FLAGSHIP 2.6.3: AI ALGORITHM FOR (SATELLITE) IMAGING RECONSTRUCTION

REPORT FOR WP6 MEETING, 19/03/2024

A. Tricomi[1,2,3], G. Piparo[1], G. A. Anastasi[2], E. Tramontana[2], V. Strati[4]

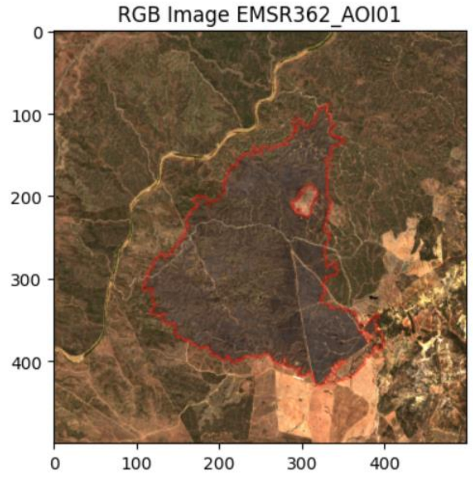
1. INFN Sezione di Catania
2. Università degli studi di Catania
3. Centro Siciliano di Fisica Nucleare e Struttura della Materia (CSFNSM)
4. Università degli studi di Ferrara

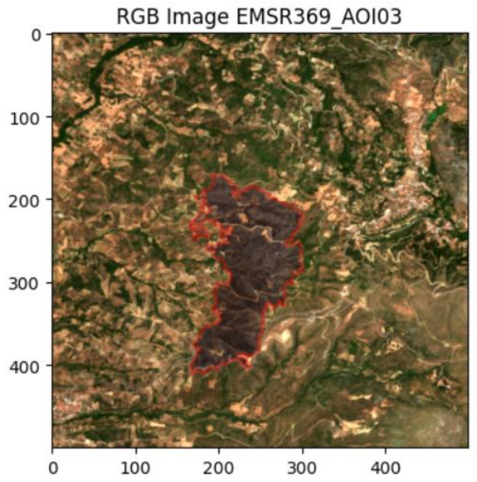
ICSC
Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

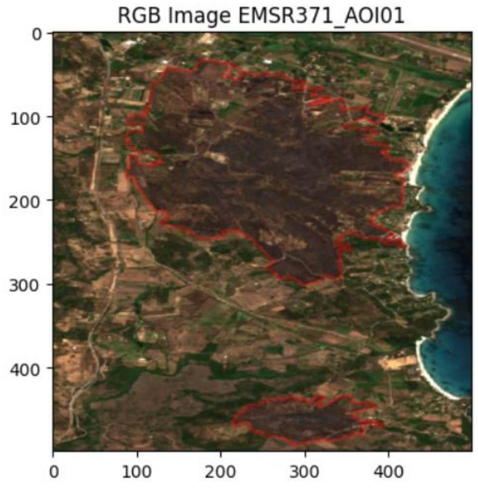
ICSC
Centro Nazionale di Ricerca in HPC,
Big Data and Quantum Computing

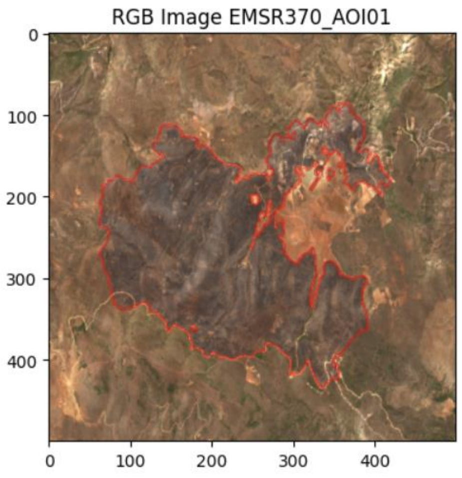
WILDFIRE DATASET FOR BURNT AREA DELIMITATION WITH CNN

- Using the AgriSentinel library and historical fire information taken from the COPERNICUS Emergency Management Service (List of EMS Rapid Mapping Activations | COPERNICUS EMERGENCY MANAGEMENT SERVICE), **we created some test datasets for delimiting burnt areas.**
- The dataset consists of up to 178 512x512 pixels images and was used to train and test a simple Convolutional Neural Network, **which gave fairly good results**.
- Some spectral bands and vegetation indices were used as input.
- **There is a lot of room for improvement by using more images, more features, more complex and deeper models**. The preparation of the dataset for training and testing also plays an important role, as does cloud management.
- **The most interesting objective, apart from the delimitation of burnt areas and the prediction of fires by means of risk maps. It is much more complex, and it may be necessary to add data from other satellite sources and to exploit time sequences. Also, burnt area severity estimation can be a more affordable objective.**





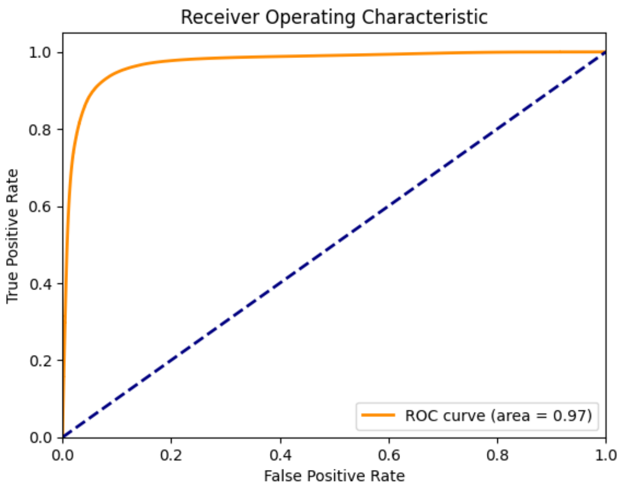


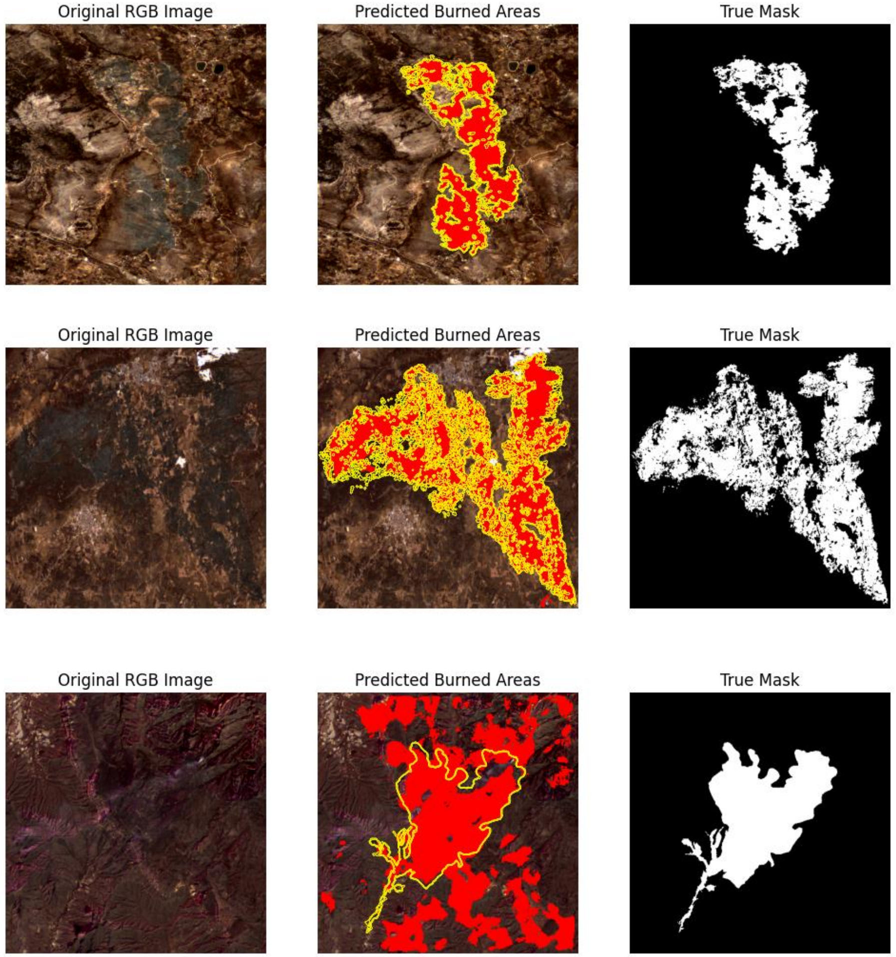


LAST TIME RESULTS

SIMPLE UNET USED FOR THIS WORK

```
def create_model(input_shape):
    inputs = Input(shape=input_shape)

    # Downsample
    c1 = Conv2D(16, (3, 3), activation='relu', padding='same')(inputs)
    c1 = Dropout(0.3)(c1)
    p1 = MaxPooling2D((2, 2))(c1)

    c2 = Conv2D(32, (3, 3), activation='relu', padding='same')(p1)
    c2 = Dropout(0.3)(c2)
    p2 = MaxPooling2D((2, 2))(c2)

    # Bottleneck
    bn = Conv2D(64, (3, 3), activation='relu', padding='same')(p2)

    # Upsample
    u1 = UpSampling2D((2, 2))(bn)
    concat1 = Concatenate()([u1, c2])
    c3 = Conv2D(32, (3, 3), activation='relu', padding='same')(concat1)
    c3 = Dropout(0.3)(c3)

    u2 = UpSampling2D((2, 2))(c3)
    concat2 = Concatenate()([u2, c1])
    c4 = Conv2D(16, (3, 3), activation='relu', padding='same')(concat2)
    c4 = Dropout(0.3)(c4)

    outputs = Conv2D(1, (1, 1), activation='sigmoid', padding='same')(c4)

    model = Model(inputs=[inputs], outputs=[outputs])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

    return model


model = create_model(input_shape=(500, 500, 13))
history = model.fit(X_train, y_train, epochs=50, batch_size = 5, validation_data=(X_test, y_test))
```





Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

RECENT IMPROVEMENTS

- In-depth study of the state of the art of b**urnt area detection and fire severity estimation** using satellite data and Deep Learning techniques.
- Evaluation of the dataset and selection of the best images (burnt areas not too small, little cloud cover, absence of unlabelled past fire events, etc.).
- **Feature engineering:** used 10 spectral bands (red, green, blue, red edge 1, red edge 2, red edge 3, NIR1, NIR2, SWIR1, SWIR2) and 9 indices (BASI2, MIRBI, NBR, NBR2, NBR+, NDRE, NDVI, NDWI, OSAVI).

RECENT IMPROVEMENTS

- In-depth study of the state of the art of **burnt area detection and fire severity estimation** using satellite data and Deep Learning techniques.
- Evaluation of the dataset and selection of the best images (burnt areas not too small, little cloud cover, absence of unlabelled past fire events, etc.).
- **Feature engineering:** used 10 spectral bands (red, green, blue, red edge 1, red edge 2, red edge 3, NIR1, NIR2, SWIR1, SWIR2) and 9 indices (BASI2, MIRBI, NBR, NBR2, NBR+, NDRE, NDVI, NDWI, OSAVI).
- **Improved the Unet presented last time and implemented custom losses.**

CUSTOM LOSS FUNCTIONS

```python
def dice_loss(y_true, y_pred):
    y_true_f = K.cast(y_true, 'float32')
    y_pred_f = y_pred
    numerator = 2 * K.sum(y_true_f * y_pred_f)
    denominator = K.sum(y_true_f + y_pred_f)
    return 1 - (numerator + K.epsilon()) / (denominator + K.epsilon())

def jaccard_loss(y_true, y_pred):
    y_true_f = K.cast(y_true, 'float32')
    y_pred_f = y_pred
    intersection = K.sum(y_true_f * y_pred_f)
    sum_ = K.sum(y_true_f + y_pred_f)
    jac = (intersection + K.epsilon()) / (sum_ - intersection + K.epsilon())
    return 1 - jac


def dice_jaccard_loss(y_true, y_pred):
    return 0.5 * dice_loss(y_true, y_pred) + 0.5 * jaccard_loss(y_true, y_pred)


def dice_jaccard_crossentropy_loss(y_true, y_pred, dice_weight=0.4, jaccard_weight=0.4, crossentropy_weight=0.2):
    dice_l = dice_loss(y_true, y_pred)
    jaccard_l = jaccard_loss(y_true, y_pred)
    crossentropy_l = binary_crossentropy(y_true, y_pred)
    total_loss = (dice_weight * dice_l) + (jaccard_weight * jaccard_l) + (crossentropy_weight * crossentropy_l)
    return total_loss
```

UNET

```python
def create_model(input_shape, weight_decay=1e-4,callbacks=None):
    inputs = Input(shape=input_shape)

    # Downsample
    c1 = Conv2D(32, (3, 3), padding='same', kernel_regularizer=l2(weight_decay))(inputs)
    #c1 = BatchNormalization()(c1)
    c1 = Activation('relu')(c1)
    c1 = Dropout(0.3)(c1)
    p1 = MaxPooling2D((2, 2))(c1)

    c2 = Conv2D(64, (3, 3), padding='same', kernel_regularizer=l2(weight_decay))(p1)
    #c2 = BatchNormalization()(c2)
    c2 = Activation('relu')(c2)
    c2 = Dropout(0.3)(c2)
    p2 = MaxPooling2D((2, 2))(c2)

    # Bottleneck
    bn = Conv2D(128, (3, 3), padding='same', kernel_regularizer=l2(weight_decay))(p2)
    #bn = BatchNormalization()(bn)
    bn = Activation('relu')(bn)
    bn = Dropout(0.3)(bn)

    # Upsample
    u1 = UpSampling2D((2, 2))(bn)
    concat1 = Concatenate()([u1, c2])
    c3 = Conv2D(64, (3, 3), padding='same', kernel_regularizer=l2(weight_decay))(concat1)
    #c3 = BatchNormalization()(c3)
    c3 = Activation('relu')(c3)
    c3 = Dropout(0.3)(c3)

    u2 = UpSampling2D((2, 2))(c3)
    concat2 = Concatenate()([u2, c1])
    c4 = Conv2D(32, (3, 3), padding='same', kernel_regularizer=l2(weight_decay))(concat2)
    #c4 = BatchNormalization()(c4)
    c4 = Activation('relu')(c4)
    c4 = Dropout(0.3)(c4)

    outputs = Conv2D(1, (1, 1), activation='sigmoid', padding='same')(c4)

    model = Model(inputs=[inputs], outputs=[outputs])
    model.compile(optimizer=Adam(learning_rate=0.001), loss=dice_jaccard_crossentropy_loss, metrics=['accuracy'])

    return model
```

ICSC
Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

RECENT IMPROVEMENTS

- In-depth study of the state of the art of **burnt area detection and fire severity estimation** using satellite data and Deep Learning techniques.
- Evaluation of the dataset and selection of the best images (burnt areas not too small, little cloud cover, absence of unlabelled past fire events, etc.).
- **Feature engineering:** used 10 spectral bands (red, green, blue, red edge 1, red edge 2, red edge 3, NIR1, NIR2, SWIR1, SWIR2) and 9 indices (BASI2, MIRBI, NBR, NBR2, NBR+, NDRE, NDVI, NDWI, OSAVI).
- **Improved the Unet presented last time and implemented custom losses.**
- **New model developed, based on a Unet-like architecture with Long Short Term Memory modules to study time series.**
- Performed a test with 23 images/time series (70% train 30% test) to compare the two developed models, with 50 epochs and batch size=2.

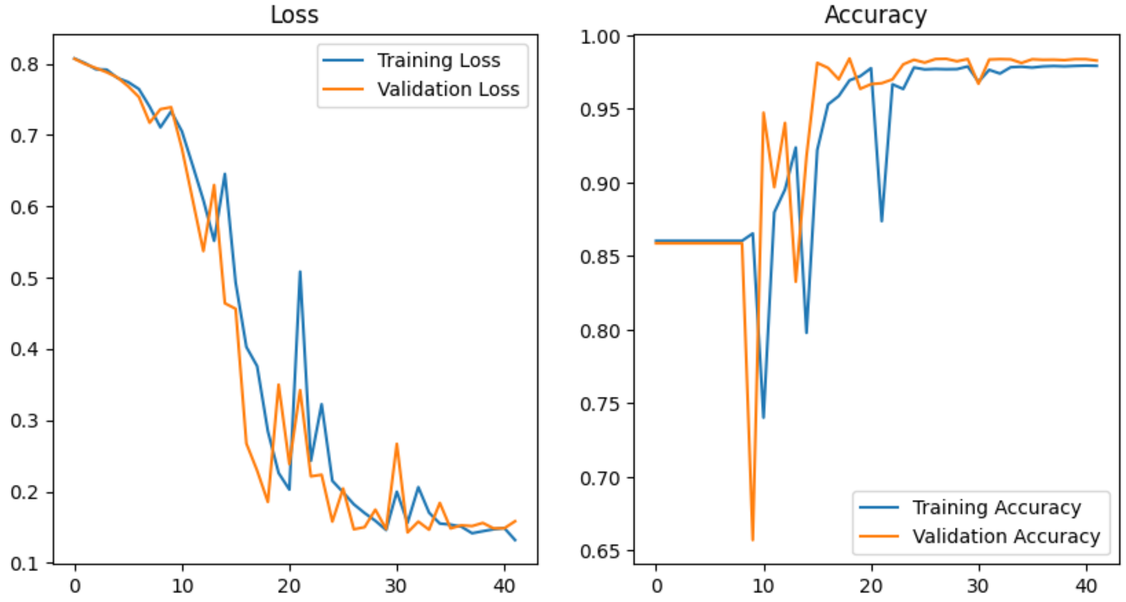
UNET-LIKE WITH LSTM

```python
def create_lstm_unet(input_shape, weight_decay=1e-4, callbacks=None):
    inputs = Input(shape=input_shape)

    # Encoder
    c1 = ConvLSTM2D(16, (3, 3), padding='same', kernel_regularizer=l2(weight_decay), return_sequences=True)(inputs)
    #c1 = TimeDistributed(BatchNormalization())(c1)
    c1 = Activation('relu')(c1)
    c1 = TimeDistributed(Dropout(0.2))(c1)
    p1 = TimeDistributed(MaxPooling2D((2, 2)))(c1)

    c2 = ConvLSTM2D(32, (3, 3), padding='same', kernel_regularizer=l2(weight_decay), return_sequences=True)(p1)
    #c2 = TimeDistributed(BatchNormalization())(c2)
    c2 = Activation('relu')(c2)
    c2 = TimeDistributed(Dropout(0.2))(c2)
    p2 = TimeDistributed(MaxPooling2D((2, 2)))(c2)

    # Bottleneck
    bn = ConvLSTM2D(64, (3, 3), padding='same', kernel_regularizer=l2(weight_decay), return_sequences=True)(p2)
    #bn = TimeDistributed(BatchNormalization())(bn)
    bn = Activation('relu')(bn)
    bn = TimeDistributed(Dropout(0.2))(bn)

    # Decoder
    u1 = TimeDistributed(UpSampling2D((2, 2)))(bn)
    u1 = ConvLSTM2D(32, (3, 3), padding='same', kernel_regularizer=l2(weight_decay), return_sequences=True)(u1)
    #u1 = TimeDistributed(BatchNormalization())(u1)
    u1 = Activation('relu')(u1)
    u1 = TimeDistributed(Dropout(0.2))(u1)
    concat1 = Concatenate(axis=-1)([u1, c2])

    u2 = TimeDistributed(UpSampling2D((2, 2)))(concat1)
    u2 = ConvLSTM2D(16, (3, 3), padding='same', kernel_regularizer=l2(weight_decay), return_sequences=True)(u2)
    #u2 = TimeDistributed(BatchNormalization())(u2)
    u2 = Activation('relu')(u2)
    u2 = TimeDistributed(Dropout(0.2))(u2)
    concat2 = Concatenate(axis=-1)([u2, c1])

    # Output layer
    outputs = ConvLSTM2D(1, (3, 3), activation='sigmoid', padding='same', return_sequences=True)(concat2)
    outputs = Lambda(lambda x: x[:, -1, :, :, :])(outputs)  # Last timestep

    model = Model(inputs=inputs, outputs=outputs)
    model.compile(optimizer=Adam(learning_rate=0.001), loss=dice_jaccard_crossentropy_loss, metrics=['accuracy'])

    return model
```

RESULTS COMPARISON: LOSS AND ACCURACY

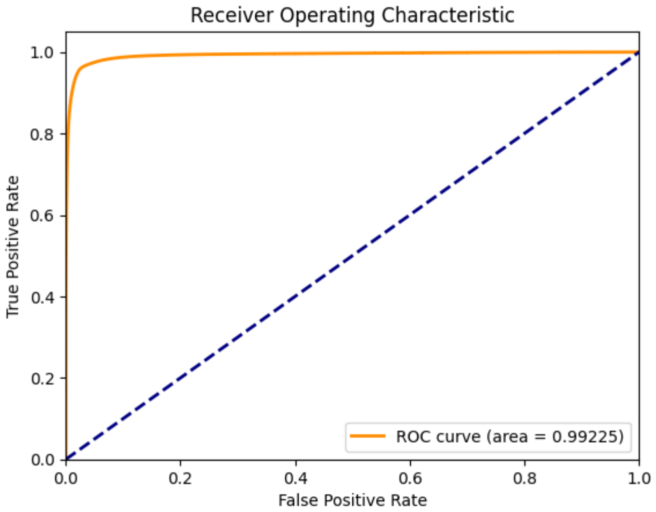
SIMPLE UNET

UNET-LIKE WITH LSTM



RESULTS COMPARISON: METRICS

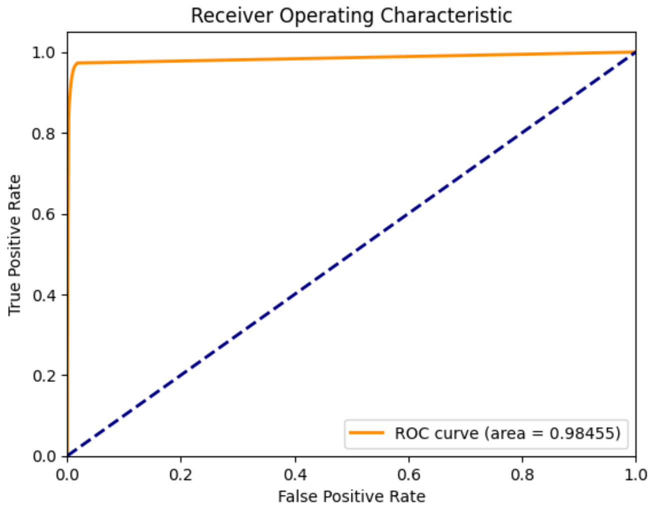
SIMPLE UNET




```
Best Threshold: 0.16654659807682037
Best F1 Score: 0.9191759863667658
Accuracy con soglia ottimale: 0.9769368852887835
Precision con soglia ottimale: 0.910044586300396
Recall con soglia ottimale: 0.9284893007894188
F1 Score con soglia ottimale: 0.9191744221356011

Mean Dice Coefficient: 0.8664173656680502

Mean Jaccard Coefficient: 0.7887315422742507
```

Unet-like with LSTM metrics are generally better as expected.

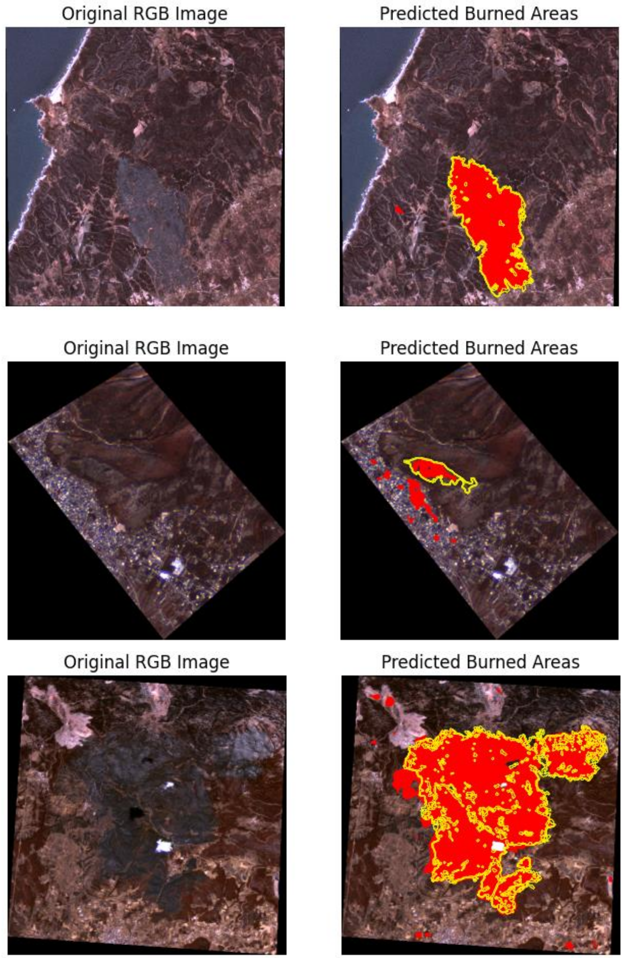
UNET-LIKE WITH LSTM




```
Best Threshold: 0.24125345051288605
Best F1 Score: 0.9457018454551233
Accuracy con soglia ottimale: 0.984549931174665
Precision con soglia ottimale: 0.9389297300175323
Recall con soglia ottimale: 0.9525692767133013
F1 Score con soglia ottimale: 0.9457003261696043

Mean Dice Coefficient: 0.8922873036551995

Mean Jaccard Coefficient: 0.832100667904353
```

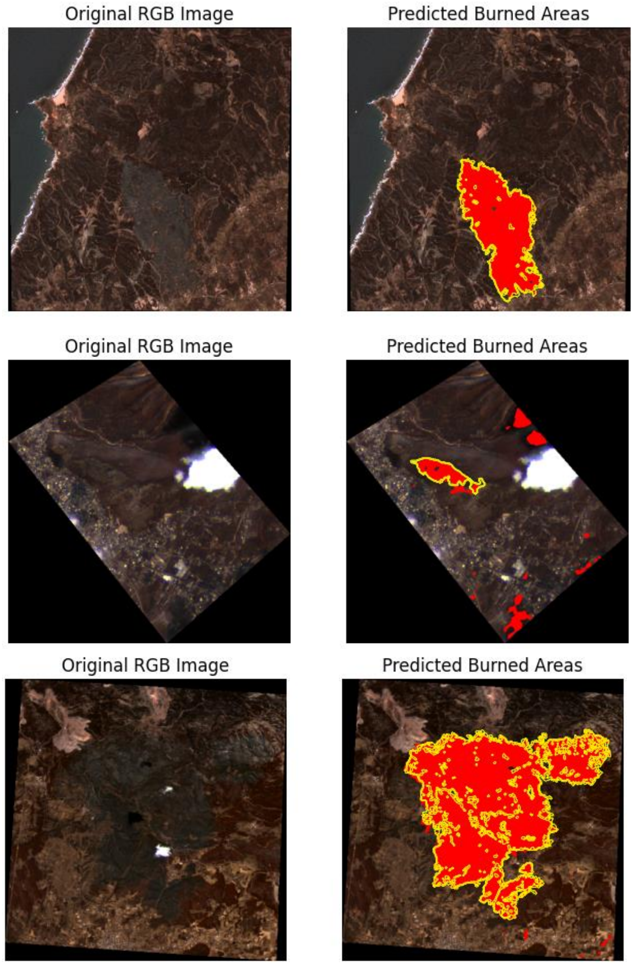
RESULTS COMPARISON: VISUAL OUTPUT

SIMPLE UNET



True label border
Predicted label

UNET-LIKE WITH LSTM



NEXT STEP

- Continuing bibliographic research.

- Improve the library for dataset management (**download higher resolution images and crop them, data augmentation**). Documentation and general improvements will be made in view of the MS8 milestone on the availability of a repository.

- Scaling of the number of images used and models as soon as resources become available. Try out other architectures.

- **Try performing other tasks, such as assessing the severity of fire damage (we already have the labels for this analysis) and fire prediction. The latter is usually carried out in the literature using also data of a different nature (e.g. meteorological), it may be difficult to do this using satellite images alone.**