Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Inteligencia Artificial

Producto Integrador de Aprendizaje: Entrenamiento de una Red Neuronal Convolucional

André Haziel Sánchez González	1841825
Rodolfo Martínez Saucedo	1843060
Juan Pablo Ambriz Amador	1864353
Luis Daniel Islas Valencia	1865181
Milly Pamela Garza Santoy	1877296

Carrera: **IB**

Ing. Daniel Isaías López Páez

21 de Noviembre de 2021

Introducción

La Red Convolucional o CNN es un tipo de Red Neuronal Artificial con aprendizaje supervisado que procesa sus capas imitando al córtex visual del ojo humano para identificar distintas características en las entradas que en definitiva hacen que pueda identificar objetos y "ver". Es un algoritmo de deep learning que está diseñado para trabajar con sus objetos asignando diferentes niveles de importancia para poder diferenciar entre estos. Para ello, la CNN contiene varias capas ocultas especializadas y con una jerarquía: esto quiere decir que las primeras capas pueden detectar líneas, curvas y se van especializando hasta llegar a capas más profundas que reconocen formas complejas como un rostro o la silueta de un animal.

Explicaremos cómo funciona el Deep Learning mediante un ejemplo hipotético de predicción sobre quién ganará el próximo mundial de fútbol. Utilizaremos aprendizaje supervisado mediante algoritmos de Redes Neuronales Artificiales.

Para lograr las predicciones de los partidos de fútbol usaremos como ejemplo las siguientes entradas:

- Cantidad de Partidos Ganados
- Cantidad de Partidos Empatados
- Cantidad de Partidos Perdidos
- Cantidad de Goles a Favor
- Cantidad de Goles en Contra
- "Racha Ganadora" del equipo (cant. máx de partidos ganados seguidos sobre el total jugado)

Y podríamos tener muchísimas entradas más, por ejemplo la puntuación media de los jugadores del equipo, o el score que da la FIFA al equipo. Como en cada partido tenemos a 2 rivales, deberemos estos 6 datos de entrada por cada equipo, es decir, 6 entradas del equipo 1 y otras 6 del equipo 2 dando un total de 12 entradas.

La predicción de salida será el resultado del partido: Local, Empate o Visitante.

Metodología y Experimentación

En la programación "tradicional" escribiríamos código en donde indicamos reglas por ejemplo "si goles de equipo 1 mayor a goles de equipo 2 entonces probabilidad de Local aumenta". Es decir que deberíamos programar artesanalmente unas reglas de inteligencia bastante extensas e interrelacionar las 12 variables. Para evitar todo ese enredo y hacer que nuestro código sea escalable y flexible a cambios recurrimos a las Redes Neuronales de Machine Learning para indicar una arquitectura de interconexiones y dejar que este modelo aprenda por sí mismo (y descubra él mismo relaciones de variables que nosotros desconocemos).

Las capas que desarrollamos son:

- **La capa de entrada** recibe los datos de entrada y los pasa a la primera capa oculta.
- **Las capas ocultas** realizan cálculos matemáticos con nuestras entradas. Uno de los desafíos al crear la Red Neuronal es decidir el número de capas ocultas y la cantidad de neuronas de cada capa.
- **La capa de Salida** devuelve la predicción realizada. En nuestro caso de 3 resultados discretos las salidas podrán ser “1 0 0” para Local, “0 1 0” para Empate y “0 0 1” para Visitante

La cantidad total de capas en la cadena le da “profundidad” al modelo. De aquí es que surge la terminología de *Aprendizaje Profundo*.

Cada conexión de nuestra red neuronal está asociada a un peso. Este peso determina la importancia que tendrá esa relación en la neurona al multiplicarse por el valor de entrada. Los valores iniciales de peso se asignan aleatoriamente (SPOILER: más adelante los pesos se ajustarán solos). Imitando a las neuronas biológicas. Cada neurona tiene una función de activación. Esta función determinará si la suma de sus valores recibidos (previamente multiplicados por el peso de la conexión) supera un umbral que hace que la neurona se active y dispare un valor hacia la siguiente capa conectada. Hay diversas Funciones de Activación conocidas que se suelen utilizar en estas redes.

Cuando todas las capas finalizan de realizar sus cómputos, se llegará a la capa final con una predicción. Por Ejemplo si nuestro modelo nos devuelve 0.6 0.25 0.15 está prediciendo que ganará Local con 60% probabilidades, será Empate 25% o que gane Visitante 15%.

Entrenar nuestra IA puede llegar a ser la parte más difícil del Deep Learning. Necesitamos:

I. Gran cantidad de valores en nuestro conjunto de Datos de Entrada
II. Gran poder de cálculo computacional

Para nuestro ejemplo de “predicción de Partidos de Fútbol para el próximo Mundial” deberemos crear una base de datos con todos los resultados históricos de los Equipos de Fútbol en mundiales, en partidos amistosos, en clasificatorios, los goles, las rachas a lo largo de los años, etc. Para entrenar nuestra máquina, debemos alimentarla con nuestro conjunto de datos de entrada y comparar el resultado (local, empate, visitante) contra la predicción obtenida. Como nuestro modelo fue inicializado con pesos aleatorios, y aún está sin entrenar, las salidas obtenidas seguramente serán erróneas.

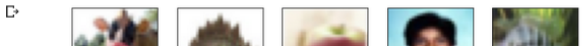
Una vez que tenemos nuestro conjunto de datos, comenzaremos un proceso iterativo: usaremos una función para comparar cuán bueno/malo fue nuestro resultado contra el resultado real. Esta función es llamada “Función Coste“. Idealmente queremos que nuestro coste sea cero, es decir sin error(cuando el valor de la predicción es igual al resultado real del partido). A medida que se entrena el modelo irá ajustando los pesos de interconexiones de las neuronas de manera automática hasta obtener buenas predicciones.

En nuestro Producto Integrador, para cada modelo cambiamos un hiper parámetro, escogimos dentro de la selección de Valores o hiperparametros a modificar: cantidad de capas, cantidad de neuronas, funciones de activación, tamaño de filtro en capas de convolución, optimizador, función de pérdida, tamaño de batch de entrenamiento, cantidad de epochs, etc

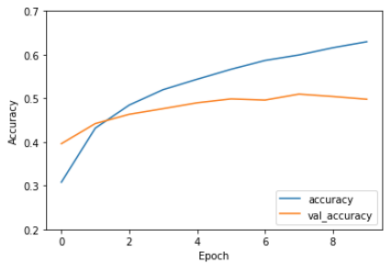
Resultados de los 3 Modelos

En el primer modelo que se realizó en Colaboratory, se utilizaron tres capas con una resolución de 32. Para intentar tener una mayor accuracy se optó por emplear 10 épocas, y se consiguió un 0.48.

```
#Capas
model = models.Sequential()
model.add(layers.Conv2D(128, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.MaxPooling2D((2, 2)))
model.summary()

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(20))
model.summary()

#Compilar y evaluar modelo
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))


plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.2, 0.7])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

#Exactitud
print("Test Acc ", test_acc)
```



```
1563/1563 [==============================] - 25s 16ms/step - loss: 1.8343 - accuracy: 0.4317 - val_loss: 1.8082 - val_accuracy: 0.4422
Epoch 3/10
1563/1563 [==============================] - 26s 16ms/step - loss: 1.6634 - accuracy: 0.4846 - val_loss: 1.7396 - val_accuracy: 0.4634
Epoch 4/10
1563/1563 [==============================] - 25s 16ms/step - loss: 1.5478 - accuracy: 0.5199 - val_loss: 1.6958 - val_accuracy: 0.4764
Epoch 5/10
1563/1563 [==============================] - 25s 16ms/step - loss: 1.4603 - accuracy: 0.5437 - val_loss: 1.6542 - val_accuracy: 0.4897
Epoch 6/10
1563/1563 [==============================] - 25s 16ms/step - loss: 1.3832 - accuracy: 0.5665 - val_loss: 1.6322 - val_accuracy: 0.4987
Epoch 7/10
1563/1563 [==============================] - 25s 16ms/step - loss: 1.3192 - accuracy: 0.5867 - val_loss: 1.6446 - val_accuracy: 0.4960
Epoch 8/10
1563/1563 [==============================] - 26s 16ms/step - loss: 1.2662 - accuracy: 0.5993 - val_loss: 1.6336 - val_accuracy: 0.5096
Epoch 9/10
1563/1563 [==============================] - 26s 16ms/step - loss: 1.2133 - accuracy: 0.6158 - val_loss: 1.6478 - val_accuracy: 0.5042
Epoch 10/10
1563/1563 [==============================] - 26s 16ms/step - loss: 1.1656 - accuracy: 0.6295 - val_loss: 1.7152 - val_accuracy: 0.4979
313/313 - 2s - loss: 1.7152 - accuracy: 0.4979 - 2s/epoch - 5ms/step
Test Acc  0.49790000915527344
```

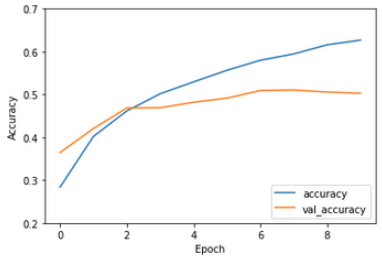


El segundo modelo fue nuestro mejor intento, con una accuracy de 0.502. Los hiperparametros que se decidieron usar fueron 10 epocas, con una resolución de 128 y una densidad de 64. Como se pudo apreciar, por lo menos con nuestros tres modelos, es que a mayor densidad y resolución, la accuracy aumentará.

```python
#Plataforma TensorFlow
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt

#Cifar 100
(train_images, train_labels), (test_images, test_labels) = datasets.cifar100.load_data(label_mode='coarse')


train_images, test_images = train_images / 255.0, test_images / 255.0

class_names =['beaver','dolphin', 'otter', 'seal', 'whale','aquarium fish',
              'flatfish', 'ray', 'shark', 'trout','orchids', 'poppies', 'roses',
              'sunflowers', 'tulips','bottles', 'bowls', 'cans', 'cups', 'plates',
              'apples', 'mushrooms', 'oranges', 'pears', 'sweet peppers''clock',
              'computer keyboard', 'lamp', 'telephone', 'television','bed',
              'chair', 'couch', 'table', 'wardrobe','bee', 'beetle', 'butterfly',
              'caterpillar', 'cockroach', 'bear', 'leopard', 'lion', 'tiger', 'wolf',
              'bridge', 'castle', 'house', 'road', 'skyscraper','cloud', 'forest',
              'mountain', 'plain', 'sea', 'camel', 'cattle', 'chimpanzee', 'elephant',
              'kangaroo', 'fox', 'porcupine', 'possum', 'raccoon', 'skunk', 'crab',
              'lobster', 'snail', 'spider', 'worm', 'baby', 'boy', 'girl', 'man',
              'woman',  'crocodile', 'dinosaur', 'lizard', 'snake', 'turtle','hamster',
              'mouse', 'rabbit', 'shrew', 'squirrel','maple', 'oak', 'palm', 'pine',
              'willow', 'bicycle', 'bus', 'motorcycle', 'pickup truck', 'train',
              'lawn-mower', 'rocket', 'streetcar', 'tank', 'tractor']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])

    plt.xlabel(super_names[train_labels[i][0]])
plt.show()
```

```python
#Capas
model = models.Sequential()
model.add(layers.Conv2D(128, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.MaxPooling2D((2, 2)))
model.summary()

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(20))
model.summary()

#Compilar y evaluar modelo
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))


plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.2, 0.7])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

#Exactitud
print("Test Acc ", test_acc)
```

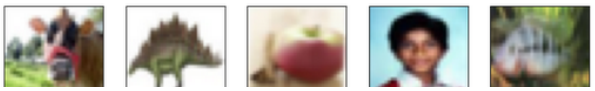
```
Epoch 1/10
1563/1563 [==============================] - 26s 16ms/step - loss: 2.3352 - accuracy: 0.2835 - val_loss: 2.0643 - val_accuracy: 0.3644
Epoch 2/10
1563/1563 [==============================] - 25s 16ms/step - loss: 1.9418 - accuracy: 0.4020 - val_loss: 1.8980 - val_accuracy: 0.4203
Epoch 3/10
1563/1563 [==============================] - 26s 17ms/step - loss: 1.7424 - accuracy: 0.4614 - val_loss: 1.7159 - val_accuracy: 0.4684
Epoch 4/10
1563/1563 [==============================] - 26s 16ms/step - loss: 1.6083 - accuracy: 0.5015 - val_loss: 1.7342 - val_accuracy: 0.4689
Epoch 5/10
1563/1563 [==============================] - 25s 16ms/step - loss: 1.5060 - accuracy: 0.5293 - val_loss: 1.6898 - val_accuracy: 0.4818
Epoch 6/10
1563/1563 [==============================] - 28s 18ms/step - loss: 1.4187 - accuracy: 0.5564 - val_loss: 1.6590 - val_accuracy: 0.4912
Epoch 7/10
1563/1563 [==============================] - 27s 17ms/step - loss: 1.3414 - accuracy: 0.5797 - val_loss: 1.6078 - val_accuracy: 0.5091
Epoch 8/10
1563/1563 [==============================] - 25s 16ms/step - loss: 1.2803 - accuracy: 0.5945 - val_loss: 1.6081 - val_accuracy: 0.5103
Epoch 9/10
1563/1563 [==============================] - 25s 16ms/step - loss: 1.2183 - accuracy: 0.6158 - val_loss: 1.6435 - val_accuracy: 0.5056
Epoch 10/10
1563/1563 [==============================] - 25s 16ms/step - loss: 1.1689 - accuracy: 0.6267 - val_loss: 1.6908 - val_accuracy: 0.5027
313/313 - 2s - loss: 1.6908 - accuracy: 0.5027 - 2s/epoch - 6ms/step
Test Acc  0.5026999711990356
```



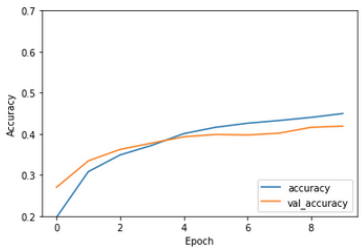
Por último, en el tercer modelo tuvimos un problema, y nuestra accuracy se redujo a 0.41. Aumentamos la resolución a 64 en las capas, pero reducimos la densidad a 32, pero seguimos usando 10 épocas. Es probable que esto último fue lo que nos perjudica, ya que no se hicieron más pruebas. Esto fue debido a la falta de tiempo, ya que al discutir en equipo decidimos utilizar solamente 10 épocas en todos los modelos para tener un hiper parámetro estándar y checar el cambio de la accuracy a partir de este.

```python
#Plataforma TensorFlow
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt

# Cifar 100
(train_images, train_labels), (test_images, test_labels) = datasets.cifar100.load_data(label_mode='coarse')
train_images, test_images = train_images / 255.0, test_images / 255.0

class_names =['beaver','dolphin', 'otter', 'seal', 'whale','aquarium fish',
              'flatfish', 'ray', 'shark', 'trout','orchids', 'poppies', 'roses',
              'sunflowers', 'tulips','bottles', 'bowls', 'cans', 'cups', 'plates',
              'apples', 'mushrooms', 'oranges', 'pears', 'sweet peppers''clock',
              'computer keyboard', 'lamp', 'telephone', 'television','bed',
              'chair', 'couch', 'table', 'wardrobe','bee', 'beetle', 'butterfly',
              'caterpillar', 'cockroach', 'bear', 'leopard', 'lion', 'tiger', 'wolf',
              'bridge', 'castle', 'house', 'road', 'skyscraper','cloud', 'forest',
              'mountain', 'plain', 'sea', 'camel', 'cattle', 'chimpanzee', 'elephant',
              'kangaroo', 'fox', 'porcupine', 'possum', 'raccoon', 'skunk', 'crab',
              'lobster', 'snail', 'spider', 'worm', 'baby', 'boy', 'girl', 'man',
              'woman',  'crocodile', 'dinosaur', 'lizard', 'snake', 'turtle','hamster',
              'mouse', 'rabbit', 'shrew', 'squirrel','maple', 'oak', 'palm', 'pine',
              'willow', 'bicycle', 'bus', 'motorcycle', 'pickup truck', 'train',
              'lawn-mower', 'rocket', 'streetcar', 'tank', 'tractor']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])

    plt.xlabel(super_names[train_labels[i][0]])
plt.show()
```

```python
#Capas de Entrenamiento
model = models.Sequential()
model.add(layers.Conv2D(64, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.MaxPooling2D((2, 2)))
model.summary()

model.add(layers.Flatten())
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(20))
model.summary()

#Compilar y Evaluacion del modelo
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))


plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.2, 0.7])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

#Exactitud
print("Test Acc ", test_acc)
```



```
dense_9 (Dense)              (None, 20)                660
=================================================================
Total params: 139,764
Trainable params: 139,764
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
1563/1563 [==============================] - 18s 11ms/step - loss: 2.5705 - accuracy: 0.1968 - val_loss: 2.3269 - val_accuracy: 0.2699
Epoch 2/10
1563/1563 [==============================] - 18s 12ms/step - loss: 2.2131 - accuracy: 0.3080 - val_loss: 2.1416 - val_accuracy: 0.3342
Epoch 3/10
1563/1563 [==============================] - 18s 11ms/step - loss: 2.0676 - accuracy: 0.3487 - val_loss: 2.0455 - val_accuracy: 0.3622
Epoch 4/10
1563/1563 [==============================] - 18s 11ms/step - loss: 1.9916 - accuracy: 0.3721 - val_loss: 1.9946 - val_accuracy: 0.3776
Epoch 5/10
1563/1563 [==============================] - 18s 11ms/step - loss: 1.9138 - accuracy: 0.4006 - val_loss: 1.9413 - val_accuracy: 0.3928
Epoch 6/10
1563/1563 [==============================] - 18s 11ms/step - loss: 1.8568 - accuracy: 0.4159 - val_loss: 1.9373 - val_accuracy: 0.3985
Epoch 7/10
1563/1563 [==============================] - 19s 12ms/step - loss: 1.8158 - accuracy: 0.4259 - val_loss: 1.9215 - val_accuracy: 0.3973
Epoch 8/10
1563/1563 [==============================] - 18s 11ms/step - loss: 1.7886 - accuracy: 0.4325 - val_loss: 1.8915 - val_accuracy: 0.4017
Epoch 9/10
1563/1563 [==============================] - 17s 11ms/step - loss: 1.7604 - accuracy: 0.4401 - val_loss: 1.8638 - val_accuracy: 0.4159
Epoch 10/10
1563/1563 [==============================] - 17s 11ms/step - loss: 1.7365 - accuracy: 0.4495 - val_loss: 1.8681 - val_accuracy: 0.4188
313/313 - 1s - loss: 1.8681 - accuracy: 0.4188 - 1s/epoch - 4ms/step
Test Acc  0.4187999963760376
```



Discusión

Aprovechamos la plataforma de Tensor Flow ya que nos ofrece varios niveles de abstracción, crear y entrenar modelos mediante la API que ayudó a que los primeros pasos y el aprendizaje automático fueran sencillos. Creamos nuestra base convolucional siguiendo el ejemplo de TF, aprovechando la base de datos CIFAR 100 y empezamos a definir nuestras capas.

En un inicio estábamos teniendo problemas con los arreglos 2D así como contando con diferentes tipos de resoluciones, esto nos generaba un porcentaje de accuracy demasiado bajo de acuerdo a lo solicitado. Gracias a este intento pudimos efectuar los cambios necesarios para arreglar el código y los valores. El proceso era simple y gracias a esto pudimos notar los errores y tener un porcentaje mayor al acercamiento.

```
#Plataforma TensorFlow
import tensorflow as tf

from tensorflow.keras import datasets, layers, models
from keras.layers import Conv2D, Dense, MaxPooling2D, Flatten
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix

(train_images, train_labels), (test_images, test_labels) = datasets.cifar100.load_data()

#Clases CFAR100
class_names =['beaver','dolphin', 'otter', 'seal', 'whale','aquarium fish',
              'flatfish', 'ray', 'shark', 'trout','orchids', 'poppies', 'roses',
              'sunflowers', 'tulips','bottles', 'bowls', 'cans', 'cups', 'plates',
              'apples', 'mushrooms', 'oranges', 'pears', 'sweet peppers''clock',
              'computer keyboard', 'lamp', 'telephone', 'television','bed',
              'chair', 'couch', 'table', 'wardrobe','bee', 'beetle', 'butterfly',
              'caterpillar', 'cockroach', 'bear', 'leopard', 'lion', 'tiger', 'wolf',
              'bridge', 'castle', 'house', 'road', 'skyscraper','cloud', 'forest',
              'mountain', 'plain', 'sea', 'camel', 'cattle', 'chimpanzee', 'elephant',
              'kangaroo', 'fox', 'porcupine', 'possum', 'raccoon', 'skunk', 'crab',
              'lobster', 'snail', 'spider', 'worm', 'baby', 'boy', 'girl', 'man',
              'woman',  'crocodile', 'dinosaur', 'lizard', 'snake', 'turtle','hamster',
              'mouse', 'rabbit', 'shrew', 'squirrel','maple', 'oak', 'palm', 'pine',
              'willow', 'bicycle', 'bus', 'motorcycle', 'pickup truck', 'train',
              'lawn-mower', 'rocket', 'streetcar', 'tank', 'tractor']

# Arreglos 2D 1D
```

```
flatten_5 (Flatten)          (None, 4096)              0

dense_10 (Dense)             (None, 256)               1048832

dense_11 (Dense)             (None, 128)               32896

dense_12 (Dense)             (None, 100)               12900

=================================================================
Total params: 1,136,196
Trainable params: 1,136,196
Non-trainable params: 0

_________________________________________________________________
Epoch 1/20
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-6-3c3c50b15b6f> in <module>()
     69 history = model.fit(train_images, train_labels, epochs=20,
     70                     verbose=1,
---> 71                     validation_data=(test_images, test_labels))
     72
     73 plt.plot(history.history['accuracy'], label='accuracy')

                                  1 frames
/usr/local/lib/python3.7/dist-packages/tensorflow/python/framework/func_graph.py in autograph_handler(*args, **kwargs)
   1127           except Exception as e:  # pylint:disable=broad-except
   1128             if hasattr(e, "ag_error_metadata"):
-> 1129               raise e.ag_error_metadata.to_exception(e)
   1130             else:
   1131               raise

ValueError: in user code:

    File "/usr/local/lib/python3.7/dist-packages/keras/engine/training.py", line 878, in train_function  *
        return step_function(self, iterator)
    File "/usr/local/lib/python3.7/dist-packages/keras/engine/training.py", line 867, in step_function  **
        outputs = model.distribute_strategy.run(run_step, args=(data,))
    File "/usr/local/lib/python3.7/dist-packages/keras/engine/training.py", line 860, in run_step  **
        outputs = model.train_step(data)
    File "/usr/local/lib/python3.7/dist-packages/keras/engine/training.py", line 808, in train_step
        y_pred = self(x, training=True)
    File "/usr/local/lib/python3.7/dist-packages/keras/utils/traceback_utils.py", line 67, in error_handler
        raise e.with_traceback(filtered_tb) from None
    File "/usr/local/lib/python3.7/dist-packages/keras/engine/input_spec.py", line 263, in assert_input_compatibility
        raise ValueError(f'Input {input_index} of layer "{layer_name}" is '

    ValueError: Input 0 of layer "sequential_5" is incompatible with the layer: expected shape=(None, 64, 32, 3), found shape=(None, 32, 32, 3)
```

Conclusión

Juan Pablo Ambriz Amador:

Las redes neuronales convolucionales nos permiten analizar rápida y eficientemente nuestro conjunto de elementos objetos o imágenes seleccionados, en esta actividad implementamos el uso de esta variación de aprendizaje profundo entrenando 3 modelos aprovechando la plataforma de Tensor Flow y los hiperparametros controlados por nosotros, obteniendo diferente nivel de exactitud y precisión para el resultado de la evaluación. Poder desarrollar redes neuronales nos brinda grandes posibilidades y oportunidades ya que se puede aprovechar en dispositivos administrados automáticamente; como lo son las herramientas eléctricas, dispositivos de reconocimiento y diagnóstico médicos, controles de motores automáticos y muchos más sistemas integrados. Me gusto utilizar la base de datos CIFAR 100 ya que nos permitió realizar nuestro modelo de una manera más rápida y sencilla, así como fue divertido ver los resultados gráficos con imágenes del conjunto de datos.

Milly Pamela Garza Santoy:

En esta actividad pude conocer más al respecto de cómo funcionan las redes Neuronales, pude desarrollar nuestro propio modelo de reconocimiento de imágenes y elementos basándonos en la integración de Tensor Flow y Python. Tuvimos dificultades con la programación y evaluación de nuestros modelos de Red Convolucional ya que no logramos obtener una precisión alta con la primera propuesta de nuestro código y estuvimos realizando varios cambios para que funcionara correctamente. Es notable el gran uso y versatilidad que se le puede dar a este tipo de diseños y modelos para el desarrollo de sistemas automatizados, continuaré estudiando y revisando las maneras para poder interactuar

André Haziel Sánchez González:

En esta actividad se aprendieron muchas cosas, ya que nos permitió ir practicando con las redes neuronales, especialmente usando la aplicación de Tensor Flow. Se nos complicó un poco esta tarea, pero al momento de ir investigando y efectuando nuestros modelos se nos fue facilitando. En conclusión el aprender y conocer más de estas tecnologías nos permite poder emplearlas de manera eficiente y poder aprovechar las ventajas de cada una en nuestros proyectos de diseño electrónico y dispositivos, brindándonos herramientas y ayudas para nuestro desarrollo en la ingeniería biomédica.

Luis Daniel Islas Valencia:

Las redes neuronales artificiales son una tecnología computacional que puede utilizarse en distintas aplicaciones y en esta actividad pudimos trabajar con una para desarrollarla y complementar los conceptos que vimos en clase. Estas pueden desarrollarse en periodos de tiempo razonables y pueden realizar tareas

concretas mejor que otras tecnologías convencionales. Las redes presentan por lo general fallos del sistema, además proporcionan alto grado de paralelismo en el procesamiento de datos. Se trabajó con herramientas de machine y deep learning, las cuales se abordan en el curso que hicimos como tarea además de conceptos abordados en clase, éstas que facilitan el procesamiento de datos y la toma de decisiones en 3 etapas de acuerdo a entradas. Se utilizó la base de datos CIFAR 100 para realizar nuestro modelo de mejor manera, además pudimos ver resultados gráficos del nivel de accuracy obtenido.

Rodolfo Martínez Saucedo:

Gracias a esta actividad fue posible adentrarnos un poco más a las posibilidades que ofrece hoy en día la inteligencia artificial así como al funcionamiento de las redes neuronales. Se trabajó con herramientas de machine y deep learning, las cuales se abordan en el curso que hicimos como tarea además de hacer uso de elementos y modelos basados en la programación/Python. Se batalló al momento de realizar el código ya que no se entendía del todo lo que eran los modelos de Red Convolucional, que una vez hecha y trabajada la teoría, además del haber recurrido a clases pasadas, se pudo sacar adelante.

Referencias bibliográficas

I. Aprende Machine Learning (14 de noviembre de 2017). Aprendizaje Profundo: una Guía Rápida. Recuperado el 18 de noviembre de 2021, de: https://www.aprendemachinelearning.com/aprendizaje-profundo-una-guia-rapida/

II. Bootcamp AI (23 de noviembre de 2019). Intro a las redes neuronales convolucionales. Recuperado el 18 de noviembre de 2021, de: https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8

III. MathWorks (n.d.). Redes neuronales convolucionales. Recuperado el 18 de noviembre de 2021, de: https://la.mathworks.com/discovery/convolutional-neural-network-matlab.html

IV. Tensor Flow (11 de noviembre de 2021). Red neuronal convolucional (CNN). Recuperado el 19 de noviembre de 2021, de: https://www.tensorflow.org/tutorials/images/cnn?hl=es-419