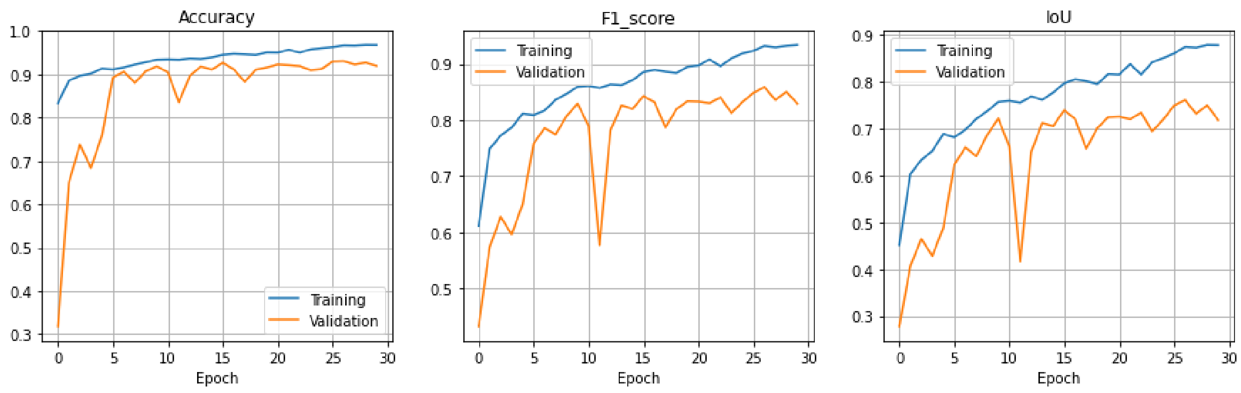
```
- val_F1_score: 0.6495 - val_false_negatives: 380130.0000 -
val_false_positives: 2392858.0000 - val_true_negatives: 6047872.0000 -
val_true_positives: 2647940.0000
Epoch 6/30
88/88 [==============================] - 126s 1s/step - loss: 0.2296 -
accuracy: 0.9109 - F1_score: 0.8088 - false_negatives: 2718804.0000 -
false_positives: 1366426.0000 - true_negatives: 33077910.0000 -
true_positives: 8712066.0000 - val_loss: 0.3689 - val_accuracy: 0.8923
- val_F1_score: 0.7577 - val_false_negatives: 993113.0000 -
val_false_positives: 241814.0000 - val_true_negatives: 8198916.0000 -
val_true_positives: 2034957.0000
Epoch 7/30
88/88 [==============================] - 125s 1s/step - loss: 0.2210 -
accuracy: 0.9159 - F1_score: 0.8169 - false_negatives: 2560052.0000 -
false_positives: 1298237.0000 - true_negatives: 33146100.0000 -
true_positives: 8870818.0000 - val_loss: 0.2757 - val_accuracy: 0.9061
- val_F1_score: 0.7857 - val_false_negatives: 944675.0000 -
val_false_positives: 131790.0000 - val_true_negatives: 8308940.0000 -
val_true_positives: 2083395.0000
Epoch 8/30
88/88 [==============================] - 126s 1s/step - loss: 0.2041 -
accuracy: 0.9224 - F1_score: 0.8360 - false_negatives: 2275129.0000 -
false_positives: 1285621.0000 - true_negatives: 33158712.0000 -
true_positives: 9155741.0000 - val_loss: 0.3277 - val_accuracy: 0.8804
- val_F1_score: 0.7740 - val_false_negatives: 582036.0000 -
val_false_positives: 789532.0000 - val_true_negatives: 7651198.0000 -
val_true_positives: 2446034.0000
Epoch 9/30
88/88 [==============================] - 127s 1s/step - loss: 0.1918 -
accuracy: 0.9281 - F1_score: 0.8466 - false_negatives: 2195116.0000 -
false_positives: 1104266.0000 - true_negatives: 33340064.0000 -
true_positives: 9235754.0000 - val_loss: 0.2790 - val_accuracy: 0.9071
- val_F1_score: 0.8070 - val_false_negatives: 698490.0000 -
val_false_positives: 367114.0000 - val_true_negatives: 8073616.0000 -
val_true_positives: 2329580.0000
Epoch 10/30
88/88 [==============================] - 127s 1s/step - loss: 0.1771 -
accuracy: 0.9335 - F1_score: 0.8591 - false_negatives: 1974174.0000 -
false_positives: 1074790.0000 - true_negatives: 33369534.0000 -
true_positives: 9456696.0000 - val_loss: 0.2108 - val_accuracy: 0.9182
- val_F1_score: 0.8292 - val_false_negatives: 604128.0000 -
val_false_positives: 333930.0000 - val_true_negatives: 8106800.0000 -
val_true_positives: 2423942.0000
Epoch 11/30
88/88 [==============================] - 126s 1s/step - loss: 0.1758 -
accuracy: 0.9343 - F1_score: 0.8608 - false_negatives: 1953936.0000 -
false_positives: 1060254.0000 - true_negatives: 33384076.0000 -
true_positives: 9476934.0000 - val_loss: 0.2420 - val_accuracy: 0.9047
- val_F1_score: 0.7894 - val_false_negatives: 895594.0000 -
```

```
val_false_positives: 197174.0000 - val_true_negatives: 8243556.0000 -
val_true_positives: 2132476.0000
Epoch 12/30
88/88 [==============================] - 126s 1s/step - loss: 0.1726 -
accuracy: 0.9335 - F1_score: 0.8570 - false_negatives: 2047305.0000 -
false_positives: 1005189.0000 - true_negatives: 33439138.0000 -
true_positives: 9383565.0000 - val_loss: 0.3714 - val_accuracy: 0.8351
- val_F1_score: 0.5769 - val_false_negatives: 1680586.0000 -
val_false_positives: 210994.0000 - val_true_negatives: 8229736.0000 -
val_true_positives: 1347484.0000
Epoch 13/30
88/88 [==============================] - 127s 1s/step - loss: 0.1657 -
accuracy: 0.9366 - F1_score: 0.8634 - false_negatives: 1808928.0000 -
false_positives: 1099691.0000 - true_negatives: 33344644.0000 -
true_positives: 9621942.0000 - val_loss: 0.2908 - val_accuracy: 0.8963
- val_F1_score: 0.7819 - val_false_negatives: 819733.0000 -
val_false_positives: 370127.0000 - val_true_negatives: 8070603.0000 -
val_true_positives: 2208337.0000
Epoch 14/30
88/88 [==============================] - 127s 1s/step - loss: 0.1690 -
accuracy: 0.9355 - F1_score: 0.8621 - false_negatives: 2018655.0000 -
false_positives: 939807.0000 - true_negatives: 33504524.0000 -
true_positives: 9412215.0000 - val_loss: 0.2517 - val_accuracy: 0.9179
- val_F1_score: 0.8262 - val_false_negatives: 706460.0000 -
val_false_positives: 235570.0000 - val_true_negatives: 8205160.0000 -
val_true_positives: 2321610.0000
Epoch 15/30
88/88 [==============================] - 123s 1s/step - loss: 0.1604 -
accuracy: 0.9390 - F1_score: 0.8711 - false_negatives: 1735293.0000 -
false_positives: 1060884.0000 - true_negatives: 33383448.0000 -
true_positives: 9695577.0000 - val_loss: 0.2317 - val_accuracy: 0.9107
- val_F1_score: 0.8199 - val_false_negatives: 585912.0000 -
val_false_positives: 437786.0000 - val_true_negatives: 8002944.0000 -
val_true_positives: 2442158.0000
Epoch 16/30
88/88 [==============================] - 124s 1s/step - loss: 0.1464 -
accuracy: 0.9449 - F1_score: 0.8855 - false_negatives: 1577632.0000 -
false_positives: 949886.0000 - true_negatives: 33494448.0000 -
true_positives: 9853238.0000 - val_loss: 0.1889 - val_accuracy: 0.9265
- val_F1_score: 0.8426 - val_false_negatives: 645220.0000 -
val_false_positives: 197877.0000 - val_true_negatives: 8242853.0000 -
val_true_positives: 2382850.0000
Epoch 17/30
88/88 [==============================] - 127s 1s/step - loss: 0.1393 -
accuracy: 0.9473 - F1_score: 0.8894 - false_negatives: 1482728.0000 -
false_positives: 934085.0000 - true_negatives: 33510244.0000 -
true_positives: 9948142.0000 - val_loss: 0.2263 - val_accuracy: 0.9113
- val_F1_score: 0.8317 - val_false_negatives: 417286.0000 -
val_false_positives: 599941.0000 - val_true_negatives: 7840789.0000 -
```

```
val_true_positives: 2610784.0000
Epoch 18/30
88/88 [==============================] - 127s 1s/step - loss: 0.1405 -
accuracy: 0.9461 - F1_score: 0.8862 - false_negatives: 1512948.0000 -
false_positives: 957962.0000 - true_negatives: 33486376.0000 -
true_positives: 9917922.0000 - val_loss: 0.3388 - val_accuracy: 0.8826
- val_F1_score: 0.7866 - val_false_negatives: 454846.0000 -
val_false_positives: 891634.0000 - val_true_negatives: 7549096.0000 -
val_true_positives: 2573224.0000
Epoch 19/30
88/88 [==============================] - 128s 1s/step - loss: 0.1444 -
accuracy: 0.9447 - F1_score: 0.8838 - false_negatives: 1639680.0000 -
false_positives: 899389.0000 - true_negatives: 33544926.0000 -
true_positives: 9791190.0000 - val_loss: 0.2422 - val_accuracy: 0.9104
- val_F1_score: 0.8195 - val_false_negatives: 629341.0000 -
val_false_positives: 397754.0000 - val_true_negatives: 8042976.0000 -
val_true_positives: 2398729.0000
Epoch 20/30
88/88 [==============================] - 128s 1s/step - loss: 0.1301 -
accuracy: 0.9504 - F1_score: 0.8948 - false_negatives: 1375201.0000 -
false_positives: 899545.0000 - true_negatives: 33544784.0000 -
true_positives: 10055669.0000 - val_loss: 0.2108 - val_accuracy:
0.9156 - val_F1_score: 0.8336 - val_false_negatives: 491660.0000 -
val_false_positives: 476596.0000 - val_true_negatives: 7964134.0000 -
val_true_positives: 2536410.0000
Epoch 21/30
88/88 [==============================] - 127s 1s/step - loss: 0.1284 -
accuracy: 0.9501 - F1_score: 0.8973 - false_negatives: 1391368.0000 -
false_positives: 896911.0000 - true_negatives: 33547420.0000 -
true_positives: 10039502.0000 - val_loss: 0.2210 - val_accuracy:
0.9227 - val_F1_score: 0.8329 - val_false_negatives: 689749.0000 -
val_false_positives: 197163.0000 - val_true_negatives: 8243567.0000 -
val_true_positives: 2338321.0000
Epoch 22/30
88/88 [==============================] - 126s 1s/step - loss: 0.1143 -
accuracy: 0.9563 - F1_score: 0.9081 - false_negatives: 1138429.0000 -
false_positives: 866430.0000 - true_negatives: 33577900.0000 -
true_positives: 10292441.0000 - val_loss: 0.2354 - val_accuracy:
0.9211 - val_F1_score: 0.8300 - val_false_negatives: 710678.0000 -
val_false_positives: 193637.0000 - val_true_negatives: 8247093.0000 -
val_true_positives: 2317392.0000
Epoch 23/30
88/88 [==============================] - 127s 1s/step - loss: 0.1254 -
accuracy: 0.9502 - F1_score: 0.8959 - false_negatives: 1419083.0000 -
false_positives: 866519.0000 - true_negatives: 33577800.0000 -
true_positives: 10011787.0000 - val_loss: 0.2402 - val_accuracy:
0.9189 - val_F1_score: 0.8403 - val_false_negatives: 468271.0000 -
val_false_positives: 461638.0000 - val_true_negatives: 7979092.0000 -
val_true_positives: 2559799.0000
```

```
Epoch 24/30
88/88 [==============================] - 127s 1s/step - loss: 0.1110 -
accuracy: 0.9574 - F1_score: 0.9100 - false_negatives: 1097502.0000 -
false_positives: 858897.0000 - true_negatives: 33585432.0000 -
true_positives: 10333368.0000 - val_loss: 0.3005 - val_accuracy:
0.9093 - val_F1_score: 0.8125 - val_false_negatives: 674615.0000 -
val_false_positives: 365607.0000 - val_true_negatives: 8075123.0000 -
val_true_positives: 2353455.0000
Epoch 25/30
88/88 [==============================] - 127s 1s/step - loss: 0.1049 -
accuracy: 0.9602 - F1_score: 0.9187 - false_negatives: 1112898.0000 -
false_positives: 714534.0000 - true_negatives: 33729808.0000 -
true_positives: 10317972.0000 - val_loss: 0.2423 - val_accuracy:
0.9122 - val_F1_score: 0.8321 - val_false_negatives: 458530.0000 -
val_false_positives: 548297.0000 - val_true_negatives: 7892433.0000 -
val_true_positives: 2569540.0000
Epoch 26/30
88/88 [==============================] - 128s 1s/step - loss: 0.0982 -
accuracy: 0.9627 - F1_score: 0.9234 - false_negatives: 961621.0000 -
false_positives: 750343.0000 - true_negatives: 33693984.0000 -
true_positives: 10469249.0000 - val_loss: 0.1979 - val_accuracy:
0.9297 - val_F1_score: 0.8483 - val_false_negatives: 630455.0000 -
val_false_positives: 175237.0000 - val_true_negatives: 8265493.0000 -
val_true_positives: 2397615.0000
Epoch 27/30
88/88 [==============================] - 127s 1s/step - loss: 0.0899 -
accuracy: 0.9666 - F1_score: 0.9321 - false_negatives: 846246.0000 -
false_positives: 687234.0000 - true_negatives: 33757092.0000 -
true_positives: 10584624.0000 - val_loss: 0.1961 - val_accuracy:
0.9305 - val_F1_score: 0.8586 - val_false_negatives: 501895.0000 -
val_false_positives: 294809.0000 - val_true_negatives: 8145921.0000 -
val_true_positives: 2526175.0000
Epoch 28/30
88/88 [==============================] - 129s 1s/step - loss: 0.0923 -
accuracy: 0.9661 - F1_score: 0.9295 - false_negatives: 857464.0000 -
false_positives: 696799.0000 - true_negatives: 33747536.0000 -
true_positives: 10573406.0000 - val_loss: 0.2416 - val_accuracy:
0.9227 - val_F1_score: 0.8357 - val_false_negatives: 619342.0000 -
val_false_positives: 267705.0000 - val_true_negatives: 8173025.0000 -
val_true_positives: 2408728.0000
Epoch 29/30
88/88 [==============================] - 131s 1s/step - loss: 0.0875 -
accuracy: 0.9677 - F1_score: 0.9323 - false_negatives: 799588.0000 -
false_positives: 683803.0000 - true_negatives: 33760528.0000 -
true_positives: 10631282.0000 - val_loss: 0.2173 - val_accuracy:
0.9271 - val_F1_score: 0.8503 - val_false_negatives: 529359.0000 -
val_false_positives: 306852.0000 - val_true_negatives: 8133878.0000 -
val_true_positives: 2498711.0000
Epoch 30/30
```

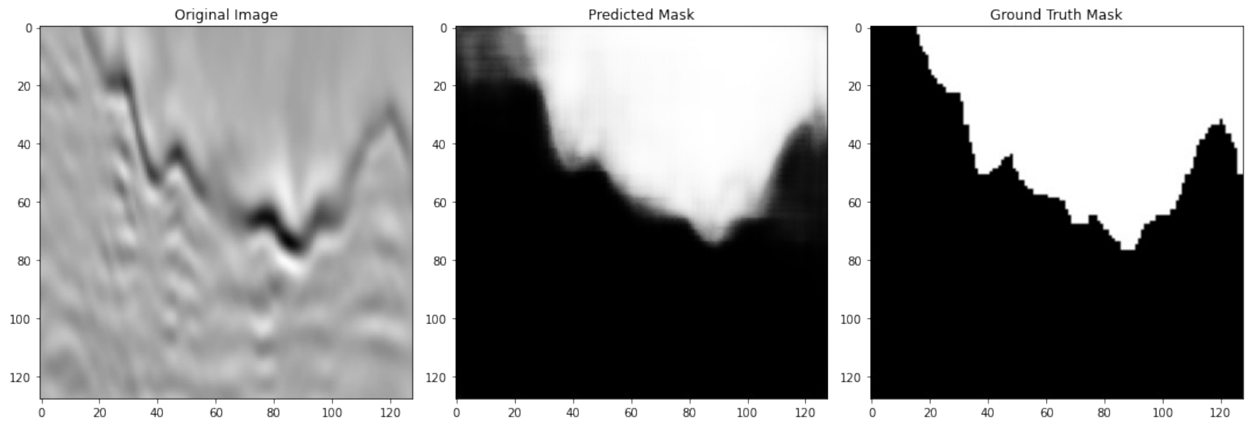
```
88/88 [==============================] - 130s 1s/step - loss: 0.0844 -
accuracy: 0.9675 - F1_score: 0.9338 - false_negatives: 773542.0000 -
false_positives: 719582.0000 - true_negatives: 33724744.0000 -
true_positives: 10657328.0000 - val_loss: 0.2427 - val_accuracy:
0.9192 - val_F1_score: 0.8291 - val_false_negatives: 675466.0000 -
val_false_positives: 251024.0000 - val_true_negatives: 8189706.0000 -
val_true_positives: 2352604.0000

# keys_to_rename = ['false_negatives_1', 'false_positives_1',
'true_negatives_1', 'true_positives_1',
#                    'val_false_negatives_1', 'val_false_positives_1',
'val_true_negatives_1', 'val_true_positives_1']

# for key in keys_to_rename:
#     new_key = key.replace('_1', '')
#     training_history.history[new_key] =
training_history.history.pop(key)

# Plot the training history
plot_training_history(training_history)
```



Task 1.3 Visualize model predictions

Make a plot that illustrates the original image, the predicted mask, and the ground truth mask.

```
def visualize_predictions(X, y_true, y_pred, num_samples=1):
    """Visualize random samples from predictions."""
    indices = np.random.choice(X.shape[0], num_samples, replace=False)

    for idx in indices:
        fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))

        ax1.imshow(X[idx], cmap='gray')
        ax1.set_title("Original Image")

        ax2.imshow(y_pred[idx].squeeze(), cmap='gray')
        ax2.set_title("Predicted Mask")
        ax3.imshow(y_true[idx].squeeze(), cmap='gray')
        ax3.set_title("Ground Truth Mask")

        plt.tight_layout()
        plt.show()

# Make predictions
predictions = model.predict(X_train_normalized)

# Call the function to visualize the samples
visualize_predictions(X_train_normalized, y_train, predictions)


110/110 [==============================] - 25s 227ms/step
```



```
USER_DETERMINED_THRESHOLD = 0.5
y_pred      = model.predict(X_test_normalized)            # Make
prediction
flat_y_pred = y_pred.flatten()                            # Flatten
prediction
flat_y_pred[flat_y_pred >= USER_DETERMINED_THRESHOLD] = 1 # Binarize
prediction (Optional, depends on output activation used)
flat_y_pred[flat_y_pred != 1]    = 0                      # Binarize
prediction (Optional, depends on output activation used)
submissionDF = pd.DataFrame()
submissionDF['ID'] = range(len(flat_y_pred))              # The
submission csv file must have a column called 'ID'
submissionDF['Prediction'] = flat_y_pred
submissionDF.to_csv('submission16.csv', index=False)

16/16 [==============================] - 4s 241ms/step
```

Part 2: Implementing U-net with transfer learning

Implement a model with the U-net structure that you have learned about in the lectures, but now with a pre-trained backbone. There are many pre-trained back-bones to choose from. Pick freely from the selection here tf.keras.applications, or here Keras model scores (nicer table in the second link). Feel free to experiment with the number of layers, loss-function, batch-normalization, etc. Many of the backbones available are quite big, so you might find it quite time-consuming to train them on your personal computers. It might be expedient to only train them for 1-5 epochs on your PCs, and do the full training on Orion in Part 3.

Task 2.1 Transfer learning model implementation

Implement a U-net model utilizing the pre-trained weights of a publically available network. **Remember to compile with the F1-score metric**.

```python
def unet_with_vgg16(input_shape):
    # Load the VGG16 model
    base_model = ks.applications.VGG16(weights='imagenet',
include_top=False, input_shape=input_shape)

    # Encoder
    encoder = base_model.get_layer('block5_pool').output  # Extract
features from the last pooling layer

    # Decoder
    x = ks.layers.UpSampling2D((2, 2))(encoder)
    x = ks.layers.concatenate([x,
base_model.get_layer('block5_conv3').output])
    x = ks.layers.Conv2D(512, (3, 3), padding='same')(x)
    x = ks.layers.BatchNormalization()(x)
    x = ks.layers.Activation('relu')(x)

    x = ks.layers.UpSampling2D((2, 2))(x)
    x = ks.layers.concatenate([x,
base_model.get_layer('block4_conv3').output])
    x = ks.layers.Conv2D(512, (3, 3), padding='same')(x)
    x = ks.layers.BatchNormalization()(x)
    x = ks.layers.Activation('relu')(x)

    x = ks.layers.UpSampling2D((2, 2))(x)
    x = ks.layers.concatenate([x,
base_model.get_layer('block3_conv3').output])
    x = ks.layers.Conv2D(256, (3, 3), padding='same')(x)
    x = ks.layers.BatchNormalization()(x)
    x = ks.layers.Activation('relu')(x)

    x = ks.layers.UpSampling2D((2, 2))(x)
    x = ks.layers.concatenate([x,
base_model.get_layer('block2_conv2').output])
    x = ks.layers.Conv2D(128, (3, 3), padding='same')(x)
    x = ks.layers.BatchNormalization()(x)
    x = ks.layers.Activation('relu')(x)

    x = ks.layers.UpSampling2D((2, 2))(x)
    x = ks.layers.concatenate([x,
base_model.get_layer('block1_conv2').output])
    x = ks.layers.Conv2D(64, (3, 3), padding='same')(x)
    x = ks.layers.BatchNormalization()(x)
    x = ks.layers.Activation('relu')(x)

    # Final output layer
    outputs = ks.layers.Conv2D(1, (1, 1), activation='sigmoid')(x)

    model = tf.keras.Model(inputs=base_model.input, outputs=outputs)

    # Freeze the layers of VGG16 (optional)
    for layer in base_model.layers:
        layer.trainable = False

    return model

# Define the U-Net model with VGG16
model_transfer = unet_with_vgg16((128, 128, 3))

model_transfer.compile(optimizer='adam',
                       loss='binary_crossentropy',
                       metrics=[F1_score,
                                tf.keras.metrics.FalseNegatives(),
                                tf.keras.metrics.FalsePositives(),
                                tf.keras.metrics.TrueNegatives(),
                                tf.keras.metrics.TruePositives()])

model_transfer.summary()
```

```
Model: "model_1"
______________________________________________________________________
____________________________
 Layer (type)                Output Shape                 Param #
Connected to
======================================================================
============================
 input_2 (InputLayer)        [(None, 128, 128, 3)]        0          []



 block1_conv1 (Conv2D)       (None, 128, 128, 64)         1792
['input_2[0][0]']
```

```
 block1_conv2 (Conv2D)       (None, 128, 128, 64)         36928
['block1_conv1[0][0]']


 block1_pool (MaxPooling2D)  (None, 64, 64, 64)           0
['block1_conv2[0][0]']


 block2_conv1 (Conv2D)       (None, 64, 64, 128)          73856
['block1_pool[0][0]']


 block2_conv2 (Conv2D)       (None, 64, 64, 128)          147584
['block2_conv1[0][0]']


 block2_pool (MaxPooling2D)  (None, 32, 32, 128)          0
['block2_conv2[0][0]']


 block3_conv1 (Conv2D)       (None, 32, 32, 256)          295168
['block2_pool[0][0]']


 block3_conv2 (Conv2D)       (None, 32, 32, 256)          590080
['block3_conv1[0][0]']


 block3_conv3 (Conv2D)       (None, 32, 32, 256)          590080
['block3_conv2[0][0]']


 block3_pool (MaxPooling2D)  (None, 16, 16, 256)          0
['block3_conv3[0][0]']


 block4_conv1 (Conv2D)       (None, 16, 16, 512)          1180160
['block3_pool[0][0]']


 block4_conv2 (Conv2D)       (None, 16, 16, 512)          2359808
['block4_conv1[0][0]']


 block4_conv3 (Conv2D)       (None, 16, 16, 512)          2359808
['block4_conv2[0][0]']
```

```
 block4_pool (MaxPooling2D)  (None, 8, 8, 512)            0
['block4_conv3[0][0]']


 block5_conv1 (Conv2D)       (None, 8, 8, 512)            2359808
['block4_pool[0][0]']


 block5_conv2 (Conv2D)       (None, 8, 8, 512)            2359808
['block5_conv1[0][0]']


 block5_conv3 (Conv2D)       (None, 8, 8, 512)            2359808
['block5_conv2[0][0]']


 block5_pool (MaxPooling2D)  (None, 4, 4, 512)            0
['block5_conv3[0][0]']


 up_sampling2d_3 (UpSamplin  (None, 8, 8, 512)            0
['block5_pool[0][0]']
 g2D)



 concatenate_3 (Concatenate  (None, 8, 8, 1024)           0
['up_sampling2d_3[0][0]',
 )
'block5_conv3[0][0]']


 conv2d_4 (Conv2D)           (None, 8, 8, 512)            4719104
['concatenate_3[0][0]']


 batch_normalization_3 (Bat  (None, 8, 8, 512)            2048
['conv2d_4[0][0]']
 chNormalization)



 activation_3 (Activation)   (None, 8, 8, 512)            0
['batch_normalization_3[0][0]'
                                                                     ]



 up_sampling2d_4 (UpSamplin  (None, 16, 16, 512)          0
['activation_3[0][0]']
```

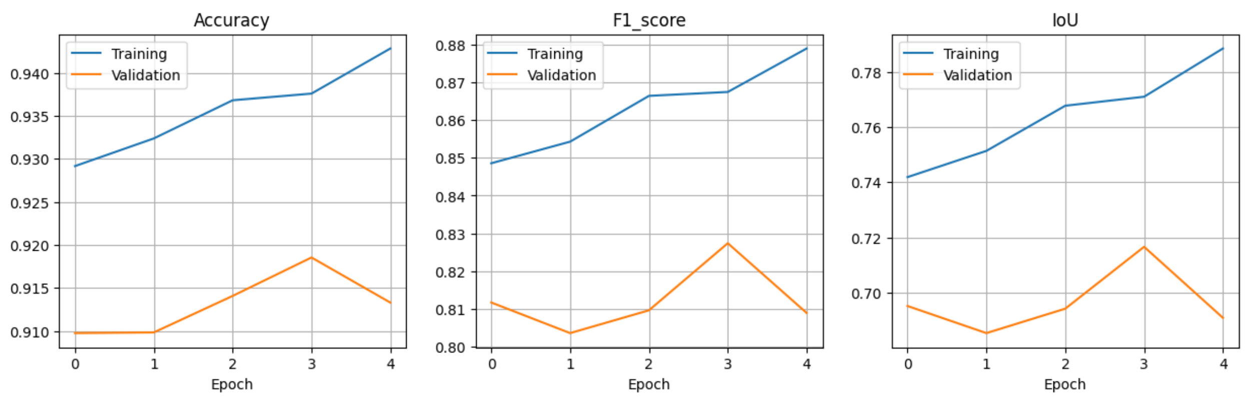
```
 g2D)



 concatenate_4 (Concatenate  (None, 16, 16, 1024)         0
['up_sampling2d_4[0][0]',
 )
'block4_conv3[0][0]']


 conv2d_5 (Conv2D)           (None, 16, 16, 512)          4719104
['concatenate_4[0][0]']


 batch_normalization_4 (Bat  (None, 16, 16, 512)          2048
['conv2d_5[0][0]']
 chNormalization)



 activation_4 (Activation)   (None, 16, 16, 512)          0
['batch_normalization_4[0][0]'
                                                                    ]



 up_sampling2d_5 (UpSamplin  (None, 32, 32, 512)          0
['activation_4[0][0]']
 g2D)



 concatenate_5 (Concatenate  (None, 32, 32, 768)          0
['up_sampling2d_5[0][0]',
 )
'block3_conv3[0][0]']


 conv2d_6 (Conv2D)           (None, 32, 32, 256)          1769728
['concatenate_5[0][0]']


 batch_normalization_5 (Bat  (None, 32, 32, 256)          1024
['conv2d_6[0][0]']
 chNormalization)



 activation_5 (Activation)   (None, 32, 32, 256)          0
['batch_normalization_5[0][0]'
                                                                    ]
 up_sampling2d_6 (UpSamplin  (None, 64, 64, 256)          0
['activation_5[0][0]']
 g2D)



 concatenate_6 (Concatenate  (None, 64, 64, 384)          0
['up_sampling2d_6[0][0]',
 )
'block2_conv2[0][0]']


 conv2d_7 (Conv2D)           (None, 64, 64, 128)          442496
['concatenate_6[0][0]']


 batch_normalization_6 (Bat  (None, 64, 64, 128)          512
['conv2d_7[0][0]']
 chNormalization)



 activation_6 (Activation)   (None, 64, 64, 128)          0
['batch_normalization_6[0][0]'
                                                                    ]



 up_sampling2d_7 (UpSamplin  (None, 128, 128, 128)        0
['activation_6[0][0]']
 g2D)



 concatenate_7 (Concatenate  (None, 128, 128, 192)        0
['up_sampling2d_7[0][0]',
 )
'block1_conv2[0][0]']


 conv2d_8 (Conv2D)           (None, 128, 128, 64)         110656
['concatenate_7[0][0]']


 batch_normalization_7 (Bat  (None, 128, 128, 64)         256
['conv2d_8[0][0]']
 chNormalization)
 activation_7 (Activation)    (None, 128, 128, 64)         0
['batch_normalization_7[0][0]'
                                                                      ]



 conv2d_9 (Conv2D)            (None, 128, 128, 1)          65
['activation_7[0][0]']


======================================================================
============================
Total params: 26481729 (101.02 MB)
Trainable params: 11764097 (44.88 MB)
Non-trainable params: 14717632 (56.14 MB)
______________________________________________________________________
____________________________
```

Task 2.2 Train the transfer learning model and plot the training history

Feel free to use the `plot_training_history` function from the provided library `utilities.py`

```
# Train the model
start_time = time()
training_history = model_transfer.fit(X_train_normalized, y_train,
                             batch_size=32,
                             epochs=1,
                             validation_split=0.2,  # Use 20% of the
data for validation
                             shuffle=True)
end_time = time()

Epoch 1/5
88/88 [==============================] - 1946s 22s/step - loss: 0.1831
- F1_score: 0.8486 - false_negatives_1: 2090635.0000 -
false_positives_1: 1159282.0000 - true_negatives_1: 33285048.0000 -
true_positives_1: 9340235.0000 - val_loss: 0.2223 - val_F1_score:
0.8117 - val_false_negatives_1: 668751.0000 - val_false_positives_1:
366310.0000 - val_true_negatives_1: 8074420.0000 -
val_true_positives_1: 2359319.0000
Epoch 2/5
88/88 [==============================] - 1773s 20s/step - loss: 0.1713
- F1_score: 0.8543 - false_negatives_1: 2054361.0000 -
false_positives_1: 1048028.0000 - true_negatives_1: 33396308.0000 -
```

```
true_positives_1: 9376509.0000 - val_loss: 0.2180 - val_F1_score:
0.8036 - val_false_negatives_1: 776808.0000 - val_false_positives_1:
257343.0000 - val_true_negatives_1: 8183387.0000 -
val_true_positives_1: 2251262.0000
Epoch 3/5
88/88 [==============================] - 1701s 19s/step - loss: 0.1601
- F1_score: 0.8664 - false_negatives_1: 1846222.0000 -
false_positives_1: 1052727.0000 - true_negatives_1: 33391608.0000 -
true_positives_1: 9584648.0000 - val_loss: 0.2296 - val_F1_score:
0.8096 - val_false_negatives_1: 792022.0000 - val_false_positives_1:
193464.0000 - val_true_negatives_1: 8247266.0000 -
val_true_positives_1: 2236048.0000
Epoch 4/5
88/88 [==============================] - 1745s 20s/step - loss: 0.1567
- F1_score: 0.8675 - false_negatives_1: 1788067.0000 -
false_positives_1: 1075300.0000 - true_negatives_1: 33369028.0000 -
true_positives_1: 9642803.0000 - val_loss: 0.2288 - val_F1_score:
0.8274 - val_false_negatives_1: 666090.0000 - val_false_positives_1:
268190.0000 - val_true_negatives_1: 8172540.0000 -
val_true_positives_1: 2361980.0000
Epoch 5/5
88/88 [==============================] - 1673s 19s/step - loss: 0.1447
- F1_score: 0.8790 - false_negatives_1: 1648799.0000 -
false_positives_1: 974043.0000 - true_negatives_1: 33470288.0000 -
true_positives_1: 9782071.0000 - val_loss: 0.2222 - val_F1_score:
0.8089 - val_false_negatives_1: 806695.0000 - val_false_positives_1:
187703.0000 - val_true_negatives_1: 8253027.0000 -
val_true_positives_1: 2221375.0000

keys_to_rename = ['false_negatives_1', 'false_positives_1',
'true_negatives_1', 'true_positives_1',
                  'val_false_negatives_1', 'val_false_positives_1',
'val_true_negatives_1', 'val_true_positives_1']

for key in keys_to_rename:
    new_key = key.replace('_1', '')
    training_history.history[new_key] =
training_history.history.pop(key)

# Plot the training history
plot_training_history(training_history)
training_time = (end_time - start_time)/3600 # in hours
print('It took %.2f hours to train the model'%(training_time))
```



```
16/16 [==============================] - 110s 7s/step
```

Part 3: Training your model Orion

Use the lecture slides from the Orion-lecture to get started.

1. Put one of your model implementations into a python script (`.py`)
2. Transfer that script to Orion.
3. Change the relevant path variables in your python script (path-to-data for example), and make sure that you record the time it takes to train the model in the script. This can be done using the `time` library for example.
4. Set up a SLURM-script to train your model, please use the example from the Orion lecture as a base.
5. Submit your SLURM job, and let the magic happen.

If you wish to use a model trained on Orion to make a Kaggle submission, remeber to save the model, such that you can transfer it to your local computer to make a prediction on `X_test`, or test the model on Orion directly if you want to.

Tips

If you compiled, trained and stored a model on Orion with a custom performance metric (such as F1-score), remember to specify that metric when loading the model on your computer again.

Loading a saved model:

```
trained_model =
tf.keras.models.load_model('some/path/to/my_trained_model.keras',
custom_objects={'F1_score': F1_score})
```

Loading a checkpoint:

```
trained_model =
tf.keras.saving.load_model('some/path/to/my_trained_model_checkpoint',
custom_objects={'F1_score': F1_score})
```

Discussion

Question 1: Which model architectures did you explore, and what type of hyperparameter optimization did you try?

Answer 1: We explored the U-Net architecture for semantic segmentation. Additionally, we applied transfer learning by using a pre-trained backbone. We experimented with various available models, such as VGG16, ResNet50, and ResNet50V2. To optimize hyperparameters, we made adjustments to the batch size, number of epochs, the quantity of neurons in each layer, and the depth of the network by varying the number of layers.

Question 2: Which of the model(s) did you choose to train on Orion, and how long did it take to train it on Orion?

Answer 2: We opted for the U-Net model on Orion and also experimented with transfer learning using the VGG16 model. The training process on Orion proved to be significantly faster compared to our own laptop. For the same training model configuration on our laptop, it took 10,800 seconds, while on Orion, it only required 473 seconds. Training on Orion was approximately 20 times faster than on a personal computer.

Question 3: What where the biggest challenges with this assignment?

Answer 3:

o The biggest challenges with this assignment were training the model on Orion. It took some time to get used to it. However, once we understood it, it was no longer a challenge.

o Another challenge was hyperparameter optimization. We had to do this manually, which is why we found it challenging. In the future, we would like to explore methods and techniques for performing hyperparameter optimization using scripting, so that parameters can be adjusted automatically.

o Additionally, we encountered challenges in improving the accuracy of the model. Despite making significant parameter changes, our efforts didn't lead to a substantial accuracy improvement. For example, in our Kaggle submission, we achieved an initial accuracy of 0.833, which we later improved to 0.888, resulting in a 6.6% increase. Perhaps by employing optimization techniques or other methods, we can further enhance the model's accuracy.

Kaggle submission

Evaluate your best model on the test dataset and submit your prediction to the Kaggle leaderboard. Link to the Kaggle leaderboard will be posted in the Canvas assignment.

```
USER_DETERMINED_THRESHOLD = 0.5
y_pred      = model_transfer.predict(X_test_normalized)            #
Make prediction
flat_y_pred = y_pred.flatten()                            # Flatten
prediction
flat_y_pred[flat_y_pred >= USER_DETERMINED_THRESHOLD] = 1 # Binarize
prediction (Optional, depends on output activation used)
flat_y_pred[flat_y_pred != 1]   = 0                       # Binarize
prediction (Optional, depends on output activation used)
submissionDF = pd.DataFrame()
submissionDF['ID'] = range(len(flat_y_pred))              # The
submission csv file must have a column called 'ID'
submissionDF['Prediction'] = flat_y_pred
submissionDF.to_csv('submission1.csv', index=False)
```

```
16/16 [==============================] - 110s 7s/step
```