Informe de los Experimentos Realizados

Mini proyecto: Redes neuronales Aplicaciones en Ciencia de Datos e Inteligencia Artificial

Alumna: Dora Novoa

Se realiza una serie de experimentos cambiando los parámetros en la primera parte, para que veamos cuales tenga mejor accuracy. Y en la segunda parte se cambia los hiperparámetros para ver cual es la mejor alternativa.

Perceptron multicapa

Parte 1: Modificaciones en la Arquitectura - Cambio de parámetros

Para imágenes de 28x28 y donde las clases son 10 (dígitos del 0 al 9) se tiene las siguientes combinaciones de parámetros dejando fijos los hiperparámetros para la evaluación.

Función Activación	Cant. neuronas	Cant. Capas ocultas *(longitud de list neuronas)*
ReLU	**[784, 512, 256]**	**3**
Tanh	**[784, 512, 256, 128]**	**4**
Softplus	**[784, 392]**	**2**
Sigmoid	**[784, 392, 128, 64, 32]**	**5**

Para esta evaluación fijaremos los hiperparámetros:

batch_size: 64 - epochs: 10 - learning_rate: 0.001 - optimizer_funcion: Adam

Entonces, en el código se realiza lo siguiente:

1. Se deja en un proceso automatizo la clase MLP_D donde se pasará por parámetro el nombre de la función de activación y la cantidad de neuronas, entre otras, así: class MLP_D(nn.Module) pasándose como parámetros (tam_imagen, capas_ocultas, tam_clase, activation_fn=nn.ReLU, dropout_prob=0.2)

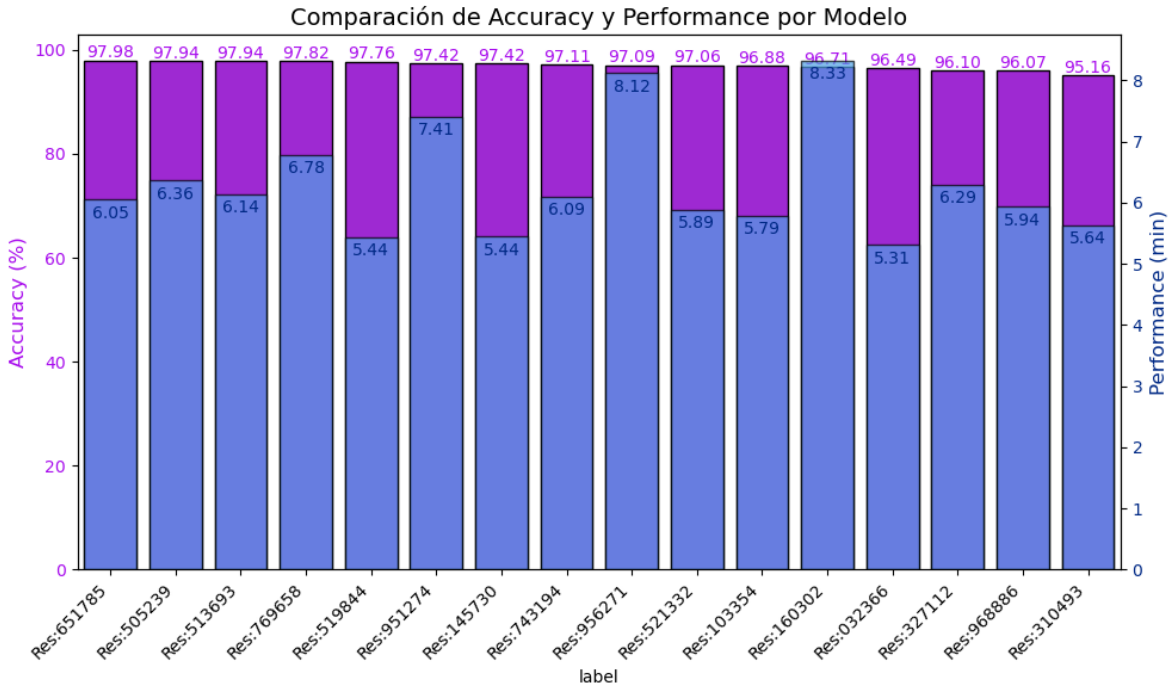
2. Se genera un archivo “combi_parametros.YAML” donde se pondrán todas las combinaciones posibles de funciones de activación con las listas de cantidad de neuronas. Lista de pares de sets que se guarda al archivo YAML, dado que se repiten las listas de cantidad de neuronas, el archivo YAML está utilizando referencias para evitar duplicación de datos &id001 se vincula al *id001
3. Se genera una función def procesar_MNIST(col_param, col_hiper, df_param, df_hiper, train_dataset, test_dataset), donde los parámetros de entrada sería la combinación de donde se encuentra la función de activación junto con su respectivo cantidad de neuronas, en todas la combinaciones posibles, y aquí se dejaría fijo los hiperparámetros. Cabe mencionar que el Preprocesamiento y carga de datos de MNIST, lo dejé fuera del ciclo for,

para optimizar el tiempo de ejecución, el train_dataset, test_dataset lo paso en la función como parámetro.

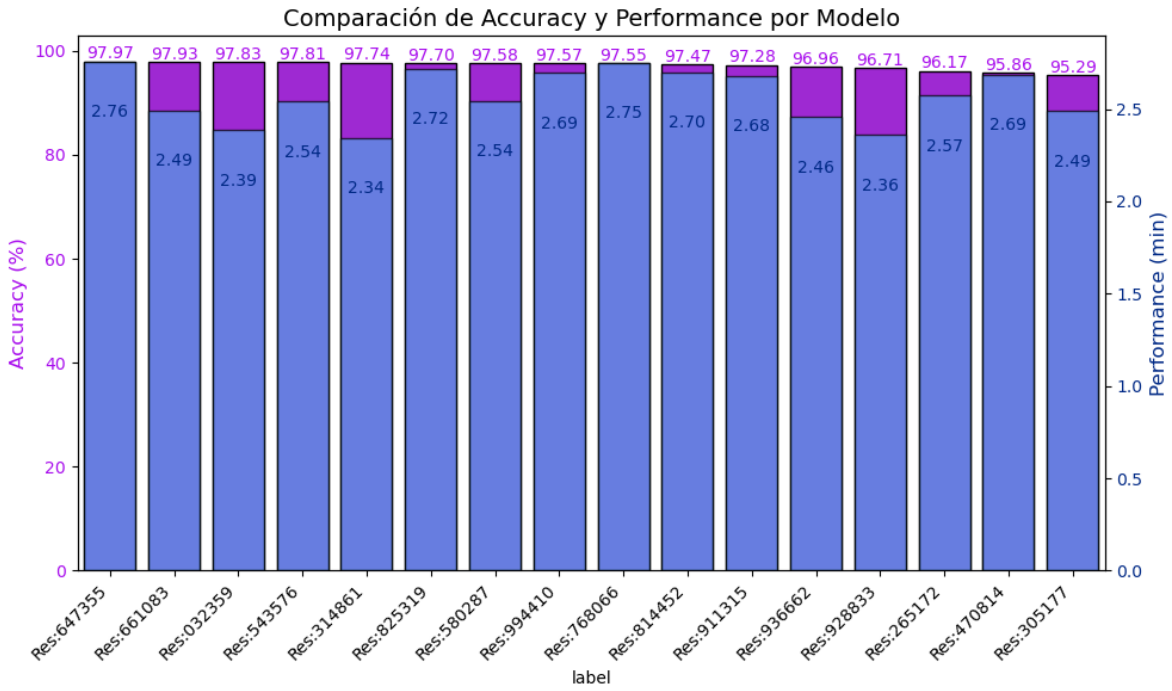
4. Al realizar este entrenamiento se realiza un ciclo for para que se vayan generando todas las posibles combinaciones de entrenamiento.
5. IMPORTANTE en este ciclo for por cada combinación de funciones de activación con lista de neuronas, se puso dos timestamp para dejar en una variable el tiempo tomado por ciclo, la diferencia es la performance.
6. Se realiza la métrica accuracy para cada combinación.
7. Los resultados como el accuracy, performance, y combinación de la función activación y cantidad de neuronas se dejaría finalmente en un dataframe para exportarlo a un gráfico y ahí se podrá analizar quien tiene el mejor accuracy y performance.
8. Este resultado finalmente se exporta a un archivo: df_final.csv
9. En el siguiente gráfico pueden visualizar el Accuracy vs Performance por Modelo, aquí se puede ver el análisis de cada modelo y su comportamiento de acuerdo a estas dos variables para encontrar la mejor distribución de arquitectura

Estudiando el código de redes convolucionales, encontré la integración del CUDA en CPU, y se ve un cambio grande en el performance.

ANTES DEL CUDA



DESPUES DEL CUDA



El mejor modelo es el set de Res:647355 con los parámetros activation_fn:Softplus – capas ocultas:4 cantidad de neuronas ocultas [784, 512, 256, 128] con un accuracy de 97.97% y Performance de 02:45,448 minutos, con los hiperparámetros optimizador_fn:Adam - batch:64 – learning rate:0.001 - epoch:10

Respuesta: en la tabla siguiente se puede ver los valores de la accuracy pero también de la performance (en minutos) de cada modelo.

label	accuracy	performance	parametros	hiperparametros	timestamp
Res:647355	97.97	02:45,4	act_fn:Softplus - c:4	op_fn:Adam - batch:64 - lr:0.001 - e:10	24-12-20_647355
Res:661083	97.93	02:29,5	act_fn:Softplus - c:5	op_fn:Adam - batch:64 - lr:0.001 - e:10	24-12-20_661083
Res:032359	97.83	02:23,1	act_fn:Softplus - c:2	op_fn:Adam - batch:64 - lr:0.001 - e:10	24-12-20_032359
Res:543576	97.81	02:32,4	act_fn:ReLU - c:2	op_fn:Adam - batch:64 - lr:0.001 - e:10	24-12-20_543576
Res:314861	97.74	02:20,5	act_fn:Sigmoid - c:2	op_fn:Adam - batch:64 - lr:0.001 - e:10	24-12-20_314861
Res:825319	97.7	02:43,2	act_fn:Softplus - c:3	op_fn:Adam - batch:64 - lr:0.001 - e:10	24-12-20_825319

label	accuracy	performance	parametros	hiperparametros	timestamp
Res:580287	97.58	02:32,7	act_fn:Sigmoid - c:3	op_fn:Adam - batch:64 - lr:0.001 - e:10	24-12-20_580287
Res:994410	97.57	02:41,6	act_fn:ReLU - c:3	op_fn:Adam - batch:64 - lr:0.001 - e:10	24-12-20_994410
Res:768066	97.55	02:45,0	act_fn:Sigmoid - c:4	op_fn:Adam - batch:64 - lr:0.001 - e:10	24-12-20_768066
Res:814452	97.47	02:42,0	act_fn:ReLU - c:5	op_fn:Adam - batch:64 - lr:0.001 - e:10	24-12-20_814452
Res:911315	97.28	02:40,7	act_fn:ReLU - c:4	op_fn:Adam - batch:64 - lr:0.001 - e:10	24-12-20_911315
Res:936662	96.96	02:27,3	act_fn:Sigmoid - c:5	op_fn:Adam - batch:64 - lr:0.001 - e:10	24-12-20_936662
Res:928833	96.71	02:21,5	act_fn:Tanh - c:2	op_fn:Adam - batch:64 - lr:0.001 - e:10	24-12-20_928833
Res:265172	96.17	02:34,4	act_fn:Tanh - c:3	op_fn:Adam - batch:64 - lr:0.001 - e:10	24-12-20_265172
Res:470814	95.86	02:41,2	act_fn:Tanh - c:4	op_fn:Adam - batch:64 - lr:0.001 - e:10	24-12-20_470814
Res:305177	95.29	02:29,3	act_fn:Tanh - c:5	op_fn:Adam - batch:64 - lr:0.001 - e:10	24-12-20_305177

Parte 2: Modificaciones del entrenamiento- Cambio de hiperparámetros

Ahora teniendo fija la mejor arquitectura anterior, se cambiará los hiperparámetros.

Algoritmo de optimización	**Tasa de aprendizaje (learning rate)**	Tamaño del Lote **Batch size**	**Número de Épocas epochs**
Adam	**0.1**	**16**	**5**
SGD	**0.01**	**32**	**10**
RMSprop	**0.001**	**64**	**15**
Adagrad	**0.0001**	**128**	**20**

Por un tema de tiempo tuve que acortar estas combinaciones que serían 16 x 16 = 256 distintas pruebas de modelos de combinación de hiperparámetros, a sólo 16, dado que en la evaluación de cada modelo se demoraba 14 min. entonces me daba un tiempo de ejecución de 59,73 horas. Esto

en Google Colab.

```
Procesado modelo 1 de 256 de la combinación de hiperparámetros Adam - 5 - 0.1 - 16
Comienza a procesar el modelo en: 2024-12-20 16:17:56.178829
Termina el procesamiento del modelo a: 2024-12-20 16:31:43.958947
Accuracy en el conjunto de prueba: 10.10%
Procesado modelo 2 de 256 de la combinación de hiperparámetros SGD - 5 - 0.1 - 16
Comienza a procesar el modelo en: 2024-12-20 16:31:48.254746
```

Algoritmo de optimización	Tasa de aprendizaje (learning rate)	Tamaño del Lote Batch size	Número de Épocas epochs
Adam	**0.1**	**16**	**5**
SGD	**0.001**	**32**	**10**

Esto tiene 16 distintas combinaciones de los hiperparámetros.

Esto es: **SGD** (Stochastic Gradient Descent), **Adam** (Adaptive Moment Estimation), **RMSProp** (Root Mean Square Propagation), **Adagrad** (Adaptive Gradient Algorithm)

Para esta evaluación fijaremos los parámetros de la arquitectura encontrados anteriormente:

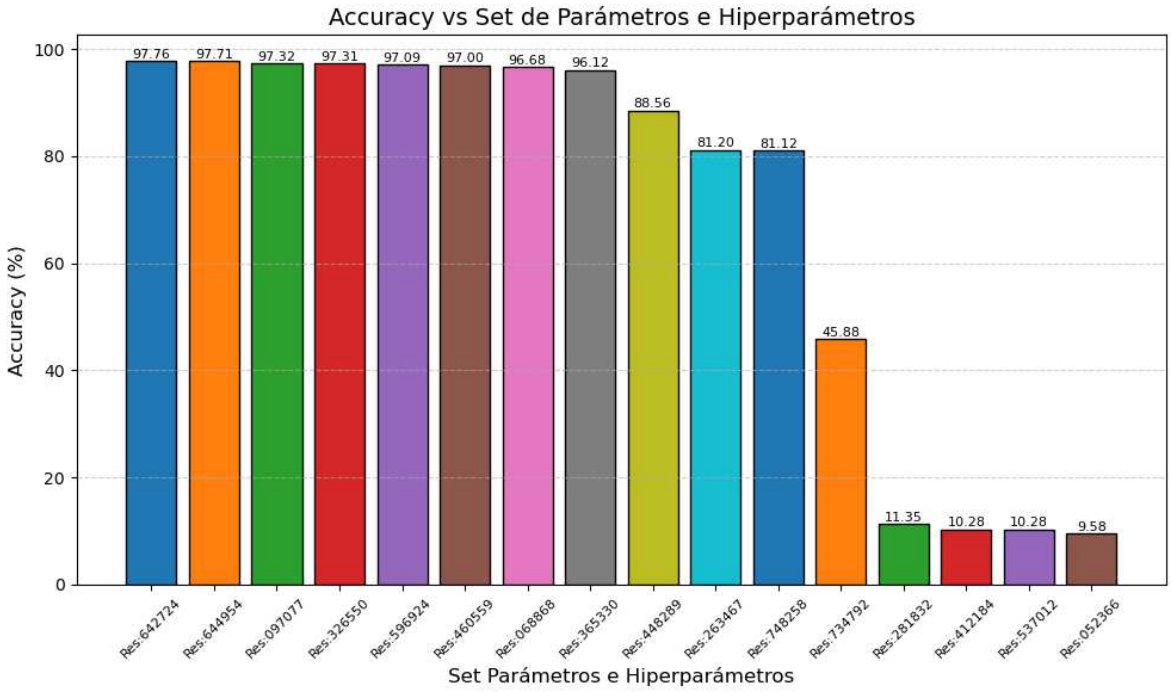
función activación: Softplus – número de capas:4 – cantidad de neuronas ocultas [784, 512, 256, 128]

Se codifica de la misma forma, automatizando los procesos de forma que se combinen los hiperparámetros para obtener el mejor accuracy con su performance.

1. Se realiza una función que realiza todas las posibles combinaciones de los hiperparámetros para dejar estas combinaciones en un archivo YAML y luego llamarlos e invocarlos en un ciclo for para el entrenamiento del modelo. Esta función de combinaciones de hiperparámetros me devuelve una lista de set de 4 hiperparámetros.
2. Proceso los 16 * 16 distintas combinaciones de hiperparámetros, pero tuve que reducirla a sólo 16 distintas combinaciones. Los puse en un df_hiper, junto con el df_param_best que queda fijo dado que fue la mejor arquitectura obtenida anteriormente. Esta vez no voy a graficar pérdida vs época sino que se guarda en un dataframe, y luego en la resultados_hiper.yaml
3. Luego de su entrenamiento, se almacenarán los datos resultados como métricas y performances como también los hiperparámetros usados, uno a uno en otro archivo YAML de resultados_hiper.YAML
4. Se rescatarán estos resultados para graficarlos y así analizarlos. Sería accuracy vs performance, ordenados al máximo accuracy. Obteniendo así el mejor modelo.
5. Además, ese resultante dataframe lo exportaremos a un archivo CVS llamado df_hiper_final.cvs

Gráfico resultante me muestra que el learning rate = 0.1 tiene el peor accuracy, cualquiera sea sus otros hiperparámetros. Los batches pequeños hacen que la red actualice los pesos más seguido, pero con mucho más ruido (o sea, las actualizaciones no son tan precisas). Esto puede ser que necesitaría hacer más iteraciones para llegar a la misma precisión que con

batches más grandes. Además, cada batch pequeño tiene un poco más de trabajo extra en cada cálculo, así que en total puede tomar más tiempo entrenar el modelo.



El mejor modelo es el set de Res:642724 con los parámetros activation_fn:Softplus – capas_ocultas:3 cantidad de neuronas [784, 512, 256] con un accuracy de 97.76% y Performance de 03:20,848 minutos, con los hiperparámetros optimazer_fn:Adam - batch:32 – learning_rate:0.001 - epoch:10

label	accuracy	performance	parametros	hiperparametros	timestamp
Res:642724	97.76	03:20,8	act_fn:Softplus - c:3	op_fn:Adam - batch:32 - lr:0.001 - e:10	24-12-20_642724
Res:644954	97.71	03:31,7	act_fn:Softplus - c:3	op_fn:SGD - batch:16 - lr:0.1 - e:10	24-12-20_644954
Res:097077	97.32	05:02,5	act_fn:Softplus - c:3	op_fn:Adam - batch:16 - lr:0.001 - e:10	24-12-20_097077
Res:326550	97.31	01:39,9	act_fn:Softplus - c:3	op_fn:Adam - batch:32 - lr:0.001 - e:5	24-12-20_326550
Res:596924	97.09	01:44,1	act_fn:Softplus - c:3	op_fn:SGD - batch:16 - lr:0.1 - e:5	24-12-20_596924
Res:460559	97.0	02:40,3	act_fn:Softplus - c:3	op_fn:SGD - batch:32 - lr:0.1 - e:10	24-12-20_460559
Res:068868	96.68	02:28,0	act_fn:Softplus - c:3	op_fn:Adam - batch:16 - lr:0.001 - e:5	24-12-20_068868
Res:365330	96.12	01:22,8	act_fn:Softplus - c:3	op_fn:SGD - batch:32 - lr:0.1 - e:5	24-12-20_365330

label	accuracy	performance	parametros	hiperparametros	timestamp
Res:448289	88.56	03:28,3	act_fn:Softplus - c:3	op_fn:SGD - batch:16 - lr:0.001 - e:10	24-12-20_448289
Res:263467	81.2	01:50,6	act_fn:Softplus - c:3	op_fn:SGD - batch:16 - lr:0.001 - e:5	24-12-20_263467
Res:748258	81.12	02:39,0	act_fn:Softplus - c:3	op_fn:SGD - batch:32 - lr:0.001 - e:10	24-12-20_748258
Res:734792	45.88	01:22,3	act_fn:Softplus - c:3	op_fn:SGD - batch:32 - lr:0.001 - e:5	24-12-20_734792
Res:281832	11.35	02:31,4	act_fn:Softplus - c:3	op_fn:Adam - batch:16 - lr:0.1 - e:5	24-12-20_281832
Res:412184	10.28	01:41,8	act_fn:Softplus - c:3	op_fn:Adam - batch:32 - lr:0.1 - e:5	24-12-20_412184
Res:537012	10.28	04:44,8	act_fn:Softplus - c:3	op_fn:Adam - batch:16 - lr:0.1 - e:10	24-12-20_537012
Res:052366	9.58	03:26,5	act_fn:Softplus - c:3	op_fn:Adam - batch:32 - lr:0.1 - e:10	24-12-20_052366

Redes convolucionales

Parte 1: Modificaciones en la Arquitectura - Cambio de parámetros

Modificar por estos parámetros: el número de filtros, el número de neuronas en la capa posterior a las capas convolucionales, agregando más capas lineales y el parámetro asociado a dropout.

Número de Filtros filters_l1	Número de Filtros filters_l2	Cant. Neuronas después de CNN final_layer_size	Dropout dropout
8	32	30	0.2
32	64	100	0.3
64	128	200	0.4

Para esto dejamos los hiperparámetros **Optimizador Adam**, **learning_rate=0.001**, **batch_size=128**, **num_epochs = 10**

En el código de todas maneras estoy tratando de dejar en un archivo 'resultado_cnn.yaml' el resultado combinaciones de parámetros, hiperparámetros, acurracy, performance, al igual que lo hice con el perceptor multicapa. Nota aparte: Tuve un problema con serialización de objetos de numpy como usé la Optimización Bayesiana, para la selección de los parámetros, los tuve que convertir a dato nativo de Python para que los pudiera guardar.

Según el profesor para optimizar el tiempo, he implementado el uso de una **Optimización Bayesiana.**

Uso de la Optimización Bayesiana, para esto la función objetivo es el aprendizaje y cálculo de métrica accuracy de la CNN.

Aquí, se ajusta los parámetros de una CNN para el dataset MNIST dado que puede ser un proceso muy demandante en términos de tiempo. Probar todas las combinaciones posibles de parámetros como el número de filtros, la cantidad de neuronas en la capa totalmente conectada, las capas adicionales y la tasa de dropout puede tomar demasiado tiempo. Por esta razón, se usa la **Optimización Bayesiana**, que es una estrategia más eficiente para encontrar configuraciones óptimas sin necesidad de explorar todo el espacio de búsqueda.

En este caso, nos enfocamos en ajustar los parámetros indicados arriba, esta optimización utilizaría un modelo probabilístico para predecir cómo los valores de los parámetros podrían afectar el rendimiento del modelo. Enfocándose en donde se encuentren mejores resultados, y principalmente reduce el tiempo de computador dada la entrega de este proyecto.

Consideraciones

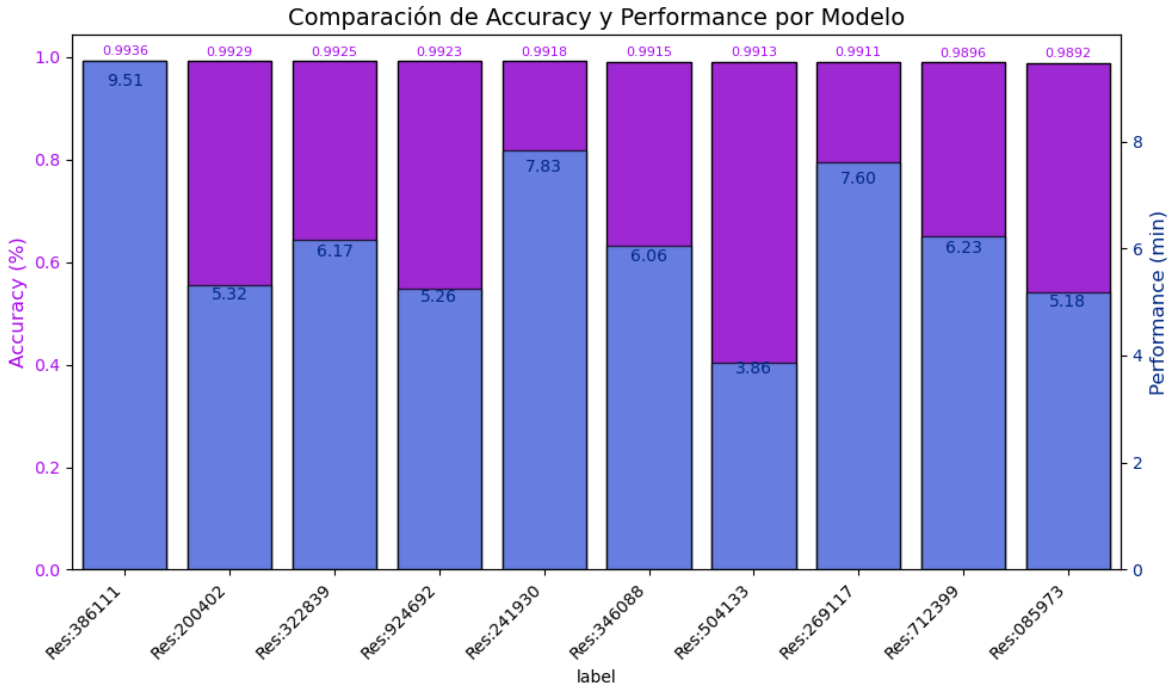
Aunque la Optimización Bayesiana es mucho más eficiente, cada combinación de estos parámetros todavía implica entrenar y evaluar el modelo, lo cual toma tiempo. Para hacerlo manejable, hemos implementado algunas estrategias: Reducción de épocas a 10, un número limitado de iteraciones de

pruebas a 10, y definí rangos razonables para cada parámetro (por ejemplo, filtros entre 8 y 64, dropout entre 0.2 y 0.4). Esperando encontrar dentro de este espacio, la mejor combinación de parámetros de configuraciones óptimas en menos tiempo para poder fijarla en el siguiente paso.

Mejor combinación de hiperparámetros:
{'filters_l1': 34, 'filters_l2': 115, 'final_layer_size': 146, 'dropout': 0.2900998503939086}
Mejor precisión obtenida: 0.9936

Pero en la operación Bayesiana, eligió filtros 34 y 115 como los óptimos.

Sin embargo, como se guarda en un archivo YAML todos los 10 ciclos de la optimización Bayesiana, se logra cargar en un dataframe para después graficarlo de formada descendente donde se obtiene le mayor accuracy y se ilustra también la performance. Los datos de este gráfico los guardo en un archivo CVS, todo esto lo pueden ver en la carpeta respuesta en Github.



El grafico están los minutos en float, pero en la tabla en minutos en si, por esto es la diferencia.

label	accuracy	Performance en min	parametros	hiperparametros	timestamp
Res:386111	0.9936	09:30,8	f_l1:34 - f_l2:115 - drp:0.2900998503939086 - fi_lay:146	op_fn:Adam - lr:0.001 - e:10	24-12-20_386111
Res:200402	0.9929	05:19,0	f_l1:13 - f_l2:91 - drp:0.3966461771613577 - fi_lay:95	op_fn:Adam - lr:0.001 - e:10	24-12-20_200402

label	accuracy	Performance en min	parametros	hiperparametros	timestamp
Res:322839	0.9925	06:10,0	f_l1:9 - f_l2:122 - drp:0.27708330050798324 - fi_lay:126	op_fn:Adam - lr:0.001 - e:10	24-12-20_322839
Res:924692	0.9923	05:15,4	f_l1:30 - f_l2:36 - drp:0.24655426808606087 - fi_lay:196	op_fn:Adam - lr:0.001 - e:10	24-12-20_924692
Res:241930	0.9918	07:50,0	f_l1:61 - f_l2:32 - drp:0.32349630192554335 - fi_lay:199	op_fn:Adam - lr:0.001 - e:10	24-12-20_241930
Res:346088	0.9915	06:03,7	f_l1:33 - f_l2:42 - drp:0.2667417222278044 - fi_lay:108	op_fn:Adam - lr:0.001 - e:10	24-12-20_346088
Res:504133	0.9913	03:51,9	f_l1:9 - f_l2:54 - drp:0.3366527037650917 - fi_lay:71	op_fn:Adam - lr:0.001 - e:10	24-12-20_504133
Res:269117	0.9911	07:36,0	f_l1:53 - f_l2:50 - drp:0.31937003158929744 - fi_lay:163	op_fn:Adam - lr:0.001 - e:10	24-12-20_269117
Res:712399	0.9896	06:13,9	f_l1:42 - f_l2:33 - drp:0.3049549320516779 - fi_lay:34	op_fn:Adam - lr:0.001 - e:10	24-12-20_712399
Res:085973	0.9892	05:10,6	f_l1:16 - f_l2:94 - drp:0.344399754453365 - fi_lay:40	op_fn:Adam - lr:0.001 - e:10	24-12-20_085973

Parte 2: Modificaciones en el Entrenamiento - Cambio de hiperparámetros

Solo nos piden cambiar dentro de los distintos tipos de algoritmos de optimización, y el learning rate. **Dejando fijo batch= 128, y épocas = 10**

Con el resultado de la mejor combinación de parámetros donde fijamos

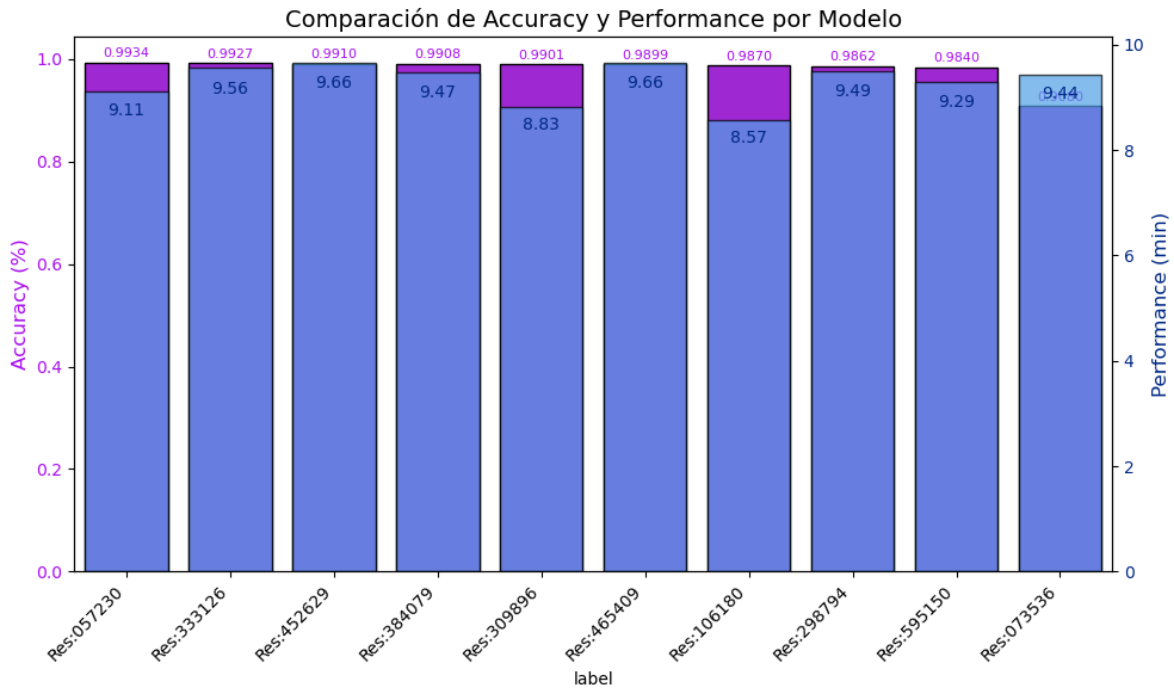
{'filters_l1': 34, 'filters_l2': 115, 'final_layer_size': 146, 'dropout': 0.2900998503939086}

Algoritmo de optimización	Tasa de aprendizaje (learning rate)
Adam	**0.1**
SGD	**0.01**
RMSprop	**0.001**
Adagrad	**0.0001**

Aquí también se usó la Optimización Bayesiana incluso con categorías para definir el algoritmo de optimización. Con este algoritmo deducimos a 1:45 minutos la ejecución para 10 ciclos.

También lo guardamos en un archivo YAML mientras procesaba los 10 ciclos en la optimización Bayesiana, donde luego de procesar los datos, lo rescatamos desde este archivo para poder graficarlos y validar los resultados y ver cómo se comportaron con las otras combinaciones de los hiperparámetros.

Tiempo de ejecución Op.Bayesiana para hiperparámetros fue: 93.80 minutos
Mejor combinación de hiperparámetros:
{'learning_rate': 0.0007792297153882995, 'optimizer_str': 'Adam'}



label	accuracy	Performance en min	parametros	hiperparametros	timestamp
Res:057230	0.9934	09:06,6	f_l1:34 - f_l2:115 - drp:0.2900998503939086 - fi_lay:146	op_fn:Adam - lr:0.0007792297153882995 - e:10	24-12-20_057230
Res:333126	0.9927	09:33,3	f_l1:34 - f_l2:115 - drp:0.2900998503939086 - fi_lay:146	op_fn:Adam - lr:0.0016722697006183672 - e:10	24-12-20_333126
Res:452629	0.991	09:39,8	f_l1:34 - f_l2:115 - drp:0.2900998503939086 - fi_lay:146	op_fn:RMSprop - lr:0.0001930783753654713 - e:10	24-12-20_452629
Res:384079	0.9908	09:28,5	f_l1:34 - f_l2:115 - drp:0.2900998503939086 - fi_lay:146	op_fn:RMSprop - lr:0.00011120513715710631 - e:10	24-12-20_384079
Res:309896	0.9901	08:49,7	f_l1:34 - f_l2:115 - drp:0.2900998503939086 - fi_lay:146	op_fn:Adam - lr:0.003918194347141745 - e:10	24-12-20_309896
Res:465409	0.9899	09:39,4	f_l1:34 - f_l2:115 - drp:0.2900998503939086 - fi_lay:146	op_fn:RMSprop - lr:0.00012966511753760403 - e:10	24-12-20_465409
Res:106180	0.987	08:33,9	f_l1:34 - f_l2:115 - drp:0.2900998503939086 - fi_lay:146	op_fn:RMSprop - lr:0.00362561763457623 - e:10	24-12-20_106180
Res:298794	0.9862	09:29,5	f_l1:34 - f_l2:115 - drp:0.2900998503939086 - fi_lay:146	op_fn:Adam - lr:0.007535384509295551 - e:10	24-12-20_298794
Res:595150	0.984	09:17,3	f_l1:34 - f_l2:115 - drp:0.2900998503939086 - fi_lay:146	op_fn:RMSprop - lr:0.009647685075720105 - e:10	24-12-20_595150
Res:073536	0.908	09:26,3	f_l1:34 - f_l2:115 - drp:0.2900998503939086 - fi_lay:146	op_fn:SGD - lr:0.0008288916866885144 - e:10	24-12-20_073536

También esta con CUDA en la performance, pero aún así se demoró bastante en procesar, pero el accuracy es del 99.34 %

Creo que debí dejar el learning rate como valor categórico en la Optimización Bayesiana, para mejores valores.

2.3 Compare el rendimiento, numero de parámetros y tiempo de ejecución.

Personalmente encuentro que el tipo de ejecución para esto es larguísimo, traté de configurarlo de tal manera que fuera todo automatizado, para no estar de uno esperando cambiando los parámetros e hiperparámetros y visualizando con gráficos el resultado para que fuera más ilustrativo.

La cantidad de parámetros que se le pasa la MLP es mucho mayor, lo que hace que la cantidad posible de combinaciones de éstas es mayor. Ahora el CUDA ayuda muchísimo a la rapidez de estas pruebas, donde modelos que se demoraban 8 minutos en promedio o más, después fue de 2 minutos. Puse ambos gráficos para su compresión y validación visual de ambas arquitecturas de redes neuronales.

Lo analizaré por Rendimiento de Accuracy

En MNIST, un MLP tiene una precisión entre 97% y 98%, dependiendo de la cantidad de capas, neuronas y regularización. Una CNN puede superar fácilmente al MLP en precisión, alcanzando valores cercanos al 99%. La CNN ofrece un mejor rendimiento en MNIST gracias a su capacidad para explotar las propiedades espaciales de las imágenes, mientras que un MLP depende más de la cantidad de neuronas.

Ahora por número de Parámetros

Los parámetros en un MLP dependen directamente del tamaño de las capas y del número de neuronas, donde el número de parámetros crece rápidamente con el tamaño de las capas, mientras que las CNN pueden compartir capas, o algo así. Donde el uso eficiente de convoluciones y pooling hace que el número de número de parámetros significativamente menor para la misma tarea.

Tiempo de Ejecución

El entrenamiento de un MLP puede ser más rápido porque no realiza operaciones convolucionales. Su tiempo de ejecución depende principalmente del tamaño del modelo y el número de épocas.

Sin embargo, el alto número de parámetros puede hacer que tarde más en converger en algunos casos, en mi caso mi computador se tardó muchísimo lo que me ha dado tiempo para realizar esta documentación.

Se indica que las operaciones convolucionales son computacionalmente más intensivas que las multiplicaciones de matrices usadas en un MLP. Esto puede hacer que una CNN tome más tiempo por época en comparación con un MLP. Pero como tiene un menor número de parámetros y la mejor capacidad de generalización pueden reducir el tiempo necesario para alcanzar una buena precisión.

Por otro lado, la combinación de hiperparámetros por la Optimización Bayesiana, fue un gran alcance, porque fije que sea sólo 10 iteraciones aun así se demoró mas de 90 minutos en CNN.

Conclusión sobre el tiempo de ejecución: El MLP puede ser más rápido por época, pero la CNN puede necesitar menos ajustes (como regularización) para alcanzar un mejor rendimiento, lo que compensa el tiempo adicional.

Conclusión General

Rendimiento: La CNN supera al MLP en precisión gracias a su capacidad para trabajar con datos espaciales. La CNN es más eficiente, utilizando menos parámetros para lograr un mejor rendimiento. Aunque las CNN pueden ser más lentas por época, son más rápidas en converger hacia un modelo óptimo.

Yo en tareas pequeñas usaría MLP pero en las que son más complejas que requieren mayor robustez usaría CNN. Si tuviera que hacer esto profesionalmente, pagaría el Google Colab Pro, porque el computador se queda muy pegado incluso procesando ciclos de 10 modelos para CNN y 16 para MLP.