Reconocimiento de Patrones

Integrantes:
Fernando Suzacq
Mercedes Marzoa
Tutor:
Guillermo Carbajal

December 3, 2017

1 Introducción

Uno de los conjuntos más clásicos del área de reconocimiento de patrones es la base de dígitos manuscritos MNIST [2]. Dicha base consiste en un conjunto de imágenes de 28x28 píxeles de dígitos escritos a mano y ha sido un estándar a la hora de comparar el desempeño de diferentes algoritmos. Sin embargo, esta base presenta una dificultad baja para técnicas relativamente sofisticadas y la comparación entre métodos termina siendo poco útil ya que quedan mal clasificados pocos patrones que son efectivamente ambiguos.

Se utilizara entonces la base "Fashion MNIST", creada por zalando research [1] la cual consiste en una nueva base de datos con algunas similitudes con la base MNIST. Esta base consiste en imágenes de 28x28 y la cantidad de clases también es igual a 10. La diferencia radica en el contenido de las imágenes, que corresponde a figuras de distintas prendas de vestir que pertenecen a 10 clases diferentes. Esta base presenta una dificultad mayor a la de la base MNIST, pero no excesivamente grande.

2 Base de Datos

La base de datos de entrenamiento consiste en 60000 imágenes de 28x28 píxeles, en escala de grises. Cada imagen corresponde a un tipo de prenda de vestir dentro del siguiente conjunto:

- 0 - Camiseta
- 1 - Pantalón
- 2 - Pulover
- 3 - Vestido
- 4 - Saco
- 5 - Sandalia
- 6 - Camisa
- 7 - Zapatilla
- 8 - Bolso
- 9 - Bota

La Figura 1 muestra un conjunto de imágenes ejemplo de la base.
La base de datos está en formato .csv, donde cada línea corresponde a un patrón. El primer valor es la clase (valor entre 0 y 9) y el resto de las columnas contiene valores entre 0 y 255 correspondientes a los valores del nivel de gris de la imagen correspondiente a la prenda. El valor de la imagen en las coordenadas (i,j) está guardado en la característica correspondiente a la posición x = 1 + i * 28 + j, donde i y j toman valores entre 0 y 27. La primera línea del archivo .csv es un encabezado indicando la clase y el número de características.

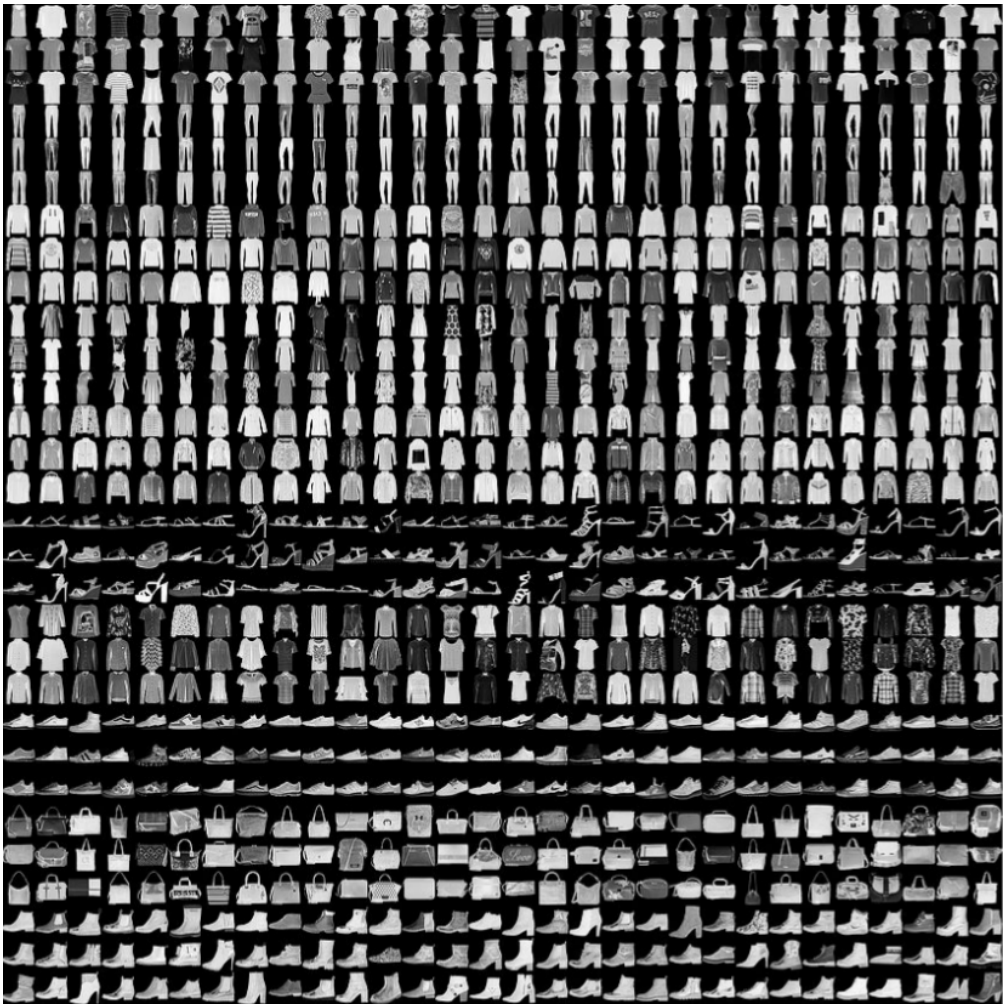


Figure 1: Conjunto de imágenes ejemplo de la base de datos Fashion MNIST

3 Marco Teórico

A continuación se describen las técnicas utilizadas en este proyecto.

3.1 Redes Neuronales

Las redes neuronales son un modelo computacional basado en un gran conjunto de unidades neuronales simples (neuronas artificiales), de forma aproximadamente análoga al comportamiento observado en los axones de las neuronas en los cerebros biológicos. Cada unidad neuronal está conectada con muchas otras y los enlaces entre ellas pueden incrementar o inhibir el estado de activación de las neuronas adyacentes. Puede existir una función limitadora o umbral en cada conexión y en la propia unidad, de tal modo que la señal debe sobrepasar un límite antes de propagarse a otra neurona. Estos sistemas aprenden y se forman a sí mismos, en lugar de ser programados de forma explícita.

La arquitectura de una red consiste en la disposición y conexiones de las neuronas. Podemos distinguir en una red, el número de capas, el tipo de las capas (ocultas o visibles), si son de entrada o de salida y la

direccionalidad de las conexiones de las neuronas.

Las redes multicapa están formadas por dos o mas capas de neuronas conectadas entre ellas.

El problema habitual de este tipo de redes es, dados un conjunto de datos ya clasificados de los que se conoce la salida deseada, se busca proporcionar los pesos adecuados de la red para que se obtenga una aproximación correcta de las salidas si la red recibe únicamente los datos de entrada.

A mediados de los años 80 se creo un algoritmo, llamado de propagación hacia atrás, que aproxima los pesos a partir de los datos objetivo. Este algoritmo de entrenamiento de la red se puede resumir muy brevemente en los siguiente puntos:

- Comenzar con algunos pesos, generalmente elegidos al azar.
- Introducir datos de entrada (en la capa de entrada) elegidos al azar entre el conjunto de datos de entrada que se va a usar para el entrenamiento.
- Dejar que la red genere un vector de datos de salida (propagación hacia delante).
- Comparar la salida generada por al red con la salida deseada.
- La diferencia obtenida entre la salida generada y la deseada (denominada error) se usa para ajustar los pesos sinápticos de las neuronas de la capa de salida.
- El error se propaga hacia atrás (back-propagation), hacia la capa de neuronas anterior, y se usa para ajustar los pesos sinápticos en esta capa.
- Se continua propagando el error hacia atrás y ajustando los pesos hasta que se alcance la capa de entradas.
- Este proceso se repetirá con los diferentes datos de entrenamiento.

A continuación se presenta brevemente dos de los tipos de rede más utilizados.

3.1.1 Fully Connected

Para las redes neuronales regulares, el tipo de capa más común es la capa completamente conectada en la que las neuronas entre dos capas adyacentes están conectadas por completo, pero las neuronas dentro de una sola capa no comparten conexiones.

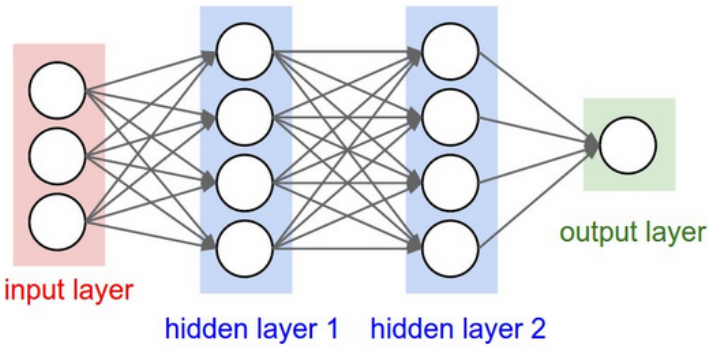


Figure 2: Red neuronal de 3 capas con tres entradas, dos capas ocultas de 4 neuronas cada una y una capa de salida.Imagen tomada de [4]

La capa de salida, a diferencia del resto de las capas la red neuronal,no suelen tener una función de activación. Esto se debe a que la última capa de salida generalmente se toma para representar los puntajes de las clases, que son números arbitrarios de valores reales, o algún tipo de objetivo de valor real.

3.1.2 Convolucionales

Las redes neuronales convolucionales suponen explícitamente que las entradas son imágenes, lo que permite codificar ciertas propiedades en la arquitectura. Esto hace que la función forward sea más eficiente de implementar y reduce enormemente la cantidad de parámetros en la red.
La capa Conv es el componente básico de una red convolucional, siendo la que realiza la mayor parte del procesamiento computacional.

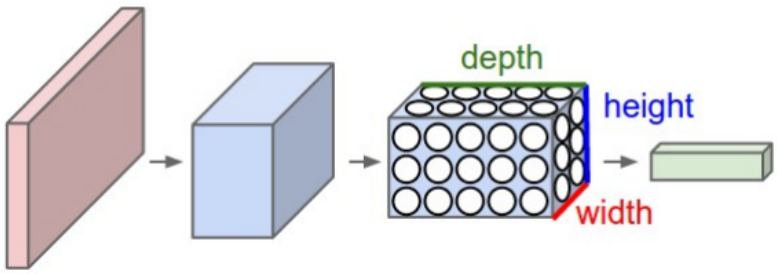


Figure 3: Red neuronal convolucional, organiza sus neuronas en tres dimensiones (ancho, alto, profundidad), como se visualiza en una de las capas. Cada capa transforma el volumen de entrada 3D a un volumen de salida 3D de activaciones neuronales.En este ejemplo, la capa de entrada roja contiene la imagen, por lo que su ancho y alto serían las dimensiones de la imagen, y la profundidad sería 3 (canales rojo, verde, azul).Imagen tomada de [4]

Los parámetros de la capa Conv consisten en un conjunto de filtros, que se pueden aprender. Cada filtro es especialmente chico (a lo ancho y alto), pero se extiende a través de la profundidad total del volumen de entrada. Por ejemplo, un filtro típico en una primera capa de una red convolucional podría tener un tamaño de 5x5x3. Durante el pase hacia adelante, se realiza la convolución de cada filtro a través del ancho y alto del volumen de entrada y se calculan los productos entre las entradas del filtro y la entrada en cualquier posición. A medida que se desliza el filtro sobre el ancho y la altura del volumen de entrada se produce un mapa de activación bidimensional que proporciona las respuestas de ese filtro en cada posición espacial. Intuitivamente, la red aprenderá los filtros que se activan cuando ve algún tipo de característica visual, como un borde de alguna orientación o una mancha de algún color en la primera capa, o eventualmente patrones completos en forma de panal o en forma de rueda en las capas superiores de la red . Entonces, se tiene un conjunto completo de filtros en cada capa CONV (por ejemplo, 12 filtros), y cada uno de ellos producirá un mapa de activación bidimensional por separado. Se apilan estos mapas de activación a lo largo de la profundidad y se produce el volumen de salida.

Se utilizan tres tipos principales de capas para construir arquitecturas de redes convolucionales: capa convolucional, capa de pooling y capa totalmente conectada (exactamente como se ve en las redes neuronales normales).

Capa de pooling
Es común insertar periódicamente una capa Pooling entre capas sucesivas de Conv. Su función es reducir progresivamente el tamaño espacial de la representación para reducir la cantidad de parámetros y el cálculo en la red y, por lo tanto, también controlar el sobreajuste. La capa de Pooling opera independientemente en cada sector de profundidad de la entrada y lo cambia de tamaño, usando la operación MAX.

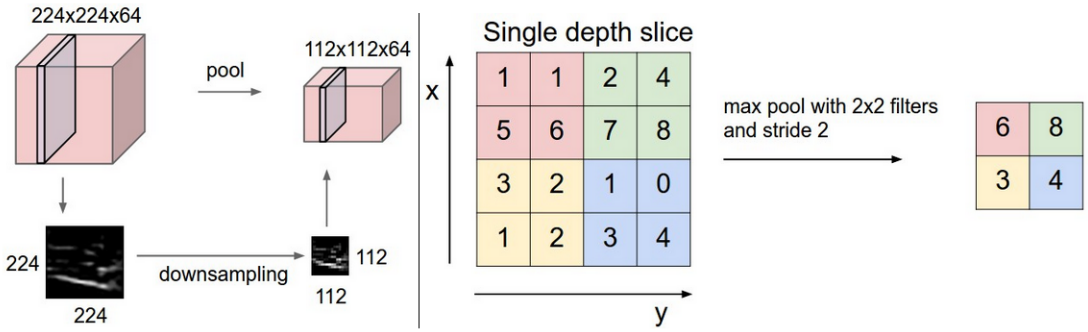


Figure 4: Capa de pooling.Imagen tomada de [4]

4 Experimentos y Resultados

Nuestro objetivo en este proyecto era investigar sobre el uso de las redes neuronales aplicadas al reconocimiento de patrones. El conjunto de datos de FashionMNIST es ideal para la aplicación de redes neuronales, y en particular de redes neuronales convolucionales, dado que es un conjunto de imágenes, y estas últimas tienen un gran desempeño en tareas de clasificación sobre cualquier tipo de imágenes. Como se puede observar en la página de github donde esta publicado FashionMNIST, las pruebas con mejores resultados de clasificación son aquellas hechas con redes, y en particular, las mejores son aquellas con las últimas arquitecturas de redes propuestas. Esto es un reflejo de lo que pasa en el campo de Visión Artificial, donde en los últimos años, las publicaciones han sido mayoritariamente sobre este tema. Lo que se buscó en este proyecto fue, en primera instancia, familiarizarnos con el diseño de las diferentes arquitecturas de redes, y con su entrenamiento. Como producto secundario de esta tarea se alcanzaron resultados de clasifiación muy buenos. Luego, en una segunda instancia, sobre algunas de las redes definidas y entrenadas, se aplicaron algunas técnicas de visualización, de manera de entender un poco más que es lo que, de manera general, aprenden estas redes. Quedó fuera del alcance de este proyecto las pruebas con arquitecturas más complejas, como por ejemplo: ResNet, DenseNet o WRN. La razón de esto es tanto una falta de tiempo, como no estar lo suficientemente familiarizados con estas arquitecturas y sus requerimientos básicos para su aplicación.

4.1 Linea Base

Para comenzar a familiarizarnos con la base de datos, y confeccionar así una línea base contra la cual comparar los resultados de las pruebas con redes neuronales, se realizaron algunas pruebas con métodos más clásicos. En particular, se utilizó Bayes, KNN y Árboles de decisión. Para realizar estas pruebas se utilizó la libreria scikit-learn. Se utilizaron los parámetros por defecto de esta librería, dado que no era la intención profundizar en estos métodos. Se dividió el conjunto de datos de entrenamiento en dos subconjuntos: el 80% utilizado para entrenamiento y el 20% restante utilizado para la validación de los algoritmos entrenados. El procedimiento de entrenamiento fue el siguiente: Una vez definido el método a entrenar, se le da como entrada el conjunto de entrenamiento y se ejecuta el algoritmo con ese conjunto, donde cada ejemplo de entrenamiento es un vector con el conjunto de píxeles de la imagen. Una vez que termina su ejecución, se evalúa el entrenamiento y la performance del algoritmo con el conjunto de validación.

Se obtuvieron los siguientes resultados:

	Bayes	Árboles de decisión	Knn
Patrones mal clasificados	5024/12000	2521/12000	1750/12000
Precisión	58.133%	78.992%	85.417%

En particular, una prueba adicional que se realizó, fue la aplicación de PCA al conjunto de entrenamiento previo a que este sea entrada del método de árboles de decisión. Nos pareció interesante realizar esta prueba dado que la entrada del conjunto de píxeles es una entrada de dimensión: 28x28=784. Por esto, una posible hipótesis es que muchos de los píxeles de la imagen no contribuyan de ninguna manera al entrenamiento de estos métodos, y por lo tanto, a la clasificación de los ejemplos. Por lo tanto, lo único que se estaría haciendo es agregar ruido como entrada, lo que puede afectar el desempeño del clasificador. Sin embargo, los resultados obtenidos no fueron mucho mejores:

	Árboles de decisión	Árboles de decisión con PCA
Patrones mal clasificados	2521/12000	3026/12000
Precisión	78.992%	74.783%

Teniendo entonces nuestra línea base, procedimos a enfocarnos en los métodos con redes neuronales.

4.2 Fully Connected

Inicialmente se decidió realizar pruebas con redes Fully Connected. Se tomaron arquitecturas de distintos lados [3], se entrenaron para reproducir sus resultados, y luego se realizaron modificaciones sobre ellas para visualizar el efecto de los cambios sobre tanto el entrenamiento de las mismas como el impacto sobre el resultado. Se utilizó Keras [6] como librería para la construcción de las redes, dada su facilidad de uso y su posibilidad de utilizar Tensorflow como backend. Esto nos permitió incluso, hacer uso de la GPU de las máquinas para acelerar sustancialmente el entrenamiento.

Tomando el conjunto de datos de entrenamiento como entrada, se realizaron pruebas con redes FC de 2, 3 y 5 capas. Cuanta mayor profundidad, se tiene mayor cantidad de parámetros y mayor capacidad de aprendizaje de la red, pero por contrapartida, las redes son más difíciles de entrenar, tomando más tiempo para llegar a su convergencia y también siendo más difícil encontrar hiperparametros adecuados.

En todos los casos, se realizó una extensión a la interfaz brindada por Keras al momento de entrenar, haciendo uso de sus callbacks, para detener el entrenamiento si no hay una mejora en la precisión sobre el set de validación, en un período de 5 epochs. Esto era necesario para permitirnos realizar mayor cantidad de pruebas y no malgastar el tiempo en entrenar redes que ya convergieron, estaban sobre ajustando, o simplemente no estaban dando buenos resultados.

Se realizo una primera prueba con las redes de 2 capas, con la siguiente arquitectura:[Dense + Relu + Batchnorm] x 2 + Softmax

Luego, se agregó una capa de profundidad, realizando pruebas con 3 capas: [Dense + Relu + Batchnorm] x 3 + Softmax

En la primera de ellas, las capas tenían 128 nodos. Se utilizó ADAM como optimizador, con un learning rate de 0,0005. Se utilizaron capas de BatchNormalization para facilitar el entrenamiento. Se puede observar que esta red sobreajusta muchísimo, obteniéndose el mejor resultado al final del primer epoch. Se entrenó hasta la epoch 20 porque a este momento no se contaba con la extensión al entrenamiento para detenerlo si no esta mejorando.

Luego, se decidió modificar el learning rate. Se aumentó a 0,003, con la hipótesis de que al tener un learning rate más alto, se podría evitar o salir de algún mínimo local en el cual la anterior configuración podría establecerse. Sorprendentemente esto contribuyó a mejorar el sobreajuste, viendo las gráficas de entrenamiento y validación mucho más juntas. Sin embargo, se obtuvieron peores resultados.

Finalmente, se decidió modificar realizar una prueba modificando el optimizador (y bajando apenas el learning rate a 0.002). Se utilizó NADAM, que es una variante de ADAM pero con Nesterov momentum. Adicionalmente, duplicamos el batch size. Nuevamente, el pensamiento era el mismo que en la hipótesis anterior, pensando que teniendo momentum y un batch más grande se podría converger más rapido y de manera más efectiva. Sin embargo, los resultados fueron considerablemente menores.

Para la siguiente prueba decidimos aumentar considerablemente la capacidad de la red, y construir una red de 5 capas. Decidimos modificar principalmente agregar más capas en lugar de aumentar la cantidad de neuronas por capas, porque si bien ámbas modificaciones logran aumentar la capacidad de la red, introducir más capas aumenta el poder de discriminación de la misma. Si bien esto tiene el efecto negativo de hacer que esta sea más difícil de entrenar, consideramos que una red con 5 capas no es lo suficientemente profunda para sufrir demasiado los problemas de las redes realmente profundas. Los resultados con esta arquitectura fueron los mejores, pero no por un margen considerable. Cabe notar que si fue considerablemente más largo el tiempo de entrenamiento con esta red que con las anteriores.

4.3 Fully Connected + HOG

Luego se decidió utilizar como entrada HOG además del valor de los pixeles. Para ello se utilizo la siguiente arquitectura: [Dense + Relu + Batchnorm] x 3 + Softmax, Adam como optimizador y un learning rate de 0.0001.

4.4 Redes Convolucionales

Luego de experimentar con las redes Fully Connected, se decidió probar las redes convolucionales, debido a las grandes ventajas que presentan las mismas para trabajar con imágenes Se tomaron arquitecturas de distintos lados [7] [5], se entrenaron para reproducir sus resultados, y luego se realizaron modificaciones sobre ellas para visualizar el efecto de los cambios sobre el entrenamiento de las mismas y el impacto sobre el resultado. Se utilizó Keras al igual que para las Fully Connected.
Para la primera prueba se utilizo la siguiente arquitectura que se muestra en la figura 9.

Primero se probó con un filtro de 3x3 y luego con un filtro de 5x5, lo que no genero practicante ningún cambio. Luego se probo cambiar el learning rate de 0.001 a 0.01. Se puede ver en los gráficos de la imagen 9 como mejorar los resultados, en particular

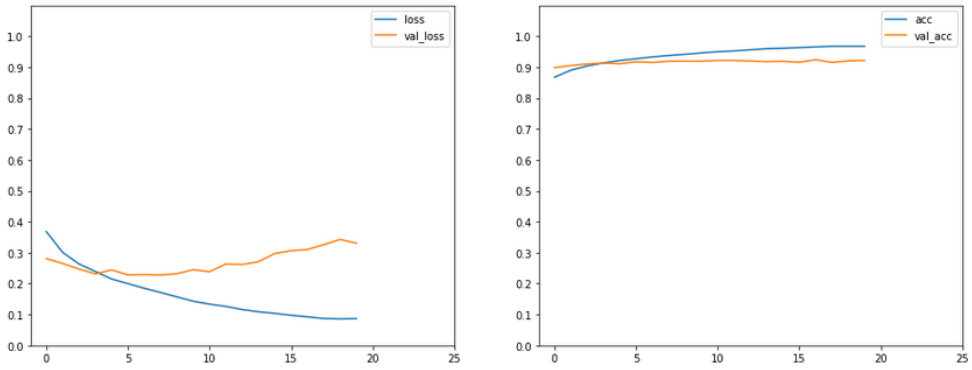


Figure 6: Resultado utilizando un learning rate = 0.001

Layer (type)	Output Shape	Param #
conv2d_13 (Conv2D)	(None, 24, 24, 32)	832
conv2d_14 (Conv2D)	(None, 20, 20, 32)	25632
max_pooling2d_7 (MaxPooling2	(None, 10, 10, 32)	0
conv2d_15 (Conv2D)	(None, 6, 6, 64)	51264
conv2d_16 (Conv2D)	(None, 2, 2, 64)	102464
max_pooling2d_8 (MaxPooling2	(None, 1, 1, 64)	0
flatten_4 (Flatten)	(None, 64)	0
dense_7 (Dense)	(None, 256)	16640
leaky_re_lu_7 (LeakyReLU)	(None, 256)	0
dropout_7 (Dropout)	(None, 256)	0
dense_8 (Dense)	(None, 256)	65792
leaky_re_lu_8 (LeakyReLU)	(None, 256)	0
dropout_8 (Dropout)	(None, 256)	0
preds (Dense)	(None, 10)	2570

Total params: 265,194
Trainable params: 265,194
Non-trainable params: 0

Figure 5: Arquitectura utilizada para la red convolucional.

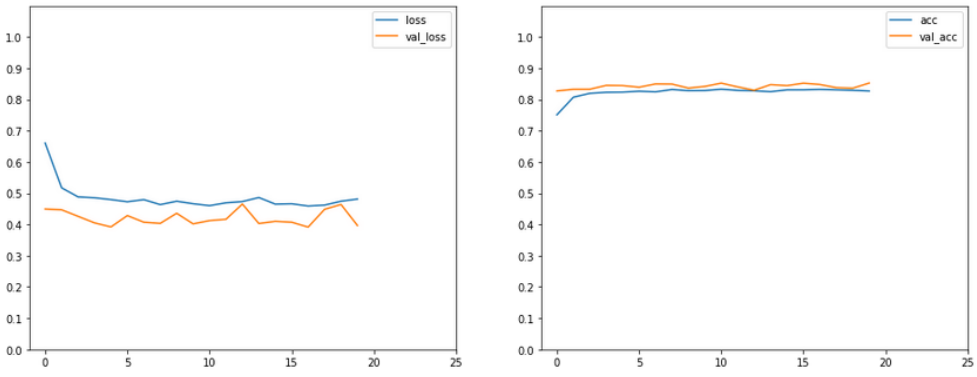


Figure 7: Resultado utilizando un learning rate = 0.01

Luego se probó con la siguiente arquitectura:

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	(None, 28, 28, 1)	0
batch_normalization_4 (Batch	(None, 28, 28, 1)	4
conv2d_5 (Conv2D)	(None, 24, 24, 64)	1664
max_pooling2d_3 (MaxPooling2	(None, 12, 12, 64)	0
conv2d_6 (Conv2D)	(None, 8, 8, 512)	819712
max_pooling2d_4 (MaxPooling2	(None, 4, 4, 512)	0
flatten_2 (Flatten)	(None, 8192)	0
dense_4 (Dense)	(None, 128)	1048704
dropout_3 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 64)	8256
dropout_4 (Dropout)	(None, 64)	0
dense_6 (Dense)	(None, 10)	650

Total params: 1,878,990
Trainable params: 1,878,988
Non-trainable params: 2

Figure 8: Arquitectura utilizada para la red convolucional.

Variando de la misma forma que en los casos anteriores se llego a que utilizar un leraning rate de 0.001 generaba los mejores resultados.

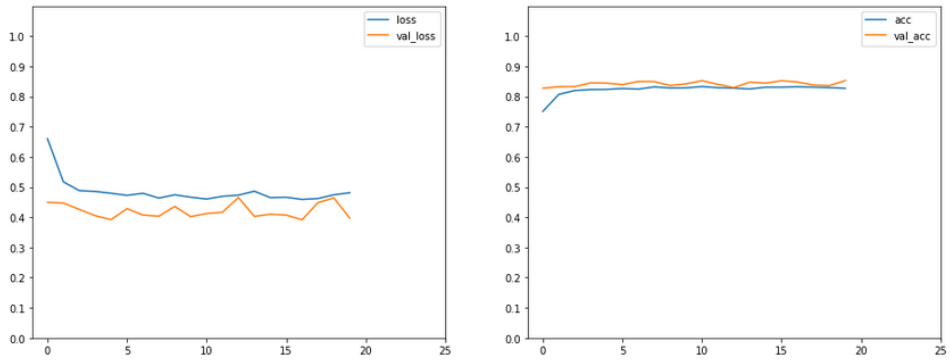


Figure 9: Arquitectura utilizada para la red convolucional con learning rate = 0.01

Por ultimo, se probo con otra arquitectura, obteniendo los mejores resultados:

Layer (type)	Output Shape	Param #
conv2d_15 (Conv2D)	(None, 28, 28, 32)	320
conv2d_16 (Conv2D)	(None, 28, 28, 64)	18496
max_pooling2d_9 (MaxPooling2	(None, 14, 14, 64)	0
dropout_11 (Dropout)	(None, 14, 14, 64)	0
conv2d_17 (Conv2D)	(None, 14, 14, 128)	73856
conv2d_18 (Conv2D)	(None, 12, 12, 256)	295168
max_pooling2d_10 (MaxPooling	(None, 4, 4, 256)	0
dropout_12 (Dropout)	(None, 4, 4, 256)	0
flatten_5 (Flatten)	(None, 4096)	0
dense_12 (Dense)	(None, 256)	1048832
leaky_re_lu_5 (LeakyReLU)	(None, 256)	0
dropout_13 (Dropout)	(None, 256)	0
dense_13 (Dense)	(None, 256)	65792
leaky_re_lu_6 (LeakyReLU)	(None, 256)	0
dense_14 (Dense)	(None, 10)	2570

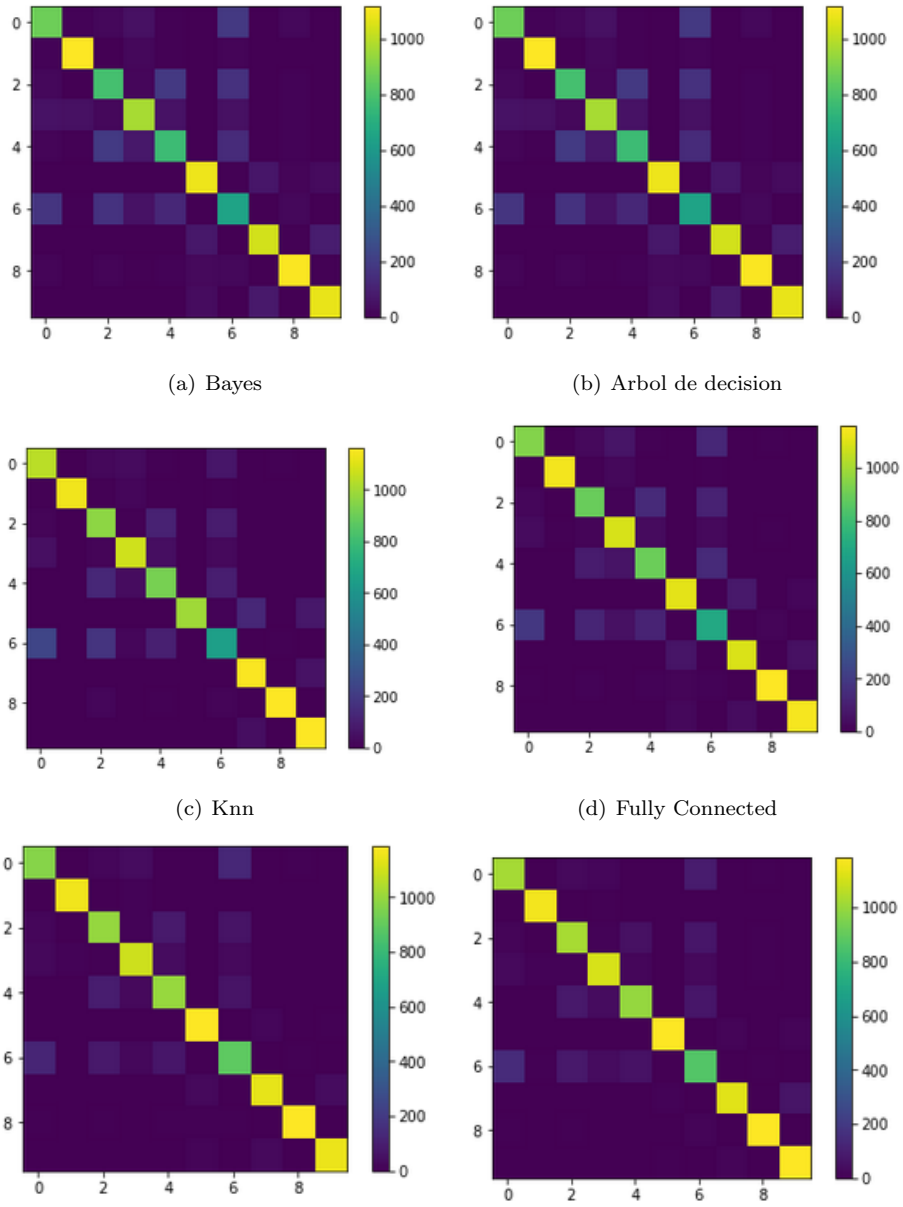
Total params: 1,505,034
Trainable params: 1,505,034
Non-trainable params: 0

Figure 10: Resultado utilizando un learning rate = 0.01

.

4.5 Resultados

Se decidió realizar las matrices de confusión para las mejores arquitecturas, a ver si notábamos que clases esta confundiendo y si las mismas cambian según el tipo de red utilizada, pero no se notaron grandes cambios entre las arquitecturas.



(a) Bayes

(b) Arbol de decision

(c) Knn

(d) Fully Connected

(e) Fully Connected - HOG

(f) Convolucional

Figure 11: Matriz de confusión para los diferentes métodos

Como se puede ver, la clase que más confunde es la 6 y la 4, que son la camisa y el saco respectivamente.

En la siguiente tabla se pueden ver los resultados obtenidos para cada método.

Red	Precisión
Fully connected - 3 capas	0.863833
Fully connected - 5 capas	0.869333
Fully connected - HOG	0.849667
Fully connected - HOG + pixels	0.9
Convolucional	0.923000

Resultados obtenidos para las diferentes arquitecturas de las redes

En particular, para los datos de test con la mejor arquitectura obtenida se llegó a 94,1% de accuracy.

4.6 Visualización

Existen varios enfoques para comprender y visualizar las redes convolucionales, en parte como respuesta a la crítica común de que las características aprendidas en una Red Neural no son interpretables. En esta sección, examinamos brevemente algunos de estos enfoques.

4.6.1 Capa de activación

La técnica de visualización más directa es mostrar las activaciones de la red durante el pase hacia adelante. Una peligrosa trampa que se puede notar fácilmente con esta visualización es que algunos mapas de activación pueden ser todos cero para muchas entradas diferentes, lo que puede indicar filtros muertos, y puede ser un síntoma de altas tasas de aprendizaje. A continuación se puede observar el mapa de activación de la clase pantalón, en el cual se ve como el agujero entre las piernas corresponde al área a la cual más atención se le presta.

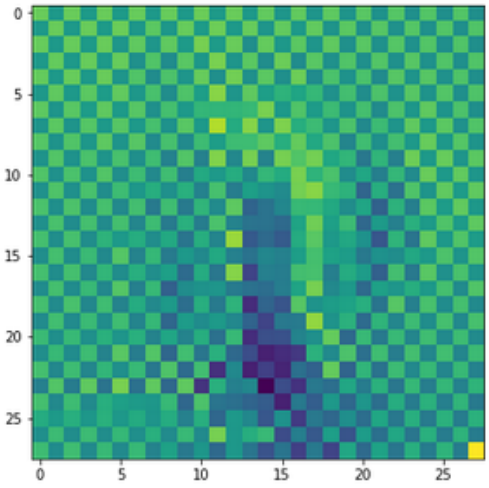
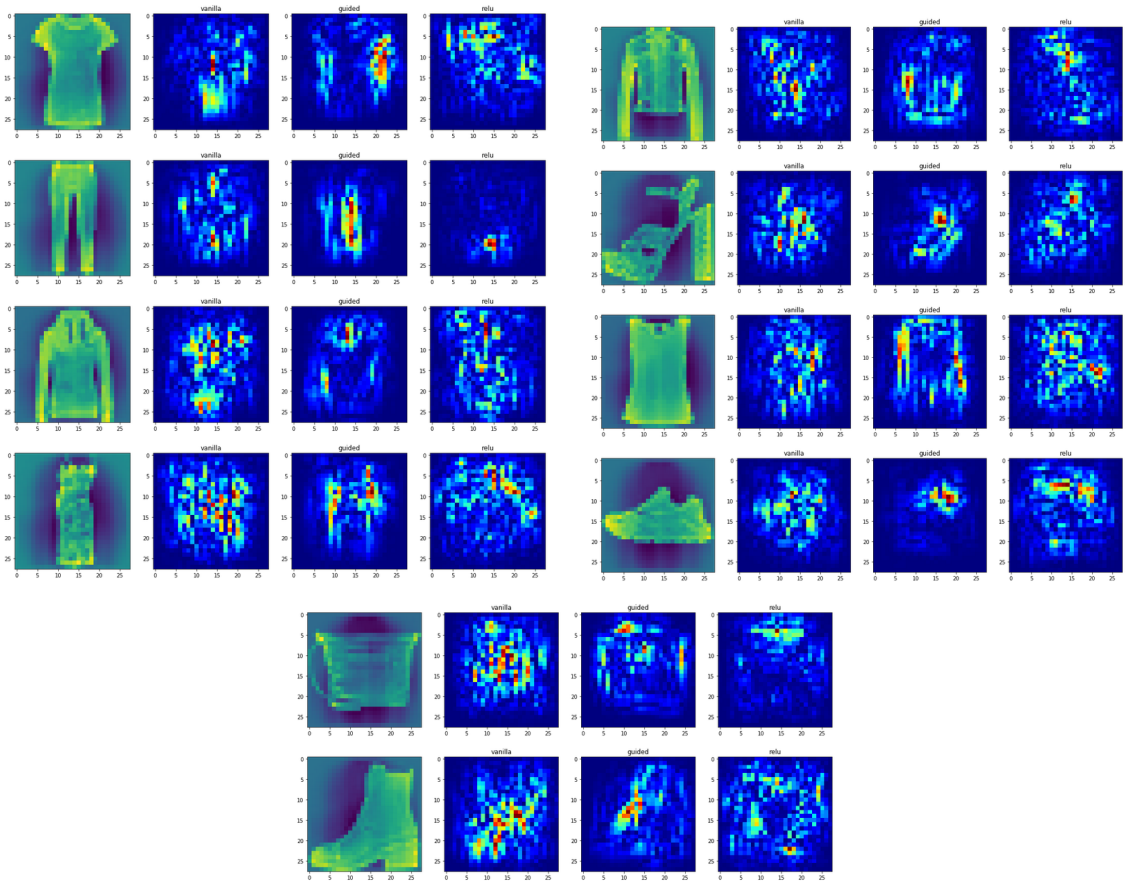


Figure 12: Mapa de activación del pantalón

.

4.6.2 Saliency maps

Busca determinar que pixeles se utilizan en el entrenamiento, para ello se calcula el gradiente de la categoría de salida con respecto a la imagen de entrada. Esto debería indicarnos cómo cambia el valor de la categoría de salida con respecto a un pequeño cambio en los píxeles de la imagen de entrada. Todos los valores positivos en los gradientes nos dicen que un pequeño cambio en ese píxel aumentará el valor de salida.



4.6.3 Filtros de convolución

Una estrategia común es visualizar los pesos. Estos suelen ser más interpretables en la primera capa CONV que mira directamente a los datos en bruto de píxeles, pero también es posible mostrar los pesos del filtro más profundamente en la red. Los pesos son útiles para visualizar porque las redes bien entrenadas generalmente muestran filtros agradables y suaves sin ningún patrón ruidoso. Los patrones ruidosos pueden ser un indicador de una red que no se ha entrenado durante el tiempo suficiente, o posiblemente una regularización muy baja que puede haber llevado a un ajuste excesivo.En la figura 13 se pueden observar los filtros de la primera capa de convolución.

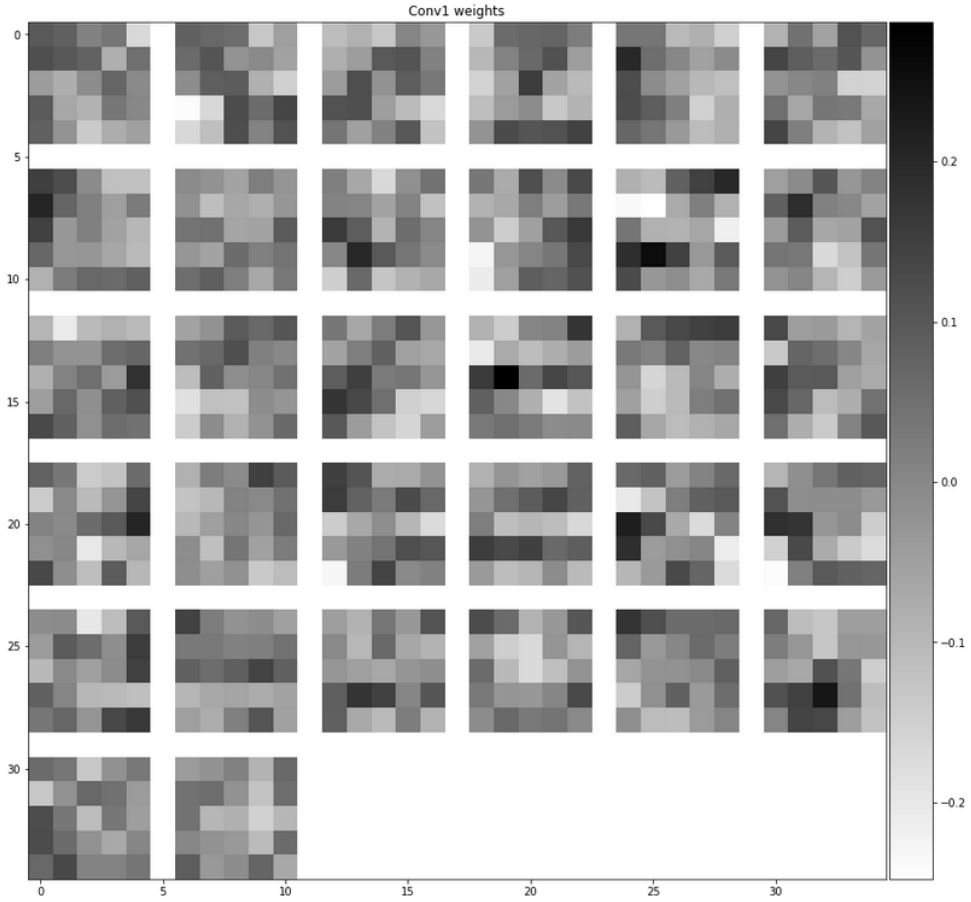


Figure 13: Filtros de la primera capa de convolución

.

5 Conclusiones

Se realizó la clasificación de la base de datos Fashion MNIST, la cual consiste en imágenes de ropa. Se logró un accuracy de 94% para la clasificación de la base de test.
En primer lugar se implementaron diferentes métodos clásicos, bayes, árbol de decisión y knn para obtener una noción de los valores a obtener. Como la idea no fue trabajar en profundidad con estos métodos, no se realizó una búsqueda exhaustiva de paramentos.
Dado que la base de datos es de imágenes y las redes neuronales presentan buenos resultados cuando se trabaja con imágenes, se decidió realizar la clasificación utilizando redes neuronales. Se exploraron diferentes tipos y arquitecturas, en particular, se utilizaron redes fully connected y redes convolucionales.
Se probó utilizar las imágenes en crudo como entrada así como HOG, obteniendo mejores resultados para las fully connected utilizando HOG. Se estudió en detalle como ajustar los parámetros de las arquitecturas para obtener mejores resultados.
Por último se estudiaron técnicas de visualización para entender como funcionan las redes neuronales, observando los filtros de convolución, las capas de activación, etc.

[TRABAJO A FUTURO]

References

[1] Base de datos fashion-mnist. https://github.com/zalandoresearch/fashion-mnist. Accedida: 27-11-2017.

[2] Base de datos mnist. http://yann.lecun.com/exdb/mnist. Accedida: 27-11-2017.

[3] Classifying clothes using tensorflow. https://medium.com/tensorist/classifying-fashion-articles-using-tensorflow-fashion-mnist-f22e8a04728a. Accedida: 27-11-2017.

[4] Cs231n convolutional neural networks for visual recognition. http://cs231n.github.io/. Accedida: 27-11-2017.

[5] keras-cnn-fashion-mnisty. https://www.kaggle.com/chria5k/keras-cnn-fashion-mnist. Accedida: 27-11-2017.

[6] Keras: The python deep learning library. https://keras.io/. Accedida: 27-11-2017.

[7] A vgg-like cnn in keras for fashion-mnist with 94 http://www.statsblogs.com/2017/09/07/a-vgg-like-cnn-in-keras-for-fashion-mnist-with-94-accuracy/. Accedida: 27-11-2017.