

Proyecto Final - Deep Learning

Gourmet AI

Introducción

La comida saludable es una de las acciones más directas que podemos tener hacia nuestra salud a corto, mediano y largo plazo. Desde una mejora en nuestros tejidos musculares, hasta un fortalecimiento de nuestro sistema inmunológico (Beneficios De Comer Saludable, 2021). Sin embargo, el estilo de vida actual de muchas personas no permite introducir comidas saludables en la dieta diaria. Teniendo que recurrir al menos dos veces por semana a restaurantes de comida rápida. El objetivo de este proyecto es utilizar la inteligencia artificial, específicamente redes neuronales, para facilitar el descubrimiento de recetas a través de algoritmos de visión por computadora y de recomendación. Las recomendaciones se realizan a través de dos modelos: una *CNN* que identifica ingredientes de comida en una foto y una *Fully Connected NN* para realizar las predicciones utilizando la salida del modelo anterior. El modelo de detección de ingredientes mostró resultados aceptables y permitió predecir recetas con el segundo modelo así como el modelo de recomendación de recetas permitió sugerir recetas relacionadas a los ingredientes que recibía el modelo.

Antecedentes

El *dataset* de los ingredientes contiene imágenes de 12 categorías: manzana, banano, naranja, tomate, zanahoria, pan, queso, mango, brócoli, uva, limón y piña. Estas imágenes se obtuvieron del repositorio de imágenes de Google (Open Images V7), el conjunto de datos contiene las imágenes y archivos indicando coordenadas de las cajas para cada elemento que se encuentra en esta (Imagen 1).

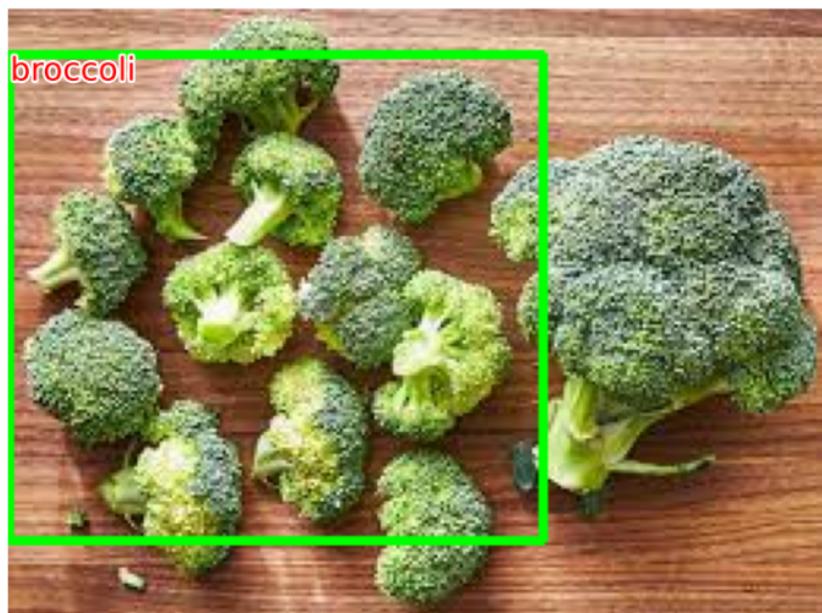


Imagen 1 - Demostración de Google Open Images.

Se utilizó *Python* para realizar todo el procedimiento técnico del proyecto. Principalmente las librerías: *Numpy*, *Tensorflow*, *Pytorch*, *Pandas*, *Matplotlib*, *OpenCV*. Asimismo para ejecutar los modelos se utilizó la herramienta *Google Collaboratory* para aprovechar el uso de la tarjeta gráfica que se provee.

Para entrenar el modelo clasificadorio se realizó una transformación del conjunto de datos proveído por Google. Esto debido a que el proceso para detección de objetos y clasificación necesita generación de posibles “candidatos” y un vector de cambio de cuatro componentes para cada candidato indicando cuánto se debería de mover para encajar perfectamente en las coordenadas proporcionadas por el conjunto de datos importado. Para generar esta información es necesario entender el concepto *intersection over union (IOU)*, este es una que indica la cantidad de área que dos rectángulos tienen en común (Rosebrock, 2023). Esta se calcula de la siguiente manera: $iou = \frac{\text{área compartida}}{\text{área total}}$. Entonces si dos rectángulos están perfectamente traslapados, el valor de este cálculo sería un valor cercano a 1 (Imagen 2).

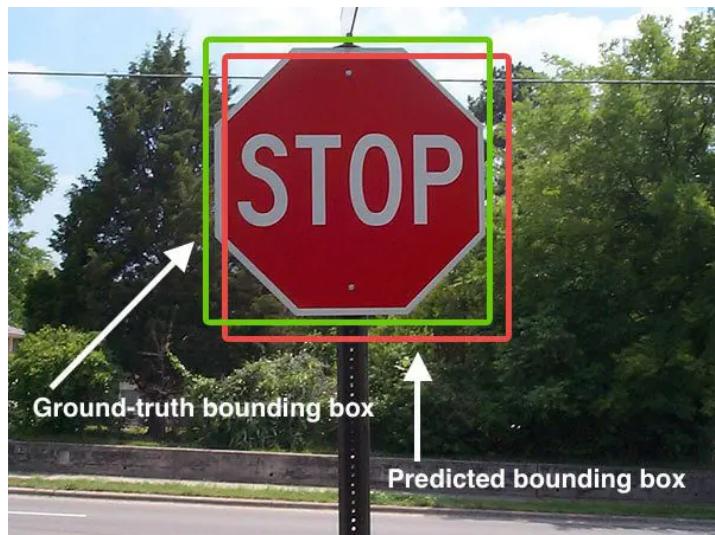


Imagen 2 - Ejemplo de *iou*.

Además, para generar los posibles rectángulos a comparar se está utilizando el paquete *selectivesearch*, este es un algoritmo que genera regiones propuestas que contienen “píxeles parecidos”. Entonces es muy probable que los objetos de interés se encuentren en alguna de esas regiones seleccionadas (Imagen 3).

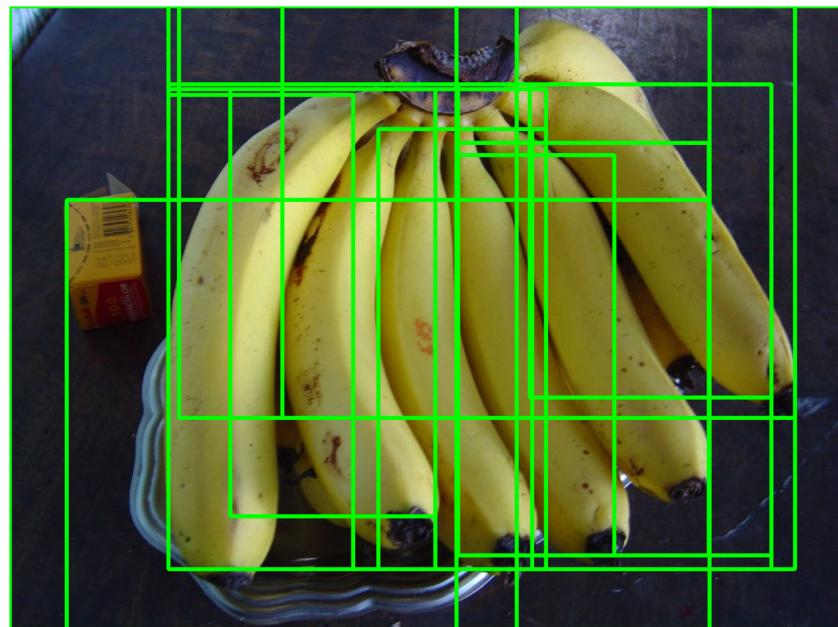


Imagen 3 - Candidatos para una imagen de bananos.

La generación de este conjunto de datos es uno de los procesos más tardados del modelo y una de las razones por las cuales este no puede ser usado en tiempo real, para estos casos es mejor utilizar un modelo como “*YOLO*”.

El dataset de recetas

Las imágenes fueron obtenidas del conjunto de datos de kaggle: **Food.com Recipes and Interactions**. El cual contiene 231,637 recetas indicando el nombre de la receta, minutos de preparación, el id de la persona que contribuyó la receta, la fecha en que fue enviada la receta, tags de identificación, niveles de nutrición, cantidad de pasos, los pasos a seguir, descripción de la receta, ingredientes de la receta y cantidad de ingredientes.

Para la manipulación de datos se emplearon las librerías AST, NumPy y Pandas. En la preparación de los datos, se utilizaron herramientas de Sklearn, Keras y TensorFlow. Específicamente, se usó MultiBinarizer para generar matrices que indican la presencia o ausencia de ingredientes en las recetas, y LabelEncoder para el encoding de los IDs de las recetas. Desde Keras, se aplicó preprocessing.pad_sequences para completar las secuencias hasta alcanzar el tamaño máximo requerido como input para el modelo, y Tokenizer para la tokenización de los ingredientes. Adicionalmente, se empleó to_categorical para realizar un one-hot encoding de los labels de los IDs de las recetas. Finalmente, se utilizó Keras para la construcción del modelo.

Análisis Exploratorio

Conjunto de imágenes

bread	197
apple	117
banana	110
orange	64
carrot	63
cheese	57
lemon	51
tomato	33
mango	31
pineapple	23
broccoli	22

Imagen 4 - Tabla de distribución por categoría de las imágenes

- Las imágenes más comunes son: pan, manzana y bananos.
- La cantidad de imágenes parece ser poca, sin embargo estas se utilizan para generar aún más datos.

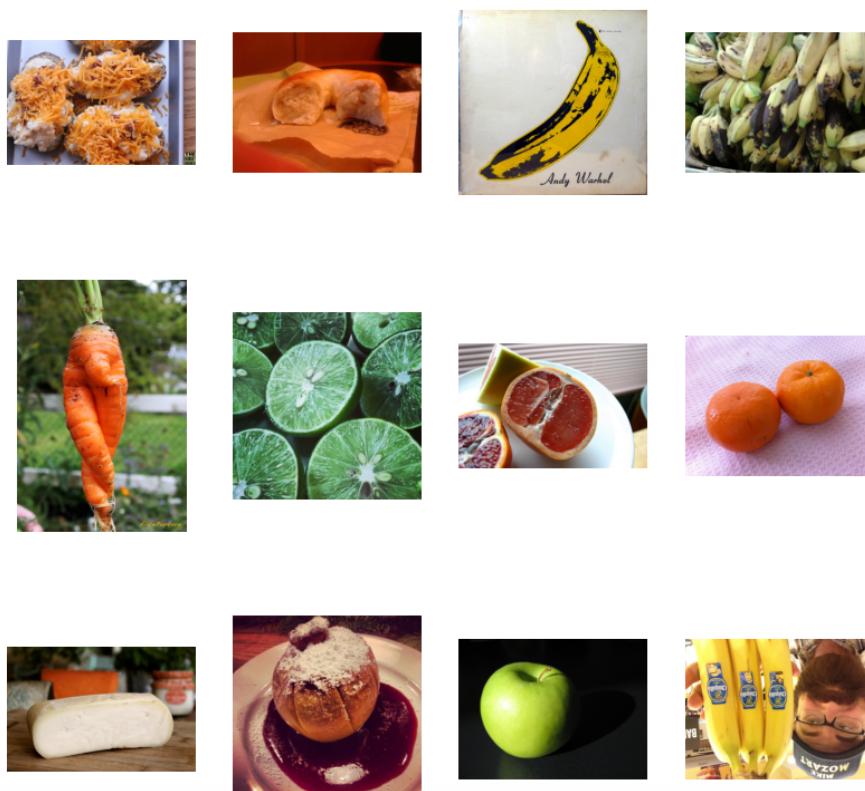


Imagen 5 - Algunas imágenes del conjunto de datos importado

Conjunto de recetas

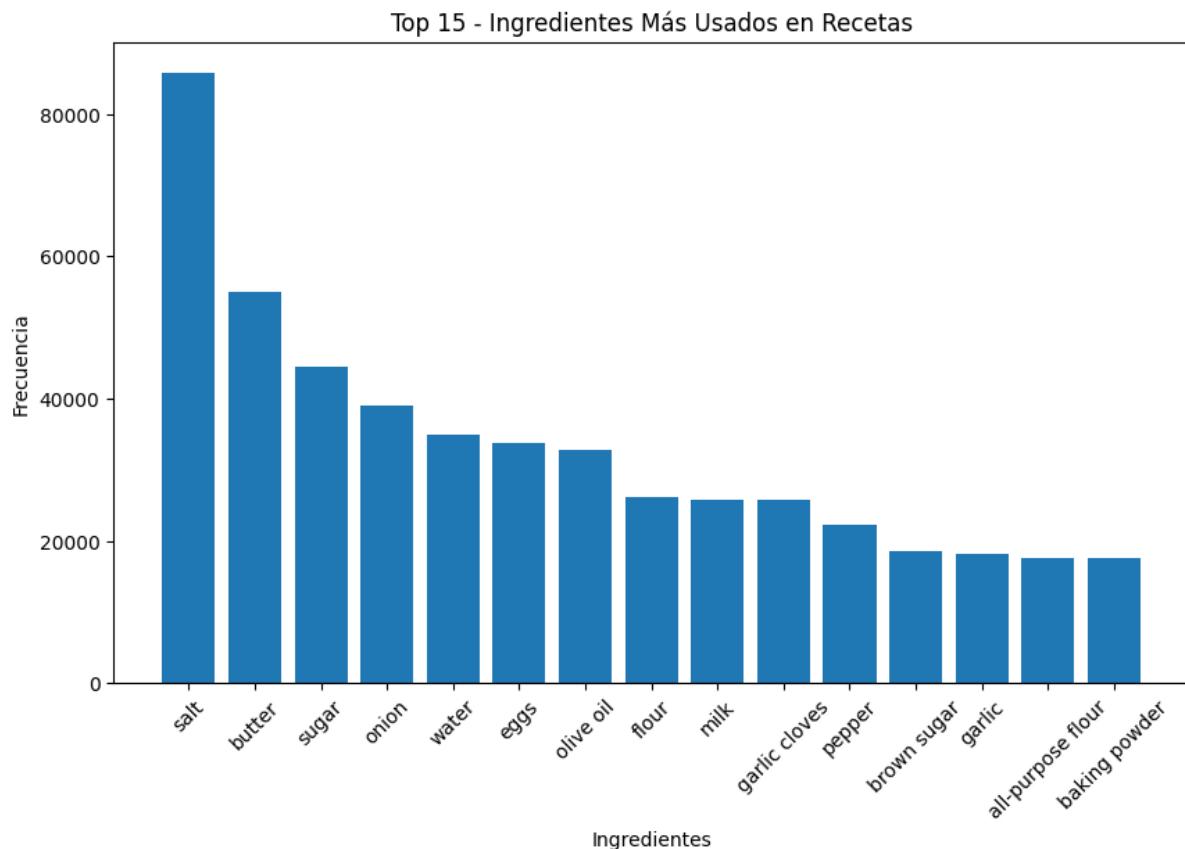


Imagen 6 - Ingredientes más usados en recetas

Los ingredientes más comunes para el conjunto de recetas son: sal, mantequilla, azúcar, cebolla, agua, huevos, entre otros.

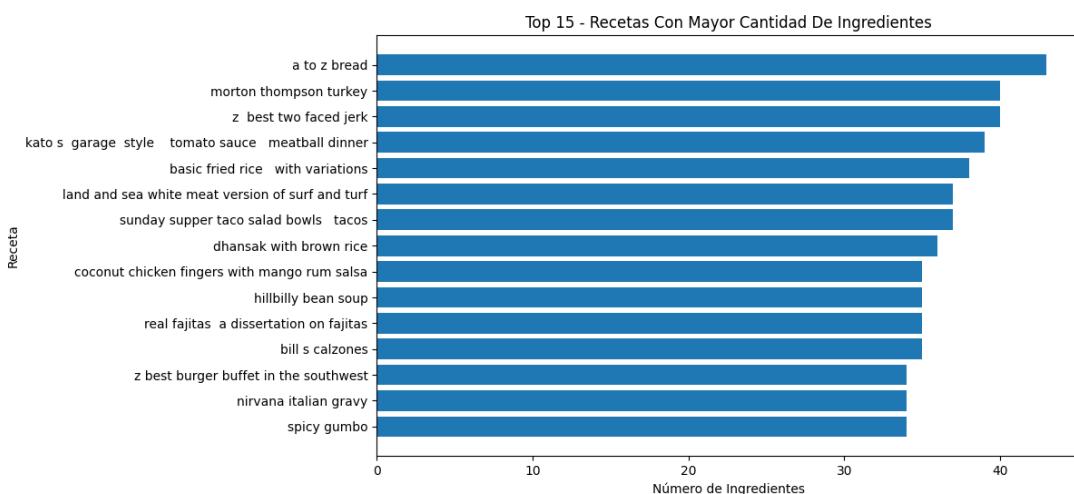


Imagen 7 - Recetas con mayor cantidad de ingredientes

Podemos ver que las recetas con más ingredientes son saladas.

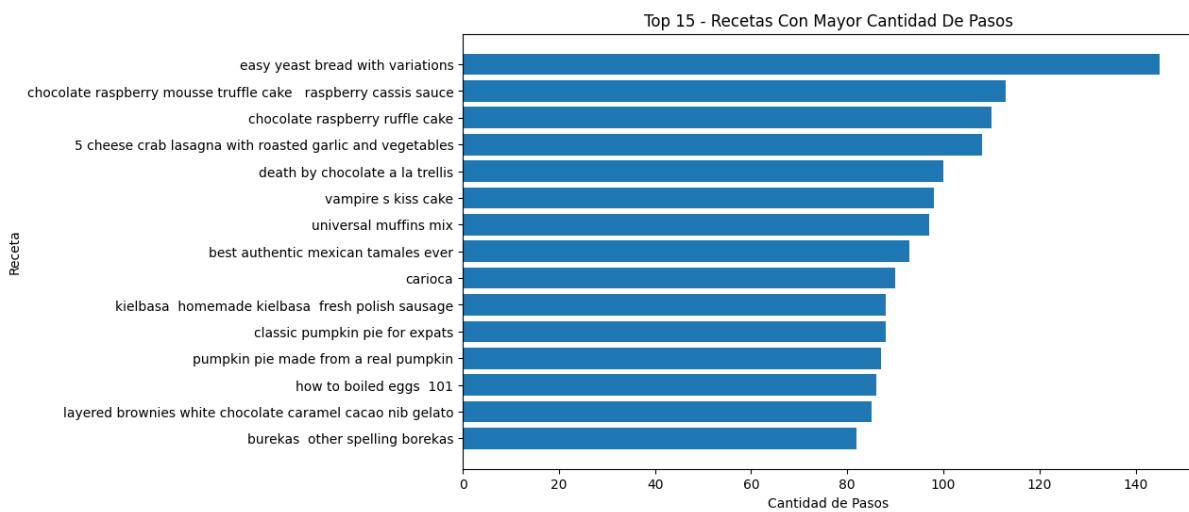


Imagen 8 - Recetas con mayor cantidad de pasos

Sin embargo, podemos ver que las recetas que más pasos tienen son de recetas dulces.

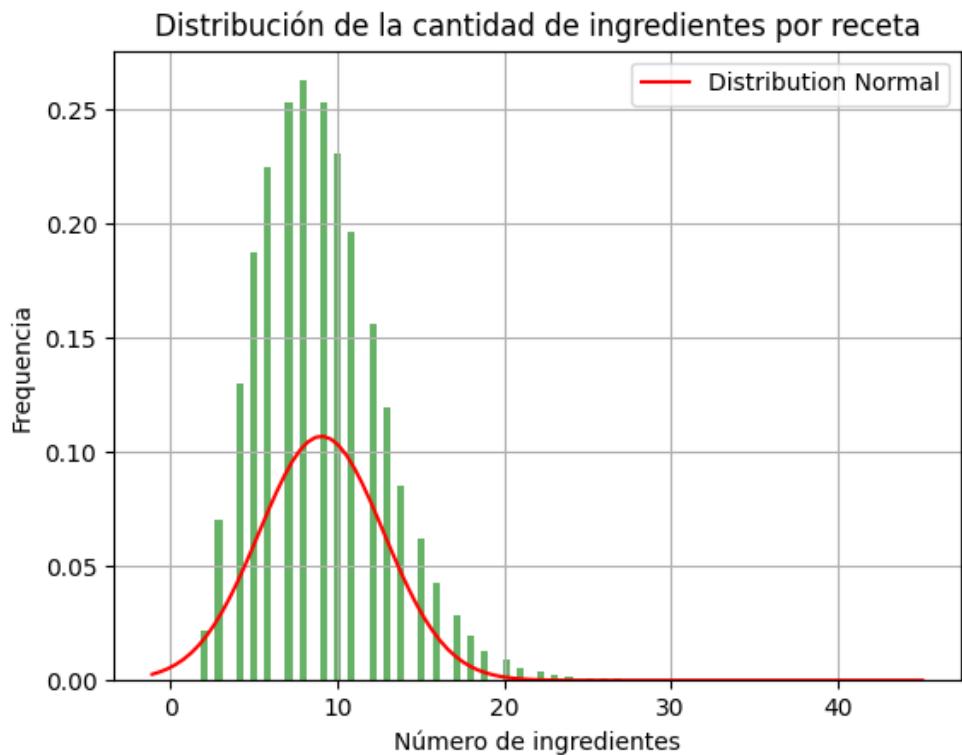


Imagen 9 - Distribución de la cantidad de ingredientes por receta

Respecto a la distribución de ingredientes podemos ver que hay una distribución normal con un sesgo positivo.

Metodología

Modelo de clasificación de imágenes

Para entrenar un modelo que clasifica imágenes de ingredientes se decidió utilizar *transfer learning* con el modelo *VGG16*. Este modelo fué entrenado en la Universidad de Oxford en 2014 y es utilizado debido a su buen rendimiento en problemas de todo propósito en clasificación de imágenes (Ayyadevara, 2020). A continuación se muestra la arquitectura general de este modelo:

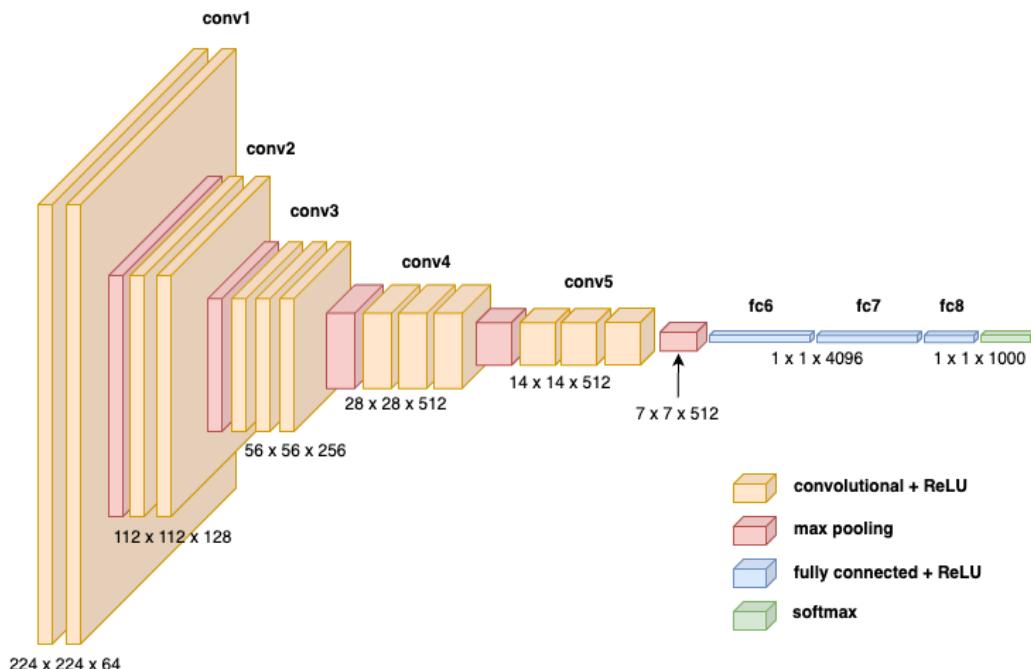


Imagen 10 - Arquitectura de modelo VGG16

Este cuenta con 16 capas y está separado en tres componentes principales: Una red neuronal convolucional, un *adaptive average pooling* y una red clasificatoria (*softmax*). El tipo de *transfer learning* que se escogió es extracción de features, por lo que se descartó el último componente del modelo y se reemplazó por un clasificador que sería entrenado posteriormente. Para detectar múltiples ingredientes en una imagen el modelo propuesto realiza dos cosas: clasificar ingredientes y hacer una regresión para determinar los cuadros en donde estarán las predicciones. Para ambas se utilizó el modelo *VGG16* como *backbone*. El modelo de clasificación es una red neuronal *fully connected* con 12 neuronas en la última capa y una función de activación *Cross Entropy* para determinar la probabilidad de cada clase. Para el modelo de regresión se tiene una capa final con 4 neuronas, cada neurona representa un eje del cuadro donde está la imagen, y una función de activación de tangente hiperbólica para enfatizar cambios drásticos en los pesos de la red y una función de pérdida L1, dado que estamos tratando de minimizar la distancia entre dos rectángulos. A continuación se muestra un resumen de la arquitectura de ambos modelos:

```

(backbone): VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential()
)
(cls_score): Linear(in_features=25088, out_features=12, bias=True)
(bbox): Sequential(
  (0): Linear(in_features=25088, out_features=512, bias=True)
  (1): ReLU()
  (2): Linear(in_features=512, out_features=4, bias=True)
  (3): Tanh()
)
(cel): CrossEntropyLoss()
(sll): L1Loss()
)

```

Imagen 11 - Captura de pantalla modelo cls score y modelo bbox

En la captura de pantalla, el modelo `cls_score` es el clasificadorio, y el modelo `bbox` es el de regresión. Para entrenar el modelo las imágenes tuvieron que ser ajustadas a un tamaño de 244x244 y se normalizaron sus canales con los siguientes valores: media: [0.485,0.456,0.406] y desviación estándar: [0.229,0.224,0.225]. Esto para acomodar las imágenes al modelo *VGG16*.

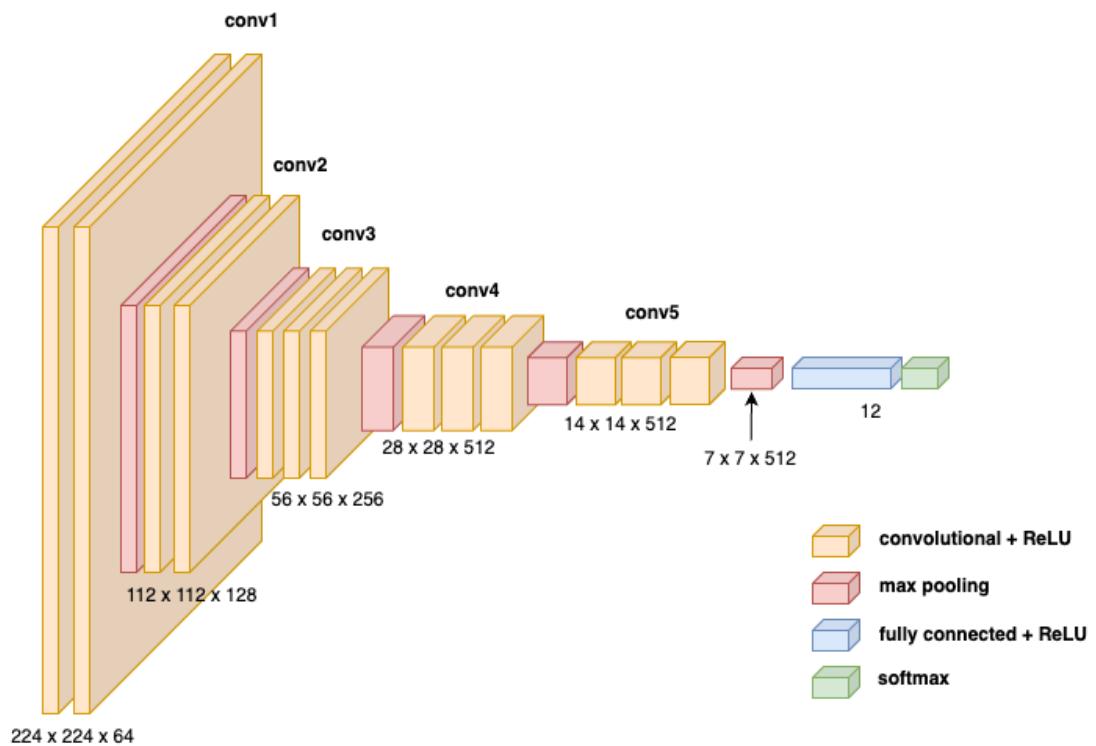


Imagen 12 - Modelo clasificación de imágenes 1

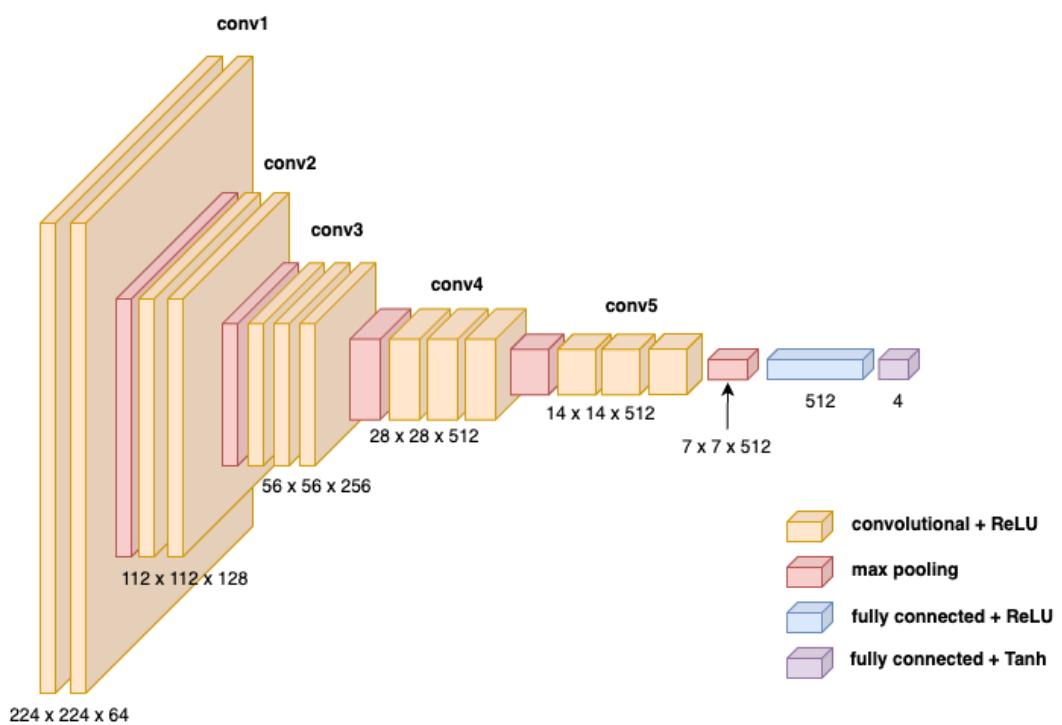


Imagen 13 - Modelo clasificación de imágenes 2

Para todos los modelos se utilizó un optimizador *SDG* dado que dio mejores resultados que el optimizador ADAM. Por último, se tuvo que combinar ambas funciones de pérdida para poder validar el aprendizaje del modelo. Esta se calcula de la siguiente manera: $perdida = perdida_clasificacion + lmb * perdida_regresión$. De esta manera se puede controlar el parámetro lambda para asignar la importancia de la pérdida de regresión. En este caso utilizamos un valor de 10.

Modelo de recomendación de recetas en base a ingredientes - Con capa de embedding

Para este modelo se optó por implementar una red neuronal, la cual se estructura de forma secuencial, iniciando con una capa de *embedding*. Esta capa es esencial para el procesamiento de texto, ya que permite mapear índices de enteros a vectores densos, representando así las palabras. Esta representación captura de manera más efectiva la información y las relaciones entre las palabras.

El tamaño de entrada de la capa de *embedding* corresponde al vocabulario total, equivalente a la cantidad de ingredientes únicos identificados en las recetas. La salida de esta capa es el tamaño del vector de *embedding*, donde cada palabra se representa mediante un vector de esta dimensión específica. Además, se utiliza el parámetro 'input_length' en la capa de *embedding* para definir la longitud máxima de las secuencias de entrada, asegurando así que todas tengan el mismo tamaño. Para manejar secuencias más cortas que la longitud máxima, se aplica un *padding*, garantizando que todas las entradas mantengan una dimensión uniforme. (saxena, 2020)

A continuación, la capa *Flatten* transforma el *output* de la capa anterior de una dimensión 2D (matriz) a una dimensión 1D (vector), siendo esto necesario porque la capa *Dense* espera un *input* en un formato 1D.

Después, una capa densa con 120 neuronas usa una función de activación ReLU. Finalmente, el tamaño de la capa final es determinada por la cantidad de recetas, lo cual representa el número de clases en la tarea de clasificación. En esta última capa se usa una función de activación *softmax* para hacer una clasificación multiclase y permite devolver la probabilidad de distribución a lo largo de las clases objetivo. (Wood, 2019)

Al compilar el modelo, se seleccionó el optimizador Adam y se eligió la función de pérdida 'categorical cross entropy'. Esta función se consideró apropiada para el problema debido a su naturaleza de clasificación multiclase, donde cada etiqueta es mutuamente excluyente, lo que la hace ideal para este tipo de tarea. ("What Are the Advantages and Disadvantages of Using Cross-Entropy Loss for Classification Tasks?," 2023) Además, la métrica seleccionada para evaluar durante el entrenamiento fue la precisión del modelo, lo que permitirá medir de manera efectiva su desempeño.

Modelo de recomendación de recetas - Sin capa de *embedding*

Para este modelo más sencillo, se implementó una red neuronal estructurada de forma secuencial. Comenzando con una capa densa (Dense) de 128 neuronas, cuya entrada corresponde al número total de ingredientes únicos en las recetas. Se utilizó la función de activación ReLU en esta capa para introducir no linealidades en el modelo, lo que permite el aprendizaje de patrones más complejos. (Brownlee, 2019)

Posteriormente, se añadió otra capa densa con 64 neuronas, también utilizando ReLU como función de activación. La capa final es una capa densa con un número de neuronas igual a la cantidad de recetas en el conjunto de datos. En esta capa, se empleó la función de activación *softmax* para convertir las salidas en probabilidades, asignando a cada neurona la probabilidad de que la entrada correspondiera a una clase específica.

Al compilar el modelo, se optó por el optimizador Adam y la función de pérdida 'categorical cross entropy', considerada apta para problemas de clasificación multiclase con etiquetas mutuamente excluyentes. La precisión del modelo se seleccionó como la métrica de evaluación durante el entrenamiento, proporcionando una medida efectiva de su rendimiento.

Resultados

Modelos para clasificación de imágenes:

- Modelo 1: Este modelo fue capaz de generalizar correctamente algunas clases. Sin embargo la mayoría de predicciones las catalogaba como la clase “naranja”. A continuación se muestra la curva de aprendizaje y algunas pruebas.

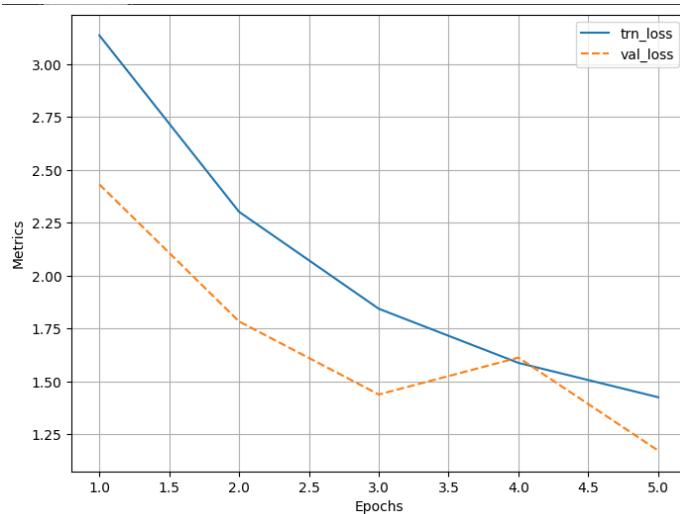


Imagen 14 - Curva de aprendizaje modelo 1



Imagen 15 - Tomates clasificados como “naranjas”

- Modelo 2: Tiene un mejor rendimiento con el conjunto de validación, se muestran clasificaciones para cada clase. El único ingrediente que no logró generalizar el modelo fué el mango. Se adjuntará un zip con más imágenes reconocidas por el modelo.

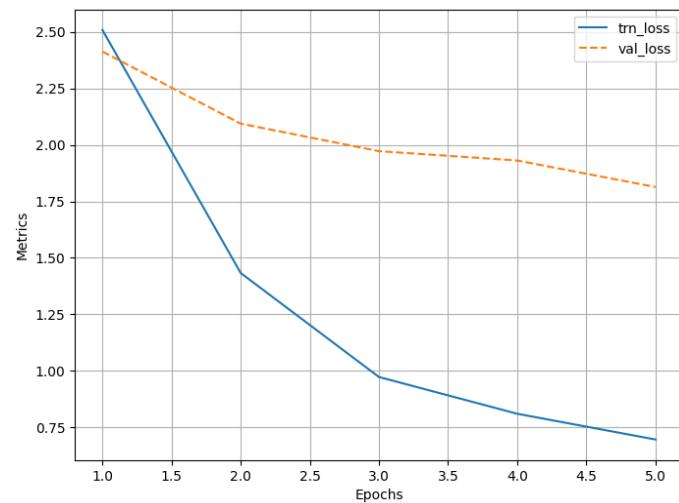


Imagen 16 - curva de aprendizaje modelo 2

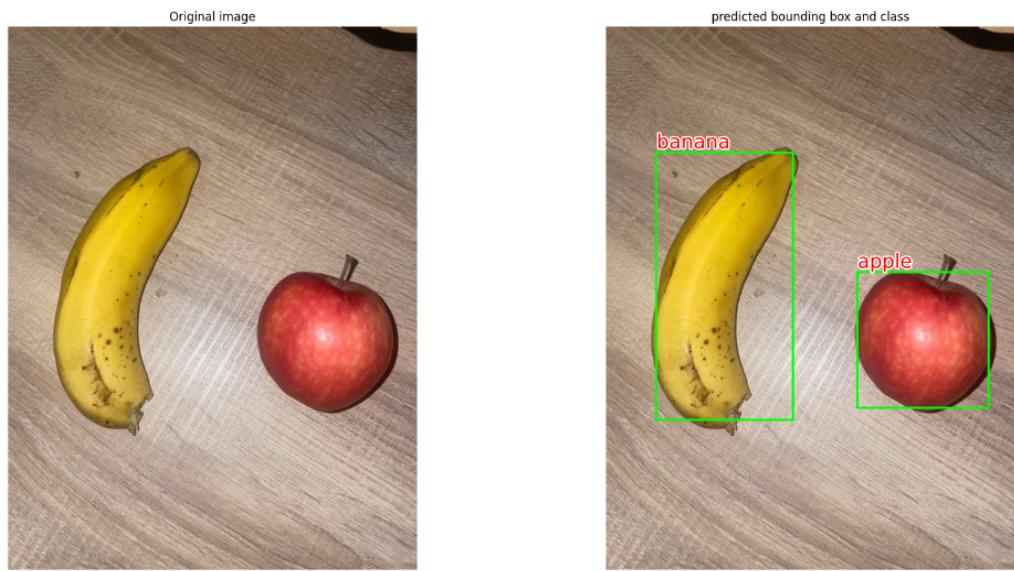


Imagen 17 - Identificación de banana y manzana

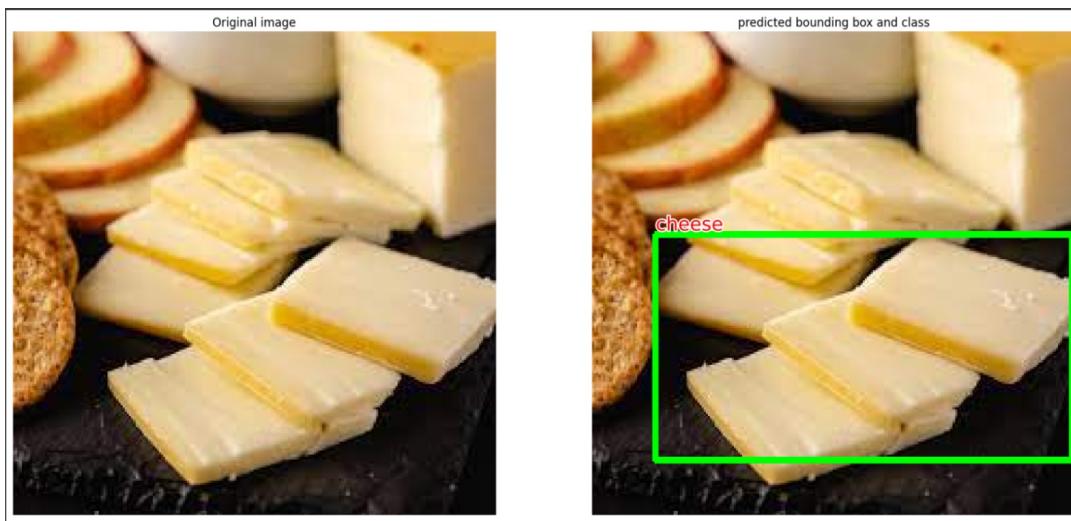


Imagen 17 - Identificación de queso

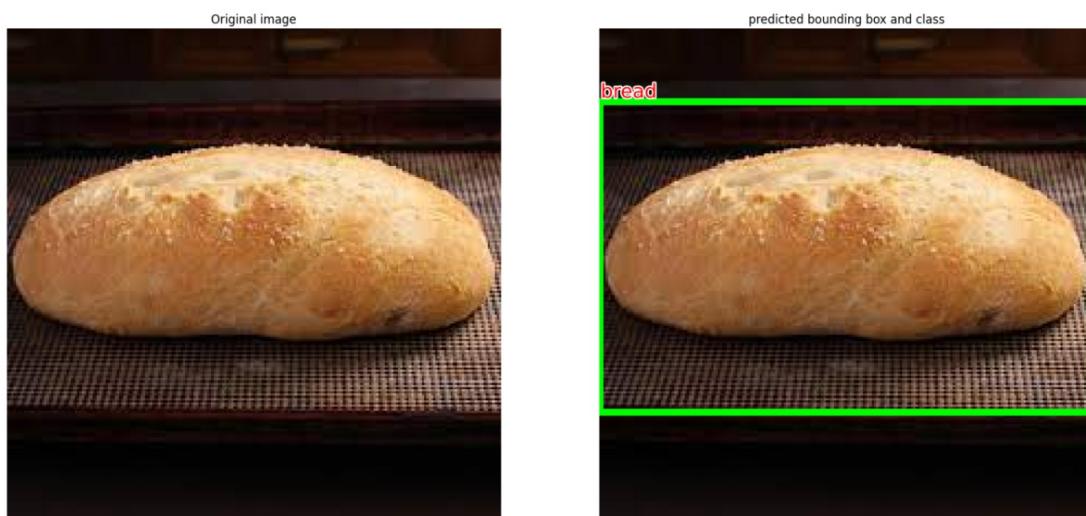


Imagen 18 - Identificación de pan

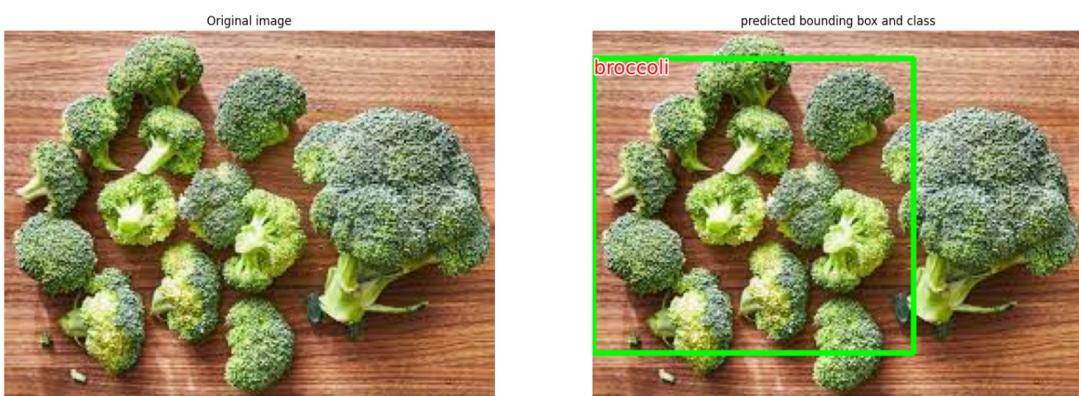


Imagen 19 - Identificación de brócoli

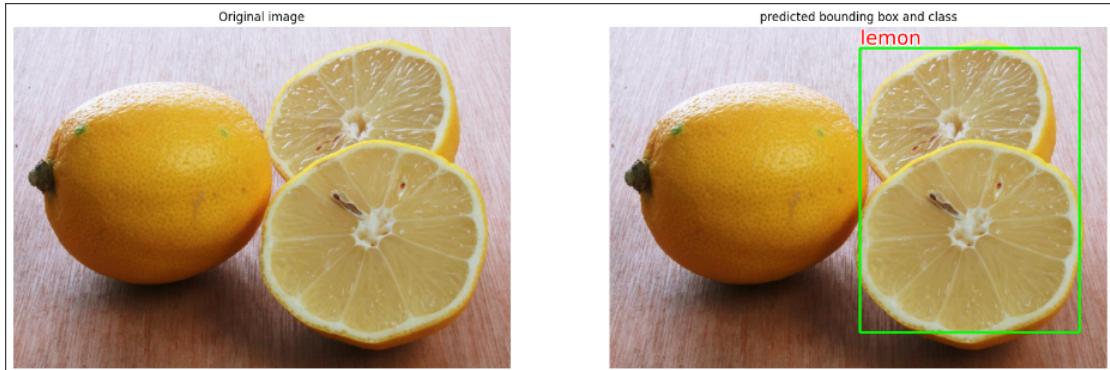


Imagen 20 - Identificación de limones



Imagen 21 - Identificación de mango

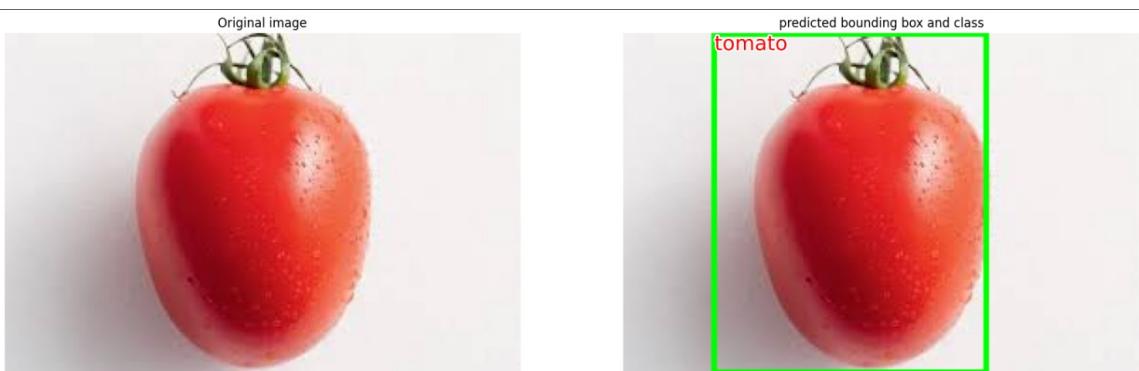


Imagen 22 - Identificación de tomate



Imagen 23 - Identificación de zanahoria



Imagen 24 - Identificación de piña

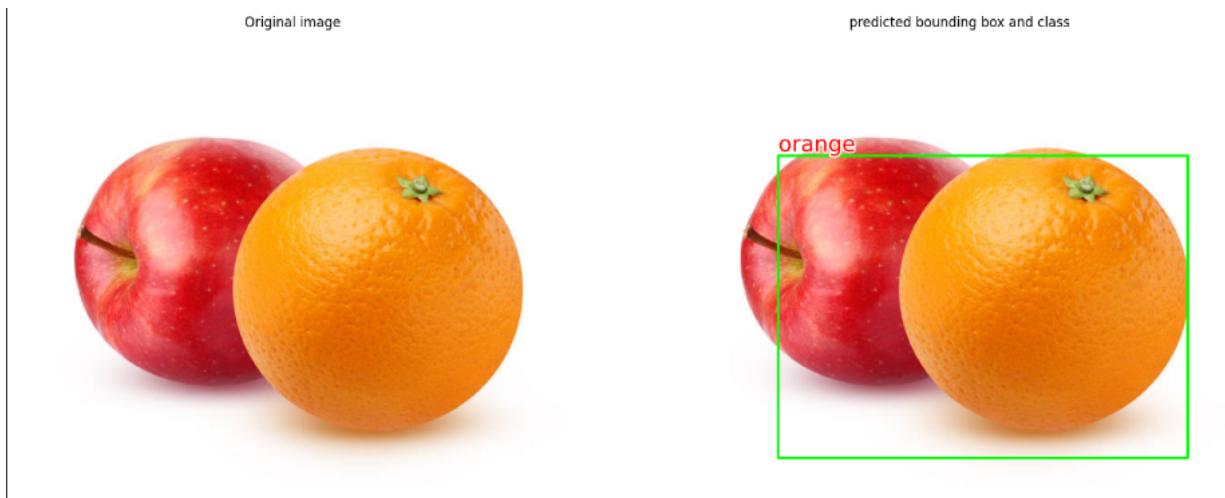


Imagen 25 - Identificación de naranja

- Modelo 3: No hubo cambios significativos en comparación al modelo anterior.

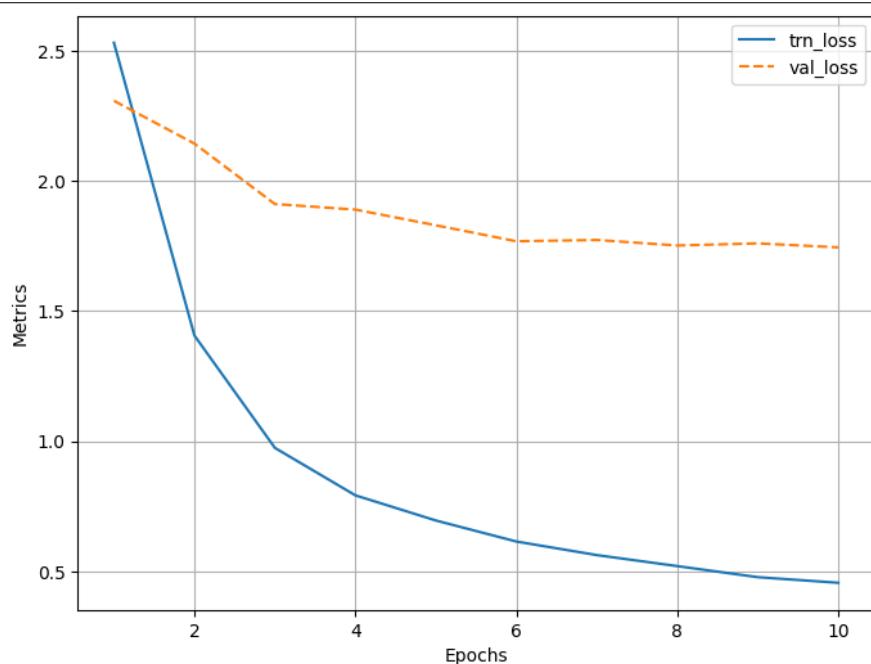


Imagen 26 - curva de aprendizaje modelo 3

Modelo 4:

Modelo con embedding

```
Epoch 1/15
350/350 [=====] - 2s 6ms/step - loss: 9.3843 - accuracy: 0.0000e+00
Epoch 2/15
350/350 [=====] - 2s 6ms/step - loss: 9.3341 - accuracy: 1.7872e-04
Epoch 3/15
350/350 [=====] - 2s 6ms/step - loss: 9.1603 - accuracy: 0.0013
Epoch 4/15
350/350 [=====] - 2s 6ms/step - loss: 7.2686 - accuracy: 0.0774
Epoch 5/15
350/350 [=====] - 2s 6ms/step - loss: 2.2492 - accuracy: 0.5804
Epoch 6/15
350/350 [=====] - 2s 7ms/step - loss: 0.2999 - accuracy: 0.9368
Epoch 7/15
350/350 [=====] - 2s 7ms/step - loss: 0.0955 - accuracy: 0.9785
Epoch 8/15
350/350 [=====] - 3s 7ms/step - loss: 0.0592 - accuracy: 0.9855
Epoch 9/15
350/350 [=====] - 2s 6ms/step - loss: 0.0440 - accuracy: 0.9899
Epoch 10/15
350/350 [=====] - 2s 7ms/step - loss: 0.0370 - accuracy: 0.9923
Epoch 11/15
350/350 [=====] - 2s 7ms/step - loss: 0.0341 - accuracy: 0.9930
Epoch 12/15
350/350 [=====] - 2s 7ms/step - loss: 0.0314 - accuracy: 0.9934
Epoch 13/15
...
Epoch 14/15
350/350 [=====] - 2s 7ms/step - loss: 0.0265 - accuracy: 0.9941
Epoch 15/15
350/350 [=====] - 2s 7ms/step - loss: 0.0261 - accuracy: 0.9945
```

Imagen 27 - modelo embedding

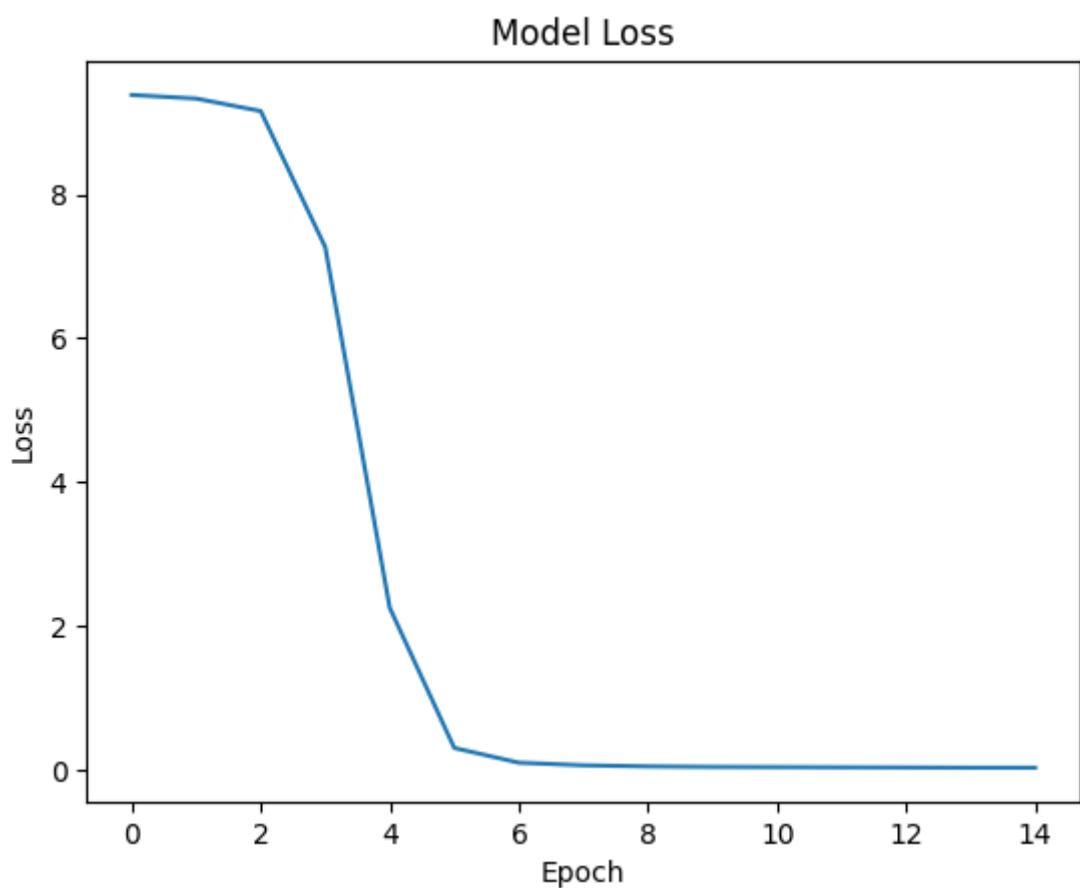


Imagen 28 - curva pérdida modelo embedding

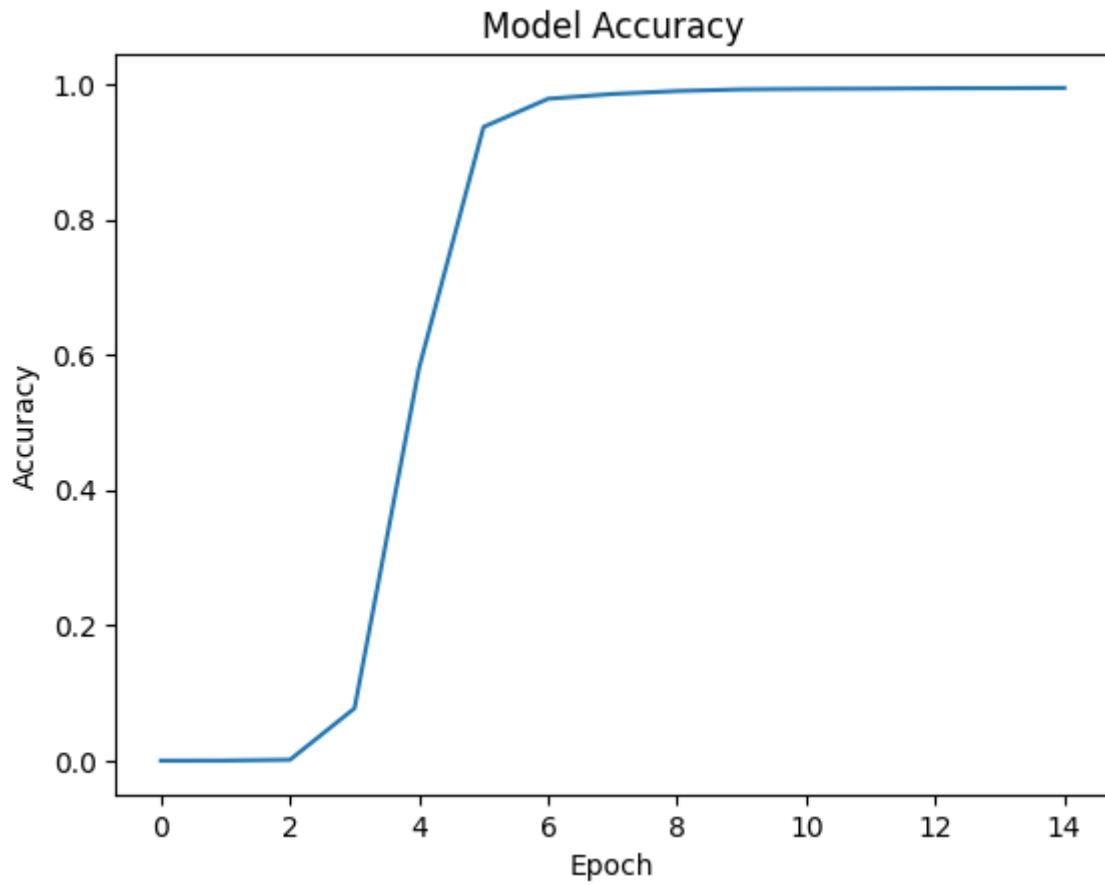


Imagen 29 - curva *accuracy* modelo embedding

```

try_ingredients = ['bread', 'cheese']
try_ingredients_sequence = tokenizer.texts_to_sequences([try_ingredients])
try_ingredients_padded = pad_sequences(try_ingredients_sequence, maxlen=padded_sequences.shape[1])

predictions = model.predict(try_ingredients_padded)
predicted_recipe_id = label_encoder.inverse_transform(np.argmax(predictions))

# Top 3 de recetas mas probables
top_3 = predictions.argsort()[0][-3:][::-1]
for i in top_3:
    recipe_id = label_encoder.inverse_transform([i])[0]
    recipe_name = recipes[recipes['id'] == recipe_id]['name'].values[0]
    probability = predictions[0][i]
    print(f'{recipe_id} {recipe_name}: {probability*100:.2f}%')
    ✓ 0.0s

1/1 [=====] - 0s 28ms/step
[9518] 30 second sandwich: 92.82%
[531520] cheeze bread: 2.84%
[37603] cheese pulls: 1.82%

```

Imagen 29 - resultado modelo de predicción

Prueba con todos los ingredientes:

16	chile rellenos	43026	45	52268	2002-10-14	['60-minutes-or-less', 'time-to-make', 'course...']	[94.0, 10.0, 0.0, 11.0, 11.0, 21.0, 0.0]	9	['drain green chiles', 'sprinkle cornstarch on...']	a favorite from a local restaurant no longer i...	{egg roll wrap, whole green chilies, cheese, c...	5
----	----------------	-------	----	-------	------------	---	--	---	---	---	---	---

```
try_ingredients = ['egg roll wrap', 'whole green chilies', 'cheese', 'cornstarch', 'oil']
try_ingredients_sequence = tokenizer.texts_to_sequences([try_ingredients])
try_ingredients_padded = pad_sequences(try_ingredients_sequence, maxlen=padded_sequences.shape[1])

predictions = model.predict(try_ingredients_padded)
predicted_recipe_id = label_encoder.inverse_transform([np.argmax(predictions)])

# Top 3 de recetas mas probables
top_3 = predictions.argsort()[0][-3:][::-1]
for i in top_3:
    recipe_id = label_encoder.inverse_transform([i])[0]
    recipe_name = recipes[recipes['id'] == recipe_id]['name'].values[0]
    probability = predictions[0][i]
    print(f'{recipe_id} {recipe_name}: {probability*100:.2f}%')

✓ 0.0s
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<locals>.predict_function at
1/1 [=====] - 0s 33ms/step
[43026] chile rellenos: 99.91%
[60692] pepsi spam wrap: 0.01%
[294096] tangy tuna pasta salad: 0.01%
```

Imagen 30 - segunda prueba con modelo de predicción

Modelo 5:

Modelo sin embedding

```
Epoch 1/15
350/350 [=====] - 2s 4ms/step - loss: 9.3839 - accuracy: 0.0000e+00
Epoch 2/15
350/350 [=====] - 1s 4ms/step - loss: 9.3192 - accuracy: 3.5743e-04
Epoch 3/15
350/350 [=====] - 1s 4ms/step - loss: 8.1820 - accuracy: 0.0138
Epoch 4/15
350/350 [=====] - 1s 4ms/step - loss: 4.6300 - accuracy: 0.2619
Epoch 5/15
350/350 [=====] - 1s 4ms/step - loss: 1.4774 - accuracy: 0.7082
Epoch 6/15
350/350 [=====] - 1s 4ms/step - loss: 0.3249 - accuracy: 0.9391
Epoch 7/15
350/350 [=====] - 2s 5ms/step - loss: 0.1347 - accuracy: 0.9727
Epoch 8/15
350/350 [=====] - 2s 4ms/step - loss: 0.0884 - accuracy: 0.9810
Epoch 9/15
350/350 [=====] - 1s 4ms/step - loss: 0.0704 - accuracy: 0.9841
Epoch 10/15
350/350 [=====] - 1s 4ms/step - loss: 0.0577 - accuracy: 0.9869
Epoch 11/15
350/350 [=====] - 1s 4ms/step - loss: 0.0537 - accuracy: 0.9872
Epoch 12/15
350/350 [=====] - 1s 4ms/step - loss: 0.0494 - accuracy: 0.9877
Epoch 13/15
...
Epoch 14/15
350/350 [=====] - 1s 4ms/step - loss: 0.0443 - accuracy: 0.9887
Epoch 15/15
350/350 [=====] - 1s 4ms/step - loss: 0.0416 - accuracy: 0.9891
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Imagen 29 - modelo sin embedding

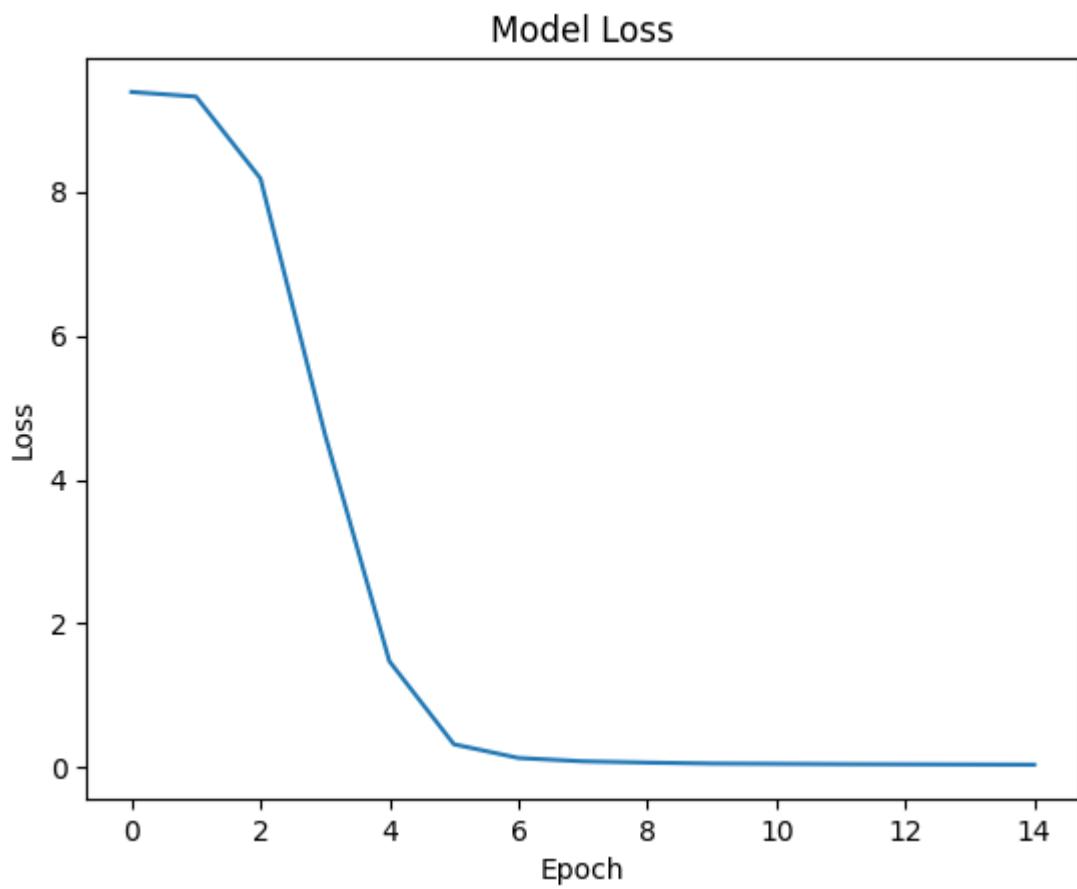


Imagen 30 - pérdida modelo sin embedding

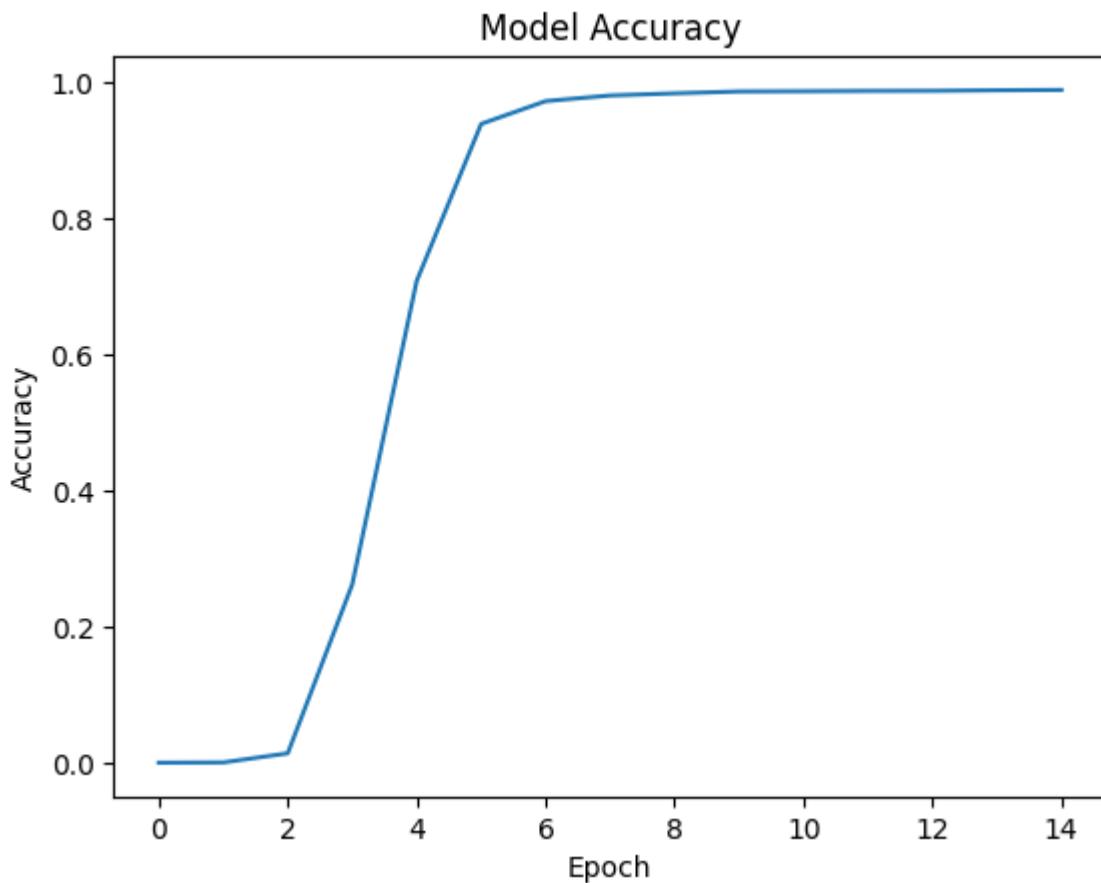


Imagen 31 - accuracy modelo sin embedding

Ingredientes: bread and cheese.

```

input_binary = mlb.transform([try_ingredients])

# Predicciones
predictions = model.predict(np.array(input_binary))[0]

# Top 3 de recetas mas probables
top_3 = predictions.argsort()[-3:][::-1]
for i in top_3:
    recipe_id = label_encoder.inverse_transform([i])[0]
    recipe_name = recipes[recipes['id'] == recipe_id]['name'].values[0]
    probability = predictions[i]
    print(f'{recipe_id} {recipe_name}: {probability*100:.2f}%')

✓ 0.0s
1/1 [=====] - 0s 30ms/step
[531520] cheeze bread: 83.13%
[9518] 30 second sandwich: 8.42%
[468219] caramelized cheese covered grilled cheese sandwich: 1.68%
```

Imagen 31 - predicción modelo sin embedding

Prueba con todos los ingredientes de una receta:

```
input_binary = mlb.transform([try_ingredients])

# Predicciones
predictions = model.predict(np.array(input_binary))[0]

# Top 3 de recetas mas probables
top_3 = predictions.argsort()[-3:][::-1]
for i in top_3:
    recipe_id = label_encoder.inverse_transform([i])[0]
    recipe_name = recipes[recipes['id'] == recipe_id]['name'].values[0]
    probability = predictions[i]
    print(f'{recipe_id} {recipe_name}: {probability*100:.2f}%')

✓ 0.0s

WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_f
1/1 [=====] - 0s 35ms/step
[43026] chile rellenos: 99.52%
[176453] wrapped asparagus cheese: 0.20%
[248587] black olive appetizers: 0.10%
```

Imagen 32 - predicción modelo con embedding

Discusión

El primer modelo que se entrenó para detección de imágenes no tuvo el rendimiento deseado al probarlo con un set de prueba. Mayormente clasificaba los ingredientes como “naranjas” y en algunos casos no encontraba ninguno. Además su entrenamiento duró dos horas a pesar de usar la herramienta premium de *Google Colab*. Al analizar las posibles causas de estos resultados se llegaron a las siguientes conclusiones:

- Algunas imágenes tienen rectángulos muy pequeños en comparación a toda la imagen: Por ejemplo hay fotos de mesas con muchos ingredientes y la caja que encierra a la categoría deseada es muy pequeña, por lo que el algoritmo de *selectivesearch* no lograba encontrar correctamente los candidatos. A continuación se muestra un ejemplo de este caso:



Imagen 33 - un broccoli está “escondido” en un plato de comida

- No se modificó el tamaño de las imágenes: Algunas imágenes tenían una resolución de 1024 x 1024, esto es innecesario para un modelo de clasificación dado que no se pierde ninguna información al reducir el tamaño de la imagen y es la causa principal del proceso largo de entrenamiento.
- No se seleccionaron imágenes al azar: Las imágenes que se proporcionaron al modelo no fueron ordenadas de manera aleatoria antes, por lo que algunas clases quedaron fuera del conjunto de entrenamiento.

En el segundo modelo se abordaron estos problemas: primero solo se tomaron en cuenta las imágenes en donde el rectángulo que indicaba la presencia de un ingrediente tomaba un área de al menos 20% de toda la imagen, de esta forma se ignoraron entradas que podrían confundir a la red. Además, se redujo el ancho de todas las imágenes a 540 píxeles, manteniendo siempre la proporción entre ancho - altura. Por último se tomaron muestras de manera aleatoria antes de crear el conjunto de entrenamiento y de validación. Estas acciones influyeron drásticamente en los resultados. Ahora las gráficas de aprendizaje tienen más sentido dado que la pérdida de validación siempre es mayor a la de entrenamiento. A pesar que los valores son mayores, al visualizar manualmente las imágenes se encontraron mejores resultados.

El tercer modelo es una mejora del anterior, se utilizó la aumentación de datos para intentar mejorar las predicciones que realiza el modelo, también se incrementó la cantidad de épocas. La técnica de aumento de datos que se utilizó es *Gaussian Blur* que de manera aleatoria modifica las imágenes para hacerlas borrosas. El motivo de esta decisión es para abordar más casos de calidad en las fotografías, dado que en la práctica un modelo no siempre recibirá imágenes de la mejor calidad. A pesar de esto no se vieron resultados significativos que justifiquen los compromisos de tiempo realizados al entrenar el modelo.

Respecto a los modelos de clasificación de recetas, modelos 4 y 5, se observó que ambos alcanzaron un alto porcentaje de precisión y una pérdida mínima. Esta eficacia se debe a que fueron entrenados utilizando el conjunto completo de recetas. Si el conjunto se dividiera en secciones de entrenamiento y prueba, se produciría un sobreajuste, ya que cada receta tiene un único id, lo que impide que el modelo generalice los datos adecuadamente. La precisión de los modelos se manifiesta especialmente cuando se ingresan todos los ingredientes de una receta, resultando en un alto porcentaje de recomendación para ambos modelos.

Por otra parte, fue interesante observar que, mientras un modelo con embedding puede recomendar recetas que contienen los ingredientes proporcionados, también sugiere recetas con ingredientes similares cuando la relación directa es menor. En cambio, un modelo sin embedding tiende a recomendar recetas que mantienen una relación más directa con los ingredientes ingresados, aunque la correlación ingredientes-receta disminuya.

Conclusiones

- El uso de *transfer learning* con el modelo *VGG16* permite simplificar el entrenamiento para detección de objetos.
- El entrenamiento de un modelo de detección de objetos requiere dos acciones: regresión de regiones propuestas y clasificación de imágenes.
- Mientras que el modelo con embedding es capaz de recomendar recetas no solo basándose estrictamente en los ingredientes ingresados sino también en ingredientes similares, el modelo sin embedding se enfoca más en recomendar recetas que correspondan directamente a los ingredientes proporcionados. Esto indica que el embedding añade una capa de flexibilidad y comprensión contextual en las recomendaciones de recetas.
- La habilidad de los modelos para recomendar recetas de manera precisa cuando se ingresan todos los ingredientes sugiere una alta eficacia en la correlación entre ingredientes y recetas.

Enlace de github: Enlace de github:

https://github.com/MGonza20/ProyectoFinal_DeepLearning

Referencias bibliográficas

Beneficios de comer saludable. (2021, May 22). Centers for Disease Control and Prevention.

<https://www.cdc.gov/nutrition/resources-publications/spanish/beneficios-de-comer-saludable.html>

Simonyan, K. (2014, Sept 4). Very deep convolutional networks for Large-Scale image recognition. arXiv.org. <https://arxiv.org/abs/1409.1556>

Ayyadevara, V. K. R. (2020). HANDS-ON COMPUTER VISION WITH PYTORCH: Implement Deep Learning to Build Real-world Computer. . . Vision Applications Using Python.

Rosebrock, A. (2023, May 12). Intersection over Union (IOU) for object detection - PyImageSearch. PyImageSearch.

<https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>

saxena, sawan. (2020, October 3). Understanding Embedding Layer in Keras -

Analytics Vidhya - Medium. Retrieved November 24, 2023, from Medium website:

<https://medium.com/analytics-vidhya/understanding-embedding-layer-in-keras-bbe3ff1327ce>

Wood, T. (2019, May 17). Softmax Function. Retrieved November 24, 2023, from

DeepAI website:

<https://deepai.org/machine-learning-glossary-and-terms/softmax-layer>

What are the advantages and disadvantages of using cross-entropy loss for classification tasks? (2023). Retrieved November 24, 2023, from LinkedIn.com website:

<https://www.linkedin.com/advice/0/what-advantages-disadvantages-using-cross-entropy>

Brownlee, J. (2019, January 8). A Gentle Introduction to the Rectified Linear Unit (ReLU) - MachineLearningMastery.com. Retrieved November 24, 2023, from MachineLearningMastery.com website:

<https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/>