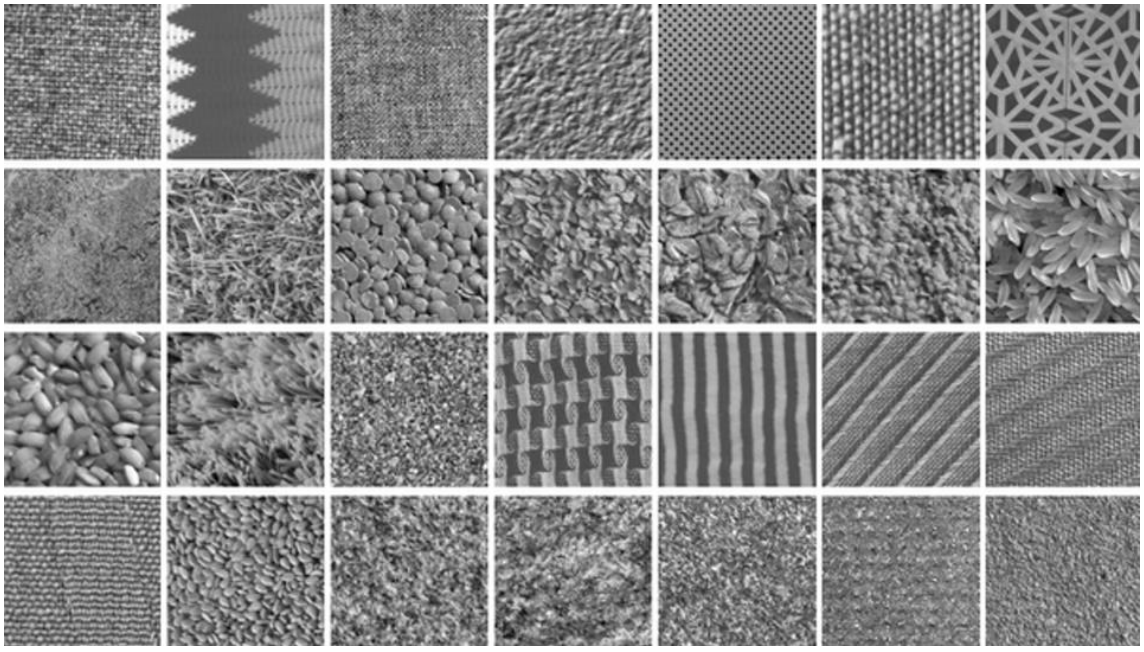
universidad de león

Máster en Robótica y Sistemas inteligentes

CLASIFICACIÓN DE TEXTURAS



Aitor García Blanco

Índice

1.Introducción

En este proyecto, se tiene como objeto diseñar e implementar diferentes sistemas, capaces de clasificar imágenes de texturas en seis categorías específicas.

Para lograr este objetivo, se emplean diferentes enfoques de aprendizaje automático y aprendizaje profundo. Los diferentes métodos, procesarán las imágenes directamente o partirán de un conjunto de características que se obtienen al aplicar un descriptor sobre la imagen.

Además, intentaremos obtener los mejores resultados empleando las configuraciones más efectivas y robustas mediante el análisis de diferentes métricas.

En el siguiente repositorio se encuentra el proyecto en Python para este proyecto:

ULE-Informatica-2024-2025/vico24-AigarciabFabero: vico24-AigarciabFabero created by GitHub Classroom

En esta memoria, se representan los resultados, se comentan y se proporciona una visión global del proyecto. Para un mayor detalle, se recomienda revisar el repositorio donde se detallan los pasos a seguir.

2.Descripción del dataset

El *dataset* que se emplea para la clasificación de texturas, se puede encontrar en el siguiente repositorio: pCloud - KylbergTextureDatasetV1.

En el conjunto de imágenes que nos proporciona el *dataset*, tenemos una distribución de 6 clases diferentes de texturas: lino, cojín, semillas de lino, arena, asiento y piedra. Para cada clase, disponemos de 40 imágenes con formato .png y en escala de grises; de dimensiones 576x576. Durante todo el procesamiento, las imágenes se redimensionan con un tamaño de 128x128.

3.Descriptores de la imagen

Para el desarrollo de esta práctica, se nos pide aplicar descriptores locales o globales sobre nuestro dataset; para obtener las características de las imágenes y poder así, aplicar algoritmos de machine learning de clasificación como pueden ser KNN o MSV.

Para los descriptores globales, las texturas que presentan patrones repetitivos claros y fuertes contrastes como pueden ser las telas, GLCM sería una buena opción. HOG, podría ser útil si las texturas presentan patrones de bordes y orientaciones predominantes (el problema a abordar), aunque presenta limitaciones para texturas irregulares como podría ser la piedra, la semilla de lino o la arena.

Por otro lado, tenemos los descriptores locales. Estos, son especialmente efectivos en escenarios de texturas complejas y heterogéneas. Detectan puntos clave o características distintivas a nivel local, lo que es beneficioso para capturar detalles finos y variaciones dentro de la textura. Puede ser especialmente útil en texturas como la arena, las semillas de lino o la piedra, pudiendo ser así, superior a un descriptor global.

Como los descriptores locales presentan una dificultad añadida y, atendiendo a las características de las imágenes, emplearemos durante este primer proceso, el modelo de descriptor global HOG.

Para conocer la configuración que se ha empleado en el descriptor, se recomienda seguir durante la lectura el repositorio; donde encontraremos con detalle las diferentes explicaciones.

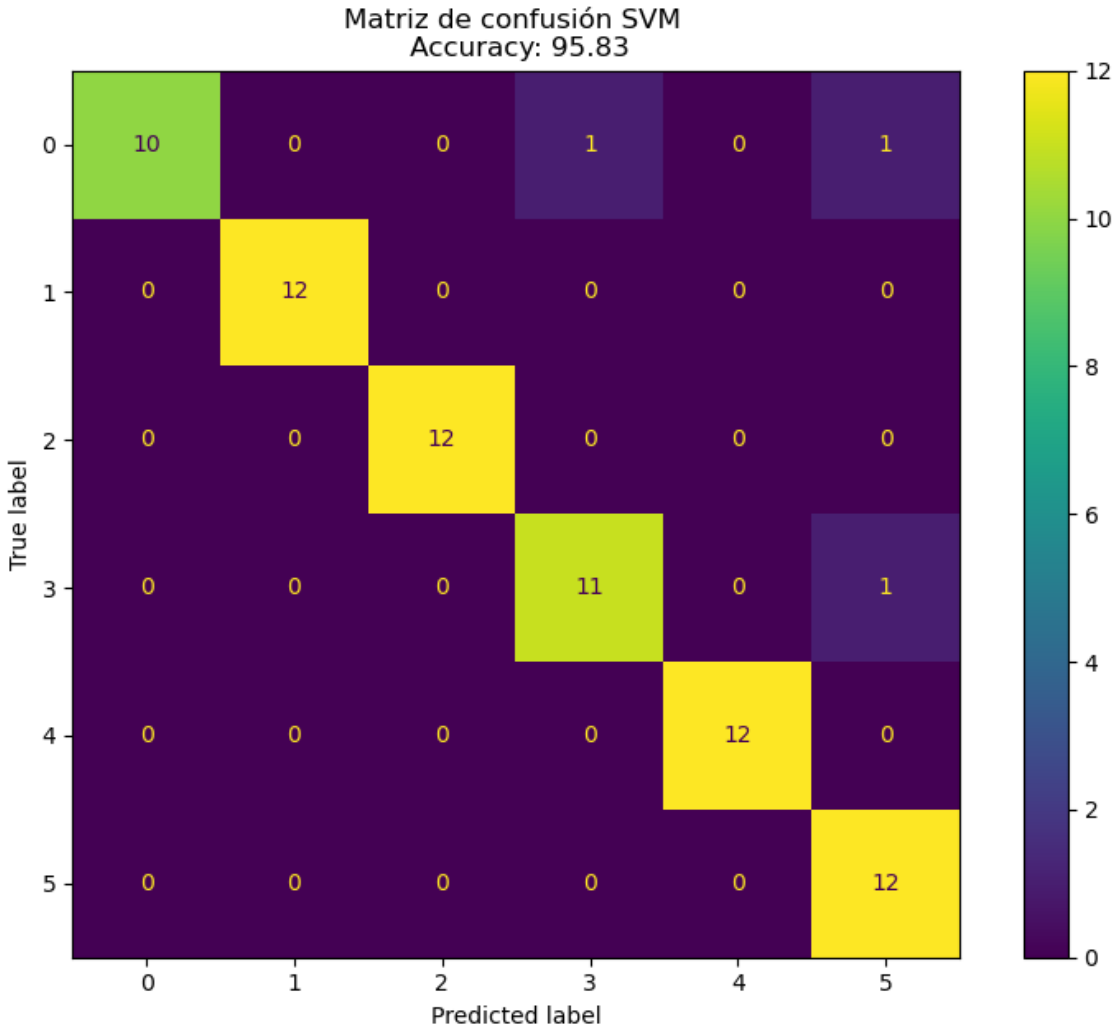
Recordemos que el conjunto de entrenamiento y test se ha dividido de forma homogénea con el método *StratifieldShuffleSplit* con un *Split* del 30%.

a.Machine Learning

En este apartado, se han probado dos métodos diferentes de aprendizaje supervisado, KNN y SVM. Como los resultados con KNN no eran tan buenos como los de SVM, nos quedamos con los resultados de este último.

La mejor configuración obtenida tras aplicar un *Grid* sobre el modelo SVM, nos arroja una precisión del (96 ± 3) %.

Los resultados obtenidos con la matriz de confusión son gratamente satisfactorios. Tal y como podemos apreciar en la siguiente imagen



b.Fully Connected Network

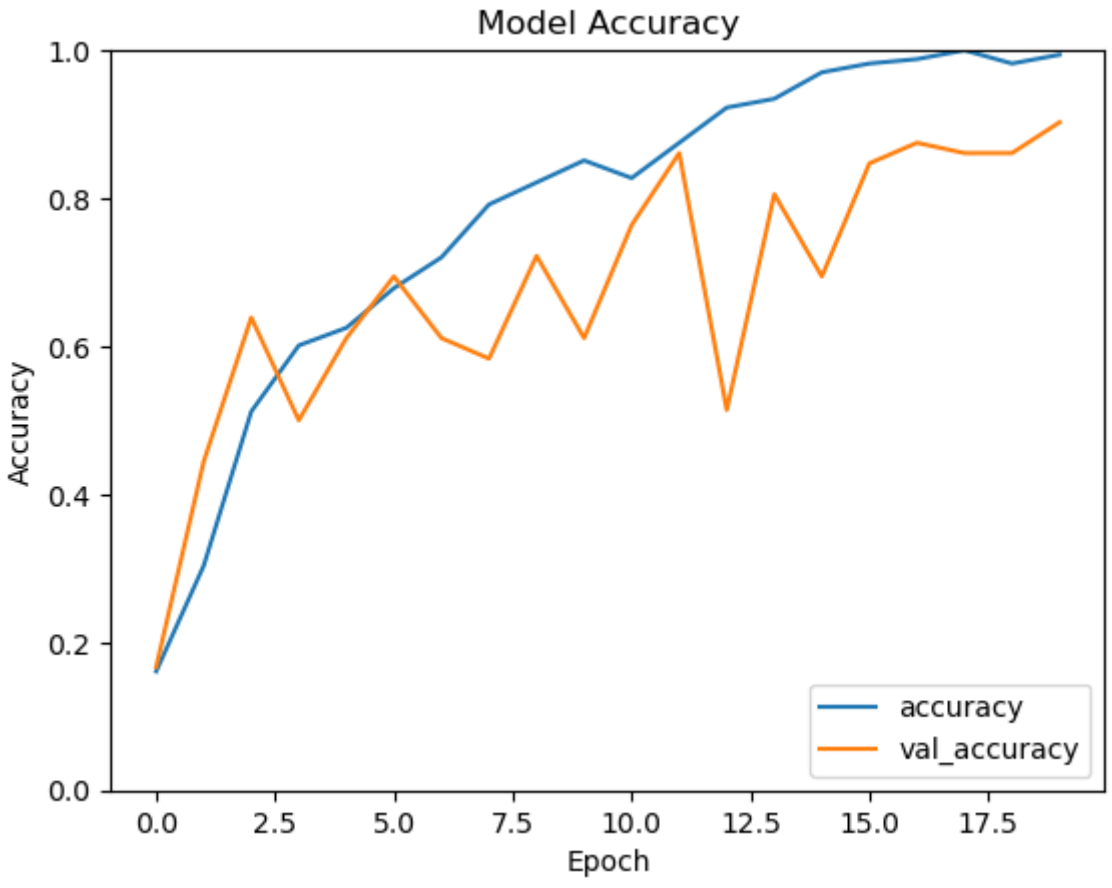
En este bloque, se van a emplear el conjunto de características que hemos obtenido de aplicar los descriptores sobre nuestro conjunto de imágenes para entrenar una red *fully connected network*.

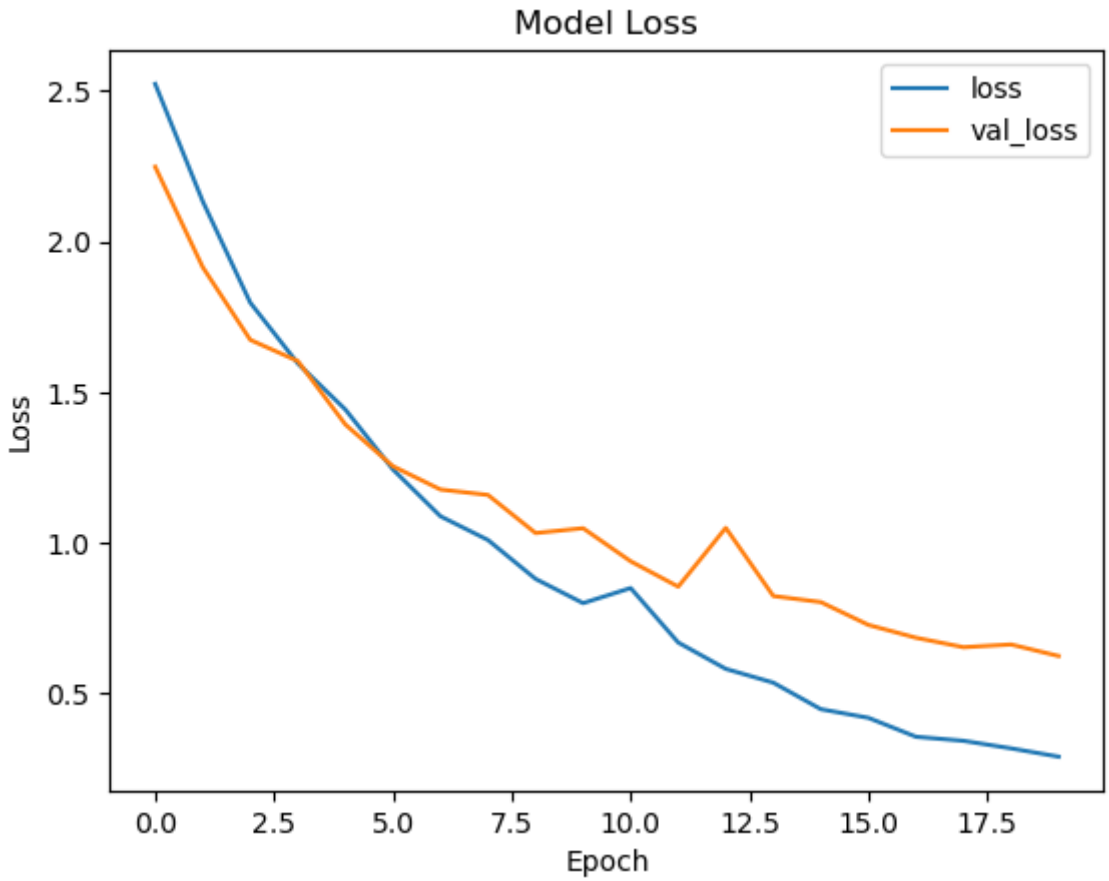
Inicializamos tanto el optimizado Adam como el SGD, pero nos quedamos con Adam puesto que nos arroja mejores resultados. Como función de pérdida, empleamos *categorical_crossentropy*, adecuado para problemas de multiclase y, se especifica el *accuracy* como métrica para evaluar el rendimiento del modelo

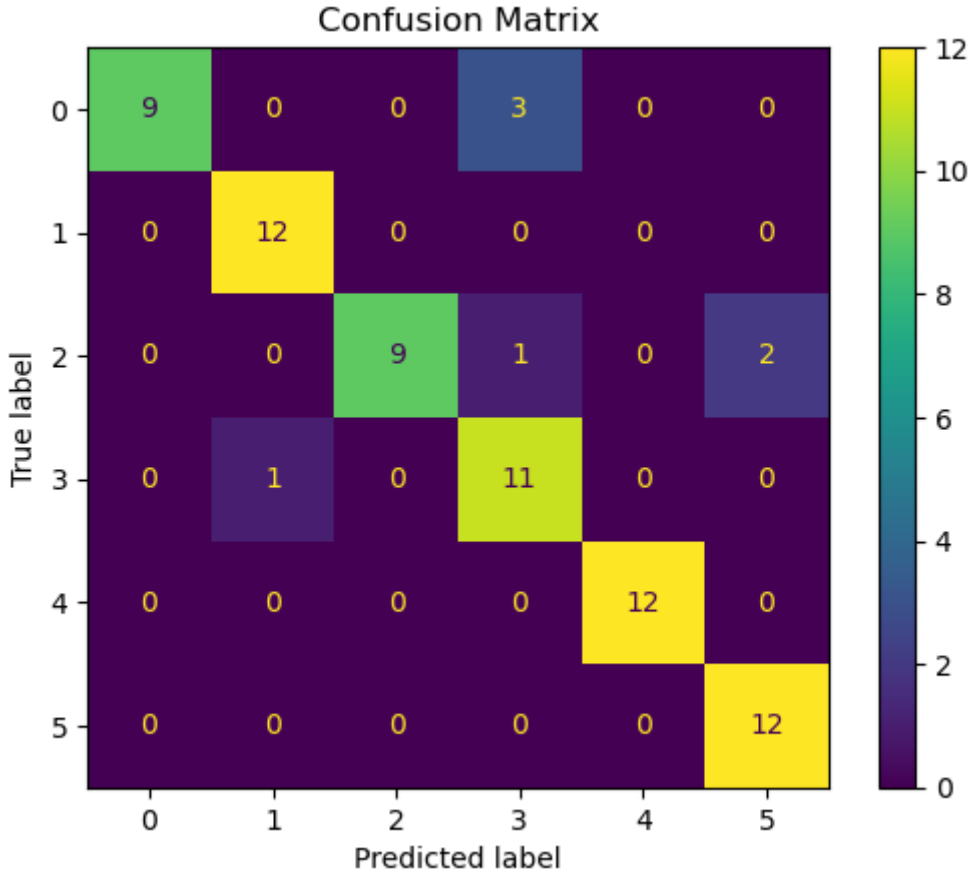
Este modelo, está compuesto por 6 capas densas, con funciones de activación relu. Cada capa densa presenta 256, 128, 128, 64, 32 y 6 neuronas. La capa final es una capa densa con 6 neuronas y activación *softmax* con el fin de realizar la clasificación multiclase. Finalmente, mostramos la estructura de nuestra red neuronal mediante la sentencia *model.summary()*.

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_22 (Dense)            (None, 256)               2304256

 dense_23 (Dense)            (None, 128)               32896

 dense_24 (Dense)            (None, 64)                8256

 dense_25 (Dense)            (None, 32)                2080

 dense_26 (Dense)            (None, 6)                 198

=================================================================
Total params: 2,347,686
Trainable params: 2,347,686
Non-trainable params: 0
```

Entrenamos nuestro modelo con 20 *epoch* y 20 características por *batch*. Obtenemos un *Accuracy* de 0.9028 y un *loss* de 0.6249




Model Loss
loss
val_loss
Loss
Epoch
2.5
2.0
1.5
1.0
0.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Confusion Matrix
True label
Predicted label
9 0 0 3 0 0
0 12 0 0 0 0
0 0 9 1 0 2
0 1 0 11 0 0
0 0 0 0 12 0
0 0 0 0 0 12
0 1 2 3 4 5
12 10 8 6 4 2 0

4.Imagen como vector

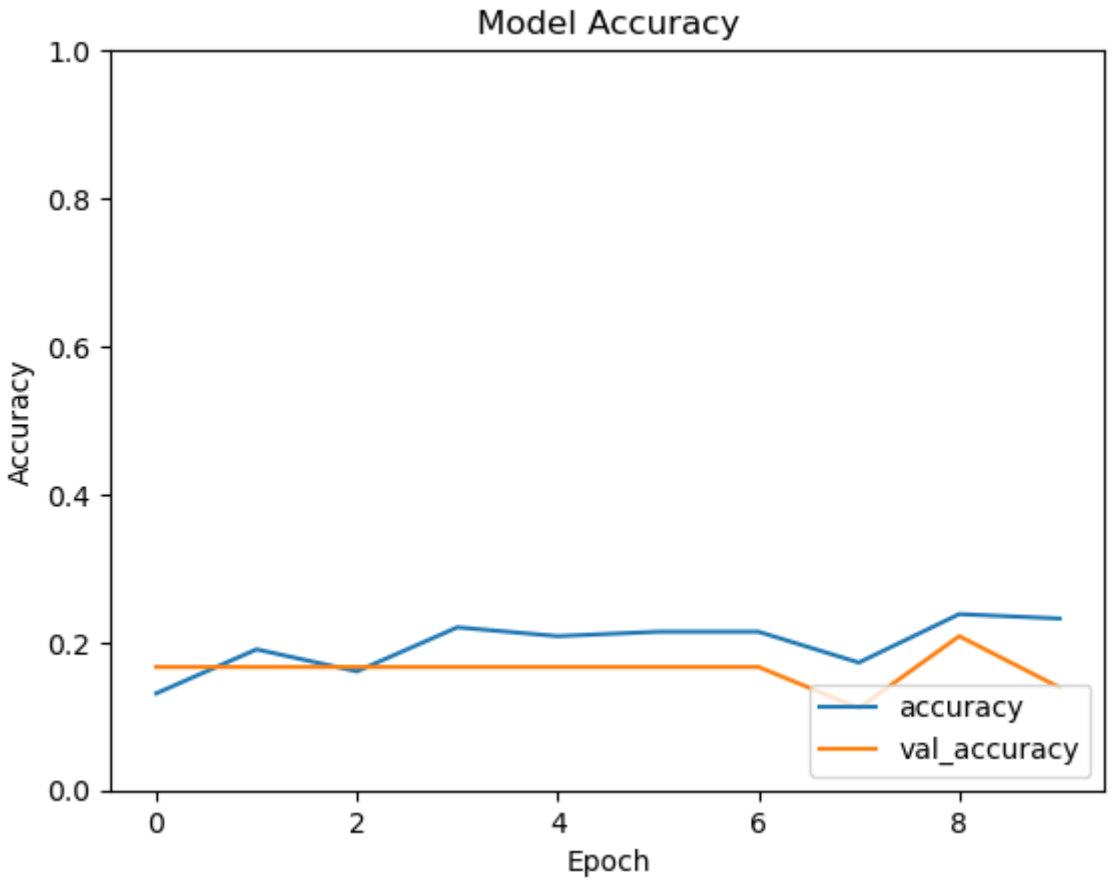
Tanto en este bloque como en el siguiente, ya no vamos a emplear las características obtenidas con el descriptor HOG, sino que, vamos a tener como entrada las imágenes.

Proponemos una red neuronal secuencial a la que llamaremos model2. Para procesar directamente las imágenes empleamos una primera capa *Flatten* que aplanará las entradas de tamaño (128,128) en un vector unidimensional. Luego, empleamos tres capas neuronales densas con 128 neuronas, función de activación *Relu* y un *kernel_regularizer*=l2(0.001) para intentar prevenir sobreajustes. Después de cada capa densa, se añade una capa de normalización por lotes (*BatchNormalization*) para acelerar el entrenamiento y mejorar la estabilidad, seguidas de una capa (*Dropout*) con una tasa de 0.5 para reducir de nuevo, la probabilidad de sobreajuste. Finalmente, el modelo incluye una capa de 6 neuronas y una función de activación *Softmax* para poder realizar la multiclasificación.

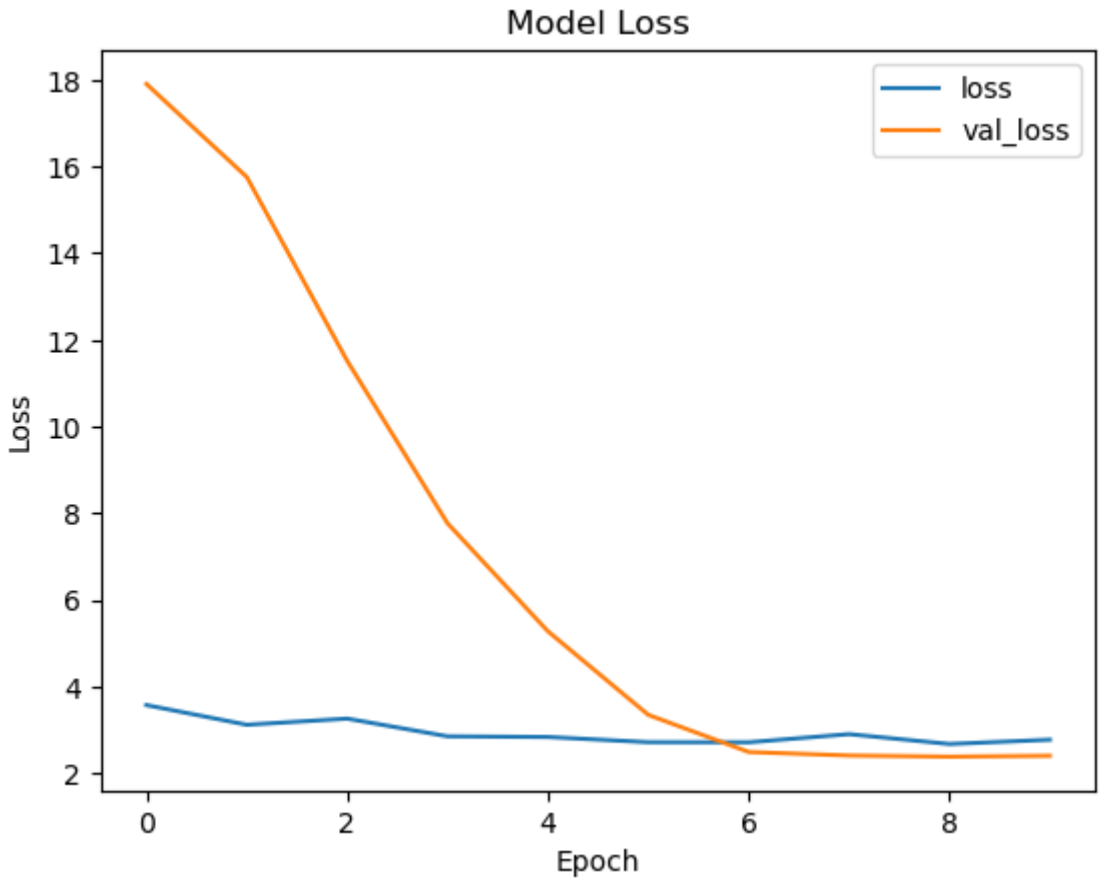
```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_4 (Flatten)         (None, 16384)             0

 dense_27 (Dense)            (None, 128)               2097280

 batch_normalization_6 (Batc  (None, 128)              512
 hNormalization)

 dropout_6 (Dropout)         (None, 128)               0

 dense_28 (Dense)            (None, 128)               16512

 batch_normalization_7 (Batc  (None, 128)              512
 hNormalization)

 dropout_7 (Dropout)         (None, 128)               0

 dense_29 (Dense)            (None, 64)                8256

 batch_normalization_8 (Batc  (None, 64)               256
 hNormalization)

...
Total params: 2,123,718
Trainable params: 2,123,078
Non-trainable params: 640
```

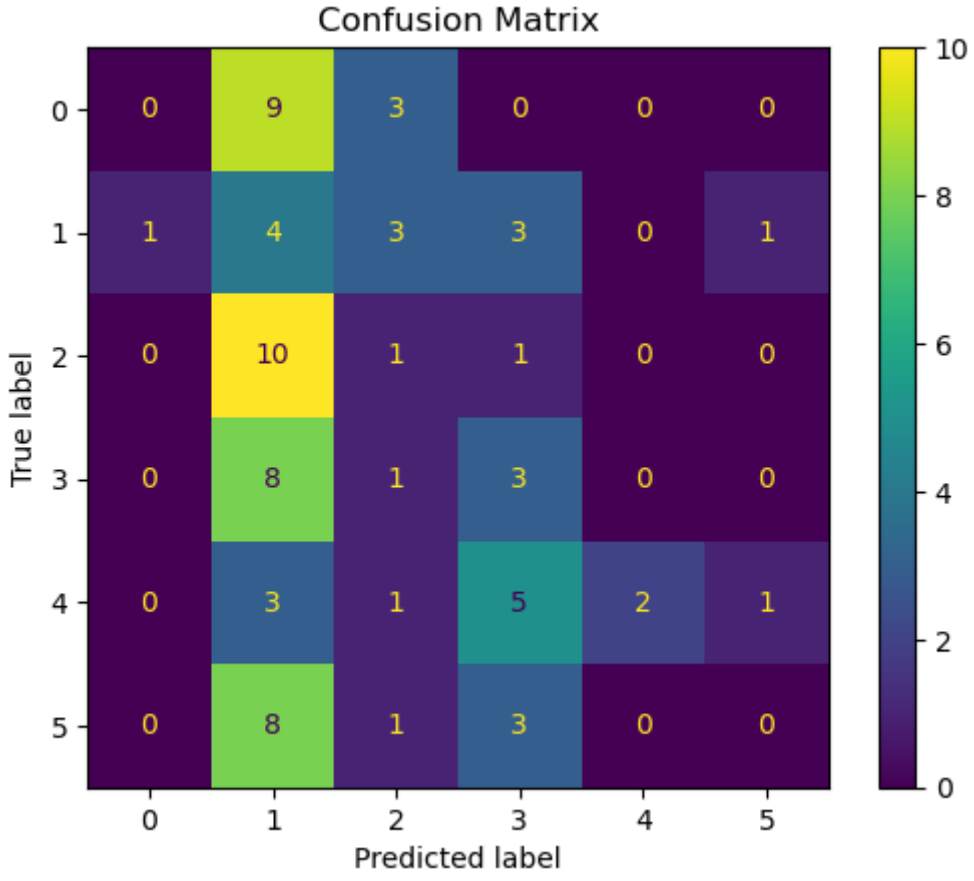
En el siguiente fragmento de código, establecemos el optimizador que se va a emplear al entrenar el modelo. Inicializamos tanto el optimizado Adam como el SGD, pero nos quedamos con Adam puesto que nos arroja mejores resultados. Como función de pérdida empleamos *categorical_crossentropy*, adecuado para problemas de multiclase y, se especifica el *accuracy* como métrica para evaluar el rendimiento del modelo.

Se entrena el modelo con 15 *epoch* y un tamaño de *batch* de 20.



Este es un claro ejemplo de *underfitting*. El modelo, no es capaz de ajustarse correctamente al conjunto de datos de entrenamiento ni de validación.


Model Loss
loss
val_loss
Loss
Epoch
18
16
14
12
10
8
6
4
2
0
2
4
6
8


Confusion Matrix
True label
Predicted label
0 9 3 0 0 0
1 4 3 3 0 1
0 10 1 1 0 0
0 8 1 3 0 0
0 3 1 5 2 1
0 8 1 3 0 0
0 1 2 3 4 5
10
8
6
4
2
0

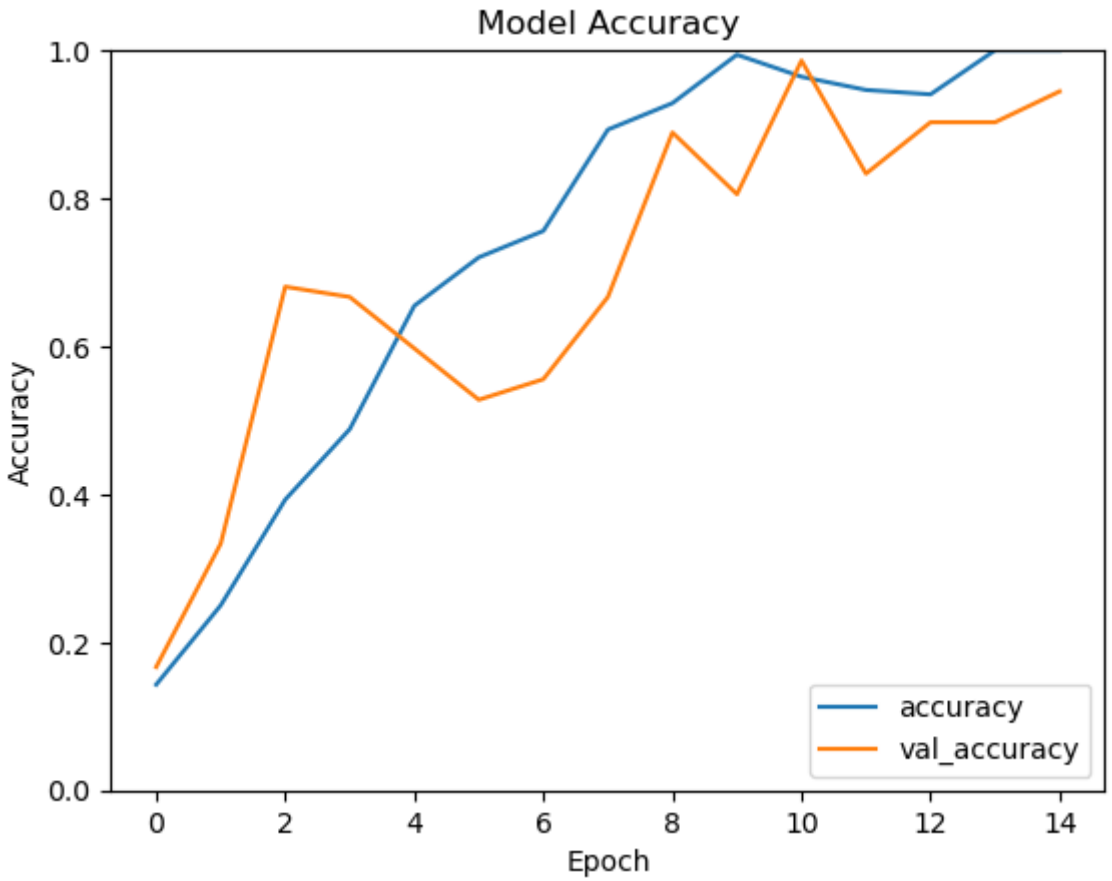
5.CNN

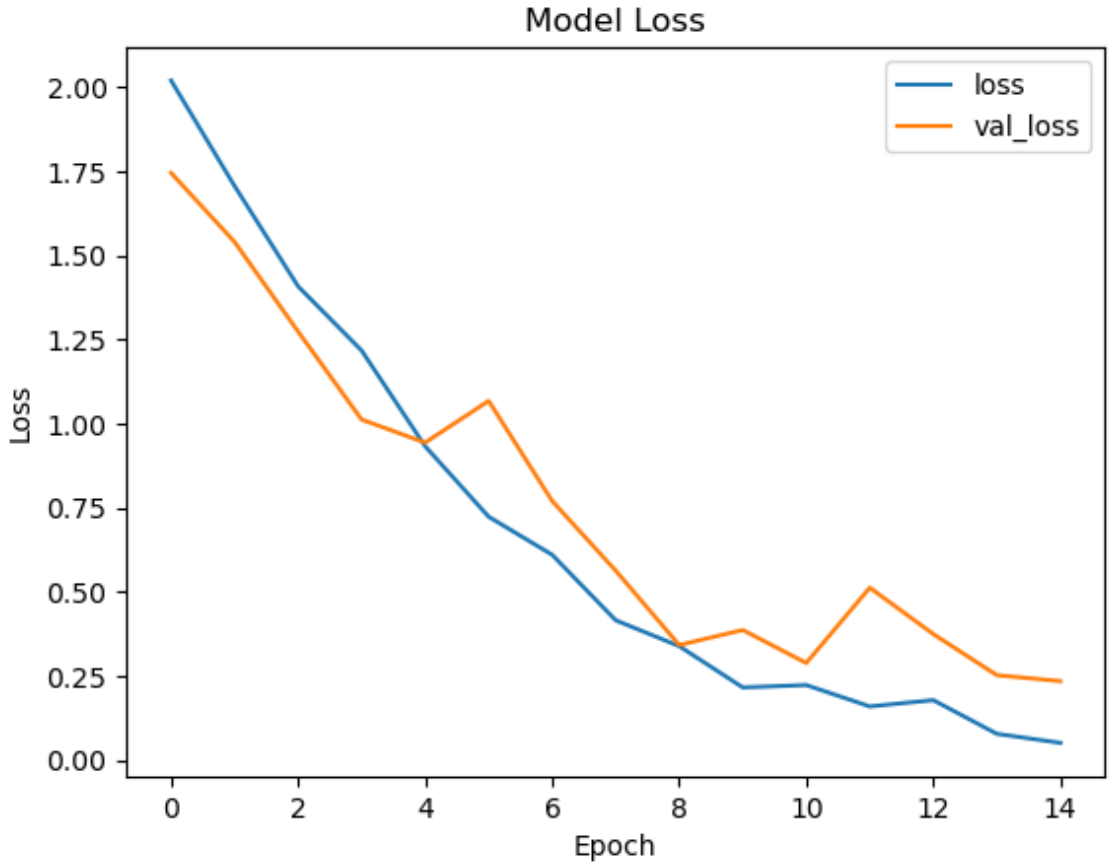
En este punto, vamos a utilizar redes neuronales convolucionales para clasificar. Recordemos que la entrada son las imágenes.

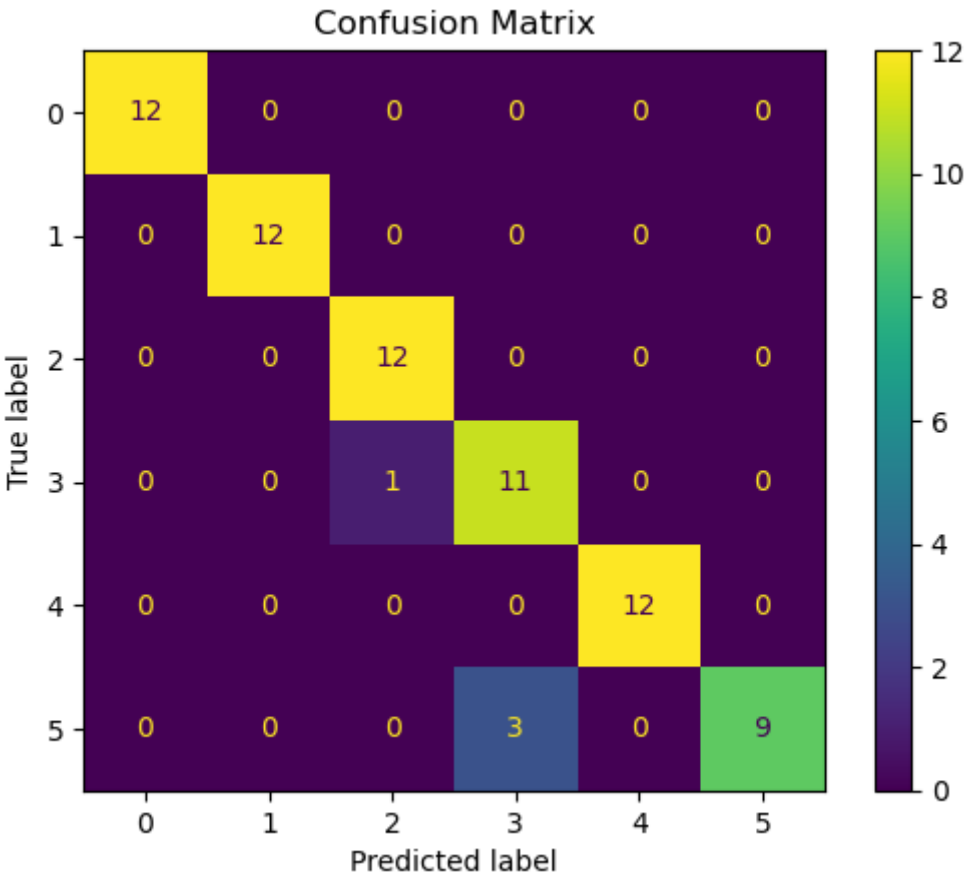
```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 128, 128, 32)      320

 max_pooling2d_4 (MaxPooling  (None, 64, 64, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 64, 64, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 32, 32, 64)       0
 2D)

 flatten_5 (Flatten)         (None, 65536)             0

 dense_31 (Dense)            (None, 64)                4194368

 dense_32 (Dense)            (None, 6)                 390

=================================================================
Total params: 4,213,574
Trainable params: 4,213,574
Non-trainable params: 0
```

Para el proceso de entrenamiento seleccionamos 15 *epoch* y el tamaño del *batch* 7, proporcional al número de imágenes de entrenamiento de cada clase.

Nuestro modelo tiene un *accuracy* de 0.9444 y un *loss* de 0.2350. Sin embargo, el máximo se obtiene con un valor de *accuracy* de 0.9861 y *loss* 0.2889.


Model Accuracy
Accuracy
Epoch
accuracy
val_accuracy
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
12
14


Model Loss
Loss
Epoch
loss
val_loss
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
2
4
6
8
10
12
14


Confusion Matrix
True label
Predicted label
0
1
2
3
4
5
12 0 0 0 0 0
0 12 0 0 0 0
0 0 12 0 0 0
0 0 1 11 0 0
0 0 0 0 12 0
0 0 0 3 0 9
12
10
8
6
4
2
0

6.Conclusión

Para concluir el proyecto de clasificación de texturas, implementamos y evaluamos diversos enfoques de aprendizaje automático y profundo; con resultados satisfactorios. Entre las estrategias principales, el modelo SVM destacó en términos de precisión, alcanzando un 96 %. Las redes neuronales convolucionales (CNN) también mostraron rendimientos prometedores al procesar directamente las imágenes, debido a su capacidad para extraer automáticamente características relevantes.

En contraste, al utilizar redes completamente conectadas (fully connected networks), solo obtuvimos resultados satisfactorios cuando empleamos el descriptor HOG para extraer características que luego alimentamos a la red. Cuando utilizamos las imágenes en bruto como entrada, el rendimiento no superó el 25% de precisión. Este resultado, subraya la importancia crucial de la extracción de características en el procesamiento de imágenes para redes neuronales. Las CNN abordan este desafío mediante la integración de capas convolucionales que capturan eficientemente patrones espaciales y texturales de las imágenes.

Es importante destacar que, en todas las situaciones donde obtuvimos resultados satisfactorios, se evidenció cierto grado de sobreajuste. Esto indica que los modelos están ajustándose demasiado a los datos de entrenamiento, comprometiendo su capacidad de generalización a nuevos conjuntos de datos. Por el contrario, en el caso no satisfactorio, la red no logró entrenar ni predecir eficazmente, lo que sugiere una falta de capacidad para aprender patrones significativos sin una etapa previa de extracción de características.

En resumen, la extracción efectiva de características es esencial para el éxito en la clasificación de texturas. Las arquitecturas que incorporan esta etapa, ya sea mediante descriptores como HOG o a través de capas convolucionales en las CNN, ofrecen un rendimiento superior. Sin embargo, es fundamental abordar el sobreajuste para garantizar que los modelos desarrollados sean robustos y generalicen adecuadamente a nuevos datos.