Abhishek Singh Dhadwal
B.Tech, Computer Science and Engineering (Year-III)
Visvesvaraya National Institute of Technology, Nagpur

Investigative Report on the compatibility of MultiResUNet for facial segmentation

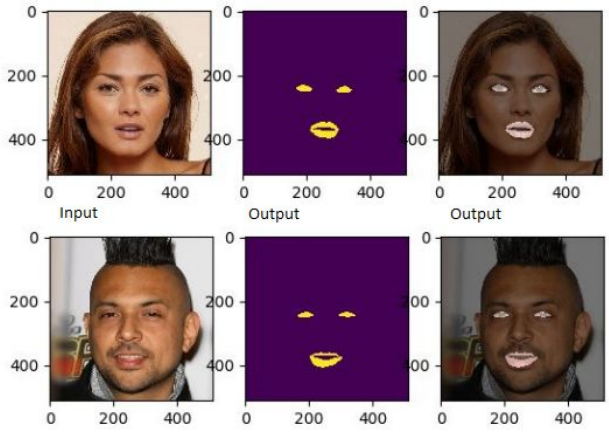
10th December 2019

OVERVIEW

As per the task provided by a company, we aim to utilise and discover the compatibility provided by MultiResUNet models, originally utilised for Medical Image segmentation, and check the feasibility of utilising the aforementioned method in order to provide segmentation of facial features (lips and eyes in our case) .

GOALS

1. Provision of segmented output masks along with their amalgamations as shown below:



2. No publicly documented/ researched methods of facial recognition should be utilised for the task.

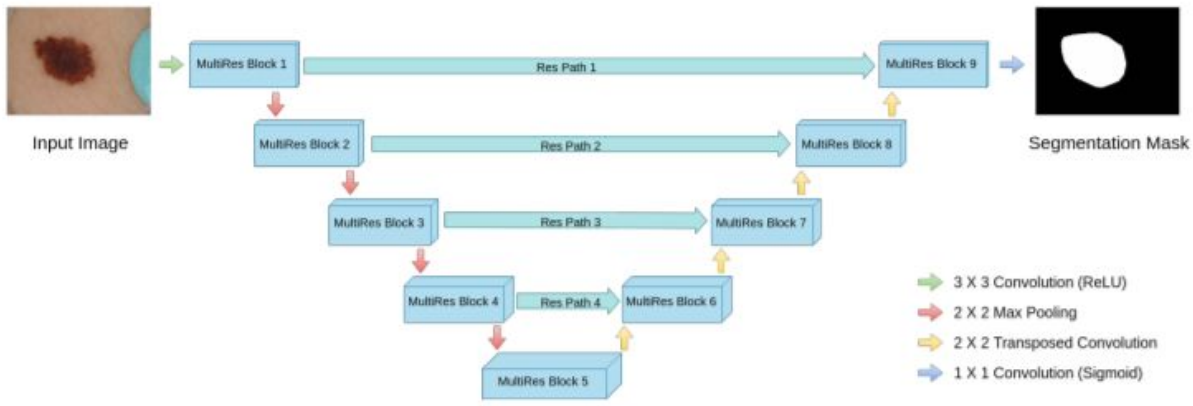
SPECIFICATIONS

- The MultiResUNet[1] model is an under-research development model for the ResNet architecture, primarily used for advancements in medical image segmentation.
- We utilise the CelebAMaskHQ[2] dataset in accordance to its license for educational purposes.
- Primary development, training and testing is performed by various Google Colab[3], a research tool for machine learning education and research. It's a Jupyter notebook environment that requires no setup to use.

WHY MULTI-RES-U-NET?

The UNet[4] model relies on the strong use of data augmentation to use the available annotated samples more efficiently. The architecture consists of a contracting path to capture context and a symmetric expanding path that enables precise localization. We show that such a network can be trained end-to-end from very few images and outperforms the prior best method (a sliding-window convolutional network).

The model is reportedly fast. Segmentation of a 512x512 image takes less than a second on a recent GPU according to the author[4].



Proposed MultiResUNet architecture

The MultiResUNet model improves upon the aforementioned model, and aids in the removal of the vanishing gradient issues, accompanied by improvements of up to 10% in accuracy.

PREPROCESSING, TRAINING AND EVALUATION

The following sections are describing entities of this notebook[5], which contains the model and all of the features mentioned.

Data Preprocessing

The runtime and memory constraints provided by Colab proved to be a challenge, given the sheer size of the dataset.

Irrespective of the method used, the following folders should be created, and should contain the corresponding images as mentioned below. Store these in Drive for the following format :

Head folder : **CelebAMaskHQ**

Subfolders :

- **images** : The input images for testing
- **l_eye** : Ground truths for left eye masks
- **r_eye** : Ground truths for right eye masks
- **l_lip** : Ground truths for lower lip masks
- **u_lip** : Ground truth for upper lip masks

The images can be found in the original CelebAMask-HQ zip file, and in order to search for masks specifically, just search the corresponding term in the masks folder. We store masks as per our requirements only in order to manage storage constraints on Drive.

Eg: To find images related to left eye masks, just search for l_eye in the masks folder.

The folders should contain images in the following format :

For image '0.jpg', the left eye mask should be '0_l_eye.png' and stored in the l_eye folder, the right eye mask should be '0_r_eye.png' and stored in the r_eye folder, the lower lip mask should be '0_l_lip.png' and stored in the l_lip folder, the upper lip mask should be '0_u_lip.png' and stored in the u_lip folder.

In order to format the images in the manner provided, just paste the images in their corresponding folders, and run the “EXTRA : Pre processing of input files” section at the bottom of the notebook. The code snippet will do the rest.

There are two methods for image processing created, and can be used as per the user’s discretion(Ref - Loading the Images section):

1. **The Straight-Forward (memory heavy, but simple) method :**

   - We go through each image, check if all the masks required are present (as the dataset is missing certain masks for around 2,000 to 3,000 images), and resize them to 256 x 192 in order to fit the network specifications

   - The images and ground truth masks are stored in X and Y respectively, and we use np.maximum to combine all the individual masks into one input mask.

   - A pseudo-checkpoint mechanism is also used in order to accommodate runtime interruptions, and saves the arrays after every 2000 iterations. Hence we can re-run our snippet even after interruptions, leading to minimum losses for production time.

2. **The Pseudo-Parallelism implementation (faster) method :**

   - We use similar individual tasks as above, but, in order to speed up runtime we have divided our data to three slices, and run all slices parallely in order to reduce the time required and handle any and all constraints. The individual notebooks used are referenced in the parent notebook, and can be run in order to speed up tasks.

   - In case if we want to combine the arrays created (X1,X2,X3,Y1,Y2,Y3) we can run the 'Run in order to derive data from original parsing method' code snippet.

   - This is not necessary as we can directly use the 'improved version' code snippet after getting our arrays, which is mentioned below.

Training

We use the **Adam optimizer** in combination with the **binary cross-entropy** loss function in order to train our network.

The X, Y lists are converted to numpy arrays for convenience.

Furthermore, the images are divided by 255 to bring down the pixel values to [0...1] range.

On the other hand the segmentation masks are converted to binary (0 or 1) values.Using Sklearn *train_test_split* we split the data randomly into 80% training and 20% testing data.

Evaluation

Evaluation metrics : Dice Coefficient, Jaccard Index, accuracy.

Since, at the time of creation we do not have any inbuilt methods for Dice Coefficients and the Jaccard Index, we have created functions for them in our notebook.

After every epoch, we compute the values of Jaccard Index and Dice Coefficient, and save the predicted segmentation of first 10 images. The best model is also saved.

NOTE : The pyplot diagram created here will not look like the final outputs defined by the task. This is done in order to compensate for memory constraints. For proper outputs, kindly use the demo below.

DEMO AND TESTED MODEL INFORMATION

This section is describing entities of our demo notebook[6], which contains the trained models along with input and output methods.

The demo notebook is created in a manner such that it does not need any access to any and all datasets created and can be used as per the user's convenience.

We use **gdown** in order to access the models and weights created by using the previous notebook. These models are stored on my drive and are accessible publicly, and can be downloaded by going through the links mentioned in the comments of the first snippet, or by simply uncommenting the :

```
# files.download('results_'+model_json.split('.')[0]+'/result_'+str(img_names[i].split('.')[0])+'.png')
```

line in the 'Boilerplate Code to Run Model' code snippet.

The user should upload the picture(s) they want to test, and run snippets according to the model they need.

Models researched and provided in the Demo:

Model F10k:

This model has been trained on a 80-20 split amongst the first 10,000 images of the dataset.

- Number of epochs: 20
- Time taken : [1:29:40, 267.74s/it]
- Jaccard Index(final) : 0.8452393049122288
- Dice Coefficient(final) : 0.9139967587791317
- Accuracy(final) : 99.80 %

Model FD10 :

The dataset for this model has been split twice:

1. Three sets of 10,000 images each.
2. Each set trained on a 80-20 split.

Each split of the data was used to train the model for 10 epochs each. The split was performed in order to compensate for RAM bottlenecks in our system.

- Number of epochs: 10+10+10 = 30
- Time taken : [2:29:04, 2958.94s/it]
- Jaccard Index(final) : 0.8331437322988224
- Dice Coefficient(final) : 0.9071035040844939
- Accuracy(final) : 99.70 %

EXTRA : Base Testing Model

This model returns only the left eye segmented as a mask and was made for TESTING PURPOSES ONLY

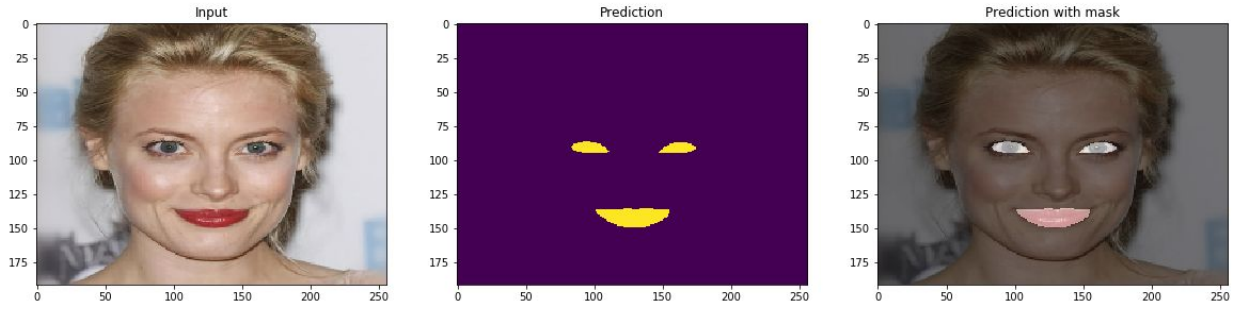
This model has been trained on a 80-20 split amongst the first 250 images of the dataset, and was used in order to test the original model mentioned in the paper.

- Number of epochs: 10
- Time taken : [0:20:40]
- Jaccard Index(final) : 0.39899180087352404
- Dice Coefficient(final) : 0.495551362130639337
- Accuracy(final) : 99.80 %

NOTE : THIS MODEL IS SIMPLY A PRECURSOR TO OUR ACTUAL MODELS MENTIONED ABOVE AND SHOULD NOT BE CONSIDERED AS FEASIBLE FOR ANY ASPECTS

CONCLUSION

The MultiResUNet model shows a lot of scope for non medical image segmentation usage, and can be utilised for facial segmentation for the above Dataset, with the **model FD10** being a current recommendation for the task. Further implementations with multiple variations can also be searched in order to maximize the potential of our given model.



Output for a random image using the FD10 model

FUTURE IMPROVEMENTS/SCOPE OF PROJECT:

These are improvements planned for the project to be implemented in its future versions.

1. Providing an input mechanism that will parse from the original file structure itself
2. Creating file reading formats without adulteration of original file names (i.e use lstrip(0) instead of renaming files themselves)
3. Test for possible overfitting and increase freedom for data splitting
4. Utilise and store callbacks for graphing improvements
5. **Usage of generators** instead of iterators in order to streamline processes and further improve memory efficiency.
6. Testing further variations in models, dataset sizes and parameters/optimizers.

BIBLIOGRAPHY

1. MultiResUNet : Rethinking the U-Net Architecture for Multimodal Biomedical Image Segmentation https://arxiv.org/pdf/1902.04049.pdf
2. CelebAMaskHQ - large-scale face image dataset that has **30,000** high-resolution face images selected from the CelebA dataset by following CelebA-HQ. https://github.com/switchablenorms/CelebAMask-HQ/blob/master/README.md
3. Google Colab - https://colab.research.google.com/
4. U-Net: Convolutional Networks for Biomedical Image Segmentation https://arxiv.org/pdf/1505.04597.pdf
5. Notebook containing model creation, preprocessing and evaluation data https://drive.google.com/file/d/1H26uaN10rU2V7MnX8vRdE3J0ZMoO7wq2/view?usp=sharing
6. The notebook containing our demo https://colab.research.google.com/drive/10Vb4Ukv4xOG35bS1sUAVp2j2YA2L1xjZ
7. The original MultiResUNet implementation https://github.com/nibtehaz/MultiResUNet