Memoria práctica 1

Ana Márquez Moncada

El objetivo es implementar una red neuronal que clasifique imágenes según su categoría.
Se parte de un dataset de imágenes de 32 × 32 píxeles de objetos de 5 categorías diferentes. Los datos están encapsulados en una estructura de tipo diccionario y almacenados en formato Pickle.
Tras visualizar las primeras imágenes del dataset, se pueden observar que son imágenes en color (3 canales) de emojis que se pueden clasificar en las siguientes categorías:

- **0:** Transportes aéreos
- **1:** Aves
- **2:** Transporte acuático
- **3:** Personas
- **4:** Gatitos y otros felinos

Lo primero que he hecho es dividir el dataset en un conjunto de entrenamiento y otro de validación. Tras varias pruebas, finalmente he escogido 800 muestras de entrenamiento y 205 de validación. El batch size final es de 10, es el que finalmente me ha dado mejores resultados.

Arquitectura de red:

La arquitectura que he propuesto es una red convolucional con 3 capas ocultas. La primera es una convolución, luego una pooling y por último otra convolucional.
Las capas convolucionales extraen las características de una zona.
Parámetros:

- Canales de entrada.
- Número de filtros (hay un canal de salida por cada filtro).
- Tamaño del filtro
- Stride: Cuánto desliza el filtro (por ejemplo stride =1 desliza un pixel el filtro entre cada convolución).
- Padding (relleno): Se agrega un píxel adicional en los bordes de la imagen, se rellenan con 0. Esto es muy útil para evitar que se reduzca la dimensión al convolucionar.

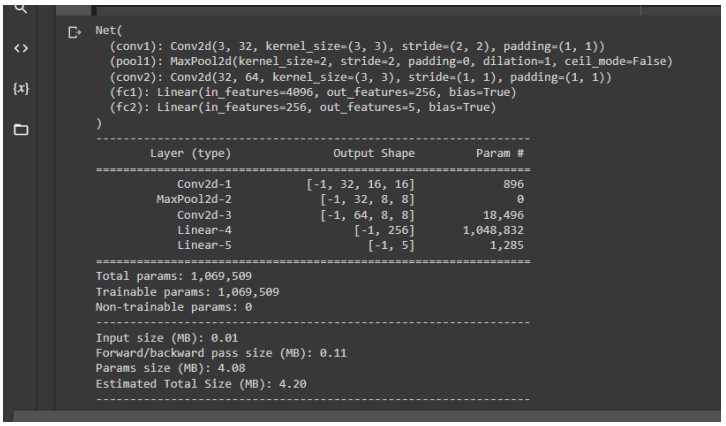
La capa MaxPool2d selecciona la zona de activación más fuerte.
Capa con el objetivo de condensar información, quedarse con los máximos. Sus parámetros son:

- Tamaño de filtro.
- Stride (Cuánto desliza el filtro).

La primera capa tiene 3 canales de entrada (RGB), 32 filtros con tamaño 3x3, stride de 2 y padding activo. La segunda (MaxPool2D) usa un filtro 2x2. La última tiene los 32 canales de entrada y 64 filtros con tamaño 3x3, stride de 1 y padding activo.

Por último se aplican dos capas Linear, la última devuelve 5 salidas que corresponden a las 5 clases. En cuanto a funciones de activación, he utilizado funciones ReLU. He probado otras funciones, pero ninguna daba un resultado mejor.

Resumen de la red final:



```
Net(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc1): Linear(in_features=4096, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=5, bias=True)
)
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 16, 16]             896
         MaxPool2d-2            [-1, 32, 8, 8]               0
            Conv2d-3            [-1, 64, 8, 8]          18,496
            Linear-4                 [-1, 256]       1,048,832
            Linear-5                   [-1, 5]           1,285
================================================================
Total params: 1,069,509
Trainable params: 1,069,509
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.11
Params size (MB): 4.08
Estimated Total Size (MB): 4.20
----------------------------------------------------------------
```

Otra estructura de red que también me ha dado muy buenos resultados es parecida a la anterior, pero usando stride = 1 en la primera capa y añadiendo otra MaxPool2d al final (4 capas ocultas en total). Esta estructura daba prácticamente los mismos resultados pero iba el doble de lenta (probablemente por el stride), por lo que finalmente me quedé con la primera. Dejando stride=2 en la primera capa y añadiendo sólo el MaxPool2d, el resultado era similar, pero en general los resultados eran ligeramente peores.

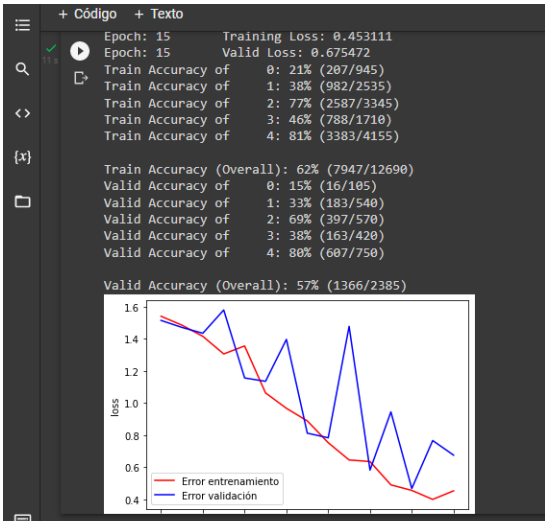
Resultados y Evaluación

A continuación voy a hablar de los cambios más relevantes que fui haciendo para obtener la red definitiva, y algunas conclusiones a las que llegué.

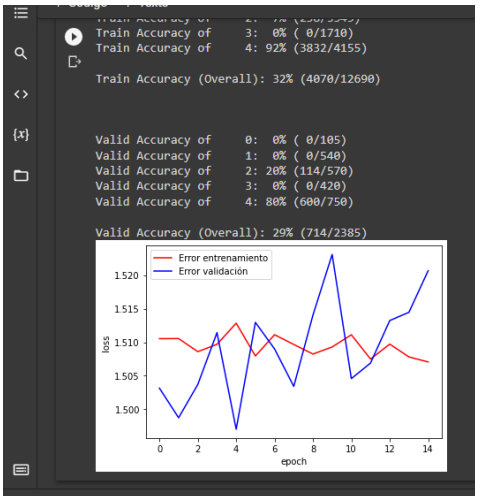
Inicialmente tenía una red diferente a la definitiva:

```
self.conv1=nn.Conv2d(num_channels,16, 3, stride=1, padding=0 )
self.pool1=nn.MaxPool2d(2)
self.conv2=nn.Conv2d(16, 32, 3, stride=1, padding=0)
self.pool2=nn.MaxPool2d(2);
self.conv3=nn.Conv2d(32, 64, 3, stride=1, padding=0 )
self.fc1=nn.Linear(64*4*4,256)
self.fc2=nn.Linear(256, 5)
```

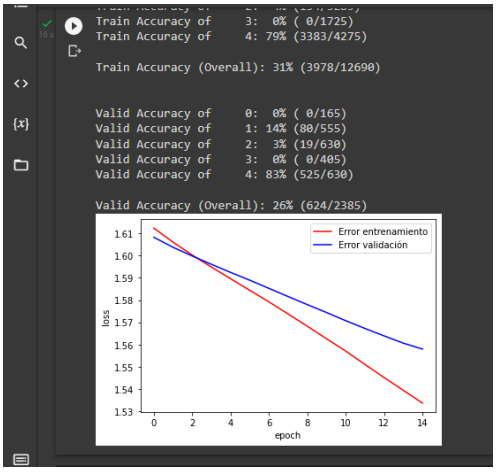
Y utilizaba un batch_size de 20, learning rate de 0.05 y 15 epoch. El conjunto de entrenamiento inicial era de 845 muestras de entrenamiento y 160 de validación. El resultado inicial fue bastante malo: La precisión era muy baja:



Al cambiar a learning rates altos, me salía peor. Con lr=0.5:

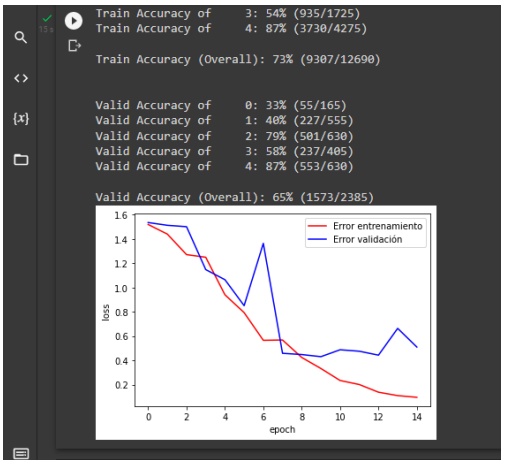


Y si ponía learning rates bajos, disminuye muy lentamente la loss. Con lr=0.001

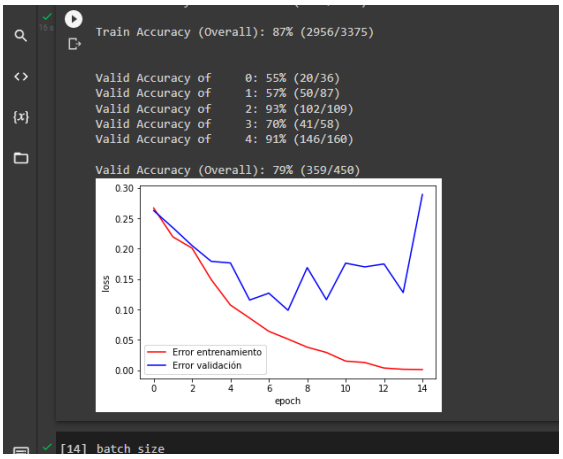


Por lo que si el learning rate es muy bajo, la función de pérdida apenas mejora.
Si es muy alto, puede divergir e irse a valores infinitos, o no llegar nunca a la solución.

Finalmente encontré un learning rate que se ajustaba mejor, un lr de 0.1:

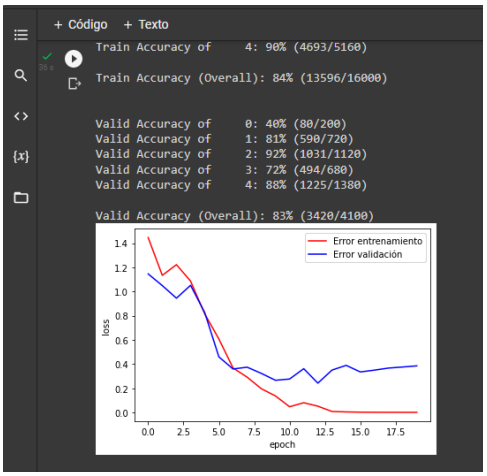


Aun así la precisión seguía siendo muy baja. Sigo probando combinaciones, y cambio el conjunto de entrenamiento y el batch size. Con batch size=10 los resultados eran mejores:

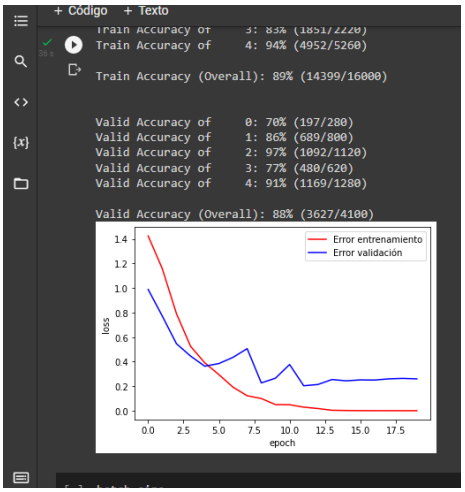


Por lo que es mejor poner un batch_size bajo. La red se entrena por lotes, ajusta la función de error tras cada lote. Se toman las n primeras muestras y entrena la red, luego las n siguientes…(siendo n el batch_size). Esto requiere menos memoria y aumenta la velocidad, ya que se actualizan los pesos después de cada propagación. Aún así, tampoco conviene que sea muy bajo, por lo que usé un batch_size= 10 y un batch_size=5.

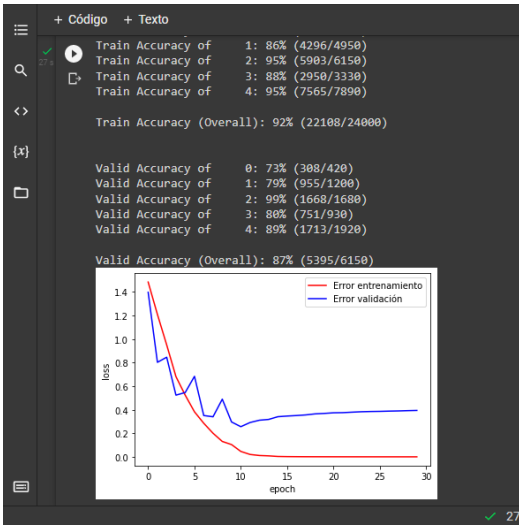
Cambié el número de imágenes de entrenamiento a 800 y validación a 205, ya que tenía muy pocas para validación. Con 20 epoch, los resultados mejoraron:



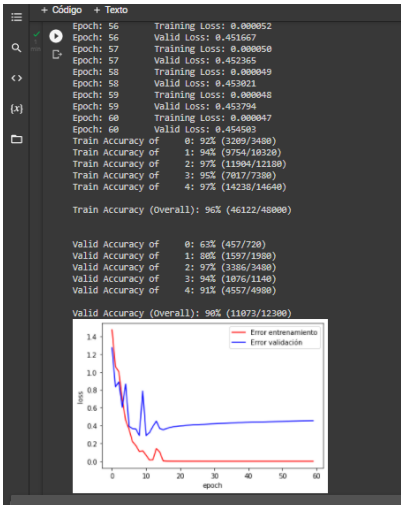
Seguí cambiando los parámetros del entrenamiento (learning rate, epoch y batch size), pero apenas conseguía mejorar. La precisión aún no era suficientemente alta. Decidí cambiar la red, quitando capas, inicialmente quité la última capa conv2d. También añadí relleno en las capas (padding).



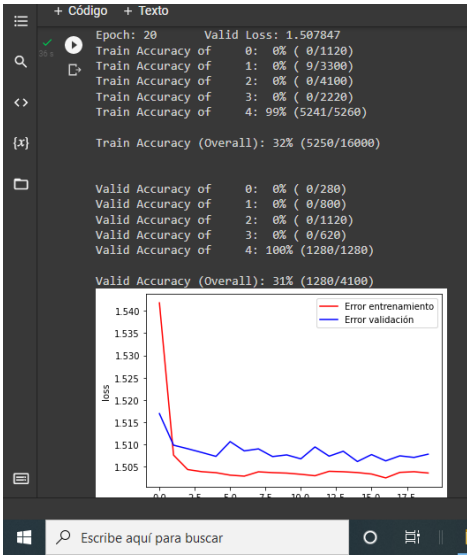
Y quité la última capa MaxPool2d y probé dándole el valor de stride=2 en la primera capa convolucional. Esto último no me cambió mucho la precisión, pero iba más rápido. También cambié el número de epoch: a partir de 30 prácticamente no disminuía la loss de entrenamiento, por lo que puse 30 epoch.



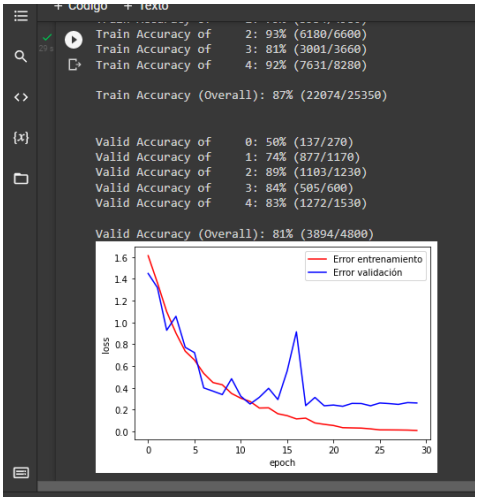
Con 60 epoch, por ejemplo, se consigue más precisión y disminuye más la loss, pero requiere mucho tiempo, además se produce overfitting (la loss de validación va aumentando en vez de disminuir)



Otros cambios que probé, con malos resultados, fue cambiar la función de activación. Probé una función softmax. Los resultados fueron:

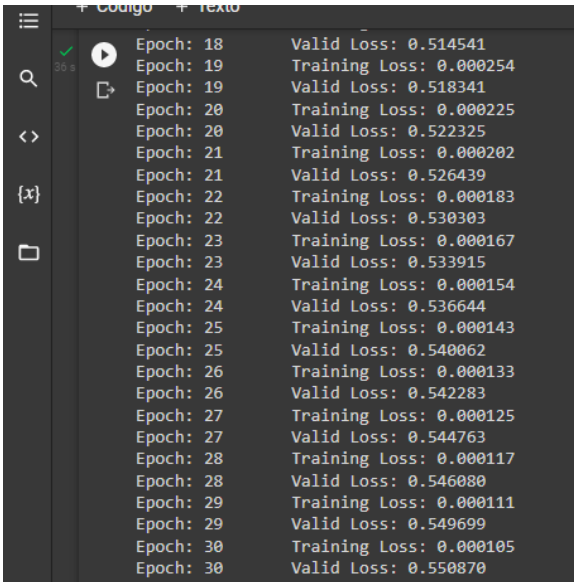


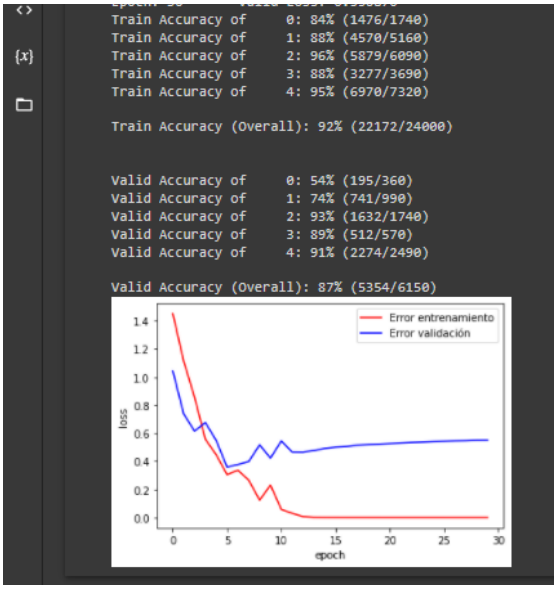
Y probando sigmoid:



Daban siempre peores resultados, por lo que la función ReLU es la que mejor se ajusta a los parámetros del modelo.

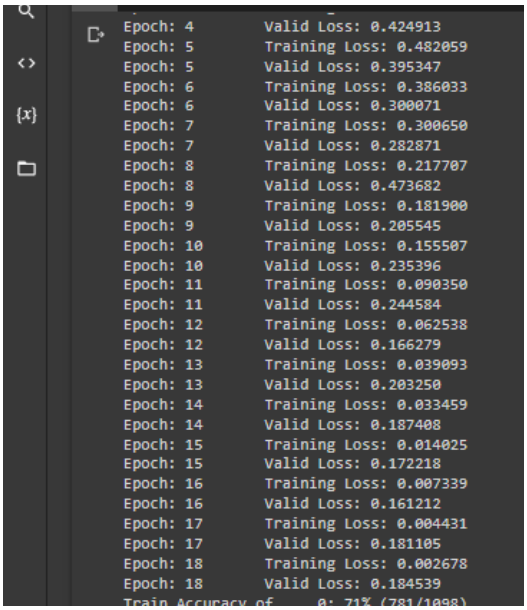
Resultados del modelo (batch_size=5, learning rate=0.1, 30 epoch):

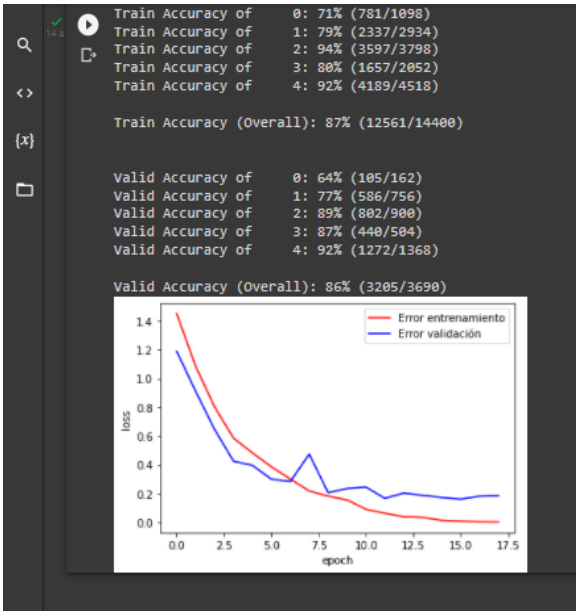




Con esta configuración conseguí muy buena precisión, sobre todo en los datos de entrenamiento. La loss del entrenamiento va disminuyendo, aunque la de la validación llega un punto en el que no disminuye, incluso aumenta. Esto es debido a que hay overfitting: el sistema clasifica cada vez mejor las muestras de entrenamiento, pero cada vez distingue peor las de validación. La precisión es 92% entrenamiento y 87% validación.

Probando, he comprobado que la loss de validación empieza a aumentar aproximadamente en la epoch 16, así que he cambiado el número de epoch a 16. También he cambiado el batch_size a 10, ya que da mejores resultados para la loss de validación. Y he cambiado el learning rate a 0.07.





Se puede ver que, aunque el porcentaje de precisión es inferior al del caso anterior, se evita el overfitting, ya que corta las epoch antes de que empiece a aumentar la loss de validación; Esto es más importante en este caso que la precisión de entrenamiento, es decir clasificará mejor los datos de test.