Clasificación de imágenes hiperespectrales de frutas y verduras, mediante aprendizaje profundo.

Jair Sebastián Barrera Tulcán

Ingeniería Mecatrónica

Director del trabajo de grado:

Jorge Alexis Herrera Ramírez

INSTITUTO TECNOLÓGICO METROPOLITANO

Abril 30 de 2019

"Hasta la fecha, no se ha diseñado un ordenador que sea consciente de lo que está haciendo; pero, la mayor parte del tiempo, nosotros tampoco lo somos"

Marvin Minsky

RESUMEN

En la actualidad la aplicación de técnicas de aprendizaje profundo (Deep Learning) para la clasificación de imágenes hiperespectrales ha permitido un mejor aprovechamiento de la información proporcionada por estas imágenes. Basados en este hecho, se presenta el uso de redes neuronales convolucionales de dos dimensiones (2D-CNN) para la clasificación de 13 clases de frutas y verduras escogidas aleatoriamente, de una base de datos publica de 42 imágenes hiperespectrales.

Para el desarrollo de este trabajo se parte de la selección de una arquitectura de aprendizaje profundo, con el fin de aplicarse en la clasificación de tres imágenes hiperespectrales construidas artificialmente usando partes extraídas de la base de datos de 42 imágenes hiperespectrales mencionada. Cada una de estas tres imágenes contiene las mismas clases, pero mezcladas espacialmente en diferentes formas. Estas imágenes se pre procesaron y generaron usando Matlab®. A partir de estas imágenes hiperespectrales, se procede a realizar el entrenamiento y predicción en lenguaje Python implementando librerías de TensorFlow. Se realizó una comparación de resultados de clasificación usando máquinas de soporte vectorial, técnica clásica de clasificación y esto mostró que la 2D-CNN tiene un mejor desempeño. La arquitectura 2D-CNN superó el 98% de precisión en la clasificación (para las tres imágenes generadas), y aunque con altos resultados en los tres casos, también se pudo observar que el modelo es dependiente de la forma en la que se distribuyen espacialmente los píxeles de cada clase en las imágenes hiperespectrales construidas para este trabajo.

Palabras claves: Aprendizaje profundo, inteligencia artificial, clasificación, máquina de aprendizaje, imágenes hiperespectrales.

RECONOCIMIENTOS

Antes de nada quiero darle las gracias y la gloría a Dios, quiero agradecer a mi padre que en paz descanse que ha sido y será la guía en cada paso que doy, agradecer también el apoyo incondicional tanto económico como moral de mi abuelito Franco Barrera Meneses para que mi sueño se hiciera realidad, agradecer a mi familia; Alicia Tulcán, madre, David, Zuleyma y Ricardo Barrera Tulcán, hermanos, por el apoyo incondicional, agradecer a mi compañera sentimental Marcela de la Rosa, agradecer a mi asesor de trabajo de grado por cada una de las asesorías y consejos recibidos, agradecer a mi profesora Norma Patricia Guarnizo por los consejos recibidos y por último, agradecer a esta hermosa institución universitaria ITM por entregarme todos los conocimientos para cumplir el sueño de este nuevo joven Ingeniero Mecatrónico que le entrega a la sociedad.

ACRÓNIMOS

SVM Máquina de soporte vectorial (Support Vector Machine)

KNN Vecinos más cercanos (K-nearest neighbors)

Red neuronal (NN, Neural Network)

CNN Red neuronal convolucional (Convolutional Neural Network)

LDA Análisis discriminante lineal (Linear Discriminant Analysis)

PCA Análisis de componentes principales (Principal Component Analysis)

LADA Análisis discriminante adaptativo de la localidad (Locality Adaptive Discriminant Analysis)

2DCNN Red neuronal convolucional de dos dimensiones (2-Dimension Convolutional Neural Network)

3D-CNN de tres dimensiones (3-Dimension Convolutional Neural Network)

2D-R-CNN Redes neuronales recurrente de dos dimensiones (2 Dimension Recurrent Neural Network)

3D-R-CNN de tres dimensiones (3 Dimension Recurrent Convolutional Neural Network)

LSTM Memoria a corto plazo y a largo plazo (Long Short-Term Memory),

MugNet Red de grano múltiple (Multi-grained Network)

TABLA DE CONTENIDO

1. INTRODUCCIÓN

En la actualidad la óptica y la fotónica han avanzado en el análisis y procesado de imágenes hiperespectrales, por lo que es una técnica que viene siendo usada con resultados satisfactorios en diversos campos tecnológicos y de investigación que van desde la agricultura (Wendel & Underwood, 2016), pasando por la minería (J. Wang, Zhang, Tong, & Sun, 2012), hasta la medicina (Noor et al., 2016). La valiosa información contenida en las imágenes a múltiples longitudes de onda puede proporcionar una descripción más refinada de los componentes en la superficie de los materiales y así mejorar su detección, discriminación y caracterización (Burger & Gowen, 2011). Por ejemplo, Robert Ennis et al. (Ennis, Schiller, Toscani, & Gegenfurtner, 2018) generaron una base de datos de 42 imágenes hiperespectrales de frutas y verduras hiperespectrales para estudiar la relación de colores entre el interior y el exterior de estos objetos.

Para la clasificación de imágenes hiperespectrales se han propuesto ciertos métodos de inteligencia artificial, específicamente del aprendizaje de máquinas (ML, Machine Learning) (Pereira & Borysov, 2018) como por ejemplo son: las máquinas de soporte vectorial (SVM, support vector machine) (Pereira & Borysov, 2018) o la clasificación basada en los K-vecinos más cercanos (K-nearest neighbor classifiers) (Pereira & Borysov, 2018). Recientemente, los algoritmos dentro del área del machine learning conocidos como de aprendizaje profundo (Deep Learning) (Deng, 2014), y específicamente los basados en arquitecturas de redes neuronales convolucionales (CNN), que han demostrado una gran utilidad en múltiples actividades, entre ellas, tareas en el campo de visión artificial e imágenes hiperespectrales (X. Yang et al., 2018). Por esta razón, este trabajo se centrará en la implementación y análisis de la arquitectura de la red neuronal convolucional de dos dimensiones (2D-CNN, 2-Dimension Convolutional Neural Network) para la clasificación de imágenes hiperespectrales.

Las redes neuronales (NN, Neural Network) no son más que un modelo artificial y simplificado del cerebro humano, que es el ejemplo más perfecto del que disponemos para un sistema que es capaz de adquirir conocimiento a través de la experiencia. Una red neuronal es "un nuevo sistema para el tratamiento de la información, cuya unidad básica de procesamiento está inspirada en la célula fundamental del sistema nervioso humano: la neurona" (Alberto Ruiz Marta Susana Basualdo Autor & Jorge Matich, n.d.). Entonces, una CNN es una red neuronal multicapa compuesta en su estructura básica por una capa de entrada (input layer), una capa de convolución (conv layer), una capa de activación (Relu layer), una capa de agrupación (Pooling layer) y una capa totalmente conectada (FC, Full Conect layer). La capa de entrada es donde se reciben las imágenes (Matriz de datos); la capa Convolucional es la capa donde se obtienen características de la imagen por medio de un filtro para extraer datos que sean necesarios; la capa de activación es la capa donde se aplica una función de activación; la capa de agrupación es la capa donde se reduce la cantidad de datos, es decir, esta capa selecciona los parámetros más comunes; y la FC es la capa que entrega el resultado de la red.

A través de este trabajo, se pretende aplicar Deep learning enfocado a la clasificación de imágenes hiperespectrales de frutas y verduras, ya que mediante las características espectrales que poseen las pulpas o la parte interna de este tipo de alimentos se puede encontrar una alternativa en la clasificación. Por ejemplo hay frutas como la fresa, la frambuesa o la mora que pueden poseer características similares en su textura o también se pueden encontrar muestras similares en su geometría como la papa con la cebolla, que hacen que su clasificación se convierta en confusa o bastante compleja cuando se depende de solamente descriptores de geometría y textura Por tanto, la combinación con información espectral puede aportar precisión en la clasificación de alimentos con características similares de textura o geometría.

1.1 OBJETIVOS.

1.1.1 General

Implementar una arquitectura computacional de aprendizaje profundo para la clasificación de imágenes hiperespectrales de frutas y verduras obtenidas de una base de datos pública

1.1.2 Específicos

- Estudiar las arquitecturas de aprendizaje profundo propuestas en el estado del arte para la clasificación en imágenes hiperespectrales identificando la más adecuada para muestras de frutas y verduras.

- Seleccionar una base de datos pública de imágenes hiperespectrales de frutas y verduras y determinar los requisitos para ser usada en las arquitecturas de aprendizaje profundo identificadas en el objetivo anterior.

- Implementar la arquitectura seleccionada teniendo en cuenta los procedimientos adecuados de entrenamiento y predicción para las imágenes hiperespectrales de la base de datos seleccionada.

- Evaluar los resultados obtenidos comparándolos con la metodología clásica del aprendizaje de máquinas conocida como máquinas de soporte vectorial (SVM).

Este documento se encuentra estructurado de la siguiente manera: en la sección 3 se presenta una metodología tipo tutorial, haciendo que el lector pueda replicar los resultados obtenidos de esta red neuronal convolucional de dos dimensiones, en la sección 4 se dan a conocer los resultados obtenidos de la red neuronal convolucional de dos dimensiones y en la última sección se encuentran las conclusiones extraídas a partir de los resultados. Para la profundización de temas se puede revisar las referencias y final del documento, en el apéndice, se encuentran los códigos con los que se desarrolló este documento.

2. MARCO TEÓRICO

Este trabajo se centra en un área del conocimiento que es el aprendizaje de máquinas para la clasificación de imágenes hiperespectrales, por tanto, ese será el énfasis dado a este marco teórico.

2.1 Aprendizaje de máquinas

El aprendizaje de máquinas es un campo de la inteligencia artificial que a menudo utiliza técnicas estadísticas para dar a las computadoras la capacidad de "aprender", es decir, mejorar progresivamente el rendimiento en una tarea específica. Existen varios algoritmos de aprendizaje de máquinas o algoritmos de decisión, los cuales pueden ser: redes neuronales, naive bayes, máquinas de soporte vectorial SVM, Arboles de decisión, vecinos más cercanos KNN, entre otros (Pereira & Borysov, 2018). Todos estos han sido usados para diversas aplicaciones, como pueden ser: visión artificial, reconocimiento de patrones, reconocimiento voz, búsqueda en internet, control inteligente, clasificación de imágenes entre muchas más aplicaciones (Ping Duan, Chengjun Ding, Minglu Zhang, & Genqun Cui, 2006). Este aprendizaje se divide en dos grandes grupos: el aprendizaje supervisado y no supervisado. Las redes neuronales convolucionales artificiales hacen parte del grupo de aprendizaje no supervisado, o sea, un aprendizaje automático que es capaz de tomar decisiones por medio del entrenamiento con una base de datos.

Recientemente para la clasificación de imágenes hiperespectrales, varios autores se han enfocado en la inteligencia artificial empleando aprendizaje profundo, debido a que su rendimiento es bastante elevado respecto a su capacidad de aprendizaje y representación de características. En este método se explotan muchas capas de etapas de procesamiento

de información en arquitecturas jerárquicas para el aprendizaje de características sin supervisión. Comúnmente se ha recurrido a ciertos métodos clásicos del ML para el análisis de las imágenes hiperespectrales, sin embargo, debido a la alta cantidad de dimensiones en la representación de los datos obtenidos en estas imágenes, se presenta el fenómeno que se conoce como maldición de la dimensión o fenómeno de Hughes (Hughes, 1968), haciendo que los datos disponibles a analizar se vuelvan dispersos y complejos de procesar estadísticamente. Así ciertos métodos del ML, como: el análisis discriminante lineal (LDA, Linear Discriminant Analysis) (Shuiwang Ji & Jieping Ye, 2008), el análisis discriminante cuadrático, el análisis discriminante logarítmico, el análisis de componentes principales (PCA, Principal Component Analysis) (Burges, 2009) y el clasificador de los K-vecinos más cercanos (K-nearest neighbor classifiers), sean difícilmente aplicables en la clasificación de imágenes hiperespectrales (Q. Wang, Meng, & Li, 2017; Q. Wang, Yuan, Du, & Li, 2019; X. Yang et al., 2018). El Análisis discriminante adaptativo de la localidad (LADA, Locality Adaptive Discriminant Analysis) (Q. Wang et al., 2017) se ha realizado como solución frente al fenómeno de dimensionalidad, haciendo que se convierta en un método de clasificación para imágenes hiperespectrales. Otro método clásico de clasificación como la SVM se ha propuesto para la clasificación de imágenes hiperespectrales, pero su modelado limitado y su poder de representación son insuficientes en casos de clasificaciones complejas (Melgani & Bruzzone, 2004). Para este trabajo se compara el rendimiento de aprendizaje profundo vs las SVM.

Por último, se traen a colación algunos de los métodos del Deep learning implementados en la literatura y que están enfocados a la clasificación de imágenes hiperespectrales. Estos son: CNN de dos dimensiones (2D-CNN, 2 Dimension Convolutional Neural Network) (X. Yang et al., 2018), CNN de tres dimensiones (3D-CNN, 3 Dimension Convolutional Neural Network) (X. Yang et al., 2018), redes neuronales recurrentes de dos dimensiones(2D-R-CNN, 2 Dimension Recurrent Neural Network) (X. Yang et al., 2018), R-CNN de tres dimensiones (3D-R-CNN, 3 Dimension Recurrent Convolutional Neural Network) (X. Yang et

al., 2018), memoria a corto plazo y a largo plazo (LSTM, Long Short-Term Memory) (Ma, Filippi, Wang, & Yin, 2019), red de grano múltiple (MugNet, Multi-grained Network)(Pan, Shi, & Xu, 2018) y redes neuronales de dos canales (two-channel deep convolutional neural network) (J. Yang, Zhao, Chan, & Yi, 2016). Estas arquitecturas son comparadas frente a otras arquitecturas de clasificación, mostrando la predicción en escenas hiperespectrales de acceso público y que son conocidas como: INDIAN PINES, PAVIA UNIVERSITY, y SALINAS. En estas pruebas las redes 3D-R-CNN, 2D-R-CNN y la 3D-CNN sobresalen por tener resultados con una precisión en la predicción superiores al 98%; seguidos de las redes 2D-CNN y CNN de dos canales con unas precisiones de predicción superiores al 95%; y finalmente seguidos por la red MugNet y LSTM con predicciones superiores al 90% y 83%, respectivamente.

3. METODOLOGÍA

La metodología se desarrolla tipo tutorial o manual, con textos descriptivos, con el fin de que el lector pueda replicar los resultados obtenidos, de la misma manera se pretende que este sirva como guía para implementaciones futuras dentro del laboratorio de óptica y fotónica del ITM. Dicho lo anterior, la metodología se divide en cuatro secciones: en la primera sección se realiza la selección de la arquitectura de la red neuronal convolucional de dos dimensiones a partir de una comparación de la aplicación de aprendizaje profundo para la clasificación de imágenes hiperespectrales en escenas de: Indian Pines, Pavia University y Salinas. La segunda sección muestra el pre procesamiento de imágenes hiperespectrales, realizando un paso a paso para obtener las imágenes hiperespectrales de la base de datos de frutas y verduras, con las que se obtendrán las imágenes hiperespectrales a ser clasificadas. La tercera sección da a conocer los pasos para generar las imágenes hiperespectrales y sus respectivas imágenes de etiquetas con las que se obtendrán los resultados de la clasificación de la 2DCNN. En la cuarta sección se desarrollan procedimientos que por medio del entrenamiento y la predicción de la 2DCNN se puedan extraer los resultados que se compararán con la máquina de soporte vectorial (SVM).

3.1 Selección de arquitectura de aprendizaje profundo.

Se revisaron 7 arquitecturas de aprendizaje profundo enfocado a la clasificación de imágenes hiperespectrales: 2D-CNN, 3D-CNN, 2D-R-CNN, 3D-R-CNN, LSTM, MugNet y redes neuronales de dos canales. Los resultados se analizaron respecto a la precisión de

predicción general de todas las clases en tres escenas: la primera en la escena de Indian Pines (16 clases), véase la tabla 1, la segunda escena de Pavia University (9 clases), véase la tabla 2, y la tercera escena de Salinas (16 clases), véase la tabla 3, obteniendo los siguientes resultados:

Tabla 1. Comparación de los resultados obtenidos en la predicción respecto a la escena de Indian pines para diferentes arquitecturas de aprendizaje profundo. Resultados recopilados de: (Pan et al., 2018; J. Yang et al., 2016; X. Yang et al., 2018)

	2DCNN	3DCNN	2DRCNN	3D-R-CNN	LSTM	MugNet	NN 2 canales
Predicción en la escena Indian pines	97.08%	98.92%	99.19%	99.50%	-----	90.65%	95.58%

Tabla 2. Comparación de los resultados obtenidos en la predicción respecto a la escena de Pavia University para diferentes arquitecturas de aprendizaje profundo. Resultados recopilados de: (Ma et al., 2019; X. Yang et al., 2018)

	2DCNN	3DCNN	2DRCNN	3D-R-CNN	LSTM	MugNet	NN 2 canales
Predicción en la escena Pavia University	95.46%	98.49%	99.19%	99.97%	83.41%	--------	----------

Tabla 3. Comparación de los resultados obtenidos en la predicción respecto a la escena de Salinas para diferentes arquitecturas de aprendizaje profundo . Resultados recopilados de: (Ma et al., 2019; X. Yang et al., 2018).

	2DCNN	3DCNN	2DRCNN	3D-R-CNN	LSTM	MugNet	NN 2 canales
Predicción en la escena Salinas	98.96%	99.08%	99.47%	99.8%	84.07%	--------	----------

De acuerdo con la tabla 1, la tabla 2 y la tabla 3 se ha escogido la red neuronal convolucional de dos dimensiones para el desarrollo de este trabajo, debido a que tiene un alto rendimiento, cercano al de las demás CNN y superando las arquitecturas LSTM, MugNet y NN de 2 canales. Con respecto a la 3DRCNN, ésta es una arquitectura de alta complejidad en la implementación y de alto coste computacional, análogamente sucede para las arquitecturas 2DRCNN y 3DRCNN, lo que supone unos requerimientos altos en el hardware. Dados estos requerimientos, la implementación de estas arquitecturas implica un problema de recursos que sobrepasa la disponibilidad existente para este trabajo. Dados los porcentajes de predicción de la 2DCNN que supera el 97% para Indian Pines, el 95% para Pavia University y el 98% para Salinas, se elige como la arquitectura a implementar considerando un compromiso entre los resultados esperados, la complejidad de implementación y requerimientos en capacidad de procesamiento del hardware disponible. Los resultados extraídos de la literatura muestran que si se mantiene una buena calidad en los datos se pueden esperar resultados satisfactorios con esta arquitectura 2DCNN.

3.2 Pre procesamiento de imágenes hiperespectrales

A partir de la descarga de la base de datos Giessen de imágenes de frutas y verduras hiperespectrales, (Ennis et al., 2018), se debe descargar *internals_RAW.tar.gz* del siguiente link: https://zenodo.org/record/1186372#.XOoch8hKjIU, imágenes que corresponden a la parte interna de varias frutas y verduras. Esta base de datos está disponible para investigadores en visión de color y otros campos de investigación. De forma aleatoria, de una totalidad de 42 imágenes que contiene la base datos, se escogieron 13 tipos de imágenes entre frutas (11) y verduras (2), nombradas a continuación: coco, mora, kiwi, limón, naranja, pepino, frambuesa, tomate, calabaza, ají rojo, pimentón, papa y rábano.

La descarga correspondiente a cada fruta y verdura viene con dos archivos: un archivo de tipo .RAW y otro archivo de metadatos con extensión .hdr. El archivo .RAW contiene la imagen con la totalidad de sus datos en bruto, y el archivo .hdr contiene los datos del sensor de la cámara hiperespectral de escaneo de espejo (Specim VNIR HS-CL-30-V8E-OEM) con el que se obtuvieron las imágenes hiperespectrales de la base de datos de frutas y verduras. Véase la figura 1.

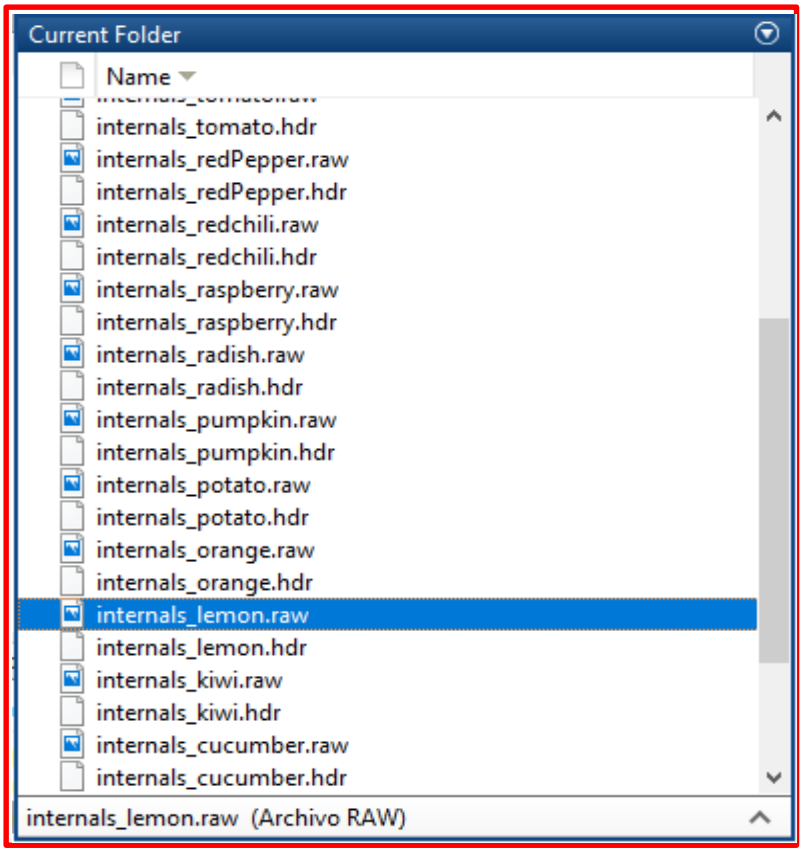


Figura 1. En la seccion delimitada por el color rojo se encuentran los archivos de algunos de los trece elementos descargados. En el Current Folder se puede apreciar tanto el .raw como el .hdr.

Los datos que se toman del archivo con extensión .hdr para formar el cubo de datos o imagen hiperespectral son: el *interleave* que es cómo está organizada la captura (en la figura 2 se observa el valor bil: banda intercalada por línea); el *data type* que corresponde a la profundidad de los datos guardados, que corresponde a 12 bits en este caso, aunque se empaquetan como datos de 16 bits (dos Bytes); los *samples* corresponden a la altura del

cubo de datos (primera dimensión); las *lines* corresponden al ancho de la imagen (segunda dimensión) y las *bands* corresponden a las bandas espectrales de la imagen (tercera dimensión). Véase la figura 2.

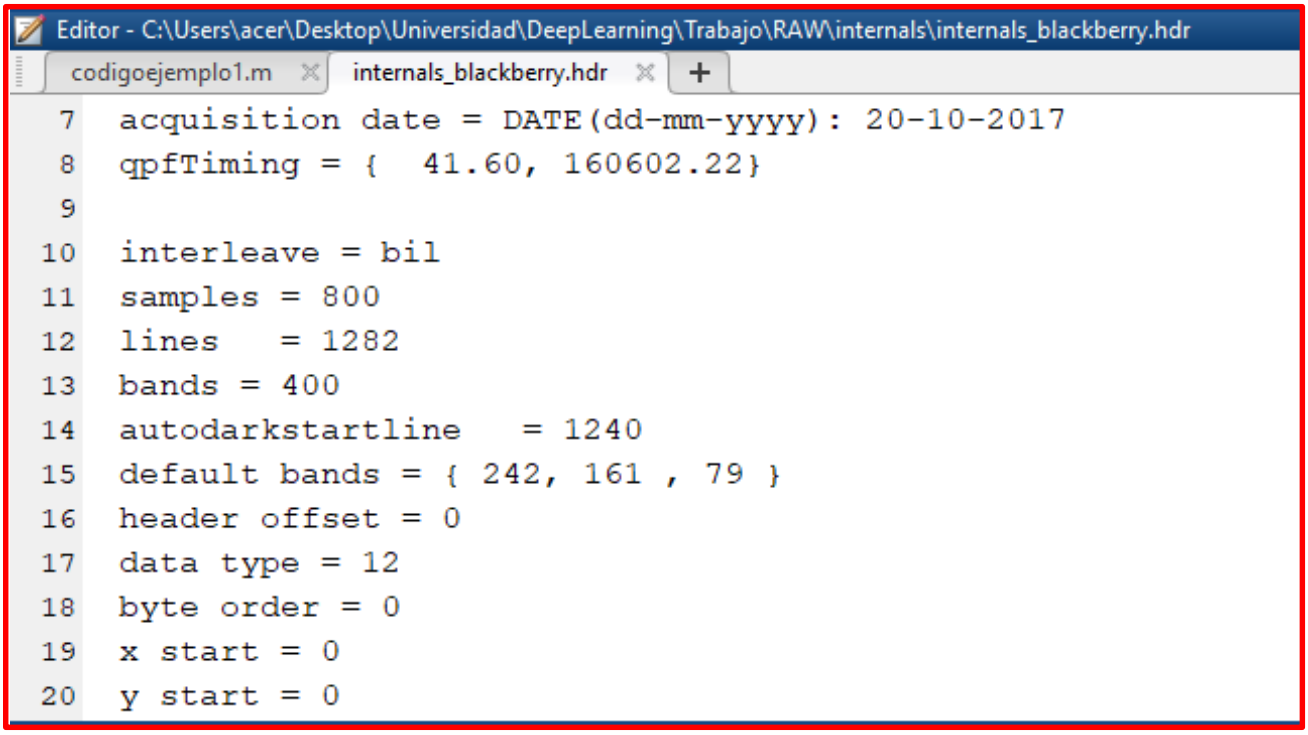


Figura 2. En la imagen se muestran los datos contenidos en el archivo de tipo .hdr para el sensor de la cámara hiperespectral de escaneo de espejo (Specim VNIR HS-CL-30-V8E-OEM) con el que se obtuvieron las imágenes hiperespectrales de la base de datos de frutas y verduras.

Por medio de un código implementado en el entorno Matlab® se transforman los archivos anteriores .raw y .hdr a un archivo .mat, que corresponde a una matriz de tres dimensiones, es decir, con un alto, un ancho y una profundidad. En este caso la tercera dimensión corresponde con las longitudes de onda de las bandas espectrales de la imagen, por lo que esta sería la dimensión espectral. En totalidad entonces se tienen 2 dimensiones espaciales y una espectral. En la figura 3 se muestra el tipo de archivo al que corresponde la matriz de tres dimensiones .mat de una de las frutas y verduras obtenidas. Tomemos como ejemplo

nuevamente a la mora, en primer lugar, se obtiene una matriz de tres dimensiones, con los siguientes valores: Blackberry (1282x800x400), véase la figura 3.

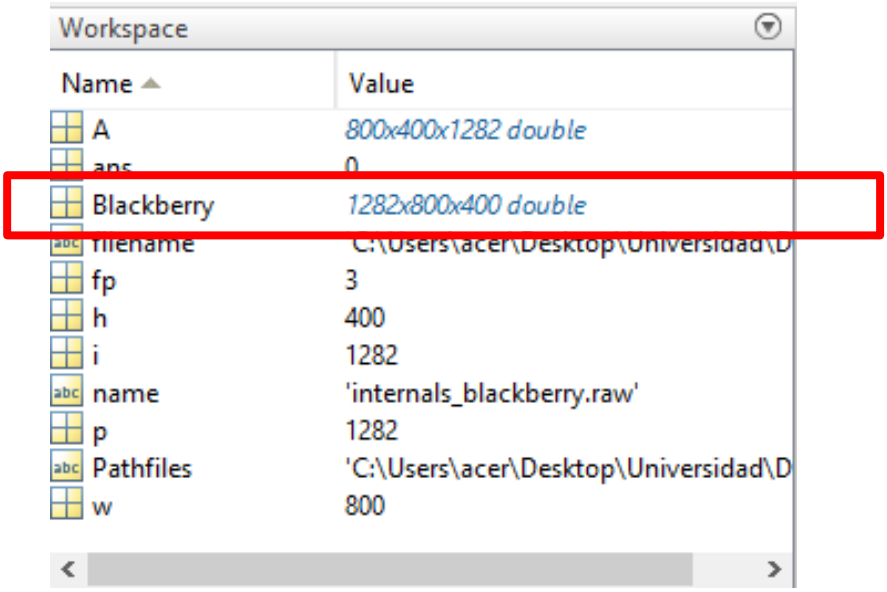


Figura 3. En las sección delimitada por el color rojo se pude ver la matriz que contiene la imagen hiperespectral de la mora a partir de los archivos de extension .raw y .hdr con ayuda de Matlab ®.

A estas matrices 3D (imágenes hiperespectrales) se les hace un recorte, en el mismo código, ya que la imagen posee el fondo en donde fue capturada y para este preprocesamiento es innecesario, y después se gira 90 grados a la derecha para obtener la sección de donde se extrae la parte interna de la fruta o verdura. El recorte cumple con el objetivo de extraer en su mayoría únicamente la fruta. El giro de 90 grados cumple la función de invertir el alto y el ancho de las imágenes ya que las imágenes hiperespectrales tenían los datos invertidos y hacían incómoda la visualización. Al final se obtiene la mayor parte interna de la mora, lo que da la siguiente matriz: Blackberry (92x133x400). Véase la figura 4.

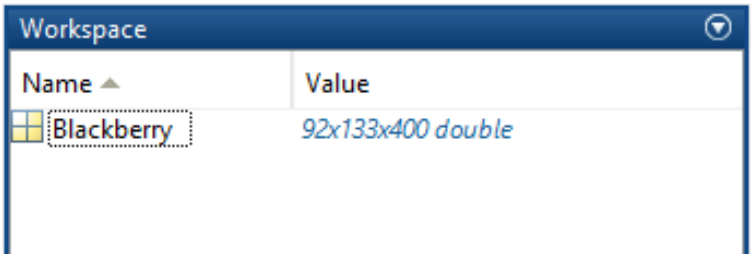


Figura 4. Matriz que contiene la imagen hiperespectral de la parte interna de la mora luego de realizar un recorte donde apararece en su mayor parte la fruta, seguido de realizar un giro de 90° a la derecha para ubicar bien las dimensiones alto y ancho.

La visualización de la matriz de tres dimensiones (Blackberry) que contiene a la fruta, sólo se puede hacer por cada canal espectral, es decir, se escoge un canal de los 400 canales que forman la imagen hiperespectral y así se extrae una imagen espectral o canal determinado. La visualización de imágenes en Matlab® se lleva a cabo usando diferentes funciones, entre ellas la función imagesc que requiere como argumento de entrada una matriz de como máximo 3 planos de profundidad o canales, que es lo que corresponde típicamente a una imagen RGB. En la figura 5 se muestra la imagen resultado de escoger el canal 240 del cubo hiperespectral de la mora usando la expresión: imagesc (Blackberry(: , : , 240)).

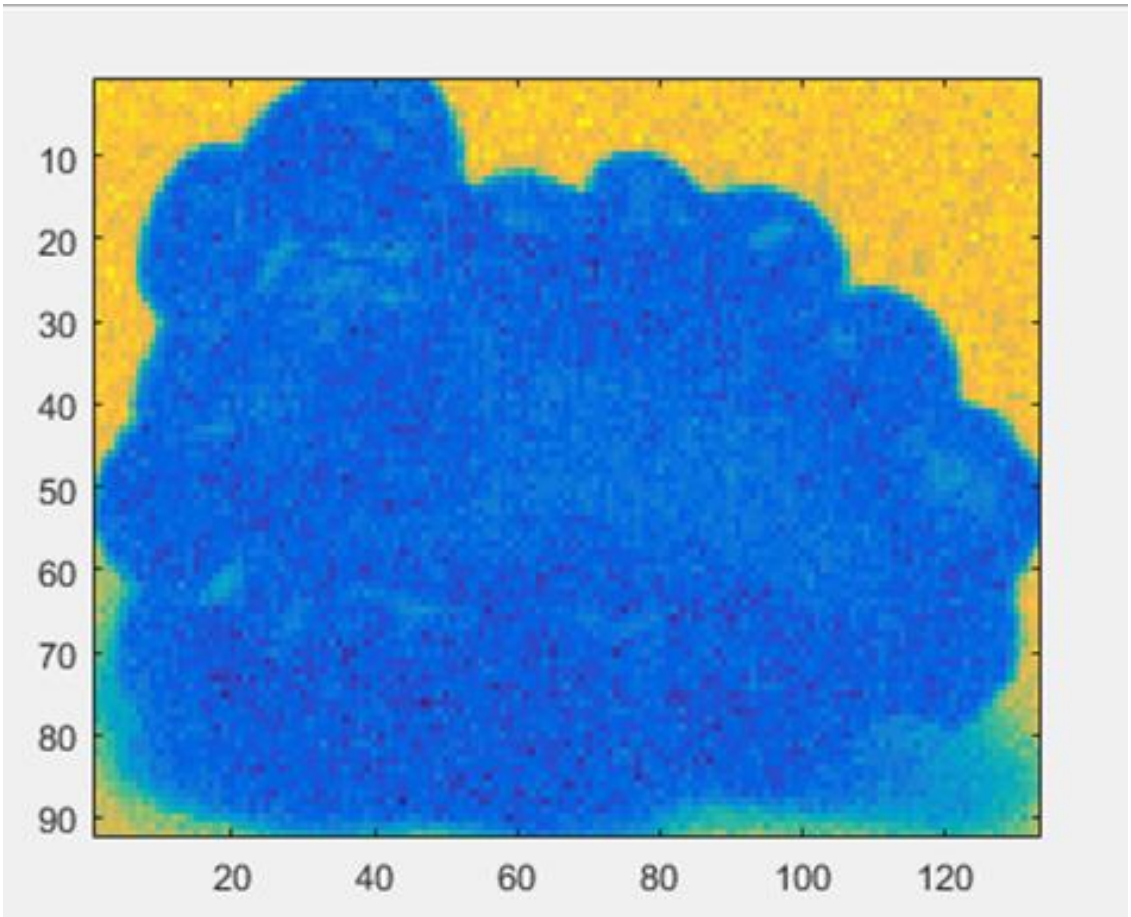


Figura 5. Imagen de la mora en el canal 240. Las dimensiones tomarian los siguientes valores, alto: 92, ancho:133 y en la profundidad donde se encuentran las bandas espectrales, el canal 240.

3.3 Generación de imagen hiperespectral e imagen de etiquetas

Se generaron tres tipos de imágenes hiperespectrales a partir de la combinación de zonas con diferentes geometrías extraídas de las 13 imágenes de frutas y verduras seleccionadas anteriormente realizando uno o dos recortes de cada una de estas 13 imágenes hiperespectrales. Esto con el fin de analizar la respuesta de la red neuronal convolucional

de dos dimensiones ante diferentes combinaciones espaciales de las muestras. Es decir, en total se desarrollaron seis imágenes con ayuda de Matlab®, tres imágenes hiperespectrales con zonas de diferentes frutas y verduras y tres imágenes correspondientes de etiquetas. Las dimensiones de estas imágenes combinadas son: 145 de alto x 145 de ancho x 200 de profundidad. Acá se redujeron el número de bandas espectrales iniciales, por tanto, se tomaron las bandas de la 80 hasta la 279, que equivale a tomar las longitudes de onda desde los 465nm a 710nm. Las etiquetas están dadas de la siguiente forma: 1 para Mora, 2 para Coco, 3 para Pepino, 4 para Papa, 5 para Kiwi, 6 para Rábano, 7 para Pimentón, 8 para Tomate, 9 para Calabaza, 10 para Frambuesa, 11 para Limón, 12 para Naranja, y 13 para Ají rojo.

3.3.1 Generación de imagen hiperespectral e imagen de etiquetas con zonas de forma regular.

Para la primera imagen hiperespectral combinada se recortaron zonas cuadradas y rectangulares de las imágenes de las muestras, y se reorganizaron en la imagen final sin que haya alguna separación entre cada una de las zonas. Esta imagen la llamaremos Imagen Hiperespectral de Zonas Regulares (ZR). Ante estas geometrías tan regulares, es de esperar que la red neuronal convolucional de dos dimensiones no se vea afectada, al menos dentro de cada una de las regiones delimitantes. La figura 6 muestra la imagen para el canal 100 utilizando el procedimiento descrito para visualizar canales específicos de una imagen hiperespectral. La figura 7 muestra la imagen de etiquetas para las 13 clases de muestras contenidas en la imagen hiperespectral ZR.

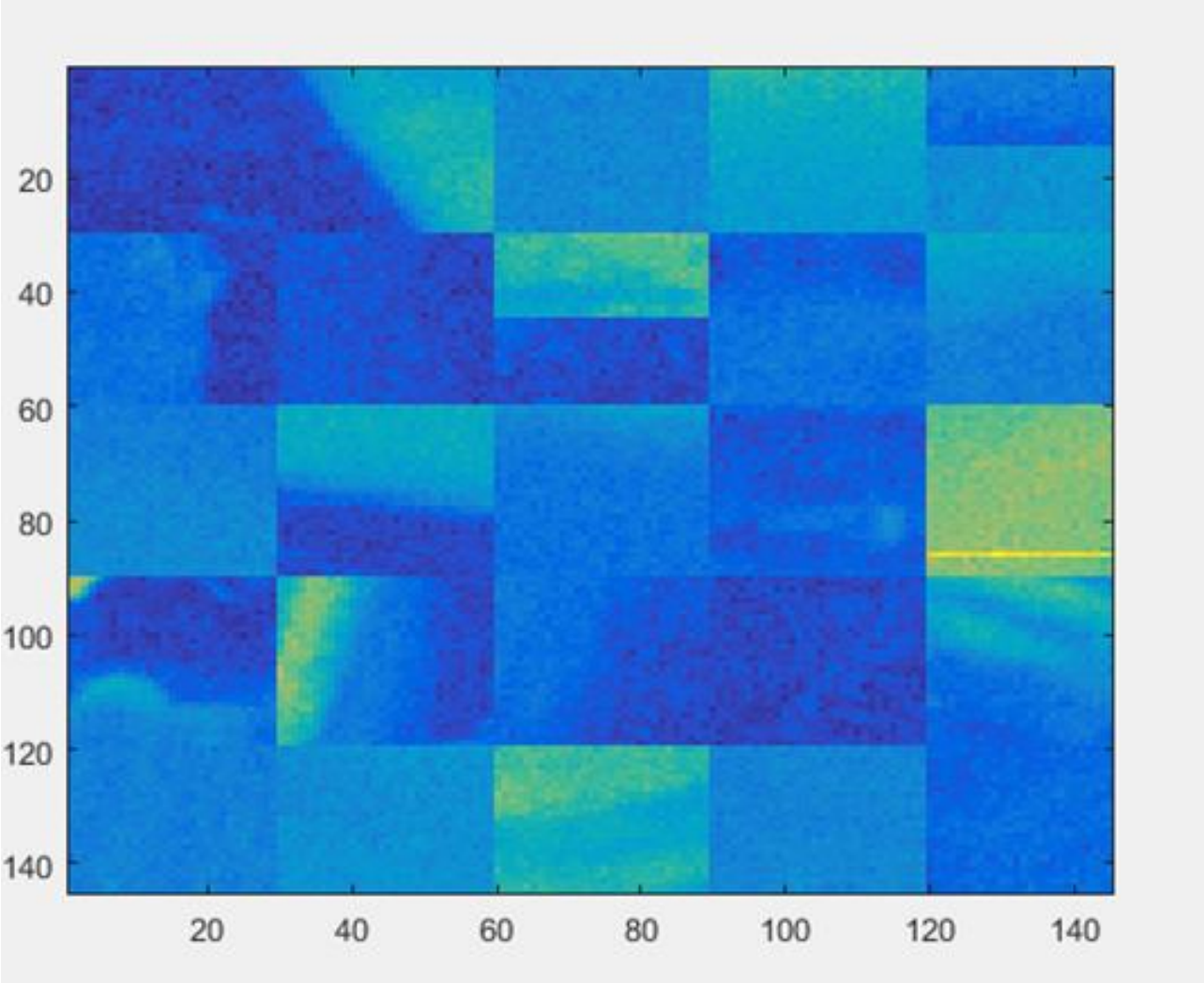


Figura 6. Imagen del canal 100 de la primera imagen hiperespectral ZR generada a partir de zonas cuadradas y rectangulares.

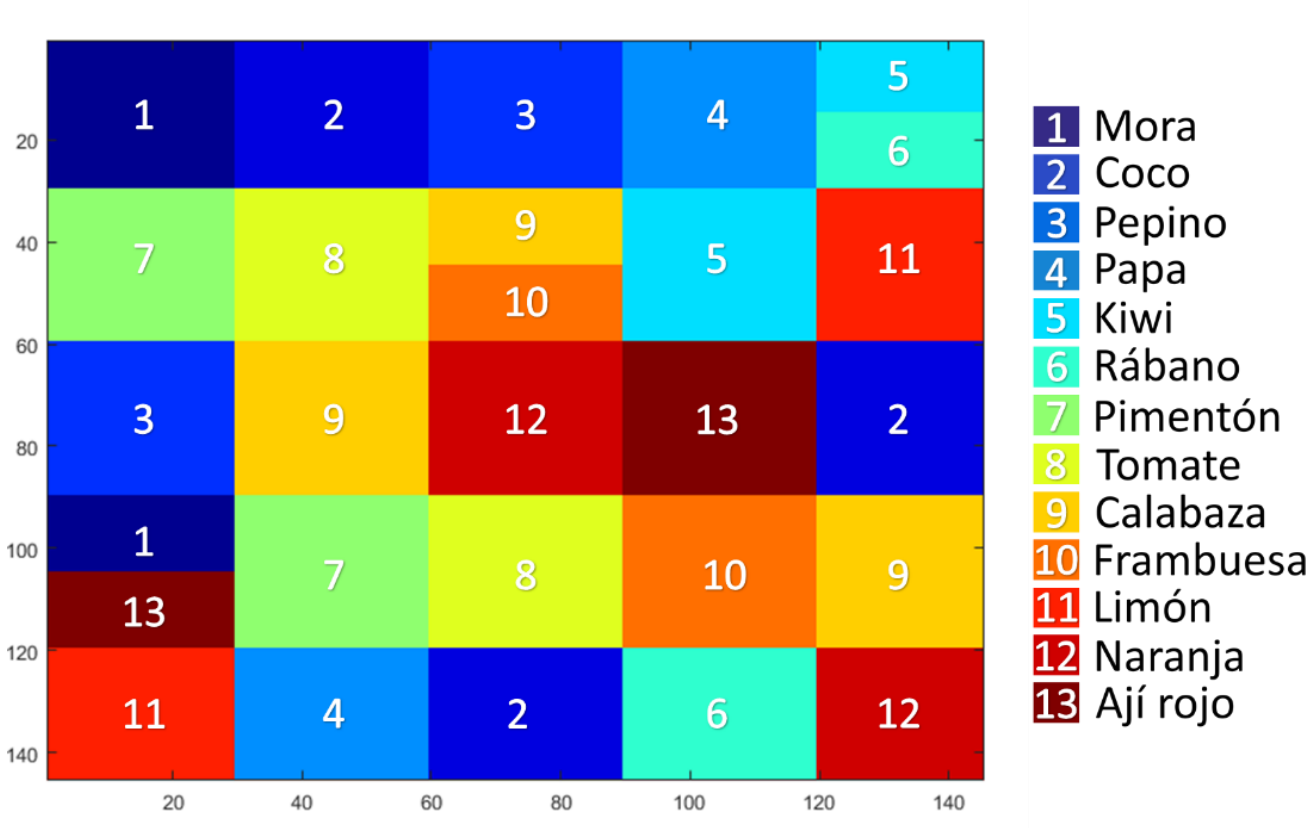


Figura 7. Imagen de etiquetas para la imagen hiperespectral ZR.

3.3.2 Generación de imagen hiperespectral e imagen de etiquetas con zonas irregulares.

Para la segunda imagen hiperespectral combinada se recortaron zonas irregulares de las imágenes de las muestras por medio de la función imfreehand de Matlab® que permite

realizar trazos como el usuario desee. Estas zonas recortadas se reorganizaron en la imagen final como se muestra en la figura 8. Esta imagen la llamaremos Imagen Hiperespectral de Zonas Irregulares (ZI). Ante estas geometrías irregulares, se quería comprobar si la red neuronal convolucional de dos dimensiones se veía afectada por la distribución irregular de las clases (zonas irregulares). La figura 8 muestra la imagen para el canal 100 utilizando el procedimiento descrito para visualizar canales específicos de una imagen hiperespectral. La figura 9 muestra la imagen de etiquetas para las 14 clases de muestras contenidas en la imagen hiperespectral ZI. Para esta imagen ZI aparece una nueva clase con la etiqueta 0, que corresponde a aquellas zonas donde no hay muestras y equivale a zonas de separaciones entre las zonas recortadas.

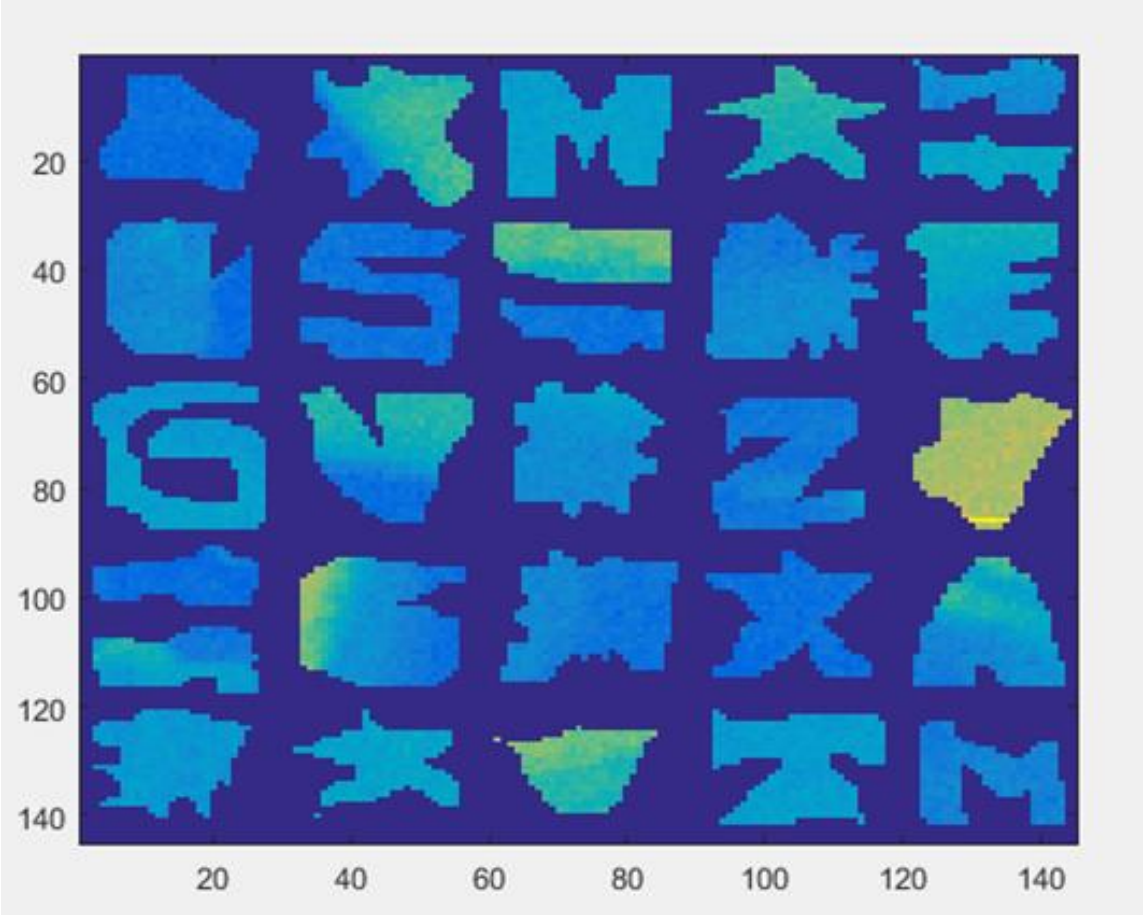


Figura 8. Imagen del canal 100 de la imagen hiperespectral ZI generada a partir de zonas irregulares.

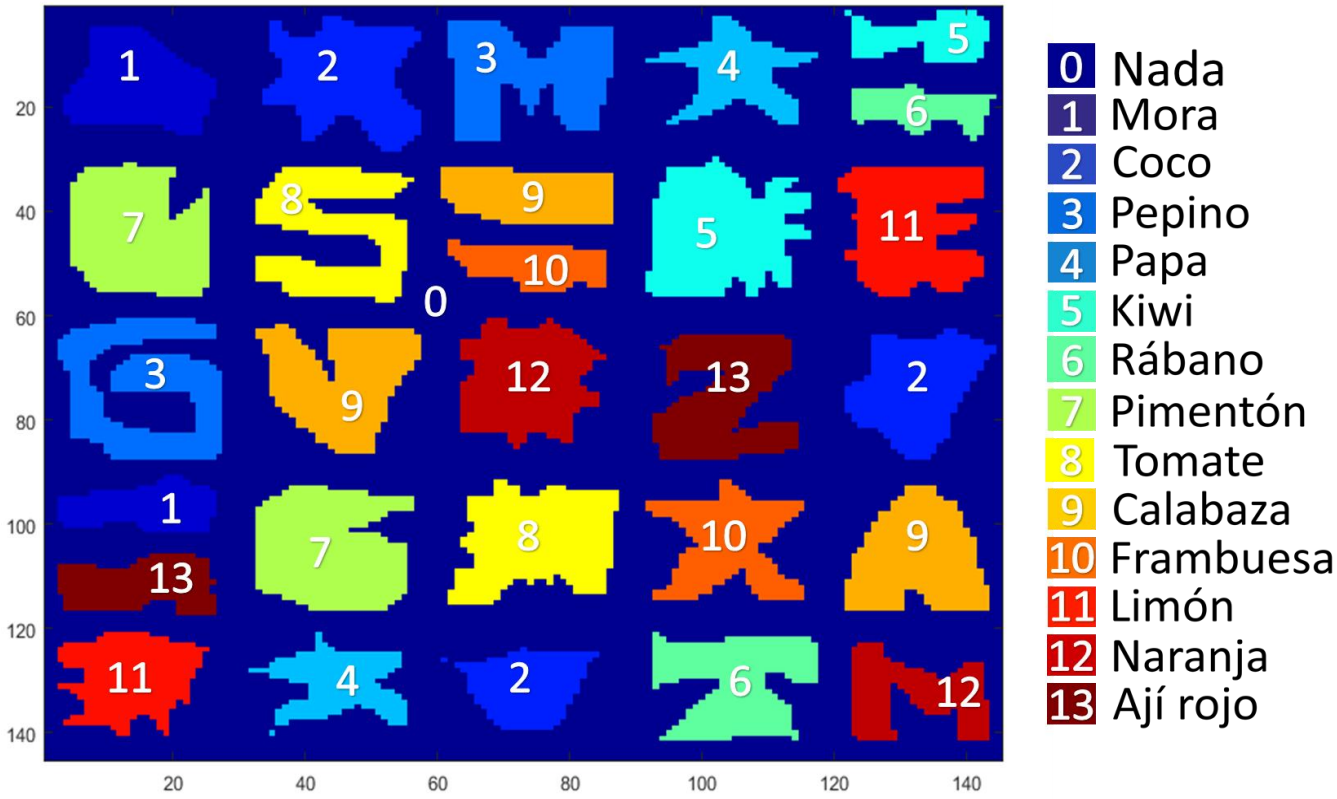


Figura 9. Imagen de etiquetas para la imagen hiperespectral ZI.

3.3.3 Generación de imagen hiperespectral e imagen de etiquetas con zonas irregulares dentro de otra zona irregular.

Para la última imagen hiperespectral combinada se recortaron zonas irregulares de las imágenes de las muestras por medio de la función imfreehand de Matlab® que permite realizar trazos como el usuario desee. En este caso se permitió que dentro de zonas de una clase se incluyeran zonas irregulares más pequeñas de otra clase de muestra, lo que hace que algunas zonas provenientes de una muestra quedaran dentro de otra zona como se muestra en la figura 10. Esta imagen la llamaremos Imagen Hiperespectral de Zonas Irregulares dentro de otra Zona (ZIZ). Con esta distribución espacial específica de las muestras se quería comprobar si la red neuronal convolucional de dos dimensiones cambiaba en su desempeño de clasificación ante el aumento en la complejidad de la distribución y el tamaño de las muestras. En la figura 10 se observa la imagen para el canal 100 utilizando el procedimiento descrito para visualizar canales específicos de una imagen hiperespectral. La figura 11 muestra la imagen de etiquetas para las 14 clases de muestras contenidas en la imagen hiperespectral ZIZ. Al igual que en la imagen ZI, en esta imagen ZIZ aparece una nueva clase con la etiqueta 0 ya que existe separación entre las zonas recortadas.

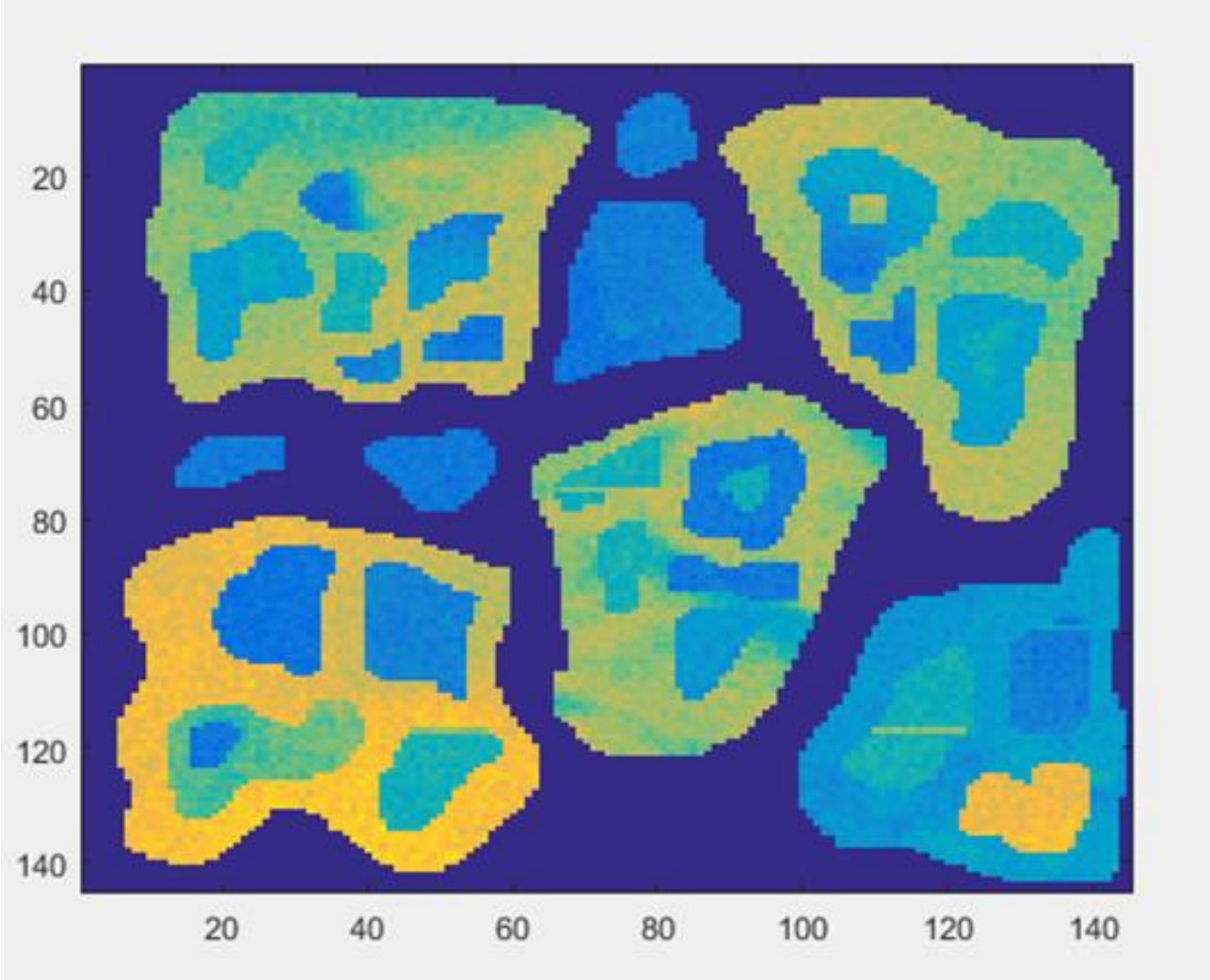


Figura 10. Imagen del canal 100 de la ultima imagen hiperespectral ZIZ generada a partir de zonas irregulares dentro de otra zona irregular.

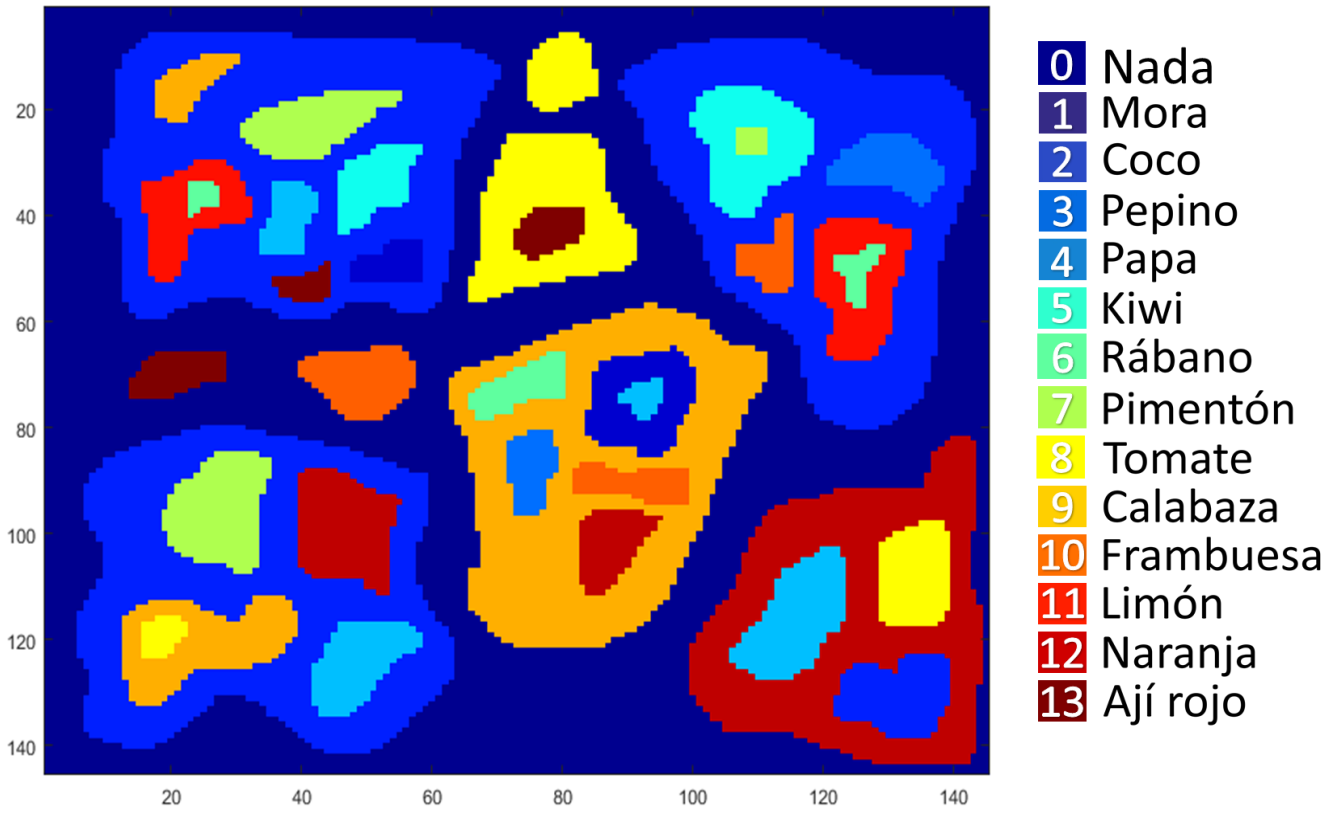


Figura 11. Imagen de etiquetas para la imagen hiperespectral ZIZ.

3.4 Entrenamiento y predicción de la 2DCNN

El desarrollo para el entrenamiento y evaluación de acierto en la predicción de la 2DNN se lleva a cabo siguiendo una arquitectura donde se proponen tres fases para su composición (X. Yang et al., 2018): extracción de parches, extracción de características e identificación de etiquetas. Véase la figura 12.

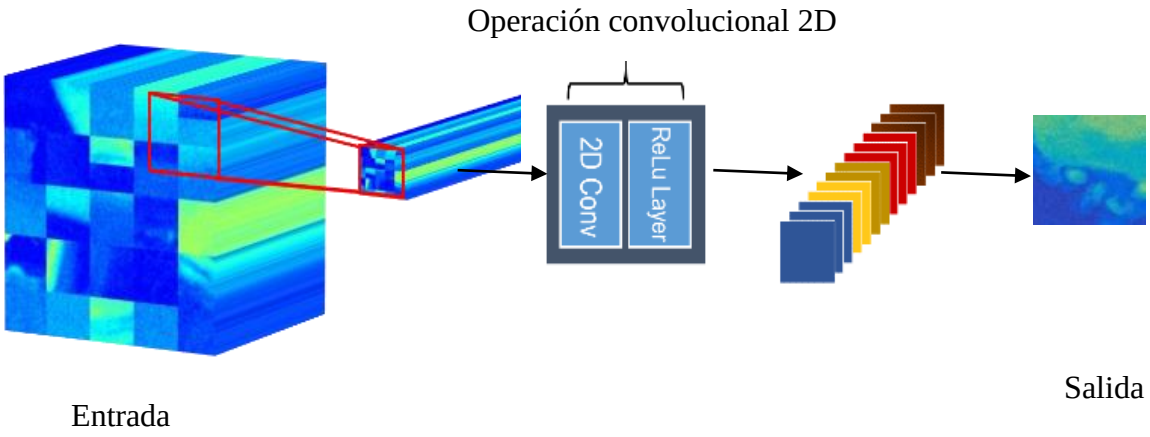


Figura 12. Muestra el funcionamiento del modelo de la 2D-CNN. El recuadro rojo muestra el kernel o parche espacial para realizar la operación convolucional 2D que se lleva a cabo para todas las bandas espectrales, es decir, para las 200 bandas que poseen las tres imágenes hiperespectrales ZR, ZI y ZIZ generadas en la sección 3.3. Enseguida se obtiene la extracción de características que realiza la operación convolucional 2D para obtener mapas de características. Finalmente se da la clasificación como salida.

De esta manera el objetivo es realizar una predicción de las etiquetas de cada píxel de la imagen hiperespectral, mediante las tres fases mencionadas. Es decir, la 2D-CNN se centra en recorrer la imagen de manera espacial en cada una de las bandas espectrales, con un kernel o parche para extraer las características espaciales. Al final de la arquitectura se da la predicción de acuerdo a las características extraídas. Usando el entorno Python y utilizando las librerías: TensorFlow, matplotlib, scipy, pillow, opencv-python, se implementa la arquitectura 2D-CNN. Esta arquitectura está compuesta por: la capa de entrada, tres capas convolucionales 2D, tres capas de activación, una capa totalmente conectada y para mostrar los resultados de predicción probabilística de las clases en general, se le adapta una capa de función softmax. Esta arquitectura no posee la capa típica de agrupación ya que se requiere mantener la mayor cantidad de datos posibles, es decir, tantos píxeles como sea posible de la imagen. Véase la figura 13.

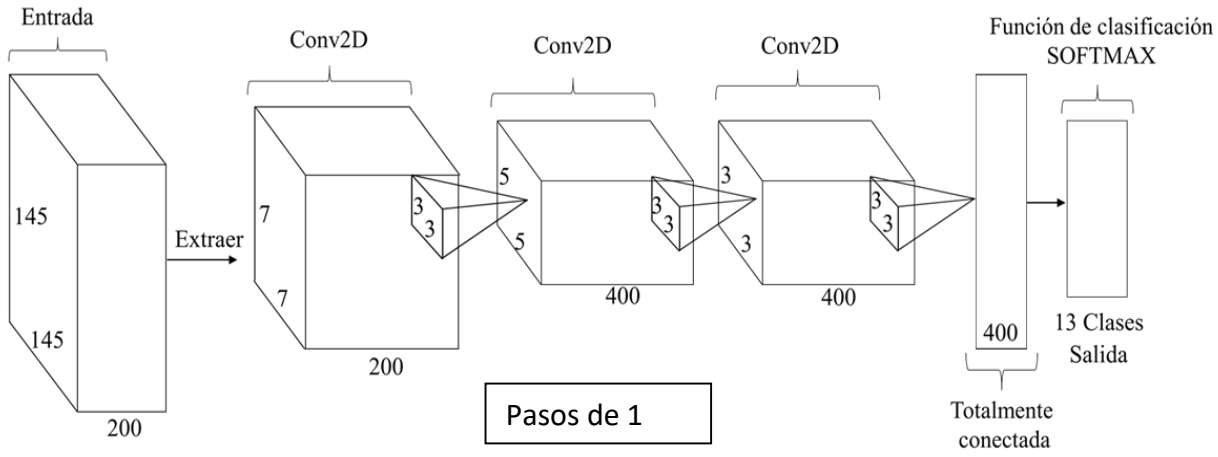


Figura 13. Se muestra la arquitectura de la arquitectura 2D-CNN, que se compone de: la capa de entrada de la imagen hiperespectral de donde se extraerán las características, seguido de tres capas de convolución, seguido de la capa totalmente conectada, y por último la capa de función de clasificación general SOFTMAX.

4. RESULTADOS Y DISCUSIÓN

Se realizó con satisfacción la generación de las imágenes hiperespectrales con distintas formas geométricas para ver cómo se desempeñaba la arquitectura 2DCNN en cuanto a la predicción en cada una de ellas. Al probar esta red en diferentes imágenes no solamente podemos ver su funcionamiento en imágenes con muestras de formas convencionales, sino también en imágenes con muestras de formas irregulares y entremezcladas que hagan que la clasificación se pueda ver comprometida por la complejidad espacial en la distribución de las clases a identificar.

Las pruebas de la red neuronal convolucional de dos dimensiones se realizaron con el 80% de los datos de entrada para el entrenamiento, dejando un 20% para la respectiva prueba de clasificación. Para el entrenamiento y las pruebas de predicción se utilizó un ordenador *acer* con procesador Intel core i5 de séptima generación y con 4 Gigas de memoria RAM. Los resultados de estas pruebas se obtuvieron usando el procesamiento de la CPU, ya que la tarjeta gráfica disponible en el equipo no era compatible para la implementación de esta arquitectura. Bajo estas condiciones el tiempo de entrenamiento de la red fue de 13 a 15 horas aproximadamente.

Para evaluar el rendimiento del aprendizaje profundo con respecto a las técnicas convencionales de clasificación del aprendizaje de máquinas se hace una comparación con la técnica de máquinas de soporte vectorial (SVM). A continuación, se muestran los resultados para la clasificación de las imágenes hiperespectrales con sus imágenes de etiquetas. Para el desarrollo de los resultados se mostrará la imagen resultado de clasificación vs la imagen de etiquetas, seguido de una tabla de comparación entre los porcentajes de predicción de la SVM y 2DCNN. Para el resultado de la 2DCNN y de SVM en la imagen hiperespectral ZR, véase la figura 14 y 15. Para el resultado de la 2DCNN y de SVM

en la imagen hiperespectral ZI, véase la figura 16 y 17 y para para el resultado de la 2DCNN y de SVM en la última imagen hiperespectral ZIZ, véase la figura 18 y 19. La comparación global y el análisis entre estas dos técnicas, véase la tabla 4.

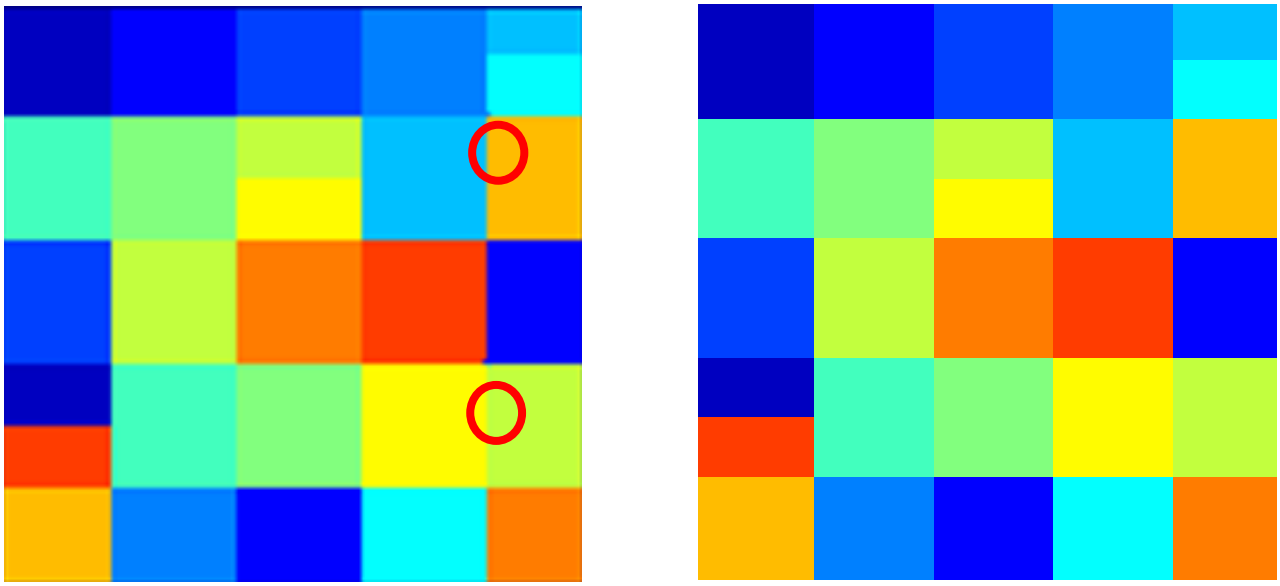


Imagen de clasificación Imagen de etiquetas

Figura 14. Clasificación de la 2DCNN en la imagen hiperespectral ZR con una precisión de 99.922%. Los puntos rojos de la imagen izquierda señalan los errores de predicción.

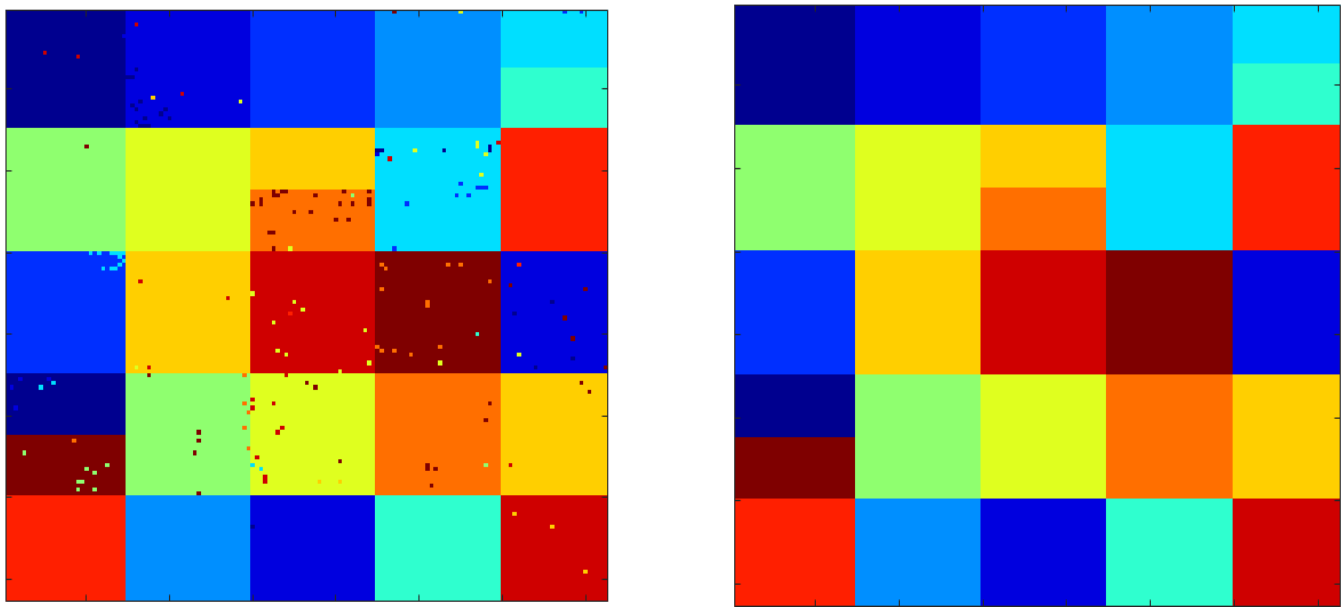


Imagen de clasificación Imagen de etiquetas

Figura 15. Clasificación de SVM en la imagen hiperespectral ZR con una precisión de 85.52%.

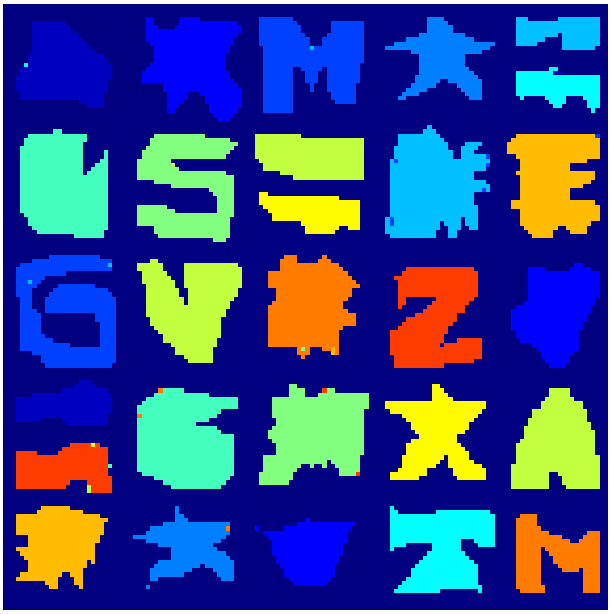


Imagen de clasificación

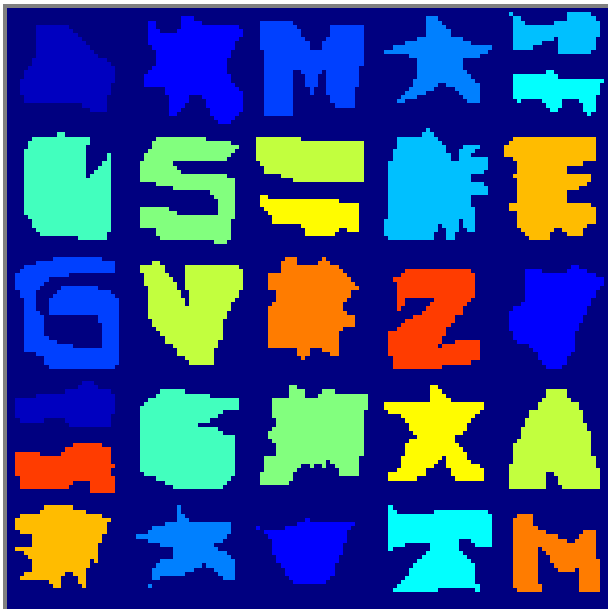


Imagen de etiquetas

Figura 16. Clasificación de la 2DCNN en la imagen hiperespectral ZI con una precisión de 98.56%.

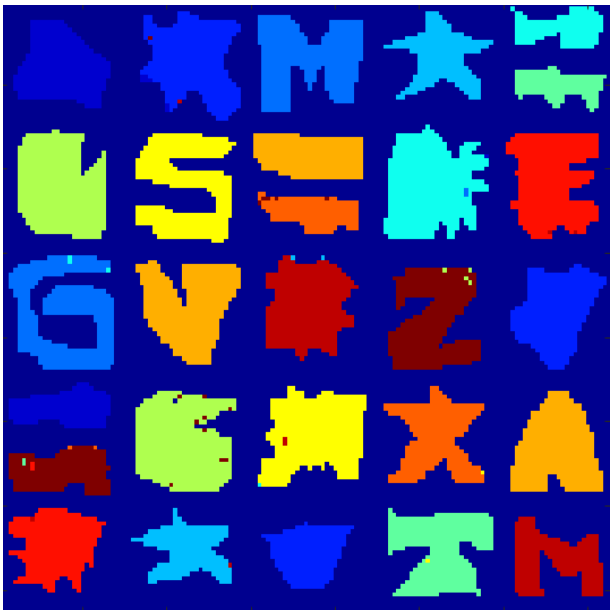


Imagen de predicción

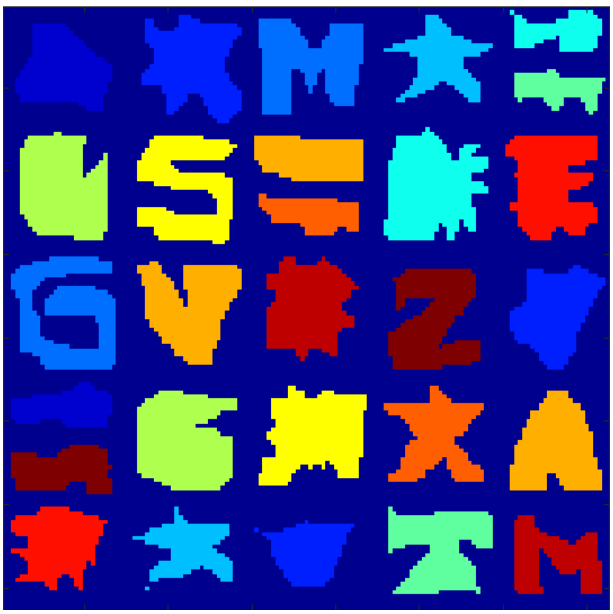


Imagen de etiquetas

Figura 17. Clasificación de SVM en la imagen hiperespectral ZI con una precisión de 99.57%.

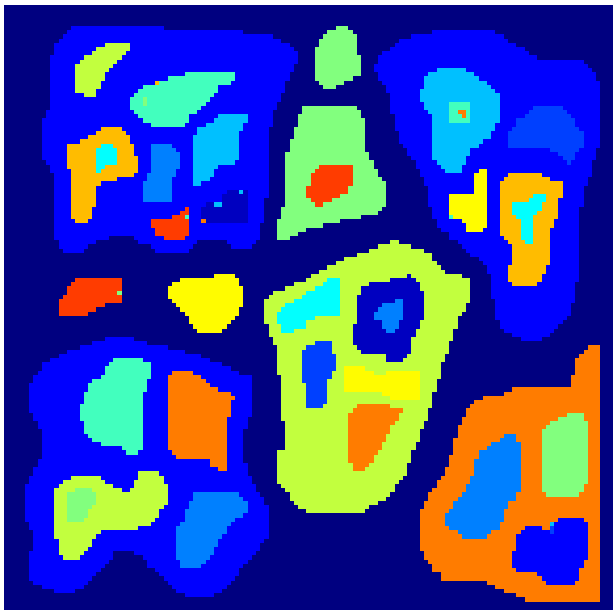


Imagen de clasificación

Imagen de etiquetas

Figura 18. Clasificación de la 2DCNN en la imagen hiperespectral ZIZ con una precisión de 99.244%.

Imagen de clasificación

Imagen de etiquetas

Figura 19. Clasificación de SVM en la imagen hiperespectral ZIZ con una precisión de 28.62%.

Tabla 4. Primera comparación de resultados entre SVM y 2DCNN

Precisión	SVM	2DCNN
Clasificación en la imagen hiperespectral ZR	**85.52%**	**99.922%**
Clasificación en la imagen hiperespectral ZI	**99.57%**	**98.56%**
Clasificación en la imagen hiperespectral ZIZ	**28.62%**	**99.244%**

Los resultados de la tabla 4 muestran la capacidad que posee la arquitectura 2DCNN para imágenes con muestras de diferentes formas y distribuciones o mezclas como son las imágenes ZR, ZI y ZIZ generadas en este trabajo. Mientras que las SVM solo obtienen buenos resultados en los casos dónde las clases están bien separadas espacialmente como sucede en la imagen ZI. Se puede decir que ante una imagen de mayor complejidad como la ZIZ las SVM son incapaces de crear fronteras de clasificación no lineales que den cuenta de esta complejidad. Esto muestra que la 2DCNN tiene en cuenta no solo la información espacial, sino también la información espectral y es capaz de combinarla adecuadamente para clasificar las muestras en contextos complejos espacialmente.

Ahora se muestran los resultados de test de clasificación sobre datos diferentes a los usados en el entrenamiento para la generación del modelo de predicción. Es decir, se realizaron una totalidad de 6 cruces de acuerdo a la tabla 5 con la finalidad de ver el impacto en el comportamiento de predicción de un modelo entrenado con una base de datos en pruebas con bases de datos diferentes.

Tabla 5. Orden de los cruces en pruebas de clasificación de una imagen hiperespectral con datos diferentes a los usados en el entrenamiento del modelo de predicción.

Numeración para pruebas de clasificación cruzadas	**Test con datos de imagen ZR**	**Test con datos de imagen ZI**	**Test con datos de imagen ZIZ**
Modelo generado con imagen ZR	----------	1A figura 20	2B figura 21
Modelo generado con imagen ZI	2A figura 22	----------	2B figura 23
Modelo generado con imagen ZIZ	3A figura 24	3B Figura25	----------

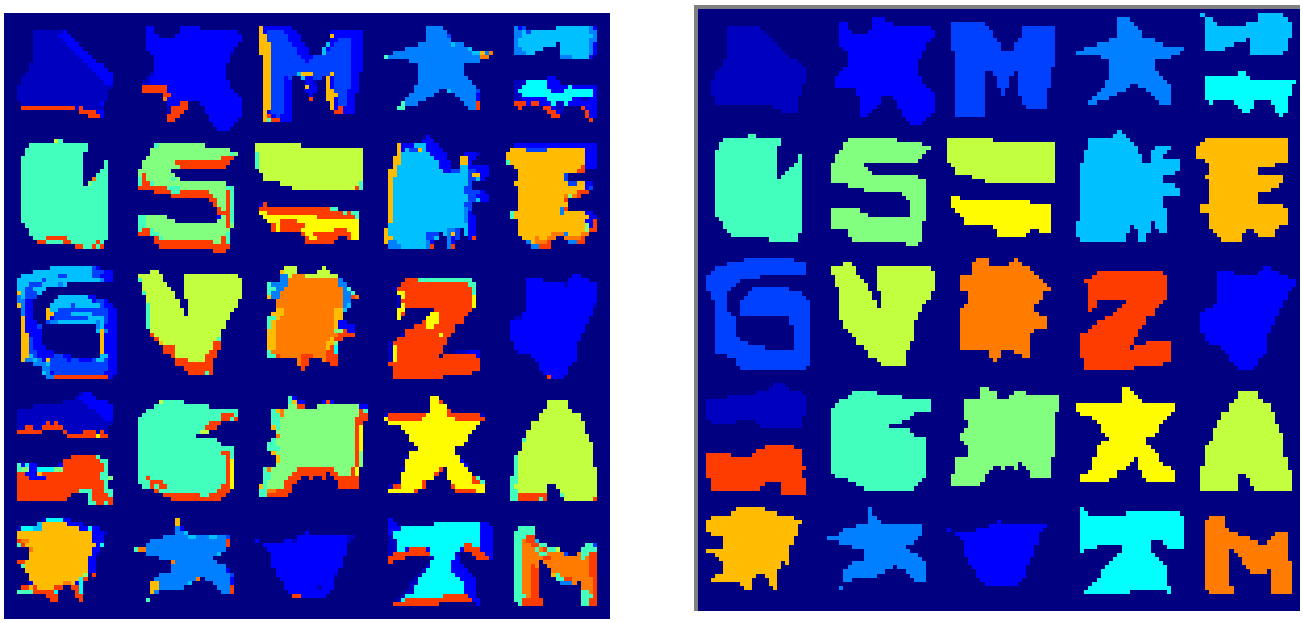


Imagen de clasificación Base de datos

Figura 20. Resultado de clasificación del modelo entrenado con la imagen hiperespectral ZR en la base de datos de la imagen hiperespectral ZI con un precisión de 74.07%. Cruce 1A

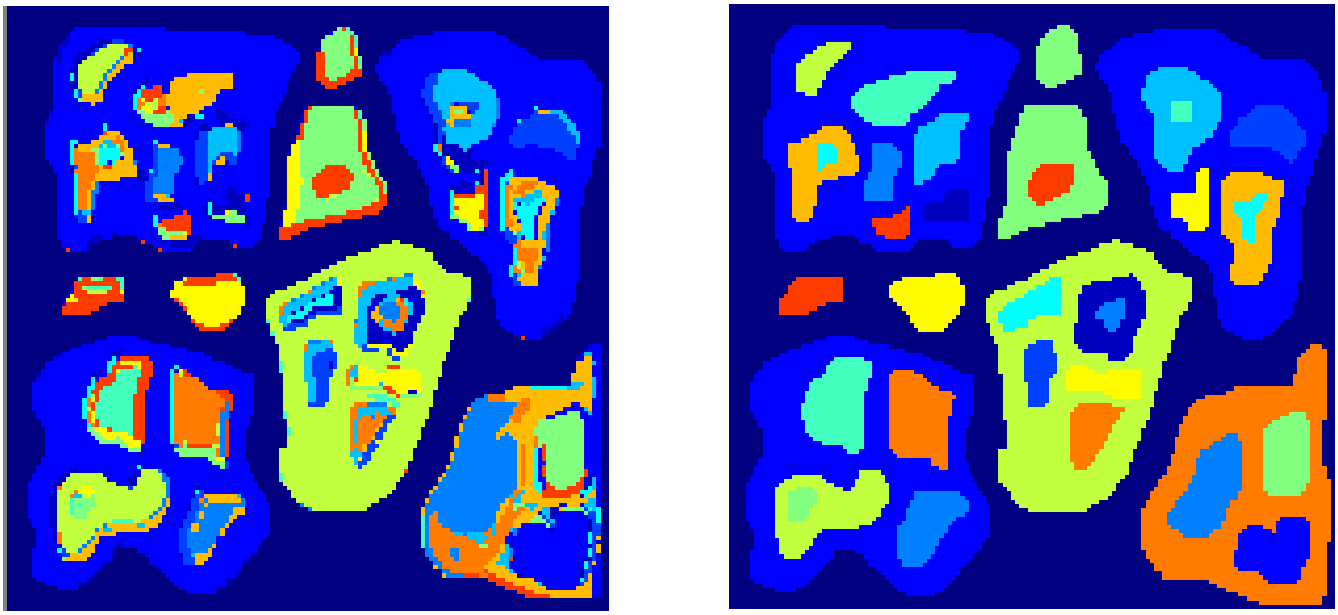


Imagen de clasificación Base de datos

Figura 21. Resultado de clasificación del modelo entrenado con la imagen hiperespectral ZR en la base de datos de la imagen hiperespectral ZIZ con una precisión de 73.129%. Cruce 1B.

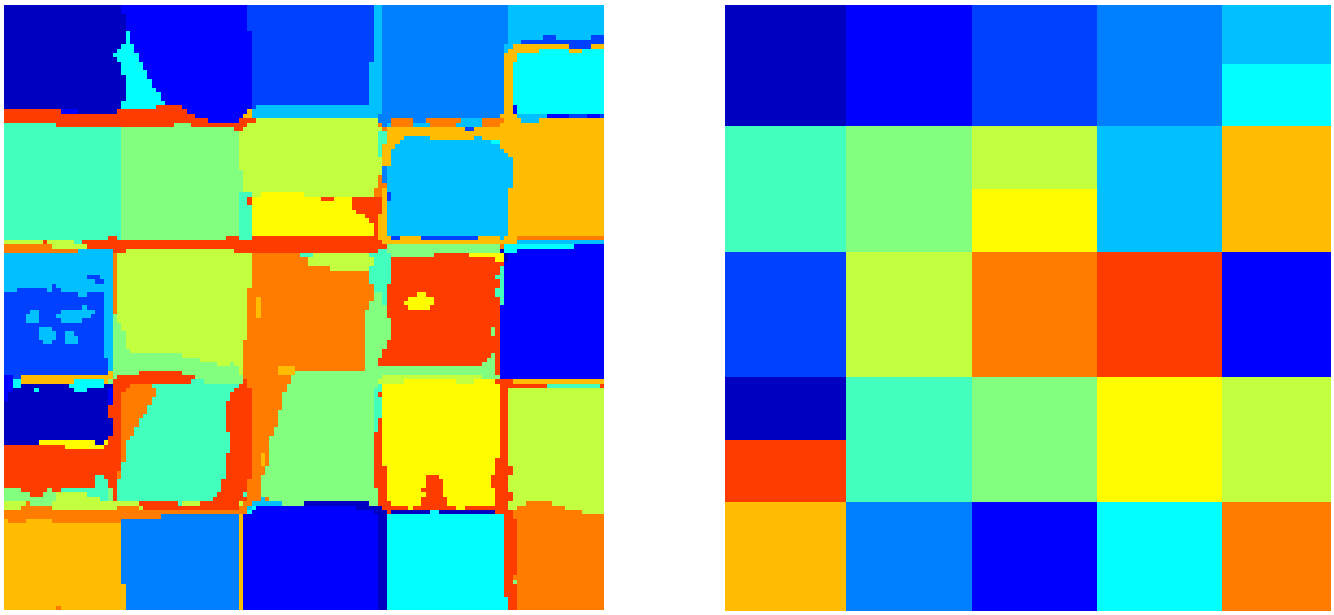


Imagen de clasificación Base de datos

Figura 22. Resultado de clasificación del modelo entrenado con la imagen hiperespectral ZI en la base de datos de la imagen hiperespectral ZR con una precisión de 81.50%. Cruce 2A.

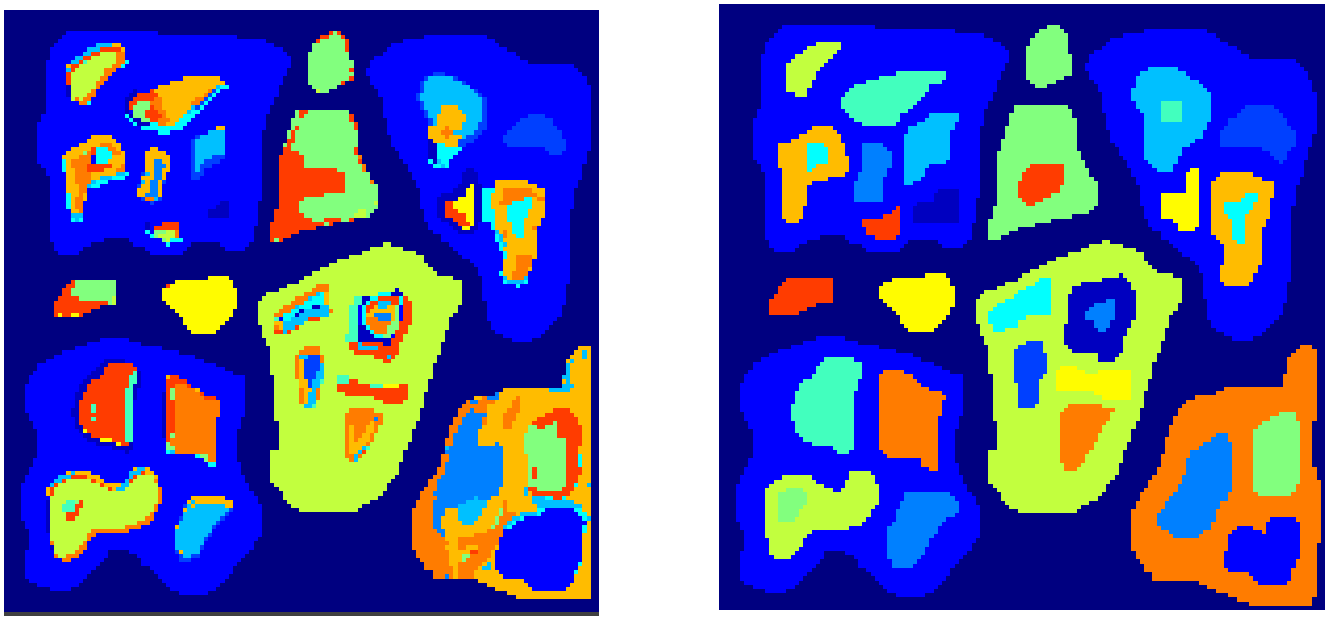


Imagen de clasificación Base de datos

Figura 23. Resultado de clasificación del modelo entrenado con la imagen hiperespectral ZI en la base de datos de la imagen hiperespectral ZIZ con una precisión de 69.84%. Cruce 2B.

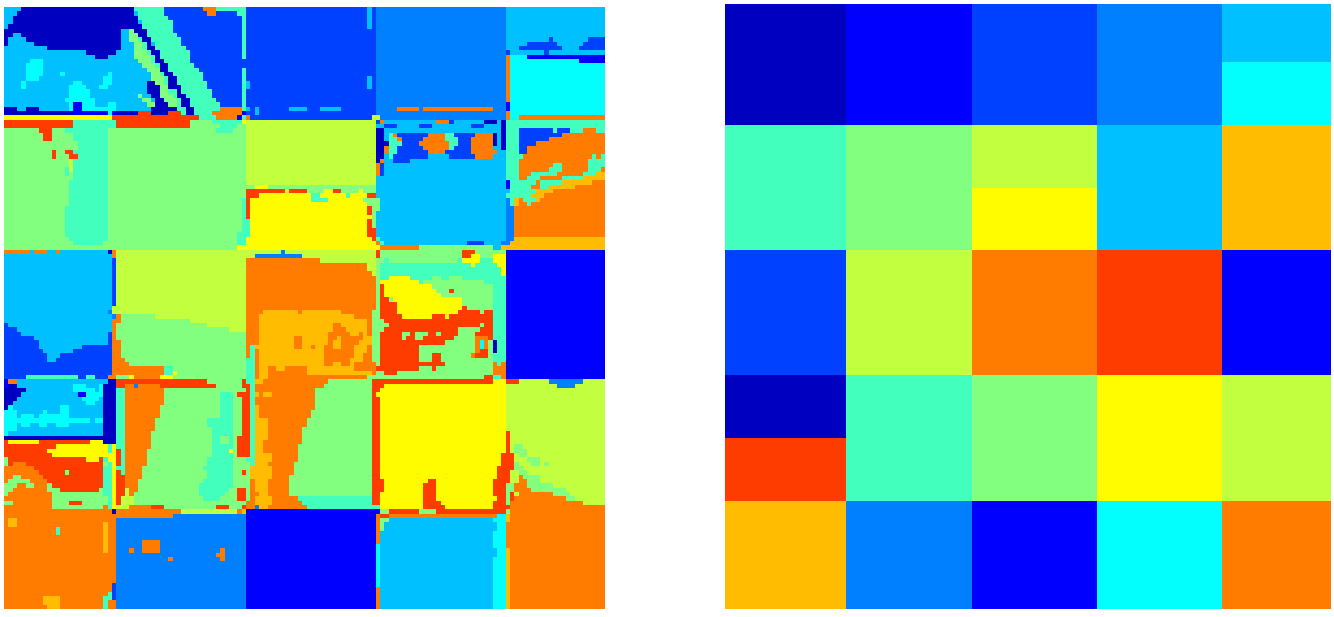


Imagen de clasificación Base de datos

Figura 24. Resultado de clasificación del modelo entrenado con la imagen hiperespectral ZIZ en la base de datos de la imagen hiperespectral ZR con una precisión de 57.981%. Cruce 3A.

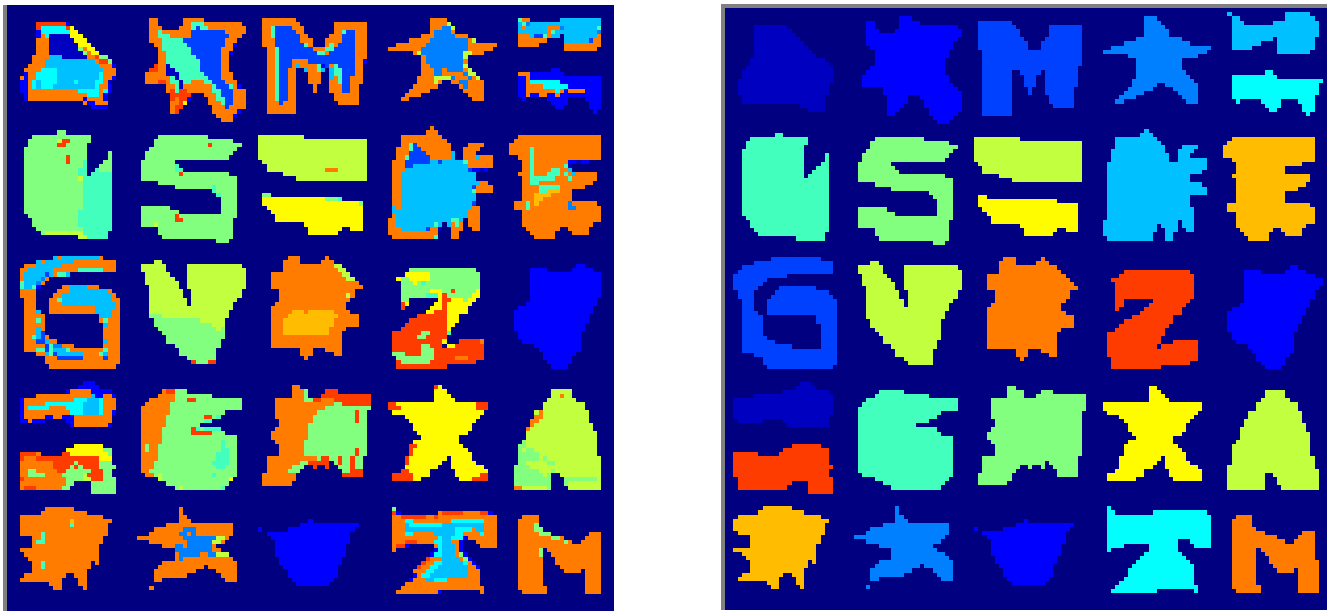


Imagen de clasificación Base de datos

Figura 25. Resultado de clasificación del modelo entrenado con la imagen hiperespectral ZIZ en la base de datos de la imagen hiperespectral ZI con una precisión de 43.338%%. Cruce 3B

Tabla 6. Comparación de los resultados de clasificación para los cruces de los modelos entrenados de la 2DCNN usando conjuntos de prueba de otras bases de datos.

Modelo de clasificación	**Test con datos de imagen ZR**	**Test con datos de imagen ZI**	**Test con datos de imagen ZIZ**
Modelo generado con imagen ZR	99.922%	**74.07%**	**73.129%**
Modelo generado con imagen ZI	**81.50%**	98.56%	**69.84%**
Modelo generado con imagen ZIZ	**57.981%**	**43.338%**	99.244%

Como se puede inferir de los resultados de la tabla 6, los modelos generados por la 2DCNN aprenden tanto de la información espectral como espacial, de manera que son dependientes no solo de los espectros de cada clase, que son similares en todas las tres bases de datos, sino también de las formas o figuras que posee la base de datos, haciendo que el modelo clasifique más o menos de una manera específica de acuerdo a la base de datos de entrenamiento. Esa especificidad es evidente, como se mostró, en los resultados de clasificación sin cruces de conjuntos de datos de prueba y entrenamiento donde se supera el 98% para las tres bases de datos de este trabajo. Los cruces dan a mostrar que, aunque tengan las mismas clases para la clasificación y haya similitud espectral entre los datos en cada una de las bases de datos usadas para entrenamiento, los modelos aprenden para la clasificación tanto de la información espectral como espacial y por eso los resultados no son tan buenos en estos casos de cruces.

5. CONCLUSIONES, RECOMENDACIONES Y TRABAJO FUTURO

En este trabajo se realizó la implementación de aprendizaje profundo para la clasificación de frutas y verduras por medio de la arquitectura 2DCNN. Los resultados dan una clara muestra del poder que poseen estas arquitecturas y su robustez para la clasificación en imágenes hiperespectrales de frutas y verduras. Los tres modelos entrenados logran clasificar a un porcentaje general superior al 98%, esto se debe a las operaciones que realizan las capas convolucionales en la recolección de características de manera espacial en su recorrido por todas las bandas espectrales haciendo que en todos estos datos encuentre patrones con los cuales logra clasificar de manera óptima.

De la misma manera se puede determinar que para la clasificación de imágenes hiperespectrales a través de una técnica clásica como lo son las máquinas de soporte vectorial en datos que tienen cierta complejidad espacial, imagen hiperespectral ZIZ, el cubo que esta arquitectura genera por medio de hiperplanos no logra separar adecuadamente las clases, obteniendo un bajo resultado de clasificación de 28.62% respecto a la 2DCNN.

Con respecto a los 6 cruces de conjuntos de entrenamiento y prueba que se realizaron, se puede decir que la 2DCNN realiza predicciones que en general son positivas, dando un resultado promedio de 75%. Aunque este resultado a su vez muestra que los modelos son dependientes de la distribución espacial que posean las clases en las bases de datos con las que fueron entrenados, ya que no se alcanzan los niveles de predicción dados en el caso del uso de conjuntos de entrenamiento y prueba sacados de la misma imagen hiperespectral.

En un trabajo posterior se espera hacer estas comparaciones con cruces de conjuntos de entrenamiento y prueba considerando que se iguale la cantidad de muestras por cada clase en cada uno de los conjuntos de entrenamiento, además de asegurar que estos puntos provengan de las mismas regiones de la imagen hiperespectral original de la muestra, y de

esta manera poder determinar cuál modelo es más robusto a identificar clases sin que haya influencia por variaciones del contenido espectral de las clases.

Con este trabajo se muestran las posibilidades del aprendizaje profundo en estas tareas de clasificación y por tanto se espera continuar en un trabajo futuro con la implementación de otras arquitecturas como la 3DCNN o las arquitecturas recurrentes RCNN de dos y de tres dimensiones.

REFERENCIAS

Alberto Ruiz Marta Susana Basualdo Autor, C., & Jorge Matich, D. (n.d.). *Cátedra: Informática Aplicada a la Ingeniería de Procesos-Orientación I Redes Neuronales: Conceptos Básicos y Aplicaciones*. Retrieved from https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monograias/matich-redesneuronales.pdf

Burger, J. E., & Gowen, A. A. (2011). The interplay of chemometrics and hyperspectral chemical imaging. *2011 3rd Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*, 1–4. https://doi.org/10.1109/WHISPERS.2011.6080856

Burges, C. J. C. (2009). Dimension Reduction: A Guided Tour. *Foundations and Trends® in Machine Learning*, *2*(4), 275–364. https://doi.org/10.1561/2200000002

Deng, L. (2014). A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Transactions on Signal and Information Processing*, *3*, e2. https://doi.org/10.1017/atsip.2013.9

Ennis, R., Schiller, F., Toscani, M., & Gegenfurtner, K. R. (2018). Hyperspectral database of fruits and vegetables. *Journal of the Optical Society of America A*, *35*(4), B256. https://doi.org/10.1364/JOSAA.35.00B256

Hughes, G. (1968). On the mean accuracy of statistical pattern recognizers. *IEEE Transactions on Information Theory*, *14*(1), 55–63. https://doi.org/10.1109/TIT.1968.1054102

Ma, A., Filippi, A. M., Wang, Z., & Yin, Z. (2019). Hyperspectral image classification using similarity measurements-based deep recurrent neural networks. *Remote Sensing*, *11*(2), 1–19. https://doi.org/10.3390/rs11020194

Melgani, F., & Bruzzone, L. (2004). Classification of hyperspectral remote sensing images

with support vector machines. *IEEE Transactions on Geoscience and Remote Sensing*, *42*(8), 1778–1790. https://doi.org/10.1109/TGRS.2004.831865

Noor, S. S. M., Michael, K., Marshall, S., Ren, J., Tschannerl, J., & Kao, F. J. (2016). The properties of the cornea based on hyperspectral imaging: Optical biomedical engineering perspective. *International Conference on Systems, Signals, and Image Processing*, *2016–June*, 1–4. https://doi.org/10.1109/IWSSIP.2016.7502710

Pan, B., Shi, Z., & Xu, X. (2018). MugNet: Deep learning for hyperspectral image classification using limited samples. *ISPRS Journal of Photogrammetry and Remote Sensing*, *145*, 108–119. https://doi.org/10.1016/j.isprsjprs.2017.11.003

Pereira, F. C., & Borysov, S. S. (2018). Machine Learning Fundamentals. In *Mobility Patterns, Big Data and Transport Analytics* (pp. 9–29). https://doi.org/10.1016/b978-0-12-812970-8.00002-6

Ping Duan, Chengjun Ding, Minglu Zhang, & Genqun Cui. (2006). Study on Layered Intelligent Control for Mobile Manipulator Based on Neural Network. *2006 6th World Congress on Intelligent Control and Automation*, 2705–2708. https://doi.org/10.1109/WCICA.2006.1712855

Shuiwang Ji, & Jieping Ye. (2008). Generalized Linear Discriminant Analysis: A Unified Framework and Efficient Model Selection. *IEEE Transactions on Neural Networks*, *19*(10), 1768–1782. https://doi.org/10.1109/tnn.2008.2002078

Wang, J., Zhang, L., Tong, Q., & Sun, X. (2012). The Spectral Crust project—Research on new mineral exploration technology. *2012 4th Workshop on Hyperspectral Image and Signal Processing (WHISPERS)*, 1–4. https://doi.org/10.1109/WHISPERS.2012.6874254

Wang, Q., Meng, Z., & Li, X. (2017). Locality Adaptive Discriminant Analysis for Spectral-Spatial Classification of Hyperspectral Images. *IEEE Geoscience and Remote Sensing Letters*, *14*(11), 2077–2081. https://doi.org/10.1109/LGRS.2017.2751559

Wang, Q., Yuan, Z., Du, Q., & Li, X. (2019). GETNET: A General End-to-End 2-D CNN Framework for Hyperspectral Image Change Detection. *IEEE Transactions on*

Geoscience and Remote Sensing, *57*(1), 3–13. https://doi.org/10.1109/TGRS.2018.2849692

Wendel, A., & Underwood, J. (2016). Self-supervised weed detection in vegetable crops using ground based hyperspectral imaging. *Proceedings - IEEE International Conference on Robotics and Automation*. https://doi.org/10.1109/ICRA.2016.7487717

Yang, J., Zhao, Y., Chan, J. C.-W., & Yi, C. (2016). Hyperspectral image classification using two-channel deep convolutional neural network. *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, 5079–5082. https://doi.org/10.1109/IGARSS.2016.7730324

Yang, X., Ye, Y., Li, X., Lau, R. Y. K., Zhang, X., & Huang, X. (2018). Hyperspectral Image Classification With Deep Learning Models. *IEEE Transactions on Geoscience and Remote Sensing*, *56*(9), 5408–5423. https://doi.org/10.1109/TGRS.2018.2815613

APÉNDICE

APÉNDICE A

En esta sección se muestra el código con el que se realizó pre procesamiento de imágenes.

```
clear all;

Pathfiles =
'C:\Users\acer\Desktop\Universidad\DeepLearning\Trabajo\RAW\i
nternals';

name = 'internals_blackberry.raw';

filename = strcat(Pathfiles,'\',name);

fp = fopen(filename);

h = 400;
w = 800;
p=1282;
A = zeros(w,h,p);

for i=1:p;
A(:,:,i) = fread(fp,[w h],'uint16',0,'l');
disp(i)
end
fclose(fp);
% As = zeros(1271,800,400);
% for i=1:400
%     temporal = A(:,i,:);
%     temporal = shiftdim(temporal,2);
%     As(:,:,i)= temporal;
% end
Blackberry=shiftdim(A,2);%Colocar numero de bandas en la
posicion de profundidad
RS=Blacberry(583:664,259:328,:);%Recorte de imagen(y,x)
```

```
J=imrotate(RS,90); %Rotar la data derecha
save Blueberryaranadano.mat J -v7.3;% guardar la data
R = As(:,:,[242]);%mostrar la imagen en la banda 242

% B =
multibandread(filename,[1271,800,400],'uint16',0,'bil','ieee-
le');
R2=J(:,:,[150]);
```

APÉNDICE B

En esta sección se muestra el código con el que se realizó las imágenes hiperespectrales y las imágenes de etiquetas.

```
clear all
clc
load 'Imagecapri.mat'
figure
imagesc(Caprich(:,:,100));

%%OJOOOOO
h1 = imfreehand; % imfreehand
h1co = imfreehand;
h1coco = imfreehand;
%%
h2 = imfreehand; % imfreehand Blacberry
h2mora = imfreehand;
%%
h3 = imfreehand; % imfreehand
h3kiwy = imfreehand;
%%
h4 = imfreehand; % imfreehand
h4Lemon=imfreehand;
%%
h5 = imfreehand; % imfreehand
h5Orange= imfreehand;
%%
h6 = imfreehand; % imfreehand
h6pepino=imfreehand;
%%
h7 = imfreehand; % imfreehand
h7pimenton=imfreehand;
```

```
%%
h8 = imfreehand; % imfreehand
h8potato=imfreehand;
%%
h9 = imfreehand; % imfreehand
h9calabaza=imfreehand;
h9calabaza1=imfreehand;
%%
h10 = imfreehand; % imfreehand
h10Rabano=imfreehand;
%%
h11 = imfreehand; % imfreehand
h11Aji= imfreehand;
%%
h12 = imfreehand; % imfreehand
h12Tomate= imfreehand;
%%
h13 = imfreehand; % imfreehand
h13frambuesa=imfreehand;
%%
%Coco 1
bfruta_c = createMask(h1); %multiplicar por la clase para la
imagen de labels
bfruta_co = createMask(h1co);
bfruta_coc = createMask(h1coco);
bfruta_coco=bfruta_c+bfruta_co+bfruta_coc;
%%
%Mora 2
bfruta_Mora = createMask(h2)+createMask(h2mora);
%%
%Kiwy 3
bfruta_Kiwyberry = createMask(h3)+createMask(h3kiwy);
%%
%Lemon 4
bfruta_Lemon = createMask(h4)+createMask(h4Lemon);
%%
%Orange 5
bfruta_Orange = createMask(h5)+createMask(h5Orange);
%%
%Pepino 6
bfruta_Pepino = createMask(h6)+createMask(h6pepino);
%%
%Pimenton 7
bfruta_Pimenton = createMask(h7)+createMask(h7pimenton);
%%
%Potato 8
```

```
bfruta_Potato = createMask(h8)+createMask(h8potato);
%%
%Calabaza 9
bfruta_Calabaza =
createMask(h9)+createMask(h9calabaza)+createMask(h9calabaza1)
;
%%
%Rabano 10
bfruta_Rabano = createMask(h10)+createMask(h10Rabano);
%%
%Aji 11
bfruta_Aji = createMask(h11)+createMask(h11Aji);
%%
%Tomate 12
bfruta_Tomatec = createMask(h12)+createMask(h12Tomate);
%%
%Frambuesa 13
bfruta_Raspberry = createMask(h13)+createMask(h13frambuesa);

%%
%Máscara
figure
Mask=bfruta_coco+bfruta_Mora+bfruta_Kiwyberry+bfruta_Lemon+bf
ruta_Orange;
Mask1=bfruta_Pepino+bfruta_Pimenton+bfruta_Potato+bfruta_Cala
baza+bfruta_Rabano;
Image_mask=bfruta_Aji+bfruta_Tomatec+bfruta_Raspberry+Mask+Ma
sk1;
imagesc(Image_mask);


%%
%Labels
blabel_1=2*bfruta_coco+1*bfruta_Mora+5*bfruta_Kiwyberry+11*bf
ruta_Lemon;
blabel_2=12*bfruta_Orange+3*bfruta_Pepino+7*bfruta_Pimenton+4
*bfruta_Potato;
blabel_3=9*bfruta_Calabaza+13*bfruta_Aji+6*bfruta_Rabano+8*bf
ruta_Tomatec+10*bfruta_Raspberry;
Image_labels=blabel_1+blabel_2+blabel_3;

% cocoRedMask = cocoRed.*bw;
for i=1:1:200
    Hyper(:,:,i)=Caprich(:,:,i).*Image_mask;
end
```

```
%
% for i=1:2:200
%     Orange1(:,:,i)=Orange(:,:,i);
% end
```

APÉNDICE C

En esta sección se muestra el código con el que se realizó el entrenamiento y clasificación de las imágenes hiperespectrales.

- Leer los archivos .mat

```
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 25 10:29:12 2018

@author: madcas
"""
from scipy.io import loadmat
import cv2
import numpy as np

data = loadmat('Imagecapri.mat')
dataFull= data['Caprich']
#
data = loadmat('Imagecapri.mat')
dataCorrected= data['Caprich']
#
```

```
data = loadmat('Caprich_gt.mat')
dataGT= data['Caprich_gt']



img_gt = cv2.applyColorMap(dataGT*16, cv2.COLORMAP_JET)

maxV= np.amax(dataFull[...])
print(maxV)
img= np.array(dataFull[...,100]/6,dtype=np.uint8)
img = cv2.applyColorMap(img*4, cv2.COLORMAP_JET)

cv2.imshow("image",img)
cv2.imshow("image GT",img_gt)
k = cv2.waitKey(0)
```

- **Generación de la base de datos**

```
# -*- coding: utf-8 -*-
"""
Created on Tue Apr 24 15:26:20 2018

@author:
"""

from random import shuffle
import glob
import h5py
import numpy as np
import cv2
```

```
from scipy.io import loadmat
import cv2
from cnn_utils import *


shuffle_data = True  # shuffle the addresses before saving
hdf5_path = 'datasetHyp_13fruits.hdf5'  # address to where you want to save the hdf5 file

data= readFiles_hyp()

dataFull, dataCorrected, dataGT=data

f=7 #tamaño del patch
v= int((f-1)/2)
stride= 1
P=0


#numPatchs= int(((dataCorrected.shape[0] - f + 2*P)/stride + 1)**2)
numPatchs=sum(sum(dataGT[v:dataCorrected.shape[0]-v,        v:dataCorrected.shape[1]-
v]!=0))

images= np.zeros((numPatchs,f,f,dataCorrected.shape[2]))
labels= np.zeros((numPatchs,1))
indices= np.zeros((numPatchs,2))

cont=0
```

```
for j in range(3,dataCorrected.shape[0]-v,stride):
   for i in range(3,dataCorrected.shape[1]-v,stride):
      if(dataGT[j,i]!=0):
         images[cont,...]= dataCorrected[j-v:j+v+1,i-v:i+v+1,:]
         labels[cont,0]= dataGT[j,i]
         indices[cont,0]=j
         indices[cont,1]=i
         cont+=1
#print(cont)
index_test= np.arange(numPatchs)
np.random.shuffle(index_test)

images=images[index_test,...]
labels=labels[index_test,...]
indices=indices[index_test,...]

#
#
#print(cont)
train_shape = (len(images[0:int(0.8*len(images)),...]), f, f, dataCorrected.shape[2])
test_shape = (len(images[int(0.8*len(images)):,...]), f, f, dataCorrected.shape[2])
#
##
hdf5_file = h5py.File(hdf5_path, mode='w')
###
###
hdf5_file.create_dataset("train_img", train_shape, np.uint16)
hdf5_file.create_dataset("test_img", test_shape, np.uint16)
```

```
hdf5_file["train_img"][...] = images[0:int(0.8*len(images)),...]
hdf5_file["test_img"][...] = images[int(0.8*len(images)):,...]

hdf5_file.create_dataset("train_labels", (len(labels[0:int(0.8*len(images)),:]),1), np.uint8)
hdf5_file["train_labels"][...] = labels[0:int(0.8*len(images)),:]

hdf5_file.create_dataset("test_labels", (len(labels[int(0.8*len(images)):,:]),1), np.uint8)
hdf5_file["test_labels"][...] = labels[int(0.8*len(images)):,:]

hdf5_file.create_dataset("indices", (numPatchs,2), np.uint8)
hdf5_file["indices"][...] = indices

#
hdf5_file.close()
###
```

- **Guardar modelo 2DCNN**

```
# -*- coding: utf-8 -*-
"""
Editor de Spyder

Este es un archivo temporal
"""

import tensorflow as tf
import numpy as np
from tensorflow.python.saved_model import tag_constants
```

```
import tensorflow as tf
# device_name = tf.test.gpu_device_name()
# if device_name != '/device:GPU:0':
#   raise SystemError('GPU device not found')
# print('Found GPU at: {}'.format(device_name))

# Create some variables.
v1 = tf.get_variable("v1", shape=[3], initializer = tf.zeros_initializer)
v2 = tf.get_variable("v2", shape=[5], initializer = tf.zeros_initializer)

inc_v1 = v1.assign(v1+1)
dec_v2 = v2.assign(v2-1)

# Add an op to initialize the variables.
init_op = tf.global_variables_initializer()

# Add ops to save and restore all the variables.
saver = tf.train.Saver()

# Later, launch the model, initialize the variables, do some work, and save the
# variables to disk.
with tf.Session() as sess:
  sess.run(init_op)
  # Do some work with the model.
  inc_v1.op.run()
  dec_v2.op.run()
```

```
 # Save the variables to disk.
 save_path = saver.save(sess,
"C:/Users/acer/Desktop/Universidad/DeepLearning/Trabajo/Codigo_profe/2D_CNN/tmp/
fruits13_softmax.ckpt")
  print("Model saved in path: %s" % save_path)

tf.reset_default_graph()
```

- **Restaurar modelo 2DCNN**

```
# -*- coding: utf-8 -*-
"""
Created on Fri Oct 26 11:33:35 2018

@author: madcas
"""

import tensorflow as tf
import numpy as np
from tensorflow.python.saved_model import tag_constants

tf.reset_default_graph()

# Create some variables.
v1 = tf.get_variable("v1", shape=[3])
v2 = tf.get_variable("v2", shape=[5])

# Add ops to save and restore all the variables.
saver = tf.train.Saver()
```

```
# Later, launch the model, use the saver to restore variables from disk, and
# do some work with the model.
with tf.Session() as sess:
  # Restore variables from disk.
  saver.restore(sess,
"C:/Users/acer/Desktop/Universidad/DeepLearning/Trabajo/Codigo_profe/2D_CNN/tmp/
fruits13_softmax.ckpt")
  print("Model restored.")
  # Check the values of the variables
  print("v1 : %s" % v1.eval())
  print("v2 : %s" % v2.eval())
```

- **Código de funciones para cargar datos**

```
import math
import numpy as np
import h5py
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.python.framework import ops
from scipy.io import loadmat
import cv2

def load_dataset():
    train_dataset = h5py.File('datasets/train_signs.h5', "r")
    train_set_x_orig = np.array(train_dataset["train_set_x"][:]) # your train set features
    train_set_y_orig = np.array(train_dataset["train_set_y"][:]) # your train set labels
```

```
    test_dataset = h5py.File('datasets/test_signs.h5', "r")
    test_set_x_orig = np.array(test_dataset["test_set_x"][:]) # your test set features
    test_set_y_orig = np.array(test_dataset["test_set_y"][:]) # your test set labels

    classes = np.array(test_dataset["list_classes"][:]) # the list of classes

    train_set_y_orig = train_set_y_orig.reshape((1, train_set_y_orig.shape[0]))
    test_set_y_orig = test_set_y_orig.reshape((1, test_set_y_orig.shape[0]))

    return train_set_x_orig, train_set_y_orig, test_set_x_orig, test_set_y_orig, classes


def random_mini_batches(X, Y, mini_batch_size = 64, seed = 0):
    """
    Creates a list of random minibatches from (X, Y)

    Arguments:
    X -- input data, of shape (input size, number of examples) (m, Hi, Wi, Ci)
    Y -- true "label" vector (containing 0 if cat, 1 if non-cat), of shape (1, number of
examples) (m, n_y)
    mini_batch_size - size of the mini-batches, integer
    seed -- this is only for the purpose of grading, so that you're "random minibatches are
the same as ours.

    Returns:
    mini_batches -- list of synchronous (mini_batch_X, mini_batch_Y)
    """
```

```
    m = X.shape[0]                  # number of training examples
    mini_batches = []
    np.random.seed(seed)

    # Step 1: Shuffle (X, Y)
    permutation = list(np.random.permutation(m))
    shuffled_X = X[permutation,:,:,:]
    shuffled_Y = Y[permutation,:]

    # Step 2: Partition (shuffled_X, shuffled_Y). Minus the end case.
    num_complete_minibatches = math.floor(m/mini_batch_size) # number of mini
batches of size mini_batch_size in your partitionning
    for k in range(0, num_complete_minibatches):
        mini_batch_X = shuffled_X[k * mini_batch_size : k * mini_batch_size +
mini_batch_size,:,:,:]
        mini_batch_Y = shuffled_Y[k * mini_batch_size : k * mini_batch_size +
mini_batch_size,:]
        mini_batch = (mini_batch_X, mini_batch_Y)
        mini_batches.append(mini_batch)

    # Handling the end case (last mini-batch < mini_batch_size)
    if m % mini_batch_size != 0:
        mini_batch_X = shuffled_X[num_complete_minibatches * mini_batch_size : m,:,:,:]
        mini_batch_Y = shuffled_Y[num_complete_minibatches * mini_batch_size : m,:]
        mini_batch = (mini_batch_X, mini_batch_Y)
        mini_batches.append(mini_batch)
```

```
    return mini_batches


def convert_to_one_hot(Y, C):
    Y = np.eye(C)[Y.reshape(-1)].T
    return Y


def forward_propagation_for_predict(X, parameters):
    """
    Implements the forward propagation for the model: LINEAR -> RELU -> LINEAR -> RELU -
> LINEAR -> SOFTMAX

    Arguments:
    X -- input dataset placeholder, of shape (input size, number of examples)
    parameters -- python dictionary containing your parameters "W1", "b1", "W2", "b2",
"W3", "b3"
                the shapes are given in initialize_parameters

    Returns:
    Z3 -- the output of the last LINEAR unit
    """

    # Retrieve the parameters from the dictionary "parameters"
    W1 = parameters['W1']
    b1 = parameters['b1']
    W2 = parameters['W2']
    b2 = parameters['b2']
```

```
    W3 = parameters['W3']
    b3 = parameters['b3']
                                        # Numpy Equivalents:
    Z1 = tf.add(tf.matmul(W1, X), b1)                # Z1 = np.dot(W1, X) + b1
    A1 = tf.nn.relu(Z1)                              # A1 = relu(Z1)
    Z2 = tf.add(tf.matmul(W2, A1), b2)               # Z2 = np.dot(W2, a1) + b2
    A2 = tf.nn.relu(Z2)                              # A2 = relu(Z2)
    Z3 = tf.add(tf.matmul(W3, A2), b3)               # Z3 = np.dot(W3,Z2) + b3

    return Z3

def predict(X, parameters):

    W1 = tf.convert_to_tensor(parameters["W1"])
    b1 = tf.convert_to_tensor(parameters["b1"])
    W2 = tf.convert_to_tensor(parameters["W2"])
    b2 = tf.convert_to_tensor(parameters["b2"])
    W3 = tf.convert_to_tensor(parameters["W3"])
    b3 = tf.convert_to_tensor(parameters["b3"])

    params = {"W1": W1,
              "b1": b1,
              "W2": W2,
              "b2": b2,
              "W3": W3,
              "b3": b3}

    x = tf.placeholder("float", [12288, 1])
```

```
    z3 = forward_propagation_for_predict(x, params)
    p = tf.argmax(z3)

    sess = tf.Session()
    prediction = sess.run(p, feed_dict = {x: X})

    return prediction

def readFiles_hyp():

    # data = loadmat('Indian_pines.mat')
    # dataFull = data['indian_pines']
    # #
    # #
    # data = loadmat('Indian_pines_corrected.mat')
    # dataCorrected = data['indian_pines_corrected']
    # #
    # data = loadmat('Indian_pines_gt.mat')
    # dataGT = data['indian_pines_gt']

    data = loadmat('Imagecapri.mat')
    dataFull = data['Caprich']
    #
    data = loadmat('Imagecapri.mat')
    dataCorrected = data['Caprich']
    #
    data = loadmat('Caprich_gt.mat')
```

```
    dataGT = data['Caprich_gt']

    data=(dataFull, dataCorrected, dataGT)

    return data

def load_dataset_hyp():


    hdf5_path = 'datasetHyp_fruits.hdf5'
    subtract_mean = False
    # open the hdf5 file
    hdf5_file = h5py.File(hdf5_path, "r")
    # subtract the training mean
    if subtract_mean:
        mm = hdf5_file["train_mean"][0, ...]
        mm = mm[np.newaxis, ...]
    # Total number of samples
    data_num = hdf5_file["train_img"].shape[0]
    #
    #print(data_num)

    train_img = np.uint16(hdf5_file["train_img"][0:, ...])
    test_img = np.uint16(hdf5_file["test_img"][0:, ...])

    train_labels = np.uint8(hdf5_file["train_labels"][0:, ...])
    test_labels = np.uint8(hdf5_file["test_labels"][0:, ...])
```

```
    indices = np.uint8(hdf5_file["indices"][...])

    hdf5_file.close()

    return train_img, train_labels, test_img, test_labels, indices
```

- **Código de entrenamiento**

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Mon Jun 11 12:50:13 2018

@author: madcas
"""

import math
import numpy as np
import h5py
import matplotlib.pyplot as plt
import scipy
from PIL import Image
from scipy import ndimage
import tensorflow as tf
from tensorflow.python.framework import ops
from cnn_utils import *
from scipy.io import loadmat
import cv2
```

```
#data= readFiles_hyp()
#
#dataFull, dataCorrected, dataGT=data
#
#
##
=============================================================================
====
## Visualize the hyperspectral images
##
=============================================================================
====
#
#index=100
#
#img_gt = cv2.applyColorMap(dataGT*16, cv2.COLORMAP_JET)
#
#img_full= np.array(dataFull[...,index]/4,dtype=np.uint8)
#
#img_full=cv2.applyColorMap(img_full, cv2.COLORMAP_JET)
#
#cv2.imshow("image",img_full)
#cv2.imshow("image GT",img_gt)
#k = cv2.waitKey(0)
#
```

```
##
================================================================================
====



X_train_orig, Y_train_orig, X_test_orig, Y_test_orig, indices= load_dataset_hyp()     #
Cargar la base de datos
#
## Visualizar uno de los ejemplos de la base de datos
#
#index = 6
#plt.figure(1)
#plt.imshow(X_train_orig[index])
#print ("y = " + str(np.squeeze(Y_train_orig[:, index])))



# imprimir características de la base de datos
X_train = X_train_orig/1024
X_test = X_test_orig/1024.
Y_train = convert_to_one_hot((Y_train_orig-1), 16).T
Y_test = convert_to_one_hot((Y_test_orig-1), 16).T
print ("Número de ejemplos de entrenamiento: " + str(X_train.shape[0]))
print ("Número de ejemplos de testing: " + str(X_test.shape[0]))
print ("X_train shape: " + str(X_train.shape))
print ("Y_train shape: " + str(Y_train.shape))
```

```
print ("X_test shape: " + str(X_test.shape))
print ("Y_test shape: " + str(Y_test.shape))
conv_layers = {}



def create_placeholders(n_H0, n_W0, n_C0, n_y):
    """
    Crea los placeholders para la sesión.

    Argumentos:
    n_H0 -- Escalar, height de la imagen de entrada
    n_W0 -- Escalar, width de la imagen de entrada
    n_C0 -- Escalar, Número de canales de entrada
    n_y -- Escalar, Número de clases


    Returna:
    X -- placeholder para los datos de entrada, de tamaño [None, n_H0, n_W0, n_C0] y
dtype "float"
    Y -- placeholder para las etiquetas de entrada, de tamaño [None, n_y] y dtype "float"
    """

    #### Haga su código acá ### (≈2 lines)

    X = tf.placeholder(tf.float32, shape=(None, n_H0, n_W0, n_C0))
    Y = tf.placeholder(tf.float32, shape=(None, n_y))
```

```
    ### Fin ###

    return X, Y


def initialize_parameters():
    """
    Inicializa los parámetros (Pesos) para construir la red neuronal convolucional con
tensorflow. El tamaño es
                    W1 : [4, 4, 3, 8]
                    W2 : [2, 2, 8, 16]
    Returna:
    parameters -- Un diccionario de tensores que contiene W1, W2
    """

    tf.set_random_seed(1)                   #

    #### Haga su código acá ### (≈2 lines)

    W1 = tf.get_variable("W1", [3, 3, 200, 400], initializer =
tf.contrib.layers.xavier_initializer(seed = 0))
    W2 = tf.get_variable("W2", [3, 3, 400, 400], initializer =
tf.contrib.layers.xavier_initializer(seed = 0))
    W3 = tf.get_variable("W3", [3, 3, 400, 400], initializer =
tf.contrib.layers.xavier_initializer(seed = 0))

    ### Fin ###
```

```
    parameters = {"W1": W1,
                  "W2": W2,
                  "W3": W3}

    return parameters


def forward_propagation(X, parameters):
    """
    Implementa la propagación hacia adelante del modelo

    CONV2D -> RELU -> MAXPOOL -> CONV2D -> RELU -> MAXPOOL -> FLATTEN ->
FULLYCONNECTED

    Argumentos:
    X -- placeholder de entrada (ejemplos de entrenamiento), de tamaño (input size,
number of examples)
    parameters -- Diccionario que contiene los parámetros "W1", "W2" desde
initialize_parameters

    Returna:
    Z3 -- Salida de la última unidad LINEAR
    """

    # Obtención de los pesos desde "parameters"
    W1 = parameters['W1']
    W2 = parameters['W2']
    W3 = parameters['W3']
```

```
    #### Haga su código acá ###

    # CONV2D: stride of 1, padding 'SAME'

    Z1 = tf.nn.conv2d(X, W1, strides = [1,1,1,1], padding = 'VALID')

    # RELU

    A1 = tf.nn.relu(Z1)

    # MAXPOOL: window 8x8, stride 8, padding 'SAME'

#    P1 = tf.nn.max_pool(A1, ksize = [1,8,8,1], strides = [1,8,8,1], padding = 'SAME')

    # CONV2D: filters W2, stride 1, padding 'SAME'

    Z2 = tf.nn.conv2d(A1,W2, strides = [1,1,1,1], padding = 'VALID')

    # RELU

    A2 = tf.nn.relu(Z2)

    # MAXPOOL: window 4x4, stride 4, padding 'SAME'

#    P2 = tf.nn.max_pool(A2, ksize = [1,4,4,1], strides = [1,4,4,1], padding = 'SAME')

    Z3 = tf.nn.conv2d(A2,W3, strides = [1,1,1,1], padding = 'VALID')
```

```
    # RELU

    A3 = tf.nn.relu(Z3)

    # FLATTEN

    F = tf.contrib.layers.flatten(A3)

#    # FULLY-CONNECTED without non-linear activation function (not not call softmax).
#    # 6 neurons in output layer. Hint: one of the arguments should be
"activation_fn=None"

    Z4 = tf.contrib.layers.fully_connected(F, 16, None)

    ### Fin ###

    return Z4



def compute_cost(Z4, Y):
    """
    Calcula la función de costo

    Argumentos:
    Z3 -- Salida del forward propagation (Salida de la última unidad LINEAR), de tamaño (6,
Número de ejemplos)
    Y -- placeholders con el vector de etiquetas "true", del mismo tamaño que Z3
```

```
    Returns:
    cost - Tensor de la función de costo
    """

    #### Haga su código acá ### (≈2 lines)

    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits = Z4, labels = Y))

    ### Fin ###

    return cost


def model(X_train, Y_train, X_test, Y_test, learning_rate = 0.008, num_epochs = 500,
minibatch_size = 64, print_cost = True):
    """
    Implementa una Red Neuronal Convolucional de 3-Capas en Tensorflow:
    CONV2D -> RELU -> MAXPOOL -> CONV2D -> RELU -> MAXPOOL -> FLATTEN ->
FULLYCONNECTED

    Argumentos:
    X_train -- Conjunto de entrenamiento, de tamaño (None, 64, 64, 3)
    Y_train -- Etiquetas del conjunto de entrenamiento, de tamaño (None, n_y = 6)
    X_test -- Conjunto de datos de Test, de tamaño (None, 64, 64, 3)
    Y_test -- Etiquetas del conjunto de Test, de tamaño  (None, n_y = 6)
    learning_rate -- factor de aprendizaje en la optimización
    num_epochs -- Número de epocas en el ciclo de optimización
```

```
    minibatch_size -- Tamaño del minibatch
    print_cost -- True: imprime el costo cada 100 epocas

    Returna:
    train_accuracy -- Número Real, Accuracy del conjunto de entrenamiento (X_train)
    test_accuracy -- Número Real, Accuracy del conjunto de Test(X_test)
    parameters -- parameters aprendidos por el modelo. Estos pueden ser usados para predecir.
    """

    ops.reset_default_graph()                    # Permite correr nuevamente el modelo sin sobreescribir las tf variables
    tf.set_random_seed(1)                        #  (tensorflow seed)
    seed = 3                                     #
    (m, n_H0, n_W0, n_C0) = X_train.shape
    n_y = Y_train.shape[1]
    costs = []                                   # Para almacenar el costo

    ##
================================================================================
====
    ## GPU
    ##
================================================================================
====

    # device_name = tf.test.gpu_device_name()
    # if device_name != '/device:CPU:0':
```

```
#   raise SystemError('GPU device not found')
# print('Found GPU at: {}'.format(device_name))

#config = tf.ConfigProto()
#config.gpu_options.allow_growth = True                    #



# Crear los PlaceHolders

X, Y = create_placeholders(n_H0, n_W0, n_C0, n_y)

# Inicializar Parámetros

parameters = initialize_parameters()

# Forward propagation: Construir el forward propagation en el grafo de tensorflow

Z4 = forward_propagation(X, parameters)

# Cost function: Incluir la  función de costo en el grafo de tensorflow

cost = compute_cost(Z4, Y)

# Backpropagation: Define el optimizador. Usar AdamOptimizer para minimizar el costo.

optimizer = tf.train.AdamOptimizer().minimize(cost)
```

```
# Inicializar todas las variables globales

init = tf.global_variables_initializer()

saver = tf.train.Saver()

# Iniciar la sesión
#with tf.Session(config=config) as sess:
with tf.Session() as sess:

    # Run init
    sess.run(init)

    # Loop de entrenamiento
    for epoch in range(num_epochs):

        minibatch_cost = 0.
        num_minibatches = int(m / minibatch_size) # número de minibatches de tamaño
minibatch_size en el conjunto de entrenamiento         seed = seed + 1
        minibatches = random_mini_batches(X_train, Y_train, minibatch_size, seed)


        for minibatch in minibatches:

            # Seleccionar un minibatch

            (minibatch_X, minibatch_Y) = minibatch
```

```
            # IMPORTANT: The line that runs the graph on a minibatch.

            # Run the session to execute the optimizer and the cost, the feedict should
contain a minibatch for (X,Y).

            _ , temp_cost = sess.run([optimizer, cost], {X: minibatch_X, Y: minibatch_Y})

            minibatch_cost += temp_cost / num_minibatches



        # Imprime el costo
        if print_cost == True and epoch % 5 == 0:
            print ("Cost after epoch %i: %f" % (epoch, minibatch_cost))
        if print_cost == True and epoch % 1 == 0:
            costs.append(minibatch_cost)


    # Graficar la función de costo
    plt.figure(2)
    plt.plot(np.squeeze(costs))
    plt.ylabel('cost')
    plt.xlabel('iterations (per tens)')
    plt.title("Learning rate =" + str(learning_rate))
    plt.show()

    # Calcular las predicciones correctas
#       predict_op = tf.argmax(Z4, 1)
#       correct_prediction = tf.equal(predict_op, tf.argmax(Y, 1))
```

```
    predict_op = tf.nn.softmax(Z4)  # Apply softmax to logits
    correct_prediction = tf.equal(tf.argmax(predict_op, 1), tf.argmax(Y, 1))



    # Calcular la predicción sobre el conjunto de test
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
    print(accuracy)
    train_accuracy = accuracy.eval({X: X_train, Y: Y_train})
    test_accuracy = accuracy.eval({X: X_test, Y: Y_test})
    print("Train Accuracy:", train_accuracy)
    print("Test Accuracy:", test_accuracy)

    save_path = saver.save(sess,
"C:/Users/acer/Desktop/Universidad/DeepLearning/Trabajo/Codigo_profe/2D_CNN/tmp/
fruits13_softmax.ckpt")
    print("Model saved in path: %s" % save_path)

#       sess.close()

    return train_accuracy, test_accuracy, parameters

_, _, parameters = model(X_train, Y_train, X_test, Y_test)
```

- Código donde se muestra las imágenes de clasificación

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Mon Jun 11 12:50:13 2018

@author: madcas
"""

import math
import numpy as np
import h5py
import matplotlib.pyplot as plt
import scipy
from PIL import Image
from scipy import ndimage
import tensorflow as tf
from tensorflow.python.framework import ops
from cnn_utils import *
from scipy.io import loadmat
import cv2



#data= readFiles_hyp()
#
#dataFull, dataCorrected, dataGT=data
```

```
#
#
##
================================================================================
====
## Visualize the hyperspectral images
##
================================================================================
====
#
#index=100
#
#img_gt = cv2.applyColorMap(dataGT*16, cv2.COLORMAP_JET)
#
#img_full= np.array(dataFull[...,index]/4,dtype=np.uint8)
#
#img_full=cv2.applyColorMap(img_full, cv2.COLORMAP_JET)
#
#cv2.imshow("image",img_full)
#cv2.imshow("image GT",img_gt)
#k = cv2.waitKey(0)
#
##
================================================================================
====
```

```
X_train_orig, Y_train_orig, X_test_orig, Y_test_orig, indices = load_dataset_hyp()     #
Cargar la base de datos
#
## Visualizar uno de los ejemplos de la base de datos
#
#index = 6
#plt.figure(1)
#plt.imshow(X_train_orig[index])
#print ("y = " + str(np.squeeze(Y_train_orig[:, index])))



# imprimir características de la base de datos
X_train = X_train_orig/1024
X_test = X_test_orig/1024.
Y_train = convert_to_one_hot((Y_train_orig-1), 16).T
Y_test = convert_to_one_hot((Y_test_orig-1), 16).T
print ("Número de ejemplos de entrenamiento: " + str(X_train.shape[0]))
print ("Número de ejemplos de testing: " + str(X_test.shape[0]))
print ("X_train shape: " + str(X_train.shape))
print ("Y_train shape: " + str(Y_train.shape))
print ("X_test shape: " + str(X_test.shape))
print ("Y_test shape: " + str(Y_test.shape))
conv_layers = {}
```

```
def create_placeholders(n_H0, n_W0, n_C0, n_y):
    """
    Crea los placeholders para la sesión.

    Argumentos:
    n_H0 -- Escalar, height de la imagen de entrada
    n_W0 -- Escalar, width de la imagen de entrada
    n_C0 -- Escalar, Número de canales de entrada
    n_y -- Escalar, Número de clases


    Returna:
    X -- placeholder para los datos de entrada, de tamaño [None, n_H0, n_W0, n_C0] y
dtype "float"
    Y -- placeholder para las etiquetas de entrada, de tamaño [None, n_y] y dtype "float"
    """

    #### Haga su código acá ### (≈2 lines)

    X = tf.placeholder(tf.float32, shape=(None, n_H0, n_W0, n_C0))
    Y = tf.placeholder(tf.float32, shape=(None, n_y))

    ### Fin ###

    return X, Y


def initialize_parameters():
```

```
    """
    Inicializa los parámetros (Pesos) para construir la red neuronal convolucional con
tensorflow. El tamaño es
                    W1 : [4, 4, 3, 8]
                    W2 : [2, 2, 8, 16]
    Returna:
    parameters -- Un diccionario de tensores que contiene W1, W2
    """

    tf.set_random_seed(1)                   #

    #### Haga su código acá ### (≈2 lines)

    W1 = tf.get_variable("W1", [3, 3, 200, 400])
    W2 = tf.get_variable("W2", [3, 3, 400, 400])
    W3 = tf.get_variable("W3", [3, 3, 400, 400])

    ### Fin ###

    parameters = {"W1": W1,
                  "W2": W2,
                  "W3": W3}

    return parameters


def forward_propagation(X, parameters):
    """
```

```
    Implementa la propagación hacia adelante del modelo

    CONV2D -> RELU -> MAXPOOL -> CONV2D -> RELU -> MAXPOOL -> FLATTEN ->
FULLYCONNECTED

    Argumentos:
    X -- placeholder de entrada (ejemplos de entrenamiento), de tamaño (input size,
number of examples)
    parameters -- Diccionario que contiene los parámetros "W1", "W2" desde
initialize_parameters

    Returna:
    Z3 -- Salida de la última unidad LINEAR
    """

    # Obtención de los pesos desde "parameters"
    W1 = parameters['W1']
    W2 = parameters['W2']
    W3 = parameters['W3']

    #### Haga su código acá ###

    # CONV2D: stride of 1, padding 'SAME'

    Z1 = tf.nn.conv2d(X, W1, strides = [1,1,1,1], padding = 'VALID')

    # RELU
```

```
    A1 = tf.nn.relu(Z1)

    # MAXPOOL: window 8x8, stride 8, padding 'SAME'

#    P1 = tf.nn.max_pool(A1, ksize = [1,8,8,1], strides = [1,8,8,1], padding = 'SAME')

    # CONV2D: filters W2, stride 1, padding 'SAME'

    Z2 = tf.nn.conv2d(A1,W2, strides = [1,1,1,1], padding = 'VALID')

    # RELU

    A2 = tf.nn.relu(Z2)

    # MAXPOOL: window 4x4, stride 4, padding 'SAME'

#    P2 = tf.nn.max_pool(A2, ksize = [1,4,4,1], strides = [1,4,4,1], padding = 'SAME')

    Z3 = tf.nn.conv2d(A2,W3, strides = [1,1,1,1], padding = 'VALID')

    # RELU

    A3 = tf.nn.relu(Z3)

    # FLATTEN

    F = tf.contrib.layers.flatten(A3)
```

```
#    # FULLY-CONNECTED without non-linear activation function (not not call softmax).
#    # 6 neurons in output layer. Hint: one of the arguments should be
"activation_fn=None"

    Z4 = tf.contrib.layers.fully_connected(F, 16, None)

    ### Fin ###

    return Z4

def compute_cost(Z4, Y):
    """
    Calcula la función de costo

    Argumentos:
    Z3 -- Salida del forward propagation (Salida de la última unidad LINEAR), de tamaño (6,
Número de ejemplos)
    Y -- placeholders con el vector de etiquetas "true", del mismo tamaño que Z3

    Returns:
    cost - Tensor de la función de costo
    """

    #### Haga su código acá ### (≈2 lines)

    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits = Z4, labels = Y))

    ### Fin ###
```

```
    return cost


def model(X_train, Y_train, X_test, Y_test, learning_rate = 0.08, num_epochs = 500,
minibatch_size = 16, print_cost = True):
    """
    Implementa una Red Neuronal Convolucional de 3-Capas en Tensorflow:
    CONV2D -> RELU -> MAXPOOL -> CONV2D -> RELU -> MAXPOOL -> FLATTEN ->
FULLYCONNECTED

    Argumentos:
    X_train -- Conjunto de entrenamiento, de tamaño (None, 64, 64, 3)
    Y_train -- Etiquetas del conjunto de entrenamiento, de tamaño (None, n_y = 6)
    X_test -- Conjunto de datos de Test, de tamaño (None, 64, 64, 3)
    Y_test -- Etiquetas del conjunto de Test, de tamaño  (None, n_y = 6)
    learning_rate -- factor de aprendizaje en la optimización
    num_epochs -- Número de epocas en el ciclo de optimización
    minibatch_size -- Tamaño del minibatch
    print_cost -- True: imprime el costo cada 100 epocas

    Returna:
    train_accuracy -- Número Real, Accuracy del conjunto de entrenamiento (X_train)
    test_accuracy -- Número Real, Accuracy del conjunto de Test(X_test)
    parameters -- parameters aprendidos por el modelo. Estos pueden ser usados para
predecir.
    """
```

```
   ops.reset_default_graph()                    # Permite correr nuevamente el modelo sin
sobreescribir las tf variables
   tf.set_random_seed(1)                        #  (tensorflow seed)
   seed = 3                                     #
   (m, n_H0, n_W0, n_C0) = X_train.shape
   n_y = Y_train.shape[1]
   costs = []                                   # Para almacenar el costo

   ##
==========================================================================
====
   ## GPU
   ##
==========================================================================
====

   # device_name = tf.test.gpu_device_name()
   # if device_name != '/device:GPU:0':
   #   raise SystemError('GPU device not found')
   # print('Found GPU at: {}'.format(device_name))
   #
   # config = tf.ConfigProto()
   # config.gpu_options.allow_growth = True                    #



   # Crear los PlaceHolders
```

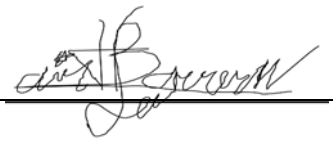
```
    X, Y = create_placeholders(n_H0, n_W0, n_C0, n_y)

    # Inicializar Parámetros

    parameters = initialize_parameters()

    # Forward propagation: Construir el forward propagation en el grafo de tensorflow

    Z4 = forward_propagation(X, parameters)

    # Cost function: Incluir la  función de costo en el grafo de tensorflow

#    cost = compute_cost(Z4, Y)

    # Backpropagation: Define el optimizador. Usar AdamOptimizer para minimizar el costo.

#    optimizer = tf.train.AdamOptimizer().minimize(cost)

    # Inicializar todas las variables globales

#    init = tf.global_variables_initializer()

    saver = tf.train.Saver()

    # Iniciar la sesión
    with tf.Session() as sess:

        # Run init
```

```
    saver.restore(sess,
"C:/Users/acer/Desktop/Universidad/DeepLearning/Trabajo/Codigo_profe/2D_CNN/tmp/
fruits13_softmax.ckpt")
    print("Model restored.")
#     sess.run(init)

    # Loop de entrenamiento
#     for epoch in range(num_epochs):
#
#         minibatch_cost = 0.
#         num_minibatches = int(m / minibatch_size) # número de minibatches de tamaño
minibatch_size en el conjunto de entrenamiento         seed = seed + 1
#         minibatches = random_mini_batches(X_train, Y_train, minibatch_size, seed)
#
#
#         for minibatch in minibatches:
#
#             # Seleccionar un minibatch
#
#             (minibatch_X, minibatch_Y) = minibatch
#
#             # IMPORTANT: The line that runs the graph on a minibatch.
#
#             # Run the session to execute the optimizer and the cost, the feedict should
contain a minibatch for (X,Y).
#
#             _ , temp_cost = sess.run([optimizer, cost], {X: minibatch_X, Y: minibatch_Y})
```

```
#
#             minibatch_cost += temp_cost / num_minibatches
#
#
#         # Imprime el costo
#         if print_cost == True and epoch % 5 == 0:
#             print ("Cost after epoch %i: %f" % (epoch, minibatch_cost))
#         if print_cost == True and epoch % 1 == 0:
#             costs.append(minibatch_cost)


    # Graficar la función de costo
#     plt.figure(2)
#     plt.plot(np.squeeze(costs))
#     plt.ylabel('cost')
#     plt.xlabel('iterations (per tens)')
#     plt.title("Learning rate =" + str(learning_rate))
#     plt.show()

    # Calcular las predicciones correctas
#     predict_op = tf.argmax(Z4, 1)
    predict_op = tf.nn.softmax(Z4)
    correct_prediction = tf.equal(tf.argmax(predict_op, 1), tf.argmax(Y, 1))

#     correct_prediction = tf.equal(predict_op, tf.argmax(Y, 1))

    # Calcular la predicción sobre el conjunto de test
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

```
#       print(accuracy)
        train_accuracy = accuracy.eval({X: X_train, Y: Y_train})
        test_accuracy = accuracy.eval({X: X_test, Y: Y_test})
        X_total= np.concatenate((X_train,X_test),axis=0)
        Y_total= np.concatenate((Y_train,Y_test),axis=0)
        pred=predict_op.eval({X: X_total, Y:Y_total})
        print("Train Accuracy:", train_accuracy)
        print("Test Accuracy:", test_accuracy)

#       save_path = saver.save(sess,
"/Users/acer/Desktop/Universidad/DeepLearning/Trabajo/Codigo_profe/2D_CNN/tmp/m
odel.ckpt")
#       print("Model saved in path: %s" % save_path)

#       sess.close()

        return train_accuracy, test_accuracy, parameters, pred

_, _, parameters, pred = model(X_train, Y_train, X_test, Y_test)

#ImagenResult=pred.reshape(139,height,1);



#Y_total= np.concatenate((Y_train,Y_test),axis=0)

pred=np.argmax(pred[...], axis=1)
```

```
#=====================================================

images= np.zeros((145, 145), dtype=np.uint8)

for k in range(len(indices)):
   images[indices[k,0],indices[k,1]]= pred[k]+1

images = cv2.applyColorMap(images*16, cv2.COLORMAP_JET)
cv2.imshow("imaimages_predge",images)



data= readFiles_hyp()

dataFull, dataCorrected, dataGT=data



#
=========================================================================
====
# Visualize the hyperspectral images
#
=========================================================================
====

img_gt = cv2.applyColorMap(dataGT*16, cv2.COLORMAP_JET)

cv2.imshow("image GT",img_gt)
k = cv2.waitKey(0)
```

FIRMA ESTUDIANTES 

FIRMA ASESOR 

FECHA ENTREGA: 30-05-2019

FIRMA COMITÉ TRABAJO DE GRADO DE LA FACULTAD _____

RECHAZADO___ ACEPTADO____ ACEPTADO CON MODIFICACIONES_____

ACTA NO._____

FECHA ENTREGA: _____

FIRMA CONSEJO DE FACULTAD_____

ACTA NO._____

FECHA ENTREGA: _____