

# Clasificación de cultivos utilizando imágenes satelitales mediante redes neuronales recurrentes-convolucionales.

Santiago Luciano Zúñiga\*

Diciembre de 2020

## Resumen

El presente trabajo se expone la entrada realizada en la competencia [Clasificación de cultivos utilizando imágenes satelitales](#), organizada por [Matba Rofex](#), [Bolsa de Cereales de Buenos Aires](#), [Bolsa de Comercio de Rosario](#) y [Meta:Data](#), finalizado el 14 de Diciembre de 2020. La consigna de la misma, extraído de la web oficial, es (...) construir un método de aprendizaje supervisado que mediante imágenes satelitales y la verdad de campo de cosechas previas pueda clasificar cultivos para distintos puntos geográficos del departamento de General López, Santa Fé. Se diseño una red neuronal profunda consistiendo de capas recurrentes LSTM (long short-term memory), densas y convolucionales, obteniendo un *balanced accuracy* final de 61.1 %, calculado sobre la totalidad del set de testing. Todo el código y base de datos usados se encuentra en [GitHub](#).

## Introducción y breve reseña de métodos existentes

Para resolver el problema de clasificación y predicción de cultivos, partiendo de imágenes satelitales u otras fuentes, los métodos de *machine-learning* han ganado gran popularidad en los últimos años, dado el avance tecnológico y el fácil acceso a la información y al hardware requerido. Entre ellos, los más comunes son *Support Vector Machine* (SVM) [[Mathur and Foody, 2008](#)]; métodos basados en arboles de decisión [[Friedl and Brodley, 1997](#)] como *Random Forest* (RF) [[Tatsumi et al., 2015](#)] o el método estado-del-art *eXtreme Gradient Boosting* (XGBoost) [[Saini and Ghosh, 2019](#)]; *k-Nearest Neighbors* (*k*-NN) [[Ahmad et al., 2011](#)]; principio de máxima verosimilitud [[Murthy et al., 2003](#)]; y *redes neuronales artificiales* (ANN). De este grupo, las redes neuronales resultan ser el sector del *machine-learning* que más ha crecido en los últimos años, con nuevos modelos matemáticos y aplicaciones de software apareciendo cada año. Debido a su versatilidad y amplio lugar para la innovación es el camino que fue elegido para el presente trabajo. Sin buscar ser exhaustivos, podemos dar un breve resumen las distintas opciones de arquitecturas de redes neuronales para resolver el problema de clasificación de cultivos a partir de imágenes satelitales:

**Métodos basados en clasificación de imágenes espectrales** Se trata al problema como si fuera un problema clásico de clasificación de imágenes, generalmente usando arquitecturas ya preentrenadas como VGG, GoogleNet (Inception Net), ResNet, etc. y luego realizar *transfer learning* y *fine tuning*, [[Alhassan et al., 2020](#), [Yang et al., 2020](#)].

**Métodos basados en clasificación de imágenes espectrales con evolución temporal** Similares al grupo anterior, pero se agrega una capa Conv3D a la entrada para estudiar la evolución temporal de las imágenes, [[Ji et al., 2018](#)].

**Métodos basados en transformers** Heredando métodos usados en *natural language processing* (NLP), se busca estudiar los patrones espacio-espectrales de las imágenes, [[Li et al., 2020](#)].

**Métodos basados en segmentación semántica espacial** Se procesan las imágenes espectrales y se busca una segmentación semántica como si fueran imágenes comunes, utilizando arquitecturas como U-Net, [[Rustowicz et al., 2019](#)].

---

\*[slzunigap@gmail.com](mailto:slzunigap@gmail.com)

**Métodos basados en las series temporales a nivel pixel** En este caso se toman las series temporales de una o más bandas para un pixel o lugar geográfico en particular. Las arquitecturas pueden estar basadas en capas Conv1D, RNN, LSTM o una combinación de estas, [Campos-Taberner et al., 2020, Zhong et al., 2019, Mazzia et al., 2019, Xu et al., 2020].

Luego de estudiar cuidadosamente los datos de entrenamiento suministrados, se decidió seguir, de estos grandes grupos, el de utilizar las series temporales de cada punto geográfico, sin considerar información espacial. El modelo planteado se basa en capas LSTM-FC-CNN-FC y se describirá en detalle más adelante. Para el entrenamiento y predicción se necesitan extraer para cada punto suministrado las series temporales correspondientes a las bandas espectrales del satélite elegido. Un *constraint* importante para el diseño del algoritmo realizado fue que los datos requieran una cantidad mínima de preprocesamiento y que el modelo final fuera lo más general posible, sin necesitar separar los datos por campaña u otras clasificaciones previas. Si bien esto evidentemente resulta en menor precisión final, se obtiene un algoritmo más robusto y fácil de emplear *en campo*, es decir, fuera del ambiente académico de la competencia donde realmente pueda ser necesario.

## Área de interés y preprocesamiento de datos

Los datos de entrenamiento suministrados se hallan en el departamento de General López, provincia de Santa Fe, Argentina ( $11.558 \text{ km}^2$ ), correspondiendo a relevamientos realizados por la Bolsa de Cereales de Buenos Aires<sup>1</sup> y la Bolsa de Comercio de Rosario<sup>2</sup> durante las campañas 2010/2019 y 2019/2020. Los mismos, junto al cultivo declarado, se pueden ver en la Fig. 1. Es importante destacar que las clases provistas (14 en total) se encuentran sumamente desbalanceadas, como se puede inferir de la Tabla 1. Además, existía un dato de clase "S/M" (Soja o Maíz) que se decidió ignorar dada su ambigüedad. Por su parte, se proveen 555 datos en el set de validación, cuya correspondiente etiqueta de cultivo fue provista al terminar la competencia.

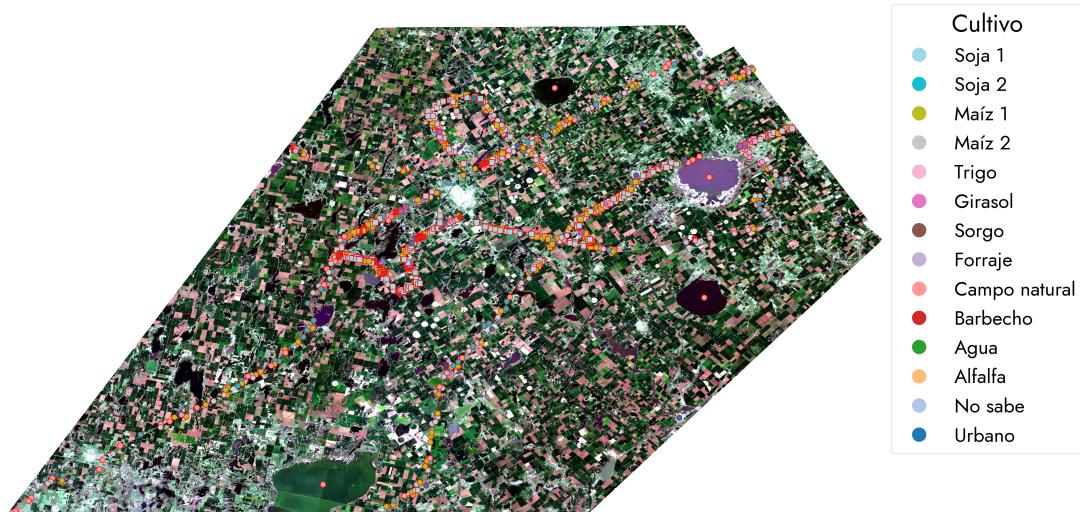


Figura 1: Datos de entrenamiento provistos, correspondientes a la campaña 2018/2019 (○) y 2019/2020 (□). Imagen de fondo capturada en Febrero de 2020, satélite Sentinel 2, bandas RGB.

Como se dijo anteriormente, para la clasificación se utilizará la serie temporal entre los meses de Septiembre y Abril para cada punto de interés. Este periodo de meses se eligió por comprender los periodos de plantación, crecimiento y cosecha de los principales cultivos. Los datos provistos solo dan información de su posición geográfica y campaña, dejando libre la elección de la base de datos satelital a utilizar. Se decidió, por su frecuencia temporal, utilizar la provista por los satélites Sentinel 2 A/B, [Segarra et al., 2020]. Se usó la colección Level-1C, es decir, sin corrección por refractancia atmosférica, ya que la versión corregida (Level-2A) no

<sup>1</sup>[www.bolsadecereales.com](http://www.bolsadecereales.com)

<sup>2</sup>[www.bcr.com.ar](http://www.bcr.com.ar)

Cultivo	2018/2019	2019/2020	Total [ % ]
Soja 1	155	189	40.51
Maíz 1	82	128	24.73
Soja 2	1	88	10.48
Campo natural	25	57	9.65
Forraje	10	45	6.47
No sabe	9	25	4.00
Urbano	-	12	1.41
Barbecho	-	6	0.70
Sorgo	6	-	0.70
Maíz 2	1	3	0.47
Agua	1	1	0.23
Alfalfa	2	-	0.23
Trigo	-	2	0.23
Girasol	1	-	0.11
<b>Total</b>	<b>293</b>	<b>556</b>	<b>100</b>

Tabla 1: Resumen de la cantidad de datos provistos en el set de entrenamiento para cada tipo de cultivo, para cada campaña y en total.

se encontraba disponible para los meses de Septiembre a Diciembre de 2018. Si bien esto puede introducir errores, especialmente en la banda azul, se decidió no generar los datos faltantes de Level-2A y utilizar en su lugar Level-1C para todos los puntos de entrenamiento, con el objetivo de que la generación de las series temporales sea sencilla y fácil de aplicar en campo. Para la obtención de los datos se utilizó la API de Python de Google Earth Engine <sup>3</sup> (GEE). Con los datos de entrenamiento y validación se generaron sendos tensores  $X_{train} \in \mathcal{R}^{itr,t,b}$  y  $X_{test} \in \mathcal{R}^{ite,t,b}$ , donde  $itr$  e  $ite$  son el número de puntos geográficos de entrenamiento y testing respectivamente,  $t$  el número de muestras temporales (48 en este caso) y  $b$  el número de bandas empleadas (6: R, G, B, NIR, SWIR 1 y SWIR 2). Por lo tanto, los tensores finalmente tendrán tamaños (849, 48, 6) para entrenamiento y (555, 48, 6) para testing. Se puede observar el procedimiento en el archivo `createTrain.py` para el set de entrenamiento y `createTest.py` para el set de validación. Se puede resumir el algoritmo en los siguientes pasos:

---

#### Algoritmo 1: Obtención y preprocesamiento de datos

---

1. Generar la lista de coordenadas de interés.
2. Generar la colección (`ee.ImageCollection`) de imágenes de GEE, colección COPERNICUS/S2, entre el 1ro de Septiembre de 2018/19 y el 30 de Abril 2019/20 según corresponda.
3. Descargar para cada punto la serie temporal de las bandas R, G, B, NIR, SWIR 1, SWIR 2 y máscara de nubes de 60 metros (bandas B2, B3, B4, B5, B8, B11, B12, QA60). Para esto se mapea la función `get_data` a toda la colección de imágenes mediante `imageCollection.map()`. Se realiza un promediado en un área de 100m de diámetro para disminuir el ruido. Luego con la función `get_pd` se genera una base de datos de pandas para cada punto.
4. En la función `get_X` se descartan los puntos con precedencia de nubes y se interpola su valor en cada banda utilizando los valores anteriores y posteriores. Luego se descartan los valores repetidos entre los satélites A y B, quedando 48 muestras temporales para cada punto espacial.
5. Se generan y guardan los arrays de numpy de las series temporales y de las etiquetas, si es el set de entrenamiento.

Como se puede observar el preprocesamiento es muy sencillo, solo eliminando las muestras con cobertura de nubes y reemplazándolas por una interpolación lineal. Se decidió no agregar más preprocesamiento, como filtros de envolvente positiva, de Savitzky-Golay u otros similares para mantener el proceso sencillo y fácil de replicar en campo. Además se decidió no separar

---

<sup>3</sup>[www.earthengine.google.com](http://www.earthengine.google.com)

los datos de las campañas 2018/2019 y 2019/2020, a pesar de que hubiera sido lo correcto (de la tabla 1 es claro que, por ejemplo, la proporción de campos con soja de segunda es mucho menor en una campaña). Esta decisión tuvo dos fundamentos: el primero, querer hacer un algoritmo *general*, que solo requiera las series temporales de las bandas de Sentinel 2 Level-1C y ninguna información adicional. El segundo motivo es la poca cantidad de datos de entrenamiento suministrados, con lo que segregarlos por campaña solo lograría disminuirlos aún más.

Resulta útil observar la evolución de las series temporales para las distintas clases. A fin de facilitar la visualización, se calculó el índice de vegetación de diferencia normalizada, *NDVI*, definido como

$$NDVI = \frac{NIR - R}{NIR + R}, \quad (1)$$

donde NIR y R son las bandas del infrarrojo cercano y rojas, respectivamente. El valor del NDVI es proporcional al contenido de clorofila de la vegetación [Liang et al., 2012]. Otras superficies, como agua o suelo, darán valores muy pequeños o incluso negativos. En la Fig. 2 se muestra la evolución temporal del mismo para cada clase en todos los puntos geográficos suministrados en el set de entrenamiento, junto a la media. Se observa, por ejemplo para las sojas de primera y segunda, una clara tendencia temporal correspondiente a las fechas de cultivo, crecimiento y cosecha. En otros casos, como campo natural o forraje, los datos son mucho más variados y erráticos, producto de la diversidad de estos cultivos. Se espera que al proveerle a la red neuronal todas las bandas ésta aprenda a discernir las sutilizas de cada clase y generar las relaciones no lineales entre las bandas que sean necesarias.

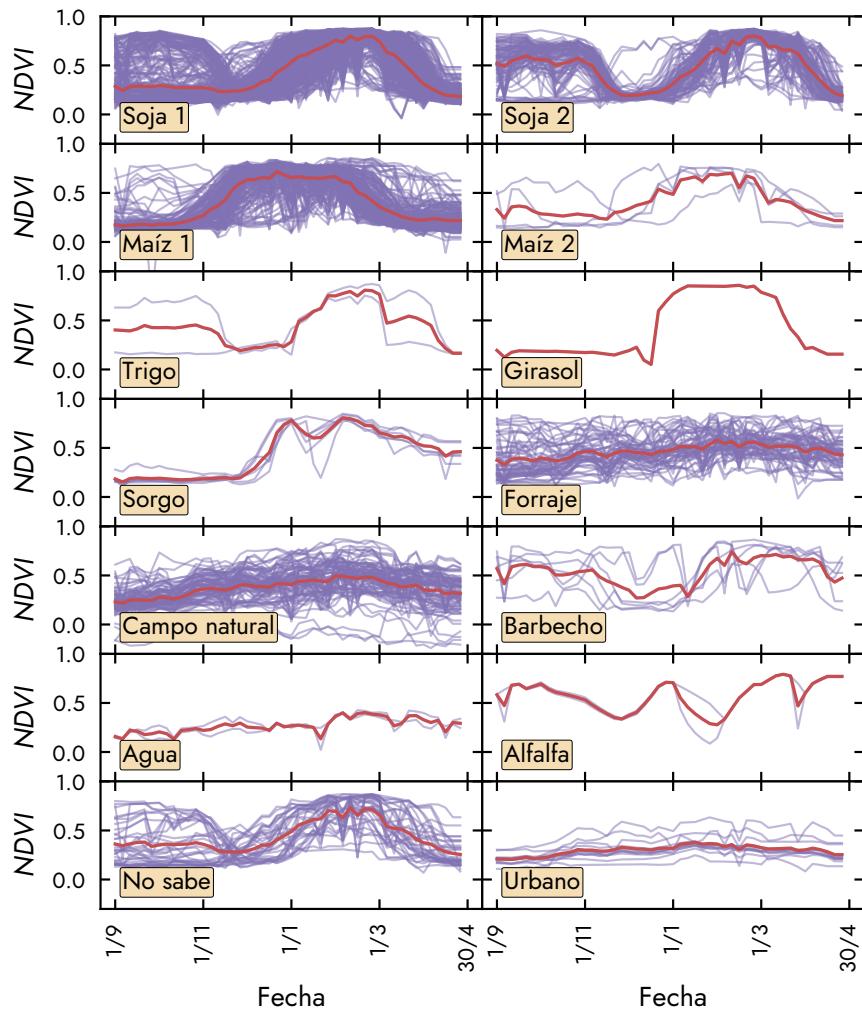


Figura 2: Evolución del NDVI para cada clase. Cada punto provisto corresponde a una línea violeta. En rojo, la media.

Finalmente, para la última etapa de entrenamiento se decidió realizar *data augmentation*

sobre los datos de entrenamiento. Para ésto se utilizó el paquete `ttsaug`<sup>4</sup>. Específicamente, la *augmentation* elegida fue *time warping*, también conocida como *window warping* [Guennec et al., 2016], dada la naturaleza de los datos utilizados (no son reversibles, pocos datos para agregar ruido de manera significativa, riesgo de cambiar la clasificación al deformar la señal, etc.). La interpretación *física* sería pensar en cultivos que crecieron más lento o más rápido debido a condiciones climatológicas u otras, por lo que sus curvas de crecimiento se verán *estiradas* o *comprimidas*. En la Fig. 3 se muestra, a modo de ejemplo, la serie temporal de la banda del infrarrojo cercano para un punto geográfico en particular y los datos artificiales creados a partir del mismo. El código utilizado se encuentra al final del archivo `createTrain.py`. El tensor generado,  $X_{aug}$  tiene tamaño (9339, 48, 6)

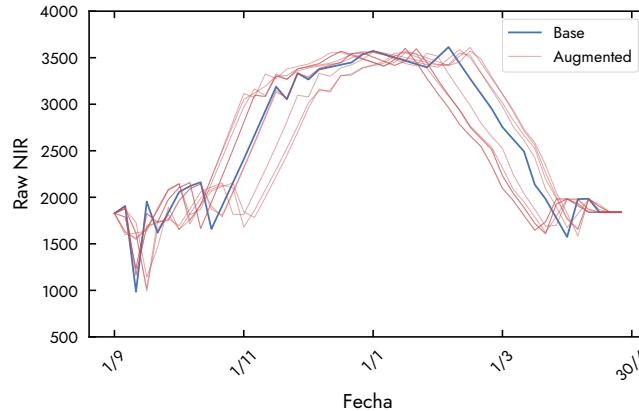


Figura 3: Ejemplo del proceso de *data augmentation* empleado. Se muestra la banda NIR de un punto geográfico elegido al azar y los 10 datos nuevos generados.

## Método y entrenamiento

### Modelo elegido y justificación

Como se dijo con anterioridad, la arquitectura elegida consta de módulos LSTM-FC-CNN-FC. El razonamiento detrás de cada capa es la siguiente:

**Extracción de correlaciones temporales** Para esto se utiliza una capa de 32 módulos LSTM para extraer características temporales abstractas de alto nivel de forma jerárquica de las series temporales de entrada. La capa está compuesta de unidades o módulos LSTM, [Gers, 1999, Zhou et al., 2016], en los que información seleccionada se guarda a tiempo  $t$  y es transportada selectivamente a tiempo  $t + 1$  para construir las conexiones entre cada paso temporal. Comparado a una capa RNN normal la ventaja es el aprendizaje adaptativo de los módulos LSTM para determinar en qué medida conservar la información del estado anterior y en qué medida “olvidarlo”. Seguido de esto se utiliza una capa densa distribuida temporalmente para conservar la multidimensionalidad de los datos y de esta forma poder extraer características (*features*) y correlaciones tanto temporales como espectrales (entre bandas) simultáneamente.

**Extracción de patrones** Para esto se utilizan capas convolucionales, [LeCun et al., 1990], seguidas de una capa densa. Cada capa convolucional extrae las características principales del mapa temporal-espectral creado y de manera jerárquica y no lineal extrae patrones de alto nivel. Al utilizar un tamaño de filtro de  $3 \times 3$  en una primera capa y  $7 \times 7$  en la que le sigue, aumentamos la dimensionalidad de los datos. Seguido de esto, una capa densa extrae las relaciones entre cada uno de los patrones obtenidos de las capas convolucionales. Para ésta última se utilizó la función de activación exponencial lineal (*Exponential Linear* o *ELU*) para evitar los problemas de *vanishing gradient* y *dying ReLU*.

<sup>4</sup><https://ttsaug.readthedocs.io/en/stable/>

**Clasificación multiclas** Para esto se utiliza una capa densa de 14 unidades (igual que el número de clases) con activaciones *softmax*. Dicha capa mapea la salida de la última capa densa a un espacio de probabilidades de dimensión 14, en este caso, donde cada posibilidad es la correspondiente a que el tensor de entrada corresponda a una clase en particular.

El modelo usado puede verse en la Fig. 4.

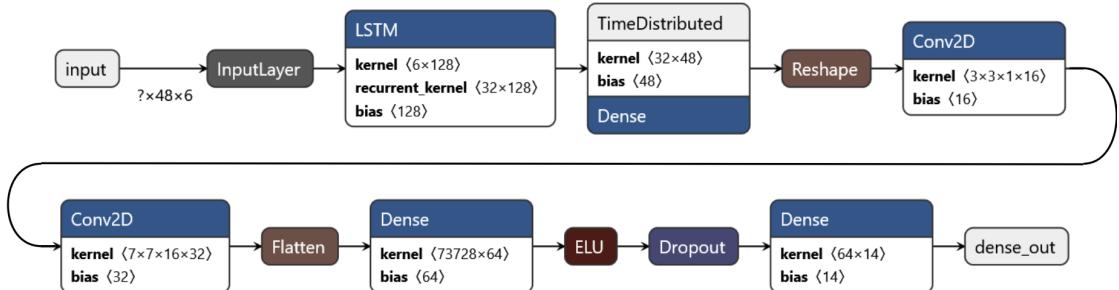


Figura 4: Arquitectura de la red neuronal diseñada..

## Entrenamiento

Se utilizó la API keras usando como backend tensorflow 2.3.0, en Python 3.6. Previo al entrenamiento, se normalizaron los datos restando su media y dividiendo por su desviación estándar, usado el scaler StandardScaler del paquete sklearn. Dada la cantidad de parámetros a entrenar (4.751.422) se decidió utilizar ciertas técnicas para mejorar el desempeño y disminuir el overfitting sobre el set de entrenamiento. En particular se dividió la rutina de entrenamiento en tres partes, que se detallan a continuación.

### Parte I

Se entrenó durante 500 epochs, con un learning rate alto y batch size pequeño, *sin la capa densa después de las convolucionales*. Con esto se espera que si bien la capa densa de salida va a overfittear el dataset de entrenamiento, esta luego se descartará y se espera que las capas LSTM, densa distribuida temporalmente y las convolucionales aprendan de manera satisfactoria. El código se puede ver en el archivo [parteI.py](#). Se utilizó el dataset de entrenamiento original, sin augmentation.

### Parte II

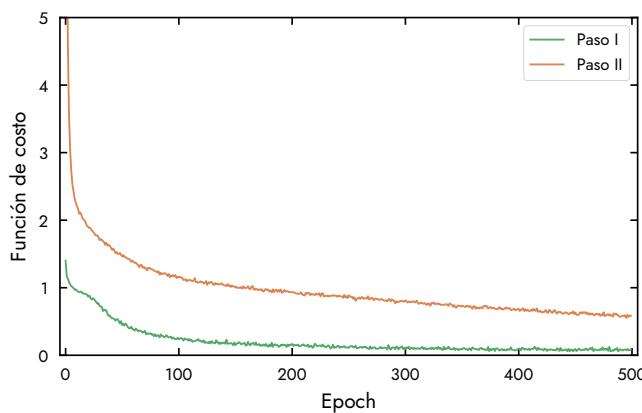


Figura 5: Evolución de la función de costo para las primeras dos partes del entrenamiento.

Se agrega la capa densa faltante y se entrena utilizando los pesos de la parte anterior, exceptuando en las últimas dos capas. Ver el archivo [parteII.py](#). El valor de los hiperparámetros,

con los que se llama la función `createModelII`, se obtuvieron luego de realizar una validación cruzada (*cross-validation*) con *k-folding* de 10 folds. Como se observa en el valor de los mismos, se requiere una gran cantidad de regularización (tanto L2 como dropout), producto de los pocos datos de entrenamiento. En la Figura 5 se observa la evolución de la función de costo sobre los datos de entrenamiento para las primeras dos partes. De la misma se puede observar que la la parte II se beneficiaría si se continua el entrenamiento, pero se decidió parar en 500 epochs para frenar el overfitting. Se utilizó el dataset de entrenamiento original, sin augmentation. Además, se empleó el optimizador *Adam* con *AMSGrad*, [Reddi et al., 2019], y *cosine annealing*.

### Parte III

En esta última de entrenamiento se buscó solucionar dos problemas:

1. La poca cantidad de datos de etrenamiento produce una falta de estabilidad y robustez en los resultados, siendo sumamente dependientes de la naturaleza estocástica del aprendizaje. Para atacar este problema se utilizaron dos métodos de ensamble distintos:
  - Para agregar robustez frente a la influencia de los procesos aleatorios propios del aprendizaje profundo se utilizó *stochastic weight averaging*, [Izmailov et al., 2019]. Sin entrar en detalles, la idea principal es que los pesos de la red, una vez que la misma ya se encuentra entrenada, fluctúan alrededor del mínimo u óptimo. Por lo tanto, utilizando un learning rate pequeño y *cosine annealing* se capturan los pesos cada cierto número de epochs (3 a 5, generalmente) y se promedian entre si.
  - Para agregar robustez frente a los pocos datos de entrenamiento, se utilizó la técnica conocida como *Bootstrap aggregating* o *bagging*, [Breiman, 1996]. Para esto se utilizó el set con augmentations y se tomaron 20 sets de 1000 muestras cada uno, de manera aleatoria y con reposición. Luego se utiliza el promedio de la clasificación de los 20 modelos distintos como clasificación final. Ver el archivo `parteIII.py`. La idea principal es de esta forma reducir la variancia y el overfitting.
2. Como se vio al analizar los datos, estos se encuentran sumamente desbalanceados. Luego de probar varias alternativas, se decidió emplear la opción `class_weight` de `model.fit()` de keras. Se utilizaron las proporciones del dataset de entrenamiento original.

En resumen: mediante la técnica de *bagging* se entrenaron 20 modelos distintos y se toma como clasificación final el promedio de las clasificaciones de los mismos. Además, en cada uno, se empleó *stochastic weight averaging* y se utilizaron las proporciones de las clases para balancear la función de costo. El algoritmo es el siguiente:

---

**Algoritmo 2:** Métodos de ensamble usados en la etapa final de entrenamiento.

---

1. Generar 20 sets de 1000 muestras cada uno, sampleados de manera aleatoria y con reposición del set *augmented* (de 9339 muestras).
  2. Cada uno de estos sets se utilizó para entrenar un modelo distinto, partiendo de los pesos de la parte II. Se configura la opción `class_weight` con las proporciones del set de entrenamiento original. Luego se entrena por 80 epochs con *learning rate* constante, para terminar con 120 epochs más aplicando SWA.
  3. Finalmente, se unen los 20 modelos generados en uno solo, promediando las probabilidades a la salida de la capa densa con activación *softmax* de todos ellos.
- 

A modo de instrucciones, si si quiere crear los datasets y entrenar la red desde cero, basta con correr `createTrain.py` y `createTest.py`, seguido de `parteI.py`, `parteII.py` y `parteIII.py` en orden. Con esto quedará generado el archivo `modelFinal.h5` para realizar las predicciones con `predict.py`. Sin embargo, para ahorrar tiempo computacional se proveen todos los archivos intermedios y finales generados en [GitHub](#), salvo por `modelFinal.h5` que se encuentra en [Google Drive](#). Es importante recordar que, de correr el entrenamiento desde cero, los resultados pueden ser levemente distintos a los presentados, debido a la naturaleza estocástica de las redes neuronales.

# Resultados

Finalmente se obtiene el modelo final, `modelFinal.h5`. Para obtener la clasificación final sobre el set de testing suministrado, en formato `.csv` se puede utilizar el archivo `predict.py`. Luego de finalizada la competencia se suministró la clasificación correcta de este set, con el que se generó la matriz de confusión de la Fig. 6. Se observan que hay categorías, como por ejemplo agua, que tienen 100 % de falla al clasificar. Otras, como las sojas, son en un 10 % confundidas entre si. La categoría “no sabe” no solo fue clasificada erróneamente en el 87 % de las ocasiones (poco sorprendentemente), sino que “ensucia” a otros cultivos, como soja de 1ra y maíz de 2da, en las que el 10 % y 50 % se clasifican como “no sabe”. En congruencia con lo visto en la Fig. 2, las clases “campo natural” y “forraje” se confunden entre si. De lo observado en la matriz se desprenden varias ideas para una futura mejora del modelo, que se discutirán en la siguiente sección. El *balanced accuracy score*, calculado con el paquete `sklearn` da como resultado final de la clasificación una precisión balanceada de **61.1 %**.

	Soja 1	10.5%	2.6%	0.4%	0.9%	0.0%	0.4%	1.3%	0.9%	1.3%	0.0%	0.0%	9.2%	0.0%
Soja 1	72.5%	10.5%	2.6%	0.4%	0.9%	0.0%	0.4%	1.3%	0.9%	1.3%	0.0%	0.0%	9.2%	0.0%
Soja 2	12.1%	84.5%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	3.4%	0.0%
Maíz 1	1.4%	0.7%	94.3%	0.0%	0.0%	0.0%	0.0%	1.4%	0.7%	0.0%	0.0%	0.0%	0.7%	0.7%
Maíz 2	0.0%	0.0%	0.0%	50.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	50.0%	0.0%
Trigo	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Girasol	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Sorgo	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Forraje	2.8%	0.0%	0.0%	0.0%	0.0%	0.0%	2.8%	72.2%	13.9%	0.0%	0.0%	0.0%	5.6%	2.8%
Campo natural	3.7%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	27.8%	66.7%	0.0%	0.0%	0.0%	1.9%	0.0%
Barbecho	0.0%	66.7%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	33.3%	0.0%	0.0%	0.0%	0.0%
Agua	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Alfalfa	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
No sabe	45.5%	18.2%	9.1%	0.0%	0.0%	0.0%	0.0%	9.1%	0.0%	0.0%	0.0%	0.0%	13.6%	4.5%
Urbano	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	14.3%	0.0%	0.0%	0.0%	0.0%	0.0%	85.7%
	Soja 1	Soja 2	Maíz 1	Maíz 2	Trigo	Girasol	Sorgo	Forraje	Campo natural	Barbecho	Agua	Alfalfa	No sabe	Urbano

Figura 6: Matriz de confusión para el dataset de testing. Los porcentajes mostrados corresponden a cada cultivo en particular, ie., el número de clasificaciones es dividido por el total de muestras de la clase correcta.

En la Figura 7 se muestran dos ejemplos para dos clases distintas de los tensores de entrada (tamaño  $48 \times 6$ ) y la respectiva activación de la capa densa distribuida temporalmente, tamaño  $48 \times 48$ . Este mapa de activación muestra correlaciones temporales-espectrales, que luego las capas convolucionales y densas se encargarán de extraer. Sin embargo, resulta útil ver que las activaciones para “maíz 1” y “urbano” son completamente distintas, mientras que se parecen más entre si dentro de la misma clase.

## Conclusiones y mejoras

Uno de los principales requisitos de diseño autoimpuesto fue que los datos de entrada tuvieran la menor cantidad de preprocesamiento posible, para así facilitar la clasificación “en campo” o de manera automatizada para estimar cosechas y planear políticas públicas y privadas. Se considera que eso se logró satisfactoriamente, acompañado de una precisión final adecuada. Además, las predicciones se pueden realizar en computadoras con muy pocos requerimientos de hardware. Para modelos a futuro se consideran una serie de mejoras para aumentar la precisión, sin aumentar la complejidad del modelo o del preprocesamiento de manera significativa:

- Separar los datos en campañas, y entrenar cada modelo por separado u realizar *transfer learning* y *fine tuning* sobre el modelo ya entrenado. Cada año presenta proporciones

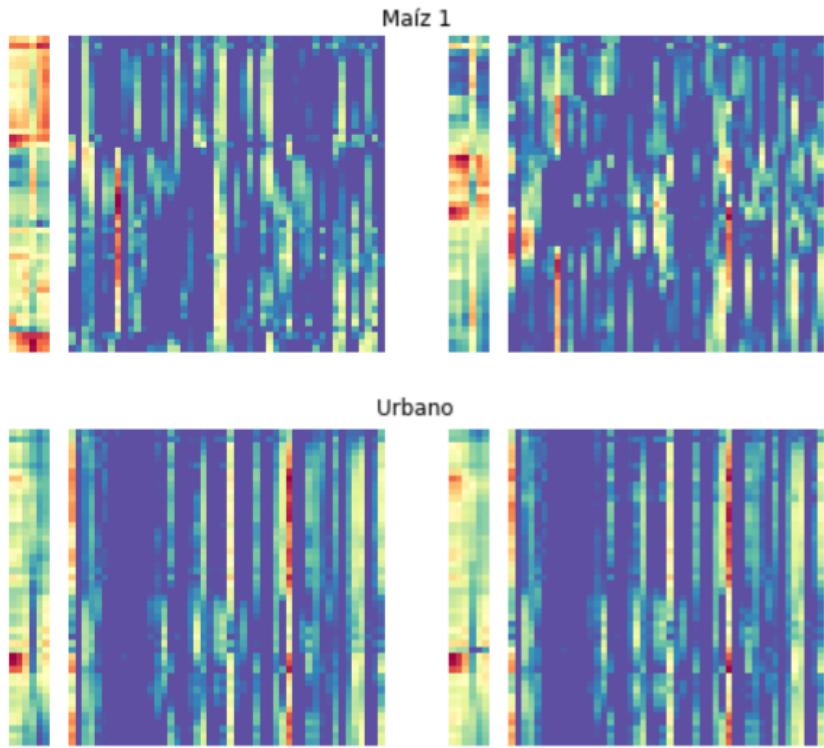


Figura 7: Dos ejemplos para dos clases distintas de los tensores de entrada (altura = muestras temporales, ancho = número de bandas, ie.,  $48 \times 6$ ) junto a la activación a la salida de la capa densa distribuida temporalmente (tamaño  $48 \times 48$ ).

distintas de cultivos (por ejemplo de soja de 2da, ver 1) y sería útil tener en cuenta esto al momento de entrenar. En el presente trabajo esto se ignoró a fin de crear un *único* modelo para facilitar el uso automatizado, independiente del año, a costo de un porcentaje de precisión final.

- De la matriz de confusión, Fig. 6, se desprenden varias mejoras a implementar. En primer lugar, el agua se puede clasificar *a priori* con sus trazas de NDVI, sin necesidad de clasificarlo mediante la red neuronal. En segundo lugar, ignorar completamente la clase “no sabe”. Resulta difícil poder clasificar la falta de conocimiento del personal que hizo el relevamiento sobre un determinado cultivo, y solo genera una perdida importante de precisión final.
- Estudiar de manera sistemática, mediante estadísticas de años anteriores o actuales, cuáles son los cultivos que se buscan clasificar. Disminuir el número de clases que la red debe aprender resulta en un mejor desempeño sin realizar cambios de importancia en la arquitectura.

## Referencias

- [Ahmad et al., 2011] Ahmad, I., Siddiqi, M. H., Fatima, I., Lee, S., and Lee, Y.-K. (2011). Weed classification based on Haar wavelet transform via k-nearest neighbor (k-NN) for real-time automatic sprayer control system. In *Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication - ICUIMC '11*, Seoul, Korea. ACM Press.
- [Alhassan et al., 2020] Alhassan, V., Henry, C., Ramanna, S., and Storie, C. (2020). A deep learning framework for land-use/land-cover mapping and analysis using multispectral satellite imagery. *Neural Computing and Applications*, 32(12):8529–8544.
- [Breiman, 1996] Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

- [Campos-Taberner et al., 2020] Campos-Taberner, M., García-Haro, F. J., Martínez, B., Izquierdo-Verdiguier, E., Atzberger, C., Camps-Valls, G., and Gilabert, M. A. (2020). Understanding deep learning in land use classification based on Sentinel-2 time series. *Scientific Reports*, 10(1):17188.
- [Friedl and Brodley, 1997] Friedl, M. and Brodley, C. (1997). Decision tree classification of land cover from remotely sensed data. *Remote Sensing of Environment*, 61(3):399–409.
- [Gers, 1999] Gers, F. (1999). Learning to forget: Continual prediction with LSTM. In *9th International Conference on Artificial Neural Networks: ICANN '99*, volume 1999, pages 850–855, Edinburgh, UK. IEE.
- [Guennec et al., 2016] Guennec, A. L., Malinowski, S., and Tavenard, R. (2016). Data Augmentation for Time Series Classification using Convolutional Neural Networks. In *Workshop on Advanced Analytics and Learning on Temporal Data*, Riva Del Garda, Italy.
- [Izmailov et al., 2019] Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., and Wilson, A. G. (2019). Averaging Weights Leads to Wider Optima and Better Generalization. *arXiv:1803.05407 [cs, stat]*.
- [Ji et al., 2018] Ji, S., Zhang, C., Xu, A., Shi, Y., and Duan, Y. (2018). 3D Convolutional Neural Networks for Crop Classification with Multi-Temporal Remote Sensing Images. *Remote Sensing*, 10(2):75.
- [LeCun et al., 1990] LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1990). Handwritten Digit Recognition with a Back-Propagation Network. In *Advances in Neural Information Processing Systems (NIPS 1989)*, Denver, CO. Morgan Kaufmann.
- [Li et al., 2020] Li, Z., Chen, G., and Zhang, T. (2020). A CNN-Transformer Hybrid Approach for Crop Classification Using Multitemporal Multisensor Images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13:847–858.
- [Liang et al., 2012] Liang, S., Li, X., and Wang, J., editors (2012). *Advanced Remote Sensing*. Academic Press, Amsterdam ; Boston, 1st ed edition.
- [Mathur and Foody, 2008] Mathur, A. and Foody, G. M. (2008). Crop classification by support vector machine with intelligently selected training data for an operational application. *International Journal of Remote Sensing*, 29(8):2227–2240.
- [Mazzia et al., 2019] Mazzia, V., Khaliq, A., and Chiaberge, M. (2019). Improvement in Land Cover and Crop Classification based on Temporal Features Learning from Sentinel-2 Data Using Recurrent-Convolutional Neural Network (R-CNN). *Applied Sciences*, 10(1):238.
- [Murthy et al., 2003] Murthy, C. S., Raju, P. V., and Badrinath, K. V. S. (2003). Classification of wheat crop with multi-temporal images: Performance of maximum likelihood and artificial neural networks. *International Journal of Remote Sensing*, 24(23):4871–4890.
- [Reddi et al., 2019] Reddi, S. J., Kale, S., and Kumar, S. (2019). On the Convergence of Adam and Beyond. *arXiv:1904.09237 [cs, math, stat]*.
- [Rustowicz et al., 2019] Rustowicz, R. M., Cheong, R., Wang, L., Ermon, S., Burke, M., and Lobell, D. (2019). Semantic Segmentation of Crop Type in Africa: A Novel Dataset and Analysis of Deep Learning Methods. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.
- [Saini and Ghosh, 2019] Saini, R. and Ghosh, S. K. (2019). Crop classification in a heterogeneous agricultural environment using ensemble classifiers and single-date Sentinel-2A imagery. *Geocarto International*, pages 1–19.
- [Segarra et al., 2020] Segarra, J., Buchaillot, M. L., Araus, J. L., and Kefauver, S. C. (2020). Remote Sensing for Precision Agriculture: Sentinel-2 Improved Features and Applications. *Agronomy*, 10(5):641.
- [Tatsumi et al., 2015] Tatsumi, K., Yamashiki, Y., Canales Torres, M. A., and Taipe, C. L. R. (2015). Crop classification of upland fields using Random forest of time-series Landsat 7 ETM+ data. *Computers and Electronics in Agriculture*, 115:171–179.

- [Xu et al., 2020] Xu, J., Zhu, Y., Zhong, R., Lin, Z., Xu, J., Jiang, H., Huang, J., Li, H., and Lin, T. (2020). DeepCropMapping: A multi-temporal deep learning approach with improved spatial generalizability for dynamic corn and soybean mapping. *Remote Sensing of Environment*, 247:111946.
- [Yang et al., 2020] Yang, S., Gu, L., Li, X., Jiang, T., and Ren, R. (2020). Crop Classification Method Based on Optimal Feature Selection and Hybrid CNN-RF Networks for Multi-Temporal Remote Sensing Imagery. *Remote Sensing*, 12(19):3119.
- [Zhong et al., 2019] Zhong, L., Hu, L., and Zhou, H. (2019). Deep learning based multi-temporal crop classification. *Remote Sensing of Environment*, 221:430–443.
- [Zhou et al., 2016] Zhou, P., Shi, W., Tian, J., Qi, Z., Li, B., Hao, H., and Xu, B. (2016). Attention-Based Bidirectional Long Short-Term Memory Networks for Relation Classification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 207–212, Berlin, Germany. Association for Computational Linguistics.