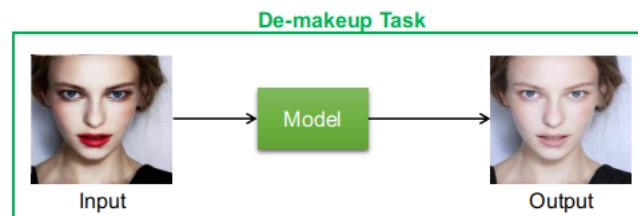


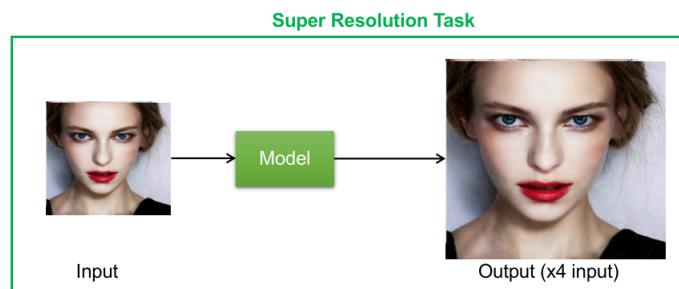
# De-makeup Project

Ngày 8 tháng 2 năm 2023

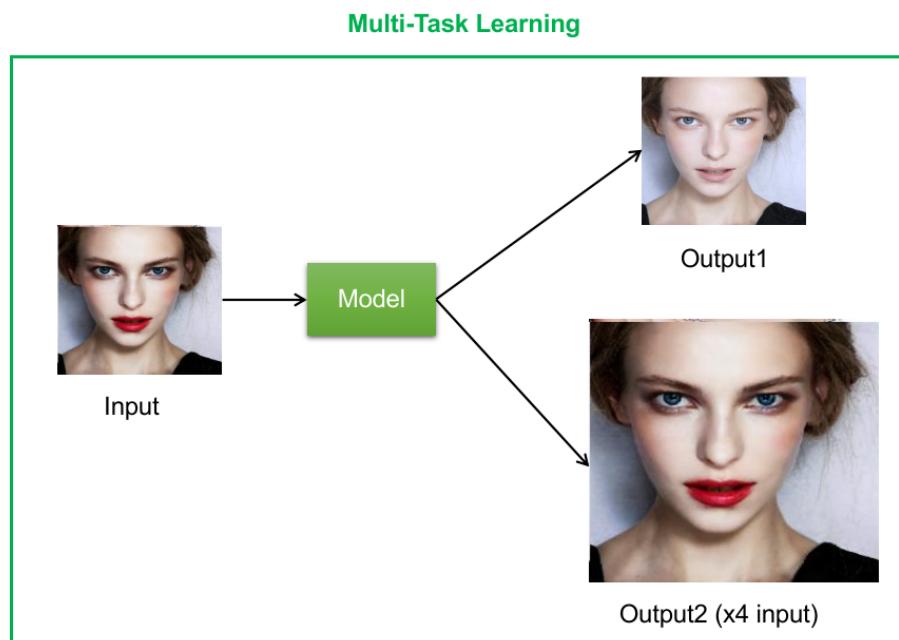
## Phần I: Mô Tả Project



Hình 1: De-makeup Task



Hình 2: Super Resolution Task



Hình 3: Multi-task learning

**Kiến thức sẽ sử dụng trong De-makeup Project:** Project sẽ là sự kết hợp kiến thức bao gồm kiến trúc Unet trong Domain Conversion, Style Transfer (sử dụng GAN), Super Resolution, và Multi-Task learning (các phần kiến thức mới các bạn có thể sẽ được học trong các tuần tới, trong project này một số kiến thức mới sẽ được dùng như black box, ví dụ như sử dụng GAN, Super Resolution model có sẵn hỗ trợ cho project).

**De-makeup Project:** Mục tiêu chính của project này sẽ thực hiện task de-makeup, model sẽ nhận input là ảnh với gương mặt đã được makeup, sau đó model sẽ phải output ra kết quả dự đoán gương mặt đó khi chưa makeup (Hình 1). Kiến trúc model được sử dụng trong project này sẽ xoay quanh kiến trúc Unet.

**Multi-task Learning:** Gần đây Multi-task learning cũng là một hướng nghiên cứu được chú ý với ý tưởng là nếu chỉ học một task cụ thể thì đôi khi model có thể sẽ chỉ tập trung vào các thông tin task chính mà bỏ qua những thông tin hữu ích khác có thể giúp model học tốt hơn. Do đó nếu model học thêm các task khác có thể bổ sung tri thức (các thông tin này) liên quan giữa các task giúp cải thiện performance của model. Trong project này chúng ta cũng sẽ thử nghiệm multi-task learning bằng cách thực hiện thêm một task thứ hai là Super Resolution task, model nhận input là một ảnh và output ra ảnh input với **size tăng 4 lần** (Hình 2). Như vậy ta sẽ có một multi-task model gồm hai task De-makeup và Super Resolution (Hình 3)

**Note:** Vì sử dụng GAN để tạo ra de-makeup data (sẽ có ảnh makeup tốt và makeup tệ) do đó project sẽ được xem như loại bỏ makeup dù makeup là tốt hay tệ để khôi phục lại gương mặt trước khi makeup. Điều này tương tự cho Super Resolution data

#### Tài Liệu Tham Khảo:

1. **Style Transfer:** [link1](#), [link2](#)
2. **Makeup Transfer:** [Lipstick ain't enough: Beyond Color Matching for In-the-Wild Makeup Transfer](#)
3. **Super Resolution:** [Enhanced Deep Residual Networks for Single Image Super-Resolution](#) và [Local Texture Estimator for Implicit Representation Function](#)
4. **Unet:** [U-Net: Convolutional Networks for Biomedical Image Segmentation](#)

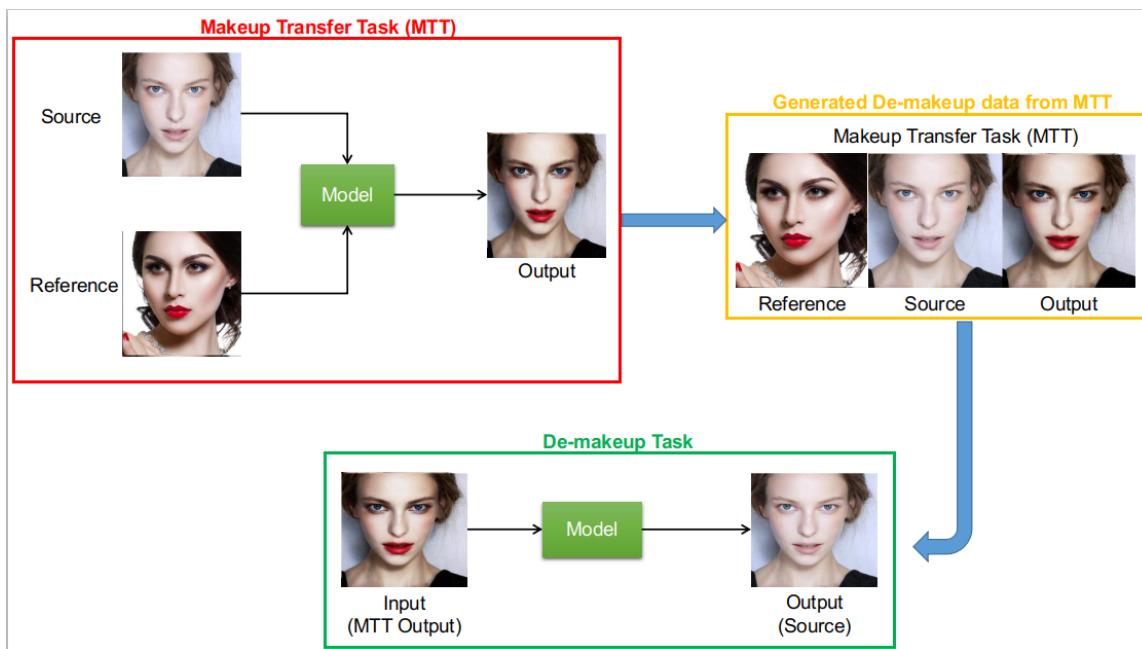
#### Yêu cầu project:

- Thực hiện demak-up task bằng cách xây dựng một model dựa trên kiến trúc Unet (thành phần các layer các bạn tùy ý sử dụng)
- Thực hiện demak-up task bằng cách xây dựng một model dựa trên kiến trúc Unet với encoder là ResNet34 và **không** dùng skip connection giữa encoder và decoder
- Thực hiện demak-up task bằng cách xây dựng một model dựa trên kiến trúc Unet với encoder là ResNet34 và **có** dùng skip connection giữa encoder và decoder
- Thực hiện mutitask-learning với hai task là demak-up và super resolution task bằng cách xây dựng một model dựa trên kiến trúc Unet với encoder là ResNet34 và **có** dùng skip connection giữa encoder và decoder. Đồng thời mở rộng output của decoder để thực hiện được cả hai task đồng thời

## Phần II: Các Bước Thực Hiện

### 1. Giới thiệu de-makeup dataset:

- (a) **De-makeup dataset:** Là một tập data chứa hai loại ảnh: trước (non-makeup) và sau (makeup) khi makeup được chia thành 3 tập train/val/test và có số lượng ảnh tương ứng là 5000/500/500. Mỗi ảnh sẽ có kích thước ảnh là 224x224x3. Tập data này được tạo ra dựa trên GAN model cho task Makeup transfer. Quy trình tạo ra tập data này gồm hai bước chính: **Makeup Transfer:** dùng ảnh chưa makeup (Source) thực hiện makeup transfer task (dựa trên một ảnh là makeup style (Reference), sẽ transfer style makeup này vào ảnh Source) để thu được Output là ảnh đã makeup. **Generate De-makeup Data:** sau đó ảnh Output được dùng làm input (makeup) và Source dùng làm Target (output mong muốn) (non-makeup) cho tập De-makeup dataset (Hình 4).

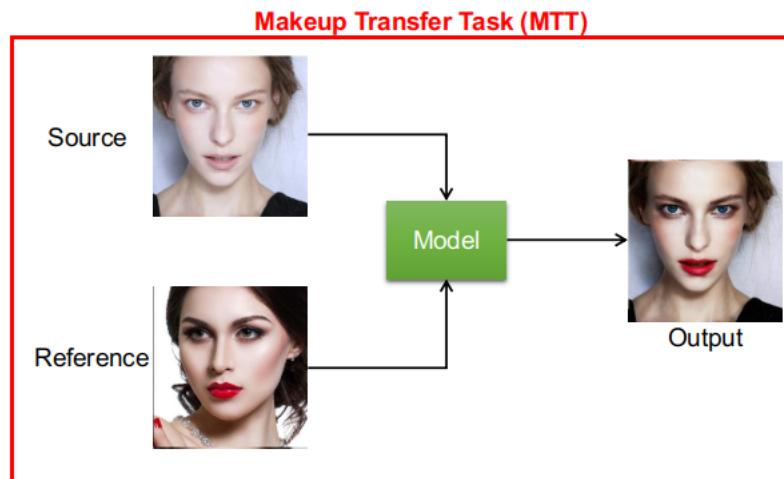


Hình 4: Luồng hoạt động generate de-makeup data từ makeup transfer task để thực hiện de-makeup task

- (b) **Makeup Transfer:** cần 2 ảnh 1 ảnh gọi là source (ảnh chưa makeup), ảnh còn lại là reference (ảnh style makeup). Nhiệm vụ là sẽ tạo ra ảnh output từ source được makeup theo phong cách của ảnh reference (Hình 5 và 6)

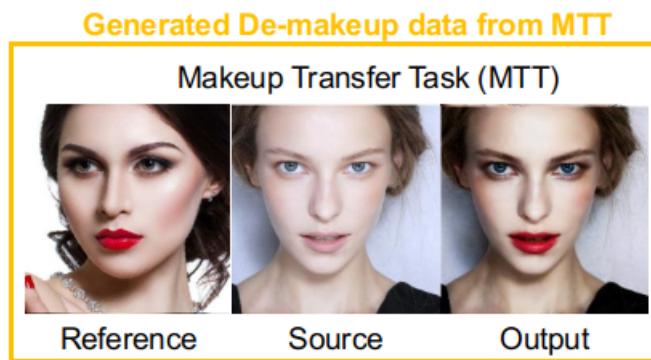


Hình 5: Makeup transfer example: hàng trên là ảnh makeup style, ảnh bên trái ngoài cùng hàng dưới là ảnh source, ảnh còn lại là kết quả sau khi thực hiện makeup transfer lên ảnh source



Hình 6: Makeup transfer Task

- (c) **Generate De-makeup Data:** Sử dụng Makeup Transfer model để biến đổi ảnh chưa makeup thành makupe. Sau đó dùng 2 ảnh này để làm data de-makeup (input: ảnh makeup và output là ảnh chưa makeup) (Hình 4 và 7).

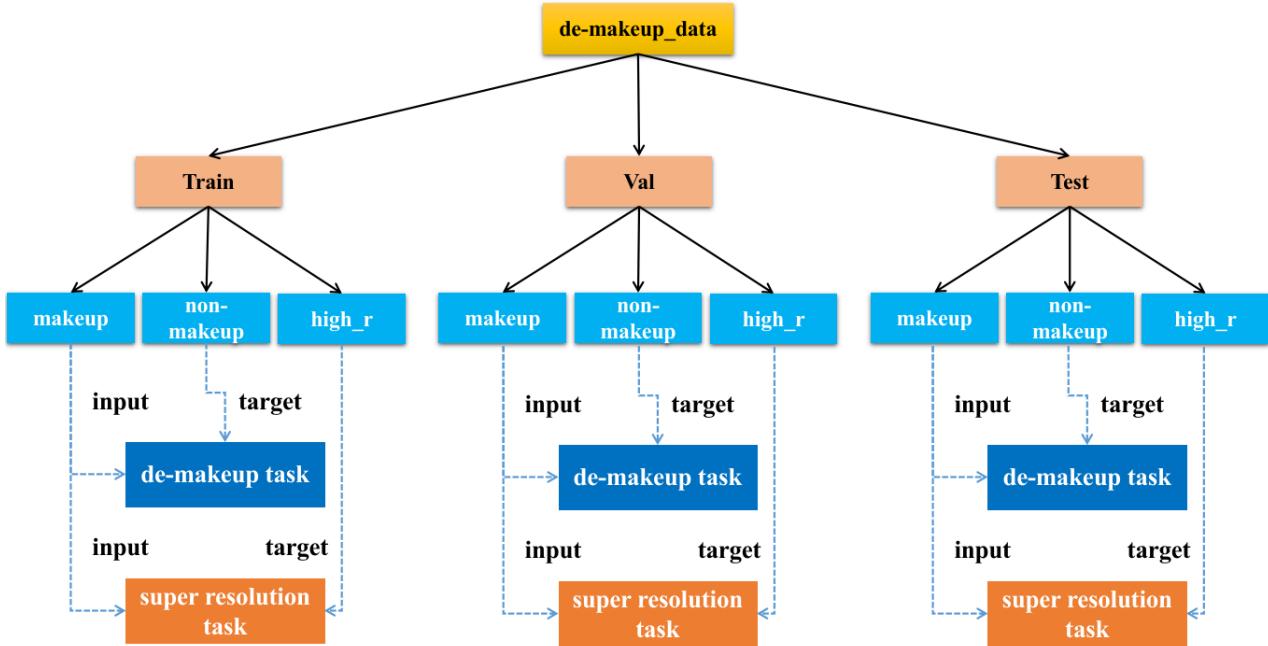


Hình 7: Makeup transfer generate de-makeup data

## 2. Giới thiệu super resolution dataset:

- (a) **Super Resolution dataset:** Là một tập data chứa hai loại ảnh: Low Resolution (LR) và High Resolution (HR) ( $x4$  lần ảnh LR) được chia thành 3 tập train/val/test và có số lượng ảnh tương ứng là 5000/500/500. Mỗi ảnh sẽ có kích thước ảnh là 224x224x3 cho LR và 896x896x3. Tập data này được tạo ra dựa trên EDSR model cho task Super Resolution. Quy trình tạo ra tập data này gồm hai bước chính: **Super Resolution:** dùng ảnh đã makeup (makeup từ de-makeup dataset) thực hiện super resolution để thu được Output là ảnh đã makeup với kích thước tăng 4 lần. **Generate Super Resolution Data:** sau đó ảnh Output được dùng làm Target (output mong muốn) cho task thứ hai của multi-task learning model (Hình 2).
- (b) **Note:** Do đó ảnh input của de-makeup task chung chính là ảnh LR và ảnh HR là ảnh target cho multi-task model được tạo ra từ EDSR-LTE super resolution mode
- (c) **Kiến trúc data của project:** Chúng ta sẽ nhận được data cho cả hai task: de-makeup và

super resolution (Hình 8). Tùy thuộc vào yêu cầu từng task mà ta sẽ load data the input và target phù hợp



Hình 8: Cấu trúc data của project

### 3. Setup các thư viện cần thiết

- pip install -q -U "tensorflow-text==2.8.\*"
- pip install -q tf-models-official==2.7.0
- pip install tensorflow\_addons

## 1 Setup

```
In [ ]: 1 !pip install -q -U "tensorflow-text==2.8.*"
```

```
In [ ]: 1 !pip install -q tf-models-official==2.7.0
```

```
In [ ]: 1 !pip install tensorflow-addons
```

### 4. Configuration và Load data:

**Config Parameters):** (Hình 9) Ở đây ta sẽ cấu hình train 300 epochs (EPOCH), và mỗi step sẽ lấy 64 samples (BATCH\_SIZE). Các bạn lưu ý IMG\_PATH là đường dẫn đến folder chứa train, val và test data. Tùy thuộc vào cấu hình máy mà các bạn lựa chọn tham số phù hợp

**Load Image cho De-makeup task):** (Hình 9) là hàm load một sample nhận đầu vào (image\_file) là một list gồm 2 elmeent chứa đường dẫn của ảnh makeup và non-makeup. Hàm này sẽ dựa vào đường dẫn để load và trả về 2 ảnh

## 4 Load Data

### 3 Config Parameters

```

1 BATCH_SIZE = 64
2 IMG_HEGIHT = 224
3 IMG_WIDHT = 224
4
5 IMG_CHANNEL = 3
6 BUFFER_SIZE = BATCH_SIZE*10
7
8 IMG_PATH = "./demake_up_data"
9 np.random.seed(25)
10 EPOCHS = 300

```

```

1 def load(image_file):
2     makeup_img_file, non_img_file = tf.split(image_file, 2)
3
4     makeup_img = tf.io.read_file(makeup_img_file[0])
5     makeup_img = tf.image.decode_jpeg(makeup_img, channels=IMG_CHANNEL)
6
7     non_img = tf.io.read_file(non_img_file[0])
8     non_img = tf.image.decode_jpeg(non_img, channels=IMG_CHANNEL)
9
10    # Convert both images to float32 tensors
11    makeup_img = tf.cast(makeup_img, tf.float32)
12    non_img = tf.cast(non_img, tf.float32)
13
14    return makeup_img, non_img

```

Hình 9: Configuration và hàm load ảnh input và target

**Load Image cho Multi-task learning:** (Hình 10) là hàm load một sample nhận đầu vào (image\_file) là một list gồm 3 elmeent chứa đường dẫn của ảnh makeup, non-makeup, và high resolution (x4 ảnh makeup). Hàm này sẽ dựa vào đường dẫn để load và trả về 3 ảnh

```

1 def load(image_file):
2     makeup_img_file, non_img_file, hr_img_file = tf.split(image_file, 3)
3
4     makeup_img = tf.io.read_file(makeup_img_file[0])
5     makeup_img = tf.image.decode_jpeg(makeup_img, channels=IMG_CHANNEL)
6
7     non_img = tf.io.read_file(non_img_file[0])
8     non_img = tf.image.decode_jpeg(non_img, channels=IMG_CHANNEL)
9
10    hr_img = tf.io.read_file(hr_img_file[0])
11    hr_img = tf.image.decode_jpeg(hr_img, channels=IMG_CHANNEL)
12
13    # Convert both images to float32 tensors
14    makeup_img = tf.cast(makeup_img, tf.float32)
15    non_img = tf.cast(non_img, tf.float32)
16    hr_img = tf.cast(hr_img, tf.float32)
17
18    return makeup_img, non_img, hr_img

```

Hình 10: Hàm load ảnh input và hai target cho multi-task learning

**Processing Image:** (Hình 11) là processing cho de-makup task (sẽ tương tự cho multi-task learning bằng cách thực hiện cho ảnh high resolution). Đối với load ảnh cho mục đích train thì sẽ có 50% tỉ lệ flip ảnh theo chiều ngang, tiếp theo ảnh sẽ scale vào range [0, 1]. Đối với load ảnh cho validate và test thì chỉ scale và không flip

```

1 @tf.function()
2 def random_flip(makeup_img, non_img):
3     if tf.random.uniform(() > 0.5:
4         # Random mirroring
5         makeup_img = tf.image.flip_left_right(makeup_img)
6         non_img = tf.image.flip_left_right(non_img)
7
8     return makeup_img, non_img
9
10
11 def processing_image(makeup_img, non_img):
12     makeup_img = (makeup_img / 255.0)
13     non_img = (non_img / 255.0)
14
15     return makeup_img, non_img
16
17
18 def load_image_train(image_file):
19     makeup_img, non_img = load(image_file)
20     makeup_img, non_img = random_flip(makeup_img, non_img)
21     makeup_img, non_img = processing_image(makeup_img, non_img)
22
23     return makeup_img, non_img
24
25
26 def load_image_val(image_file):
27     makeup_img, non_img = load(image_file)
28     makeup_img, non_img = processing_image(makeup_img, non_img)
29
30     return makeup_img, non_img

```

Hình 11: Các hàm xử lý ảnh

```

1 def prep_data(path):
2     makeup_img_list = [os.path.join(path, f) for f in os.listdir(path)]
3     data_list = [[i, i.replace('makeup', 'non-makeup')] for i in makeup_img_list]
4     return data_list
5
6 train_data_list = prep_data(str(IMG_PATH + '/train/makeup/'))
7 val_data_list = prep_data(str(IMG_PATH + '/val/makeup/'))
8 test_data_list = prep_data(str(IMG_PATH + '/test/makeup/'))
9
10 np.random.shuffle(train_data_list)
11 np.random.shuffle(val_data_list)
12 np.random.shuffle(test_data_list)

```

```

1 train_dataset = tf.data.Dataset.from_tensor_slices(train_data_list)
2 train_dataset = train_dataset.map(load_image_train,
3                                     num_parallel_calls=tf.data.AUTOTUNE)
4 train_dataset = train_dataset.shuffle(BUFFER_SIZE)
5 train_dataset = train_dataset.batch(BATCH_SIZE)
6

```

```

1 val_dataset = tf.data.Dataset.from_tensor_slices(val_data_list)
2 val_dataset = val_dataset.map(load_image_val)
3 val_dataset = val_dataset.batch(BATCH_SIZE)

```

```

1 test_dataset = tf.data.Dataset.from_tensor_slices(test_data_list)
2 test_dataset = test_dataset.map(load_image_val)
3 test_dataset = test_dataset.batch(BATCH_SIZE)

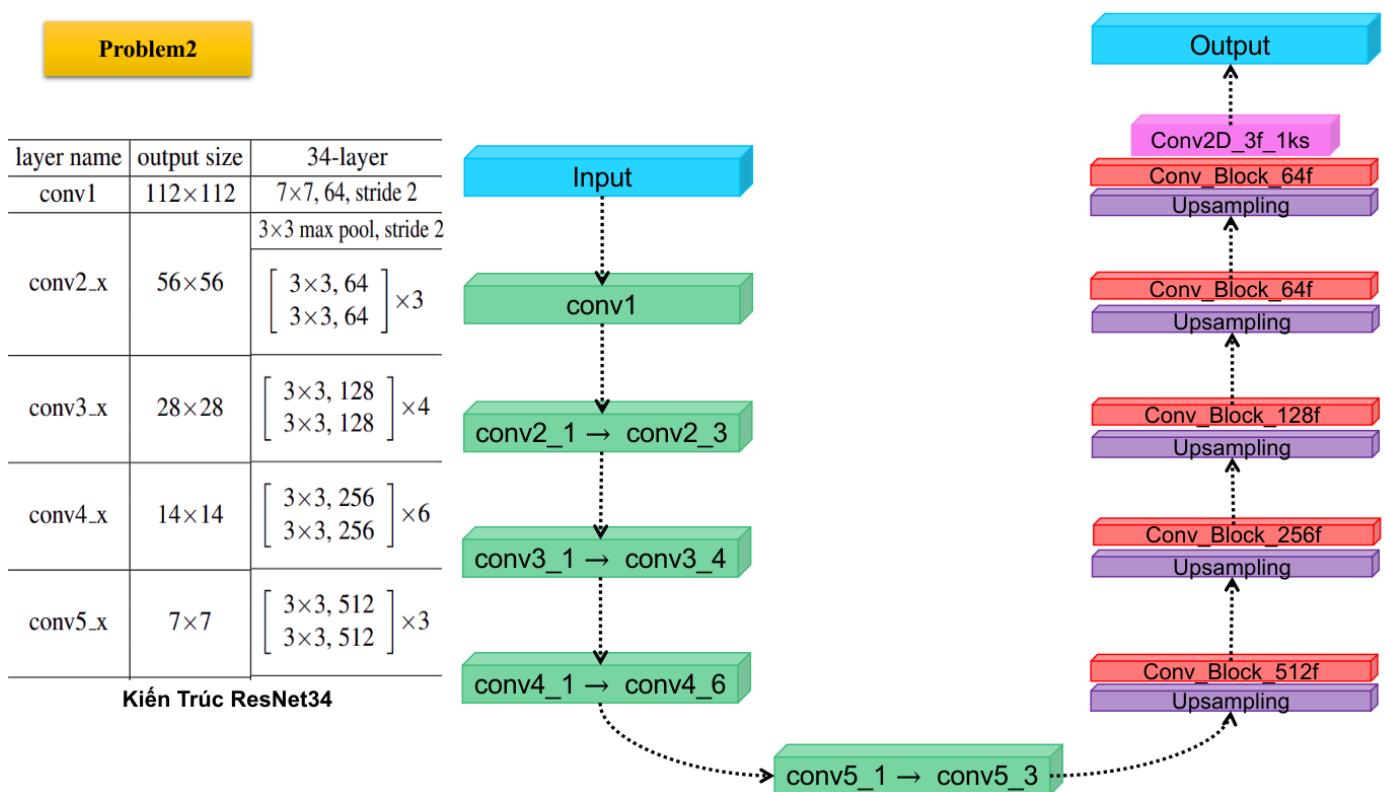
```

Hình 12: Các hàm chuẩn bị list ảnh và load train, val và test data

**Load Train/Val/Test data:** (Hình 12) Hàm pre\_data nhận đầu vào là path (đường dẫn nơi chứa ảnh (ví dụ train)). Vì trong train/val/test folder sẽ có 3 subfolder là makeup, non-makeup và high\_r (Hình 8). Trong mỗi sample (là một element của data\_list) sẽ là một list chứa 2 element (đường dẫn cho ảnh makeup và non-makeup) cho de-makeup task, còn với task multi-task learning ta làm tương tự nhưng cần 3 element vì cần có ảnh high resolution. Sau đó ta build TensorFlow input pipeline (dùng tf.data) cho tập train, val và test (lưu ý tập train cần shuffle)

## 5. Build Model

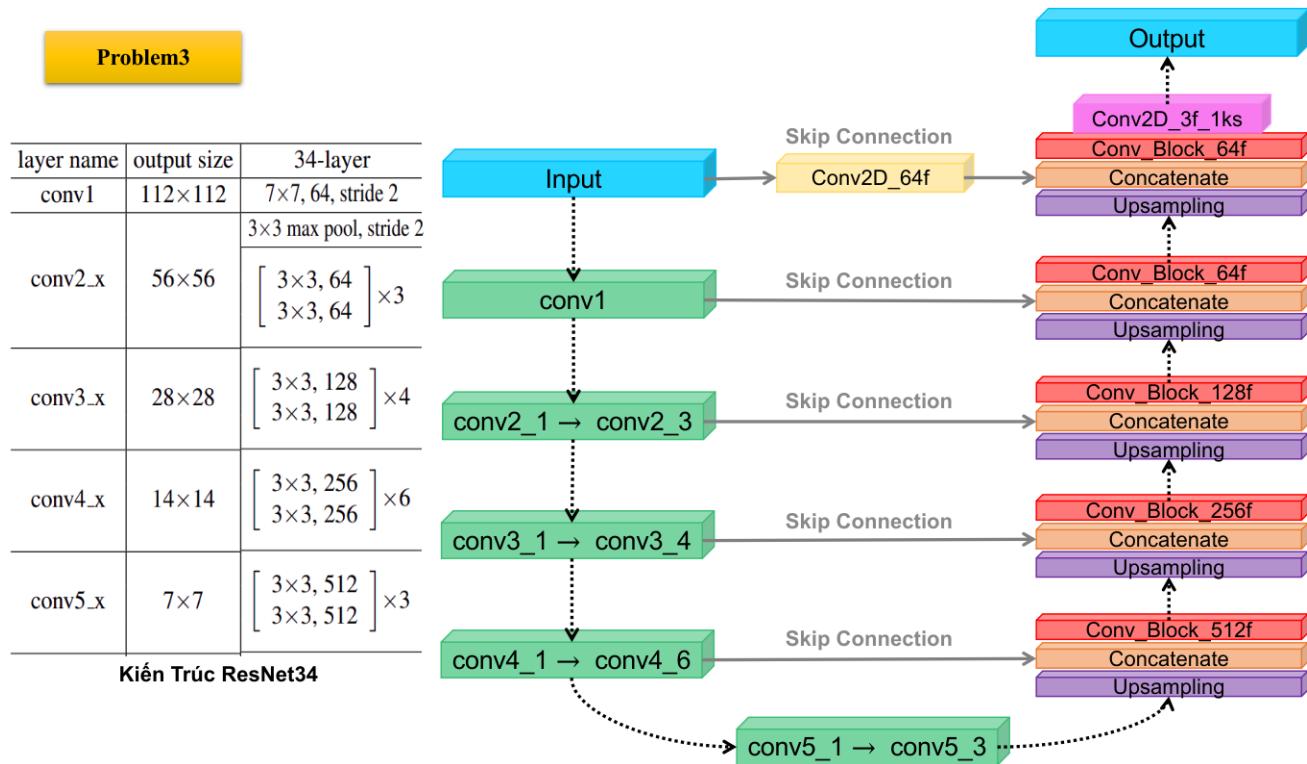
**Unet dùng ResNet34 và không skip-connection:** (Hình 13) Chúng ta sẽ sử dụng một phần ResNet34 (loại bỏ phần MLP Classification) làm encoder để trích xuất đặc trưng của ảnh. Output của encoder là layer cuối cùng của conv5\_3 (feature map 7x7 - kích thước ảnh giảm 32 lần) sẽ đi qua decoder gồm các upsampling block (Conv2DTranspose, Conv2D) giúp cho feature map x2 kích thước nhưng channel cũng sẽ giảm đi một nữa, và tại block cuối cùng một Conv2D dùng để điều chỉnh lại số lượng channel=3 và feature map này có kích thước 224x224 cũng chính là ảnh output (loại bỏ makeup) mong muốn. [Chi tiết thực hiện code model](#)



Hình 13: Kiến trúc Unet dùng ResNet34 và không skip connection

**Unet dùng ResNet34 và có skip-connection:** (Hình 14) Chúng ta sẽ thực hiện tương tự như **Unet dùng ResNet34 và không skip-connection** nhưng lần này có kèm theo skip connection. Các output tại mỗi block cuối cùng của ResNet34 (conv1, conv2\_3, conv3\_4, conv4\_6)(trước khi feature map size bị giảm đi một nữa) sẽ được trích xuất ra và thực hiện skip connection với decoder block bằng toán tử concatenate để giúp model học được tốt hơn thông qua việc cung cấp nhiều thông tin hơn của các tầng feature map thay vì chỉ thông tin của conv5\_3. Chú ý skip connection cũng được thực hiện tại feature map có size 224x224, và theo thực nghiệm thay vì skip connection với ảnh input thì dùng feature sau khi qua conv2D sẽ cho kết quả tốt hơn. [Chi tiết](#)

[thực hiện code model](#)



Hình 14: Kiến trúc Unet dùng ResNet34 và có skip connection

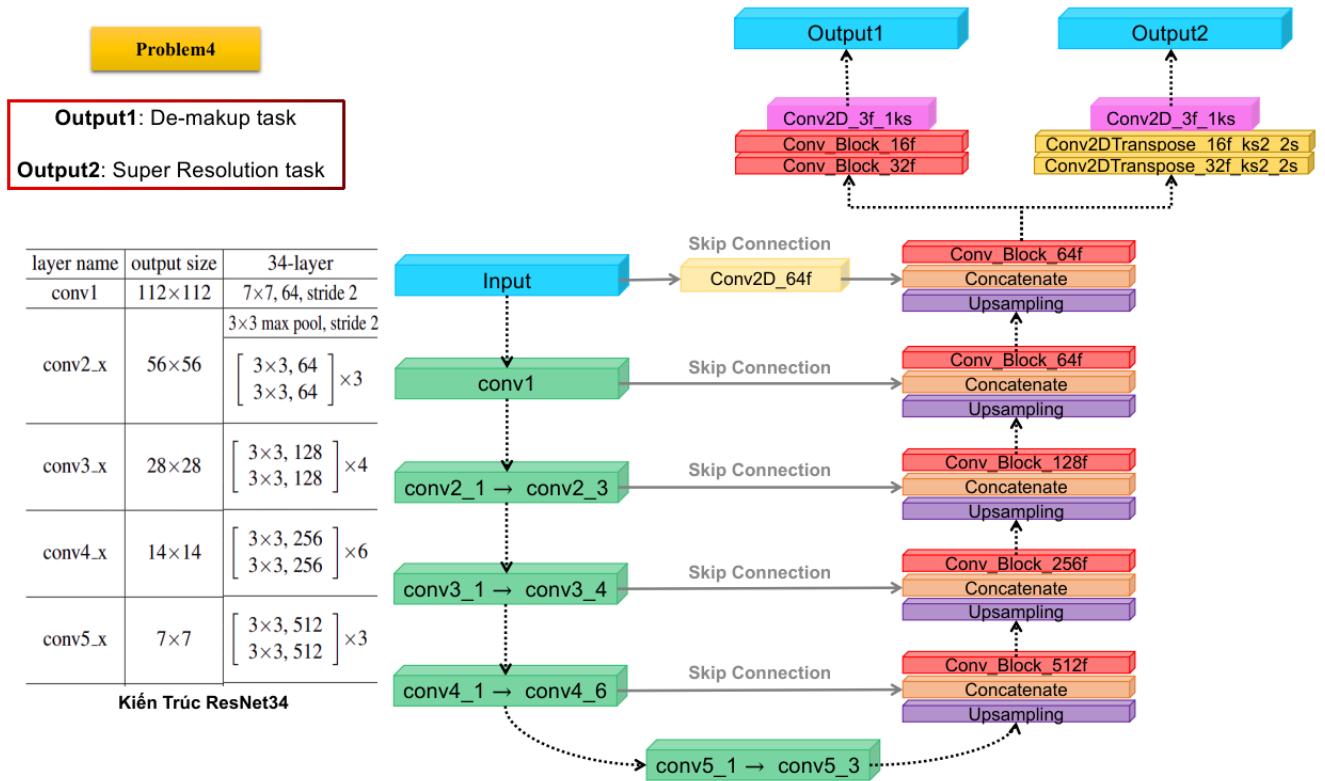
**Unet cho Multi-task learning:** (Hình 15) Chúng ta sẽ thực hiện tương tự như **Unet dùng ResNet34 và có skip-connection** nhưng output sẽ được mở rộng ra để thực hiện task de-makeup và super resolution. Tại output của upsampling block cuối cùng ta sẽ ra thành 2 nhánh: nhánh 1 (output1) sẽ phải trải qua 2 Conv\_Block để refine lại các feature trước khi qua Conv2D để điều chỉnh channel=3 và tạo thành ảnh đã loại bỏ makeup, nhánh 2 (output2) sẽ hoạt động tương tự nhưng thay 2 Conv\_Block bằng 2 Conv2DTranspose để x4 feature map và qua Conv2D để chỉnh channel=3 tạo ra ảnh high resolution có kích thước x4 lần ảnh input. [Chi tiết thực hiện code model](#)

## 6. Configuration và Train

**Note:** Đây là ví dụ cho de-makeup task, đối với multi-task learning sẽ thực hiện tương tự và chúng ta chỉ tính thêm phần high resolution image (các bạn có thể tham khảo thêm trong phần hint).

**Optimizer:** Vì project yêu cầu sử dụng AdamW optimizer nên chúng ta sẽ cấu hình theo Hình 16. Chúng ta cần tính tổng train step (num\_train\_step) bằng số train step trong 1 epoch (step\_per\_epoch) nhân với tổng số epoch (EPOCHS). Sau đó ta tính warup step = 10% tổng số train step. Cuối cùng ta truyền các tham số này cùng với initial learning để khởi tạo AdamW. Có nghĩa là 10% step đầu tiên sẽ tăng dần learning rate cho đến intial learning rồi phần còn lại của learning rate sẽ thực hiện theo AdamW

**Evaluation và Plot ảnh:** (Hình 17) hàm evaluate dùng để đánh giá kết quả trên tập val hoặc test, nhận 3 tham số model: là model dùng predict, epoch: epoch hiện tại, và dataset là tập val hoặc test. Ngoài ra hàm generate\_images có chức năng vẽ 3 hình makeup, non-makeup và kết quả predict của model trong lúc train, giúp show ảnh train hoặc validation. Hàm này nhận makeup\_img và non\_img là batch ảnh và chỉ show sample đầu tiên trong batch



Hình 15: Kiến trúc Unet cho Multi-task learning

## 6.2 Optimizer

```

1 from official.nlp import optimization # to create AdamW optimizer
2 steps_per_epoch = tf.data.experimental.cardinality(train_dataset).numpy()
3 num_train_steps = steps_per_epoch * EPOCHS
4 num_warmup_steps = int(0.1*num_train_steps)
5
6 init_lr = 1e-2
7 generator_optimizer = optimization.create_optimizer(init_lr=init_lr,
8                                         num_train_steps=num_train_steps,
9                                         num_warmup_steps=num_warmup_steps,
10                                        optimizer_type='adamw')

```

Hình 16: Cách sử dụng AdamW optimizer

**Train một step:** (Hình 18) hàm train\_step sẽ thực hiện train 1 step (1 batch ảnh). Đầu tiên hàm dùng model dự đoán kết quả trên một batch ảnh (makeup\_img) (input) và thu được (pred\_non). Tiếp theo ta tính loss theo hàm MSE bằng cách dùng pred\_non và non\_img (target). Cuối cùng ta tính gradient cho các weights trong model và cập nhật weights theo AdamW optimizer

**Train:** (Hình 18) hàm fit sẽ thực hiện train toàn bộ EPOCH (300 lần) đồng thời mỗi epoch khi evaluate trên tập val thu được kết quả tốt hơn (dựa trên PSNR metric) sẽ vẽ ảnh 1 sample để tiện theo dõi quá trình train

## 6.1 Show Generated Images and Evaluation Function

```

1 def evaluate(model, epoch, dataset):
2     psnr_non_mean = 0.0
3     psnr_sr_mean = 0.0
4     count = 0
5     for makeup_img, non_img in dataset:
6         pred_non = model([makeup_img], training=False)
7         psnr_non = tf.image.psnr(pred_non, non_img, max_val=1.0)
8         _psnr_non_mean = tf.math.reduce_mean(psnr_non)
9         psnr_non_mean += _psnr_non_mean
10        count = count + 1
11    psnr_non_mean = psnr_non_mean/count
12    print('----- psnr_non: ', psnr_non_mean.numpy(), '----- epoch: ', epoch, ' count: ', count)
13
14    return psnr_non_mean
15
16
17 def generate_images(model, makeup_img, non_img):
18     pred_non = model([makeup_img], training=False)
19     plt.figure(figsize=(15,20))
20     display_list = [makeup_img[0], non_img[0], pred_non[0]]
21     title = ['Input', 'Non-makeup', 'Predicted']
22
23     for i in range(3):
24         plt.subplot(1, 3, i+1)
25         plt.title(title[i])
26         plt.imshow(display_list[i])
27         plt.axis('off')
28     plt.show()

```

Hình 17: Dùng PSNR để đánh giá kết quả predict và hàm show ảnh trong lúc train

## 7 Training

```

1 @tf.function
2 def train_step(model, makeup_img, non_img):
3     with tf.GradientTape() as tape:
4         # output
5         pred_non = model([makeup_img], training=True)
6         loss = tf.reduce_mean(tf.square(pred_non-non_img))*100
7
8     generator_gradients = tape.gradient(loss, model.trainable_variables)
9     generator_optimizer.apply_gradients(zip(generator_gradients, model.trainable_variables))
10
11    return loss
12
13
14 def fit(model, train_ds, epochs, val_ds):
15    best_psnr = 0.0
16    step_counter = 0
17    for epoch in range(epochs):
18        # Train
19        total_loss = 0.0
20        for makeup_img, non_img in train_ds:
21            loss = train_step(model, makeup_img, non_img)
22            total_loss = total_loss + loss
23            step_counter += 1
24        total_loss = total_loss/step_counter
25        print('epoch: {} loss: {}'.format(epoch, total_loss))
26
27    pnsr = evaluate(model, epoch, val_ds)
28    if best_psnr < pnsr:
29        best_psnr = pnsr
30
31    for makeup_img, non_img in val_ds.take(1):
32        generate_images(model, makeup_img, non_img)
33
34

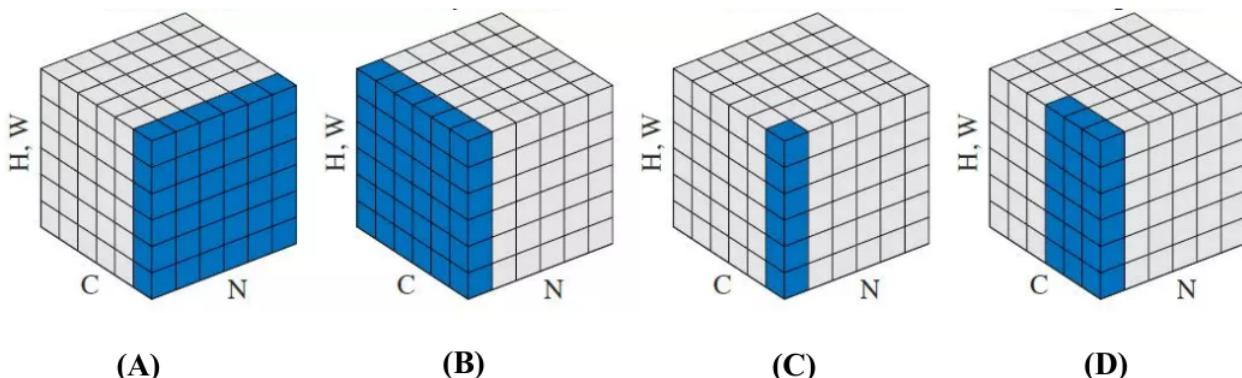
```

Hình 18: Các hàm thực hiện train

### Phần III: Câu Hỏi

## A. Phần cơ bản

- Thành phần chính để cấu thành một network có kiến trúc theo Unet là
    - (A). Encoder
    - (B). Decoder
    - (C). Skip-connection
    - (D). Tất cả đều đúng
  - Các bạn đọc paper **An Overview of Multi-Task Learning in Deep Neural Networks** trang 2 và 3, thì cơ bản có thể chia multi-task learning model thành mấy loại và là loại gì:
    - (A). 1 loại: hard parameter sharing
    - (B). 1 loại: soft parameter sharing
    - (C). 2 loại: soft và hard parameter sharing
    - (D). 3 loại: soft, hard and mix parameter sharing
  - Trong một bài toán Input là image được xử lý các giá trị pixel trong range [0,1] và output cũng là một ảnh thì layer cuối cùng nên sử dụng activation function nào
    - (A). ReLu
    - (B). Sigmoid
    - (C). Softmax
    - (D). Tất cả đều đúng
  - Trong hình 19 thì đâu là hình biểu diễn cách hoạt động của Intance Normalization



Hình 19: Các kỹ thuật Normalization

- (A). Hình (A) (B). Hình (B)  
(C). Hình (C) (D). Hình (D)

5. Mục đích chính của GELU activation function được tạo ra là

(A). Hội tụ tốt hơn và nhanh hơn so với Sigmoid (B). Có tính chất regularization ví dụ như Dropout  
(C). Tất cả đều sai (D). Tất cả đều đúng

6. Các phương pháp có thể sử dụng trong việc tăng kích thước feature map (upsampling method) ở thành phần decoder trong Unet là

(A). Transposed convolution (B). Nearest-neighbor Interpolation  
(C). Tất cả đều sai (D). Tất cả đều đúng

7. Trong Unet thì feature map khi đi qua từng layer trong Encoder sẽ:
- Size của feature map sẽ bị giảm và số lượng channel tăng lên
  - Size của feature map sẽ tăng và số lượng channel bị giảm
  - Size của feature map sẽ bị giảm và số lượng channel giảm
  - Size của feature map sẽ tăng và số lượng channel tăng lên
8. Trong Unet thì feature map khi đi qua từng layer trong Decoder sẽ:
- Size của feature map sẽ bị giảm và số lượng channel tăng lên
  - Size của feature map sẽ tăng và số lượng channel bị giảm
  - Size của feature map sẽ bị giảm và số lượng channel giảm
  - Size của feature map sẽ tăng và số lượng channel tăng lên
9. Makeup transfer task được miêu tả ở project này dùng để thực hiện generate data cho de-makeup có đặc điểm là
- Input là ảnh chưa makeup, đi vào model sẽ output ra ảnh đã makeup theo style phù hợp nhất với gương mặt input
  - Input sẽ yêu cầu 2 ảnh: source (ảnh chưa makeup) và reference (ảnh style makeup), sau đó output ra ảnh source đã makeup theo style của reference
  - Input sẽ yêu cầu 2 thông tin: source (ảnh chưa makeup) và style (text miêu tả style makeup), sau đó output ảnh source đã makeup theo miêu tả trong text
  - Input sẽ yêu cầu 2 đoạn text: source (text miêu tả gương mặt), và style (miêu tả style makeup), sau đó output ảnh có gương mặt như miêu tả của source và đã makeup theo miêu tả trong style
10. Trong một bài toán Input là image và output cũng là một ảnh, nếu layer cuối cùng sử dụng tanh activation function thì nên processing Input:
- Để có các giá trị pixel trong range [0,1]
  - Để có các giá trị pixel trong range [-1,1]
  - Để có các giá trị pixel trong range [0,255]
  - Tất cả đều đúng

**B. Phần nâng cao. Các bạn không cần trả lời câu hỏi này. Các câu hỏi này sẽ được thảo luận trong buổi TA** NOTE: Các bạn phải sử dụng GELU activation function, Instance Normalization, và AdamW optimizer

1. **Problem 1:** Các bạn sẽ nhận được De-makeup dataset bao gồm 5000/500/500 samples tương ứng với Train/Val/Test. Nhiệm vụ của các bạn là phải load được tập data này, xây dựng kiến trúc Unet bất kỳ dựa vào kiến thức đã được học để tạo ra model để thực hiện de-makeup task với yêu cầu:
- **Input:** ảnh 224x224x3 (ảnh makeup)
  - **Output:** ảnh 224x224x3 (ảnh loại bỏ makeup)
  - Trong tập De-makeup sẽ có hai folder, makeup được dùng làm input và non-makeup được dùng làm output. [Chi tiết thực hiện code model](#)
2. **Problem 2:** Tương tự như **Problem 1** thực hiện de-makeup task nhưng phải xây dựng Unet gồm có encoder sử dụng ResNet34 và tự xây dựng phần decoder tương ứng để model thực hiện được de-makeup task với yêu cầu:

- **Input:** ảnh 224x224x3 (ảnh makeup)
- **Output:** ảnh 224x224x3 (ảnh loại bỏ makeup)
- **Encoder** dùng ResNet34, **Không dùng skip connection giữa encoder và decoder.** [Chi tiết thực hiện code model](#)

**3. Problem 3:** Tương tự như **Problem 2** thực hiện de-makeup task với Unet gồm có encoder sử dụng ResNet34 và tự xây dựng phần decoder tương ứng để model thực hiện được de-makeup task với yêu cầu:

- **Input:** ảnh 224x224x3 (ảnh makeup)
- **Output:** ảnh 224x224x3 (ảnh loại bỏ makeup)
- **Encoder** dùng ResNet34, **Có dùng skip connection giữa encoder và decoder.** [Chi tiết thực hiện code model](#)

**4. Problem 4:** Thủ nghiệm với Multi-task learning cho de-makeup task và super resolution task. Tương tự như **Problem 3** thực hiện de-makeup task với Unet gồm có encoder sử dụng ResNet34 và tự xây dựng phần decoder tương ứng để model thực hiện được de-makeup task. Bên cạnh đó phải mở rộng output để thực hiện được Super Resolution task để tăng size ảnh input lên 4 lần:

- **Dataset:** Bên cạnh tập De-makeup dataset sẽ có một folder được thêm vào là hr\_image là ảnh input nhưng kích thước gấp 4 lần và được dùng cho output của Super Resolution task
- **Input:** ảnh 224x224x3 (ảnh makeup)
- **Output:** De-makeup task - ảnh 224x224x3 (ảnh loại bỏ makeup). Super Resolution task ảnh 896x896x3 (ảnh makeup x4)
- **Encoder** dùng ResNet34, **Có dùng skip connection giữa encoder và decoder,** thực hiện multi-task learning gồm de-makeup và super resolution task. [Chi tiết thực hiện code model](#)

## Phần IV: Hướng Dẫn Sử Dụng Hint

**Chi tiết thực hiện code model:** Ở đây sẽ hướng dẫn sử dụng hint cho problem Unet cho Multi-task learning các problem khác đơn giản hơn và được thực hiện tương tự bằng cách loại bỏ một số thành phần không cần thiết. Vì project yêu cầu xây dựng Unet dựa trên ResNet34, nên khi xây dựng model sẽ thực hiện 2 bước chính: xây dựng ResNet34 dùng Gelu và InstanceNorm. Tiếp theo xây dựng Unet dựa vào ResNet34.

**ResNet34:** (Hình 20) Các bạn xây dựng ResNet34 như đã từng làm ở bài tập trong tuần học Advanced CNN (skip connection topic). Đầu tiên các bạn xây dựng identity\_block qua 2 Conv2D layer đi kèm với InstanceNorm và Gelu (thay vì BatchNorm và Relu như paper gốc) và thực hiện skip connection add. Thực hiện tương tự cho project\_block nhưng ở phần skip connection có thêm một Conv2D cũng dùng InstanceNorm và Gelu. Sau khi hoàn thiện 2 block trên thi các bạn xây dựng model trong hàm build\_model theo thiết kế của ResNet34. Bên dưới là cách dùng InstanceNorm và Gelu trong Tensorflow

- tfa.layers.InstanceNormalization()
- layers.Activation('gelu')

**Res34Unet:** Để xây dựng Unet dựa trên ResNet34 với multi-task learning chúng ta cần thực hiện 3 bước chính: xây dựng encoder (ResNet34), xây dựng decoder, và thiết kế output cho multi-task.

```

def identity_block(self, inputs, filters):
    ##### Your Code Here #####
    ...
    Các bạn thực hiện tương tự identity_block như bài tập của topic ResNet
    Lưu ý:
        Thay Relu -> Gelu Ex: layers.Activation('gelu')
        Thay BatchNormalization -> tfa.layers.InstanceNormalization()
    ...
    #####
    return x

def projection_block(self, inputs, filters, strides=2):
    ##### Your Code Here #####
    ...
    Các bạn thực hiện tương tự projection_block như bài tập của topic ResNet
    Lưu ý:
        Thay Relu -> Gelu Ex: layers.Activation('gelu')
        Thay BatchNormalization -> tfa.layers.InstanceNormalization()
    ...
    #####
    return x

def build_model(self, classes, inputs):
    ...
    Lưu ý không khai báo inputs layer ở trong hàm build_model
    chúng ta sẽ nhận vào từ tham số của hàm (nhận được trong lúc build Res34Unet)
    ...
    ##### Your Code Here #####
    # Các bạn thực hiện tương tự projection_block như bài tập của topic ResNet
    # conv1 + max pool
    ...
    inputs -> (zero padding 3x3)
    -> (Conv2D 7x7 stride=2, filter=64)
    -> (batch norm)
    -> (ReLU)
    -> (zero padding 1x1)
    -> (Max pooling 3x3, stride=2)
    ...

```

Hình 20: Hint xây dựng ResNet34

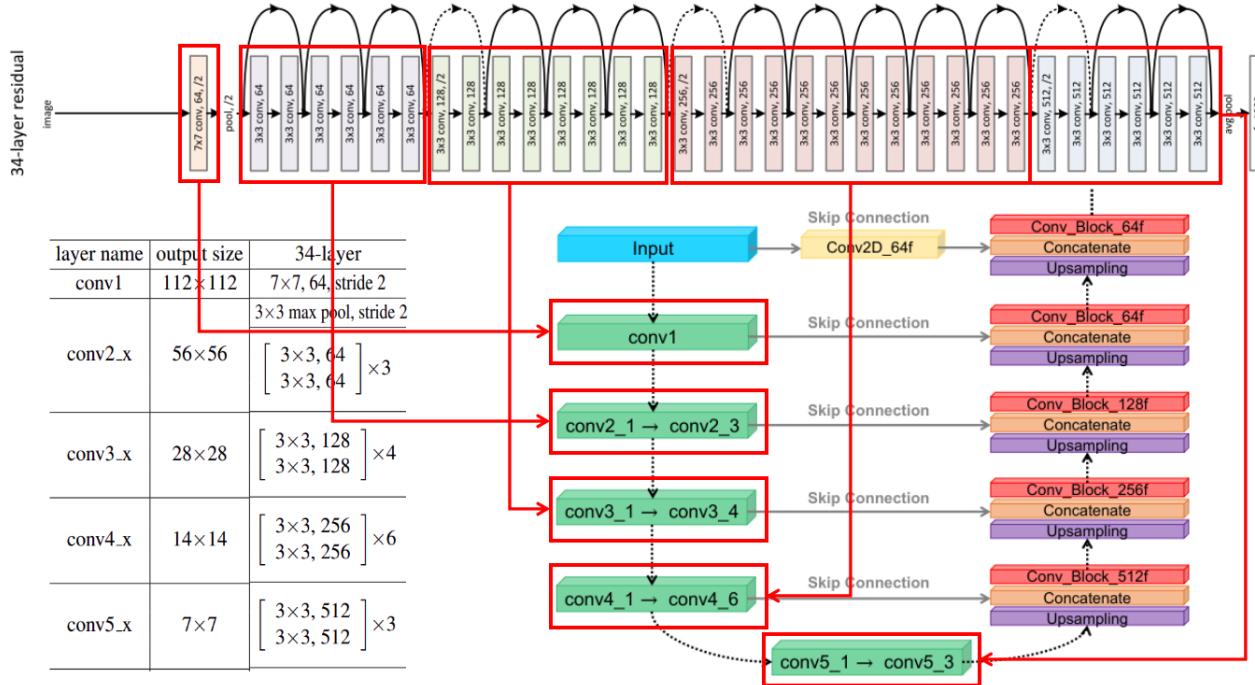
- **Encoder** (ResNet34): (Hình 21 và 22) Chúng ta sẽ sử dụng output của mỗi layer cuối cùng trong từng block của ResNet (conv1, conv2\_3, conv3\_4 và conv4\_6) chính là output có feature map trước khi bị downsample size 1/2 cho skip connection để concatenate với các block trong decoder. Bên cạnh đó output của conv5\_3 được dùng làm bridge để nối encoder với decoder. Tại hình 22 line 31 chúng ta sẽ khởi tạo layer Input với shape nhận được từ biến input\_shape. Line 40 chúng ta sẽ khởi tạo ResNet34 (bằng cách gọi method build\_model ở hình 20) với input layer. Line 50-53, và line 59 ta sẽ dùng method get\_layer để lấy output của từng layer thích hợp cho skip connection và bridge bằng cách truyền vào index của layer đó (được khởi tạo ở line 3 và 4 hình 23) (ví dụ eb1 là output của layer conv2\_3). Đây là ví dụ để lấy output của bridge:

– br = backbone.get\_layer(index=self.bridge\_block\_id).output

- **Decoder**: (Hình 23 và 24). Để xây decoder chúng ta sẽ xây dựng upsample\_concatenation\_block và conv\_block. Trong đó conv\_block (xxxfilter chính là số lượng filter) sẽ gồm một Conv2D có kernel\_size = 3x3 và dùng padding='same', (stride=1 mặc định) để giữ nguyên feature map size, một InstanceNorm và một Gelu activation function. Đối với upsample\_concatenation\_block sẽ là một Conv2DTranspose kernel\_size= 2x2, stride=2, padding='same' dùng làm tăng x2 feauture map size, sau đó concatenate với skip-connection từ encoder và cuối cùng đi qua conv\_block. Hình 24 với mỗi upsample\_concatenation\_block sẽ nhận skip-connection từ encoder tương ứng, ví dụ db1 sẽ concatenate skip-connection từ eb1 là output của layer conv2\_3 trong ResNet34 với output từ đã được upsample từ upsample\_concatenation\_block trước đó.

- **Output**: (Hình 25) Việc thực hiện upsample cũng diễn ra tương tự cho skip-connection (line 87 và 88) là feature từ ảnh input được đi qua Conv2D kernel\_size= 3x3, filters=64 và padding='same' trước khi tạo ra 2 output cho multi-task.

- **Output1** (line 90,91 và 92) dùng cho de-makeup task nên sẽ đi qua 2 conv\_block với số lượng filters lần lượt là 32, và 16, tiếp theo nó sẽ đi qua Conv2D với kernel\_size = 1x1 và filters=3 để điều chỉnh thành ảnh output (loại bỏ makeup) có 3 channel.
- **Output2** (line 94,95 và 96) dùng cho super resolution task nên sẽ đi qua 2 Conv2DTranspose kernel\_size=2x2, stride=2, và số lượng filter lần lượt là 32 và 16 để tăng ảnh lên x4 lần.

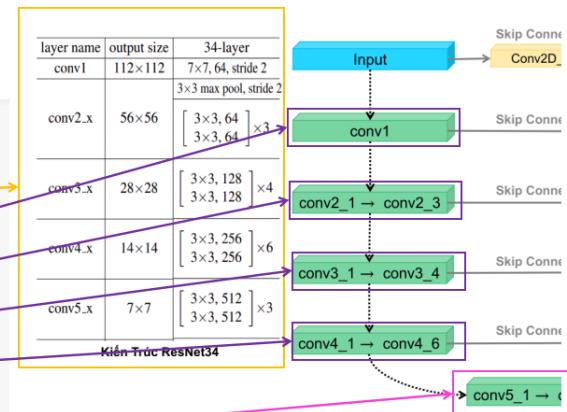


Hình 21: Trích xuất output của từng block trong ResNet34 để thực hiện skip connection cho decoder

```

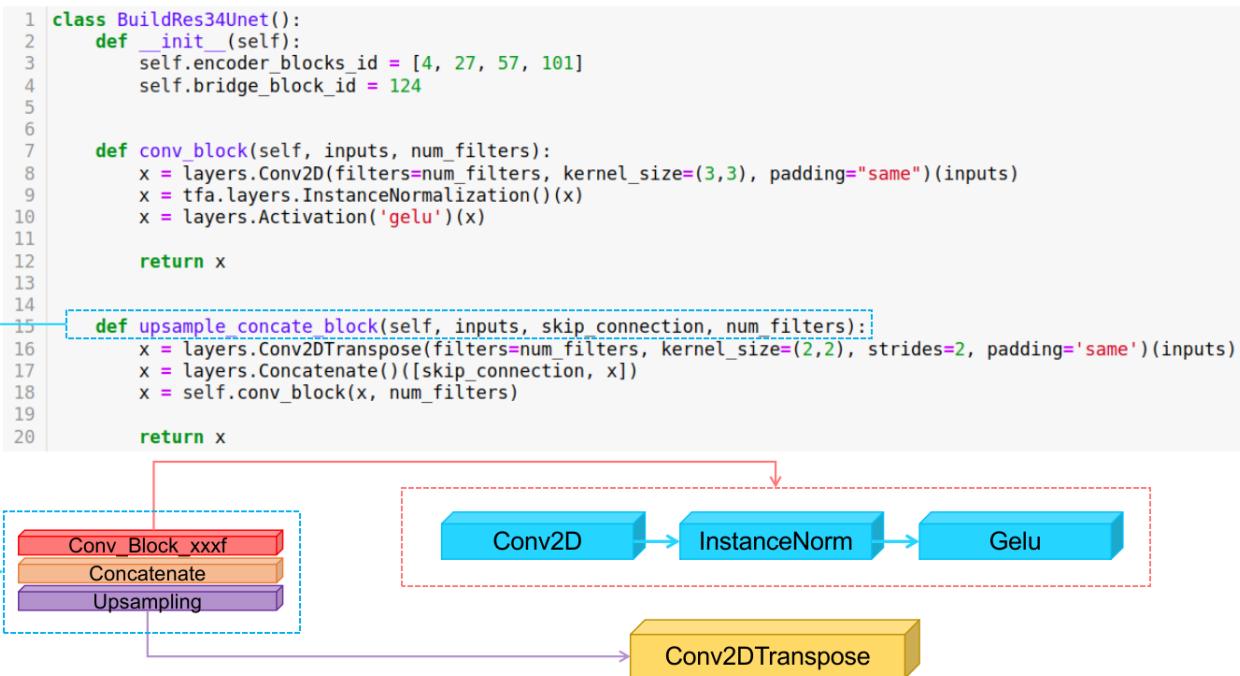
23     def build_model(self, input_shape):
24         ##### Your Code Here #####
25         # Các bạn thực hiện xây dựng Unet dựa trên ResNet34
26
27         # input layer
28         ...
29         input_layer (layers.Input) nhận shape = input_shape
30         ...
31         inputs =
32
33         # encoder
34         ...
35         Tạo instance từ class BuildResNet34()
36         build backbone bằng build_model method của build_resnet34 instance
37         nhän inputs=inputs, số classes nên để 1000 cho giống mặc định
38         ...
39         build_resnet34 = BuildResNet34()
40         backbone =
41
42         ...
43         Trích xuất output của từng layer để thực dụng làm thành phần skip-connection
44         cho decoder (Tham khảo các lăy output layer br của hint dành cho Problem 2)
45         eb0 = dùng backbone.get_layer với index = element [0] của self.encoder_blocks_id
46         eb1 = dùng backbone.get_layer với index = element [1] của self.encoder_blocks_id
47         eb2 = dùng backbone.get_layer với index = element [2] của self.encoder_blocks_id
48         eb3 = dùng backbone.get_layer với index = element [3] của self.encoder_blocks_id
49
50         eb0 =
51         eb1 =
52         eb2 =
53         eb3 =
54
55         # bridge
56         ...
57         br = dùng backbone.get_layer với index = self.encoder_blocks_id
58         ...
59         br =

```

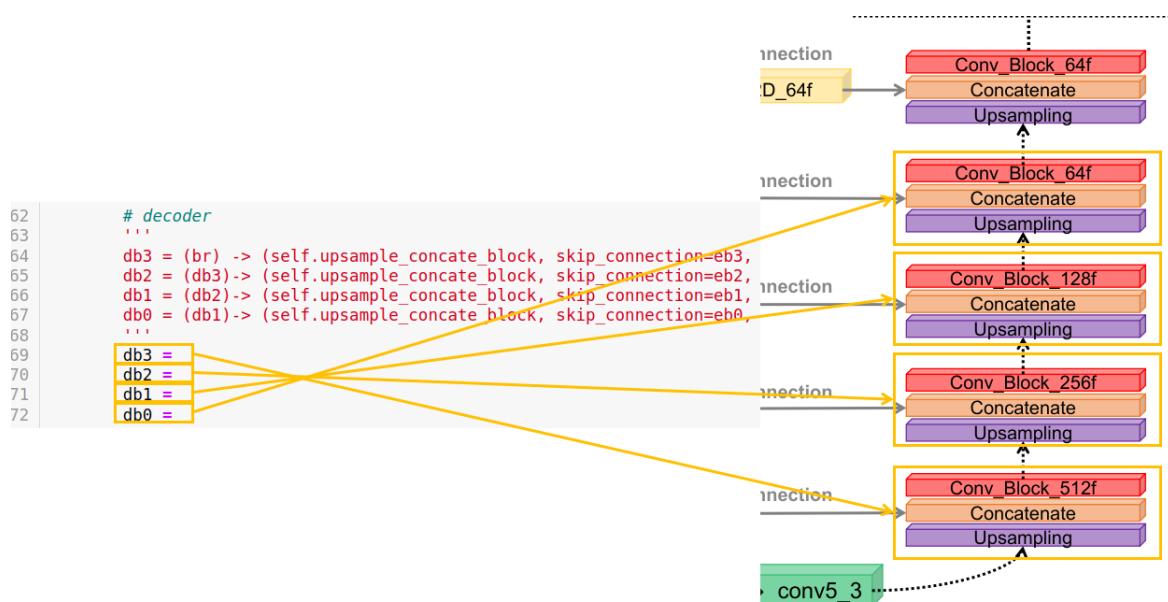


Hình 22: Hint encoder và trích xuất cho skip connection

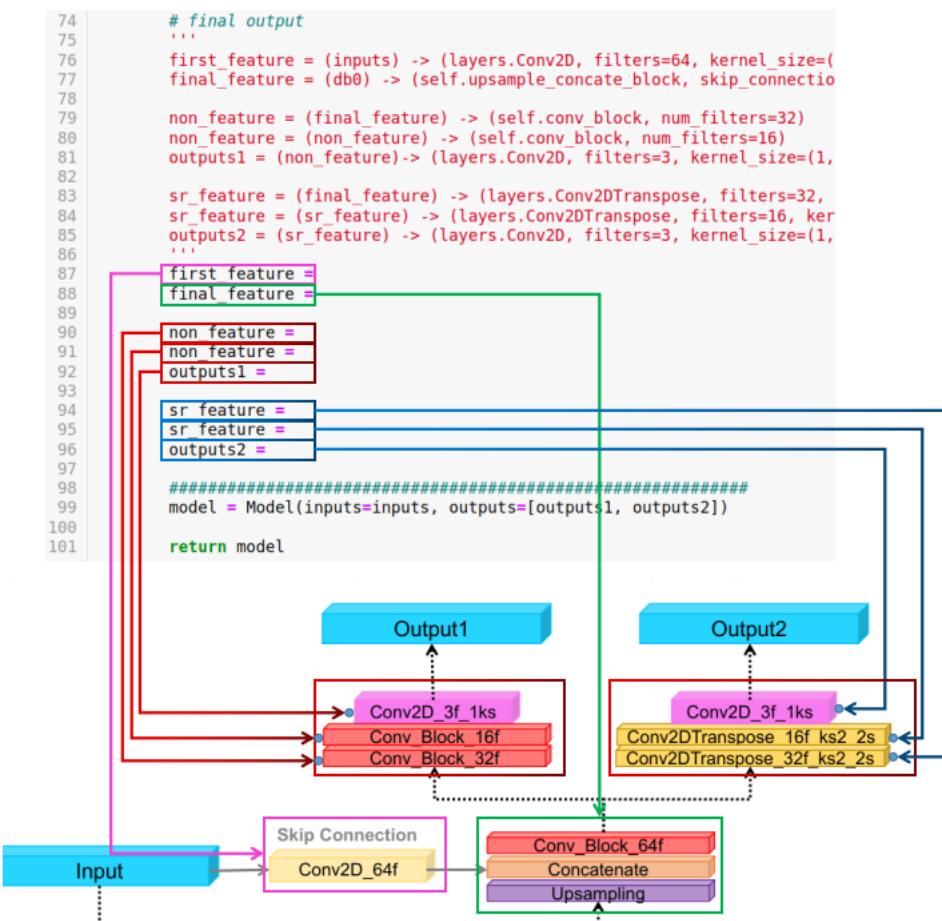
Cuối cùng sẽ đi qua Conv2D tương tự như Output1 để tạo ảnh high resolution có kích thước x4 lần ảnh Input



Hình 23: Các block được sử dụng trong decoder



Hình 24: Hint decoder



Hình 25: Hint multi-task output