In [1]:

```
%load_ext autoreload
%autoreload 2
```

In [2]:

```
#%autoreload # When utils.py is updated
from utils_unet_resunet import *
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from model.models import Model_3
from model.losses import WBCE
root_path = 'imgs/'
```

In [3]:

```
# Define data type
img_type = 'FUSION'

if img_type == 'FUSION':
    image_array = np.load(root_path+'New_Images/fus_stack.npy')


if img_type == 'OPT':
    image_array = np.load(root_path+'New_Images/opt_stack.npy')


if img_type == 'SAR':
    image_array = np.load(root_path+'New_Images/sar_stack.npy')
print('Image stack:', image_array.shape)

final_mask1 = np.load(root_path+'New_Images/'+'final_mask1.npy')
print('Labels stack:', final_mask1.shape)

h_, w_, channels = image_array.shape
n_opt_layers = 20
```

```
Image stack: (10000, 7000, 24)
Labels stack: (10000, 7000)
```

In [4]:

```
# Create tile mask
mask_tiles = create_mask(final_mask1.shape[0], final_mask1.shape[1], grid_size=(5, 4))
image_array = image_array[:mask_tiles.shape[0], :mask_tiles.shape[1],:]
final_mask1 = final_mask1[:mask_tiles.shape[0], :mask_tiles.shape[1]]

print('mask: ',mask_tiles.shape)
print('image stack: ', image_array.shape)
print('ref :', final_mask1.shape)
#plt.imshow(mask_tiles)
```

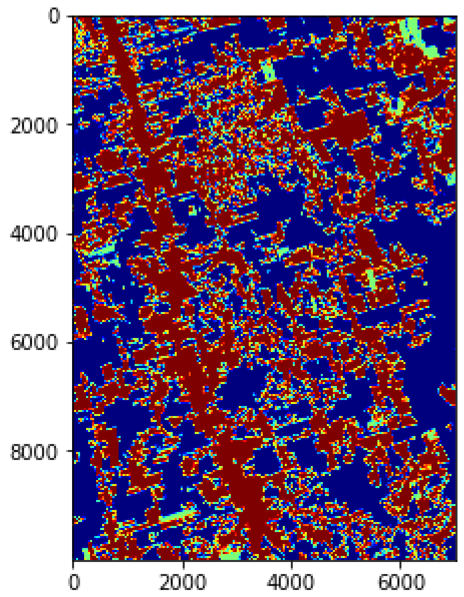
```
Tiles size:  2000 1750
Mask size:  (10000, 7000)
mask:  (10000, 7000)
image stack:  (10000, 7000, 24)
ref : (10000, 7000)
```

In [5]:

```
plt.figure(figsize=(10,5))
plt.imshow(final_mask1, cmap = 'jet')
```

Out[5]:

```
<matplotlib.image.AxesImage at 0x2c6748581f0>
```



In [6]:

```
# Define tiles for training, validation, and test sets
tiles_tr = [1,3,5,8,11,13,14,20]
tiles_val = [6,19]
tiles_ts = (list(set(np.arange(20)+1)-set(tiles_tr)-set(tiles_val)))

mask_tr_val = np.zeros((mask_tiles.shape)).astype('float32')
# Training and validation mask
for tr_ in tiles_tr:
    mask_tr_val[mask_tiles == tr_] = 1

for val_ in tiles_val:
    mask_tr_val[mask_tiles == val_] = 2

mask_amazon_ts = np.zeros((mask_tiles.shape)).astype('float32')
for ts_ in tiles_ts:
    mask_amazon_ts[mask_tiles == ts_] = 1
```

In [7]:

```
# Create ixd image to extract patches
overlap = 0.7
patch_size = 128
batch_size = 32
im_idx = create_idx_image(final_mask1)
patches_idx = extract_patches(im_idx, patch_size=(patch_size, patch_size), overlap=over
patches_mask = extract_patches(mask_tr_val, patch_size=(patch_size, patch_size), overla
del im_idx
```

In [8]:

```
# Selecting index trn val and test patches idx
idx_trn = np.squeeze(np.where(patches_mask.sum(axis=(1, 2))==patch_size**2))
idx_val = np.squeeze(np.where(patches_mask.sum(axis=(1, 2))==2*patch_size**2))
del patches_mask

patches_idx_trn = patches_idx[idx_trn]
patches_idx_val = patches_idx[idx_val]
del idx_trn, idx_val

print('Number of training patches:  ', len(patches_idx_trn), 'Number of validation patc
```

```
Number of training patches:   17110 Number of validation patches 4116
```

In [9]:

```
# Extract patches with at least 2% of deforestation class
X_train = retrieve_idx_percentage(final_mask1, patches_idx_trn, patch_size, pertentage
X_valid = retrieve_idx_percentage(final_mask1, patches_idx_val, patch_size, pertentage
print(X_train.shape, X_valid.shape)
del patches_idx_trn, patches_idx_val
```

```
(1158, 128, 128) (341, 128, 128)
```

In [10]:

```
def batch_generator(batches, image, reference, target_size, number_class):
    """Take as input a Keras ImageGen (Iterator) and generate random
    crops from the image batches generated by the original iterator.
    """
    image = image.reshape(-1, image.shape[-1])
    reference = reference.reshape(final_mask1.shape[0]*final_mask1.shape[1])
    while True:
        batch_x, batch_y = next(batches)
        batch_x = np.squeeze(batch_x.astype('int64'))
        #print(batch_x.shape)
        batch_img = np.zeros((batch_x.shape[0], target_size, target_size, image.shape[-
        batch_ref = np.zeros((batch_x.shape[0], target_size, target_size, number_class)

        for i in range(batch_x.shape[0]):
            if np.random.rand()>0.5:
                batch_x[i] = np.rot90(batch_x[i], 1)
            batch_img[i] = image[batch_x[i]]
            batch_ref[i] = tf.keras.utils.to_categorical(reference[batch_x[i]] , number

        yield (batch_img, batch_ref)

train_datagen = ImageDataGenerator(horizontal_flip = True,
                                   vertical_flip = True)
valid_datagen = ImageDataGenerator(horizontal_flip = True,
                                   vertical_flip = True)

y_train = np.zeros((len(X_train)))
y_valid = np.zeros((len(X_valid)))

train_gen = train_datagen.flow(np.expand_dims(X_train, axis = -1), y_train,
                              batch_size=batch_size,
                              shuffle=True)

valid_gen = valid_datagen.flow(np.expand_dims(X_valid, axis = -1), y_valid,
                              batch_size=batch_size,
                              shuffle=False)

number_class = 3
train_gen_crops = batch_generator(train_gen, image_array, final_mask1, patch_size, numb
valid_gen_crops = batch_generator(valid_gen, image_array, final_mask1, patch_size, numb

```

In [11]:

```
exp = 3
path_exp = root_path+'experiments/exp'+str(exp)
path_models = path_exp+'/models'
path_maps = path_exp+'/pred_maps'

if not os.path.exists(path_exp):
    os.makedirs(path_exp)
if not os.path.exists(path_models):
    os.makedirs(path_models)
if not os.path.exists(path_maps):
    os.makedirs(path_maps)
```

In [12]:

```
# Define model
input_shape = (patch_size, patch_size, channels)
nb_filters = [32, 64, 128]

method = 'unet'
if method == 'unet':
   model = build_unet(input_shape, nb_filters, number_class)

if method == 'resunet':
   model = build_resunet(input_shape, nb_filters, number_class)

model = Model_3(nb_filters, number_class, n_opt_layers)
```

In [13]:

```
# Parameters of the model
weights = [0.2, 0.8, 0]
adam = Adam(lr = 1e-3 , beta_1=0.9)
loss = weighted_categorical_crossentropy(weights)
#loss = WBCE(weights = weights)
#loss = WBCE(weights = weights, class_indexes = [0, 1])

```

In [14]:

```
metrics_all = []
times=5
for tm in range(0,times):
    print('time: ', tm)

    rows = patch_size
    cols = patch_size
    adam = Adam(lr = 1e-4 , beta_1=0.9)

    loss = weighted_categorical_crossentropy(weights)
    #loss = WBCE(weights = weights)
    #loss = WBCE(weights = weights, class_indexes = [0, 1])

    #if method == 'unet':
    #   model = build_unet(input_shape, nb_filters, number_class)

    #if method == 'resunet':
    #   model = build_resunet(input_shape, nb_filters, number_class)

    model = Model_3(nb_filters, number_class, n_opt_layers)
    model.build((None,)+input_shape)

    model.compile(optimizer=adam, loss=loss, metrics=['accuracy'])
    model.summary()

    earlystop = EarlyStopping(monitor='val_loss', min_delta=0.0001, patience=10, verbos
    #earlystop = EarlyStopping(monitor='val_loss', min_delta=0.0001, patience=10, verbo
    #checkpoint = ModelCheckpoint(path_models+ '/' + method +'_'+str(tm)+'.h5', monitor
    checkpoint = ModelCheckpoint(path_models+ '/' + method +'_'+str(tm)+'.h5', monitor=
    lr_reduce = ReduceLROnPlateau(factor=0.9, min_delta=0.0001, patience=5, verbose=1)
    callbacks_list = [earlystop, checkpoint]
    # train the model
    start_training = time.time()
    history = model.fit(train_gen_crops,
                              steps_per_epoch=len(X_train)*3//train_gen.batch_size,
                              validation_data=valid_gen_crops,
                              validation_steps=len(X_valid)*3//valid_gen.batch_size,
                              epochs=100,
                              callbacks=callbacks_list)
    end_training = time.time() - start_training
    metrics_all.append(end_training)
    del model, history
```

```
s: 0.1177 - val_sar_loss: 1.1420 - val_fusion_loss: 0.1097 - val_loss: 1.3
695

Epoch 00011: val_loss did not improve from 1.26032
Epoch 12/100
108/108 [==============================] - 19s 179ms/step - opt_accuracy:
0.8897 - sar_accuracy: 0.7763 - fus_accuracy: 0.8726 - opt_loss: 0.0583 -
sar_loss: 1.2103 - fusion_loss: 0.0657 - loss: 1.3343 - val_opt_accuracy:
0.8613 - val_sar_accuracy: 0.7951 - val_fus_accuracy: 0.8606 - val_opt_los
s: 0.1505 - val_sar_loss: 1.1366 - val_fusion_loss: 0.1424 - val_loss: 1.4
296

Epoch 00012: val_loss did not improve from 1.26032
Epoch 13/100
108/108 [==============================] - 19s 178ms/step - opt_accuracy:
```

```
0.8947 - sar_accuracy: 0.7741 - fus_accuracy: 0.8782 - opt_loss: 0.0547 -
sar_loss: 1.2059 - fusion_loss: 0.0615 - loss: 1.3222 - val_opt_accuracy:
0.8601 - val_sar_accuracy: 0.7945 - val_fus_accuracy: 0.8598 - val_opt_los
s: 0.1468 - val_sar_loss: 1.1723 - val_fusion_loss: 0.1403 - val_loss: 1.4
594
```

In [15]:

```
# Test Loop
time_ts = []
n_pool = 3
n_rows = 5
n_cols = 4
rows, cols = image_array.shape[:2]
pad_rows = rows - np.ceil(rows/(n_rows*2**n_pool))*n_rows*2**n_pool
pad_cols = cols - np.ceil(cols/(n_cols*2**n_pool))*n_cols*2**n_pool
print(pad_rows, pad_cols)

npad = ((0, int(abs(pad_rows))), (0, int(abs(pad_cols))), (0, 0))
image1_pad = np.pad(image_array, pad_width=npad, mode='reflect')

h, w, c = image1_pad.shape
patch_size_rows = h//n_rows
patch_size_cols = w//n_cols
num_patches_x = int(h/patch_size_rows)
num_patches_y = int(w/patch_size_cols)

input_shape=(patch_size_rows,patch_size_cols, c)

#if method == 'unet':
#   new_model = build_unet(input_shape, nb_filters, number_class)

#if method == 'resunet':
#   new_model = build_resunet(input_shape, nb_filters, number_class)

new_model = Model_3(nb_filters, number_class, n_opt_layers)
new_model.build((None,)+input_shape)
adam = Adam(lr = 1e-3 , beta_1=0.9)
loss = weighted_categorical_crossentropy(weights)
new_model.compile(optimizer=adam, loss=loss, metrics=['accuracy'], run_eagerly=True)

for tm in range(0,times):
    print('time: ', tm)
    #model = load_model(path_models+ '/' + method +'_'+str(tm)+'.h5', compile=False)

    #for l in range(1, len(model.layers)):
    #    new_model.layers[l].set_weights(model.layers[l].get_weights())
    new_model.load_weights(path_models+ '/' + method +'_'+str(tm)+'.h5')

    start_test = time.time()
    patch_opt = []
    patch_sar = []
    patch_fus = []
    patch_comb = []

    for i in range(0,num_patches_y):
        for j in range(0,num_patches_x):
            patch = image1_pad[patch_size_rows*j:patch_size_rows*(j+1), patch_size_cols
            pred_opt, pred_sar, pred_fus, pred_comb = new_model.predict(np.expand_dims(
            del patch
            patch_opt.append(pred_opt[:,:,:,1])
            patch_sar.append(pred_sar[:,:,:,1])
            patch_fus.append(pred_fus[:,:,:,1])
            patch_comb.append(pred_comb[:,:,:,1])
            del pred_opt, pred_sar, pred_fus, pred_comb
    end_test =  time.time() - start_test

```

```
    patches_pred_opt = np.asarray(patch_opt).astype(np.float32)
    patches_pred_sar = np.asarray(patch_sar).astype(np.float32)
    patches_pred_fus = np.asarray(patch_fus).astype(np.float32)
    patches_pred_comb = np.asarray(patch_comb).astype(np.float32)

    prob_recontructed_opt = pred_reconctruct(h, w, num_patches_x, num_patches_y, patch_
    prob_recontructed_sar = pred_reconctruct(h, w, num_patches_x, num_patches_y, patch_
    prob_recontructed_fus = pred_reconctruct(h, w, num_patches_x, num_patches_y, patch_
    prob_recontructed_comb = pred_reconctruct(h, w, num_patches_x, num_patches_y, patch

    del patches_pred_opt, patches_pred_sar, patches_pred_fus, patches_pred_comb
    np.save(path_maps+'/'+'prob_opt_'+str(tm)+'.npy',prob_recontructed_opt)
    np.save(path_maps+'/'+'prob_sar_'+str(tm)+'.npy',prob_recontructed_sar)
    np.save(path_maps+'/'+'prob_fus_'+str(tm)+'.npy',prob_recontructed_fus)
    np.save(path_maps+'/'+'prob_comb_'+str(tm)+'.npy',prob_recontructed_comb)

    time_ts.append(end_test)
    del prob_recontructed_opt, prob_recontructed_sar, prob_recontructed_fus, prob_recon
    #del model
time_ts_array = np.asarray(time_ts)
# Save test time
np.save(path_exp+'/metrics_ts.npy', time_ts_array)

```

```
0.0 -8.0
time:  0
time:  1
time:  2
time:  3
time:  4
```

In [16]:

```
# Compute mean of the tm predictions maps
prob_rec_opt = np.zeros((image1_pad.shape[0],image1_pad.shape[1], times))
prob_rec_sar = np.zeros((image1_pad.shape[0],image1_pad.shape[1], times))
prob_rec_fus = np.zeros((image1_pad.shape[0],image1_pad.shape[1], times))
prob_rec_comb = np.zeros((image1_pad.shape[0],image1_pad.shape[1], times))

for tm in range (0, times):
    print(tm)
    prob_rec_opt[:,:,tm] = np.load(path_maps+'/'+'prob_opt_'+str(tm)+'.npy').astype(np.
    prob_rec_sar[:,:,tm] = np.load(path_maps+'/'+'prob_sar_'+str(tm)+'.npy').astype(np.
    prob_rec_fus[:,:,tm] = np.load(path_maps+'/'+'prob_fus_'+str(tm)+'.npy').astype(np.
    prob_rec_comb[:,:,tm] = np.load(path_maps+'/'+'prob_comb_'+str(tm)+'.npy').astype(n

mean_prob_opt = np.mean(prob_rec_opt, axis = -1)
mean_prob_sar = np.mean(prob_rec_sar, axis = -1)
mean_prob_fus = np.mean(prob_rec_fus, axis = -1)
mean_prob_comb = np.mean(prob_rec_comb, axis = -1)

np.save(path_maps+'/prob_mean_opt.npy', mean_prob_opt)
np.save(path_maps+'/prob_mean_sar.npy', mean_prob_sar)
np.save(path_maps+'/prob_mean_fus.npy', mean_prob_fus)
np.save(path_maps+'/prob_mean_comb.npy', mean_prob_comb)
```

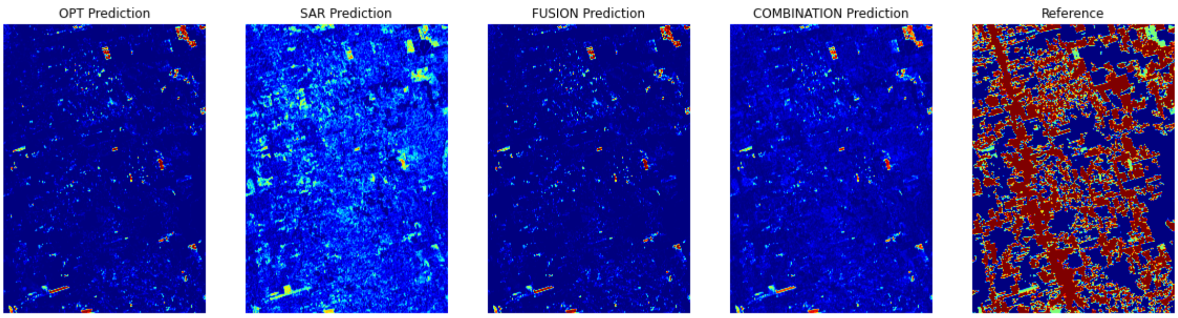
```
0
1
2
3
4
```

In [17]:

```
# Plot mean map and reference
fig = plt.figure(figsize=(20,10))
ax1 = fig.add_subplot(151)
plt.title('OPT Prediction')
ax1.imshow(mean_prob_opt, cmap ='jet')
ax1.axis('off')

ax1 = fig.add_subplot(152)
plt.title('SAR Prediction')
ax1.imshow(mean_prob_sar, cmap ='jet')
ax1.axis('off')

ax1 = fig.add_subplot(153)
plt.title('FUSION Prediction')
ax1.imshow(mean_prob_fus, cmap ='jet')
ax1.axis('off')

ax1 = fig.add_subplot(154)
plt.title('COMBINATION Prediction')
ax1.imshow(mean_prob_comb, cmap ='jet')
ax1.axis('off')

ax2 = fig.add_subplot(155)
plt.title('Reference')
ax2.imshow(final_mask1, cmap ='jet')
ax2.axis('off')
```

Out[17]:

```
(-0.5, 6999.5, 9999.5, -0.5)
```



In [18]:

```
# Computing metrics
mean_prob_opt = mean_prob_opt[:final_mask1.shape[0], :final_mask1.shape[1]]
mean_prob_sar = mean_prob_sar[:final_mask1.shape[0], :final_mask1.shape[1]]
mean_prob_fus = mean_prob_fus[:final_mask1.shape[0], :final_mask1.shape[1]]
mean_prob_comb = mean_prob_comb[:final_mask1.shape[0], :final_mask1.shape[1]]

ref1 = np.ones_like(final_mask1).astype(np.float32)

ref1 [final_mask1 == 2] = 0
TileMask = mask_amazon_ts * ref1
GTTruePositives = final_mask1==1

Npoints = 10

Pmax_opt = np.max(mean_prob_opt[GTTruePositives * TileMask ==1])
ProbList_opt = np.linspace(Pmax_opt,0,Npoints)

Pmax_sar = np.max(mean_prob_sar[GTTruePositives * TileMask ==1])
ProbList_sar = np.linspace(Pmax_sar,0,Npoints)

Pmax_fus = np.max(mean_prob_fus[GTTruePositives * TileMask ==1])
ProbList_fus = np.linspace(Pmax_fus,0,Npoints)

Pmax_comb = np.max(mean_prob_comb[GTTruePositives * TileMask ==1])
ProbList_comb = np.linspace(Pmax_comb,0,Npoints)

metrics_opt = matrics_AA_recall(ProbList_opt, mean_prob_opt, final_mask1, mask_amazon_t
metrics_sar = matrics_AA_recall(ProbList_sar, mean_prob_sar, final_mask1, mask_amazon_t
metrics_fus = matrics_AA_recall(ProbList_fus, mean_prob_fus, final_mask1, mask_amazon_t
metrics_comb = matrics_AA_recall(ProbList_comb, mean_prob_comb, final_mask1, mask_amaz

np.save(path_exp+'/acc_metrics_opt.npy',metrics_opt)
np.save(path_exp+'/acc_metrics_sar.npy',metrics_sar)
np.save(path_exp+'/acc_metrics_fus.npy',metrics_fus)
np.save(path_exp+'/acc_metrics_comb.npy',metrics_comb)
```

```
0.9999894380569458

D:\Ferrari\proj_1\projeto\utils_unet_resunet.py:200: RuntimeWarning: invalid
value encountered in longlong_scalars
  precision_ = TP/(TP+FP)

0.8888795004950629
0.7777695629331801
0.6666596253712973
0.5555496878094144
0.44443975024753146
0.33332981268564865
0.22221987512376584
0.11110993756188292
0.0
0.6341693997383118

D:\Ferrari\proj_1\projeto\utils_unet_resunet.py:200: RuntimeWarning: invalid
value encountered in longlong_scalars
  precision_ = TP/(TP+FP)

0.5637061331007216
0.49324286646313137
```

```
0.42277959982554114
0.35231633318795097
0.2818530665503608
0.21138979991277057
0.14092653327518034
0.07046326663759017
0.0
0.9999988079071045
```

```
D:\Ferrari\proj_1\projeto\utils_unet_resunet.py:200: RuntimeWarning: invalid
value encountered in longlong_scalars
  precision_ = TP/(TP+FP)
```

```
0.8888878292507596
0.7777768505944146
0.6666658719380696
0.5555548932817247
0.4444439146253798
0.3333329359690348
0.2222219573126898
0.1111109786563449
0.0
0.8772140383720398
```

```
D:\Ferrari\proj_1\projeto\utils_unet_resunet.py:200: RuntimeWarning: invalid
value encountered in longlong_scalars
  precision_ = TP/(TP+FP)
```

```
0.7797458118862576
0.6822775854004755
0.5848093589146932
0.48734113242891103
0.38987290594312884
0.2924046794573466
0.19493645297156448
0.09746822648578224
0.0
```

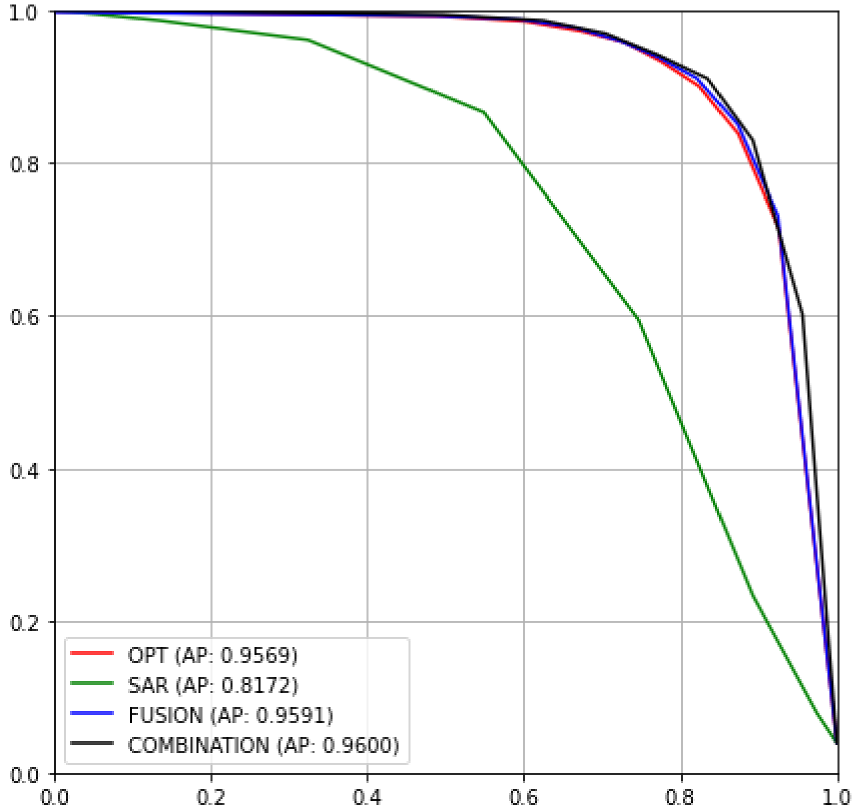
In [19]:

```
# Complete NaN values
metrics_copy_opt = metrics_opt.copy()
metrics_copy_opt = complete_nan_values(metrics_copy_opt)

metrics_copy_sar = metrics_sar.copy()
metrics_copy_sar = complete_nan_values(metrics_copy_sar)

metrics_copy_fus = metrics_fus.copy()
metrics_copy_fus = complete_nan_values(metrics_copy_fus)

metrics_copy_comb = metrics_comb.copy()
metrics_copy_comb = complete_nan_values(metrics_copy_comb)
```

In [20]:

```
# Comput Mean Average Precision (mAP) score
Recall_opt = metrics_copy_opt[:,0]
Precision_opt = metrics_copy_opt[:,1]
AA_opt = metrics_copy_opt[:,2]

Recall_sar = metrics_copy_sar[:,0]
Precision_sar = metrics_copy_sar[:,1]
AA_sar = metrics_copy_sar[:,2]

Recall_fus = metrics_copy_fus[:,0]
Precision_fus = metrics_copy_fus[:,1]
AA_fus = metrics_copy_fus[:,2]

Recall_comb = metrics_copy_comb[:,0]
Precision_comb = metrics_copy_comb[:,1]
AA_comb = metrics_copy_comb[:,2]

DeltaR_opt = Recall_opt[1:]-Recall_opt[:-1]
AP_opt = np.sum(Precision_opt[:-1]*DeltaR_opt)
print('OPT mAP', AP_opt)

DeltaR_sar = Recall_sar[1:]-Recall_sar[:-1]
AP_sar = np.sum(Precision_sar[:-1]*DeltaR_sar)
print('SAR mAP', AP_sar)

DeltaR_fus = Recall_fus[1:]-Recall_fus[:-1]
AP_fus = np.sum(Precision_fus[:-1]*DeltaR_fus)
print('FUSION mAP', AP_fus)

DeltaR_comb = Recall_comb[1:]-Recall_comb[:-1]
AP_comb = np.sum(Precision_comb[:-1]*DeltaR_comb)
print('COMBINATION mAP', AP_comb)

# Plot Recall vs. Precision curve
plt.figure(figsize=(7,7))
plt.plot(metrics_copy_opt[:,0],metrics_copy_opt[:,1], 'r-', label = f'OPT (AP: {AP_opt
plt.plot(metrics_copy_sar[:,0],metrics_copy_sar[:,1], 'g-', label = f'SAR (AP: {AP_sar
plt.plot(metrics_copy_fus[:,0],metrics_copy_fus[:,1], 'b-', label = f'FUSION (AP: {AP_
plt.plot(metrics_copy_comb[:,0],metrics_copy_comb[:,1], 'k-', label = f'COMBINATION (AP
plt.legend(loc="lower left")
ax = plt.gca()
ax.set_ylim([0,1])
ax.set_xlim([0,1])
#plt.plot(metrics_copy[:,0],metrics_copy[:,2])
plt.grid()
```

```
OPT mAP 0.9568938298986461
SAR mAP 0.81723579182166
FUSION mAP 0.9590758759231736
COMBINATION mAP 0.959979694869249
```


OPT (AP: 0.9569)
SAR (AP: 0.8172)
FUSION (AP: 0.9591)
COMBINATION (AP: 0.9600)
0.0
0.2
0.4
0.6
0.8
1.0

In [ ]:

```
1
```