IDFY – LIPS AND EYES SEGMENTATION

SURYA KETARAJU

DOCUMENTATION

First, we import all the necessary libraries for importing and preprocessing the data. The data was scattered as images and masks, so I created new folders for images and their corresponding masks.

The data I have worked with here is very less, the original dataset had over 3 lac images which was slowing down my PC to a snail's pace. I have worked a tiny fraction of the data not with the intention of getting a high accuracy but simply to display the logic I have used, since the code is transferrable and will be the same even for a large dataset.

I have used the os module for looping through the folder for image ID and the data.

After adding the path for both training and test data:

```
TRAIN_PATH = '.../Downloads/image_seg/data/'

TEST_PATH = '.../Downloads/image_seg/test/'
```

We use the OS module to walk through the folders to gather the images.

```
train_ids = next(os.walk(TRAIN_PATH))[1]

test_ids = next(os.walk(TEST_PATH))[2]

for n, id_ in tqdm(enumerate(train_ids), total=len(train_ids)):

path = TRAIN_PATH + id_

img = io.imread(path + '/image/' + id_ + '.jpg')[:,:,:IMG_CHANNELS]
img = transform.resize(img, (IMG_HEIGHT, IMG_WIDTH), mode='constant',
preserve_range=bool)
```

Doing the same process again for getting mask data

```
for n, id_ in tqdm(enumerate(train_ids), total=len(train_ids)):
        path = TRAIN_PATH + id_
        img = io.imread(path + '/image/' + id_ + '.jpg')[:,:,:IMG_CHANNELS]
        img = transform.resize(img, (IMG_HEIGHT, IMG_WIDTH), mode='constant',
        preserve_range=bool)
        #fill X_train with the img value
        X_train[n] = img
        #gathering the mask files
        mask = np.zeros((IMG_HEIGHT, IMG_WIDTH, 1), dtype=np.bool)
        for mask_file in next(os.walk(path + '/mask/'))[2]:
        mask_new = io.imread(path + '/mask/' + mask_file)
        #reducing the channels
        mask_new = cv2.cvtColor(mask_new, cv2.COLOR_BGR2GRAY)
        #adding a dimension since the above command dropped all channels
        mask_new = np.expand_dims(transform.resize(mask_new, (IMG_HEIGHT,
        IMG_WIDTH), mode='reflect', preserve_range=bool), axis=-1)
        #masking
        mask = np.maximum(mask_new, mask)
```

After loading X_train with Image data, we load Y_train with mask data:

```
Y_train[n] = mask
```

The above logic will remain the same for gathering TEST DATA as well:

```
X_test = np.zeros((len(test_ids), IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS),
dtype=np.uint8)
sizes_test = []
for n, id_ in tqdm(enumerate(test_ids), total=len(test_ids)):
path = TEST_PATH
img = io.imread(path + id_)[:,:,:IMG_CHANNELS]
sizes_test.append([img.shape[0], img.shape[1]])
```

```
img = transform.resize(img, (IMG_HEIGHT, IMG_WIDTH), mode='constant',
preserve_range=True)
X_test[n] = img
```

We will divide our pixel data by 255, so the final values will range between 0 and 1 for easier processing by Keras

```
X_train = X_train/255
Y_train = Y_train/255
X_test = X_test/255
```

Printing out the images and corresponding masks to check we’re on the right track:

```
img_new = random.randint(0, len(train_ids))
io.imshow(X_train[img_new])
plt.show()
io.imshow(np.squeeze(Y_train[img_new]))
plt.show()
```

I have used Keras’ UNET architecture for segmentation, make sure you have the module installed on your machine (pip install keras-unet). Here we’re using the custom model.

```
from keras_unet.models import custom_unet
model = custom_unet(
                input_shape=(128, 128, 3),
                use_batch_norm=False,
                num_classes=1,
                filters=128,
                dropout=0.2,
                output_activation='sigmoid')
```

Since the final output will only have one channel, we have used sigmoid activation function.

```
model.summary()

model.compile(loss=’sparse_categorical_crossentropy’, optimizer=’adam’,
metrics=['accuracy'])
```

After compiling the model, we will finally fit our data. Since the data I have used is not vast, I thought it better to just train for 10 epochs rather than longer, as it will not have much difference in our case.

```
results = model.fit(X_train, Y_train, validation_split=0.1, epochs=10)
```

Plotting the accuracies and losses during the training process

```
from keras_unet.utils import plot_segm_history
plot_segm_history(results, metrics = ['accuracy','val_accuracy'], losses=['loss', 'val_loss'])
```

Finally, making the predictions and visualizing the masks.

```
preds_train = model.predict(X_train)
preds_test = model.predict(X_test)


preds_train_mask = (preds_train > 0.45)
preds_test_mask = (preds_test > 0.45)


#VISUALIZING THE MASKS
i = random.randint(0, len(preds_train_t))

io.imshow(X_train[i])
plt.show()
io.imshow(np.squeeze(Y_train[i]))
plt.show()
io.imshow(np.squeeze(preds_train_mask[i]))
plt.show()
```