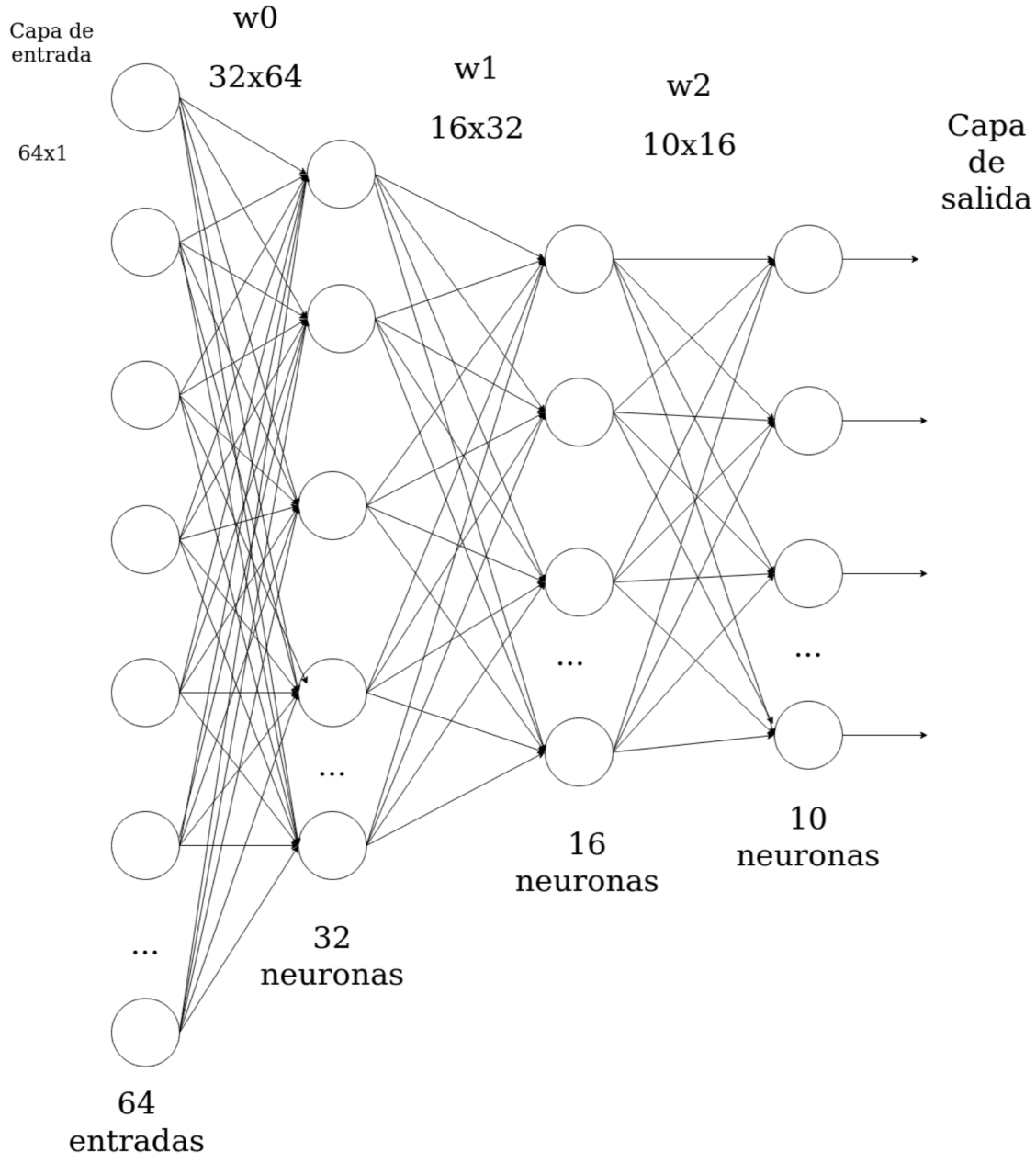
Leonardo Gracida Muñoz A01379812

Programa donde se aplica el algoritmo de back propagation para poder predecir imágenes de números escritos a mano usando la dataset de scikit-learn y Python.

Para este ejercicio la dataset que utilizamos es la digitos de scikit learn, que van de 0 al 9, imágenes de números escritos a mano, son imágenes con un tamaño de 8x8.

Utilizamos una red con la siguiente estructura:



Cómo son diez tipos de salidas diferentes vamos a usar diez neuronas de salida declarando un label a cada neurona de la capa de salida.

Además de utilizar las siguientes fórmulas de Back Propagation para poder actualizar los pesos de la red:

- Fórmula del error de la capa de salida:

$$\delta_k = O_k(1 - O_k)(t_k - O_k)$$

- Fórmula del error de las capas ocultas:

$$\delta_h = O_h(1 - O_h) \sum_{i=j}^{n} w_{jh}\delta_k$$

- Fórmula de la delta w:

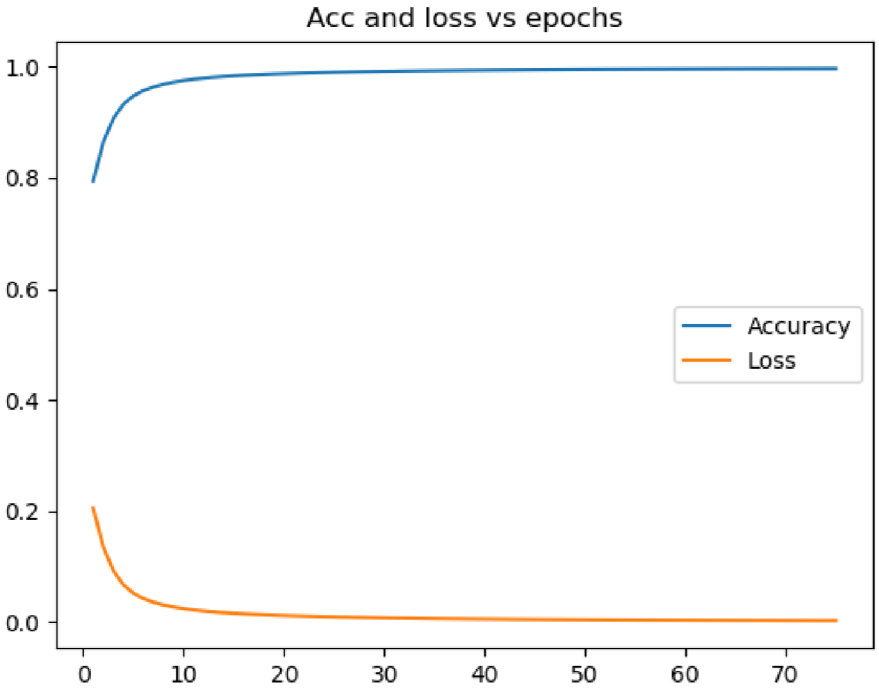
$$\Delta w = n * \delta_z * x_i$$

- Fórmula de actualización del peso:

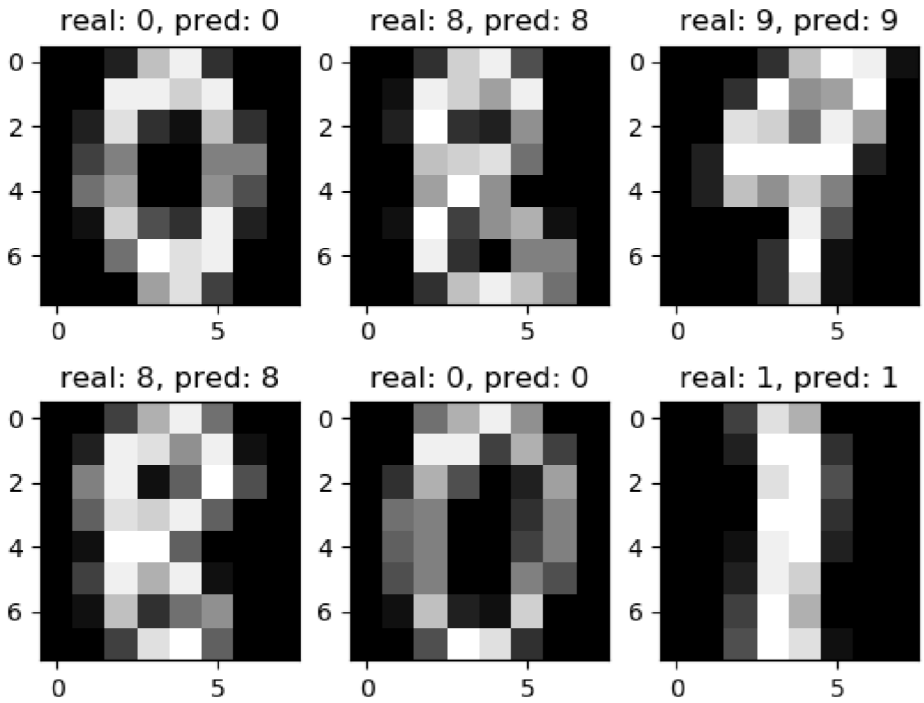
$$w_i = w + \alpha * \Delta w$$

- Curva de comportamiento del entrenamiento del modelo:

Leonardo Gracida Muñoz A01379812



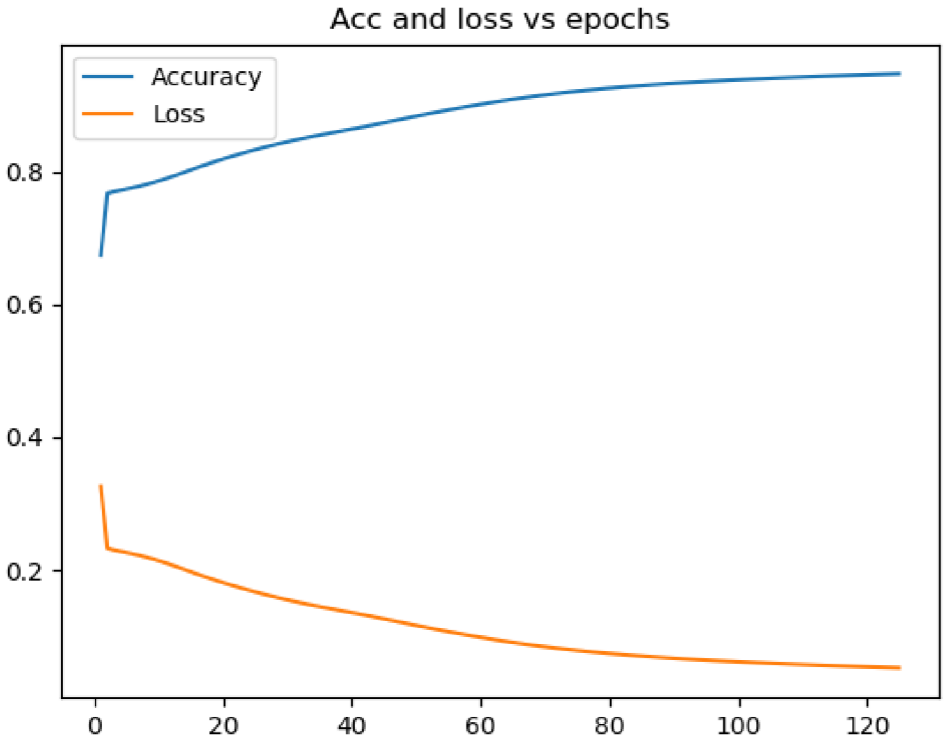
- Algunas predicciones hechas:



- El archivo train.py entrena la red y guarda los pesos obtenidos en un archivo pkl como la dataset de test (model: modelo.pkl, dataset test: test.pkl).
- El archivo test.py abre estos pesos guardados y la dataset de test del los archivos pkl y muestra la precisión del modelo en la dataset de test.
- Pruebas utilizando tres diferentes proporciones de train y test:

En estas pruebas utilizamos un lr de 0.01 y 125 epochs:

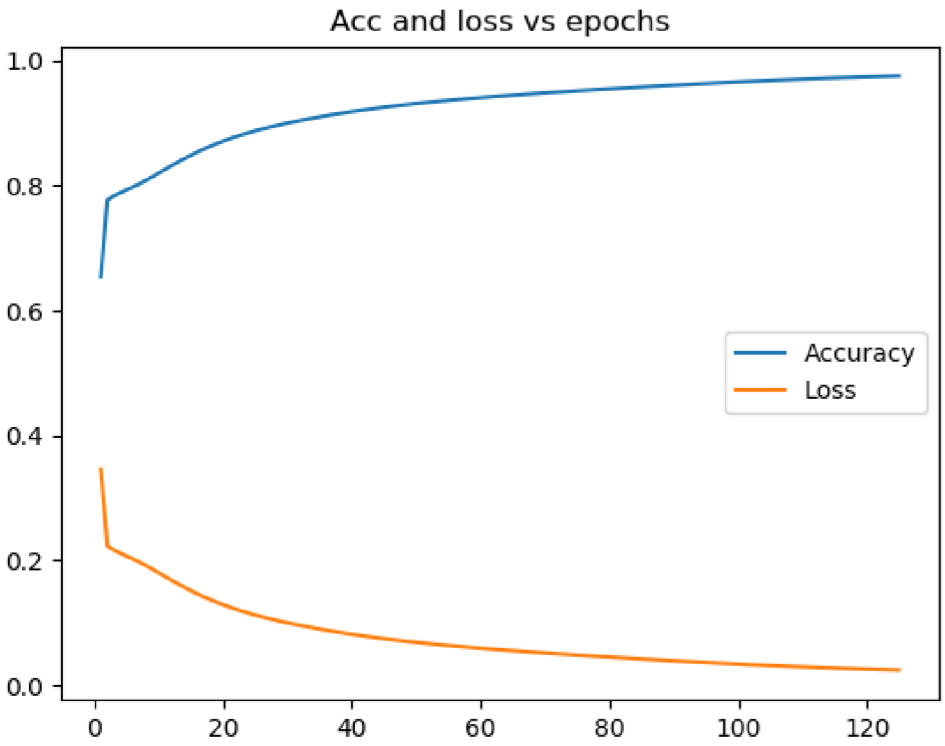
- Usando 30% de test:



Accuracy del modelo con la parte de test: 94.16%
Pérdida del modelo con la parte de test: 0.058

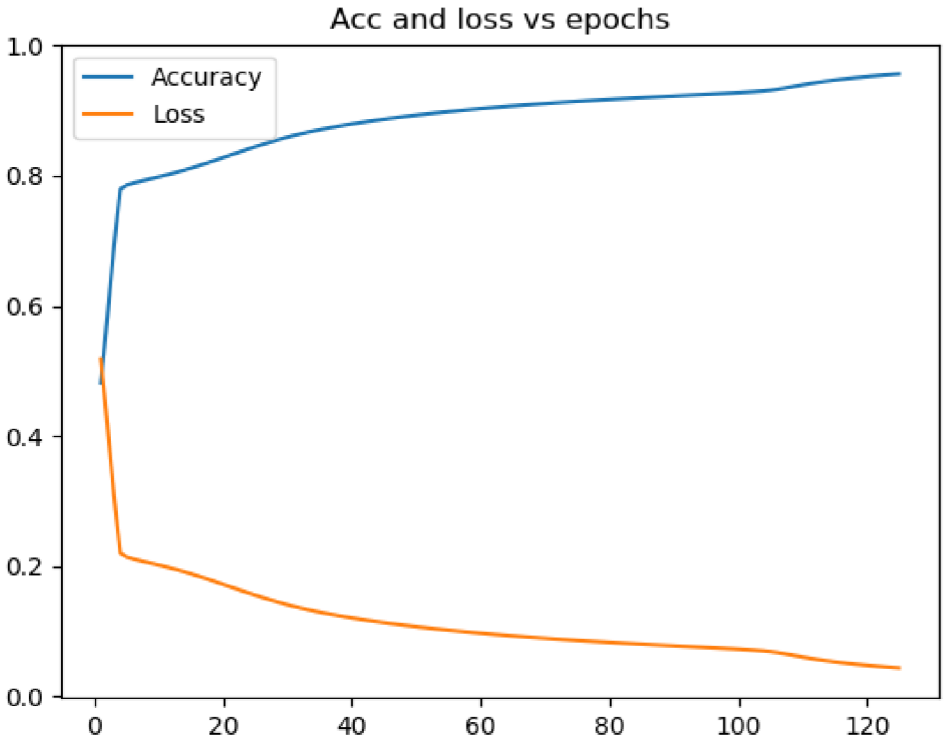
- Usando 20% de test:

Leonardo Gracida Muñoz A01379812



Accuracy del modelo con la parte de test: 94.84%
Pérdida del modelo con la parte de test: 0.051

- Usando 40% de test:



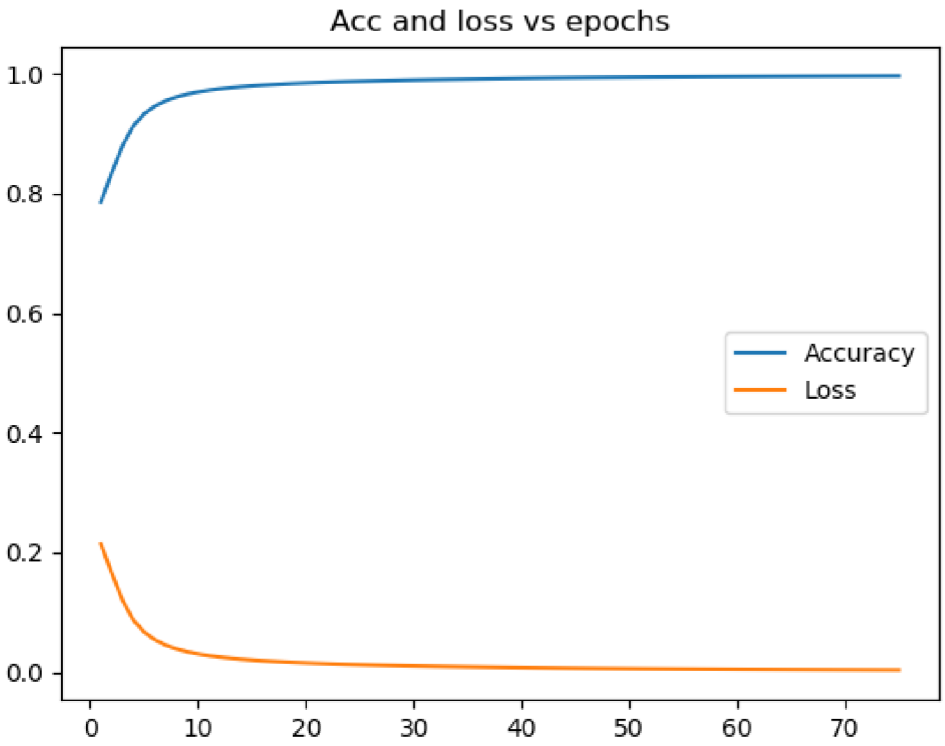
Accuracy del modelo con la parte de test: 94.09%
Pérdida del modelo con la parte de test: 0.059

Como podemos ver, en estas pruebas la mejor división de la dataset es el 20% de test aproximadamente. Ya que usado 40% el entrenamiento del modelo se acerca al área de underfitting, ya que al tener menos muestras para entrenar en cada epoch este no logra aprender todos lo parámetros y no logra llegar al mínimo local, lo mismo pasa con la separación del 30%, todavía le falta para poder entrenar o encontrar el mínimo global o acercarse más a él. Teniendo esto sentido, ya que al tener una cantidad considerablemente grande de targets o labels, necesitamos una cantidad igual de considerable de muestras de cada categoría, es por eso que la más pequeña dio mejores resultados.

- Cambio de los hiperparametros:

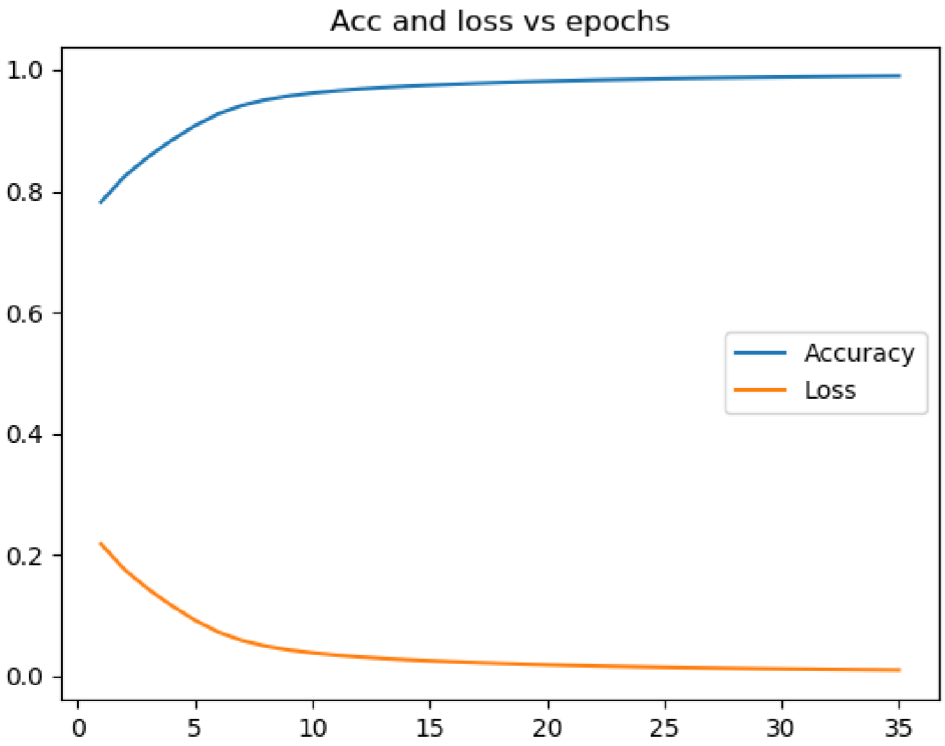
En nuestro caso vamos a cambiar el número de epochs y el learning rate.

- lr de 0.1, 75 epochs y un test del 20%:



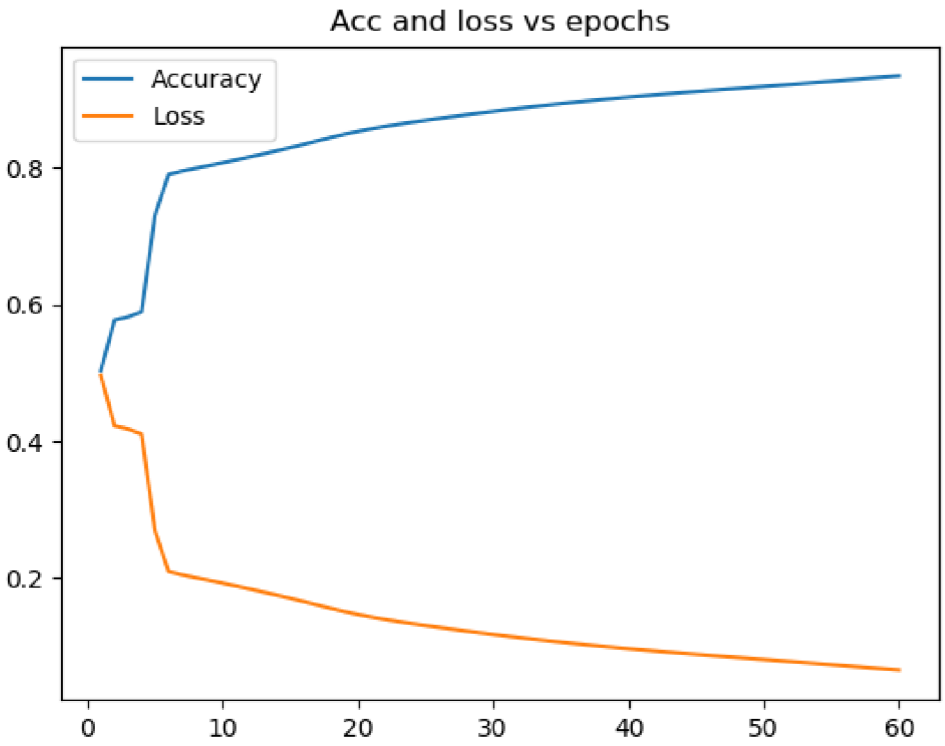
Accuracy del modelo con la parte de test: 98.4%
Pérdida del modelo con la parte de test: 0.016

- lr de 0.1, 35 epochs y un test del 20%:



Accuracy del modelo con la parte de test: 96.62%
Pérdida del modelo con la parte de test: 0.034

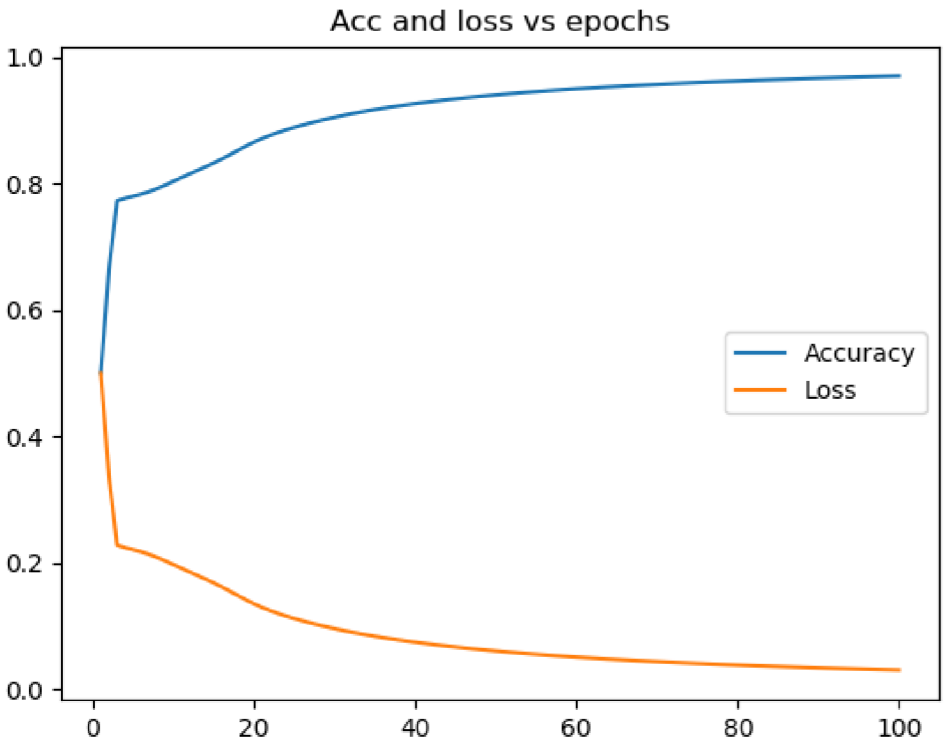
- lr de 0.01, 60 epochs y un test del 20%:



Accuracy del modelo con la parte de test: 92.6%
Pérdida del modelo con la parte de test: 0.074

- lr de 0.01, 100 epochs y un test del 20%:

Leonardo Gracida Muñoz A01379812



Accuracy del modelo con la parte de test: 93.7%
Pérdida del modelo con la parte de test: 0.062

- Análisis:

Como podemos ver nuestro modelo no necesita un learning rate tan pequeño como tampoco necesita tantas epochs, es por eso que se declaró el usar 75 epochs con un lr de 0.1, ya que es el que mejor nos a dado comportamiento al hacer las pruebas en la parte de test, viendo que no necesitamos dar pasos tan pequeños para poder llegar al mínimo global, además de dar el comportamiento de acc y loss más plano o que converge de una manera más uniforme viendo las gráficas. Mostrando que el bias y varianza de nuestro modelo son pequeños, ya que nuestro modelo converge uniformemente, y su loss en las pruebas es pequeño, mostrando que si se equivoca no devuelve una respuesta alejada de la ideal. Demostrando que nuestro modelo no está en overfitting ni en underfitting, sino que está entrenado de manera óptima.