1st Summer School Physical Engineering UNI 2022-Lima,Perú

Deep learning: Clasificación de imagenes.

Chaico Cahuana, Angel Tomas[1]

[1]Facultad de Ciencias, Universidad Nacional de Ingeniería
[1]angel20170238b@gmail.com

February 28, 2022

Resumen

Se presenta las etapas en la tarea de clasificación de imágenes usando redes neuronales. De la data de kaggle se utilizan las imágenes de perros y gatos, las cuales se dividen en data de entrenamiento y de prueba. Para el método de evaluación de utiliza Cross entropy loss y SGD. Para realizar las tareas se utilizan las clases que nos ofrece pytorch, las cuales se utilizan en el código y se presentan en este proyecto. Las gráficas de los resultados del entrenamiento nos indica que se necesita mayor estudio sobre la propuesta de modelos en la tarea de clasificación y la toma de mayor cantidad de datos.

1 Metodología

1.1 Datos

Se utiliza como datos las imagenes de perros y gatos del dataset de kaggle. La cual contiene dos clases: gatos $\rightarrow 0$ y perros $\rightarrow 1$. Primero se descarga desde kaggle dos carpetas que contienen las datos de entrenamiento y de prueba, mediante la clase

torchvision.datasets.ImageFolder

, se redimensionan y se transforman a tensor.

De los 10000 datos se toman 2000 de entrenamiento y 500 de prueba para acortar el tiempo de ejecución. Para este objetivo se utiliza la siguiente clase

torch.utils.data.subset

La data de entrenamiento contiene imágenes a color y se estudia para $32 \cdot 32$ píxeles, ver figura (1) y (2) donde se muestra un perro y un gato respectivamente para pixeles de $100 \cdot 100$.

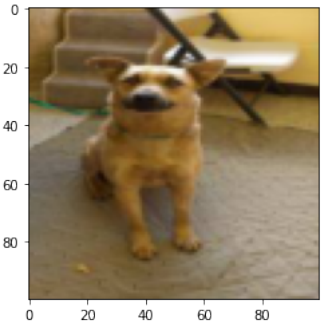

Figura 1: perro

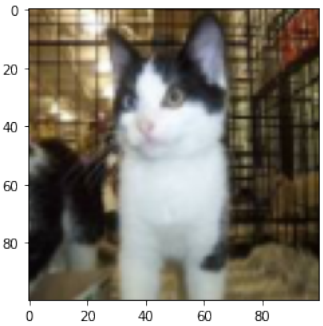

Figura 2: gato

Los datos (de entrenamiento y prueba) se reagrupan (batch) en grupos de 32 mediante la clase

torch.utils.data.DataLoader

donde como parámetro se utiliza el **shuffle=True**, lo cual los agrupa tomando elementos aleatoriamente.

Con lo mencionado asta ahora ya tenemos los datos de entrenamiento y prueba listos para ponerlos en entrenamiento según el modelo.

1.2 Método de entrenamiento del modelo.

Para definir el método de entrenamiento se necesita un criterio para evaluar que tan buena es la predicción del modelo y un método para mejorar dicha predicción. En el primer caso se utiliza

torch.nn.CrossEntropyLoss

la cual combina las clases *nn.LogSoftmax()* y *nn.NLLLoss()*. Para el segundo caso se utiliza

torch.optim.SGD.

la cual obtiene el mínimo a través de la gradiente y tomando pasos aleatorios.

Una vez definido los métodos, se pasa a especificar el entrenamiento:

- Se evalúa el modelo sobre los diferentes batch que se creo previamente.
- Se calcula la diferencia entre el objetivo y la predicción del modelo a través de *CrossEntropyLoss*.
- Se mejoran los parámetros del modelo usando *SGD* y se vuelve repetir.

1.3 Modelo

Se utilizan dos modelos a probar:

- Un modelo simple en cada capa, una clasificación lineal. Para la función de activación se utilizara la función ReLU (evitando neuronas muertas).
- El otro modelo es tomando en cuenta los bloques residuales.

2 Resultados

Se evalúa para la data de prueba, tomando diferentes learning rate, tomando 20 épocas para el entrenamiento. El eje x representa las épocas.

Se presenta la evaluación del modelo 1 para diferentes lr:

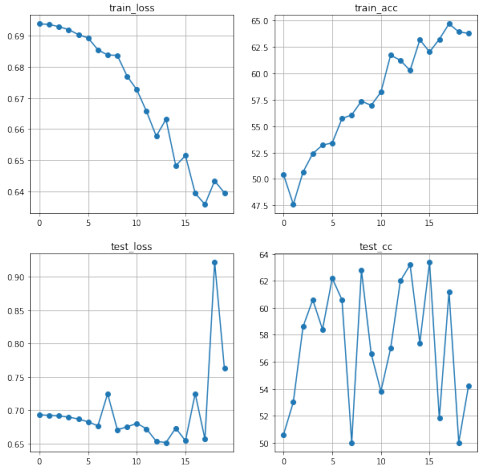

Figura 3: Evaluación del modelo 1, con $lr = 0{,}1$

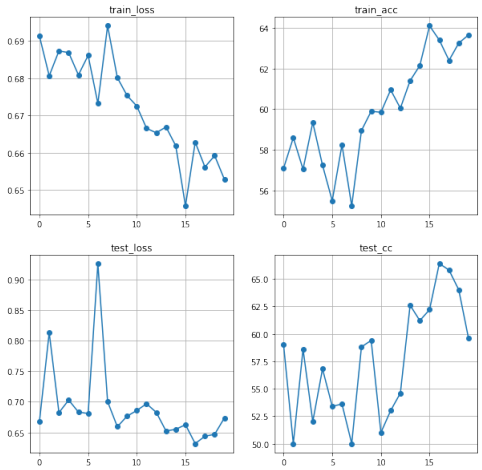

Figura 4: Evaluación del modelo 1, con $lr = 0{,}5$

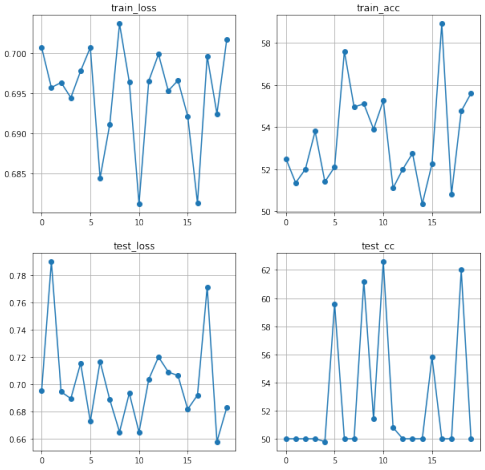

Figura 5: Evaluación del modelo 1, con $lr = 1$

Evaluación del modelo 2 para diferentes lr:

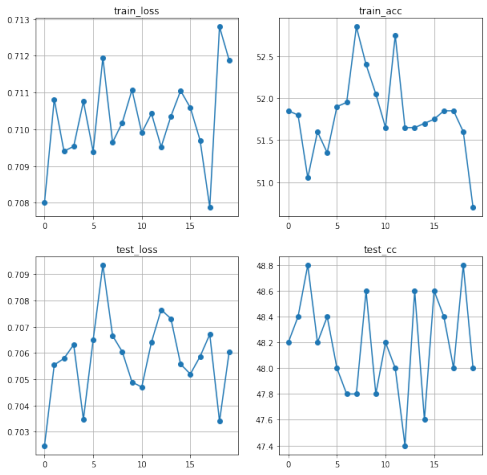

Figura 6: Evaluación del modelo 2, con $lr = 0{,}1$

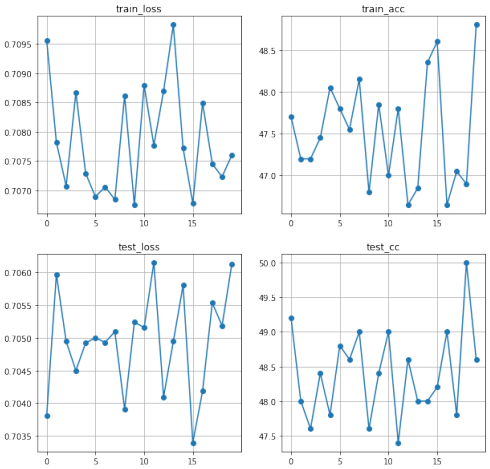

Figura 7: Evaluación del modelo 1, con $lr = 0{,}5$

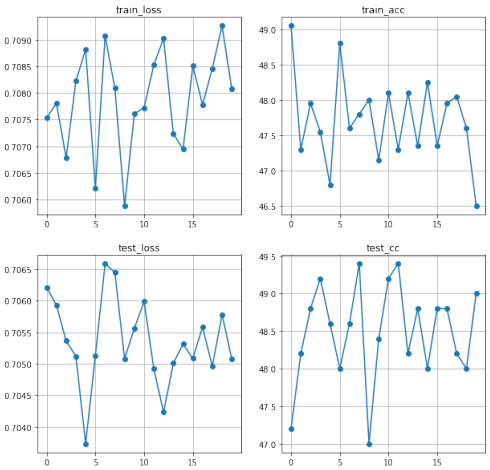

Figura 8: Evaluación del modelo 1, con $lr = 1$

Se presenta 5 imágenes para la predicción de cada modelo, ver figura (9) y (10).

3 Discusión de resultados

Según las gráficas (3),(4) y (5) vemos que a mayor valor de lr para el modelo 1, hay menor tendencia en las gráficas, además las mediciones en la precisión son altas desde inicio. La falta de tendencia se debe a que los pasos son muy grandes, lo cual no logran converger a un mínimo, sin embargo falta más análisis para determinar porque esto es así. Un mejor modelo podría ser propuesto.

Para el modelo 2 la tendencia es aun menor en cualquier caso, no se logra apreciar una tendencia clara a mejorar al pasar las épocas. Se puede concluir además que los resultados se deben a que se deben tomar mas datos en el entrenamiento, ya que con los tomados los modelos no logran mejorar en su predicción. Con lo cual podemos decir que no importa que modelo usemos, si no hay data suficiente el modelo no sirve. Queda para futuros trabajos investigar que cantidad de datos se necesitan para tener un modelo con predicción relativamente buena.

Conclusiones

Se presento las etapas de un proyecto de redes neuronales, sin embargo se necesita más estudio en la propuesta de los modelos en las capas. La parte mas importante en destacar en este trabajo es presentar las etapas con las funciones que pytorch nos ofrece: data, preparación de la data, método de entrenamiento, propuesta de modelos y la evaluación de los modelos.

Agradecimientos

Por último, quiero agradecer al grupo AEPIF UNI[1], en especial a Elvin Mark Muñoz Vega y Andersson Andreé Romero Deza, que quienes con sus consejos fueron el motor de arranque y mi constante motivación, muchas gracias por su paciencia y comprensión.

¡Muchas gracias por todo!

Referencias

[1] Documentación pytorch.

[2] Documentación matplotlib

[3] Guía de la clase Dataloader

[4] Documentación de la clase CrossEntropyLoss.

[1] Asociación de Estudiantes de Pregrado de Ingeniería Física UNI

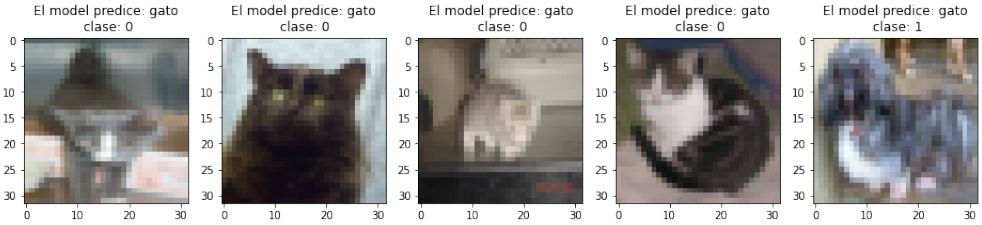

Figura 9: Predicción del modelo 1, con $lr = 0{,}5$

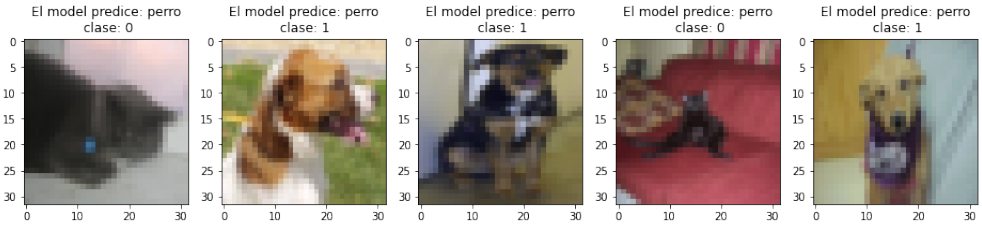

Figura 10: Predicción del modelo 2, con $lr = 0{,}5$

[5] Cross entropy loss in pytorch.