Semantic Segmentation of the Aerial Images

Arash Azari

July 2021 - version 1.0

1 Semantic Segmentation of the Satellite Images

- In the original data files that I have, there are 30 images, 24 masked images (ground truth) and 1 outlier where the masked and image do not match.
- I used the Convolutional Neural Network U-Net which previous reports suggest that it performs well for semantic segmentation cases.
- As the number of images are too small for training, I have decided to use some of the augmentation techniques and increase the number of images (and their masks) to improve the network training .
- The augmentation techniques that I used are small angle rotation, flip left to right and top to bottom, and zoom in. Each training image and its mask went through the same augmentation process. Now the total number of images/masks, including the original ones, are 324 for each set.

- The predicted segmentations are satisfactory in some cases, but in overall I do not get any sharp segmentation with clear boundaries and correct labeling compared to the validation/ground truth images (I have not used any threshold to manipulate the grayscale output).
- Another option which I have not tried was the manipulation or training based on the colors, the similarity between the color of the streets and the roofs is challenging and there must be some solutions for that.
- I have used two metrics for comparison; "Accuracy" and "Mean Intersection-Over-Union" and played with different parameters to tune the training parameters like the optimizer, the learning rate, the metrics. Besides that, I used the focal loss function.
- Something which is puzzling, is the performance of the Mean Intersection-Over-Union evaluation metric in Keras package; as its name suggests, it should present the best performance among other metrics for semantic segmentation specially in case of binary segmentation, but in my case it does not perform as expected, probably I missed something, a parameter or additional analysis or option. The focal loss function performs slightly better that binary_crossentropy loss function while using "Accuracy as metrics in both cases.

Initial version - April 2021

- Loading the required libraries

(not all of them are necessary)

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

```
Collecting tensorflow-addons
  Downloading tensorflow_addons-0.14.0-cp37-cp37m-manylinux_2_12_x86_64.manylinu
x2010_x86_64.whl (1.1 MB)
    ␣
 ↪|âŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰLâŰ
 ↪1.1 MB 4.3 MB/s
Requirement already satisfied: typeguard>=2.7 in
/usr/local/lib/python3.7/dist-packages (from tensorflow-addons) (2.7.1)
Installing collected packages: tensorflow-addons
Successfully installed tensorflow-addons-0.14.0
```

[2]:
```python
from keras.models import Model, load_model
from keras.layers import Input
from keras.layers.core import Dropout, Lambda
from keras.layers.convolutional import Conv2D, Conv2DTranspose
from keras.layers.pooling import MaxPooling2D
from keras.layers.merge import concatenate
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras import backend as K
from tensorflow.keras.metrics import MeanIoU

import tensorflow as tf
from tensorflow.keras.layers.experimental import preprocessing
import tensorflow_addons as tfa


import pandas as pd
import numpy as np
import random
import os
import glob
import sys
import cv2

from IPython.display import Image, display
from tensorflow.keras.preprocessing.image import load_img
import PIL
from PIL import ImageOps
from PIL import Image

import matplotlib.pyplot as plt
%matplotlib inline

from skimage.io import imread, imshow
from skimage.transform import resize

# No Warning Messages!
```

```
import warnings
warnings.filterwarnings('ignore')
```

- Mounting the Google Drive where the data are located.

[3]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

- Defining the width, height, and the number of channels of the input images.

[4]:
```
IMG_HEIGHT = 256
IMG_WIDTH = 256
IMG_CHANNELS = 3
```

- Loading input images and the corresponding masks (ground truth)

[5]:
```
#load training and mask images as a list
train_images = []

#for directory_path in glob.glob("/content/drive/MyDrive/Colab Notebooks/Pics/
 ↪Aug/train"):
#    for img_path in glob.glob(os.path.join(directory_path, "*.png")):
fnames = [img_path for img_path in glob.glob("/content/drive/MyDrive/Colab␣
 ↪Notebooks/Pics/Aug/train/*.png")]
fnames.sort()
for img_path in fnames:
        #reading colored images
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        #option for resize
        img = cv2.resize(img, (IMG_HEIGHT, IMG_WIDTH))
        #change colors channel order
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        train_images.append(img)
#Convert list to array
train_images = np.array(train_images)

#mask/label
train_masks = []
fnames = [mask_path for mask_path in glob.glob("/content/drive/MyDrive/Colab␣
 ↪Notebooks/Pics/Aug/labels/*.png")]
fnames.sort()
for mask_path in fnames:
#for directory_path in glob.glob("/content/drive/My Drive/Colab Notebooks/Pics/
 ↪Aug/labels"):
#    for mask_path in glob.glob(os.path.join(directory_path, "*.png")):
        #loading grayscale image
        mask = cv2.imread(mask_path, 0)
        mask = cv2.resize(mask, (IMG_HEIGHT, IMG_WIDTH))
        train_masks.append(mask)
train_masks = np.array(train_masks)
```

```
[ ]: train_masks.shape
```

```
[ ]: train_images.shape
```

- Normalizing the images to [0, 1]

```
[6]: x = np.asarray(train_images, dtype=np.float32)/255
     y = np.asarray(train_masks, dtype=np.float32)/255
```

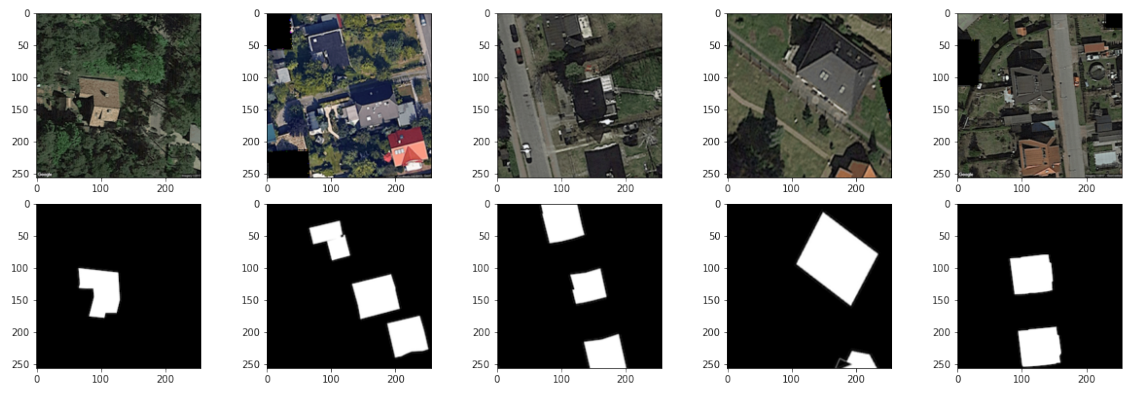
- Checking the formats

Make sure that the input, x, and validation, y, have correct dimention; 3 channel colored and 1 channel binary respectively

```
[7]: x = x.reshape(x.shape[0], x.shape[1], x.shape[2], 3)
     y = y.reshape(y.shape[0], y.shape[1], y.shape[2], 1)
```

- Check some of the images and their masks randomly to make sure they match and there is no outliers in the input data.

```
[8]: ids1 = random.sample(range(len(x)-1), 5)

     f, axarr = plt.subplots(2,5,figsize=(20, 20))
     plt.subplots_adjust(wspace=0.4, hspace=-0.78)

     for j in range(5):
        axarr[0,j].imshow(x[ids1[j]])
        axarr[1,j].imshow(y[ids1[j]].squeeze(axis=2), cmap='gray')
```



- Set aside the validation images

```
[9]: from sklearn.model_selection import train_test_split

     x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.1,␣
      ↪random_state=0)
```

```
[ ]: x_val.shape
```

- Building the UNet model;

- Filters are: encoder=[16,32,64,128,256,512], 1024 , decoder=[512,256,128,64,32,16], and [1] for the output layer.
- activation= Rectified Linear Unit, accompanied with he_normal as kernel initializer.
- Padding the same as input image (to get the same size output).
- Using dropout to prevent overfitting and improving the training.
- Using MaxPooling to reduce the computational cost (I assume MaxPooling works better for images with darker background(?))
- The convolution kernel sizes are (3, 3), except for the upsampling (transpose) (2, 2) and the final layer (1, 1), strides=1 in all cases.
- The concatenation is used as defined in the UNet model.
- The output layer samples pixel by pixel, filter = 1 and kernel =(1, 1), besides using the "sigmoid" as activation to have the output in the range of [0, 1].

- Two metrics have been used for comparison, metrics=['accuracy'] and metrics=[tf.keras.metrics.MeanIoU(num_classes=2)] besides the "adam" optimizer with smaller than default value of learning rate for the optimizer.

```
[11]: #Build the UNet model.
      inputs = tf.keras.layers.Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))
      #if the inputs are not normalized
      #s = tf.keras.layers.Lambda(lambda x: x / 255)(inputs)

      dout=0.3 #Dropout value

      #Contraction path, encoder
      c1 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',␣
       ↪kernel_initializer='he_normal', padding='same')(inputs)
      c1 = tf.keras.layers.Dropout(dout)(c1)
      c1 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',␣
       ↪kernel_initializer='he_normal', padding='same')(c1)
      p1 = tf.keras.layers.MaxPooling2D((2, 2))(c1)

      c2 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',␣
       ↪kernel_initializer='he_normal', padding='same')(p1)
      c2 = tf.keras.layers.Dropout(dout)(c2)
      c2 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',␣
       ↪kernel_initializer='he_normal', padding='same')(c2)
      p2 = tf.keras.layers.MaxPooling2D((2, 2))(c2)
```

```
c3 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(p2)
c3 = tf.keras.layers.Dropout(dout)(c3)
c3 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(c3)
p3 = tf.keras.layers.MaxPooling2D((2, 2))(c3)

c4 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(p3)
c4 = tf.keras.layers.Dropout(dout)(c4)
c4 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(c4)
p4 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(c4)

c5 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(p4)
c5 = tf.keras.layers.Dropout(dout)(c5)
c5 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(c5)
p5 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(c5)

c6 = tf.keras.layers.Conv2D(512, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(p5)
c6 = tf.keras.layers.Dropout(dout)(c6)
c6 = tf.keras.layers.Conv2D(512, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(c6)
p6 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(c6)

#Middle

cm = tf.keras.layers.Conv2D(1024, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(p6)
cm = tf.keras.layers.Conv2D(1024, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(cm)

#Expansive path, decoder

u6 = tf.keras.layers.Conv2DTranspose(512, (2, 2), strides=(2, 2),␣
 ↪padding='same')(cm)
u6 = tf.keras.layers.concatenate([u6, c6])
cu6 = tf.keras.layers.Conv2D(512, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(u6)
cu6 = tf.keras.layers.Dropout(dout)(cu6)
cu6 = tf.keras.layers.Conv2D(512, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(cu6)
```

```
u5 = tf.keras.layers.Conv2DTranspose(256, (2, 2), strides=(2, 2),␣
 ↪padding='same')(cu6)
u5 = tf.keras.layers.concatenate([u5, c5])
cu5 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(u5)
cu5 = tf.keras.layers.Dropout(dout)(cu5)
cu5 = tf.keras.layers.Conv2D(256, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(cu5)

u4 = tf.keras.layers.Conv2DTranspose(128, (2, 2), strides=(2, 2),␣
 ↪padding='same')(cu5)
u4 = tf.keras.layers.concatenate([u4, c4])
cu4 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(u4)
cu4 = tf.keras.layers.Dropout(dout)(cu4)
cu4 = tf.keras.layers.Conv2D(128, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(cu4)

u3 = tf.keras.layers.Conv2DTranspose(64, (2, 2), strides=(2, 2),␣
 ↪padding='same')(cu4)
u3 = tf.keras.layers.concatenate([u3, c3])
cu3 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(u3)
cu3 = tf.keras.layers.Dropout(dout)(cu3)
cu3 = tf.keras.layers.Conv2D(64, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(cu3)

u2 = tf.keras.layers.Conv2DTranspose(32, (2, 2), strides=(2, 2),␣
 ↪padding='same')(cu3)
u2 = tf.keras.layers.concatenate([u2, c2])
cu2 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(u2)
cu2 = tf.keras.layers.Dropout(dout)(cu2)
cu2 = tf.keras.layers.Conv2D(32, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(cu2)

u1 = tf.keras.layers.Conv2DTranspose(16, (2, 2), strides=(2, 2),␣
 ↪padding='same')(cu2)
u1 = tf.keras.layers.concatenate([u1, c1], axis=3)
cu1 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(u1)
cu1 = tf.keras.layers.Dropout(dout)(cu1)
cu1 = tf.keras.layers.Conv2D(16, (3, 3), activation='relu',␣
 ↪kernel_initializer='he_normal', padding='same')(cu1)
```

```
outputs = tf.keras.layers.Conv2D(1, (1, 1), activation='sigmoid')(cu1)

model_Acc = tf.keras.Model(inputs=[inputs], outputs=[outputs])
model_IoU = tf.keras.Model(inputs=[inputs], outputs=[outputs])
model_FL = tf.keras.Model(inputs=[inputs], outputs=[outputs])


model_Acc.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 4e-4),␣
 ↪loss='binary_crossentropy', metrics=['accuracy'])

model_IoU.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 6e-4),␣
 ↪loss='binary_crossentropy', metrics=[tf.keras.metrics.MeanIoU(num_classes=2)])

#### Using focal loss function
model_FL.compile(optimizer=tf.keras.optimizers.Adam(learning_rate = 5e-4),␣
 ↪loss=tfa.losses.SigmoidFocalCrossEntropy(alpha=0.2, gamma=2), metrics=[tf.
 ↪keras.metrics.MeanIoU(num_classes=2)])



#model.summary()
```

```
[ ]: #Focal Loss
#Modelcheckpoint
callbacks_FL = [
        tf.keras.callbacks.EarlyStopping(patience=6, monitor='val_loss'),
        tf.keras.callbacks.TensorBoard(log_dir='drive/MyDrive/Colab Notebooks/
 ↪Pics/Aug/FocalLosslogs2'),
        tf.keras.callbacks.ModelCheckpoint("drive/MyDrive/Colab Notebooks/Pics/
 ↪Aug/FocalLoss2.h5", save_best_only=True)]

results_FL = model_Acc.fit(x_train, y_train, validation_split=0.2, batch_size=8,␣
 ↪epochs=70, shuffle=True, callbacks=callbacks_FL)
```

```
Epoch 1/70
29/29 [==============================] - 266s 9s/step - loss: 0.6365 - accuracy:
0.7382 - val_loss: 0.5351 - val_accuracy: 0.7900
Epoch 2/70
29/29 [==============================] - 267s 9s/step - loss: 0.4904 - accuracy:
0.7895 - val_loss: 0.4689 - val_accuracy: 0.7900
Epoch 3/70
29/29 [==============================] - 263s 9s/step - loss: 0.3949 - accuracy:
0.7896 - val_loss: 0.3454 - val_accuracy: 0.7900
Epoch 4/70
29/29 [==============================] - 261s 9s/step - loss: 0.3460 - accuracy:
0.7912 - val_loss: 0.3360 - val_accuracy: 0.7908
Epoch 5/70
29/29 [==============================] - 268s 9s/step - loss: 0.3349 - accuracy:
```

```
0.7977 - val_loss: 0.3242 - val_accuracy: 0.8073
Epoch 6/70
29/29 [==============================] - 261s 9s/step - loss: 0.3244 - accuracy:
0.8118 - val_loss: 0.3283 - val_accuracy: 0.8206
Epoch 7/70
29/29 [==============================] - 257s 9s/step - loss: 0.3036 - accuracy:
0.8255 - val_loss: 0.3093 - val_accuracy: 0.8282
Epoch 8/70
29/29 [==============================] - 264s 9s/step - loss: 0.2826 - accuracy:
0.8411 - val_loss: 0.2798 - val_accuracy: 0.8387
Epoch 9/70
29/29 [==============================] - 257s 9s/step - loss: 0.2616 - accuracy:
0.8516 - val_loss: 0.2720 - val_accuracy: 0.8443
Epoch 10/70
29/29 [==============================] - 266s 9s/step - loss: 0.2468 - accuracy:
0.8600 - val_loss: 0.2618 - val_accuracy: 0.8488
Epoch 11/70
29/29 [==============================] - 264s 9s/step - loss: 0.2222 - accuracy:
0.8663 - val_loss: 0.2397 - val_accuracy: 0.8566
Epoch 12/70
29/29 [==============================] - 267s 9s/step - loss: 0.2146 - accuracy:
0.8703 - val_loss: 0.2192 - val_accuracy: 0.8638
Epoch 13/70
29/29 [==============================] - 263s 9s/step - loss: 0.1911 - accuracy:
0.8780 - val_loss: 0.2042 - val_accuracy: 0.8715
Epoch 14/70
29/29 [==============================] - 260s 9s/step - loss: 0.1775 - accuracy:
0.8835 - val_loss: 0.2310 - val_accuracy: 0.8589
Epoch 15/70
29/29 [==============================] - 260s 9s/step - loss: 0.1683 - accuracy:
0.8875 - val_loss: 0.1856 - val_accuracy: 0.8743
Epoch 16/70
29/29 [==============================] - 262s 9s/step - loss: 0.1589 - accuracy:
0.8906 - val_loss: 0.2384 - val_accuracy: 0.8575
Epoch 17/70
29/29 [==============================] - 265s 9s/step - loss: 0.1500 - accuracy:
0.8935 - val_loss: 0.2193 - val_accuracy: 0.8640
Epoch 18/70
29/29 [==============================] - 253s 9s/step - loss: 0.1371 - accuracy:
0.8982 - val_loss: 0.1838 - val_accuracy: 0.8762
Epoch 19/70
29/29 [==============================] - 263s 9s/step - loss: 0.1363 - accuracy:
0.8987 - val_loss: 0.1730 - val_accuracy: 0.8836
Epoch 20/70
29/29 [==============================] - 253s 9s/step - loss: 0.1310 - accuracy:
0.9003 - val_loss: 0.1902 - val_accuracy: 0.8756
Epoch 21/70
29/29 [==============================] - 258s 9s/step - loss: 0.1190 - accuracy:
```

```
0.9039 - val_loss: 0.1885 - val_accuracy: 0.8763
Epoch 22/70
29/29 [==============================] - 253s 9s/step - loss: 0.1176 - accuracy:
0.9044 - val_loss: 0.1948 - val_accuracy: 0.8747
Epoch 23/70
29/29 [==============================] - 255s 9s/step - loss: 0.1112 - accuracy:
0.9066 - val_loss: 0.1589 - val_accuracy: 0.8905
Epoch 24/70
29/29 [==============================] - 253s 9s/step - loss: 0.1037 - accuracy:
0.9091 - val_loss: 0.1688 - val_accuracy: 0.8842
Epoch 25/70
29/29 [==============================] - 255s 9s/step - loss: 0.1000 - accuracy:
0.9101 - val_loss: 0.1765 - val_accuracy: 0.8849
Epoch 26/70
29/29 [==============================] - 256s 9s/step - loss: 0.0977 - accuracy:
0.9106 - val_loss: 0.1743 - val_accuracy: 0.8863
Epoch 27/70
29/29 [==============================] - 256s 9s/step - loss: 0.0900 - accuracy:
0.9133 - val_loss: 0.1544 - val_accuracy: 0.8926
Epoch 28/70
29/29 [==============================] - 256s 9s/step - loss: 0.0879 - accuracy:
0.9139 - val_loss: 0.1665 - val_accuracy: 0.8868
Epoch 29/70
29/29 [==============================] - 253s 9s/step - loss: 0.0877 - accuracy:
0.9139 - val_loss: 0.1354 - val_accuracy: 0.8996
Epoch 30/70
29/29 [==============================] - 258s 9s/step - loss: 0.0820 - accuracy:
0.9158 - val_loss: 0.1391 - val_accuracy: 0.8985
Epoch 31/70
29/29 [==============================] - 245s 8s/step - loss: 0.0853 - accuracy:
0.9148 - val_loss: 0.1988 - val_accuracy: 0.8773
Epoch 32/70
29/29 [==============================] - 243s 8s/step - loss: 0.0830 - accuracy:
0.9154 - val_loss: 0.1770 - val_accuracy: 0.8856
Epoch 33/70
29/29 [==============================] - 251s 9s/step - loss: 0.0905 - accuracy:
0.9134 - val_loss: 0.1345 - val_accuracy: 0.8976
Epoch 34/70
29/29 [==============================] - 258s 9s/step - loss: 0.0738 - accuracy:
0.9184 - val_loss: 0.1662 - val_accuracy: 0.8914
Epoch 35/70
29/29 [==============================] - 254s 9s/step - loss: 0.0694 - accuracy:
0.9196 - val_loss: 0.1537 - val_accuracy: 0.8970
Epoch 36/70
29/29 [==============================] - 258s 9s/step - loss: 0.0689 - accuracy:
0.9197 - val_loss: 0.1408 - val_accuracy: 0.9017
Epoch 37/70
29/29 [==============================] - 251s 9s/step - loss: 0.0774 - accuracy:
```

```
0.9183 - val_loss: 0.1799 - val_accuracy: 0.8870
Epoch 38/70
29/29 [==============================] - 253s 9s/step - loss: 0.1423 - accuracy:
0.8973 - val_loss: 0.2013 - val_accuracy: 0.8701
Epoch 39/70
29/29 [==============================] - 247s 9s/step - loss: 0.1037 - accuracy:
0.9097 - val_loss: 0.1493 - val_accuracy: 0.8904
```

- Metrics=Mean_IoU

```
[ ]: #IoU
     #Modelcheckpoint
     callbacks_IoU = [
             tf.keras.callbacks.EarlyStopping(patience=6, monitor='val_loss'),
             tf.keras.callbacks.TensorBoard(log_dir='drive/MyDrive/Colab Notebooks/
      ↪Pics/Aug/IoUlogs'),
             tf.keras.callbacks.ModelCheckpoint("drive/MyDrive/Colab Notebooks/Pics/
      ↪Aug/IoU.h5", save_best_only=True)]


     results_IoU = model_IoU.fit(x_train, y_train, validation_split=0.2,␣
      ↪batch_size=8, epochs=70, shuffle=True, callbacks=callbacks_IoU)
```

```
Epoch 1/70
29/29 [==============================] - 213s 7s/step - loss: 0.6046 -
mean_io_u_2: 0.4295 - val_loss: 0.5077 - val_mean_io_u_2: 0.4306
Epoch 2/70
29/29 [==============================] - 210s 7s/step - loss: 0.5067 -
mean_io_u_2: 0.4295 - val_loss: 0.4782 - val_mean_io_u_2: 0.4306
Epoch 3/70
29/29 [==============================] - 210s 7s/step - loss: 0.4617 -
mean_io_u_2: 0.4295 - val_loss: 0.4143 - val_mean_io_u_2: 0.4306
Epoch 4/70
29/29 [==============================] - 209s 7s/step - loss: 0.3947 -
mean_io_u_2: 0.4295 - val_loss: 0.3795 - val_mean_io_u_2: 0.4306
Epoch 5/70
29/29 [==============================] - 209s 7s/step - loss: 0.3739 -
mean_io_u_2: 0.4295 - val_loss: 0.3951 - val_mean_io_u_2: 0.4306
Epoch 6/70
29/29 [==============================] - 209s 7s/step - loss: 0.3528 -
mean_io_u_2: 0.4295 - val_loss: 0.3470 - val_mean_io_u_2: 0.4306
Epoch 7/70
29/29 [==============================] - 209s 7s/step - loss: 0.3302 -
mean_io_u_2: 0.4295 - val_loss: 0.3401 - val_mean_io_u_2: 0.4306
Epoch 8/70
29/29 [==============================] - 200s 7s/step - loss: 0.3103 -
mean_io_u_2: 0.4295 - val_loss: 0.3059 - val_mean_io_u_2: 0.4306
Epoch 9/70
```

```
29/29 [==============================] - 200s 7s/step - loss: 0.2869 -
mean_io_u_2: 0.4295 - val_loss: 0.2798 - val_mean_io_u_2: 0.4306
Epoch 10/70
29/29 [==============================] - 209s 7s/step - loss: 0.2691 -
mean_io_u_2: 0.4295 - val_loss: 0.2755 - val_mean_io_u_2: 0.4306
Epoch 11/70
29/29 [==============================] - 209s 7s/step - loss: 0.2462 -
mean_io_u_2: 0.4297 - val_loss: 0.2492 - val_mean_io_u_2: 0.4306
Epoch 12/70
29/29 [==============================] - 209s 7s/step - loss: 0.2463 -
mean_io_u_2: 0.4297 - val_loss: 0.3306 - val_mean_io_u_2: 0.4306
Epoch 13/70
29/29 [==============================] - 210s 7s/step - loss: 0.2263 -
mean_io_u_2: 0.4296 - val_loss: 0.2646 - val_mean_io_u_2: 0.4306
Epoch 14/70
29/29 [==============================] - 211s 7s/step - loss: 0.2198 -
mean_io_u_2: 0.4298 - val_loss: 0.2683 - val_mean_io_u_2: 0.4306
Epoch 15/70
29/29 [==============================] - 212s 7s/step - loss: 0.2150 -
mean_io_u_2: 0.4297 - val_loss: 0.2741 - val_mean_io_u_2: 0.4306
Epoch 16/70
29/29 [==============================] - 210s 7s/step - loss: 0.1918 -
mean_io_u_2: 0.4307 - val_loss: 0.2650 - val_mean_io_u_2: 0.4306
Epoch 17/70
29/29 [==============================] - 210s 7s/step - loss: 0.1797 -
mean_io_u_2: 0.4324 - val_loss: 0.3125 - val_mean_io_u_2: 0.4306
```

- Metrics=Accuracy

```
[ ]: #Accuracy
     #Modelcheckpoint
     callbacks_Acc = [
             tf.keras.callbacks.EarlyStopping(patience=6, monitor='val_loss'),
             tf.keras.callbacks.TensorBoard(log_dir='drive/MyDrive/Colab Notebooks/
      ↪Pics/Aug/Acclogs'),
             tf.keras.callbacks.ModelCheckpoint("drive/MyDrive/Colab Notebooks/Pics/
      ↪Aug/Acc.h5", save_best_only=True)]

     results_Acc = model_Acc.fit(x_train, y_train, validation_split=0.2,␣
      ↪batch_size=8, epochs=70, shuffle=True, callbacks=callbacks_Acc)
```

```
Epoch 1/70
29/29 [==============================] - 230s 8s/step - loss: 0.5644 - accuracy:
0.7295 - val_loss: 0.5296 - val_accuracy: 0.7900
Epoch 2/70
29/29 [==============================] - 238s 8s/step - loss: 0.4849 - accuracy:
0.7894 - val_loss: 0.4945 - val_accuracy: 0.7900
Epoch 3/70
```

```
29/29 [==============================] - 235s 8s/step - loss: 0.4376 - accuracy:
0.7895 - val_loss: 0.4377 - val_accuracy: 0.7900
Epoch 4/70
29/29 [==============================] - 244s 8s/step - loss: 0.3809 - accuracy:
0.7895 - val_loss: 0.3470 - val_accuracy: 0.7900
Epoch 5/70
29/29 [==============================] - 247s 9s/step - loss: 0.3420 - accuracy:
0.7895 - val_loss: 0.3459 - val_accuracy: 0.7900
Epoch 6/70
29/29 [==============================] - 248s 9s/step - loss: 0.3179 - accuracy:
0.7914 - val_loss: 0.2954 - val_accuracy: 0.8035
Epoch 7/70
29/29 [==============================] - 239s 8s/step - loss: 0.3193 - accuracy:
0.8070 - val_loss: 0.2950 - val_accuracy: 0.8390
Epoch 8/70
29/29 [==============================] - 244s 8s/step - loss: 0.2875 - accuracy:
0.8429 - val_loss: 0.2828 - val_accuracy: 0.8529
Epoch 9/70
29/29 [==============================] - 239s 8s/step - loss: 0.2468 - accuracy:
0.8630 - val_loss: 0.2461 - val_accuracy: 0.8677
Epoch 10/70
29/29 [==============================] - 247s 9s/step - loss: 0.2204 - accuracy:
0.8703 - val_loss: 0.2401 - val_accuracy: 0.8654
Epoch 11/70
29/29 [==============================] - 241s 8s/step - loss: 0.2093 - accuracy:
0.8739 - val_loss: 0.2466 - val_accuracy: 0.8631
Epoch 12/70
29/29 [==============================] - 239s 8s/step - loss: 0.2046 - accuracy:
0.8752 - val_loss: 0.2473 - val_accuracy: 0.8589
Epoch 13/70
29/29 [==============================] - 228s 8s/step - loss: 0.1954 - accuracy:
0.8779 - val_loss: 0.2199 - val_accuracy: 0.8600
Epoch 14/70
29/29 [==============================] - 232s 8s/step - loss: 0.1693 - accuracy:
0.8859 - val_loss: 0.1904 - val_accuracy: 0.8729
Epoch 15/70
29/29 [==============================] - 238s 8s/step - loss: 0.1694 - accuracy:
0.8879 - val_loss: 0.2227 - val_accuracy: 0.8581
Epoch 16/70
29/29 [==============================] - 223s 8s/step - loss: 0.1537 - accuracy:
0.8916 - val_loss: 0.1826 - val_accuracy: 0.8771
Epoch 17/70
29/29 [==============================] - 236s 8s/step - loss: 0.1345 - accuracy:
0.8994 - val_loss: 0.1603 - val_accuracy: 0.8852
Epoch 18/70
29/29 [==============================] - 228s 8s/step - loss: 0.1243 - accuracy:
0.9025 - val_loss: 0.1842 - val_accuracy: 0.8773
Epoch 19/70
```

```
29/29 [==============================] - 236s 8s/step - loss: 0.1270 - accuracy:
0.9014 - val_loss: 0.1462 - val_accuracy: 0.8919
Epoch 20/70
29/29 [==============================] - 237s 8s/step - loss: 0.1194 - accuracy:
0.9039 - val_loss: 0.2139 - val_accuracy: 0.8641
Epoch 21/70
29/29 [==============================] - 233s 8s/step - loss: 0.1190 - accuracy:
0.9039 - val_loss: 0.2242 - val_accuracy: 0.8596
Epoch 22/70
29/29 [==============================] - 226s 8s/step - loss: 0.1138 - accuracy:
0.9064 - val_loss: 0.1671 - val_accuracy: 0.8886
Epoch 23/70
29/29 [==============================] - 233s 8s/step - loss: 0.1067 - accuracy:
0.9083 - val_loss: 0.1539 - val_accuracy: 0.8925
Epoch 24/70
29/29 [==============================] - 234s 8s/step - loss: 0.1050 - accuracy:
0.9096 - val_loss: 0.1512 - val_accuracy: 0.8895
Epoch 25/70
29/29 [==============================] - 240s 8s/step - loss: 0.1002 - accuracy:
0.9101 - val_loss: 0.1484 - val_accuracy: 0.8924
```

[ ]:
```
print(results_IoU.history.keys())
```

```
dict_keys(['loss', 'mean_io_u', 'val_loss', 'val_mean_io_u'])
```

2 Results

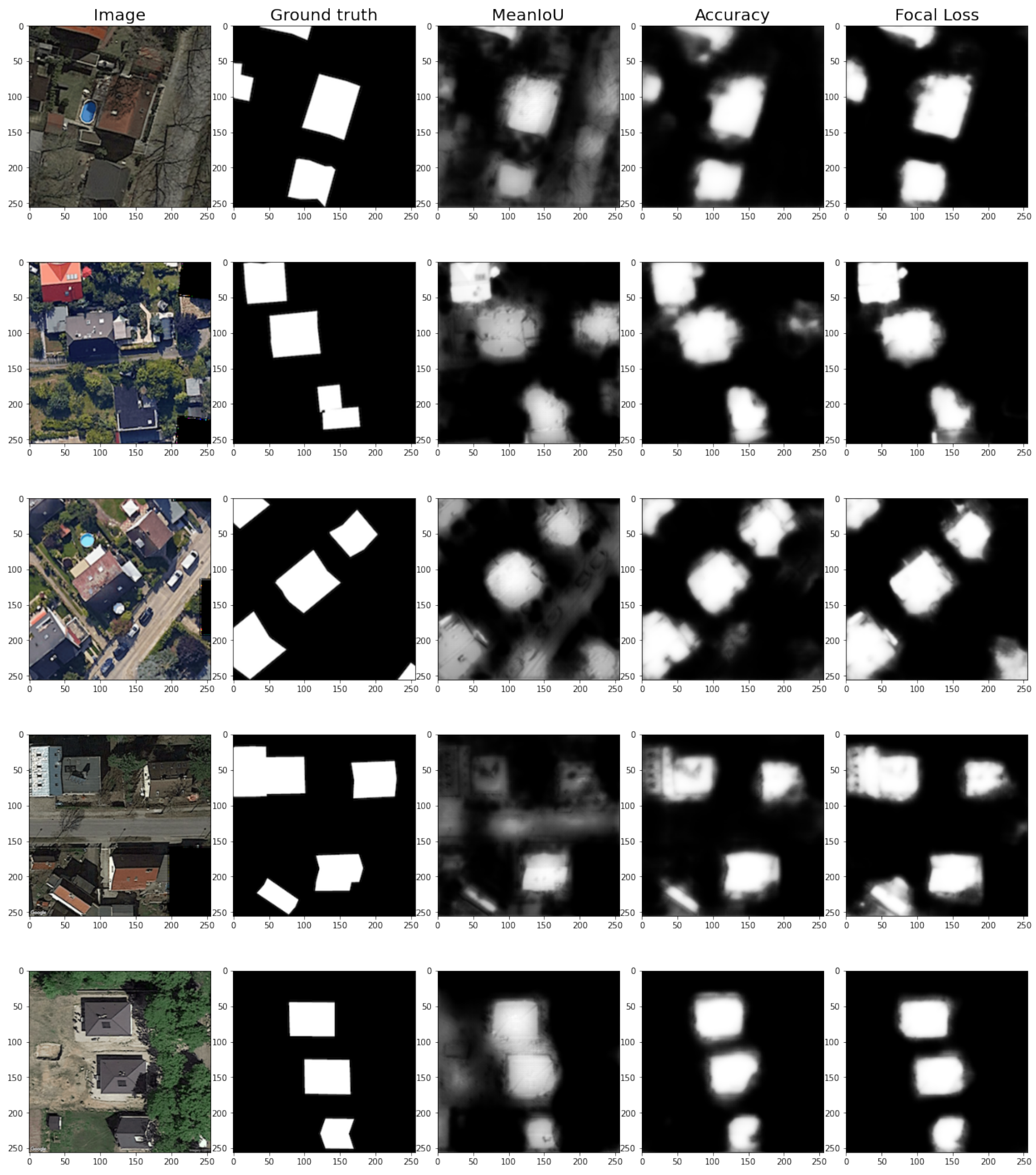
- Loading the trained models

[12]:
```
# IoU
model_IoU.load_weights('drive/MyDrive/Colab Notebooks/Pics/Aug/IoU.h5')
y_pred_IoU = model_IoU.predict(x_val)
# Acc
model_Acc.load_weights('drive/MyDrive/Colab Notebooks/Pics/Aug/Acc.h5')
y_pred_Acc = model_Acc.predict(x_val)
#FL
model_FL.load_weights('drive/MyDrive/Colab Notebooks/Pics/Aug/FocalLoss2.h5')
y_pred_FL = model_FL.predict(x_val)
```

[ ]:
```
#Defining a threshold for better visualization of the predicted outputs.
#pred_fI = (y_pred_IoU > 0.2).astype(np.uint8)
#pred_fA = (y_pred_Acc > 0.3).astype(np.uint8)
```

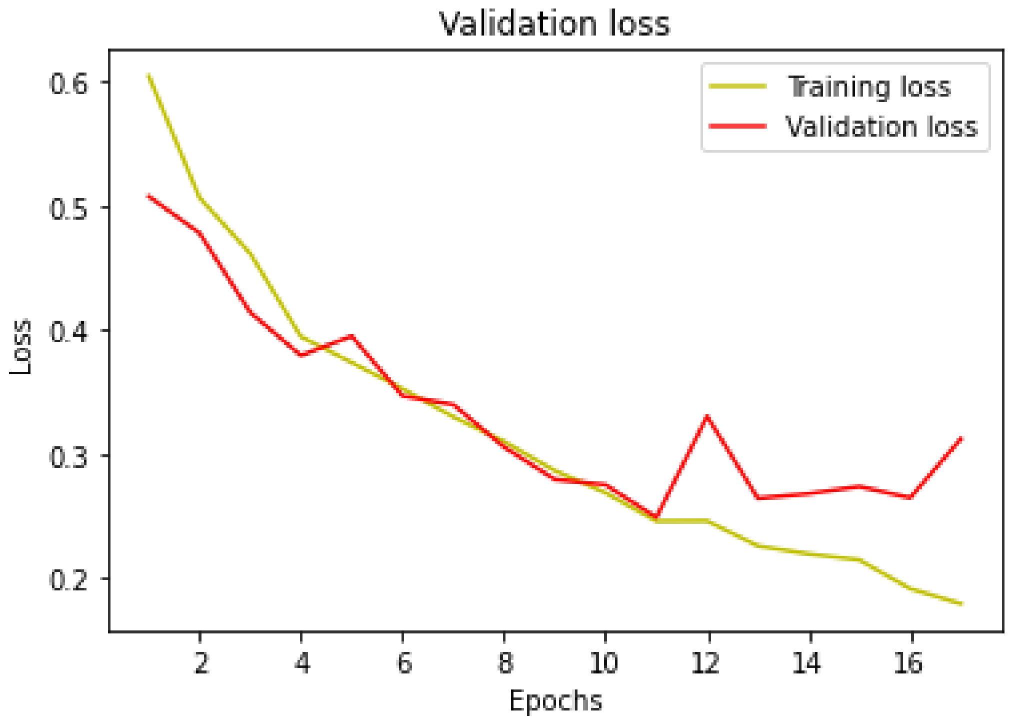
- Comparing the results of the three models, metrics= Mean_IoU and Accuracy besides focal loss, with the ground truth

[17]:
```
ids = random.sample(range(len(x_val)-1), 5)

f, axarr = plt.subplots(5,5,figsize=(25, 25))
plt.subplots_adjust(wspace=-0.4, hspace=0.3)

for j in range(5):
   axarr[j,0].imshow(x_val[ids[j]])
   axarr[j,1].imshow(y_val[ids[j]].squeeze(axis=2), cmap='gray')
   axarr[j,2].imshow(y_pred_IoU[ids[j]].squeeze(axis=2), cmap='gray')
   axarr[j,3].imshow(y_pred_Acc[ids[j]].squeeze(axis=2), cmap='gray')
   axarr[j,4].imshow(y_pred_FL[ids[j]].squeeze(axis=2), cmap='gray')
   if j==0:
     axarr[0,0].set_title('Image', fontsize=20)
     axarr[0,1].set_title('Ground truth', fontsize=20)
     axarr[0,2].set_title('MeanIoU', fontsize=20)
     axarr[0,3].set_title('Accuracy', fontsize=20)
     axarr[0,4].set_title('Focal Loss', fontsize=20)
```

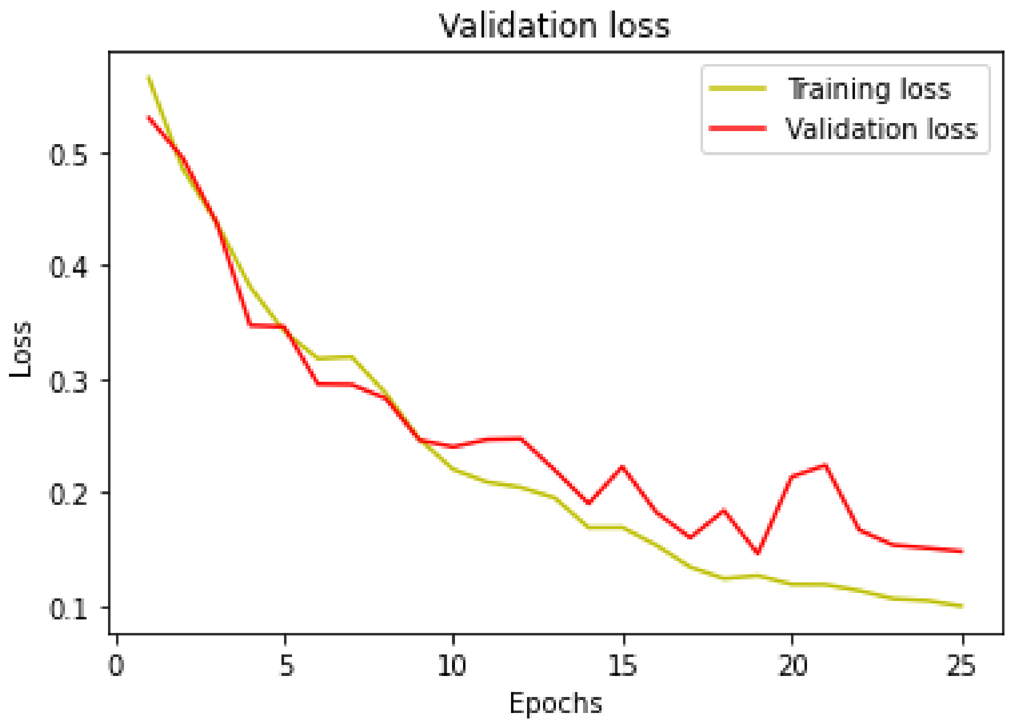


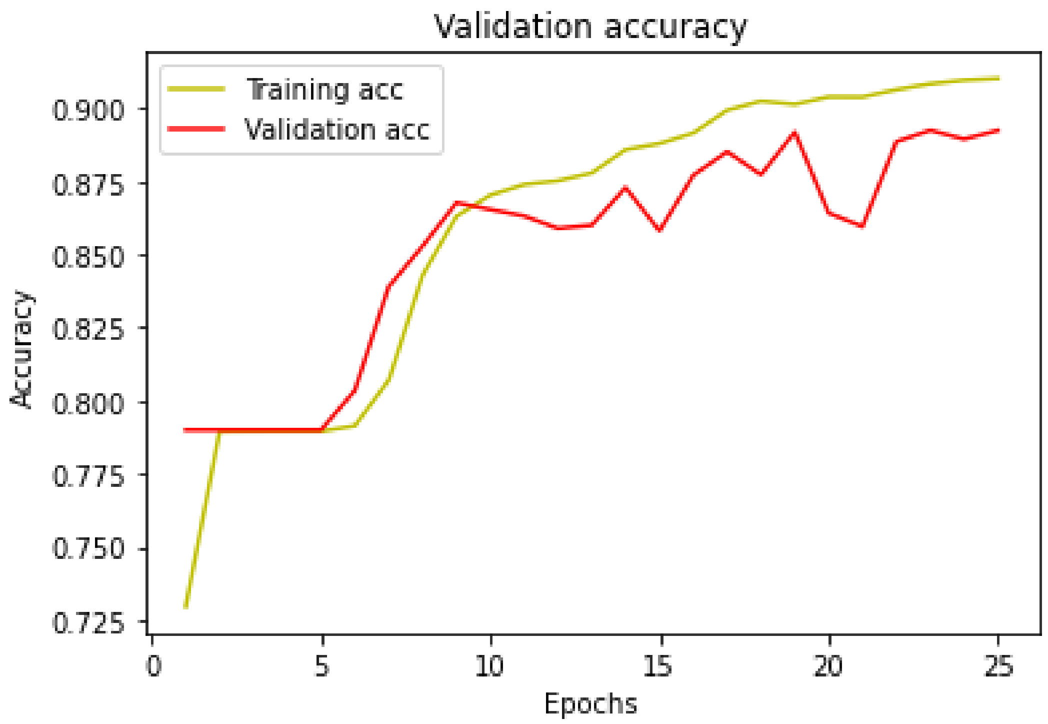
```
[ ]:
```

```
[ ]: #Plots of the loss and accuracy of the trained networks
     #IoU
     loss = results_IoU.history['loss']
     val_loss = results_IoU.history['val_loss']
     epochs = range(1, len(loss) + 1)
     plt.plot(epochs, loss, 'y', label='Training loss')
     plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

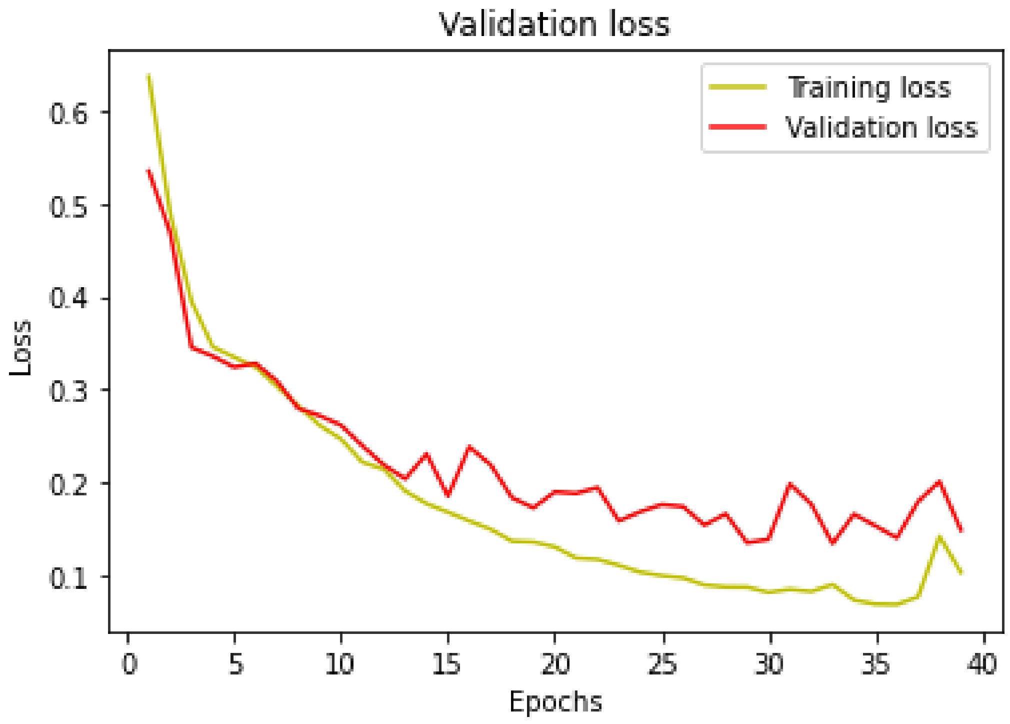


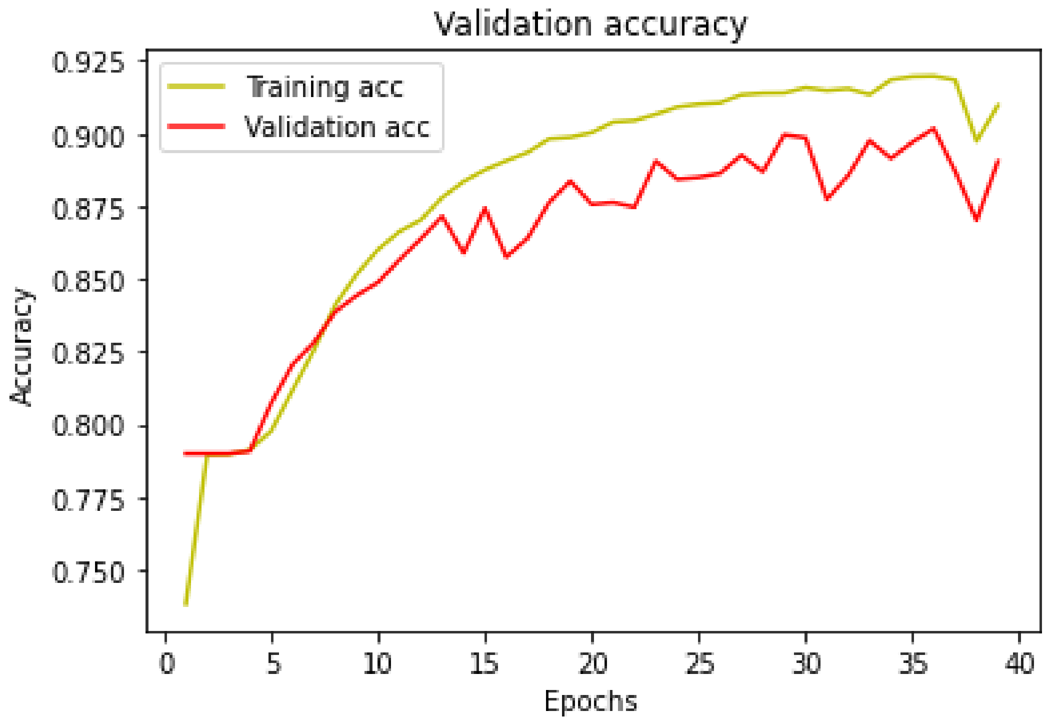
[ ]:
```
#Accuracy
loss = results_Acc.history['loss']
val_loss = results_Acc.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

acc = results_Acc.history['accuracy']
val_acc = results_Acc.history['val_accuracy']
plt.plot(epochs, acc, 'y', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```





```
[ ]: #Focal Loss
     loss = results_FL.history['loss']
     val_loss = results_FL.history['val_loss']
     epochs = range(1, len(loss) + 1)
     plt.plot(epochs, loss, 'y', label='Training loss')
     plt.plot(epochs, val_loss, 'r', label='Validation loss')
     plt.title('Validation loss')
     plt.xlabel('Epochs')
     plt.ylabel('Loss')
     plt.legend()
     plt.show()

     acc = results_FL.history['accuracy']
     val_acc = results_FL.history['val_accuracy']
     plt.plot(epochs, acc, 'y', label='Training acc')
     plt.plot(epochs, val_acc, 'r', label='Validation acc')
     plt.title('Validation accuracy')
     plt.xlabel('Epochs')
     plt.ylabel('Accuracy')
     plt.legend()
     plt.show()
```


Validation loss
Training loss
Validation loss
Loss
Epochs
0.1
0.2
0.3
0.4
0.5
0.6
0
5
10
15
20
25
30
35
40


Validation accuracy
Training acc
Validation acc
Accuracy
Epochs
0.750
0.775
0.800
0.825
0.850
0.875
0.900
0.925
0
5
10
15
20
25
30
35
40

- Trying the trained networks on the test images, without ground truth (5 images)

```
[ ]: #Check with test images, without having the ground truth
     test_images = []

     for directory_path in glob.glob("/content/drive/MyDrive/Colab Notebooks/Pics/Aug/
      ↪test"):
         for img_path in glob.glob(os.path.join(directory_path, "*.png")):
             #print(img_path)
             img = cv2.imread(img_path, cv2.IMREAD_COLOR)
             img = cv2.resize(img, (IMG_HEIGHT, IMG_WIDTH))
             img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
             test_images.append(img)
             #train_labels.append(label)
     #Convert list to array for machine learning processing
     test_images = np.array(test_images)
```

```
[ ]: xt = np.asarray(test_images, dtype=np.float32)/255
```

```
[ ]: xt = xt.reshape(xt.shape[0], xt.shape[1], xt.shape[2], 3)
```

```
[ ]: model_IoU.load_weights('drive/MyDrive/Colab Notebooks/Pics/Aug/IoU.h5')
     pred_IoUt = model_IoU.predict(xt)
     model_Acc.load_weights('drive/MyDrive/Colab Notebooks/Pics/Aug/Acc.h5')
     pred_Acct = model_Acc.predict(xt)
     model_FL.load_weights('drive/MyDrive/Colab Notebooks/Pics/Aug/FocalLoss2.h5')
     pred_FLt = model_FL.predict(xt)
```

```
[ ]: f, axarr = plt.subplots(5,4,figsize=(25, 25))
     plt.subplots_adjust(wspace=-0.6, hspace=0.3)

     for j in range(5):
        axarr[j,0].imshow(xt[j])
        axarr[j,1].imshow(pred_IoUt[j].squeeze(axis=2), cmap='gray')
        axarr[j,2].imshow(pred_Acct[j].squeeze(axis=2), cmap='gray')
        axarr[j,3].imshow(pred_FLt[j].squeeze(axis=2), cmap='gray')
        if j==0:
          axarr[0,0].set_title('Image', fontsize=20)
          axarr[0,1].set_title('MeanIoU', fontsize=20)
          axarr[0,2].set_title('Accuracy', fontsize=20)
          axarr[0,3].set_title('Focal Loss', fontsize=20)
```

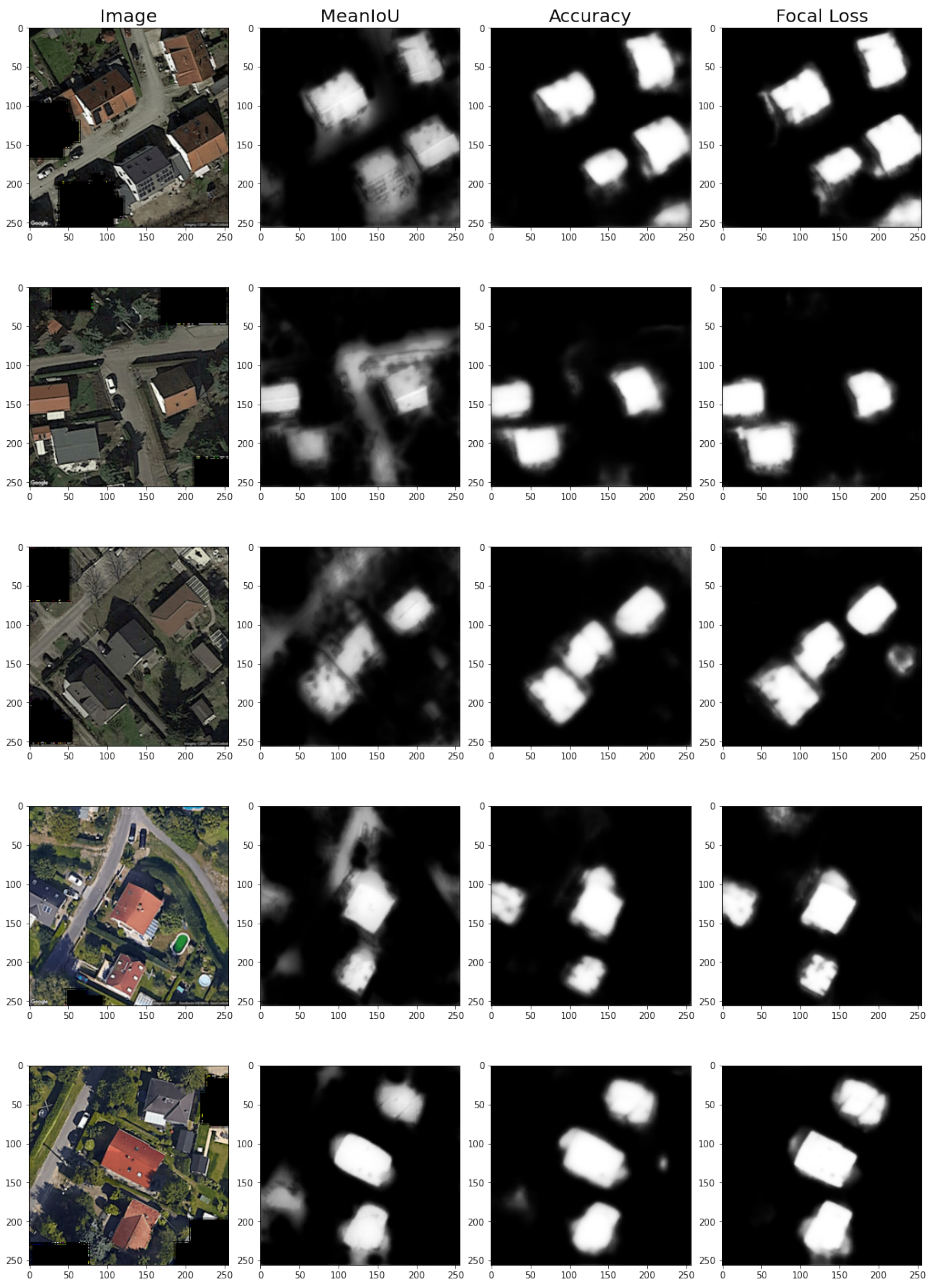


Image
MeanIoU
Accuracy
Focal Loss