DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico

Reconocimiento de imágenes con Fashion MNIST

Métodos Numéricos
Segundo Cuatrimestre de 2021

Integrante	LU	Correo electrónico
Marco Sánchez Sorondo	708/19	msorondo@live.com.ar

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

Resumen

Se realizó un análisis de los datos de FASHION MNIST. Luego, se realiza una búsqueda exhaustiva de la mejor combinación de k de kNN con α de PCA, considerando métricas como accuracy, F1, tiempo de ejecución, recall y precision. Posteriormente se procede a probar el uso de 2 métodos no lineales para reducir la dimensión de los datos: t-SNE y UMAP.
Luego se implementan diversos tipos de autoencoders para encontrar uno que ofrezca el mejor balance entre accuracy y tiempo de clasificación con kNN. Finalmente, se prueban 2 clasificadores puramente basados en redes neuronales.

Palabras clave: Clasificación, KNN, PCA, dataset, covarianza, modelo, gradiente, entrenamiento, accuracy, autoencoders, aprendizaje profundo

1. Introducción

El reconocimiento automático de imágenes es un área prolífera dentro de la informática, que gracias a los avances en el poder de cómputo de los procesadores permite la ejecución de algoritmos cada vez más potentes. Las técnicas más comúnmente utilizadas en la actualidad están basadas en el entrenamiento sobre datos para lograr cierto 'aprendizaje' sobre sus características (comúnmente conocido como 'Machine Learning'). Si bien la posibilidad de ejecutar algoritmos cada vez más potentes y sofisticados nos permite mejorar la presición de la clasificación de una imagen, el costo tanto temporal como de memoria para llevar ésto a cabo es relativamente alto, por ello, entran en juego técnicas de pre-procesamiento de los datos, utilizadas para mantener la misma información en estructuras más compactas y por lo tanto menos costosas de procesar. Para ello, en éste trabajo se evalúan diversos métodos de reducción de la dimensionalidad sobre imágenes de prendas de ropa, más específicamente, el dataset FASHION MNIST. FASHION MNIST es un dataset compuesto por 60.000 imágenes de entrenamiento, y 10.000 imágenes de validación. Cada imágen se representa como un vector de 784 componentes, donde cada componente representa la intensidad de un pixel en escala de grises. Cada imágen viene etiquetada con su clasificación correcta.

La transformación de los datos no sólo puede derivar en una mejor compresión de la información sino también en una mejora en la interpretación de los datos y por consiguiente una mejora al clasificar con respecto a los datos originales. Por ello, en éste trabajo también se implementan técnicas de clasificación con redes neuronales convolucionales, en donde los algoritmos llevan a cabo transformaciones (convoluciones) para hallar nuevas características en los pixeles y 'exagerar' las más influyentes a la hora de clasificar.

Este trabajo es una continuación de **otro llevado a cabo sobre el set de datos MNIST**, muchos de los conceptos nombrados y de las técnicas implementadas se explican allí en detalle (y se avisará cuando así sea). También se aprovechan muchos de sus resultados para tomar algunas decisiones en el presente trabajo.

2. Métodos

2.1. KNN

El principal algoritmo que utilizaremos será 'K Nearest Neighbors'. Báscamente, dada una imagen de entrada, busca las k imágenes tales que la suma de las diferencias absolutas entre pixeles es mínima. De éstas k imágenes, se toma la clase más frecuente (moda) y se la retorna como clase de la imágen de entrada.

Para más detalle, véase el trabajo anterior. [3]

2.2. PCA, T-SNE y UMAP

Muchas veces, los conjuntos de datos ocupan grandes volúmenes de memoria cuyo costo de procesamiento es muy alto como consecuencia de la gran cantidad de características a analizar (en éste caso, 784 pixeles por imágen). Toda ésta información es probablemente redundate, y existen métodos para hallar las verdaderas características subyacentes en cada clase a clasificar, y éstas características consideran la relación entre varios pixeles.
Estos tres métodos tienen como fin reducir la dimensión del conjunto de datos, hallando características que sinteticen la información presente en los pixeles y por consiguiente el costo de procesamiento a la hora de clasificar sea menor. La principal diferencia entre PCA con T-SNE y UMAP es que éstos últimos dos aplican transformaciones no lineales sobre el conjunto de datos. La limitación principal de PCA es justamente que sólo es capaz de hallar relaciones lineales entre los pixeles, y por ello se acude a éstas otras dos técnicas.

2.2.1. PCA

PCA disminuye la dimensión de los datos mediante la creación de características de redundancia mínima mediante una transformación de las muestras características (columnas) son combinaciones lineales de los pixeles de la imagen de original. Esto se lleva a cabo hallando la matriz de covarianzas entre las características de las imágenes (columnas de la matriz de la muestra), diagonalizándola para que las covarianzas sean cero (idealmente).

Para más detalle, véase el trabajo anterior. [3]

2.2.2. t-distributed Stochastic Neighbor Embedding (T-SNE)

Intuición débil El algoritmo proyecta aleatoriamente las muestras en una dimension más baja (espacio latente), luego para cada punto proyectado, se ajusta su posición atrayéndolo a los puntos de distancia más cercana en el espacio original y repeliéndolo de aquellos más lejanos.

Intuición fuerte

1. Para cada punto del espacio original, se calcula su 'similitud normalizada' [1] con el resto de los puntos.
2. Se toma la matriz de similitudes entre puntos.
3. Se proyectan las muestras de entrada aleatoriamente sobre el espacio latente.
4. Se repite el proceso anterior, pero utilizando las distancias en el espacio latente y midiendo la similitud con una distribución t de Student.
5. Se calcula la matriz de similitudes correspondiente al paso anterior.
6. Se mueven los puntos proyectados de manera tal que la nueva matriz de similitudes resulte cada vez más parecida a la original. La dirección en la que se mueven los puntos es la que minimiza la divergencia de Kullback-Leibler, utilizada como función de costo.

Formalmente Se citan las fuentes correspondientes para mayor detalles matemático [2][4].

2.2.3. Uniform Manifold Approximation and Projection (UMAP)

Intuición A grandes rasgos, el método es el mismo que T-SNE, pero con una definición distinta de distancia (en éste caso, no euclídea), una definicion distinta de 'similitud' y minimiza una función de costo distinta.

Formalmente Dado que la complejidad matemática de los términos utilizados para el método escapan al propósito del presente trabajo, se delegan su explicación a las citas bibliográficas correspondientes [1].

2.3. Redes Neuronales

Una red neuronal no es más que una forma de conceptualizar la aplicación sucesiva de funciones sobre una entrada (muestra) con el objetivo de obtener una salida. En general, ésta salida puede ser tanto un valor numérico cuantitativo (regresion), como una clase (clasificación) como un conjunto de coheficientes (que representan a la entrada de una forma alternativa), entre otras cosas. Se denomina 'pesos' a los coheficientes de las funciones aplicadas que se modifican con el objetivo de obtener un 'mejor' resultado (ya veremos que la definición de 'mejor' es definida por quien diseña la red), y 'entrenamiento' al proceso de búsqueda de pesos que 'mejor' salida producen.

2.3.1. Estructura y funcionamiento genérico

Unidad linear Dada una muestra m con valores m_i con $i = 1, 2, ...n$, una unidad linear $l(x) = w_1x_1 + w_2x_2 + ... + w_nx_n + b$ es una función que calcula una la combinación lineal de la entrada $x \in \mathbb{R}^n$ mas una constante b.

[1] La similitud entre dos puntos se calcula tomando un punto de la muestra original, luego centrandolo en una distribución normal, colocando al segundo punto en el eje x de la distribución a exactamente la distancia euclidea que tiene con el punto original, y calculando el valor de la densidad en dicho punto.

Llamamos a los $w_i \in \mathbb{R}$ 'pesos', a b sesgo y a la unidad linear 'neurona' o 'unidad'. Obsérvese que, bajo ésta definición restringida de red neuronal, una red neuronal sólo puede establecer relaciones lineales entre sus entradas.

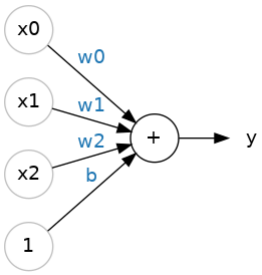


Figura 1: Unidad lineal con n=3 entradas.

Capas Una red neuronal puede componerse de multiples capas, es decir, una misma entrada x puede estar conectada a múltiples neuronas, y a su vez la salida de las neuronas pueden servir como entrada para otras. Esto otorga la posibilidad de otorgarle una mayor complejidad al modelo, lo cual generalmente deriva en un mayor rango de funciones representables y por consiguiente en la capacidad de detectar y modelar patrones de diversas naturalezas. A las capas intermedias se las llama 'ocultas'.

Activación Para poder desarrollar modelos no-lineales, se incorpora una función de activación. Una función de activación no es nada mas que una función arbitraria aplicada a la salida de la neurona. De ésta manera, la combinación resultante de la neurona incorpora la nolinearidad.
La función de activación más comúnmente usada, es ReLU (por Rectified Linear Unit), que aplica la siguiente función:

$$f(y) = \begin{cases} y, & \text{si } y \geq 0 \\ 0, & \text{caso contrario} \end{cases}$$

Otras funciones muy comunes son $tanh(y)$ y $\frac{1}{1+e^{-x}}$.

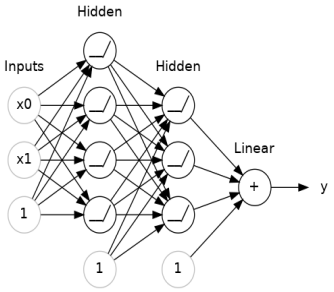


Figura 2: Una red profunda con 2 capas ocultas.

Aprendizaje ¿Cómo se ajustan los pesos w_i de manera tal que el modelo sea preciso? En la etapa de entrenamiento se ejecuta un algoritmo de optimización que maximiza la performance del modelo medida en una **función de pérdida** que calcula el error del modelo. El algoritmo muás comúnmente utilizado se llama 'descenso de gradiente' y tiene una lógica algo similar a los métodos de t-SNE y UMAP.

Descenso de gradiente

Dados una red R de pesos $w_1, w_2, ..., w_\alpha$ (de todas las capas) un conjunto de muestras M de tamaño n cada una, un conjunto de particiones de $M = \{P_1, P_2, ..., P_k\}$ donde $\#P_i = p \ \forall i = 1, ..., n$ y una función de pérdida $fp : \mathbb{R}^n \longrightarrow \mathbb{R}$...

1. Se les asigna a los pesos un valor aleatorio.
2. Se calcula $R(P_i)$ (abuso de notación: se calcula la red en funcion de todas las entradas de una partición).
3. Se calcula $fp(R(P_i))$ la función de pérdida sobre todas las predicciones del paso anterior.
4. Se calcula el gradiente de fp en el punto $w_1, w_2, ..., w_\alpha$.
5. Se actualizan $w_1, w_2, ..., w_\alpha$ en el sentido decreciente del gradiente[2].
6. Repetir desde 2. para el resto de las particiones.
7. Repetir todo el proceso desde 2. unas $e - 1$ veces más, donde e es el **número de épocas**.

Nota: No todos los algoritmos de optimización son iguales, y en la práctica generalmente se implementan variantes del recien descripto.

[2]El módulo del desplazamiento de los pesos es un parámetro establecido a priori y se lo denomina **ratio de aprendizaje.**

2.3.2. Autoencoders

Notar que el proceso descripto anteriormente asume que los valores 'correctos' para cada muestra están dados, osea que es un modelo supervisado de aprendizaje. ¿Cómo podemos aplicar ésto para reducir la dimensión de las entradas, considerando además que ésta es una tarea no supervisada? Los autoencoders se utilizan para ello, y para explicarlo primero es necesario instanciar y mostrar una imágen de la arquitectura de un autoencoder.
Un autoencoder se compone de una entrada de tamaño k y una salida de igual tamaño, y un conjunto de capas intermedias en donde la de menor dimensión se llama **cuello**. Comúnmente la red es simétrica y la capa intermedia es la de menor dimensión como se muestra en la figura 3.

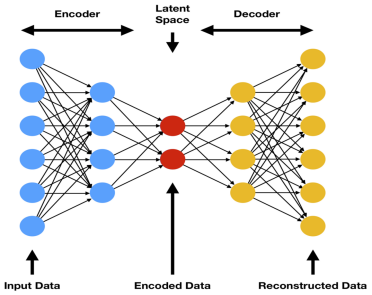


Figura 3: Un autoencoder de espacio latente con dimensión 2 y 3 capas ocultas.

Método: Al entrenar el autoencoder, se determina como valor deseado a la misma entrada de la red, es decir, se busca que dada una muestra m_i, siendo la red R, que los pesos sean tales que $R(m_i) = \hat{m}_i$ con $m_i \approx \hat{m}_i$, idealmente $m_i = \hat{m}_i$.
Para entender por qué esto funciona, dividamos en 2 a la red R: R_{enc} (el modelo que tiene como salida el espacio latente) y R_{dec} (el modelo que tiene como entrada el espacio latente)... Esto significría que
$R(m_i) = R_{dec}(R_{enc}(m_i)) = \hat{m}_i$ lo cual implica que existe un conjunto de pesos para los cuales al reducir la dimension al espacio latente con R_{enc}, podemos recuperar una imágen aproximadamente igual aplicando R_{dec}, y por lo tanto al entrenar a la red, se está maximizando la capacidad de R_{enc} para preservar la información de la entrada en el espacio latente.

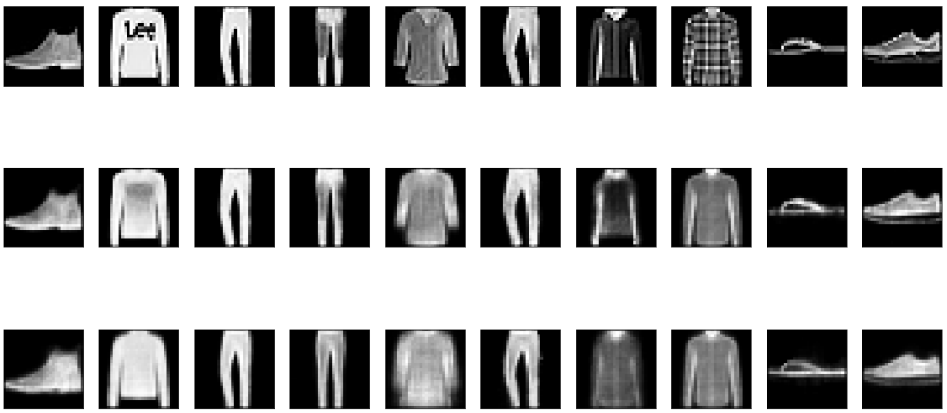


Figura 4: (De experimento 1 de Autoencoders) Imágenes originales, luego habiéndoles aplicado autoencoders en espacios latentes intermedios 15 y 5 respectivamente. Obsérvese cómo se conserva la información, pero con menor calidad a menor espacio latente intermedio.

2.4. K-Fold Cross Validation

En el presente trabajo, dado el alto costo temporal que implica la ejecución de los algoritmos y el alto volumen de datos con el que se trabaja, se decidió no acudir a ésta técnica como validación de los resultados. Vale la pena aclarar que, hay ciertas características del conjunto de datos (evidenciadas en la experimentación) que nos permiten prescindir (no totalmente) de éste algoritmo.

3. Experimentación

3.1. Introducción

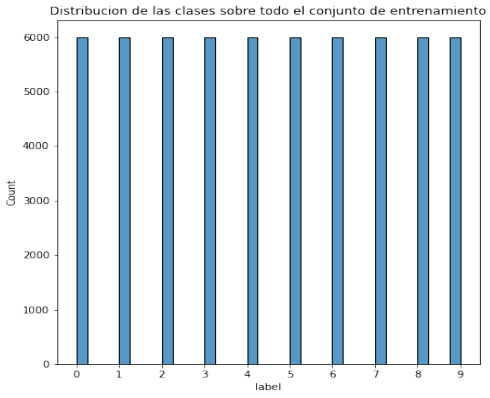
En esta sección vamos a realizar una serie de experimentos con varios objetivos:

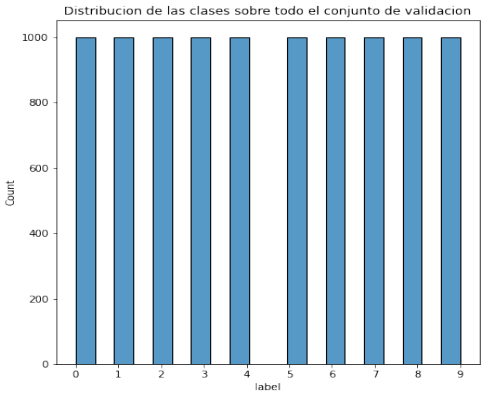
- Encontrar el par de hiperparámetros que maximizan el rendimiento y calidad de los resultados.
- Medir y comparar el rendimiento y calidad de kNN usando o no PCA, con un amplio rango de parámetros.
- Analizar la calidad de los resultados de kNN con PCA para un rango amplio de imágenes para encontrar la situación ideal de rendimiento y accuracy.

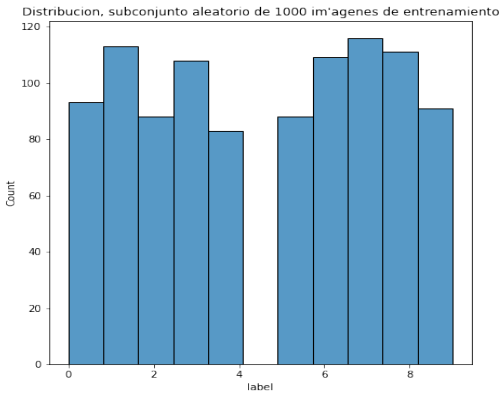
3.2. Experimentación básica

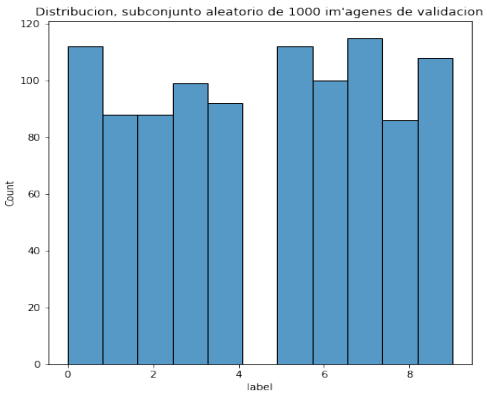
Procesador: **Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 8GB RAM**.

En éste paso se analizaron algunos aspectos sobre el conjunto de datos y se hizo una prueba de uso de algunos de los métodos a utilizar posteriormente (KNN, PCA y métricas a evaluar).









Principales conclusiones:

- Las clases del conjunto de entrenamiento y de validacion están ambas distribuidas uniformemente.
- Las clases del conjunto de entrenamiento y de validacion están desordenadas de forma aleatoria.
- Tomar un subconjunto del dataset como base para reducir el tiempo de entrenamiento probablemente no valga la pena, las disminuciones en el tiempo entrenamiento con 1/6 del conjunto (10.000 imágenes) no son considerables (60.000 imágenes tomó aproximadamente 1.28 segundos con $k, \alpha = 10, 10$ mientras que con 10.000 imágenes aproximadamente 0.92 segundos).

3.3. Mejor combinación k, α

Procesador: **Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 8GB RAM**.
El objetivo de éste experimento es el de hallar la combinacion óptima de parámetros k y α correspondientes a los algoritmos KNN y PCA respectivamente. Para ello se crearon 2 scripts: classifiers.py y best_k_alpha.py, donde el primero se encarga de tomar el dataframe y dos instancias de los parámetros y realiza un 'diagnóstico' del rendimiento del modelo haciendo un split 80/20 de entrenamiento y el segundo se encarga de probar con todas las combinaciones posibles para k en [1,3,5,...,49] y α en [1,3,5,...,49]. Se guardan los resultados para cada metrica en un CSV distinto (por ejemplo, en accuracy.csv se guarda un dataframe que muestra en la posicion i, j el accuracy para $k = i$ y $\alpha = j$). Es de esperar que a mayor α, mayor es el tiempo que tarda el algoritmo en entrenarse. Es por ello que primero seleccionaremos un α a partir del cual las metricas no mejoren considerablemente, manteniendo un tiempo de ejecucion considerablemente mejor que entrenar el modelo en dimensiones superiores.

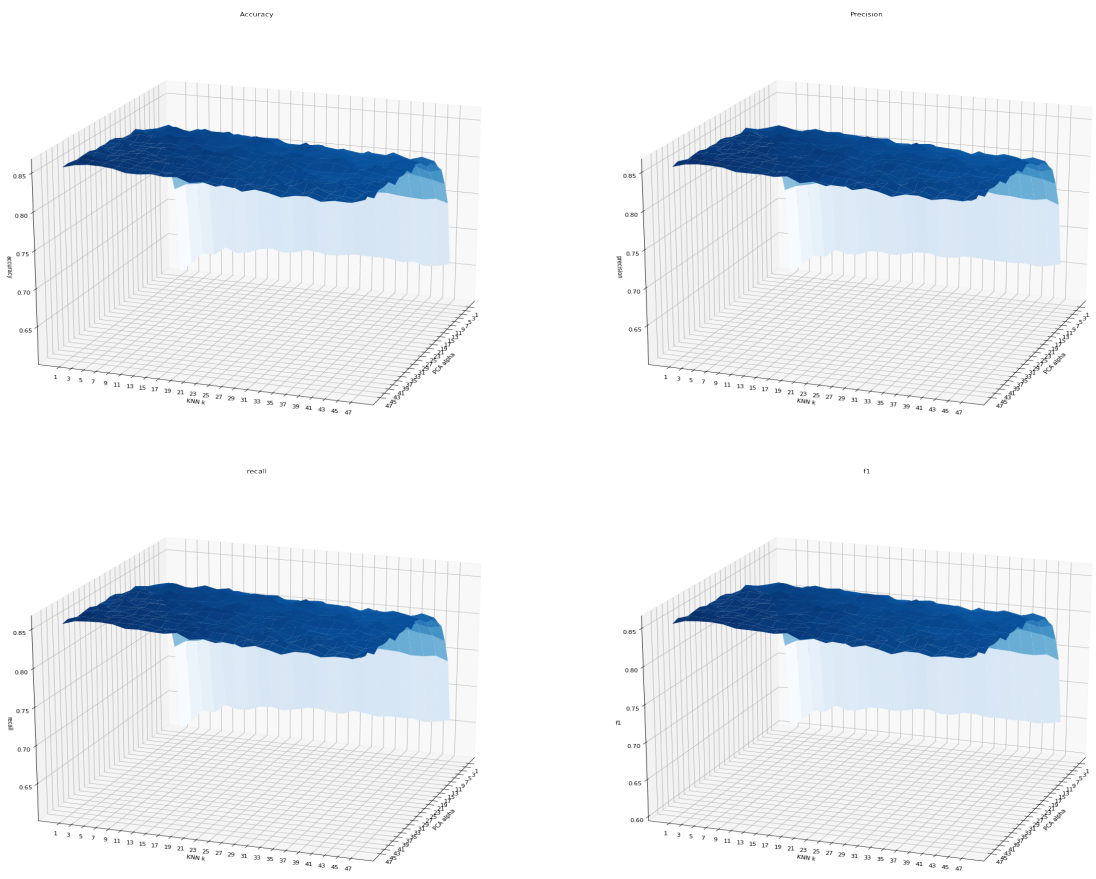


Figura 6: Variación de las metricas con respecto al par k, α

Resultados relevantes

- La variación de los errores para las metricas son prácticamente los mismos. Por ésto, de ahora en más optaremos por utilizar únicamente a accuracy como medida del error de clasificación. Esta desición no está

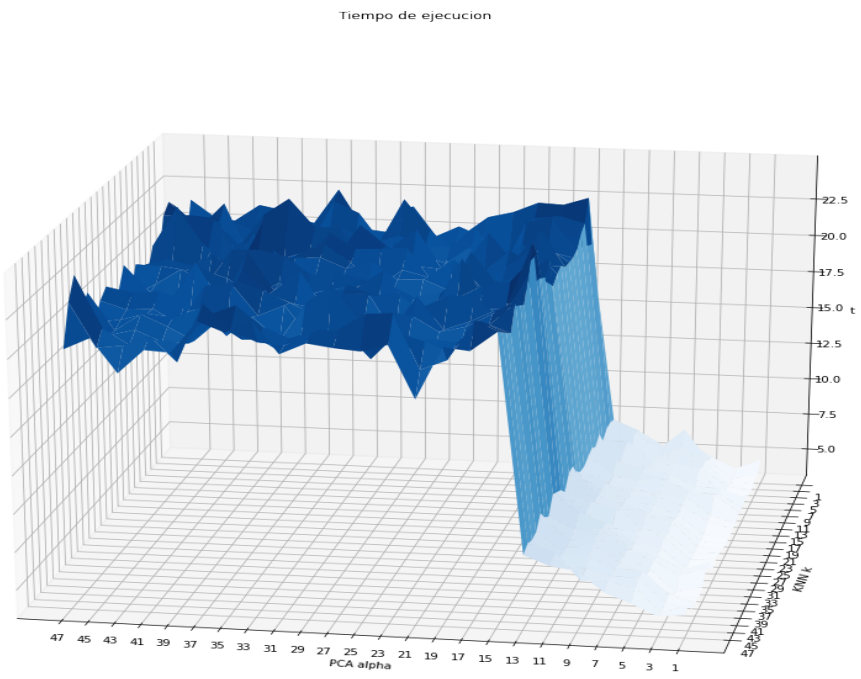


Figura 7: Variación del tiempo de ejecucion con respecto al par k, α

respaldada únicamente por estos resultados, sino que además considera los resultados del análisis exploratorio (uniformidad de las clases), y que no hay un deseo de detectar una clase en particular (no hay una clase donde acertar sea mas 'importante' que el resto).

- El impacto del k de KNN en el tiempo influye muy levemente, y a partir de $k = 9$ parece dejar de disminuir el tiempo.
- Entre $\alpha = 13$ y $\alpha = 15$ se produce un aumento drástico en el tiempo de ejecución de KNN. Es por ello que elegimos el par $(k, \alpha) = (15, 15)$, que preserva lo mejor de los dos mundos (accuracy-tiempo).

3.4. T-SNE y UMAP

Procesador: **Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz, 8GB RAM**.
Inicialmente, la idea era poder utilizar éstos dos métodos para poder mejorar la capacidad del clasidicador, y poder encontrar parámetros óptimos para poder transformar los datos de manera tal que el rendimiento mejore con respecto a PCA, pero dados los altos tiempos de ejecucion que insumía su entrenamiento se decidió no continuar con ésta experimentación. Otro aspecto que influyó también en ésta decision es la poca intuitividad del funcionamiento de éstos métodos, comprenderlos correctamente hubiera insumido bastante tiempo que se invirtieron en otros métodos. Se presentan algunos resultados obtenidos junto a observaciones para los parámetros por defecto de éstos modelos...

Observaciones

- Los tiempos de clasificación se ven reducidos drásticamente (todos fueron menores a 0.05 seg.).
- Todos los modelos logran, de alguna manera, llevar a cabo agrupaciones bastante razonables (considerando que son no supervisados).

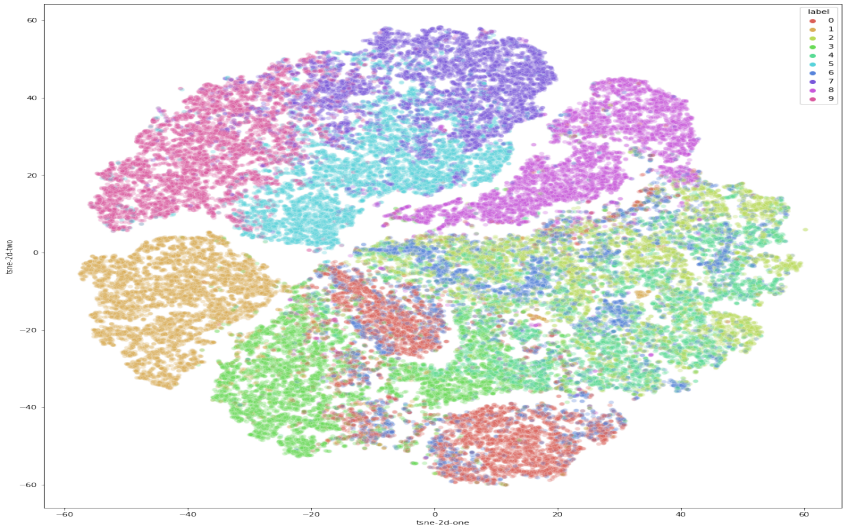


Figura 8: 2d T-SNE (default), 535 seg. entre entrenamiento y transformación

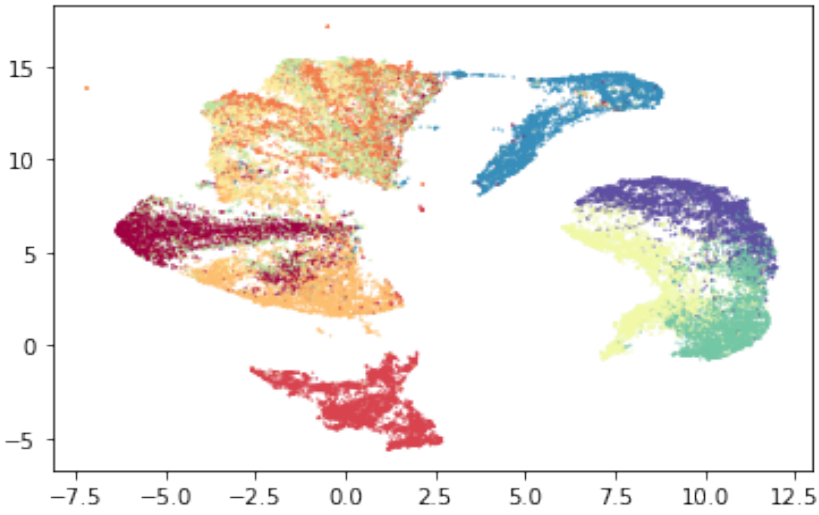


Figura 9: 2d UMAP n_neighbors=5, 43 seg. entre entrenamiento y transformación, KNN accuracy 0.75

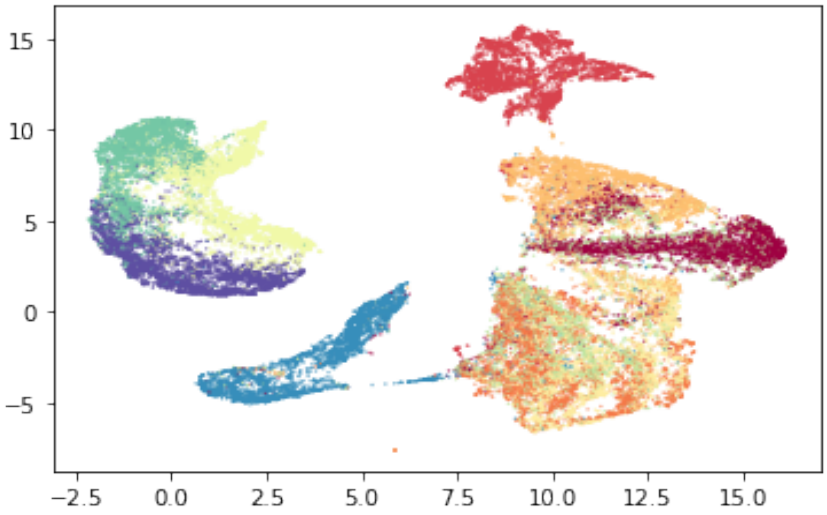


Figura 10: 2d UMAP (default), 65 seg. entre entrenamiento y transformación, KNN accuracy 0.32

3.5. Redes Neuronales

Procesador: **Tesla P100-PCIE-16GB (GPUs de Kaggle)**.

3.5.1. Autoencoders

El objetivo de ésta parte es también de reducir la dimensión del dataset, pero con autoencoders. La idea principal es ver cómo varía la evoluciona en la compresión de la información para distintos espacios latentes, distintas profundidades y distintas configuraciones de las capas. Idealmente, se busca obtener un rendimiento mejor que PCA (que a mismas dimensiones logre mejorar el accuracy de KNN o que a dimensiones menores logre obtener métricas similares).

Experimento 1 Se entrenaron autoencoders de dimension 5, 15, 25, 50, 75 y 100 para diversas profundidades, con función de activación 'ReLU' y optimizador 'ADAM'.

Hipótesis

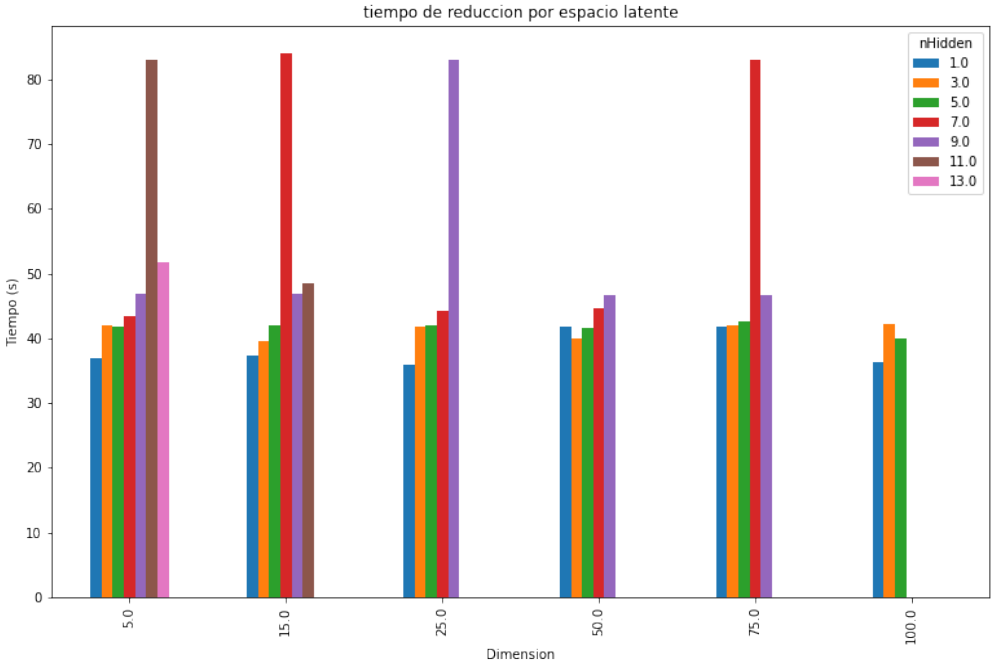
- Es de esperar que a menor dimensión, peor rendimiento de KNN para una misma profundidad (generalmente).
- A menor dimension reducida, mejor tiempo de ejecución para KNN (entrenamiento + clasificación).
- A menor profundidad de la red, menor tiempo de entrenamiento del autoencoder.
- A mayor profundidad, mejor conservación de la informacion.

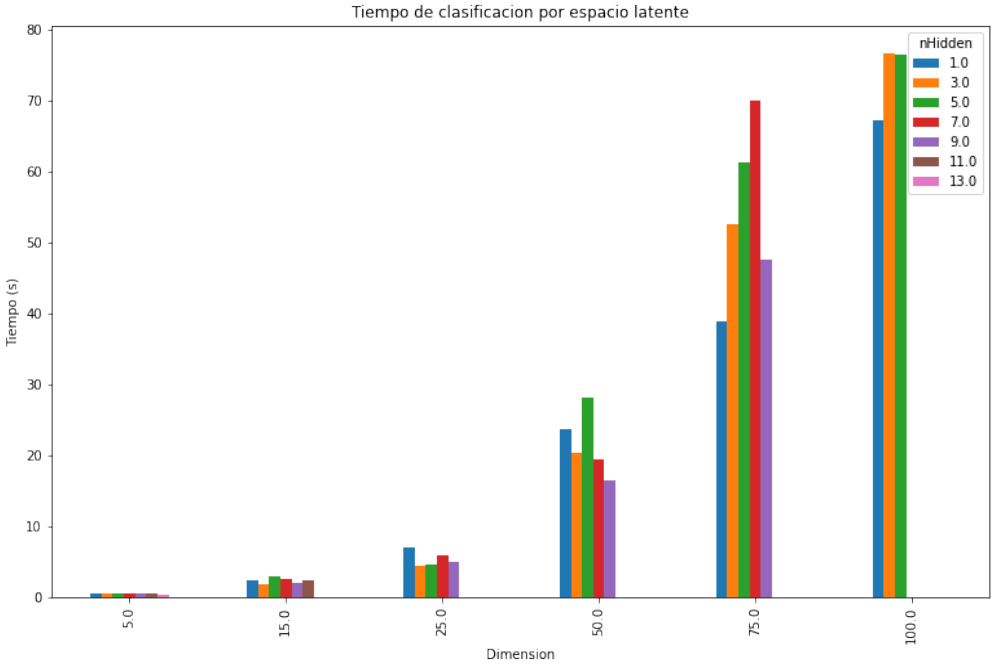
Resultados

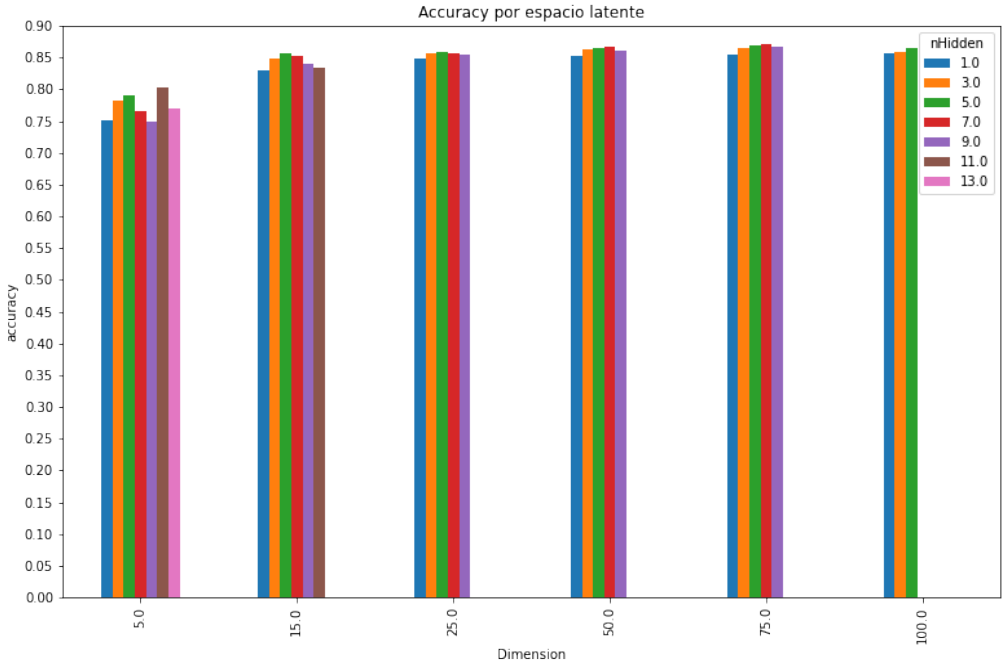
- El espacio latente parece no influir en el tiempo de entrenamiento del autoencoder (leves bajadas al subir la dimension).
- Mayor profundidad tiende a un mayor tiempo de entrenamiento del autoencoder (con un par de excepciones).
- A mayor espacio latente, leve aumento en el accuracy (osea que se logro disminuir la dimension significativamente preservandose la mayoria de la informacion).
- No parece haber una relacion directa entre profundidad de la red y la variacion del accuracy.
- Drasticas reducciones en el tiempo de clasificacion.

	Unnamed: 0	nHidden	latent_dim	t_classif	t_red	acc
0	0	1.0	5.0	0.519382	36.969881	0.7506
1	1	3.0	5.0	0.508082	41.924303	0.7832
2	2	5.0	5.0	0.502758	41.828052	0.7898
3	3	7.0	5.0	0.485336	43.488816	0.7666
4	4	9.0	5.0	0.465511	46.949082	0.7485
5	5	11.0	5.0	0.511894	83.074777	0.8029
6	6	13.0	5.0	0.436467	51.808351	0.7703
7	7	1.0	15.0	2.478351	37.325002	0.8294
8	8	3.0	15.0	1.787461	39.496266	0.8488
9	9	5.0	15.0	2.915833	41.975359	0.8570
10	10	7.0	15.0	2.674354	84.083234	0.8528
11	11	9.0	15.0	2.010873	46.957710	0.8402
12	12	11.0	15.0	2.355462	48.518561	0.8335
13	13	1.0	25.0	7.002624	35.899415	0.8475
14	14	3.0	25.0	4.464043	41.903311	0.8574
15	15	5.0	25.0	4.683609	42.016869	0.8581
16	16	7.0	25.0	5.915808	44.167066	0.8574
17	17	9.0	25.0	4.979187	83.039436	0.8544
18	18	1.0	50.0	23.635928	41.854524	0.8536
19	19	3.0	50.0	20.407004	40.053424	0.8632
20	20	5.0	50.0	28.124051	41.628927	0.8652
21	21	7.0	50.0	19.403607	44.604663	0.8664
22	22	9.0	50.0	16.479418	46.616480	0.8610
23	23	1.0	75.0	38.836747	41.847050	0.8546
24	24	3.0	75.0	52.503715	42.092988	0.8643
25	25	5.0	75.0	61.300675	42.707890	0.8684
26	26	7.0	75.0	69.892645	82.993826	0.8715
27	27	9.0	75.0	47.554794	46.709029	0.8679
28	28	1.0	100.0	67.112664	36.253248	0.8563
29	29	3.0	100.0	76.678835	42.135509	0.8589
30	30	5.0	100.0	76.476752	39.964832	0.8657

Figura 11: Tabla de características de los modelos con sus resultados


tiempo de reduccion por espacio latente
nHidden
1.0
3.0
5.0
7.0
9.0
11.0
13.0
Tiempo (s)
0
10
20
30
40
50
60
70
80
5.0
15.0
25.0
50.0
75.0
100.0
Dimension


Tiempo de clasificacion por espacio latente
nHidden
1.0
3.0
5.0
7.0
9.0
11.0
13.0
Tiempo (s)
0
10
20
30
40
50
60
70
80
5.0
15.0
25.0
50.0
75.0
100.0
Dimension



Experimento 2 Se crean modelos de igual profundidad y espacio latente, pero con capas de distinta dimensión. Se desea saber el impacto de éstas variaciones en la capacidad de reducción.
Hipótesis

- Agregar capas de dimensión intermedia-baja con respecto a la dimensión de las imágenes tiene un mejor impacto en el accuracy que agregar capas de dimensiones mas cercanas a la entrada (784).
- Agregar

Detalles de los modelos: Profundidad: 17 capas ocultas.
Dimensiones de Capas por modelo:
$[400, 350, 300, 250, 200, 150, 100, 50, 25, 15, 25, 50, 100, 150, 200, 250, 300, 350, 400]$,
$[300, 250, 200, 150, 75, 40, 30, 25, 20, 15, 20, 25, 30, 40, 75, 150, 200, 250, 300]$ y
$[200, 150, 100, 75, 40, 30, 25, 20, 17, 15, 17, 20, 25, 30, 40, 75, 100, 150, 200]$.
Llamaremos 'high-dim', 'mid-dim' y 'low-dim' a cada modelo respectivamente (refieren a si en la red predominan capas de alta, media o baja dimensión).
Optimizador: ADAM.

Resultado

```
high-dim    0.8123
mid-dim     0.7963
low-dim     0.7547
dtype: float64
```

Figura 12: Accuracy por predominancia de dimensiones por capa

Análisis: El resultado fue exactamente el contrario al esperado. La intuición de quien experimentaba era que a medida que se va reduciendo la dimensión, se vuelve más dificil conservar la informacion, y que por ello sería más conveniente concentrarse en ésa zona. Luego de éste resultado, podemos ver que las primeras capas de reducción, más cercanas al espacio de entrada tienen un impacto mayor en la conservación que aquellas más cercanas al cuello del modelo.

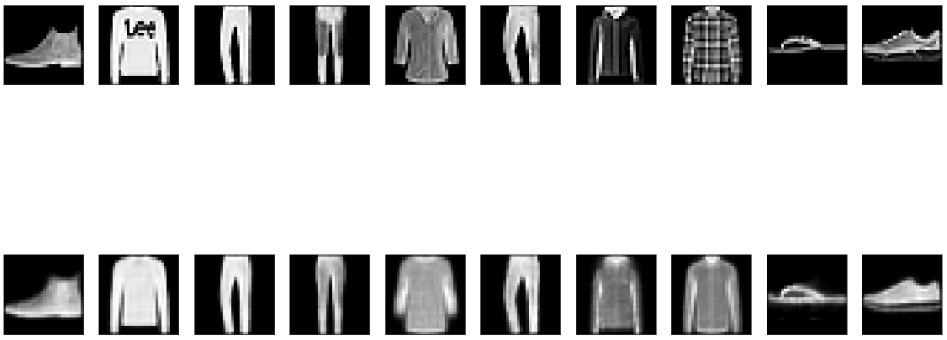


Figura 13: Imágenes originales vs. Imágenes pasadas por el autoencoder 'high-dim'.

3.5.2. Clasificador neuronal simple vs profundo

Procesador: **Google TPU v3-8 (Kaggle)**.
Se busca un accuracy aún mejor que los encontrados anteriormente, para ello se entrenaron 2 clasificadores 100 % con redes neuronales, uno más profundo que otro para evaluar el impacto de la profundidad en el accuracy.
El 'modelo simple' consta de 1 capa oculta de 128 neuronas, el 'modelo profundo' consta de 3 capas ocultas de 300, 200 y 100 neuronas. Ambos utilizan ReLU como activación, optimizador ADAM y Sparse Categorical Crossentropy como pérdida.

Hipótesis El clasificador más profundo tendrá una mejor performance.

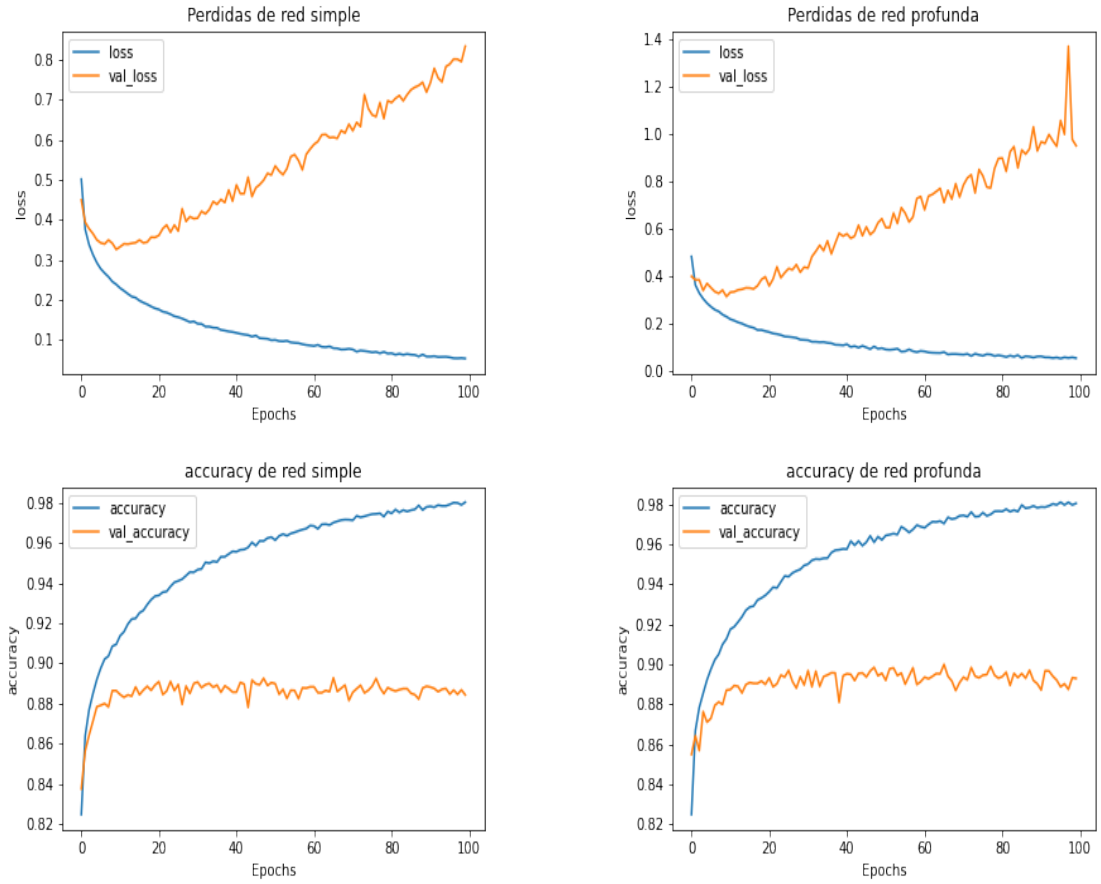


Figura 14: Evolución de pérdida y accuracy durante entrenamiento para ambas redes

Resultados

- Son los 2 modelos con más accuracy de todo el trabajo (ambos rondan el 90 %).
- Ambos se comienzan a 'overfitear' (separación de accuracy con accuracy con datos de validación) a partir de aproximadamente la época 10.
- La función de pérdida deja de decrecer aproximadamente al mismo tiempo que el overfitting. Esto marca que la función de pérdida está correctamente ligada a la tarea de clasificación.

4. Conclusiones

Se han revisado diversas técnicas que cumplieron mayormente con su objetivo.

- La base de datos de Fashion MNIST está uniformemente distribuída en cuanto a cantidades de representantes de cada clase.
- Salvo t-SNE, todos los modelos lograron obtener bajas significativas en el tiempo de clasificación manteniendo. mejorando o empeorando levemente el accuracy de kNN.
- Tanto t-SNE con UMAP logran agrupar las instancias de una manera que refleja diferencias similares a las clases verdaderas a pesar de ser métodos no supervisados.
- Si bien la mayoría de los resultados de UMAP fueron 'malos' ésto se debe a que no se llevó a cabo una búsqueda de parámetros óptimos, y vale rescatar que una de las pruebas logró un accuracy de 0.75 habiendo reducido la dimensión a 2, el espacio latente más bajo de todo el trabajo.
- Los autoencoders llegaron a obtener resultados más precisos que PCA para todas las dimensiones en las que clasificaron ambos. Además, logran codificar y reconstruir la imágen muy parecida a la original, lo cual muestra su alta capacidad para comprimir sus características subyacentes de las imágenes.
- Para PCA y KNN, el mejor par de parámetros parece ser $(\alpha, k) = (15, 15)$.

- Para autoencoders, el modelo que mejor accuracy otorga y que mejor reduce el tiempo de clasificación es el de 5 capas ocultas que reducen a dimensión 15, con la k de KNN igual a 15.
- La potencia de una red neuronal simple como clasificador es superior a la de métodos más clasicos (y de mayor complejidad conceptual) de aprendizaje automático, véase como con una red neuronal programada en 3 líneas de código se logró un accuracy sobre el conjunto de evaluación de aproximadamente 3 % más que el mejor modelo de todos los programados anteriormente con búsqueda intensiva de parámetros, a la primera prueba.

Referencias

[1] UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction, Appendix C: From t-SNE to UMAP.
`https://arxiv.org/pdf/1802.03426.pdf`

[2] t-SNE, Wikipedia
`https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding`

[3] Trabajo Práctico II, Métodos Numéricos DC UBA, 1o Cuatrimestre 2021
`https://github.com/pdbruno/metnum-tp2`

[4] sklearn.manifold.TSNE object documentation
`https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html`