COMP6685_Lab7_Keras_UNET_OXFORD_last

March 2, 2025

1 Image segmentation with a U-Net-like architecture

In this tutorial we will learn how to use a U-NET architecture, showing that the training on the Oxford Pets dataset https://www.robots.ox.ac.uk/~vgg/data/pets/.

Google Colab is suggested for this task.

The Oxford-IIIT Pet Dataset Omkar M Parkhi and Andrea Vedaldi and Andrew Zisserman and C. V. Jawahar

The Oxford-IIIT Pet dataset and annotations are roughly 800 MB. There are 37 category pet dataset with roughly 200 images for each class. The images have a large variations in scale, pose and lighting. All images have an associated ground truth annotation of breed, head ROI, and pixel level trimap segmentation.

Initialisation of the program

The program starts with the importing of typical Keras and other Python service modules. Furthermore we download data from Oxford Pets Dataset

[ ]:
```
import keras #to solve a compatibility issue of different functions

from tensorflow.keras import utils
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.optimizers import SGD, Adam, RMSprop

# importing of service libraries
import numpy as np
import matplotlib.pyplot as plt

# importing of data from Oxford Pets Dataset
#!curl -O http://www.robots.ox.ac.uk/~vgg/data/pets/data/images.tar.gz
!wget http://www.robots.ox.ac.uk/~vgg/data/pets/data/images.tar.gz
#!curl -O https://www.robots.ox.ac.uk/~vgg/data/pets/data/annotations.tar.gz
!wget https://www.robots.ox.ac.uk/~vgg/data/pets/data/annotations.tar.gz
!tar -xzvf images.tar.gz
!tar -xzvf annotations.tar.gz
```

```
images/Abyssinian_100.jpg | annotations/trimaps/Abyssinian_100.png
images/Abyssinian_101.jpg | annotations/trimaps/Abyssinian_101.png
images/Abyssinian_102.jpg | annotations/trimaps/Abyssinian_102.png
images/Abyssinian_103.jpg | annotations/trimaps/Abyssinian_103.png
images/Abyssinian_104.jpg | annotations/trimaps/Abyssinian_104.png
images/Abyssinian_105.jpg | annotations/trimaps/Abyssinian_105.png
images/Abyssinian_106.jpg | annotations/trimaps/Abyssinian_106.png
images/Abyssinian_107.jpg | annotations/trimaps/Abyssinian_107.png
images/Abyssinian_108.jpg | annotations/trimaps/Abyssinian_108.png
images/Abyssinian_109.jpg | annotations/trimaps/Abyssinian_109.png
images/Abyssinian_11.jpg | annotations/trimaps/Abyssinian_11.png
images/Abyssinian_110.jpg | annotations/trimaps/Abyssinian_110.png
images/Abyssinian_111.jpg | annotations/trimaps/Abyssinian_111.png
images/Abyssinian_112.jpg | annotations/trimaps/Abyssinian_112.png
images/Abyssinian_113.jpg | annotations/trimaps/Abyssinian_113.png
images/Abyssinian_114.jpg | annotations/trimaps/Abyssinian_114.png
images/Abyssinian_115.jpg | annotations/trimaps/Abyssinian_115.png
images/Abyssinian_116.jpg | annotations/trimaps/Abyssinian_116.png
images/Abyssinian_117.jpg | annotations/trimaps/Abyssinian_117.png
images/Abyssinian_118.jpg | annotations/trimaps/Abyssinian_118.png
images/Abyssinian_119.jpg | annotations/trimaps/Abyssinian_119.png
images/Abyssinian_12.jpg | annotations/trimaps/Abyssinian_12.png
images/Abyssinian_120.jpg | annotations/trimaps/Abyssinian_120.png
images/Abyssinian_121.jpg | annotations/trimaps/Abyssinian_121.png
images/Abyssinian_122.jpg | annotations/trimaps/Abyssinian_122.png
images/Abyssinian_123.jpg | annotations/trimaps/Abyssinian_123.png
images/Abyssinian_124.jpg | annotations/trimaps/Abyssinian_124.png
images/Abyssinian_125.jpg | annotations/trimaps/Abyssinian_125.png
images/Abyssinian_126.jpg | annotations/trimaps/Abyssinian_126.png
images/Abyssinian_127.jpg | annotations/trimaps/Abyssinian_127.png
images/Abyssinian_128.jpg | annotations/trimaps/Abyssinian_128.png
images/Abyssinian_129.jpg | annotations/trimaps/Abyssinian_129.png
images/Abyssinian_13.jpg | annotations/trimaps/Abyssinian_13.png
```

Display image and segmentation

```
[ ]: from IPython.display import Image, display
     from tensorflow.keras.preprocessing.image import load_img
     from PIL import ImageOps

     # Display input image #7
     display(Image(filename=input_img_paths[9]))

     # Display auto-contrast version of corresponding target (per-pixel categories)
     img = ImageOps.autocontrast(load_img(target_img_paths[9]))
     display(img)
```

Prepare Sequence class to load and vectorize batches of data

```
[ ]: from tensorflow.keras.preprocessing.image import load_img

class OxfordPets(keras.utils.Sequence):
    """Helper to iterate over the data (as Numpy arrays)."""

    def __init__(self, batch_size, img_size, input_img_paths, target_img_paths):
        self.batch_size = batch_size
        self.img_size = img_size
        self.input_img_paths = input_img_paths
        self.target_img_paths = target_img_paths

    def __len__(self):
        return len(self.target_img_paths) // self.batch_size

    def __getitem__(self, idx):
        """Returns tuple (input, target) correspond to batch #idx."""
        i = idx * self.batch_size
        batch_input_img_paths = self.input_img_paths[i : i + self.batch_size]
        batch_target_img_paths = self.target_img_paths[i : i + self.batch_size]
        x = np.zeros((self.batch_size,) + self.img_size + (3,), dtype="float32")
        for j, path in enumerate(batch_input_img_paths):
            img = load_img(path, target_size=self.img_size)
            x[j] = img
        y = np.zeros((self.batch_size,) + self.img_size + (1,), dtype="uint8")
        for j, path in enumerate(batch_target_img_paths):
            img = load_img(path, target_size=self.img_size,␣
↪color_mode="grayscale")
            y[j] = np.expand_dims(img, 2)
            # Ground truth labels are 1, 2, 3. Subtract one to make them 0, 1,␣
↪2:
            y[j] -= 1
        return x, y
```

Prepare U-Net Xception-style model Here the U-NET architecture is defined. In Keras, the input layer itself is not a layer, but a tensor. It's the starting tensor you send to the first hidden layer. This tensor must have the same shape as your training data Example: if you have 200 images of 160x160 pixels in RGB (3 channels), the shape of your input data is (200,160,160,3).

With respect to the orginal formulation of U-NET, in this example the subsampling/pooling layer is replace with a residual layer. Residual Network (ResNet) is a deep learning model used for computer vision applications. It is a Convolutional Neural Network (CNN) architecture designed to support hundreds or thousands of convolutional layers.

Networks with large number (even thousands) of layers can be trained easily without increasing the training error percentage. to konw more see https://deepchecks.com/glossary/resnet/

[ ]:
```python
from tensorflow.keras import layers


def get_model(img_size, num_classes):
    inputs = keras.Input(shape=img_size + (3,))

    ### [First half of the network: downsampling inputs] ###

    # Entry block
    x = layers.Conv2D(32, 3, strides=2, padding="same")(inputs)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)

    previous_block_activation = x  # Set aside residual

    # Blocks 1, 2, 3 are identical apart from the feature depth.
    for filters in [64, 128, 256]:
        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(filters, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(filters, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.MaxPooling2D(3, strides=2, padding="same")(x)

        # Project residual
        residual = layers.Conv2D(filters, 1, strides=2, padding="same")(
            previous_block_activation
        )
        x = layers.add([x, residual])  # Add back residual
        previous_block_activation = x  # Set aside next residual

    ### [Second half of the network: upsampling inputs] ###

    for filters in [256, 128, 64, 32]:
        x = layers.Activation("relu")(x)
        x = layers.Conv2DTranspose(filters, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.Activation("relu")(x)
        x = layers.Conv2DTranspose(filters, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)
        x = layers.UpSampling2D(2)(x)

        # Project residual
        residual = layers.UpSampling2D(2)(previous_block_activation)
        residual = layers.Conv2D(filters, 1, padding="same")(residual)
        x = layers.add([x, residual])  # Add back residual
        previous_block_activation = x  # Set aside next residual

    # Add a per-pixel classification layer
    outputs = layers.Conv2D(num_classes, 3, activation="softmax",␣
↪padding="same")(x)

    # Define the model
    model = keras.Model(inputs, outputs)
    return model


# Free up RAM in case the model definition cells were run multiple times
keras.backend.clear_session()

# Build model
model = get_model(img_size, num_classes)
model.summary()
```

```
Model: "functional"


 Layer (type)                 Output Shape                Param #  Connected␣
↪to

 input_layer (InputLayer)     (None, 160, 160, 3)               0  -          ␣
↪

 conv2d (Conv2D)              (None, 80, 80, 32)              896 ␣
↪input_layer[0][0]

 batch_normalization          (None, 80, 80, 32)              128 ␣
↪conv2d[0][0]
 (BatchNormalization)                                                         ␣
↪

 activation (Activation)      (None, 80, 80, 32)                0 ␣
↪batch_normalization[0…

 activation_1 (Activation)    (None, 80, 80, 32)                0 ␣
↪activation[0][0]
```

```
separable_conv2d          (None, 80, 80, 64)          2,400 ␣
↪activation_1[0][0]
(SeparableConv2D)                                                ␣
↪

batch_normalization_1     (None, 80, 80, 64)            256 ␣
↪separable_conv2d[0][0]
(BatchNormalization)                                             ␣
↪

activation_2 (Activation) (None, 80, 80, 64)              0 ␣
↪batch_normalization_1…

separable_conv2d_1        (None, 80, 80, 64)          4,736 ␣
↪activation_2[0][0]
(SeparableConv2D)                                                ␣
↪

batch_normalization_2     (None, 80, 80, 64)            256 ␣
↪separable_conv2d_1[0]…
(BatchNormalization)                                             ␣
↪

max_pooling2d             (None, 40, 40, 64)              0 ␣
↪batch_normalization_2…
(MaxPooling2D)                                                   ␣
↪

conv2d_1 (Conv2D)         (None, 40, 40, 64)          2,112 ␣
↪activation[0][0]

add (Add)                 (None, 40, 40, 64)              0 ␣
↪max_pooling2d[0][0],
                                                              ␣
↪conv2d_1[0][0]

activation_3 (Activation) (None, 40, 40, 64)              0 add[0][0] ␣
↪

separable_conv2d_2        (None, 40, 40, 128)         8,896 ␣
↪activation_3[0][0]
(SeparableConv2D)                                                ␣
↪
```

```
batch_normalization_3      (None, 40, 40, 128)              512 ␣
↪separable_conv2d_2[0]…
(BatchNormalization)                                                    ␣
↪

activation_4 (Activation)  (None, 40, 40, 128)                0 ␣
↪batch_normalization_3…

separable_conv2d_3         (None, 40, 40, 128)           17,664 ␣
↪activation_4[0][0]
(SeparableConv2D)                                                       ␣
↪

batch_normalization_4      (None, 40, 40, 128)              512 ␣
↪separable_conv2d_3[0]…
(BatchNormalization)                                                    ␣
↪

max_pooling2d_1            (None, 20, 20, 128)                0 ␣
↪batch_normalization_4…
(MaxPooling2D)                                                          ␣
↪

conv2d_2 (Conv2D)          (None, 20, 20, 128)            8,320  add[0][0] ␣
↪

add_1 (Add)                (None, 20, 20, 128)                0 ␣
↪max_pooling2d_1[0][0],
                                                                ␣
↪conv2d_2[0][0]

activation_5 (Activation)  (None, 20, 20, 128)                0 ␣
↪add_1[0][0]

separable_conv2d_4         (None, 20, 20, 256)           34,176 ␣
↪activation_5[0][0]
(SeparableConv2D)                                                       ␣
↪

batch_normalization_5      (None, 20, 20, 256)            1,024 ␣
↪separable_conv2d_4[0]…
(BatchNormalization)                                                    ␣
↪

activation_6 (Activation)  (None, 20, 20, 256)                0 ␣
↪batch_normalization_5…
```

```
separable_conv2d_5        (None, 20, 20, 256)          68,096 ␣
↪activation_6[0][0]
(SeparableConv2D)                                                 ␣
↪

batch_normalization_6     (None, 20, 20, 256)           1,024 ␣
↪separable_conv2d_5[0]…
(BatchNormalization)                                              ␣
↪

max_pooling2d_2           (None, 10, 10, 256)               0 ␣
↪batch_normalization_6…
(MaxPooling2D)                                                    ␣
↪

conv2d_3 (Conv2D)         (None, 10, 10, 256)          33,024 ␣
↪add_1[0][0]

add_2 (Add)               (None, 10, 10, 256)               0 ␣
↪max_pooling2d_2[0][0],
                                                                ␣
↪conv2d_3[0][0]

activation_7 (Activation) (None, 10, 10, 256)               0 ␣
↪add_2[0][0]

conv2d_transpose          (None, 10, 10, 256)         590,080 ␣
↪activation_7[0][0]
(Conv2DTranspose)                                                 ␣
↪

batch_normalization_7     (None, 10, 10, 256)           1,024 ␣
↪conv2d_transpose[0][0]
(BatchNormalization)                                              ␣
↪

activation_8 (Activation) (None, 10, 10, 256)               0 ␣
↪batch_normalization_7…

conv2d_transpose_1        (None, 10, 10, 256)         590,080 ␣
↪activation_8[0][0]
(Conv2DTranspose)                                                 ␣
↪
```

```
batch_normalization_8        (None, 10, 10, 256)                1,024 ␣
↪conv2d_transpose_1[0]…
(BatchNormalization)                                                         ␣
↪

up_sampling2d_1              (None, 20, 20, 256)                    0 ␣
↪add_2[0][0]
(UpSampling2D)                                                               ␣
↪

up_sampling2d                (None, 20, 20, 256)                    0 ␣
↪batch_normalization_8…
(UpSampling2D)                                                               ␣
↪

conv2d_4 (Conv2D)            (None, 20, 20, 256)               65,792 ␣
↪up_sampling2d_1[0][0]

add_3 (Add)                  (None, 20, 20, 256)                    0 ␣
↪up_sampling2d[0][0],
                                                                      ␣
↪conv2d_4[0][0]

activation_9 (Activation)    (None, 20, 20, 256)                    0 ␣
↪add_3[0][0]

conv2d_transpose_2           (None, 20, 20, 128)              295,040 ␣
↪activation_9[0][0]
(Conv2DTranspose)                                                            ␣
↪

batch_normalization_9        (None, 20, 20, 128)                  512 ␣
↪conv2d_transpose_2[0]…
(BatchNormalization)                                                         ␣
↪

activation_10                (None, 20, 20, 128)                    0 ␣
↪batch_normalization_9…
(Activation)                                                                 ␣
↪

conv2d_transpose_3           (None, 20, 20, 128)              147,584 ␣
↪activation_10[0][0]
(Conv2DTranspose)                                                            ␣
↪
```

```
batch_normalization_10      (None, 20, 20, 128)            512 ␣
↪conv2d_transpose_3[0]…
(BatchNormalization)                                                   ␣
↪

up_sampling2d_3             (None, 40, 40, 256)              0 ␣
↪add_3[0][0]
(UpSampling2D)                                                         ␣
↪

up_sampling2d_2             (None, 40, 40, 128)              0 ␣
↪batch_normalization_1…
(UpSampling2D)                                                         ␣
↪

conv2d_5 (Conv2D)           (None, 40, 40, 128)         32,896 ␣
↪up_sampling2d_3[0][0]

add_4 (Add)                 (None, 40, 40, 128)              0 ␣
↪up_sampling2d_2[0][0],
                                                               ␣
↪conv2d_5[0][0]

activation_11               (None, 40, 40, 128)              0 ␣
↪add_4[0][0]
(Activation)                                                           ␣
↪

conv2d_transpose_4          (None, 40, 40, 64)          73,792 ␣
↪activation_11[0][0]
(Conv2DTranspose)                                                      ␣
↪

batch_normalization_11      (None, 40, 40, 64)             256 ␣
↪conv2d_transpose_4[0]…
(BatchNormalization)                                                   ␣
↪

activation_12               (None, 40, 40, 64)               0 ␣
↪batch_normalization_1…
(Activation)                                                           ␣
↪

conv2d_transpose_5          (None, 40, 40, 64)          36,928 ␣
↪activation_12[0][0]
```

```
(Conv2DTranspose)                                                        ␣
↪

batch_normalization_12      (None, 40, 40, 64)            256 ␣
↪conv2d_transpose_5[0]…
(BatchNormalization)                                                     ␣
↪

up_sampling2d_5             (None, 80, 80, 128)             0 ␣
↪add_4[0][0]
(UpSampling2D)                                                           ␣
↪

up_sampling2d_4             (None, 80, 80, 64)              0 ␣
↪batch_normalization_1…
(UpSampling2D)                                                           ␣
↪

conv2d_6 (Conv2D)           (None, 80, 80, 64)          8,256 ␣
↪up_sampling2d_5[0][0]

add_5 (Add)                 (None, 80, 80, 64)              0 ␣
↪up_sampling2d_4[0][0],
                                                                ␣
↪conv2d_6[0][0]

activation_13               (None, 80, 80, 64)              0 ␣
↪add_5[0][0]
(Activation)                                                             ␣
↪

conv2d_transpose_6          (None, 80, 80, 32)         18,464 ␣
↪activation_13[0][0]
(Conv2DTranspose)                                                        ␣
↪

batch_normalization_13      (None, 80, 80, 32)            128 ␣
↪conv2d_transpose_6[0]…
(BatchNormalization)                                                     ␣
↪

activation_14               (None, 80, 80, 32)              0 ␣
↪batch_normalization_1…
(Activation)                                                             ␣
↪
```

```
conv2d_transpose_7          (None, 80, 80, 32)              9,248 ␣
↪activation_14[0][0]
(Conv2DTranspose)                                                      ␣
↪

batch_normalization_14      (None, 80, 80, 32)                128 ␣
↪conv2d_transpose_7[0]…
(BatchNormalization)                                                   ␣
↪

up_sampling2d_7             (None, 160, 160, 64)                0 ␣
↪add_5[0][0]
(UpSampling2D)                                                         ␣
↪

up_sampling2d_6             (None, 160, 160, 32)                0 ␣
↪batch_normalization_1…
(UpSampling2D)                                                         ␣
↪

conv2d_7 (Conv2D)           (None, 160, 160, 32)            2,080 ␣
↪up_sampling2d_7[0][0]

add_6 (Add)                 (None, 160, 160, 32)                0 ␣
↪up_sampling2d_6[0][0],
                                                                  ␣
↪conv2d_7[0][0]

conv2d_8 (Conv2D)           (None, 160, 160, 37)           10,693 ␣
↪add_6[0][0]


 Total params: 2,068,805 (7.89 MB)

 Trainable params: 2,065,029 (7.88 MB)

 Non-trainable params: 3,776 (14.75 KB)
```

Creation of train and validation set

Defining here a validation split

```
[ ]: import random

     # Split our img paths into a training and a validation set
```

```
val_samples = 1000
random.Random(1337).shuffle(input_img_paths)
random.Random(1337).shuffle(target_img_paths)
train_input_img_paths = input_img_paths[:-val_samples]
train_target_img_paths = target_img_paths[:-val_samples]
val_input_img_paths = input_img_paths[-val_samples:]
val_target_img_paths = target_img_paths[-val_samples:]

# Instantiate data Sequences for each split
train_gen = OxfordPets(
    batch_size, img_size, train_input_img_paths, train_target_img_paths
)
val_gen = OxfordPets(batch_size, img_size, val_input_img_paths,␣
 ↪val_target_img_paths)
```

Training of the model

[ ]:
```
# Configure the model for training.
# We use the "sparse" version of categorical_crossentropy
# because our target data is integers.
model.compile(optimizer="rmsprop", loss="sparse_categorical_crossentropy")

callbacks = [
    keras.callbacks.ModelCheckpoint("oxford_segmentation.h5",␣
 ↪save_best_only=True)
]

# Train the model, doing validation at the end of each epoch.
epochs = 15
model.fit(train_gen, epochs=epochs, validation_data=val_gen,␣
 ↪callbacks=callbacks)
```

```
/usr/local/lib/python3.11/dist-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
```

```
Epoch 1/15
199/199             0s 155ms/step -
loss: 1.6986
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```

```
199/199              33s 167ms/step -
loss: 0.2902 - val_loss: 0.3706
Epoch 9/15
199/199              33s 166ms/step -
loss: 0.2746 - val_loss: 0.3842
Epoch 10/15
199/199              35s 173ms/step -
loss: 0.2502 - val_loss: 0.3755
Epoch 11/15
199/199              33s 165ms/step -
loss: 0.2328 - val_loss: 0.3721
Epoch 12/15
199/199              34s 169ms/step -
loss: 0.2209 - val_loss: 0.4110
Epoch 13/15
199/199              34s 170ms/step -
loss: 0.2155 - val_loss: 0.3853
Epoch 14/15
199/199              33s 166ms/step -
loss: 0.2034 - val_loss: 0.3778
Epoch 15/15
199/199              35s 176ms/step -
loss: 0.1932 - val_loss: 0.3990
```

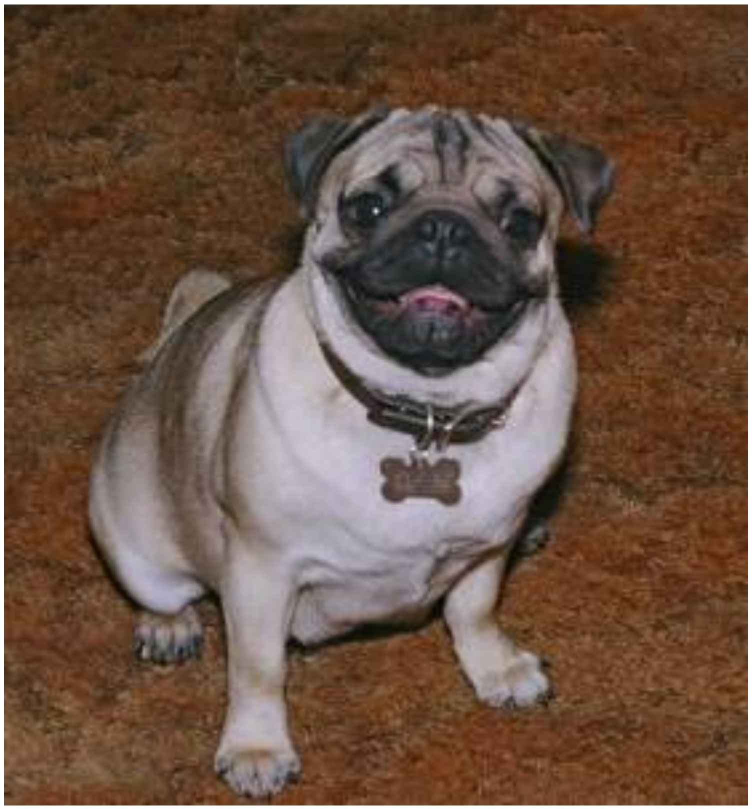
```
[ ]: <keras.src.callbacks.history.History at 0x7bd3dc08ac50>
```

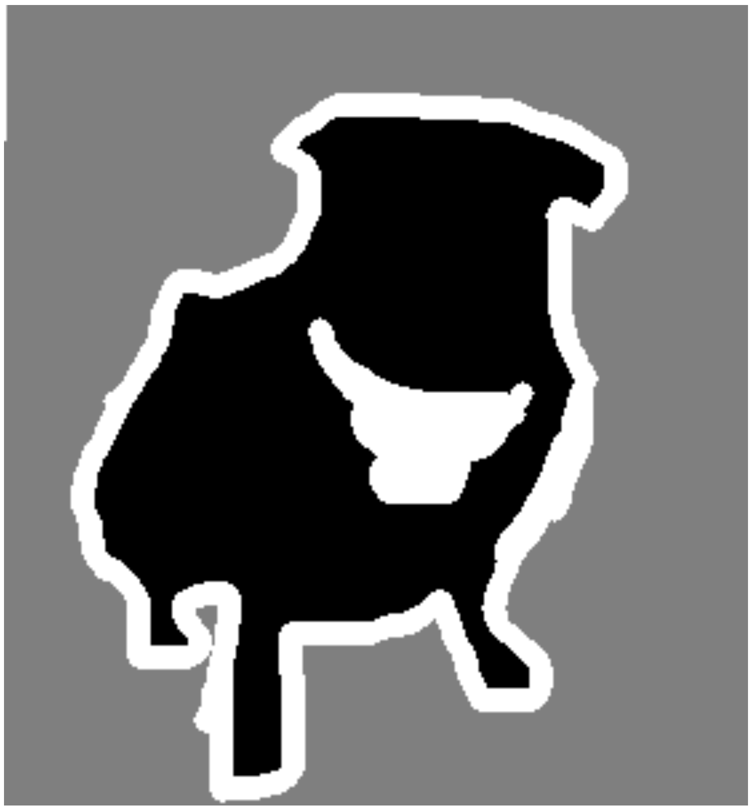
Visualize predictions

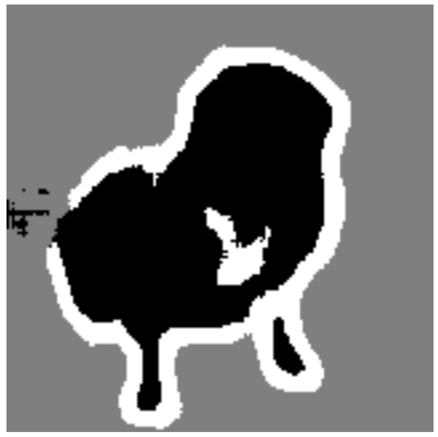
```
[ ]: # Generate predictions for all images in the validation set

     # import tensorflow.keras.preprocessing.image

     from tensorflow.keras.preprocessing.image import array_to_img

     val_gen = OxfordPets(batch_size, img_size, val_input_img_paths,␣
      ↪val_target_img_paths)
     val_preds = model.predict(val_gen)


     def display_mask(i):
         """Quick utility to display a model's prediction."""
         mask = np.argmax(val_preds[i], axis=-1)
```

```
    mask = np.expand_dims(mask, axis=-1)
    img = ImageOps.autocontrast(array_to_img(mask))
    display(img)


# Display results for validation image #10
i = 10

# Display input image
display(Image(filename=val_input_img_paths[i]))

# Display ground-truth target mask
img = ImageOps.autocontrast(load_img(val_target_img_paths[i]))
display(img)

# Display mask predicted by our model
display_mask(i)  # Note that the model only sees inputs at 150x150.
```

```
31/31                 9s 205ms/step
```







1.1 Conclusions

Today we learned to train U-NET, and to use such architecure to work on computer vision.