**Universidad de La Habana**

Facultad de Matemática y
Computación

# Aprendizaje Profundo en la clasificación de Imágenes

*Proyecto Final de Inteligencia Artificial*

Integrantes:
Sheyla Cruz Castro C-512
Laura Brito Guerrero C-512
Ariel Antonio Huerta Martín C-512

# Índice general

# Capítulo 1

# Introducción

## 1.1. Presentación del objetivo

*Deep Learning in images classification (Aprendizaje Profundo en la clasificación de imágenes)*

Se tiene un dataset de imágenes tomadas por microscopios de barrido, cada conjunto de imágenes se encuentra clasificada. El objetivo principal dada una imagen arbitraria, es identificar si pertenece a alguna clasificación del dataset que se tiene. A grandes rasgos, se pretende realizar la clasificación de imágenes mediante el aprendizaje profundo. La optimalidad de los casos correctos de la aplicación va a ser directamente proporcional con el tiempo asignado al proyecto. Para generar estos fines, se va a llevar a cabo la implementación en python, con el auxilio de la biblioteca TensorFlow. Con respecto a la relación que tiene este trabajo con otras asignaturas se puede decir que es bastante amplia. Vamos a emplear conocimiento de procesamiento de imágenes, tales como la convolución, funciones activadoras (en este caso: ReLu), y la técnica maxpolling para disminuir el tamaño de la imagen y así mantener una eficiencia espacial. Por otra parte, el aprendizaje profundo consta de una estructura de datos basada en grafos, donde en una parte específica tendremos como bien conocida es: *Fully Connected Layers*, que no es más que un grafo bipartito completo el cual interviene en el proceso de la clasificación. Y siendo más específico en la parte de la clasificación, se utilizarán modelos probabilistas. Por lo tanto, a modo de resumen tenemos que su relación con otras asignaturas es perenne, algunas de estas son : Procesamiento de Imágenes, Probabilidades, Estructura de datos y algoritmos.

## 1.2. Estructura

En la carpeta **src** se encuentra la implementación del proyecto. Dentro se encuentran los ficheros **train.py**, el cual es el primero que se tiene que ejecutar

pues el mismo se encarga de entrenar el modelo, proceso en el cual la máquina aprende de las imágenes de entrenamiento y evalúa la precisión con respecto a las imágenes de validación; y el otro fichero es **predict.py** el cual resuelve la problemática de la clasificación leyendo el modelo anteriormente entrenado dada una imagen de entrada arbitraria.
Se encuentra la carpeta **dataset**, en la cual se encuentran los conjuntos de imágenes de entrenamiento y de validación respectivamente. En este proyecto se va a clasificar en tres clases: *Plasmodium Falciparum(Malaria)*; *Sars-Cov2(Covid-19)*; y *Vibrio Cholerae(Cólera)*. Esto conlleva que exista una carpeta con imágenes de cada clase en cada conjunto.

# Capítulo 2

# Introducción a las redes neuronales profundas

## 2.1. Un poco de teoría ...

*Deep Convolutional Neural Networks (Redes neuronales convolucionales profundas)* La historia de las CNN profundas se remonta a principios de la década de 1980, pero solo hasta la década de 2010, los investigadores lograron una alta precisión en la resolución de tareas de reconocimiento de imágenes con redes neuronales convolucionales profundas. La manera en que lo lograron fue comenzando a entrenar e implementar CNN utilizando unidades de procesamiento gráficos (GPU) que aceleran significativamente los sistemas complejos basados en redes neuronales. La cantidad de datos de entrenamiento (fotos o videos) también aumentó porque las cámaras de los teléfonos móvil y las cámaras digitales comenzaron a desarrollarse rápidamente y se volvieron asequibles.

Actualmente es comúnmente utilizado para el reconocimiento de las imágenes en la instancia de pesos fijados con el algoritmo de *back propagation network*. Ayuda a resolver los problemas de análisis de datos de alta dimensión en espacio el cual provee una clase de algoritmos y desbloquea la compleja situación gracias a su propiedad simétrica.
El aprendizaje profundo en la clasificación de imágenes basada en redes neuronales inicia la nueva revolución en la inteligencia artificial y es relevante en distintos dominios: en el análisis de la señal de audio y video, reconocimiento facial, reconocimiento de desastres, reconocimiento de voz, visión de computadora y en el procesamiento del lenguaje automatizado.
Es un conjunto de algoritmos donde prepara a un modelo con gran nivel de abstracción en datos por el uso de un grafo profundo con el procesamiento de múltiples capas. Ha ganado gran popularidad en la última década por su habilidad de aprender representación de datos en una manera supervisada o no supervisada y generaliza los conjuntos de datos con el uso de representaciones

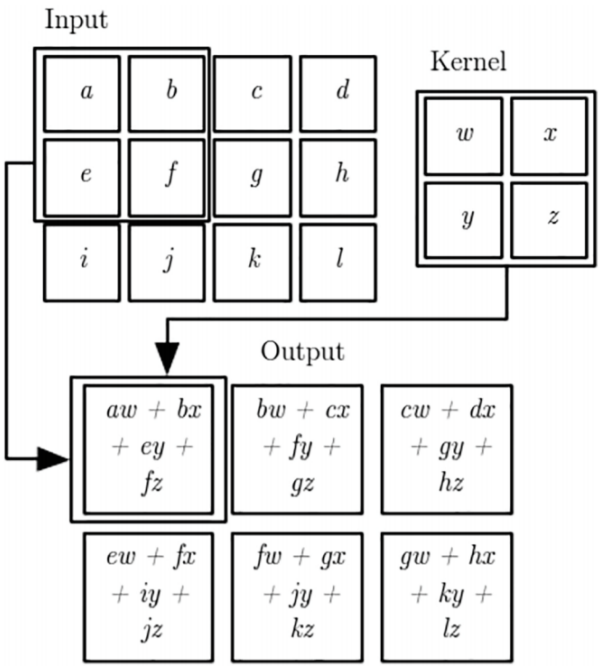
jerárquicas.
La esencial propiedad de sus algoritmos es la extracción automática de rasgos. Esto significa que el algoritmo automáticamente comprende los rasgos relevantes requeridos para la solución del problema. Las CNN tiene tres capas invisibles principales, usualmente varias capas y como son profundas tienen más de un estado de no linealidad en las transformaciones. Cada capa invisible se compone de un conjunto de neuronas responsables de entrenar un único conjunto de rasgos basados en la salida de la capa anterior. Como el número de las capas crece, la complejidad y abstracción de los datos también crece.

## 2.2. Convolution

La convolución es una transformación matemática de dos funciones que genera una tercera función. Su representación matemática es la siguiente:

$$(uv)(x) = \int u(x-t)v(t)dt \tag{2.1}$$

Gráficamente se visualiza en la siguiente imagen:




En inteligencia artificial, las redes convoluciones profundas se aplica a múltiples cascadas de kernels de convolución con aplicaciones de aprendizaje profundo.

### 2.2.1. Convolution Layer

Esta es la primera de las capas, una de las más importantes y tiene gran aplicación en el reconocimiento de imágenes. Tiene tres parámetros fundamentales: *depth*(la profundidad del volumen de salida controla el número de neuronas que conecta con la capa en la misma región del volumen de entrada), *stride*(controla como las columnas profundas alrededor de las dimensiones espaciales(ancho y altura) son localizadas, si el stride es 1 entonces el se mueve los filtros un pixel por tiempo) y *zero-padding*(coloca zeros en los bordes de la imagen de entrada, en ocasiones no se desea variar las dimensiones de la imagen de entrada).

## 2.3. ReLu

**ReLu**: Rectified Linear Units, es una función de activación que se define:

$$f(x) = max(0, x) \tag{2.2}$$

Su objetivo principal es quitar información innecesaria brindada por la imagen luego de una tranformación por convolución (en este caso).

### 2.3.1. ReLu Layer

Es la capa de neuronas que se encarga de funciones de no saturación y no linealidad o de la función loss. Sus resultados en la red neuronal explican rapidez y claridad en la etapa de entrenamiento, puesto que limpia todos aquellos píxeles que no contengan información necesaria para el aprendizaje.

## 2.4. Max Polling

Se encarga de reducir o simplificar la dimensión espacial de la información capturada en la imagen.

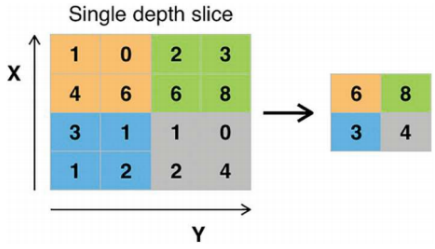



**Fig. 10.** Max pooling process.

### 2.4.1. Polling Layer

La más utilizada es **max polling**, ya que está caracterizada por ser rápida en convergencia.Como se explica en la imagen anterior se toma un tamaño de

kernel y se recorre el volumen de entrada escogiendo cada vez el valor mayor en cada ventana apreciada por el tamaño del kernel.

## 2.5. Fully Connected Layer

La entrada de esta capa es un vector de números. Cada entrada es conectada a cada una de las salidas conocido por el término **Fully Connected**. Contiene acerca del 90 porciento de los parámetros de la red. Esta capa toma básicamente como entrada la salida de la última capa de **Polling** y su salida es un vector **N-dimensional**, donde **N** es el número de clases a clasificar.

## 2.6. Loss Layer

Toma la salida del modelo y el *target* y computa su valor que no es más que la distancia entre ambas entradas. Para su cálculo tiene dos funciones principales:

1. Forward pass: calcular el valor de loss basado en la entrada y el valor del *target*,

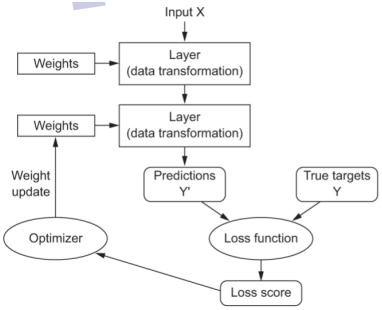
2. Backward pass: calcular el gradiente de la función loss asociado con el criterio y retorna el valor.

A estas funciones se le hacen mención a continuación.

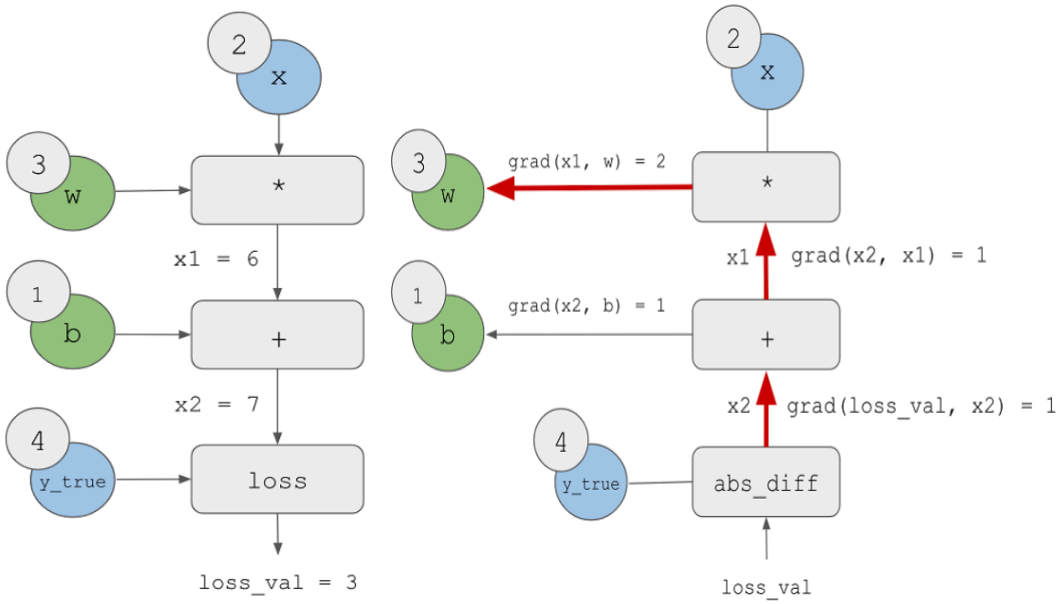
## 2.7. Back propagation Algorithm

El algoritmo de **back propagation** se introdujo originalmente en la década de 1970, pero su importancia no se apreció completamente hasta un famoso artículo de 1986 de David Rumelhart, Geoffrey Hinton y Ronald Williams. La retropropagación funciona mucho más rápido que los enfoques de aprendizaje anteriores, lo que hace posible utilizar redes neuronales para resolver problemas que anteriormente habían sido insolubles.
Se dice que la transformación implementada por una capa es parametrizada por *pesos (weights)*, inicialmente estas variables tienen valores aleatorios y es el algoritmo de *back propagation* el que ajusta estos *pesos*. Una red neuronal profunda puede tener diez millones de parámetros, buscar el valor correcto para todos ellos es una tarea exhaustiva y más que al cambiar un solo valor de ellos todos los demás se deben actualizar, ya que afecta sus comportamientos.
Para controlar la salida de una red neuronal se necesita habilitar una medida que nos diga que tan rápido está aprendiendo para llegar a la salida que se espera, este es el trabajo de la *function loss*, a veces también llamada *función objetiva* o *función de costo*. La función de pérdida toma las predicciones de la red y los verdaderos targets y computa la distancia que existe entre ellos capturando como la red termina en un ejemplo específico.

La función del optimizador es implementar la llamada del algoritmo *Back Propagation*: el algoritmo central del aprendizaje profundo. Como bien se comenta anteriormente los pesos son asignados aleatoriamente donde la red implementa una serie de transformaciones aleatorias, cuando los pesos se ajustan un poco en la dirección correcta, el valor de la función de pérdida decrece, lo que se quiere lograr es tener el menor valor de pérdida posible en el ajuste de los pesos.




Ahora bien, en el ajuste de los pesos interfieren dos sentidos en el *back propagation*: *forward pass* y *backward pass*. Como *loss_val* es la diferencia entre el valor de predicción y el valor de *true target*. Se definen los pasos en la siguiente figura, la primera se refiere a *forward pass* y la segunda a *backward pass*:




Tener en cuenta que las operaciones que se le realiza por capa viene dada por la función activadora **ReLu**, ya que ocurre una multiplicación(operación de convolución) y una suma con un *sesgo*, valor que se aprecia en *b*, hablando específicamente en el *forward pass*. Con respecto a *backward pass* se deben tener en cuenta una serie de consideraciones ya que *grad(a,b)* no es igual a *grad(b,a)*, ya

que estamos hablando de aristas invertidas en el grafo. Explicando el *backward pass* tenemos:

1. *grad(loss_val, x2)=1* ya que como *x2* varía por un monto *epsilon*, *loss_val=abs(4-x2)* varía en el mismo monto.
2. *grad(x2,x1)=1* como *x1* varía por un monto epsilon igual, *x2=x1+b=x1+1* varía en el mismo monto.
3. *grad(x2,b)=1* como *b* varía por un monto epsilon, *x2=x1+b=6+b* varía en el mismo monto.
4. *grad(x1,w)=2* como *w* varía por un monto epsilon, *x1=x*w=2*w* varía por *2 * epsilon*.

## 2.8. Proceso de entrenamiento

Un sistema debe aprender primero sus características. Debe entrenarse para predecir si un objeto es **X**, **Y**, .... Los modelos de aprendizaje profundo aprenden estas características de una manera diferente a los modelos de aprendizaje tradicional. Es por eso que los enfoques de entrenamiento de modelos también son diferentes.
Para construir un modelo ML que pueda, los científicos de datos deben especificar qué características de entrada *(Propiedades del problema)* considerara el modelo al predecir un resultado. Eso puede ser el tamaño del virus, su fisonomía, el contorno, sus interacciones. El proceso de construcción de funciones utilizando el conocimiento del dominio se denomina *ingeniería de funciones*.
Para entrenar un modelo se deben reunir las características de miles de millones de datos de los virus que se quieren clasificar. No se pueden construir funciones precisas que funcionen para cada imagen posible teniendo en cuenta complicaciones tales como la variabilidad del objeto dependiente del punto de vista, el desorden del fondo, las condiciones de iluminación o la deformación de la imagen. Debería haber otro enfoque, y existe gracias a la naturaleza de las redes neuronales.
Las redes neuronales aprenden funciones directamente de los datos con los que se entrenan, por lo que los especialistas no necesitan extraer funciones manualmente.
*El poder de las redes neuronales proviene de su capacidad para aprender la representación en sus datos de entrenamiento y como relacionarlos mejor con la variable de salida que desea predecir. En este sentido, las redes neuronales aprenden a mapear. Matemáticamente, son capaces de aprender cualquier función de mapeo y se ha demostrado que son algoritmos de aproximación universales*, señala Jason Brownlee en Crash Course on Multi-layer Perceptron Neural Networks.
Los datos de entrenamiento, en este caso, son un gran conjunto de datos que contiene muchos ejemplos de cada clase de imagen. Ej.: el conjunto de imágenes de **ImageNet** contiene más de 14 millones de imágenes anotadas por humanos

que representan 21841 conceptos (cjtos de sinónimos o conjuntos de sincronización según la jerarquía de **WordNet**), con un promedio de 1000 imágenes por concepto.
Cada imagen se encuentra anotada (etiquetada) con una categoría a la que pertenece: conjunto de virus. El algoritmo explora estos ejemplos, aprende sobre las características visuales de cada categoría y finalmente aprende a reconocer cada clase de imagen. El estilo de entrenamiento modelo se llama: *APRENDIZAJE SUPERVISADO*.
Cada capa de nodos se entrena en la salida (conjunto de funciones) producida por la capa anterior. Por lo tanto, los nodos en cada capa sucesiva pueden reconocer características más complejas y detalladas: representaciones visuales de lo que representa la imagen. Tal jerarquía de complejidad y abstracción crecientes se conoce como: *JERARQUÍA DE CARACTERÍSTICAS*.
Por lo tanto, cuantas más capas tenga la red, mayor será su capacidad predictiva.
La arquitectura líder utilizada para el reconocimiento de imágenes y las tareas de detección son las redes neuronales convolucionales (CNN). Las CNN consisten en varias capas con pequeñas colecciones de neuronas, cada una de las cuales percibe pequeñas partes de una imagen. Los resultados de todas las colecciones en una capa se superponen parcialmente para crear la representación de la imagen, lo que permite que el sistema aprenda sobre la composición de la imagen.
Usualmente alrededor de 100 imágenes son suficientes para entrenar una clase. Si las imágenes en una clase son bastantes similares, menor cantidad de imágenes deben ser suficientes. El entrenamiento de imágenes es la representación de la variación del comportamiento típica en cada clase. A modo de conclusión, a mayor complejidad de rasgos mayor cantidad de imágenes a entrenar en la clase.

# Capítulo 3

# Detalles de la implementación

El objetivo es implementar una red neuronal desde cero para la clasificación de imágenes. Se entrena un modelo con imágenes de las clases a clasificar, y luego dada una imagen de entrada el modelo reconoce a cuál clase pertenece. Para entrenar se tienen 323 imágenes, las cuales contienen imágenes de *Vibrio Falciparum*, *SarsCov2* y *Cholerae*. De imágenes de validación se tiene un total de 642 imágenes. Es decir, este proyecto abarca 965 imágenes. Se tiene como objetivo futuro mejorar la precisión de entrenamiento ya que actualmente se tiene una puntución del 96 porciento aproximadamente, lo que hace que la función de pérdida sea pequeña, pero no estamos satisfechos aún con los resultados. Aunque en la parte de la predicción ha tenido satisfactorios resultados puesto que en todos los casos ha presentado total exactitud.

## 3.1. src

### 3.1.1. train.py

#### Limpiar sesión

Lo primero que se realiza para el entrenamiento es *limpiar procesos de keras*, por qué se dice esto: para realizar el entrenamiento se consulta con la api de **Keras** la cual se encuentra dentro de **TensorFlow**, se quiere que este proceso se encuentre los más limpio posible, por lo que se procede a eliminar cualquier proceso de *Keras* en el **background** de nuestro sistema.

#### Definición de variables:

1. **epocs**: número de veces de iteración del data set durante el entrenamiento;

2. **heigth, length**: tamaño en la que se van a procesar las imágenes antes del entrenamiento para que el mismo sea estándar en todas;

3. **batch_size**: número de imágenes que se procesan en cada uno de los pasos;

4. **steps**: número de veces que se va a procesar la información en cada una de las épocas (epocs), cada época va a tener *steps* pasos;

5. **steps_validation**: al final de cada una de las épocas se van a correr *steps_validation* pasos con el dataset de validación para comprender qué tan bien está aprendiendo el algoritmo;

6. **filters_conv1, filters_conv2**: número de filtros que se va a utilizar en cada convolución, lo que signifia que luego que se realice una convolución la imagen va a tener una profundidad de *filters_convi, i =1,2*;

7. **length_filter1**: tamaño de filtro que se va a utilizar en nuestra convolución.

8. **length_pool**: tamaño de filtro de *maxpolling*;

9. **class_**: cantidad de clases a clasificar, en este caso son 3 clases.

**Preprocesamiento de imágenes**

Se toman los datos de entrenamiento y validación, y luego se preprocesan las imágenes para introducirlas en la red neuronal. Para ello, se crea un generador el cual informa cómo se va a proprocesar la información y luego se procede a la transformación de las imágenes.
En el generador se reescalan las imágenes (*rescale*), en este caso se convierten del rango de 0-255 pixeles a 0-1 pixeles, o sea, se convierten a imágenes binarias. El parámetro *shear_range* en este caso ayuda a inclinar las imágenes, lo que ayuda en el proceso de entrenamiento ya que una clase, ejemplo: *Cólera* puede aparecer en una posición aleatoria. *zoom_range* en este caso va a realizarle zoom a una parte de las imágenes de manera aleatoria, igual ayuda al entrenamiento a la hora de clasificar un fragmento de una clase, no necesariamente debe aparecer todo el cuerpo de la bacteria o virus a clasificar. *horizontal_flip* en este caso está en *true*, lo que ayuda a identificar direccionalidad puesto que la función de la misma es invertir la imagen.
Se tienen dos generadores, uno para las imágenes de entrenamiento y otro para las imágenes de validación, para estas útimas solo se realiza el reescalamiento. No se le realiza otro cambio a la validación puesto que este no es el objetivo, ya que este es nuestro conjunto modelo. En las imágenes de entrenamiento estas operaciones sí son necesarias puesto que se quiere que nuestro modelo aprenda en la mayor cantidad de escenarios posibles.
Luego de definidos los generadores se leen las imágenes de entrenamiento y validación respectivamente.

En este instante ya se tienen las imágenes preprocesadas y se procede a definir la red convolucional.

**Red neuronal**

1. **Definición**: la red que se genera es secuencial, la cual va a constar de 6 capas convolucionales.
2. **Layer Flatten**: las imágenes que llegado a este punto de la red son muy profundas y pequeñas se van a aplanar, o sea, se va a tener una sola dimensión que va a contener toda la información de la red neuronal.
3. **Layer Dense**: luego de aplanar la información todas neuronas van a estar conectadas con las capas pasadas.
4. **Layer Dropout**: a esta capa densa se va a deshabilitar (en este caso) durante el entrenamiento el 50 porciento de las neuronas cada paso, esto se realiza para evitar el sobreajuste, ya que si todas las neuronas están activadas puede que la red neuronal aprenda un camino en específico para clasificar una clase, y de esta forma la red aprenda caminos alternos cada vez. El objetivo principal de esto es hacer un modelo que se adapte mejor a información nueva.
5. **Layer Softmax**: esta capa es la que se encarga de la clasificación, solo tiene *class_* neuronas. Lo que nos informa es la proporción de probabilidad que pertenezca a cada clase, donde el resultado que se va a mostrar en la predicción es aquella clase que mayor porcentaje tenga en la probabilidad.
6. **Proceso de compilación**: este paso requiere de tres aspectos fundamentales:
   *a*) optimizador: el mecanismo que se basa el modelo para actualizarse el mismo basado en el proceso del entrenamiento.
   *b*) función de pérdida (loss): como el modelo habilita las mediciones en el entrenamiento de datos, es el que guía al entrenamiento en la dirección correcta.
   *c*) métrica: solo nos importa en este contexto 'accuracy' (la fracción de las imágenes que son correctamente clasificadas) .
7. **Entrenamiento**: como parámetros tenemos a las imágenes de entrenamiento, imágenes de validación, la cantidad de épocas, los pasos y los pasos de validación. Lo cual significa: en cada una de las épocas va a correr *steps* pasos, cuando termina la época va a correr *steps_validation* pasos de validación.
8. **Guardar modelo**: el modelo entrenado se guarda en una carpeta lo cual evita tener que entrenar el modelo cada vez que se quiera predecir. Esto se encuentra en la carpeta **model**, si no existe hay que entrenar el modelo para generar la carpeta.

### 3.1.2. predict.py

Se procede a leer el modelo entrenado de la carpeta **model**. Se declara una función **predict(img)**, el cual tiene como entrada la imagen que se quiere clasificar. Se convierte la imagen a un array de pixeles y se expande las dimensiones incluyendo una nueva dimensión a la misma para poder procesar la información sin problemas.
Se procede a predecir la imagen de nuestro modelo, este método nos devuelve un arreglo de dos dimensiones con un solo elemento que es un arreglo de *class* posiciones binarias, la posición que contenga el elemento 1 es aquella clase a la cual pertenece la imagen de entrada.