https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47

Segmentasi Citra dengn Model U-Net

Model U-Net adalah jaringan saraf konvolusional (CNN) sederhana yang digunakan untuk segmentasi biner, yaitu klasifikasi berbasis piksel latar depan dan latar belakang.Yang terutama terdiri dari dua bagian yaitu:

1. Contracting Path: menerapkan serangkaian lapisan konv dan lapisan downsampling (penyatuan maksimal) untuk mengurangi ukuran spasial
2. Expanding Path: menerapkan serangkaian lapisan upsampling untuk merekonstruksi ukuran spasial masukan.

Kedua bagian tersebut dihubungkan menggunakan lapisan penggabungan di antara level yang berbeda. Ini memungkinkan mempelajari berbagai fitur di berbagai level. Pada akhirnya kami memiliki lapisan konv 1x1 sederhana untuk mengurangi jumlah saluran menjadi 1.

Import Library

```
[3]  1 !pip install git+https://github.com/tensorflow/examples.git
```

```
[4]  1 import tensorflow as tf
```

```
[5]  1 from tensorflow_examples.models.pix2pix import pix2pix
     2
     3 import tensorflow_datasets as tfds
     4
     5 from IPython.display import clear_output
     6 import matplotlib.pyplot as plt
```

Dataset

Dataset yang akan digunakan untuk tutorial ini adalah Oxford-IIIT Pet Dataset, yang dibuat oleh Parkhi et al. Dataset terdiri dari gambar, labelnya yang sesuai, dan mask berdasarkan piksel. Mask pada dasarnya adalah label untuk setiap piksel. Setiap piksel diberikan salah satu dari tiga kategori:
Kelas 1: Piksel milik hewan peliharaan.
Kelas 2: Piksel yang berbatasan dengan hewan peliharaan.
Kelas 3: Tidak ada piksel di atas / Sekitar.

```
[6]  1 dataset, info = tfds.load('oxford_iiit_pet:3.*.*', with_info=True)
```

Kode berikut melakukan augmentasi sederhana untuk membalik gambar. Selain itu, gambar dinormalisasi menjadi [0,1]. Terakhir, seperti yang disebutkan di atas, piksel dalam mask segmentasi diberi label {1, 2, 3}. Demi kenyamanan, mari kurangi 1 dari masker segmentasi, yang menghasilkan label sebagai berikut: {0, 1, 2}.

[8]
```
def normalize(input_image, input_mask):
  input_image = tf.cast(input_image, tf.float32) / 255.0
  input_mask -= 1
  return input_image, input_mask
```

[9]
```
@tf.function
def load_image_train(datapoint):
  input_image = tf.image.resize(datapoint['image'], (128, 128))
  input_mask = tf.image.resize(datapoint['segmentation_mask'], (128, 128))

  if tf.random.uniform(()) > 0.5:
    input_image = tf.image.flip_left_right(input_image)
    input_mask = tf.image.flip_left_right(input_mask)

  input_image, input_mask = normalize(input_image, input_mask)

  return input_image, input_mask
```

[10]
```
def load_image_test(datapoint):
  input_image = tf.image.resize(datapoint['image'], (128, 128))
  input_mask = tf.image.resize(datapoint['segmentation_mask'], (128, 128))

  input_image, input_mask = normalize(input_image, input_mask)

  return input_image, input_mask
```

[11]
```
TRAIN_LENGTH = info.splits['train'].num_examples
BATCH_SIZE = 64
BUFFER_SIZE = 1000
STEPS_PER_EPOCH = TRAIN_LENGTH // BATCH_SIZE
```

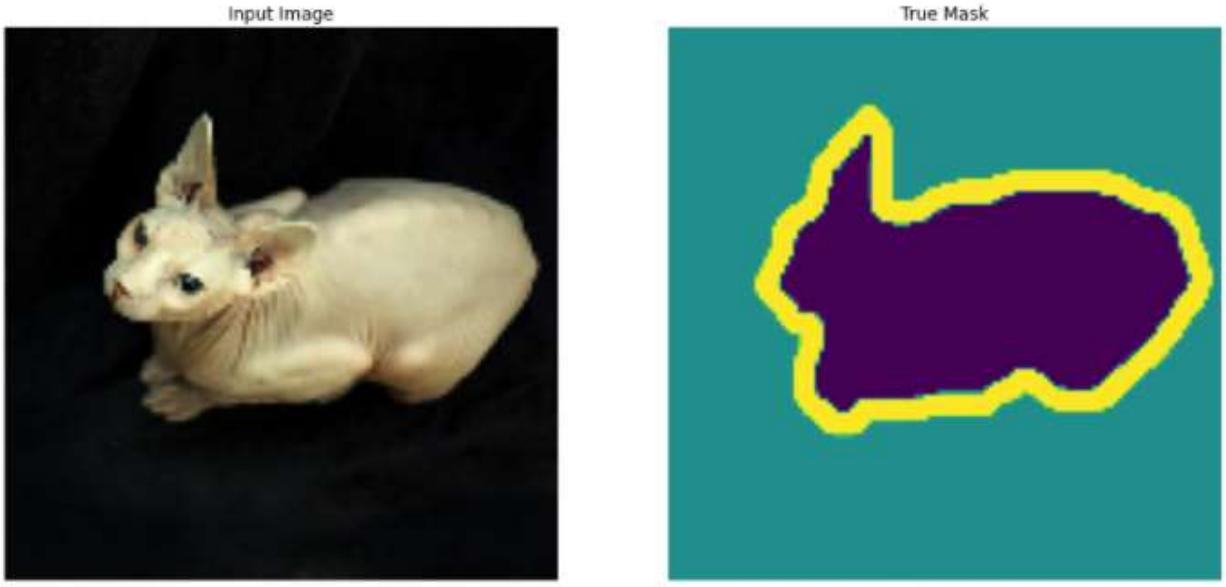
[12]
```
train = dataset['train'].map(load_image_train, num_parallel_calls=tf.data.AUTOTUNE)
test = dataset['test'].map(load_image_test)
```

[13]
```
train_dataset = train.cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE).repeat()
train_dataset = train_dataset.prefetch(buffer_size=tf.data.AUTOTUNE)
test_dataset = test.batch(BATCH_SIZE)
```

Mari kita lihat contoh gambar dan itu mask koreponding dari dataset.

```
[14]  1 def display(display_list):
      2   plt.figure(figsize=(15, 15))
      3
      4   title = ['Input Image', 'True Mask', 'Predicted Mask']
      5
      6   for i in range(len(display_list)):
      7     plt.subplot(1, len(display_list), i+1)
      8     plt.title(title[i])
      9     plt.imshow(tf.keras.preprocessing.image.array_to_img(display_list[i]))
     10     plt.axis('off')
     11   plt.show()
```

```
[15]  1 for image, mask in train.take(1):
      2   sample_image, sample_mask = image, mask
      3 display([sample_image, sample_mask])
```



Model yang digunakan di sini adalah U-Net yang dimodifikasi. U-Net terdiri dari encoder (downsampler) dan decoder (upsampler). Untuk mempelajari fitur-fitur canggih, dan mengurangi jumlah parameter yang dapat dilatih, model yang dilatih sebelumnya dapat digunakan sebagai pembuat enkode. Jadi, pembuat enkode untuk tugas ini akan menjadi model MobileNetV2 yang telah dilatih sebelumnya, yang keluaran perantaranya akan digunakan, dan dekodernya akan menjadi blok upample yang sudah diimplementasikan dalam Contoh TensorFlow dalam tutorial Pix2pix.

Alasan untuk mengeluarkan tiga saluran adalah karena ada tiga kemungkinan label untuk setiap piksel. Anggap ini sebagai multi-klasifikasi di mana setiap piksel diklasifikasikan ke dalam tiga kelas.

```
[16]  1 OUTPUT_CHANNELS = 3
```

[17]
```
base_model = tf.keras.applications.MobileNetV2(input_shape=[128, 128, 3], include_top=False)

# Use the activations of these layers
layer_names = [
    'block_1_expand_relu',   # 64x64
    'block_3_expand_relu',   # 32x32
    'block_6_expand_relu',   # 16x16
    'block_13_expand_relu',  # 8x8
    'block_16_project',      # 4x4
]
base_model_outputs = [base_model.get_layer(name).output for name in layer_names]

# Create the feature extraction model
down_stack = tf.keras.Model(inputs=base_model.input, outputs=base_model_outputs)

down_stack.trainable = False
```

[18]
```
up_stack = [
    pix2pix.upsample(512, 3),  # 4x4 -> 8x8
    pix2pix.upsample(256, 3),  # 8x8 -> 16x16
    pix2pix.upsample(128, 3),  # 16x16 -> 32x32
    pix2pix.upsample(64, 3),   # 32x32 -> 64x64
]
```

[20]
```
def unet_model(output_channels):
  inputs = tf.keras.layers.Input(shape=[128, 128, 3])

  # Downsampling through the model
  skips = down_stack(inputs)
  x = skips[-1]
  skips = reversed(skips[:-1])

  # Upsampling and establishing the skip connections
  for up, skip in zip(up_stack, skips):
    x = up(x)
    concat = tf.keras.layers.Concatenate()
    x = concat([x, skip])

  # This is the last layer of the model
  last = tf.keras.layers.Conv2DTranspose(
      output_channels, 3, strides=2,
      padding='same')  #64x64 -> 128x128

  x = last(x)

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

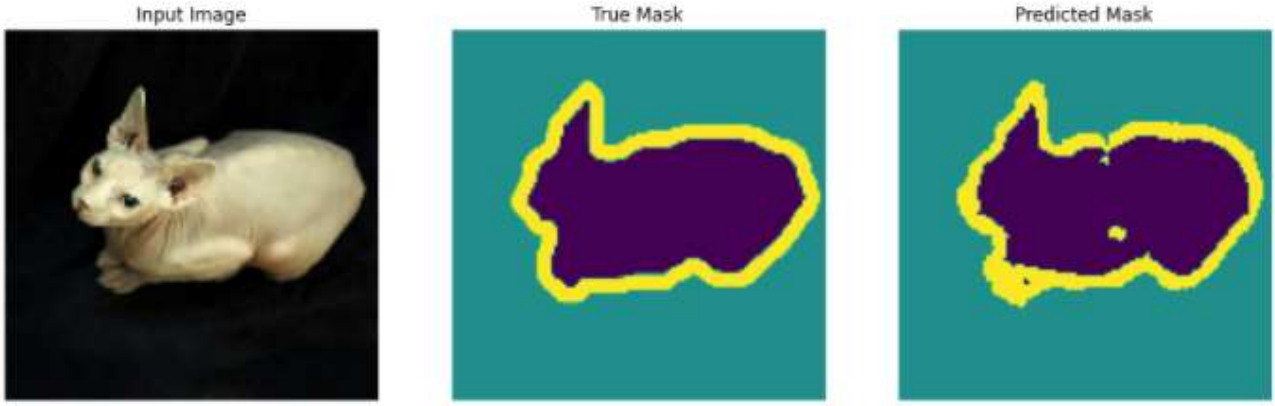
[21]
```
model = unet_model(OUTPUT_CHANNELS)
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

[22]
```
tf.keras.utils.plot_model(model, show_shapes=True)
```

[23]
```
def create_mask(pred_mask):
  pred_mask = tf.argmax(pred_mask, axis=-1)
  pred_mask = pred_mask[..., tf.newaxis]
  return pred_mask[0]
```

[24]
```
def show_predictions(dataset=None, num=1):
  if dataset:
    for image, mask in dataset.take(num):
      pred_mask = model.predict(image)
      display([image[0], mask[0], create_mask(pred_mask)])
  else:
    display([sample_image, sample_mask,
             create_mask(model.predict(sample_image[tf.newaxis, ...]))])
```

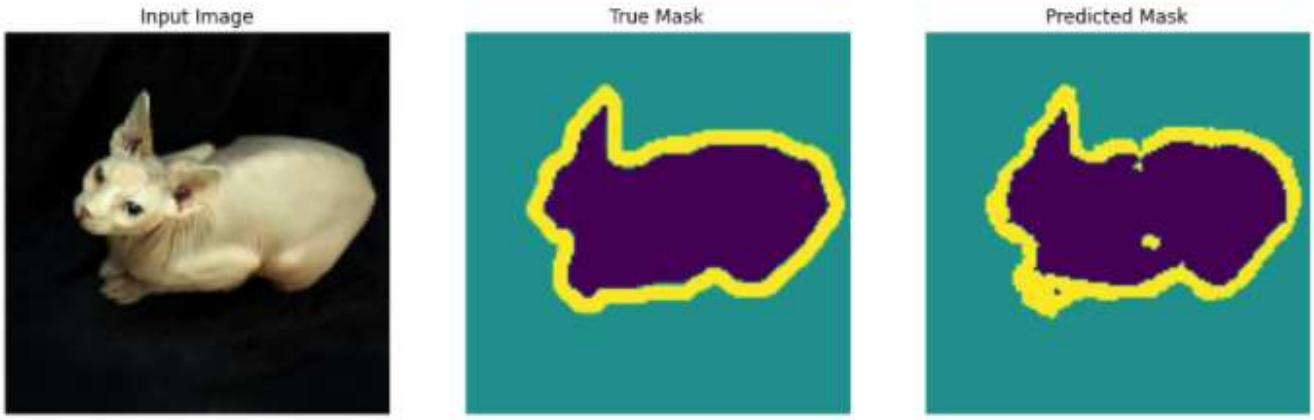
```
show_predictions()
```



[26]
```
class DisplayCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs=None):
    clear_output(wait=True)
    show_predictions()
    print ('\nSample Prediction after epoch {}\n'.format(epoch+1))
```

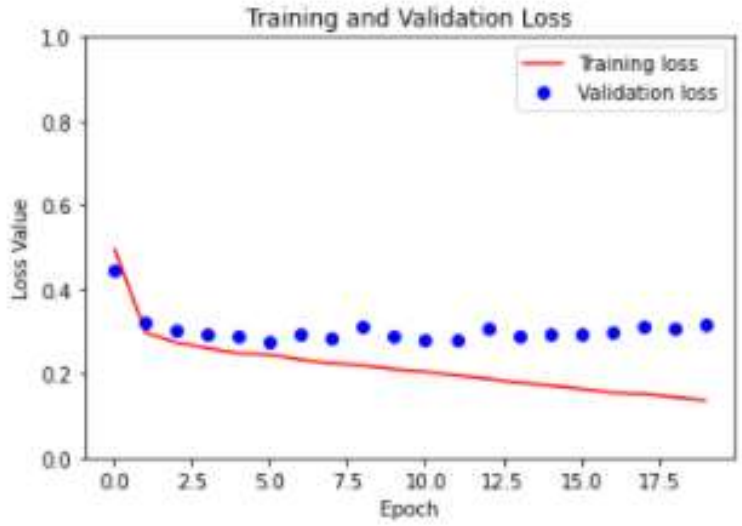
[27]
```
EPOCHS = 20
VAL_SUBSPLITS = 5
VALIDATION_STEPS = info.splits['test'].num_examples//BATCH_SIZE//VAL_SUBSPLITS

model_history = model.fit(train_dataset, epochs=EPOCHS,
                          steps_per_epoch=STEPS_PER_EPOCH,
                          validation_steps=VALIDATION_STEPS,
                          validation_data=test_dataset,
                          callbacks=[DisplayCallback()])
```




```
Sample Prediction after epoch 20
```

```
[31]  1 loss = model_history.history['loss']
      2 val_loss = model_history.history['val_loss']
      3
      4 plt.figure()
      5 plt.plot(model_history.epoch, loss, 'r', label='Training loss')
      6 plt.plot(model_history.epoch, val_loss, 'bo', label='Validation loss')
      7 plt.title('Training and Validation Loss')
      8 plt.xlabel('Epoch')
      9 plt.ylabel('Loss Value')
     10 plt.ylim([0, 1])
     11 plt.legend()
     12 plt.show()
```



```
1 show_predictions(test_dataset, 3)
```