ANEXO B

Sentinel – Unet- Keras

Índice de Contenido

1.0. Introducción ... 3
2.0. Implementación en Keras ... 4
2.1. Entorno ... 4
2.2. Arquitectura de la red ... 5
2.3. Tipos de inicialización de los pesos ... 7
2.4. Entrenamiento ... 8
2.5. Función de pérdida (Loss function) ... 10
2.6. Experimentos ... 11
2.6.1. E011-10 Matlab ... 12
2.6.2. K-001 Arquitectura Matlab en Keras (BASE) ... 15
2.6.3. K-002 Variante 1: UpSampling Bilineal ... 18
2.6.4. K-003 Variante 2: Sin Dropout, UpSampling Bilineal ... 21
2.6.5. K-004 Variante 3: Sin Dropout, UpSampling Bilineal, BatchNormalization 24
2.6.6. E011-11 Matlab NO-L2R ... 27
2.6.7. K-005 Arquitectura Matlab en Keras L2R ... 30

1.0. Introducción

En este documento se analiza con detalle la implementación en Keras de la red convolucional U-NET y se realizarán una serie de experimentos.

2.0. Implementación en Keras

2.1. Entorno

En este apartado se detallan los requisitos necesarios para ejecutar el entrenamiento y evaluación de la red.

- Nvidia drivers (versión 390)

Para instalar los drivers de nvidia se usan los siguientes comandos

```
sudo apt update
sudo apt upgrade
sudo apt install nvidia-driver-390
sudo reboot
nvidia-smi # to check instalation
```

- Cuda 11.2 (última versión)
- Cuda 10.1
- Cuda 10.0

Para instalar estas versiones de **cuda** se utilizan los siguientes comandos.

```
wget
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64/cu
da-ubuntu1804.pin
sudo mv cuda-ubuntu1804.pin /etc/apt/preferences.d/cuda-repository-pin-600
sudo apt-key adv --fetch-keys
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64/7f
a2af80.pub
sudo add-apt-repository "deb
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1804/x86_64/
/"
sudo apt-get update
sudo apt-get -y install cuda
sudo apt install cuda10-0
sudo apt install cuda10-1
sudo apt install libcudnn8
```

- Python 3.8
- Pip3 (necesita actualizarse mediante “install –upgrade pip” antes de instalar las librerías)
- Librerías de Python:
    - tensorflow
    - tensorflow-gpu
    - keras
    - opencv-python
    - sklearn
    - imageio
    - numba

2.2. Arquitectura de la red

En este apartado se muestra la implementación de la arquitectura UNet en Keras realizada. Para esta implementación se parte del código de los siguientes GitHub:
https://github.com/zhixuhao/unet/blob/master/model.py
https://gist.github.com/margaretmeehan/f6831e5f5b071ba96eabb3dd91f38bec
https://github.com/nikhilroxtomar/Multiclass-Segmentation-in-Unet/blob/master/model.py
Ya que en la mayoría de las implementaciones de UNet se realizan **clasificaciones binarias**, normalmente, se utiliza una **convolución final de un solo filtro con la activación sigmoide**. Sin embargo, si se quieren **predicciones multiclase** hay que modificar esta **última convolución** para que tenga **tantos filtros como clases** y utilizar la activación **softmax**.

La arquitectura definida en Keras se basa en la proporcionada por Matlab para reproducir sus mismos resultados y comprobar su correcto funcionamiento. La arquitectura de **Matlab difiere de la presentada en la publicación de UNet [Ronneberger2015]** ya que **añade una capa de Dropout antes de la sección "puente" y otra justo después** de esta sección de la arquitectura. Finalmente, **en las pruebas Matlab se concretó que la configuración más óptima se obtiene utilizando la mitad de los filtros convolucionales** de los que se detallan en la publicación de UNet [Ronneberger2015].

A continuación, se muestra el código **Python** que implementa el modelo **UNet multiclase** mediante **Keras**.

```
def conv_block(inputs, filters, pool=True, drop=False):
    x = Conv2D(filters, 3, padding="same")(inputs)
    #x = BatchNormalization()(x)
    x = Activation("relu")(x)

    x = Conv2D(filters, 3, padding="same")(x)
    #x = BatchNormalization()(x)
    x = Activation("relu")(x)

    if pool == True and drop == True:
        p = Dropout(0.5)(x)
        p = MaxPool2D((2, 2))(p)
        return x, p
    elif pool == True and drop == False:
        p = MaxPool2D((2, 2))(x)
        return x, p
    else:
        return x
```

```
def build_unet(shape, num_classes):
    inputs = Input(shape)
```

```
    """ Encoder """
    x1, p1 = conv_block(inputs, 32, pool=True) #16
    x2, p2 = conv_block(p1, 64, pool=True) #32
    x3, p3 = conv_block(p2, 128, pool=True) #48
    x4, p4 = conv_block(p3, 256, pool=True, drop=True) #64

    """ Bridge """
    b1 = conv_block(p4, 512, pool=False)
    drop2 = Dropout(0.5)(b1)

    """ Decoder """
    u1 = Conv2DTranspose(256, 2, strides=2, padding="same")(drop2) #UpSam
pling2D((2, 2), interpolation="bilinear")(drop2)
    c1 = Concatenate()([u1, x4])
    x5 = conv_block(c1, 256, pool=False) #64

    u2 = Conv2DTranspose(128, 2,strides=2, padding="same")(x5) #UpSamplin
g2D((2, 2), interpolation="bilinear")(x5)
    c2 = Concatenate()([u2, x3])
    x6 = conv_block(c2, 128, pool=False) #48

    u3 = Conv2DTranspose(64, 2,strides=2, padding="same")(x6) #UpSampling
2D((2, 2), interpolation="bilinear")(x6)
    c3 = Concatenate()([u3, x2])
    x7 = conv_block(c3, 64, pool=False) #32

    u4 = Conv2DTranspose(32, 2, strides=2, padding="same")(x7) #UpSamplin
g2D((2, 2), interpolation="bilinear")(x7)
    c4 = Concatenate()([u4, x1])
    x8 = conv_block(c4, 32, pool=False) #16

    """ Output layer """
    output = Conv2D(num_classes, 1, padding="same", activation="softmax")
(x8)

    return Model(inputs, output)
```

2.3. Tipos de inicialización de los pesos

En este apartado se listan todos los tipos de inicialización de los pesos de los filtros convolucionales disponibles en Keras.

- **constant**: Inicializa tensores con valores constantes.
- **glorot_normal**: Inicializador Glorot normal, también llamado Xavier normal.
- **glorot_uniform**: Inicializador Glorot uniform, también llamado Xavier uniform.
- **he_normal**: Inicializador He normal.
- **he_uniform**: Inicializador He uniform con escalado de varianza.
- **identity**: Inicializador que genera la matriz identidad.
- **lecun_normal**: Inicializador Lecun normal.
- **lecun_uniform**: Inicializador Lecun uniform.
- **ones**: Inicializador de tensores con valor 1.
- **orthogonal**: Inicializador que genera una matriz ortogonal.
- **random_normal**: Inicializador que genera tensores con una distribución normal.
- **random_uniform**: Inicializador que genera tensores con una distribución uniforme.
- **truncated_normal**: Inicializador que genera tensores con una distribución normal truncada.
- **variance_scaling**: Inicializador capaz de adaptar su escala a la forma de los pesos de los tensores.
- **zeros**: Inicializador de tensores con valor 0.

2.4. Entrenamiento

Para realizar el entrenamiento hay que cargar el modelo definido previamente, compilarlo con la llamada **compile** con los parámetros apropiados y finalmente llamar a la **función fit** con los parámetros de entrenamiento para comenzar su ejecución.

```
 model = unet(input_shape)
    model.compile(optimizer=Adam(lr=learning_rate),
                  loss="categorical_crossentropy", metrics=['accuracy'],
loss_weights=[1.1175, 6.0888,0.4397])
    val_loss_checkpoint = ModelCheckpoint(val_loss_min_fpath,
                                          monitor='val_loss',
                                          verbose=1,
                                          save_best_only=True,
                                          mode='min')
    early_stop = EarlyStopping(monitor='val_loss',
                               patience=early_stop_val_patience)
    history = model.fit(X_train,
                        y_train,
                        batch_size=batch_size,
                        epochs=num_epochs,
                        validation_data=(X_test, y_test),
                        callbacks=[early_stop, val_loss_checkpoint])
    model.save(final_model_output_fpath)
```

<u>Función compile:</u>

- **optimizer:** Declara la función optimizadora
- **loss:** Declara la función de loss.
- **metrics:** Declara las métricas a obtener.
- **loss_weights:** Declara los pesos de cada clase para el entrenamiento.
- **weighted_metrics:** Declara pesos para las métricas.
- **run_eagerly:** Permite ejecutar el código fuera de un tf.function.
- **steps_per_execution**: Para ejecutar mas de un batch simultáneamente, se utiliza en TPUs para mejorar su rendimiento.

<u>Función fit:</u>

- **X:** Datos de entrada
- **Y:** Ground truth
- **batch_size:** Tamaño del batch
- **epochs:** Número de epochs a ejecutar
- **verbose=1:** Cantidad de información devuelta por pantalla
- **callbacks=None,** Permite implementar funciones que detienen el entrenamiento
- **validation_split=0.0,** Permite separar un % de datos para utilizarlos en validación y no en entrenamiento
- **validation_data=None,** Datos para la validación
- **shuffle=True,** Barajado del dataset en cada epoch
- **class_weight=None,** Permite añadir pesos a la función de pérdida de cada clase
- **sample_weight=None,** Permite añadir pesos a las muestras del dataset.
- **initial_epoch=0,** Epoch en la que se empieza el entrenamiento.

- **steps_per_epoch=None,** Número de batchs ejecutados por epoch, si se deja sin declarar utiliza todo el dataset
- **validation_steps=None,** Número de batchs ejecutados por validación, si se deja sin declarar utiliza todo el dataset
- **validation_batch_size=None**, Tamaño de batch utilizado en la validación. Si no se detalla se utiliza el mismo valor que en el entrenamiento.
- **validation_freq=1**, Número de epochs entre cada validación
- **max_queue_size=10**, Solo se utiliza con datos de tipo generator o keras.utils.Sequence. Controla el tamaño de la cola.
- **workers=1,** Número de hilos. Solo se utiliza con datos de tipo generator o keras.utils.Sequence
- **use_multiprocessing=False,** Permite procesamiento en paralelo. Solo se utiliza con datos de tipo generator o keras.utils.Sequence

2.5. Función de pérdida (Loss function)

En este apartado se listan todas las funciones de pérdida disponibles en la implementación de Keras. Además, **Keras permite implementar una función de pérdida propia**.

Probabilistic losses:

- **binary_crossentropy**
- **categorical_crossentropy:** Se utiliza cuando hay 2 o más clases. Debe utilizar la codificación One-Hot para los datos de entrada. Esta codificación consiste en un array de tantos valores como clases se quieran predecir por cada dato. En cada posición del array se considera de forma binaria con 0 o 1 si pertenece a esa clase o no. Es decir, si se consideran 3 clases la codificación One-hot para un dato de la clase 2 sería la siguiente [0,1,0]. Esta codificación es sencilla y permite detectar estados erróneos fácilmente, sin embargo, cuando se utiliza un número elevado de clases utiliza mas memoria que otros métodos.
- **sparse_categorical_crossentropy**: Utiliza la misma función de pérdida que "categorical_crossentropy" pero en lugar de utilizar una codificación de las clases one-hot, utiliza un valor entero. Por ejemplo, la clase 2 sería equivalente al one-hot [0,0,1,0] si se utilizasen 4 clases. De esta forma se reduce el uso de memoria y la cantidad de computaciones necesarias.
- **poisson**
- **binary_crossentropy**
- **kl_divergence**

Regression losses:

- **mean_squared_error function**
- **mean_absolute_error function**
- **mean_absolute_percentage_error function**
- **mean_squared_logarithmic_error function**
- **cosine_similarity function**
- **huber**
- **log_cosh**

Hinge losses for "maximum-margin" classification:

- **hinge**
- **squared_hinge**
- **categorical_hinge**

La función de pérdida **"categorical_crossentropy" es la utiliza por defecto en aplicaciones de redes neuronales convolucionales** cuando se quiere hacer una **predicción multiclase**. En la implementación de **Keras**, **se decide utilizar su versión "sparse_categorical_crossentropy"** por su mayor sencillez al poder utilizar imágenes de **una sola banda** (en escala de grises) como ground truth para la red.

2.6. Experimentos

En este apartado se documentas los experimentos realizados con la implementación de Keras de UNet. En estos experimentos se prueba a variar la arquitectura de la red respecto a la arquitectura Matlab

El experimento **Matlab (E011-10)** utiliza **dos capas de Dropout**, **no tiene ninguna capa de BatchNormalization, implementa L2R,** y utiliza un UpSampling mediante **convoluciones transpuestas**. El experimento que sigue la arquitectura de Matlab en la **implementación de Keras** (el experimento BASE) **sigue su misma estructura, a excepción de la L2R**.

Finalmente, los **experimentos "Variante" utilizan un UpSampling Bilineal** en lugar de convoluciones transpuestas. Las **Variantes 2 y 3 no utilizan capas de Dropout**. La **Variante 3 añade capas de BatchNormalization** detrás de cada convolución. En todas las arquitecturas, tanto de Matlab, como de la implementación de Keras, se utiliza padding para que la imagen de salida sea del mismo tamaño que la de entrada, a diferencia de la publicación original en donde no se aplica.

Experimento	GlobalAccuracy	MeanRecall	MeanPrecision
E011-10 Matlab	0.69	0.65	0.58
K-001 Arquitectura Matlab en Keras (BASE)	0.66	0.63	0.57
K-002 Variante 1: UpSampling Bilineal	0.65	0.63	0.56
K-003 Variante 2: Sin Dropout, UpSampling Bilineal	0.65	0.62	0.56
K-004 Variante 3: Sin Dropout, UpSampling Bilineal, BatchNormalization	0.66	0.63	0.56
E011-11 Matlab NO-L2R	0.66	0.61	0.55
K-005 Arquitectura Matlab en Keras L2R	0.64	0.60	0.56

Tabla 1 Resumen de los experimentos

Se obtienen **resultados muy similares a los obtenidos por la implementación Matlab**, aunque ligeramente inferiores. El resto de los experimentos realizados, a pesar de cambiar la arquitectura de la red, obtienen unos resultados muy similares entre sí, siendo el mejor el equivalente a la arquitectura Matlab.

En todos los experimentos Keras se utiliza una épsilon de 1e-7 para el solver Adam, mientras que en Matlab se utiliza un valor de 1e-8, sin embargo, se han realizado varias pruebas y no se producen diferencias en los resultados ya que este valor solamente se utiliza como sustituto del valor 0 para evitar divisiones entre 0. Si se aumenta este valor excesivamente, como, por ejemplo, a 0.1, la red no entrena correctamente.

Además, se aprecia una **mejora sustancial en el rendimiento de la red al utilizar convoluciones transpuestas en lugar de filtros de upsampling bilineales**, disminuyendo el tiempo de ejecución del entrenamiento para 125 epochs de una hora y media a solamente media hora, es decir, **tres veces más rápido.**

Finalmente, **parece que en Matlab no utilizar la regularización L2 provoca un sobreajuste de la red que afecta negativamente** a los resultados obtenidos, mientras que en la implementación de **Keras si se implementa dicha regularización se afecta negativamente a los resultados debido a que la red deja de mejorar** a pesar de no tener sobreajuste.

2.6.1. E011-10 Matlab

En este apartado se muestran los resultados del experimento BASE en la **implementación de Matlab**.

Parámetros de entrada	
Parámetros del dataset	
Conjunto de entrenamiento	Train1Aumentado (10 bandas, 3 clases)
Conjunto de test	Test1Aumentado (10 bandas, 3 clases)
Parámetros de la red	
InputSize	[256 256 10]
Número de Clases	4 (3+BKG)
Profundidad de la red	**4**
Dropout	**Si**
BatchNormalization	**No**
Upsampling	**Convolución Transpuesta (2,2)**
Numero de filtros (Encoder)	**32**
Downsampling Factor (Output Stride)	8
Padding	Sí
Pesos en las clases	Median Frequency Weighting
Parámetros de entrenamiento	
Solucionador de red (solver network)	Adam
Epochs	125
BatchSize	32
LearningRate -Inicial **LearnRate-Final**	0.0005 0.0005
Gradient Clipping	1
Regularizacion L2	0.0001
Data Augmentation	No
Shuffle	Si
Momentum	-
Duración del entrenamiento	02:07:41

Tabla 2 Parámetros del experimento MATLAB E011-10

GlobalAccuracy	MeanRecall	MeanPrecision
0.69	0.65	0.58

Tabla 3 Resultados globales (MATLAB E011-10)

Matriz de confusión						
	# px predichos	PSPRPA	EDZUCA	TAVI	BACKGROUND	PRECISION
# px reales		**6,887,820**	**1,158,558**	**15,525,451**	**8,540,811**	
PSPRPA	**7,931,712**	4,479,889	182,125	954,412	2,315,286	*0.56481*
EDZUCA	**3,263,102**	648,267	752,077	1,163,114	699,644	*0.23048*
TAVI	**15,141,556**	432,644	150,600	12,902,427	1,655,885	*0.85212*
BACKGROUND	**5,776,270**	1,327,020	73,756	505,498	3,869,996	*0.66998*
RECALL		*0.6504*	*0.6491*	*0.8311*	*0.4531*	**0.68523**

Tabla 4 Matrix de confusión (MATLAB E011-10)

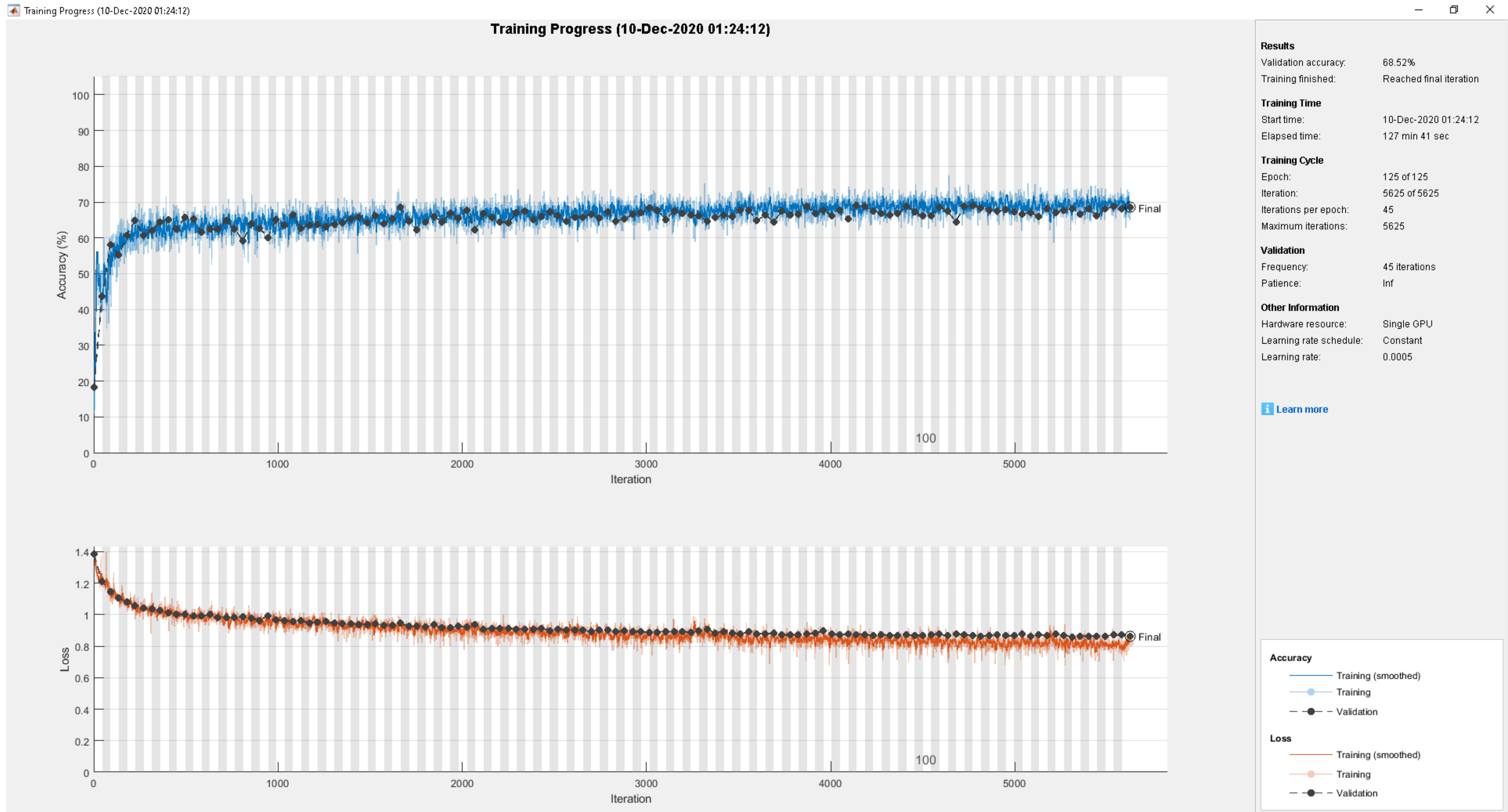


Fig 1 Función de Loss (E011-10)

Imágenes de ejemplo

BKG	EDZUCA	PSPRPA	TA

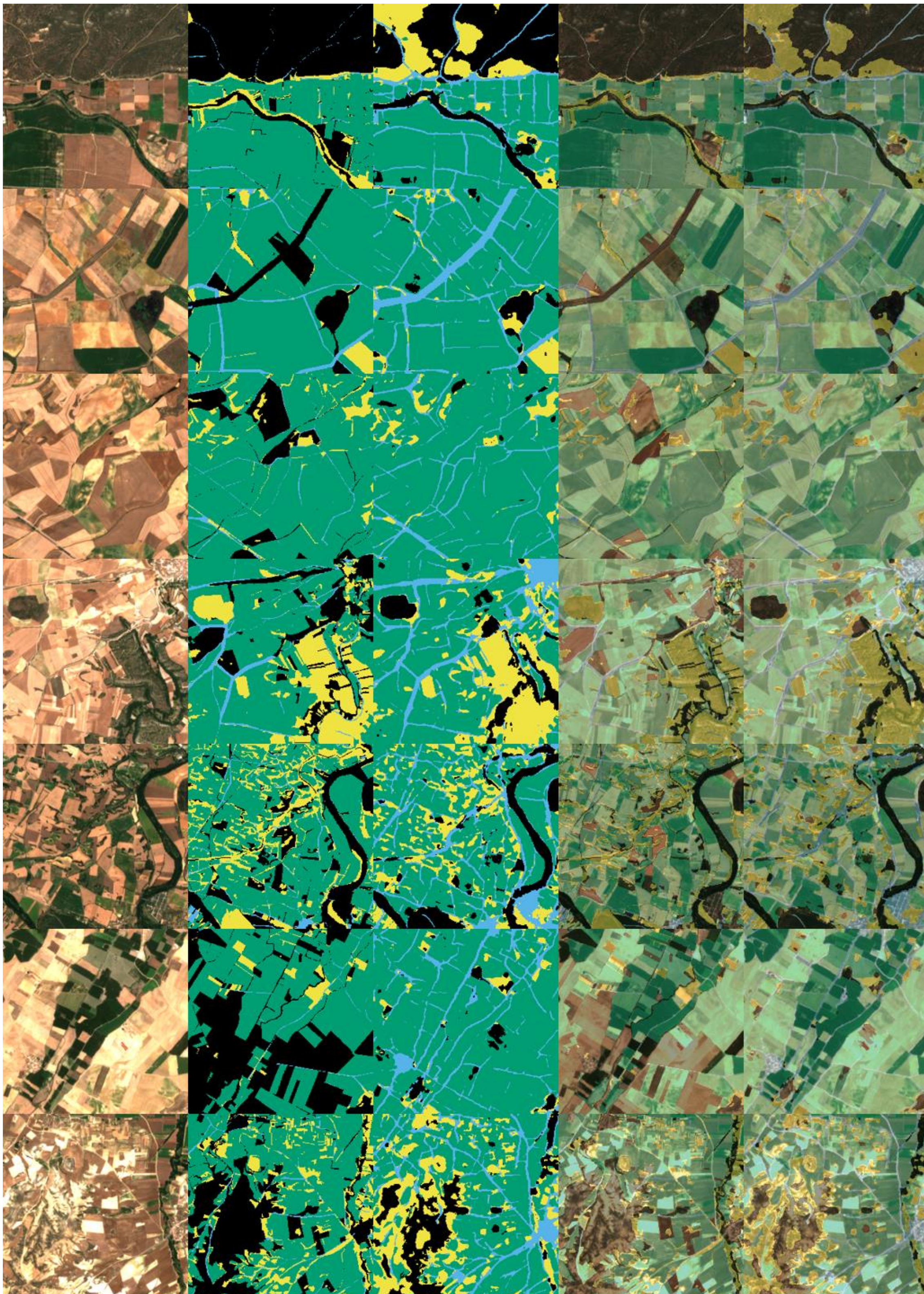


Fig 2 Imágenes de ejemplo (MATLAB E011-10) 1.Original, 2.GroundTruth, 3.Predicción, 4.Original+GT, 5.Original+Predicción

2.6.2. K-001 Arquitectura Matlab en Keras (BASE)

En este apartado se muestran los resultados del experimento BASE en la **implementación Keras**.

Parámetros de entrada	
Parámetros del dataset	
Conjunto de entrenamiento	Train1Aumentado (10 bandas, 3 clases)
Conjunto de test	Test1Aumentado (10 bandas, 3 clases)
Parámetros de la red	
InputSize	[256 256 10]
Número de Clases	4 (3+BKG)
Profundidad de la red	**4**
Dropout	**Si**
BatchNormalization	**No**
Upsampling	**Convolución Transpuesta (2,2)**
Numero de filtros (Encoder)	**32**
Padding	Sí
Pesos en las clases	Median Frequency Weighting
Parámetros de entrenamiento	
Solucionador de red (solver network)	Adam
Epochs	125 (Validacion con parada de 30 ep)
BatchSize	32
LearningRate -Inicial **LearnRate-Final**	0.0005 0.0005
Gradient Clipping	-
Regularizacion L2	-
Data Augmentation	No
Shuffle	Si
Momentum	-
Duración del entrenamiento	00:26:55

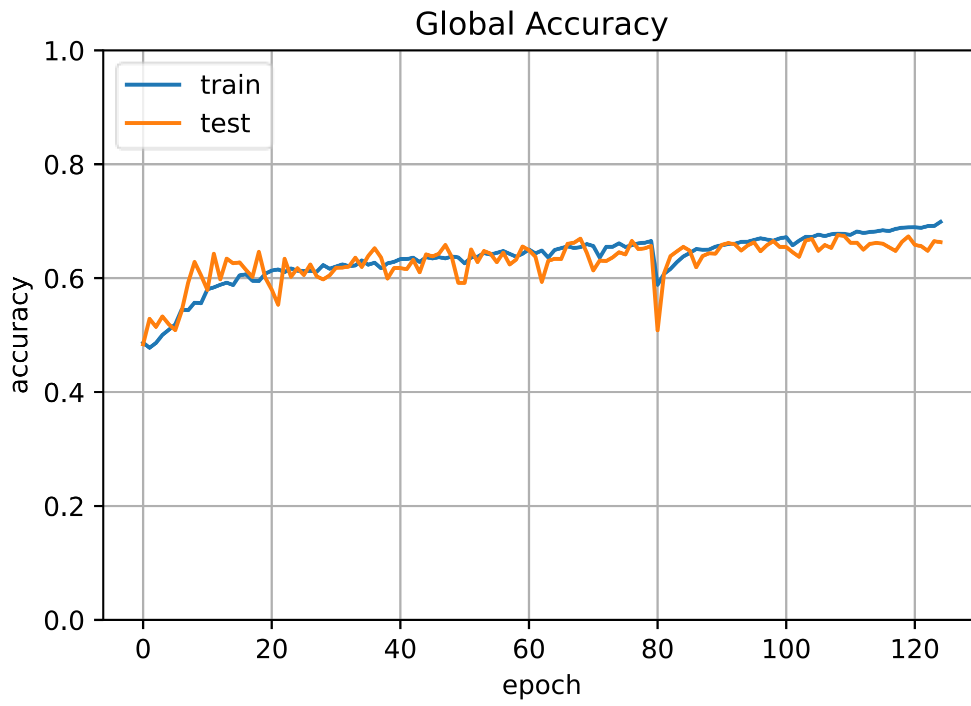
Tabla 5 Parámetros del experimento BASE

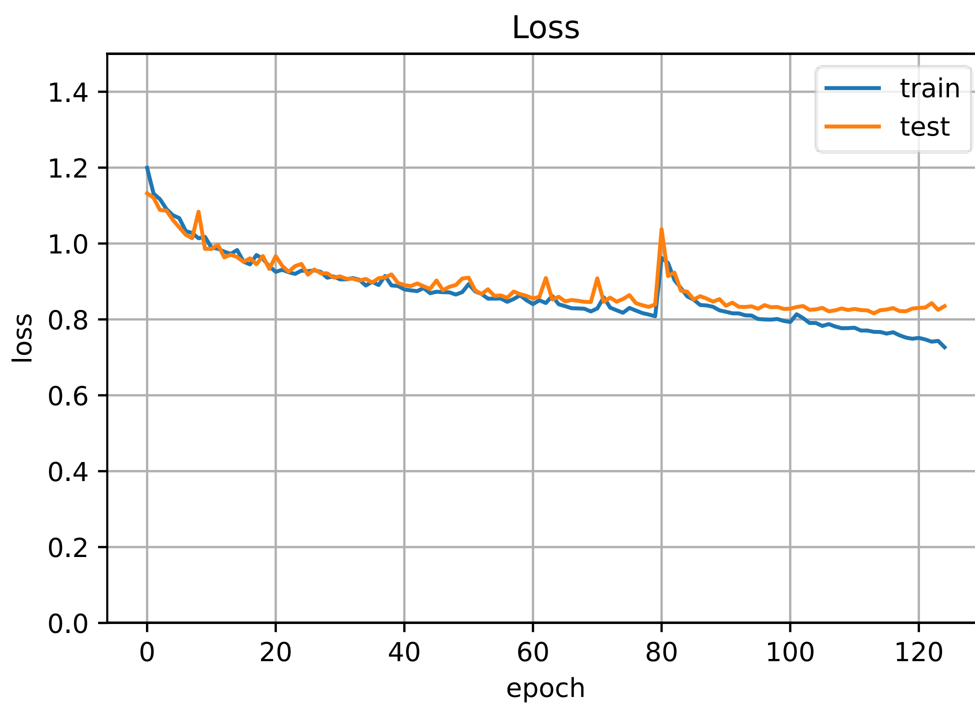
GlobalAccuracy	MeanRecall	MeanPrecision
0.66	0.63	0.57

Tabla 6 Resultados globales BASE

Matriz de confusión						
	# px predichos	PSPAPR	EDZUCA	TAVI	BKG	PRECISION
# px reales ->		**6,887,820**	**1,158,558**	**15,525,451**	**8,540,811**	
PSPAPR	**8,564,301**	4,625,829	189,698	1,082,120	2,666,654	*0.54*
EDZUCA	**4,234,529**	855,677	759,030	1,751,000	868,822	*0.18*
TAVI	**14,739,462**	392,496	147,787	12,506,892	1,692,287	*0.85*
BKG	**4,574,348**	1,013,818	62,043	185,439	3,313,048	*0.72*
RECALL		*0.67*	*0.66*	*0.81*	*0.39*	***0.66***

Tabla 7 Matriz de confusión (BASE)


Global Accuracy
train
test
accuracy
epoch
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120


Loss
train
test
loss
epoch
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120

Imágenes de ejemplo

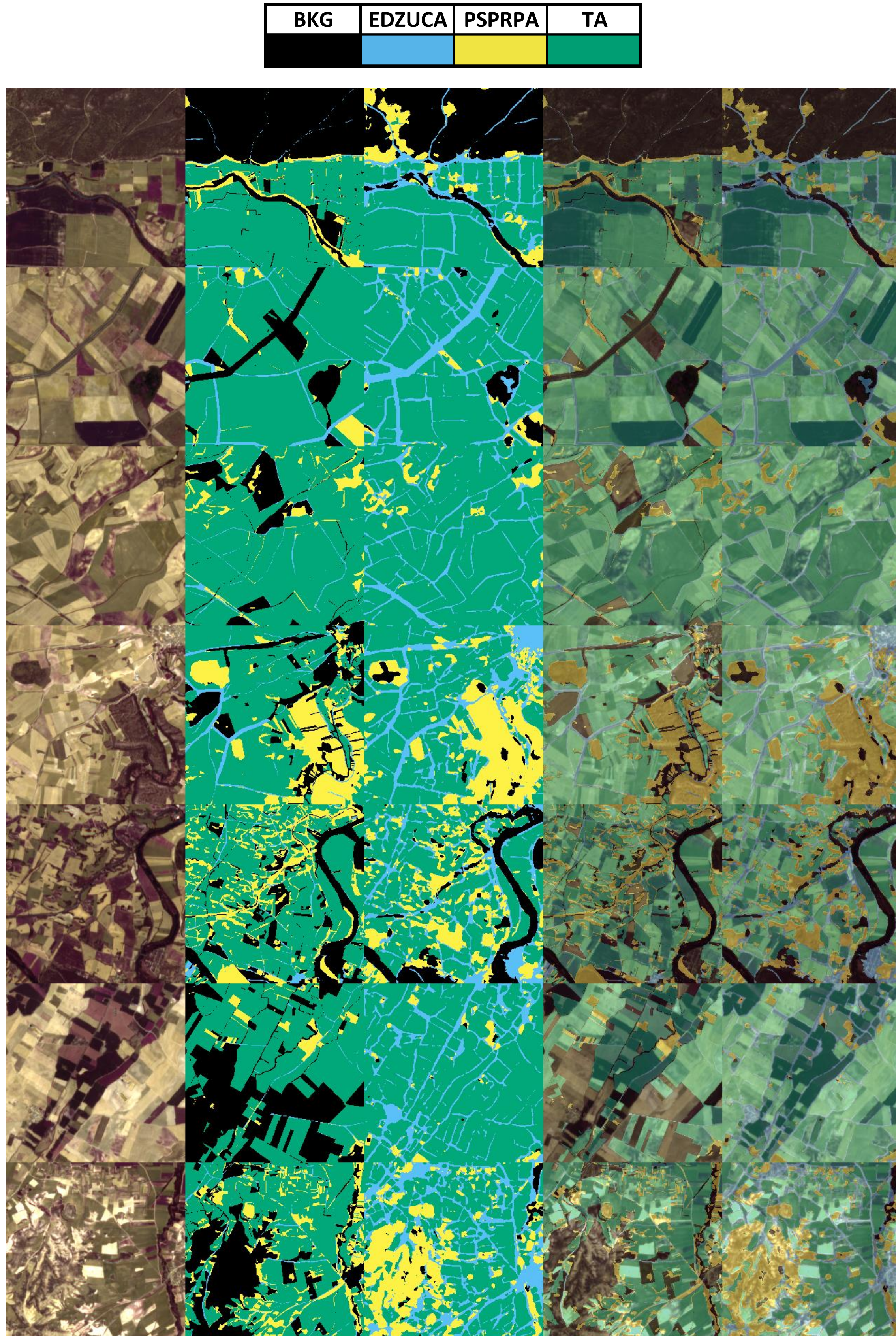


Fig 3 Imágenes de ejemplo (BASE) 1.Original, 2.GroundTruth, 3.Predicción, 4.Original+GT, 5.Original+Predicción

2.6.3. K-002 Variante 1: UpSampling Bilineal

En la sección de "decoder" se utiliza el escalado de upsampling bilineal en lugar de convoluciones transpuestas (también conocidas como up-convolution o reverse convolution) .

Parámetros de entrada	
Parámetros del dataset	
Conjunto de entrenamiento	Train1Aumentado (10 bandas, 3 clases)
Conjunto de test	Test1Aumentado (10 bandas, 3 clases)
Parámetros de la red	
InputSize	[256 256 10]
Número de Clases	4 (3+BKG)
Profundidad de la red	**4**
Dropout	**Si**
BatchNormalization	**No**
Upsampling	**Bilineal**
Numero de filtros (Encoder)	**32**
Padding	Sí
Pesos en las clases	Median Frequency Weighting
Parámetros de entrenamiento	
Solucionador de red (solver network)	Adam
Epochs	125 (Validacion con parada de 30 ep)
BatchSize	32
LearningRate -Inicial **LearnRate-Final**	0.0005 0.0005
Gradient Clipping	-
Regularizacion L2	-
Data Augmentation	No
Shuffle	Si
Momentum	-
Duración del entrenamiento	~01:20:00

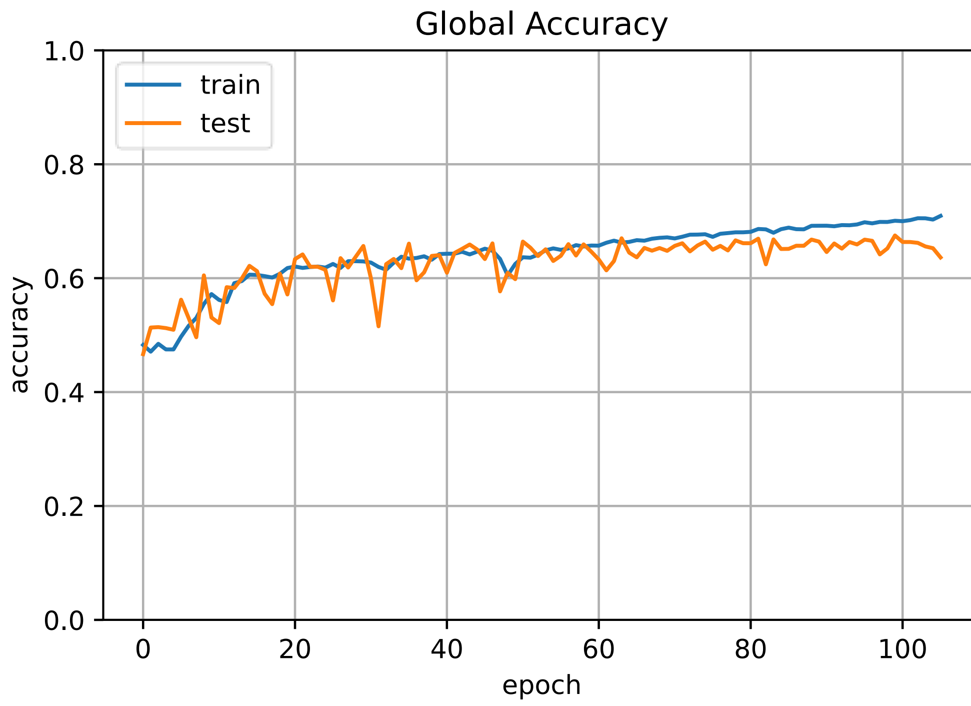
Tabla 8 Parámetros del experimento Variante 1

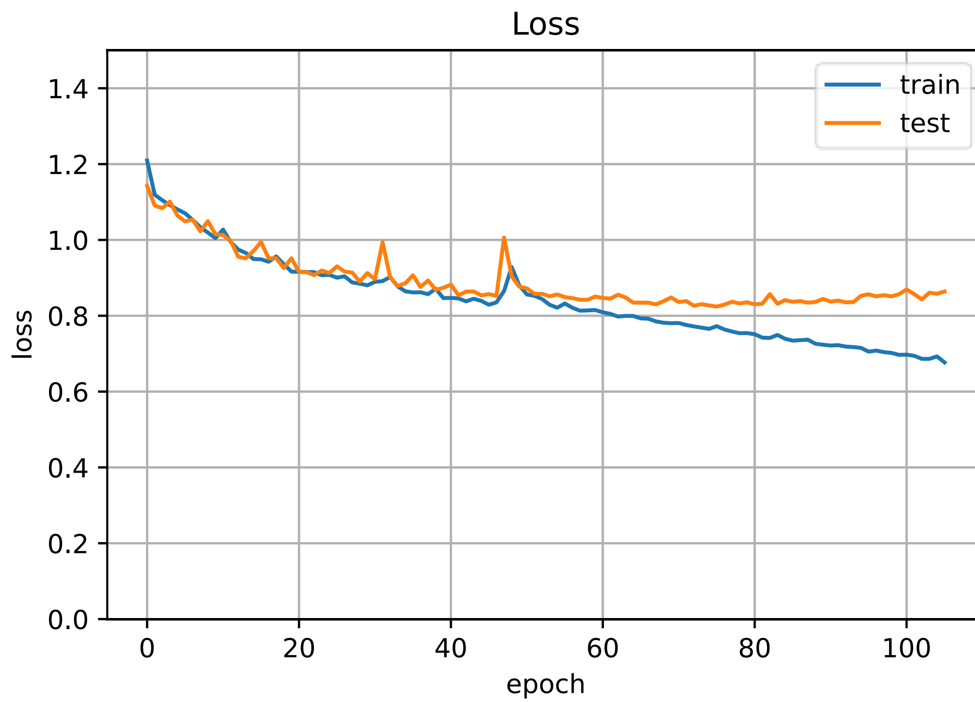
GlobalAccuracy	MeanRecall	MeanPrecision
0.65	0.63	0.56

Tabla 9 Resultados globales Variante1

Matriz de confusión						
	# px predichos	PSPAPR	EDZUCA	TAVI	BKG	PRECISION
# px reales ->		**6,887,820**	**1,158,558**	**15,525,451**	**8,540,811**	
PSPAPR	**8,166,700**	4,383,030	189,607	1,126,715	2,467,348	*0.54*
EDZUCA	**4,550,976**	866,964	772,630	2,003,847	907,535	*0.17*
TAVI	**14,272,895**	380,604	127,630	12,157,308	1,607,353	*0.85*
BKG	**5,122,069**	1,257,222	68,691	237,581	3,558,575	*0.69*
RECALL		*0.64*	*0.67*	*0.78*	*0.42*	***0.65***

Tabla 10 Matriz de confusión (Variante 1)


Global Accuracy
train
test
accuracy
epoch
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100


Loss
train
test
loss
epoch
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100

Imágenes de ejemplo

BKG	EDZUCA	PSPRPA	TA

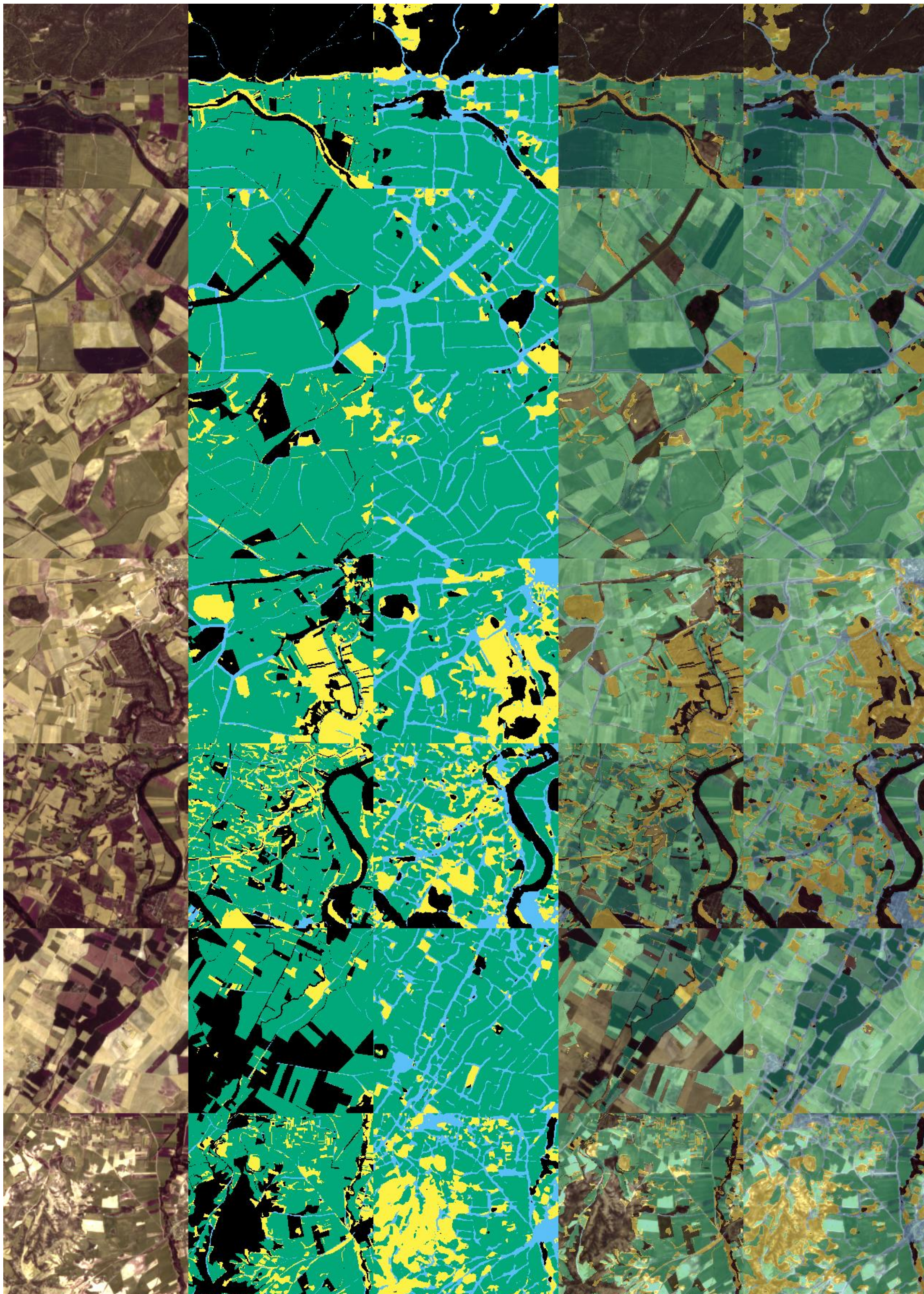


Fig 4 Imágenes de ejemplo (Variante 1) 1.Original, 2.GroundTruth, 3.Predicción, 4.Original+GT, 5.Original+Predicción

2.6.4. K-003 Variante 2: Sin Dropout, UpSampling Bilineal

Sobre la Variante 1 se eliminan las capas de Dropout de la arquitectura de la red.

Parámetros de entrada	
Parámetros del dataset	
Conjunto de entrenamiento	Train1Aumentado (10 bandas, 3 clases)
Conjunto de test	Test1Aumentado (10 bandas, 3 clases)
Parámetros de la red	
InputSize	[256 256 10]
Número de Clases	4 (3+BKG)
Profundidad de la red	**4**
Dropout	**No**
BatchNormalization	**No**
Upsampling	**Bilineal**
Numero de filtros (Encoder)	**32**
Padding	Sí
Pesos en las clases	Median Frequency Weighting
Parámetros de entrenamiento	
Solucionador de red (solver network)	Adam
Epochs	125 (Validacion con parada de 30 ep)
BatchSize	32
LearningRate -Inicial **LearnRate-Final**	0.0005 0.0005
Gradient Clipping	-
Regularizacion L2	-
Data Augmentation	No
Shuffle	Si
Momentum	-
Duración del entrenamiento	~01:20:00

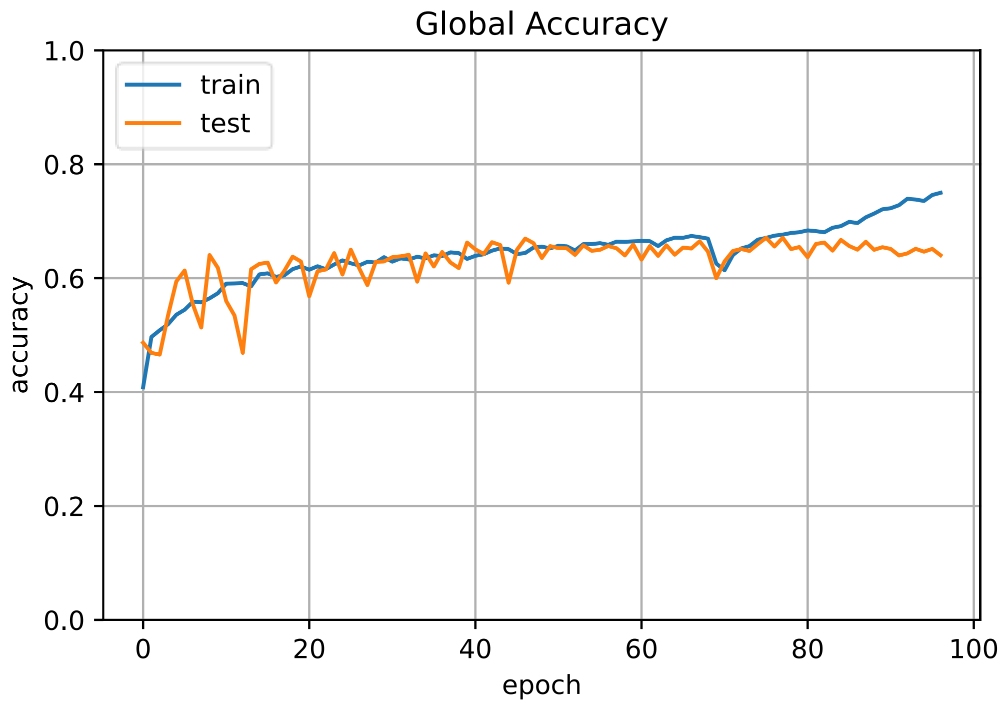
Tabla 11 Parámetros del experimento Variante 2

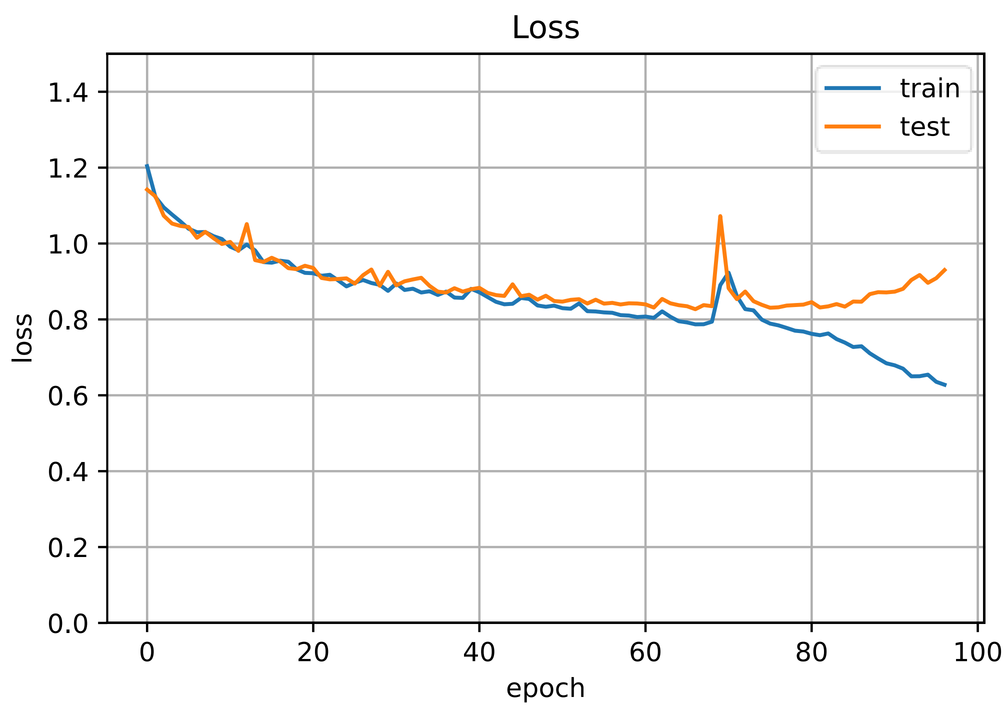
GlobalAccuracy	MeanRecall	MeanPrecision
0.65	0.62	0.56

Tabla 12 Resultados globales Variante 2

Matriz de confusión						
	# px predichos	PSPAPR	EDZUCA	TAVI	BKG	PRECISION
# px reales ->		**6,887,820**	**1,158,558**	**15,525,451**	**8,540,811**	
PSPAPR	**7,524,595**	4,122,326	179,251	1,119,658	2,103,360	*0.55*
EDZUCA	**4,514,418**	869,569	758,694	1,992,805	893,350	*0.17*
TAVI	**14,218,160**	325,382	129,143	12,138,867	1,624,768	*0.85*
BKG	**5,855,467**	1,570,543	91,470	274,121	3,919,333	*0.67*
RECALL		*0.60*	*0.65*	*0.78*	*0.46*	***0.65***

Tabla 13 Matriz de confusión (Variante 2)


Global Accuracy
accuracy
epoch
train
test
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
100


Loss
loss
epoch
train
test
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
20
40
60
80
100

Imágenes de ejemplo

BKG	EDZUCA	PSPRPA	TA

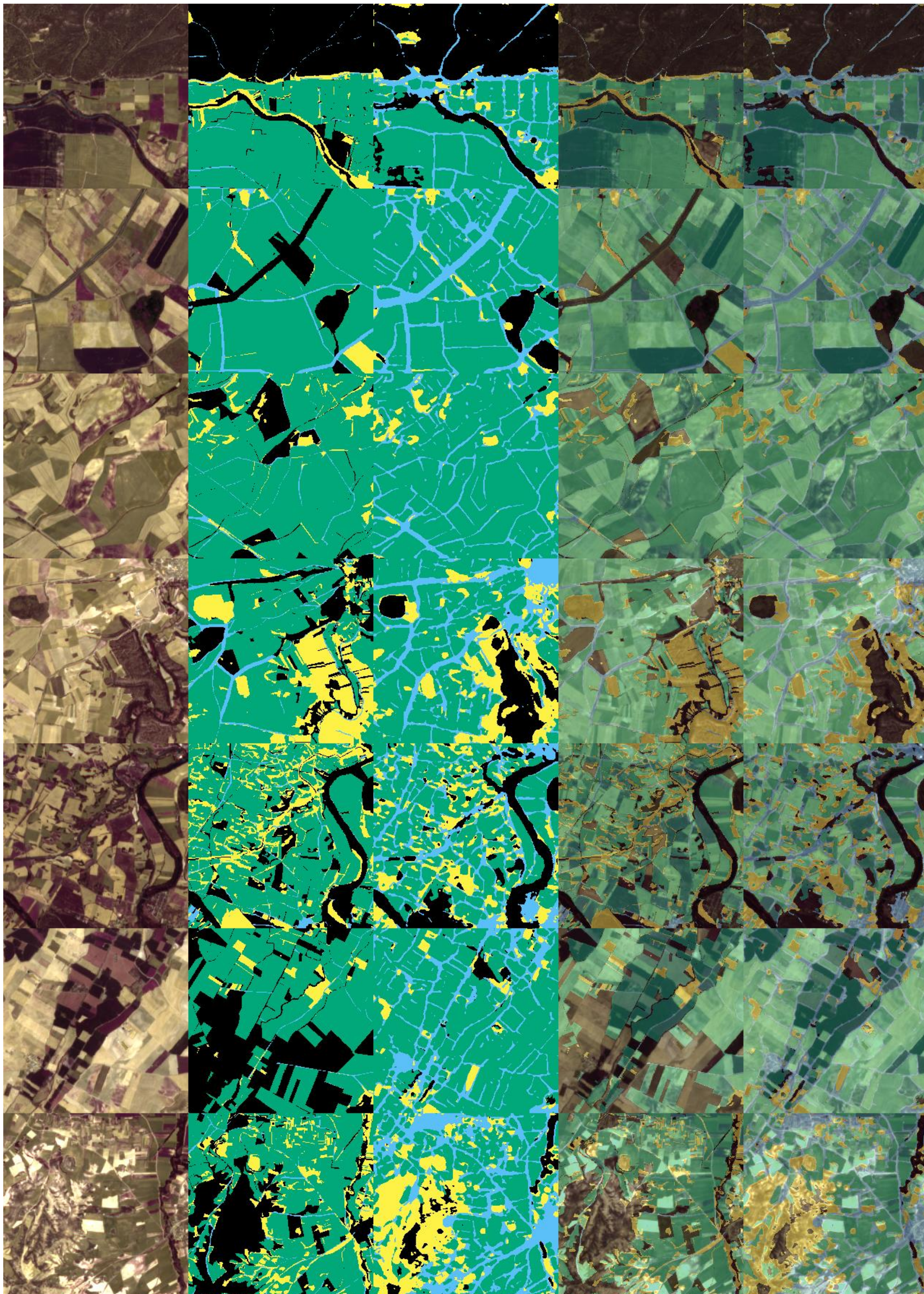


Fig 5 Imágenes de ejemplo (Variante 2) 1.Original, 2.GroundTruth, 3.Predicción, 4.Original+GT, 5.Original+Predicción

2.6.5. K-004 Variante 3: Sin Dropout, UpSampling Bilineal, BatchNormalization

Sobre la Variante 2 se añaden capas de BatchNormalization después de cada convolución.

Parámetros de entrada	
Parámetros del dataset	
Conjunto de entrenamiento	Train1Aumentado (10 bandas, 3 clases)
Conjunto de test	Test1Aumentado (10 bandas, 3 clases)
Parámetros de la red	
InputSize	[256 256 10]
Número de Clases	4 (3+BKG)
Profundidad de la red	**4**
Dropout	**No**
BatchNormalization	**Si**
Upsampling	**Bilineal**
Numero de filtros (Encoder)	**32**
Padding	Sí
Pesos en las clases	Median Frequency Weighting
Parámetros de entrenamiento	
Solucionador de red (solver network)	Adam
Epochs	125 (Validacion con parada de 30 ep)
BatchSize	32
LearningRate -Inicial **LearnRate-Final**	0.0005 0.0005
Gradient Clipping	-
Regularizacion L2	-
Data Augmentation	No
Shuffle	Si
Momentum	-
Duración del entrenamiento	~00:40:00

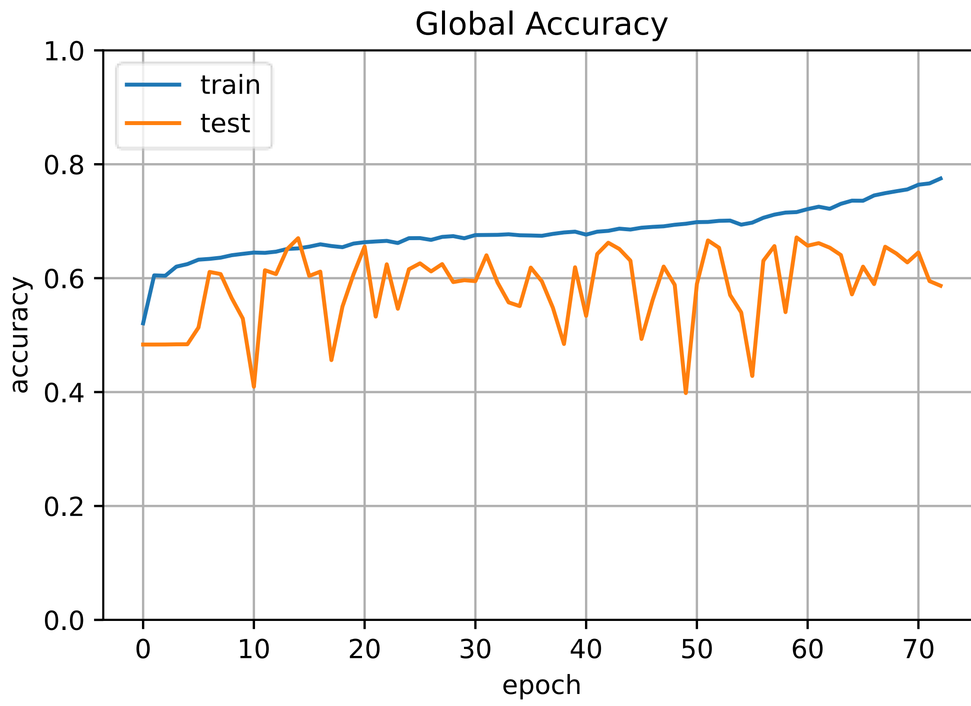
Tabla 14 Parámetros del experimento Variante 3

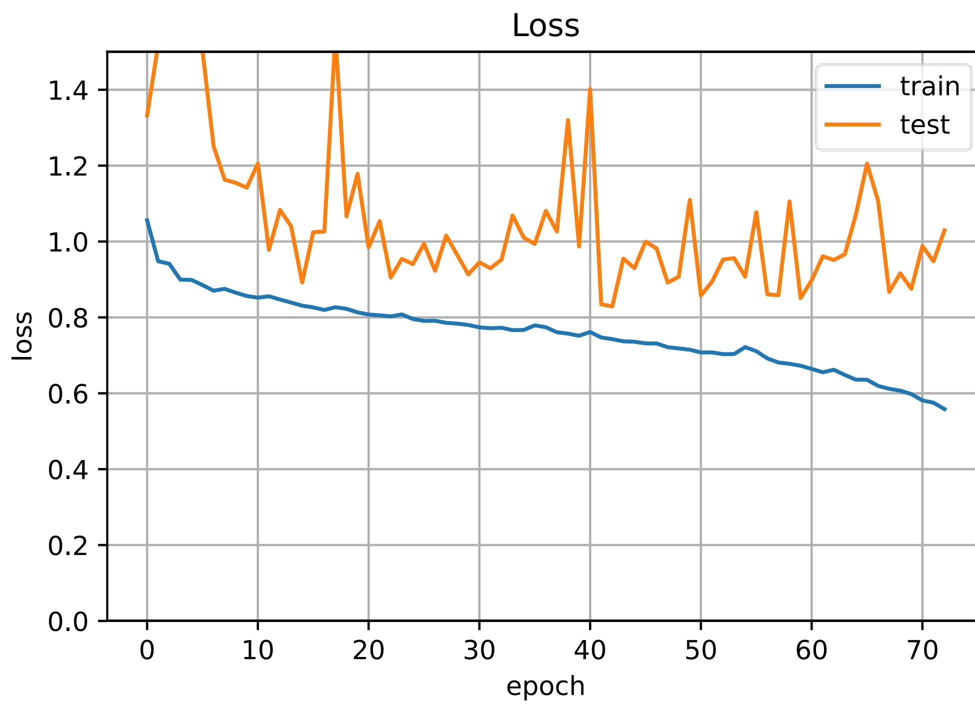
GlobalAccuracy	MeanRecall	MeanPrecision
0.66	0.63	0.56

Tabla 15 Resultados globales Variante 3

Matriz de confusión						
	# px predichos	PSPAPR	EDZUCA	TAVI	BKG	PRECISION
# px reales ->		**6,887,820**	**1,158,558**	**15,525,451**	**8,540,811**	
PSPAPR	**6,307,741**	3,702,972	134,661	543,389	1,926,719	*0.59*
EDZUCA	**3,807,163**	768,419	780,911	1,435,531	822,302	*0.21*
TAVI	**15,493,664**	821,315	160,094	12,750,963	1,761,292	*0.82*
BKG	**6,504,072**	1,595,114	82,892	795,568	4,030,498	*0.62*
RECALL		*0.54*	*0.67*	*0.82*	*0.47*	***0.66***

Tabla 16 Matriz de confusión (Variante 3)


Global Accuracy
train
test
accuracy
epoch
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60
70


Loss
train
test
loss
epoch
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60
70

Imágenes de ejemplo

BKG	EDZUCA	PSPRPA	TA

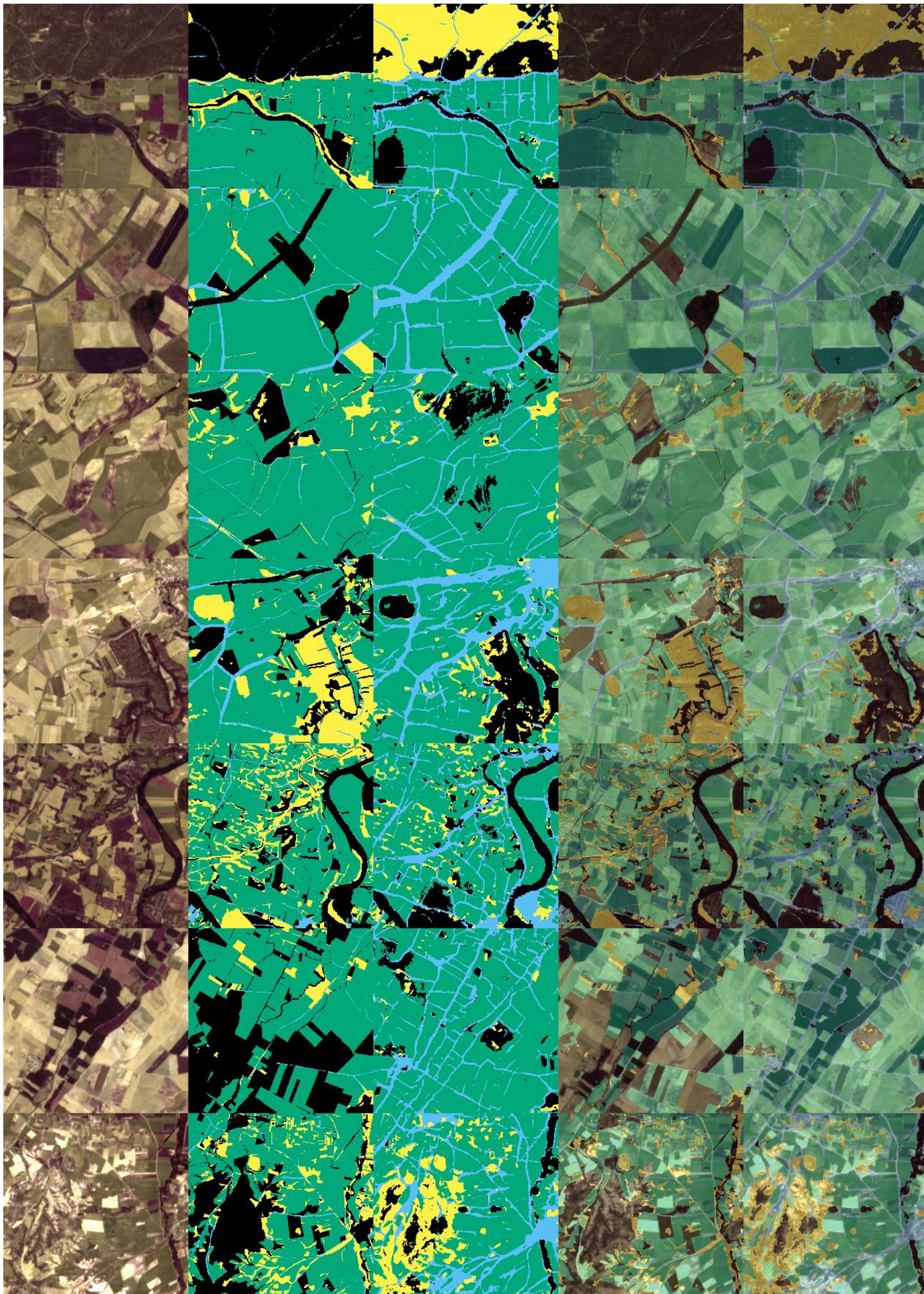


Fig 6 Imágenes de ejemplo (Variante 3) 1.Original, 2.GroundTruth, 3.Predicción, 4.Original+GT, 5.Original+Predicción

2.6.6. E011-11 Matlab NO-L2R

En este apartado se muestran los resultados del experimento BASE en la **implementación de Matlab sin L2Regularization**.

Parámetros de entrada	
Parámetros del dataset	
Conjunto de entrenamiento	Train1Aumentado (10 bandas, 3 clases)
Conjunto de test	Test1Aumentado (10 bandas, 3 clases)
Parámetros de la red	
InputSize	[256 256 10]
Número de Clases	4 (3+BKG)
Profundidad de la red	**4**
Dropout	**Si**
BatchNormalization	**No**
Upsampling	**Convolución Transpuesta (2,2)**
Numero de filtros (Encoder)	**32**
Padding	Sí
Pesos en las clases	Median Frequency Weighting
Parámetros de entrenamiento	
Solucionador de red (solver network)	Adam
Epochs	125
BatchSize	32
LearningRate -Inicial **LearnRate-Final**	0.0005 0.0005
Gradient Clipping	1
Regularizacion L2	0
Data Augmentation	No
Shuffle	Si
Momentum	-
Duración del entrenamiento	01:09:05

Tabla 17 Parámetros del experimento MATLAB E011-11

GlobalAccuracy	MeanRecall	MeanPrecision
0.66	0.61	0.55

Tabla 18 Resultados globales (MATLAB E011-11)

Matriz de confusión						
	# px predichos	PSPRPA	EDZUCA	TAVI	BACKGROUND	PRECISION
# px reales		**6,887,820**	**1,158,558**	**15,525,451**	**8,540,811**	
PSPRPA	**7,212,091**	3,924,493	197,286	887,503	2,202,809	*0.54415*
EDZUCA	**3,459,844**	665,770	677,049	1,431,341	685,684	*0.19569*
TAVI	**14,880,503**	488,320	156,004	12,530,590	1,705,589	*0.84208*
BACKGROUND	**6,560,202**	1,809,237	128,219	676,017	3,946,729	*0.60162*
RECALL		*0.5698*	*0.5844*	*0.8071*	*0.4621*	**0.65640**

Tabla 19 Matrix de confusión (MATLAB E011-10)

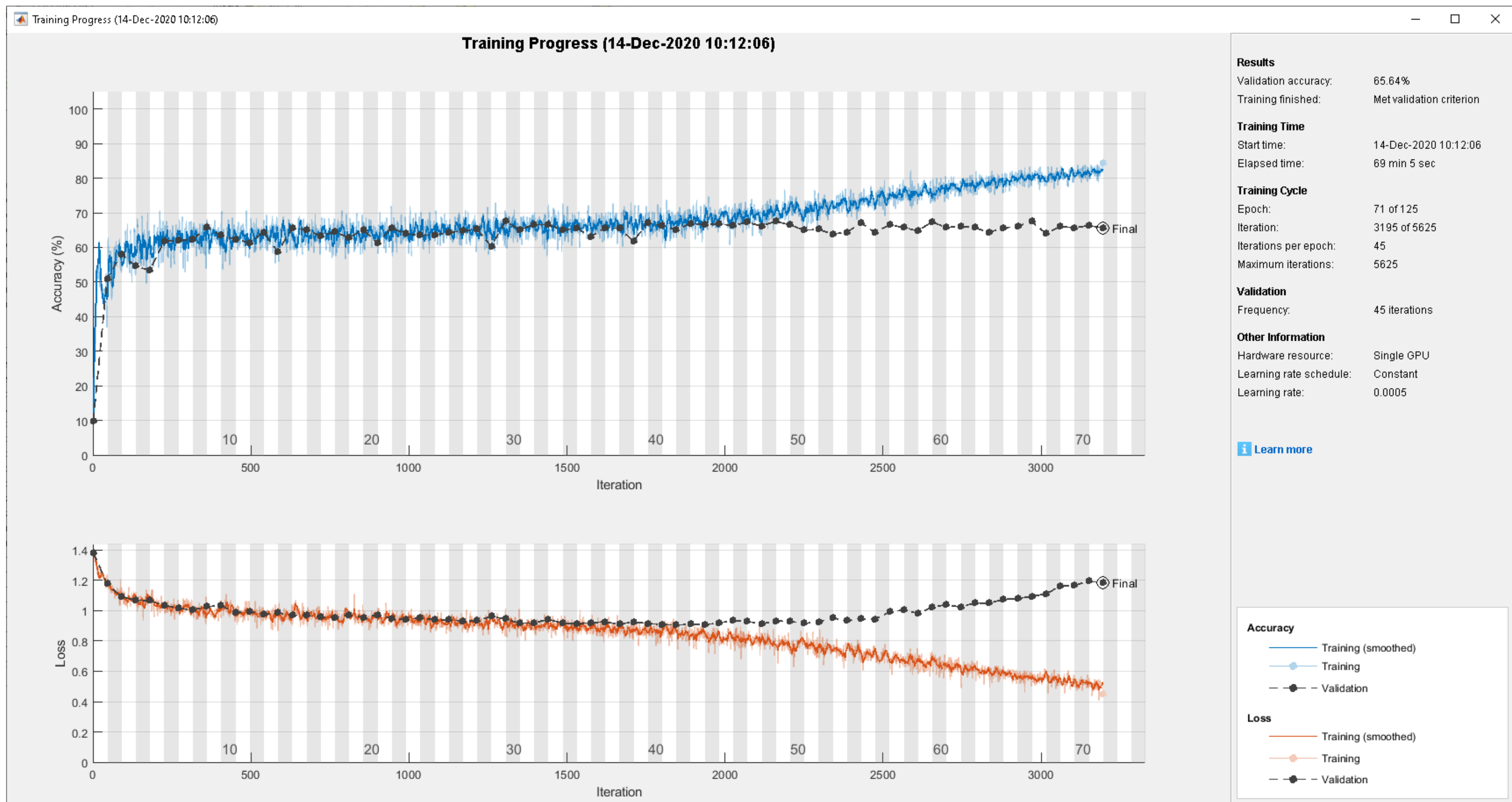


Fig 7 Función de Loss (E011-11)

Imágenes de ejemplo

BKG	EDZUCA	PSPRPA	TA

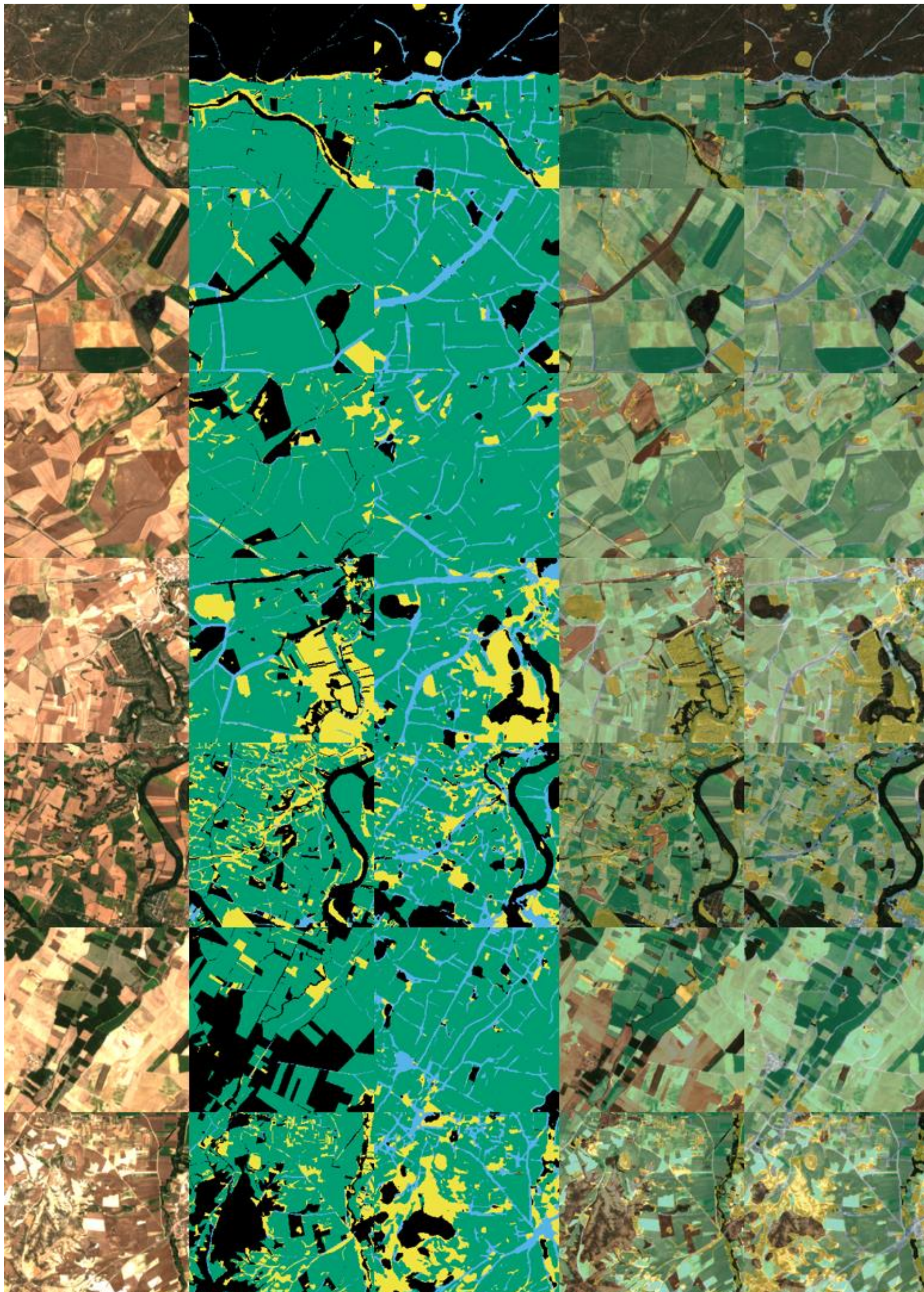


Fig 8 Imágenes de ejemplo (MATLAB E011-11) 1.Original, 2.GroundTruth, 3.Predicción, 4.Original+GT, 5.Original+Predicción

2.6.7. K-005 Arquitectura Matlab en Keras L2R

En este apartado se muestran los resultados del experimento K-005 en la **implementación Keras**. Este experimento es similar al K-001 (BASE), a excepción de que **se implementa la regularización L2 con un valor de 0.0001.**

Parámetros de entrada	
Parámetros del dataset	
Conjunto de entrenamiento	Train1Aumentado (10 bandas, 3 clases)
Conjunto de test	Test1Aumentado (10 bandas, 3 clases)
Parámetros de la red	
InputSize	[256 256 10]
Número de Clases	4 (3+BKG)
Profundidad de la red	**4**
Dropout	**Si**
BatchNormalization	**No**
Upsampling	**Convolución Transpuesta (2,2)**
Numero de filtros (Encoder)	**32**
Padding	Sí
Pesos en las clases	Median Frequency Weighting
Parámetros de entrenamiento	
Solucionador de red (solver network)	Adam
Epochs	125 (Validacion con parada de 30 ep)
BatchSize	32
LearningRate -Inicial **LearnRate-Final**	0.0005 0.0005
Gradient Clipping	-
Regularizacion L2	-
Data Augmentation	No
Shuffle	Si
Momentum	-
Duración del entrenamiento	00:26:47

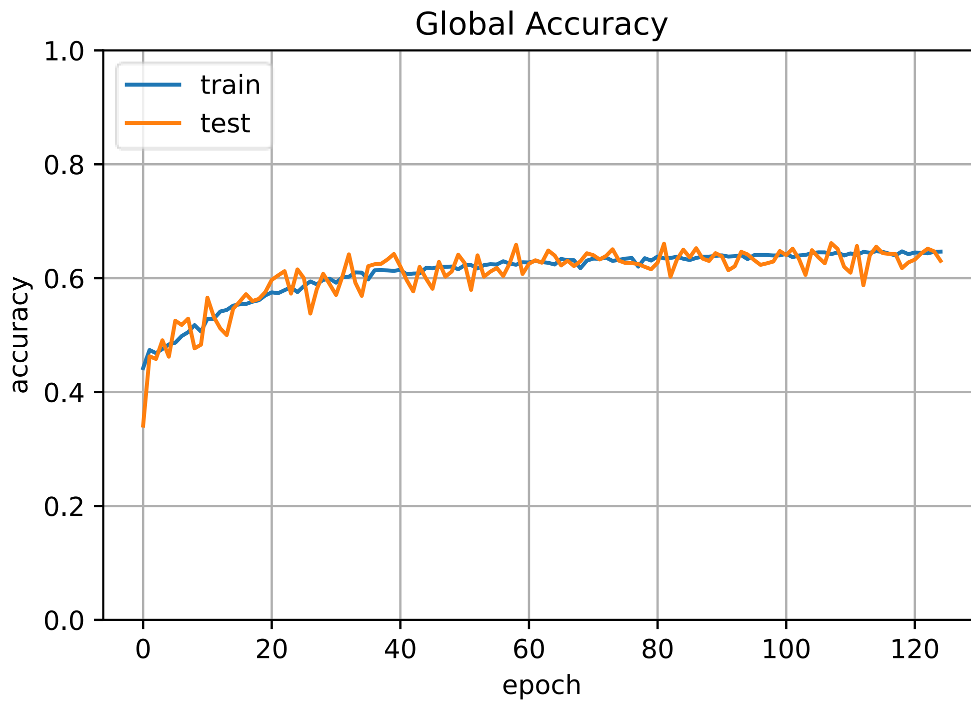
Tabla 20 Parámetros del experimento K-005

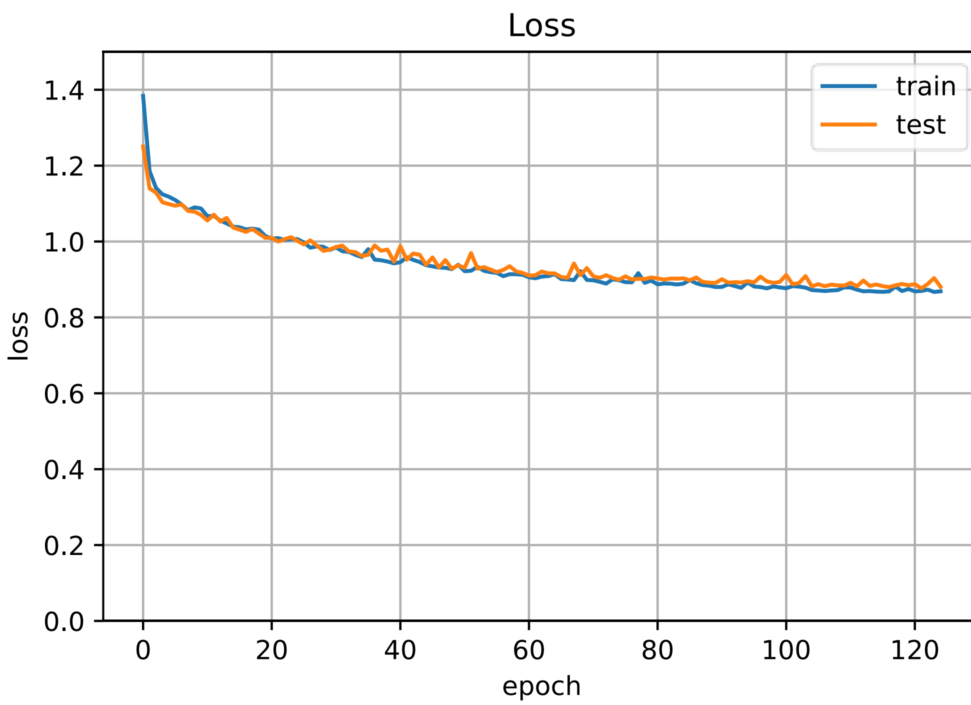
GlobalAccuracy	MeanRecall	MeanPrecision
0.64	0.60	0.56

Tabla 21 Resultados globales K-005

Matriz de confusión						
	# px predichos	PSPAPR	EDZUCA	TAVI	BKG	PRECISION
# px reales ->		**6,887,820**	**1,158,558**	**15,525,451**	**8,540,811**	
PSPAPR	**9,185,962**	4,641,502	252,030	1,311,295	2,981,135	*0.51*
EDZUCA	**3,986,560**	715,348	674,389	1,814,839	781,984	*0.17*
TAVI	**14,576,102**	410,240	165,763	12,286,161	1,713,938	*0.84*
BKG	**4,364,016**	1,120,730	66,376	113,156	3,063,754	*0.70*
RECALL		*0.67*	*0.58*	*0.79*	*0.36*	***0.64***

Tabla 22 Matriz de confusión (K-005)


Global Accuracy
train
test
accuracy
epoch
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120


Loss
train
test
loss
epoch
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120

Red UNET

Imágenes de ejemplo

BKG	EDZUCA	PSPRPA	TA

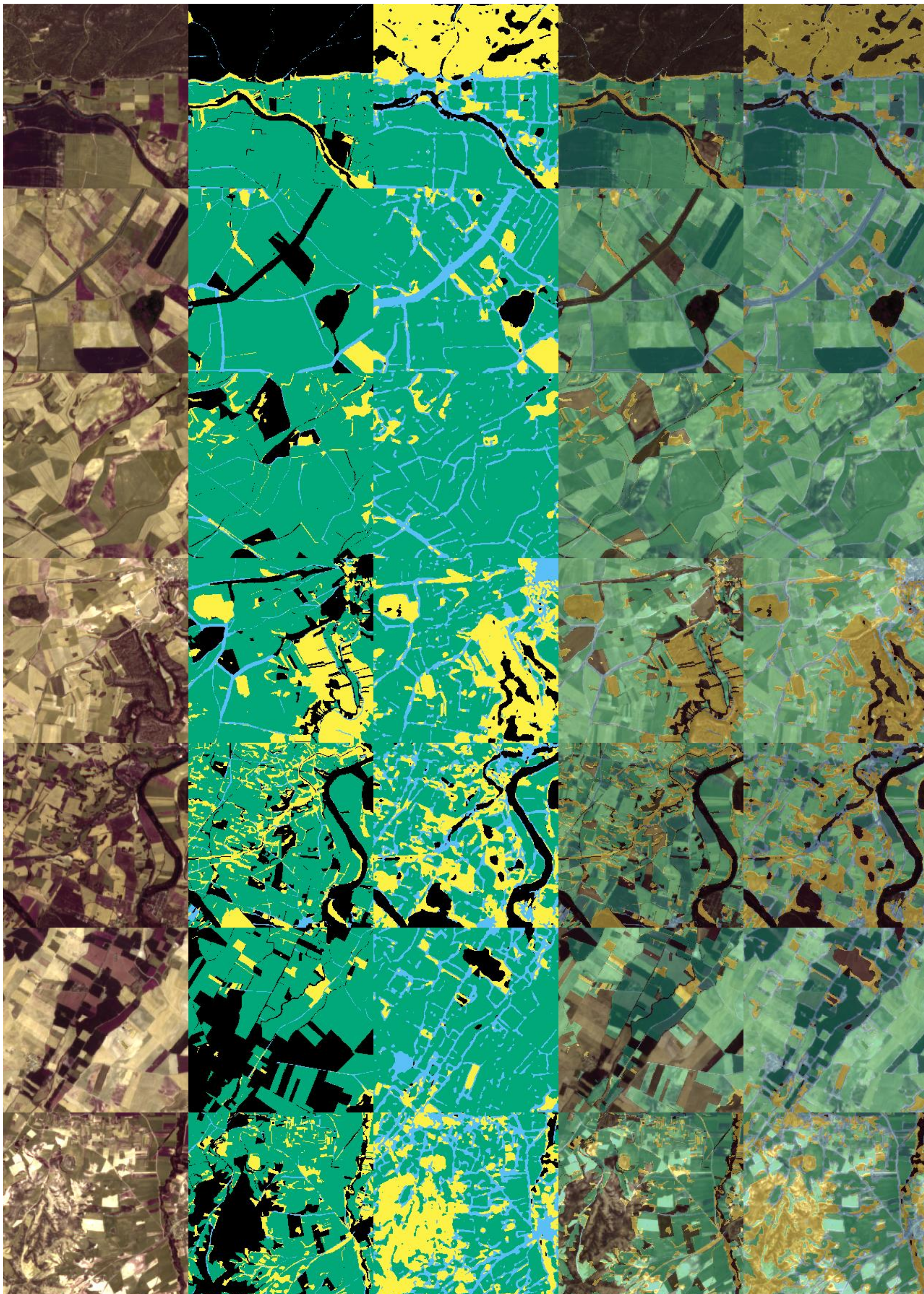


Fig 9 Imágenes de ejemplo (BASE) 1.Original, 2.GroundTruth, 3.Predicción, 4.Original+GT, 5.Original+Predicción