Clasificación de monedas

Iñigo Gastón

ÍNDICE

Urls:

Zip Proyecto General (imágenes, notebooks...)

Video

Documento en google drive (este mismo documento)

VISIÓN GENERAL

En este trabajo se han utilizado diversos modelos pre-entrenados para la clasificación de imágenes de monedas.

Aquí se pretende explicar algunas de las decisiones tomadas a lo largo de la realización del proyecto.

ORGANIZACIÓN GENERAL

```
.
├── dataPreprocessing.ipynb
├── ensemble.ipynb
├── gimnasio.ipynb
├── img
│   ├── euro-coin-dataset-master
│   ├── mixed
│   ├── test
│   ├── testPreprocesado
│   ├── train
│   └── trainPreprocesado
└── models
```

El código está dividido en 3 notebooks, *dataPreprocessing.ipynb*, *gimnasio.ipynb* y *ensemble.ipynb*.

Todas las imágenes usadas están en la carpeta *img*. Los subdirectorios *mixed*, *test* y *train* ya son conocidos. *euro-coin-dataset-master* es una copia de un repositorio de github. *testPreprocesado* contiene las imágenes de *test* modificadas. *trainPreprocesado* contiene tanto las imágenes de *train* como de *euro-coin-dataset-master* modificadas. Se han guardado tanto las imágenes "originales", sin modificar, como las preprocesadas para ahorrar tiempo y poder exportarlas a kaggle fácilmente.

La carpeta *models* contiene los "modelos" obtenidos. Son archivos con los pesos de las distintas redes neuronales entrenadas.

ANÁLISIS DEL DATASET

Distribución de clases

Nuestro conjunto de imágenes *train* tiene la siguiente distribución:

Clase	Número de imágenes	Porcentaje
5c	284	0.218967
10c	254	0.195837
1e	248	0.19121
20c	227	0.17501
50c	131	0.101002
1c	80	0.061681
2e	48	0.037008
2c	25	0.019275

Número total de imágenes (después del preprocesamiento): 1291

Idealmente todas las clases estarían igualmente representadas, teniendo unas 160 imágenes de cada clase (un 12.5%). No tener esto puede suponer un problema. Al haber tan pocos ejemplos de algunas clases la precisión de estas puede verse muy afectada.

PREPROCESAMIENTO

Corrección del ruido

Para reducir el ruido de las imágenes se ha usado un filtro de la mediana con un tamaño de ventana 5x5. Pero este solo ha modificado aquellos píxeles por encima o por debajo de unos umbrales determinados. Para ello se guarda una copia de la imágen aplicando el filtro. Se pasa la imágen a escala de grises. Aquellos píxeles mayores a 230 o menos que 20 son sustituidos en la imágen original por los píxeles de la imágen filtrada.

Este procedimiento, algo más enrevesado y lento, consigue corregir el ruido impulsivo distorsionando únicamente aquellas partes susceptibles de ser ruido. La calidad final suele ser mejor que aplicar el filtro a toda la imágen.

Outliers

Otra parte del preprocesamiento consiste en filtrar imágenes con todos los píxeles en blanco o en negro.

Otras técnicas

Se ha ampliado el rango dinámico y equalizado la imagen. Ambas funciones pertenecen a la librería de skimage.

ENTRENAMIENTO DE MODELOS

Se han usado 8 modelos, definidos y pre entrenados, disponibles en la librería tensorflow.

Transfer Learning

La idea general del transfer learning es que al entrenar un modelo con un dataset lo suficientemente grande este modelo puede ser usado como un modelo genérico.

Todos los modelos usados han sido previamente entrenados en un problema de clasificación de imágenes (Imagenet). Este consta de más de un millón de imágenes de entrenamiento para un problema de clasificación con 1000 clases distintas (coche, persona, gato...). Podemos usar estos modelos para extraer las características más relevantes de las imágenes para luego clasificarlas. Esto es más sencillo que entrenar un modelo desde cero, el cual requiere de muchas imágenes y unas arquitecturas difíciles de encontrar.

Descripción general del entrenamiento

En el notebook *gimnasio.ipynb* es donde se han obtenido todos los "pesos" guardados en la carpeta *models*.

El funcionamiento es el siguiente. La variable MODEL define qué modelo entre los posibles candidatos se va a usar. A ese modelo pre entrenado que hemos escogido se le añaden varias capas de "preprocesamiento". La primera (resnet_v2.preprocess_input, aunque ponga resnet es igual para el resto de modelos) modifica el rango de la imágen. Convierte valores de la escala 0, 255 al rango -1, 1. Esto es así ya que todos los modelos fueron entrenados con imágenes en este rango. La segunda sería el data augmentation que se comentará más adelante.

Después de estas capas vendría el modelo pre entrenado. Por último se añaden 8 neuronas correspondientes a las 8 clases que estamos clasificando.

El proceso de entrenamiento comienza actualizando exclusivamente las 8 neuronas finales. Esto es así ya que es la única parte de la red que no ha sido entrenada en ningún momento. A continuación se entrena todo el modelo durante 70 épocas (valor por defecto, se puede modificar pero en este puto todos los modelos ya han convergido). Este proceso se repite para los siguientes modelos, ResNet50V2, ResNet101V2, ResNet152V2, DenseNet121, DenseNet169, DenseNet201, VGG16, VGG19.

Data Augmentation

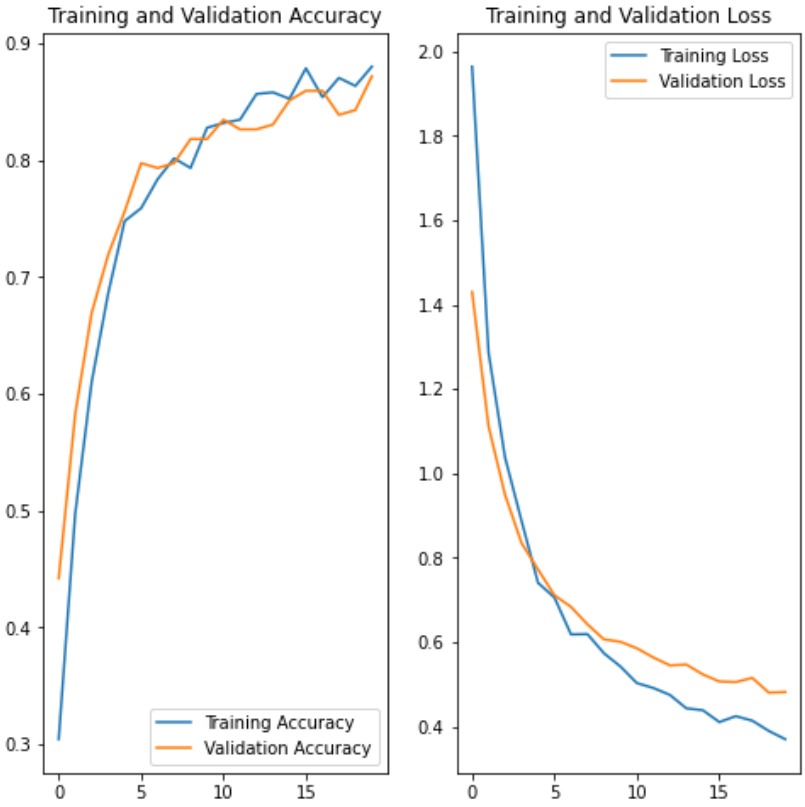
Un procedimiento muy habitual al entrenar redes neuronales convolucionales es el uso del data augmentation. Este consiste en modificar ligeramente las imágenes para ayudar al modelo a generalizar mejor. Aquí se han usado un zoom y una rotación aleatoria sobre la imágen.

Imágen original

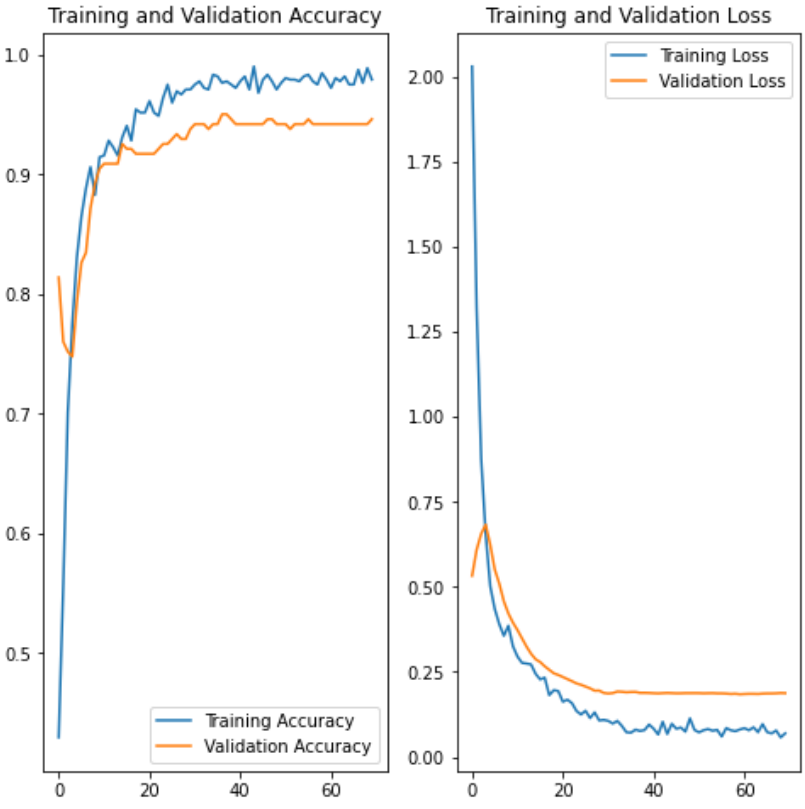
9 variaciones sobre la imágen original

Overfitting o no overfitting, esa es la cuestión

El primer modelo entrenado fue el ResNet101V2. Las curvas de aprendizaje obtenidas son las siguientes.



Entrenamiento de la última capa (8 neuronas) durante 20 épocas. A la izquierda precisión, a la derecha función de coste.



Entrenamiento de toda la red (después de haber entrenado la última capa) durante 70 épocas. A la izquierda precisión, a la derecha función de coste.

Al evaluar el modelo sobre el dataset de validación obtenemos una precisión del 94.21%.

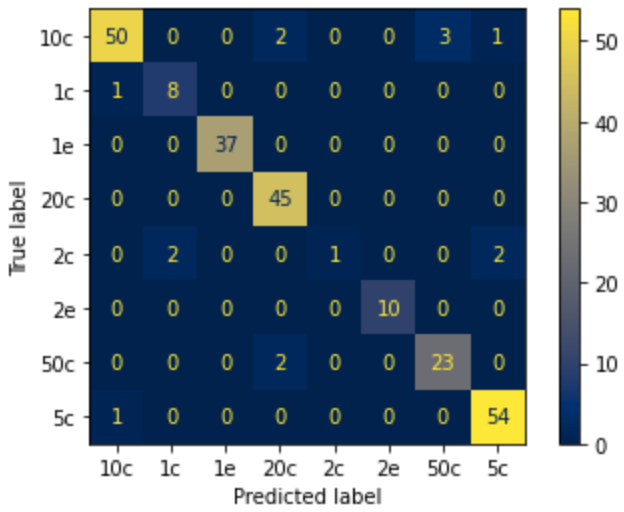
Las estadísticas por clase quedarían tal que:

clase	precisión	recall	f score	número de imágenes
10c	0.96	0.89	0.93	56
1c	0.8	0.89	0.84	9
1e	1	1	1	37
20c	0.92	1	0.96	45
2c	1	0.2	0.33	5
2e	1	1	1	10
50c	0.88	0.92	0.9	25
5c	0.95	0.98	0.96	55

Las estadísticas generales serían:

métrica	precisión	recall	f score	número de imágenes
precisión	-	-	0.94	242
media macro	0.94	0.86	0.87	242
media ponderada	0.94	0.94	0.94	242

La matriz de confusión es:



Se puede observar como las clases menos representadas (2c y 1c) son las que peor funcionan. Pero en general estamos entre un 0.87 y un 0.94 de f score (nada mal). Si predecimos las imágenes de test y las evaluamos en kaggle obtenemos un 71.688% de precisión. Es bastante menos de lo que se esperaría. Puede ser porque el dataset de test tiene muchas imágenes de las clases “minoritarias”.

Si repetimos lo anterior con cross validation o varias iteraciones los resultados son similares, luego no es problema de la muestra que hemos tomado.

Es improbable que estemos sobre entrenando, las gráficas de entrenamiento al menos no lo indican. Aunque quizás al tener un dataset desbalanceado estas no sean lo suficientemente representativas.

Para solucionar este problema se ha decidido usar más imágenes de las originalmente dadas. Esta parte de extracción de imágenes está en el notebook *dataPreprocessing.ipynb* y se comentarán algunos aspectos a continuación.

NUEVAS IMÁGENES

Las nuevas imágenes proceden del siguiente repositorio https://github.com/SuperDiodo/euro-coin-dataset.

La ventaja de este nuevo dataset es que contiene imágenes de monedas mostrando tanto la cara como la cruz. El problema es que las imágenes no muestran una moneda únicamente, luego habrá que segmentar.

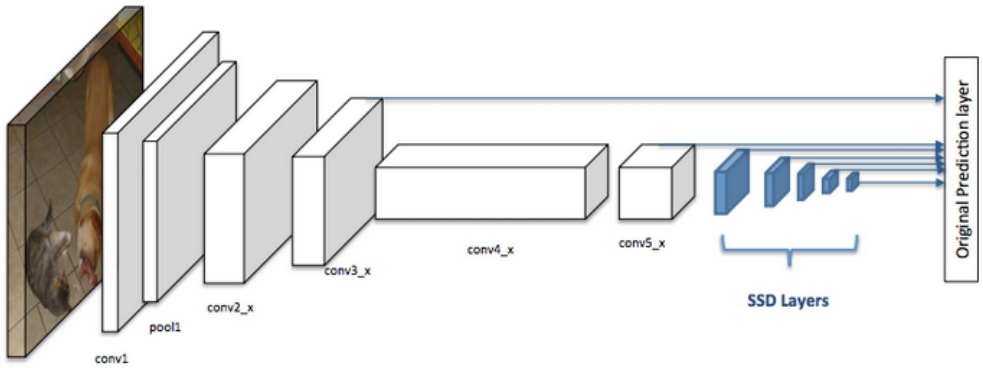
Ejemplos de imágenes del nuevo dataset

Single Shot Detector

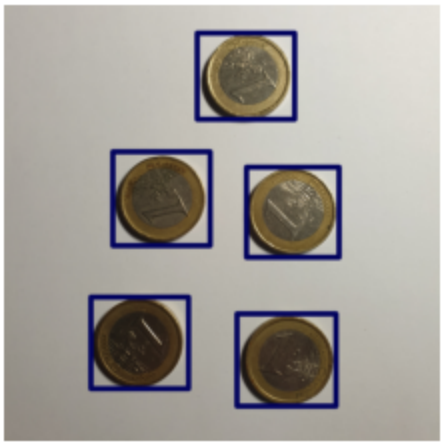
La idea es usar un modelo de detección de objetos para localizar y segmentar las diferentes monedas que haya en las imágenes. Se ha usado el siguiente modelo:

https://tfhub.dev/google/openimages_v4/ssd/mobilenet_v2/1

Corresponde a un mobilenet con algunas capas más que le permiten detectar objetos.



Arquitectura del modelo, en blanco las capas de mobilenet, en azul las del SSD.



Resultados de aplicar el modelo sobre las imágenes anteriormente mostradas.

Como se ve la segmentación no siempre es perfecta. Esto se debe a que solo se están segmentando aquellas monedas que el modelo está “muy seguro” de que son monedas. Esto se hace para evitar outliers. Es decir, si se segmenta todo lo que el modelo detecta como moneda pueden llegar a producirse errores.

Descripción del Dataset final

Después de añadir las nuevas imágenes el dataset tiene 2671 imágenes.

Clase	Número de imágenes	Porcentaje
5c	543	0.203295
10c	493	0.184575
1e	360	0.134781
20c	347	0.129914
50c	266	0.099588
1c	251	0.093972
2c	223	0.083489
2e	188	0.070386

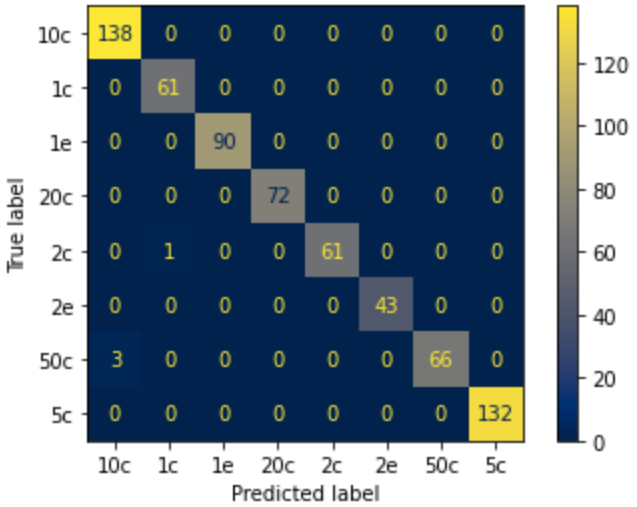
El dataset ahora está algo más balanceado.

ENSEMBLE

Después de entrenar los 8 modelos (con todas las imágenes) podemos sumar las predicciones para hacerlas más precisas.

Cada modelo o red neuronal produce un vector de 8 elementos. Cada posición del vector corresponde a la probabilidad de que una imágen sea de dicha clase. Por tanto para cada imágen se generan los vectores, se suman y se clasifica como la posición del mayor valor.

Los resultados obtenidos sobre 667 imágenes de validación son los siguientes.

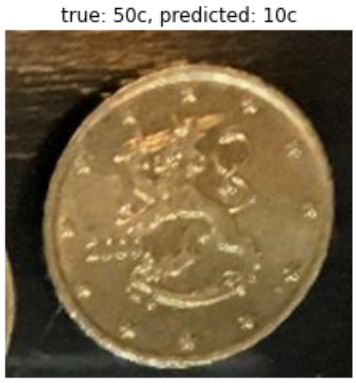


Matriz de confusión del ensemble

clase	precisión	recall	f score	número de imágenes
10c	0.98	1.00	0.99	138
1c	0.98	1.00	0.99	61
1e	1.00	1.00	1.00	90
20c	1.00	1.00	1.00	72
2c	1.00	0.98	0.99	62
2e	1.00	1.00	1.00	43
50c	1.00	0.96	0.98	69
5c	1.00	1.00	1.00	132

Las estadísticas generales serían:

métrica	precisión	recall	f score	número de imágenes
precisión	-	-	0.99	667
media macro	1.00	0.99	0.99	667
media ponderada	0.99	0.99	0.99	667



Imágenes de validación que falla el ensemble

La imagen de 2c que se está fallando es un outlier producido por el SSD comentado previamente.

La precisión del conjunto de test evaluado en kaggle es del 95%.

Nota sobre los modelos

Las gráficas y resultados comentados en este apartado son para un modelo ResNet101V2 entrenado con el dataset reducido u original (1291 imágenes). No se usa en el ensemble final.

En este documento están las gráficas de entrenamiento y resultados de todos los modelos usados (ResNets, VGGs, DenseNets...). Entrenados con 2004 imágenes y validados con 667.

BIBLIOGRAFÍA

Procesamiento de Imágenes con Python y Scikit-Image

Image classification | TensorFlow Core

Transfer learning and fine-tuning | TensorFlow Core

Building powerful image classification models using very little data

Transfer Learning using Mobilenet and Keras | by Ferhat Culfaz | Towards Data Science

Transfer Learning with VGG16 and Keras | by Gabriel Cassimiro | Towards Data Science

Hands-on Transfer Learning with Keras and the VGG16 Model – LearnDataSci

How single-shot detector (SSD) works? | ArcGIS Developer imágen de la arquitectura del SSD.