```
################################# "FUNCTIONAL" INPUTS
################################# START
## Input
PATH_TO_DATASET     = '/mnt/users/dat300-h23-31/trans/student_TGS_challenge.h5'
MODEL_NAME          = 'student_TGS'

# Parameters
BATCH_SIZE = 32
EPOCHS = 100

## Outputs
# Model file
PATH_TO_STORE_MODEL = './'
################################# "FUNCTIONAL" INPUTS
################################# END


from tqdm import tqdm # Cool progress bar

import tensorflow as tf

from utilities import *
from visualization import *

import time
import os
from time import time
import numpy as np
import pandas as pd
import tensorflow.keras as ks
import seaborn as sns
import h5py
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
import keras
import keras
from keras.layers import UpSampling2D, Concatenate
from keras.layers import Input, Conv2D, MaxPooling2D, Activation,
BatchNormalization, Dropout, Conv2DTranspose, concatenate
from tensorflow.keras.metrics import FalseNegatives, FalsePositives, TrueNegatives,
TruePositives
from tensorflow.keras.metrics import MeanIoU, Precision, Recall
from utilities import F1_score
from visualization import plot_training_history

SEED = 458
RNG = np.random.default_rng(SEED)
tf.random.set_seed(SEED)



#dataset_path = './student_TGS_challenge.h5'

# Joining paths
PATH_TO_STORE_TRAINING_HISTORY = os.path.join(PATH_TO_STORE_MODEL, MODEL_NAME +
'_training_history.png')
PATH_TO_STORE_MODEL = os.path.join(PATH_TO_STORE_MODEL, MODEL_NAME + '.keras')
```

```
# Function definition
def plot_training_history(training_history_object, list_of_metrics=None):
    """
    Input:
        training_history_object:: Object returned by model.fit() function in keras
        list_of_metrics        :: A list of metrics to be plotted. Use if you only
                                  want to plot a subset of the total set of metrics
                                  in the training history object. By Default it
will
                                  plot all of them in individual subplots.
    Output:
        fig                    :: Figure object from matplotlib
    """
    history_dict = training_history_object.history
    if list_of_metrics is None:
        list_of_metrics = [key for key in list(history_dict.keys()) if 'val_' not
in key]
    trainHistDF = pd.DataFrame(history_dict)
    # trainHistDF.head()
    train_keys = list_of_metrics
    valid_keys = ['val_' + key for key in train_keys]
    nr_plots = len(train_keys)
    fig, ax = plt.subplots(1,nr_plots,figsize=(5*nr_plots,4))
    for i in range(len(train_keys)):
        ax[i].plot(np.array(trainHistDF[train_keys[i]]), label='Training')
        ax[i].plot(np.array(trainHistDF[valid_keys[i]]), label='Validation')
        ax[i].set_xlabel('Epoch')
        ax[i].set_title(train_keys[i])
        ax[i].grid('on')
        ax[i].legend()
    fig.tight_layout()
    return fig

with h5py.File(PATH_TO_DATASET, '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]))

    print('Shapes before data preprocessing:')
    print('X_train shape:', X_train.shape)
    print('y_train shape:', y_train.shape)
    print('X_test shape:', X_test.shape)

# Normalizing input between [0,1]
X_train_normalized = X_train.astype("float32") / np.max(X_train)
X_test_normalized = X_test.astype("float32") / np.max(X_test)

print('Shapes after data preprocessing:')
print('X_train shape:', X_train.shape)
print('y_train shape:', y_train.shape)
print('X_test shape:', X_test.shape)

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,'accuracy',
                                tf.keras.metrics.FalseNegatives(),
                                tf.keras.metrics.FalsePositives(),
                                tf.keras.metrics.TrueNegatives(),
                                tf.keras.metrics.TruePositives()])

model_transfer.summary()
```

```
from time import time
# Train the model
start_time = time()
history = model_transfer.fit(
          X_train_normalized, y_train,
          validation_split=0.2,
          batch_size=BATCH_SIZE,
          epochs=EPOCHS,
          verbose=1,
          use_multiprocessing=True,
          workers=16)
end_time = time()

# Storing model and training history
training_time = end_time - start_time
print('It took %.2f seconds to train the model for %i epochs'%(training_time,
EPOCHS))
print('Storing model in %s'%PATH_TO_STORE_MODEL)
print('Storing training history in %s'%PATH_TO_STORE_TRAINING_HISTORY)
model_transfer.save(PATH_TO_STORE_MODEL)

# Plotting training history
history_plot = plot_training_history(history)
history_plot.savefig(PATH_TO_STORE_TRAINING_HISTORY)

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('submission20.csv', index=False)        # Remember to store the
dataframe to csv without the nameless index column.
```