Inteligencia artificial avanzada para la ciencia de datos II (Gpo 501)

Momento de Retroalimentación Individual: Implementación de un modelo de Deep Learning

José Edmundo Romo Castillo A01197772

Profesor

Christian Carlos Mendoza Buenrostro

Monterrey, Nuevo León a 30 de Noviembre del 2023

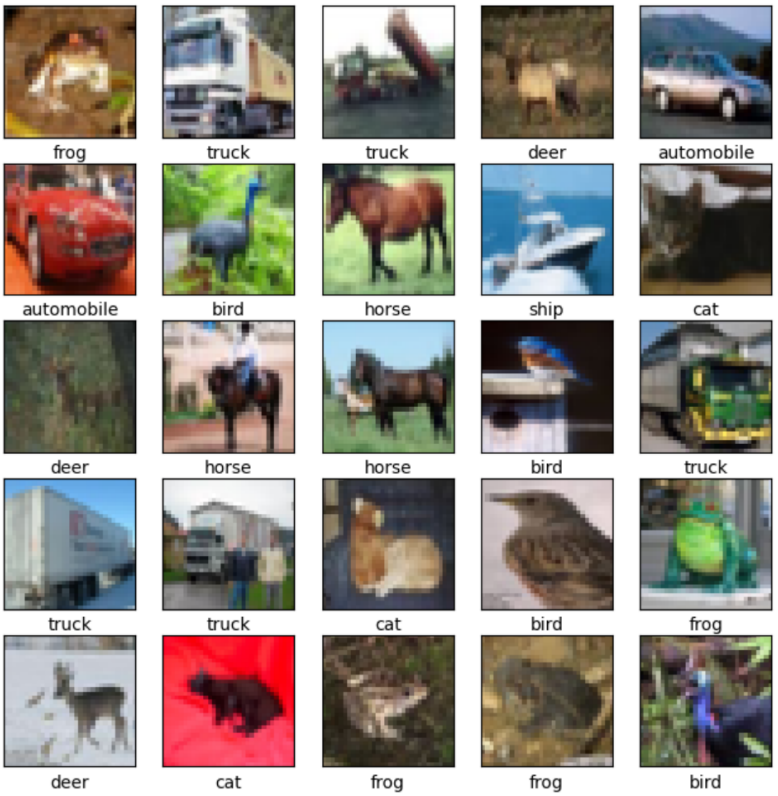
Modelo utilizado

Obtención del modelo

El código fuente fue obtenido desde Kaggle con el título de **Image Classification using CNN for Beginner**, el código fue hecho por el usuario **Anandhu H**, y dicho reporte cuenta con más de 25,000 visualizaciones. En el reporte menciona que utiliza como imagenes provenientes de CIFAR Images.

CIFAR Images

CIFAR Images se refiere a conjuntos de datos de imágenes desarrollados por el Canadian Institute for Advanced Research (CIFAR). CIFAR-10 contiene 60,000 imágenes de 32x32 píxeles, divididas en 10 clases, mientras que CIFAR-100 tiene 100 clases. Estos conjuntos son ampliamente utilizados en la investigación de aprendizaje profundo para tareas de clasificación de imágenes, siendo referencias comunes en la evaluación de modelos de visión por computadora.



Explicación de partes del código

Se utiliza una arquitectura de red Neuronal Convolucional (CNN), estas son las indicadas para el poder procesar imágenes y utilizarlas para entrenar modelos de reconocimiento de imágenes o caras, todo esto gracias a su habilidad para detectar patrones y características entre las imágenes que se le proveen de entrenamiento y posteriormente utilizar esta información para encontrar dichos patrones y las imágenes que se buscarán crear una clasificación.

Capas Convolucionales

Como base para las capas convolucionales se tienen los siguientes datos:

```
]:
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

- Se inician con 32 filtros en la primera capa, que posteriormente luego se incrementa a 64 con la segunda y tercera capa, con este se busca que el algoritmo detecte características de bajo nivel en la primera capa, y en las de mayor nivel que que busque de más alto nivel.
- La activación ‘relu’, utilizada por la eficiencia que provee en la propagación de gradientes y por evitar a las neuronas muertas que pueden llegar a surgir.

Capas más densas

```
[6]:
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

Aquí podemos observar como se agregan capas con mayor densidad, con la opción Flatten para transformar la matriz 3D cuando antes era de un vector 1D.

Con estos cambios se cambio de la cantidad de parámetros

```
]:
model.summary()
```

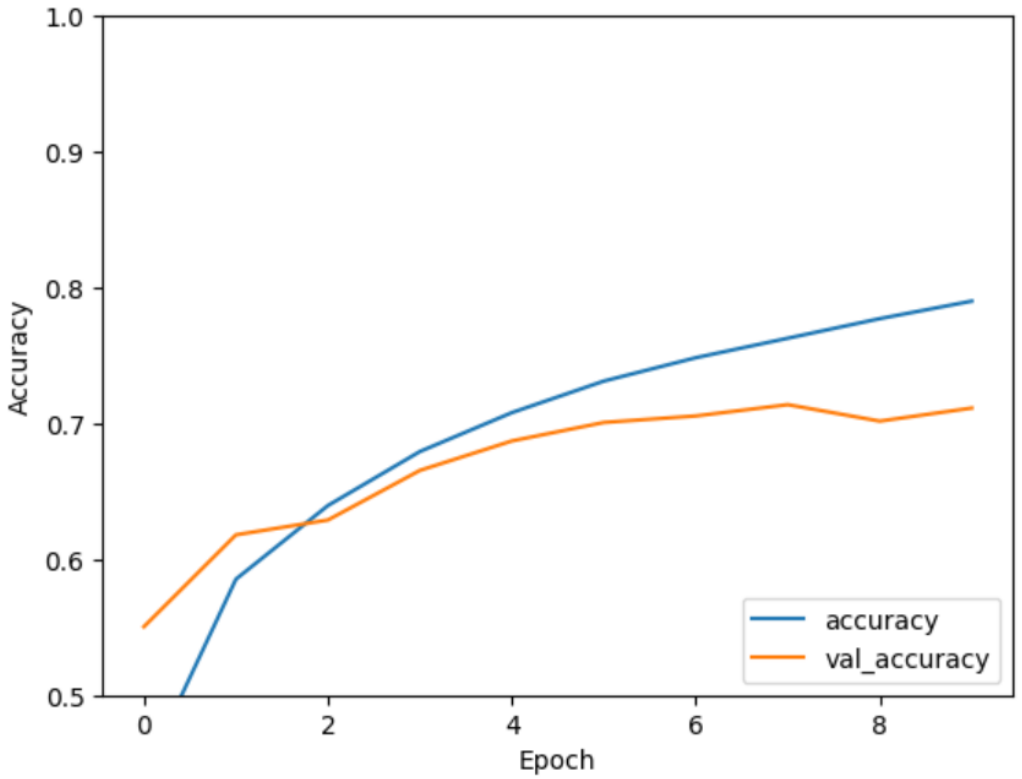
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928
=================================================================
Total params: 56,320
Trainable params: 56,320
Non-trainable params: 0
_________________________________________________________________
```

```
']:
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 64)                65600
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

Resultados originales

```
313/313 - 6s - loss: 0.8555 - accuracy: 0.7113 - 6s/epoch - 18ms/step
```




```
print('Test Accuracy is',test_acc)
```

```
Test Accuracy is 0.7113000154495239
```

Los resultados obtenidos son que se tiene un modelo con una certeza del 71.13%. Las métricas originales con las cuales se llegó a estos resultados fueron las siguientes:

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

En el reporte original se explica que CIFAR cuenta con 10 clases, razón por la cual Dense Layer también utilizaría 10 resultados.

```
# Adam is the best among the adaptive optimizers in most of the cases
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# An epoch means training the neural network with all the
# training data for one cycle. Here I use 10 epochs
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

En el modelo original utilizaron 10 epochs, épocas, para verificar el modelo, también utilizan un optimizador adaptativo, el cual en este caso es Adam. Lo que supondría realizar varias pruebas con estos parámetros para comprobar que los cambios puedan mejorar el rendimiento y certeza del modelo.

Resultados con cambios

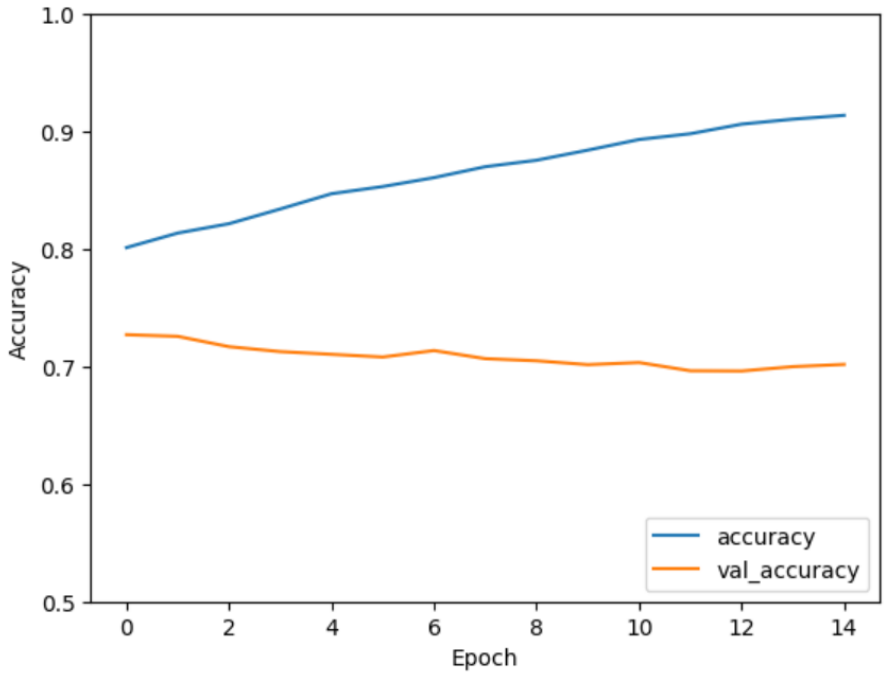
Primer intento

Epocas: 15

```
[12] plt.plot(history.history['accuracy'],label='accuracy')
     plt.plot(history.history['val_accuracy'],label = 'val_accuracy')
     plt.xlabel('Epoch')
     plt.ylabel('Accuracy')
     plt.ylim([0.5, 1])
     plt.legend(loc='lower right')

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

```
313/313 - 6s - loss: 1.4312 - accuracy: 0.7018 - 6s/epoch - 19ms/step
```




```
print('Test Accuracy is',test_acc)
```

```
Test Accuracy is 0.7017999887466431
```

Podemos observar como el resultado de validación de certeza fue peor, dando un 70.18%, pero se observa que el Accuracy en la gráfica empezó desde un punto más alto y llegó a superar el valor de 90%.

Segundo intento

Dense Layer: 128

Epocas: 7

Agregando capas

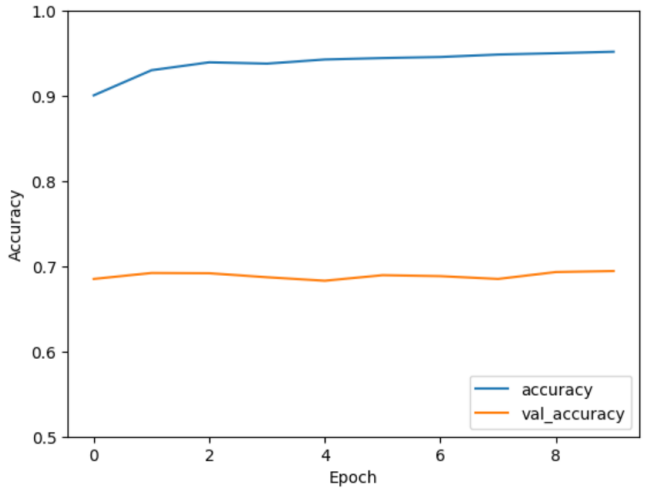
```
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10))
```

```
[24] model.summary()
```

```
plt.plot(history.history['accuracy'],label='accuracy')
plt.plot(history.history['val_accuracy'],label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

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

```
313/313 - 3s - loss: 1.9476 - accuracy: 0.6944 - 3s/epoch - 11ms/step
```




```
[22] print('Test Accuracy is',test_acc)
```

```
Test Accuracy is 0.6944000124931335
```

Aquí se hizo el cambio y se modificó el número de capas a 128, lo que modificó el número de parámetros totales a estar verificando. Lo que hizo que el resultado de certeza bajara a 69.94%.

```
=================================================================
Total params: 125268 (489.33 KB)
Trainable params: 125268 (489.33 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

El número total de parámetros cambió de 122,570 a 125,268, por lo que hubo más información que revisar por parte del algoritmo. Esto quiere decir que cuando se le entrega más parámetros del algoritmo al momento de entrenarlo, este suele tener más información para identificar patrones, por lo que puede pasar que suceda un sobre ajuste. Aunque esta bien recalcar que no esta mal tener mas parametros, pero si la cantidad es muy alta, el poodle terminará memorizando los datos de entrenamiento más que buscar los patrones de identificación.

Conclusión

Al hacer dos intentos, tanto aumentando la cantidad de épocas, como bajandolas y aumentando la cantidad de parámetros, lo único que se pudo conseguir fue un **over adjustment** y un **under adjustment**. Esto quiere decir que no por aumentar los parámetros de un mejor entrenamiento, como se puede observar esto solo provoca que el algoritmo solo detecte como válido las imágenes de entrenamiento, perdiendo el objetivo principal que es de poder relacionar imágenes nuevas con las ya vistas.