Detección de defectos superficiales en aceros haciendo uso de Deep Learning

Por: Luis David Morales Aguilar

Entrega #2- Informe proyecto

Estructura del notebook y arquitectura

Tal como se mencionó en el primer informe, los datos de la competencia se componen por 7096 imágenes de superficies de acero, obtenidas durante los diferentes procesos de fabricación de metales. Las imágenes se almacenan en una carpeta con sus respectivos nombres, mientras que el archivo CSV asociado contiene dos columnas: la primera con los nombres de las imágenes y la segunda con la clasificación correspondiente.

La clasificación está representada por un número entero del 1 al 4, asignado de la siguiente manera:

- Clase 1: Sin defectos superficiales
- Clase 2: Scratches o rayones
- Clase 3: Inclusiones no metálicas
- Clase 4: Escamas de laminación

Para esta entrega se ha realizado un solo notebook que contiene la siguiente estructura:

1- Cargar y preparación de datos: en esta sección se sube el API de autenticación y se descargan los archivos correspondientes a la competición directamente de Kaggle. Posteriormente se descomprimen y se usa Pandas para abrir el archivo que contiene las imágenes. Una vez se tienen las imágenes descomprimidas se separan los datos, en donde el 80% de la cantidad de imágenes seleccionadas (una cantidad muy inferior a la disponible) se elige para entrenamiento, y el 20% restante se elige para evaluar.

2- Preparación de los dataset de entrenamiento y validación: Para esto se hace uso de Tensor Flow, en donde se decodifican, redimensionan y se normalizan las imágenes. La siguiente figura muestra una configuración del código.

```
[9]  import tensorflow as tf

     def load_and_preprocess_image(image_name, label, image_dir):
         image_name = tf.strings.join([image_dir, "/", image_name])
         image = tf.io.read_file(image_name)
         image = tf.image.decode_jpeg(image, channels=3)
         image = tf.image.resize(image, [224, 224])  # Redimension de imagen
         image = image / 255.0  # Normalizar entre 0 y 1 del valor de los pixeles
         return image, label

     train_dataset = tf.data.Dataset.from_tensor_slices((train_images, train_labels))
     train_dataset = train_dataset.map(lambda x, y: load_and_preprocess_image(x, y, train_images_dir))
     train_dataset = train_dataset.batch(32).shuffle(100)

     val_dataset = tf.data.Dataset.from_tensor_slices((val_images, val_labels))
     val_dataset = val_dataset.map(lambda x, y: load_and_preprocess_image(x, y, train_images_dir))
     val_dataset = val_dataset.batch(32)
```

Figura 1. Configuración inicial para la preparación de los datasets.

3- Modelos: Para este trabajo se utilizan dos modelos, uno personalizado y un modelo preentrenado llamado VGG16. El primero modelo se ha construido desde cero, basado de modelos preentrenados. Este modelo posee tres capas convolucionales de 32, 64 y 128 filtros con ventanas de 3x3. Después de cada

capa se aplica el pooling para reducir las dimensiones espaciales de las caracteríticas y hacer la red más eficiente. Posteriormente se aplica el Flatten para obtener vectores unidimensionales y así alimentar las capas que le siguen. Las capas densas se disponen de la siguiente forma:

- Una capa completamente conectada con 128 unidades y activación ReLU.
- Una capa final con 4 neuronas y activación softmax para realizar la clasificación en 4 clases.

Por último, el 'sparse_categorical_crossentropy' se selecciona ya que es adecuado para la clasificación multiclase cuando las etiquetas (clasificador de defectos para el caso particular) son enteros.

Por otro lado, el modelo VGG16 se usa debido a que es un modelo clásico muy utilizado en tareas de clasificación de imágenes gracias a su diseño el cual consiste principalmente en convoluciones 3x3 y max pooling 2x2 lo que permite captar patrones y características relevantes. También trabaja con imágenes de 224x224 lo que reduce el costo computacional al no tener que usar imágenes de mayor tamaño.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_2 (Conv2D)	(None, 52, 52, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 26, 26, 128)	0
flatten (Flatten)	(None, 86528)	0
dense (Dense)	(None, 128)	11,075,712
dense_1 (Dense)	(None, 4)	516

Figura 2. Resumen de parámetros para el modelo personalizado.

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14,714,688
flatten_1 (Flatten)	(None, 25088)	0
dense_2 (Dense)	(None, 256)	6,422,784
dropout (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 4)	1,028

Figura 3. Resumen de parámetros para el modelo VGG16.

4- Métricas: Las métricas utilizadas son el acuracy o precisión y el dice. La precisión se define como el porcentaje de predicciones correctas sobre el total de elementos predichos. Un valor alto es deseado cuando los falsos positivos son penalizados severamente o son indeseables en el proceso. Para la clasificación que se llevará a cabo, un valor superior a 0,75 se considera adecuado.

Por otro lado, el dice es comúnmente utilizado en problemas de segmentación de imágenes, y cuantifica qué tan bien se alinea la región predicha con la región verdadera en la imagen. Un coeficiente de Dice superior a 0,85 se considera generalmente aceptable. Esta métrica es ampliamente utilizada en la detección de defectos en superficies metálicas [1].

Iteraciones y resultados

Las iteraciones se realizaron de la siguiente forma:

- Primera iteración

Inicialmente se escogió una cantidad de épocas de 3 y un total de 100 imágenes. El modelo personalizado arrojó los siguientes resultados.

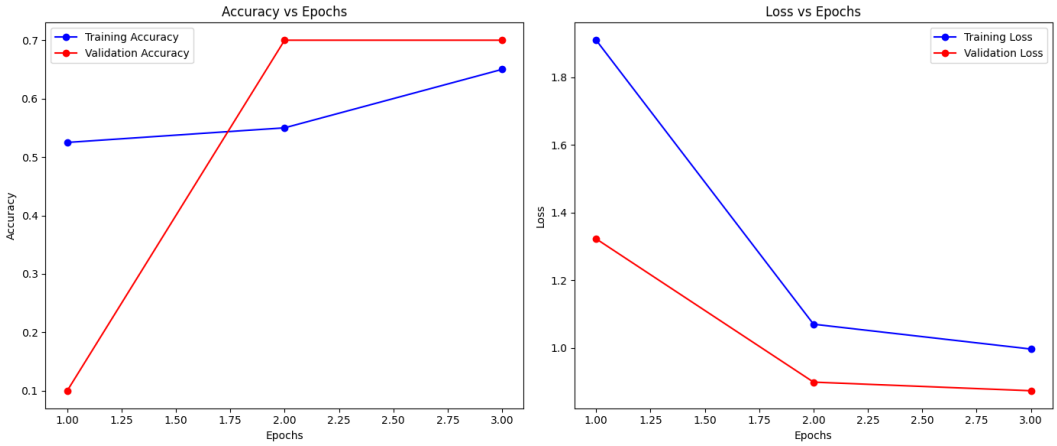

Figura 4. Evolución de la métrica de precisión en función de las épocas para el modelo personalizado con una configuración de 3 épocas y 100 imágenes.

Según estos resultados se observa un buen progreso en cuanto al entrenamiento y validación. Al ser tan pocas épocas no se podría decir que hay un sobre ajuste y se debería aumentar la cantidad de épocas para concluir al respecto. La siguiente imagen representa gráficamente cuales son las clases predichas y a las cuales se ha acertado (solo las que se encuentras en la diagonal principal)

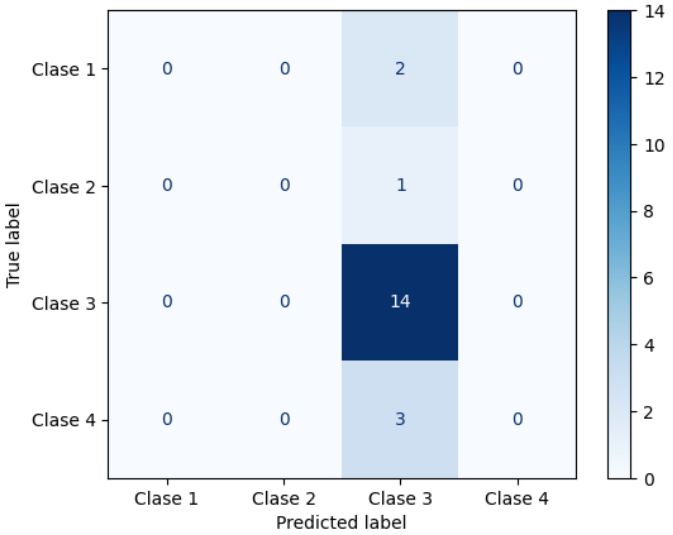

Figura 5. Matriz de confusión para el modelo personalizado bajo las condiciones de 3 epocas y 100 datos.

Al utilizar el modelo VGG16 se observa un comportamiento similar al anteriormente descrito. La matriz de confusión muestra que el modelo predice correctamente la misma cantidad del modelo personalizado, ya que se utilizan los mismos datos y la misma cantidad.

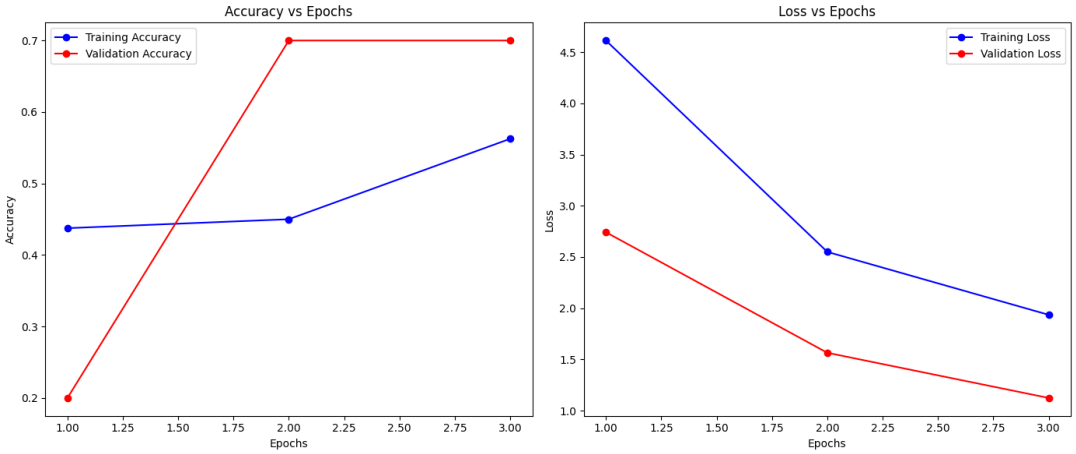

Figura 6. Evolución de la métrica de precisión en función de las épocas para el modelo personalizado con una configuración de 3 épocas y 100 imágenes.

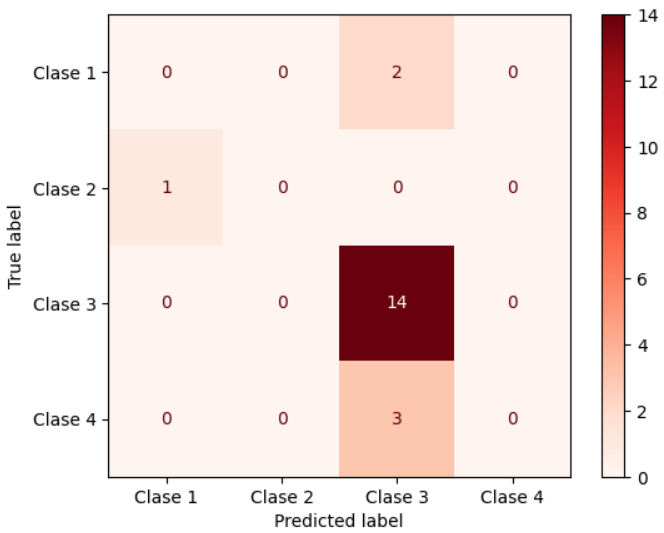

Figura 7. Matriz de confusión para el modelo VGG16 bajo las condiciones de 3 épocas y 100 datos.

El valor del dice es de 0.7 para ambos modelos. Esto quiere decir que la superposición entre el área predicha y el área real es razonablemente buena, pero no es perfecta. Esto se puede mejorar utilizando una mejor selección de los datos de entrada (en este caso las imágenes) y cambiando la arquitectura de red o el modelo pre-entrenado en su defecto.

- Segunda iteración

Para esta iteración se aumenta la cantidad de datos, seleccionando 400. Los resultados de precisión en ambos modelos no cambian con respecto a los valores obtenidos en la primera iteración. A partir de esto se puede evidenciar que para validación los valores no cambian, tal como lo muestra la figura. Esto lleva a un sobre aprendizaje, causando que la tendencia del modelo sea a clasificar las imágenes en un solo defecto, tal como lo muestran las matrices de confusión.

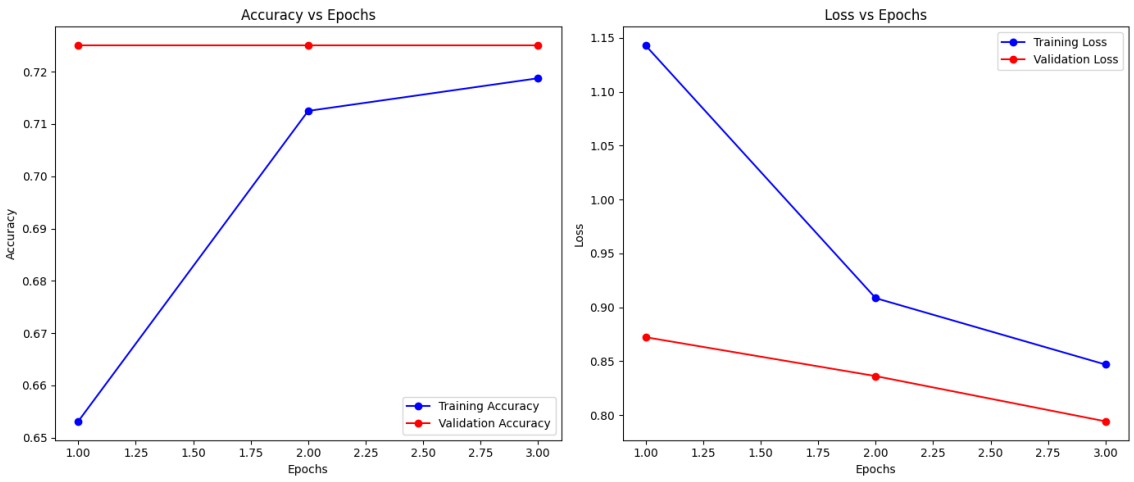

Figura 8. Evolución de la métrica de precisión en función de las épocas para el modelo personalizado con una configuración de 3 épocas y 400 imágenes.

Figura 9. Matriz de confusión para el modelo personalizado bajo las condiciones de 3 épocas y 400 datos.

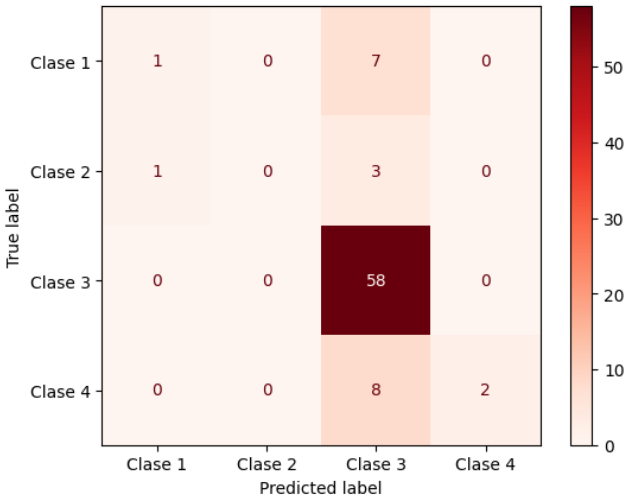

Figura 10. Matriz de confusión para el modelo VGG16 bajo las condiciones de 3 épocas y 400 datos.

- Tercera iteración:

En esta iteración se aumentó la cantidad de épocas a 5 dejando 100 datos (diferentes a los utilizados en las anteriores iteraciones. Nuevamente para el modelo personalizado se observa que la precisión no llega a un valor cercano de 0.7 (0.68 en este caso) y no evoluciona. Al ver las matrices de confusión se observa el mismo fenómeno, y pasa que como hay muchos datos de una sola clase, el modelo empieza a sobre entrenar lo que causa que clasifique otros defectos de forma incorrecta.

Sin embargo, para el modelo pre-entrenado se observa un leve aumento del valor de precisión a 0.75 y su matriz de confusión muestra que predice correctamente defectos en la clase 1 y 4, algo que no logró el modelo personalizado.

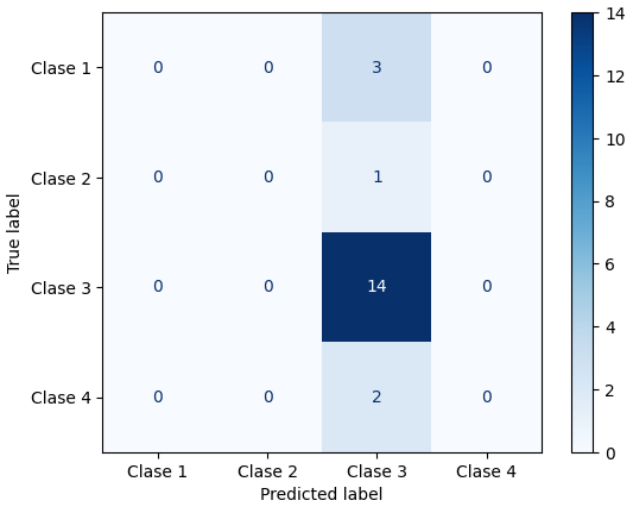

Figura 11. Matriz de confusión para el modelo personalizado bajo las condiciones de 5 épocas y 100 datos.

Figura 12. Matriz de confusión para el modelo VGG16 bajo las condiciones de 5 épocas y 100 datos.

El valor del coeficiente dice para ambos de 0.7, al igual que las iteraciones anteriores. Con base esta iteración, se puede concluir que el uso de un modelo pre-entrenado como el VGG16 y el aumento en las épocas, mejoran la métrica de evaluación de precisión, indicando una mayor cantidad de predicciones correctas.

Referencias

[1] R. Usamentiaga, D. G. Lema, O. D. Pedrayes, and D. F. Garcia, "Automated Surface Defect Detection in Metals: A Comparative Review of Object Detection and Semantic Segmentation Using Deep Learning," *IEEE Trans. Ind. Appl.*, vol. 58, no. 3, pp. 4203–4213, 2022, doi: 10.1109/TIA.2022.3151560.