nb0a7xpxr

February 4, 2025

[ ]:
```python
import os
import numpy as np
import cv2
from glob import glob
import tensorflow as tf
from sklearn.model_selection import train_test_split
import imgaug.augmenters as iaa
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import skimage.io as io
import skimage.transform as trans
import tensorflow.keras.layers as layers
from keras.models import *
from keras.optimizers import *
from keras.callbacks import ModelCheckpoint, LearningRateScheduler
from keras import backend as keras
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,␣
↪ReduceLROnPlateau, CSVLogger, TensorBoard
from tensorflow.keras.metrics import Recall, Precision
from tqdm import tqdm
```

[ ]:
```python
IMG_H = 512
IMG_W = 512
```

[ ]:
```python
def load_dataset(path, split=0.1):
    """ Loading the images and masks """
    X = sorted(glob(os.path.join(path, "images", "*")))
    Y = sorted(glob(os.path.join(path, "masks", "*")))

    """ Spliting the data into training and testing """
    split_size = int(len(X) * split)

    train_x, valid_x = train_test_split(X, test_size=split_size,␣
↪random_state=42)
    train_y, valid_y = train_test_split(Y, test_size=split_size,␣
↪random_state=42)
```

```
    train_x, test_x = train_test_split(train_x, test_size=split_size,␣
↪random_state=42)
    train_y, test_y = train_test_split(train_y, test_size=split_size,␣
↪random_state=42)

    return (train_x, train_y), (valid_x, valid_y), (test_x, test_y)

def read_image(path):
    path = path.decode()
    image = cv2.imread(path, cv2.IMREAD_COLOR)
    image = cv2.resize(image, (IMG_W, IMG_H))
    image = image / 255.0
    image = image.astype(np.float32)
    return image

def read_mask(path):
    path = path.decode()
    mask = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    mask = cv2.resize(mask, (IMG_W, IMG_H))
    mask = mask / 255.0
    mask = mask.astype(np.float32)
    mask = np.expand_dims(mask, axis=-1)
    return mask

def tf_parse(x, y):
    def _parse(x, y):
        x = read_image(x)
        y = read_mask(y)
        return x, y

    x, y = tf.numpy_function(_parse, [x, y], [tf.float32, tf.float32])
    x.set_shape([IMG_H, IMG_W, 3])
    y.set_shape([IMG_H, IMG_W, 1])
    return x, y

def tf_dataset(X, Y, batch=2):
    ds = tf.data.Dataset.from_tensor_slices((X, Y))
    ds = ds.map(tf_parse).batch(batch).prefetch(10)
    return ds
```

[ ]:
```
def conv_block(x, kernelsize, filters, dropout, batchnorm=False):
    conv = layers.Conv2D(filters, (kernelsize, kernelsize),␣
↪kernel_initializer='he_normal', padding="same")(x)
    if batchnorm is True:
        conv = layers.BatchNormalization(axis=3)(conv)
    conv = layers.Activation("relu")(conv)
    if dropout > 0:
```

```python
        conv = layers.Dropout(dropout)(conv)
    conv = layers.Conv2D(filters, (kernelsize, kernelsize), kernel_initializer='he_normal', padding="same")(conv)
    if batchnorm is True:
        conv = layers.BatchNormalization(axis=3)(conv)
    conv = layers.Activation("relu")(conv)
    return conv

def encoder_block(x, num_filters, dropout, batchnorm):
    x = conv_block(x, kernelsize=3, filters=num_filters, dropout=dropout, batchnorm=batchnorm)
    x = conv_block(x, kernelsize=3, filters=num_filters, dropout=dropout, batchnorm=batchnorm)

    p = layers.MaxPool2D((2, 2))(x)
    return x, p

def unet3plus(input_shape, num_classes=1, dropout=0.3, batchnorm=True):
    """ Inputs """
    inputs = layers.Input(input_shape, name="input_layer")

    """ Encoder """
    e1, p1 = encoder_block(inputs, 64, dropout=dropout, batchnorm=batchnorm)
    e2, p2 = encoder_block(p1, 128, dropout=dropout, batchnorm=batchnorm)
    e3, p3 = encoder_block(p2, 256, dropout=dropout, batchnorm=batchnorm)
    e4, p4 = encoder_block(p3, 512, dropout=dropout, batchnorm=batchnorm)

    """ Bottleneck """
    e5 = conv_block(p4, kernelsize=3, filters=1024, dropout=dropout, batchnorm=batchnorm)
    e5 = conv_block(e5, kernelsize=3, filters=1024, dropout=dropout, batchnorm=batchnorm)

    """ Decoder 4 """
    e1_d4 = layers.MaxPool2D((8, 8))(e1)
    e1_d4 = conv_block(e1_d4, kernelsize=3, filters=64, dropout=dropout, batchnorm=batchnorm)

    e2_d4 = layers.MaxPool2D((4, 4))(e2)
    e2_d4 = conv_block(e2_d4, kernelsize=3, filters=64, dropout=dropout, batchnorm=batchnorm)

    e3_d4 = layers.MaxPool2D((2, 2))(e3)
    e3_d4 = conv_block(e3_d4, kernelsize=3, filters=64, dropout=dropout, batchnorm=batchnorm)
```

```
    e4_d4 = conv_block(e4, kernelsize=3, filters=64, dropout=dropout,␣
↪batchnorm=batchnorm)

    e5_d4 = layers.UpSampling2D((2, 2), interpolation="bilinear")(e5)
    e5_d4 = conv_block(e5_d4, kernelsize=3, filters=64, dropout=dropout,␣
↪batchnorm=batchnorm)

    d4 = layers.Concatenate()([e1_d4, e2_d4, e3_d4, e4_d4, e5_d4])
    d4 = conv_block(d4, kernelsize=3, filters=64*5, dropout=dropout,␣
↪batchnorm=batchnorm)

    """ Decoder 3 """
    e1_d3 = layers.MaxPool2D((4, 4))(e1)
    e1_d3 = conv_block(e1_d3, kernelsize=3, filters=64, dropout=dropout,␣
↪batchnorm=batchnorm)

    e2_d3 = layers.MaxPool2D((2, 2))(e2)
    e2_d3 = conv_block(e2_d3, kernelsize=3, filters=64, dropout=dropout,␣
↪batchnorm=batchnorm)

    e3_d3 = conv_block(e3, kernelsize=3, filters=64, dropout=dropout,␣
↪batchnorm=batchnorm)

    d4_d3 = layers.UpSampling2D((2, 2), interpolation="bilinear")(d4)
    d4_d3 = conv_block(d4_d3, kernelsize=3, filters=64, dropout=dropout,␣
↪batchnorm=batchnorm)

    e5_d3 = layers.UpSampling2D((4, 4), interpolation="bilinear")(e5)
    e5_d3 = conv_block(e5_d3, kernelsize=3, filters=64, dropout=dropout,␣
↪batchnorm=batchnorm)

    d3 = layers.Concatenate()([e1_d3, e2_d3, e3_d3, d4_d3, e5_d3])
    d3 = conv_block(d3, kernelsize=3, filters=64*5, dropout=dropout,␣
↪batchnorm=batchnorm)

    """ Decoder 2 """
    e1_d2 = layers.MaxPool2D((2, 2))(e1)
    e1_d2 = conv_block(e1_d2, kernelsize=3, filters=64, dropout=dropout,␣
↪batchnorm=batchnorm)

    e2_d2 = conv_block(e2, kernelsize=3, filters=64, dropout=dropout,␣
↪batchnorm=batchnorm)

    d3_d2 = layers.UpSampling2D((2, 2), interpolation="bilinear")(d3)
    d3_d2 = conv_block(d3_d2, kernelsize=3, filters=64, dropout=dropout,␣
↪batchnorm=batchnorm)
```

```
    d4_d2 = layers.UpSampling2D((4, 4), interpolation="bilinear")(d4)
    d4_d2 = conv_block(d4_d2, kernelsize=3, filters=64, dropout=dropout, batchnorm=batchnorm)

    e5_d2 = layers.UpSampling2D((8, 8), interpolation="bilinear")(e5)
    e5_d2 = conv_block(e5_d2, kernelsize=3, filters=64, dropout=dropout, batchnorm=batchnorm)

    d2 = layers.Concatenate()([e1_d2, e2_d2, d3_d2, d4_d2, e5_d2])
    d2 = conv_block(d2, kernelsize=3, filters=64*5, dropout=dropout, batchnorm=batchnorm)

    """ Decoder 1 """
    e1_d1 = conv_block(e1, kernelsize=3, filters=64, dropout=dropout, batchnorm=batchnorm)

    d2_d1 = layers.UpSampling2D((2, 2), interpolation="bilinear")(d2)
    d2_d1 = conv_block(d2_d1, kernelsize=3, filters=64, dropout=dropout, batchnorm=batchnorm)

    d3_d1 = layers.UpSampling2D((4, 4), interpolation="bilinear")(d3)
    d3_d1 = conv_block(d3_d1, kernelsize=3, filters=64, dropout=dropout, batchnorm=batchnorm)

    d4_d1 = layers.UpSampling2D((8, 8), interpolation="bilinear")(d4)
    d4_d1 = conv_block(d4_d1, kernelsize=3, filters=64, dropout=dropout, batchnorm=batchnorm)

    e5_d1 = layers.UpSampling2D((16, 16), interpolation="bilinear")(e5)
    e5_d1 = conv_block(e5_d1, kernelsize=3, filters=64, dropout=dropout, batchnorm=batchnorm)

    d1 = layers.Concatenate()([e1_d1, d2_d1, d3_d1, d4_d1, e5_d1])
    d1 = conv_block(d1, kernelsize=3, filters=64*5, dropout=dropout, batchnorm=batchnorm)

    """ Final Output """
    # No deep supervision, just a single output
    y1 = layers.Conv2D(num_classes, kernel_size=1, padding="same")(d1)
    y1 = layers.Activation("sigmoid")(y1)

    outputs = [y1]

    model = tf.keras.Model(inputs, outputs)
    return model
```

```
if __name__ == "__main__":
    input_shape = (256, 256, 3)
    model = unet3plus(input_shape)
    model.summary()
```

[ ]:
```
smooth = 1e-15

def dice_coef(y_true, y_pred):
    intersection = tf.reduce_sum(y_true * y_pred)
    return (2.0 * intersection + smooth) / (tf.reduce_sum(y_true) + tf.
 ↪reduce_sum(y_pred) + smooth)

def dice_loss(y_true, y_pred):
    return 1.0 - dice_coef(y_true, y_pred)
```

[ ]:
```
def create_dir(path):
    if not os.path.exists(path):
        os.makedirs(path)

if __name__ == "__main__":
    """ Seeding """
    np.random.seed(42)
    tf.random.set_seed(42)

    """ Directory for storing files """
    create_dir("files")

    """ Hyperparameters """
    batch_size = 2
    lr = 1e-4
    num_epochs = 50
    model_path = os.path.join("files", "model.keras")
    csv_path = os.path.join("files", "log.csv")

    """ Dataset """
    dataset_path = "Kvasir-SEG"
    (train_x, train_y), (valid_x, valid_y), (test_x, test_y) =␣
 ↪load_dataset(dataset_path)

    print(f"Train: \t{len(train_x)} - {len(train_y)}")
    print(f"Valid: \t{len(valid_x)} - {len(valid_y)}")
    print(f"Test: \t{len(test_x)} - {len(test_y)}")

    train_dataset = tf_dataset(train_x, train_y, batch=batch_size)
    valid_dataset = tf_dataset(valid_x, valid_y, batch=batch_size)
```

```
    """ Model """
    model = unet3plus((IMG_H, IMG_W, 3))
    metrics = ["acc", tf.keras.metrics.Recall(), tf.keras.metrics.Precision(),␣
↪dice_coef]
    model.compile(loss=dice_loss, optimizer=Adam(lr), metrics=metrics)

    callbacks = [
        ModelCheckpoint(model_path, verbose=1, save_best_only=True),
        ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=2,␣
↪min_lr=1e-10, verbose=1),
        CSVLogger(csv_path),
        TensorBoard(),
        EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
    ]

    model.fit(
        train_dataset,
        validation_data=valid_dataset,
        epochs=num_epochs,
        steps_per_epoch=1000,
        callbacks=callbacks
    )
```

[ ]:
```
def read_image(path):
    x = cv2.imread(path, cv2.IMREAD_COLOR)
    x = cv2.resize(x, (IMG_W, IMG_H))
    x = x/255.0
    return x

def read_mask(path):
    x = cv2.imread(y, cv2.IMREAD_GRAYSCALE)
    x = cv2.resize(x, (IMG_W, IMG_H))
    x = x / 255.0
    x = np.expand_dims(x, axis=-1)
    x = np.concatenate([x, x, x], axis=-1)
    return x

if __name__ == "__main__":
    """ Seeding """
    np.random.seed(42)
    tf.random.set_seed(42)

    """ Directory for storing files """
    create_dir(f"results")

    """ Load the model """
```

```python
    model_path = os.path.join("files", "model.keras")
    model = tf.keras.models.load_model(model_path, custom_objects={"dice_loss":␣
↪dice_loss, "dice_coef": dice_coef})

    """ Dataset """
    dataset_path = "/content/drive/MyDrive/kvasir-seg/Kvasir-SEG"
    (train_x, train_y), (valid_x, valid_y), (test_x, test_y) =␣
↪load_dataset(dataset_path)

    print(f"Train: \t{len(train_x)} - {len(train_y)}")
    print(f"Valid: \t{len(valid_x)} - {len(valid_y)}")
    print(f"Test: \t{len(test_x)} - {len(test_y)}")

    """ Prediction """
    for x, y in tqdm(zip(test_x, test_y), total=len(test_x)):
        """ Extracting the name """
        name = x.split("/")[-1].split(".")[0]

        """ Reading the image """
        x = read_image(x)

        """ Reading the mask """
        y = read_mask(y)

        """ Prediction """
        x = np.expand_dims(x, axis=0)
        pred = model.predict(x, verbose=0)[0]
        pred = np.concatenate([pred, pred, pred], axis=-1)
        # pred = (pred > 0.5).astype(np.int32)

        """ Save final mask """
        line = np.ones((IMG_H, 10, 3)) * 255
        cat_images = np.concatenate([x[0], line, y*255, line, pred*255], axis=1)
        save_image_path = os.path.join("results",  f"{name}.jpg")
        cv2.imwrite(save_image_path, cat_images)
```