UNIVERSIDAD
DE GRANADA

Práctica alternativa:
«Estudio del problema *Fashion MNIST*»

INTELIGENCIA DE NEGOCIO
Grado en Ingeniería Informática
CURSO 2020-21

ALONSO BUENO HERRERO
76067525-Q
alonsobueno13@correo.ugr.es

Escuela Técnica Superior de
Ingenierías Informática y de Telecomunicación

Índice

Índice de cuadros

Índice de figuras

1 Descripción y análisis del problema

El problema que abordamos a continuación es de enorme interés en dos campos: el aprendizaje automático en general (ML) y la visión por computador (CV). Fashion MNIST es una base de datos que se deriva de una previa, MNIST. Esta última es también un conocidísimo banco de pruebas en la materia para programadores, investigadores, etc. MNIST se centra en el reconocimiento de dígitos numéricos de una cifra (de 0 a 9) escritos a mano. La base de datos s de 60.000 objetos (imágenes) de entrenamiento y 10.000 de test.

Las principales ventajas que rodeaban a MNIST eran: la facilidad de uso, la gran cantidad de imágenes disponibles en la base de datos, que viene incluido en los frameworks más populares del área, etc.

Pero diversos motivos dieron lugar a su pérdida de relevancia, tal y como se indica en [2]:

- Era muy sencillo: Las redes neuronales convolucionales (algoritmo de Machine Learning que veremos más adelante) alcanzan un 99.7 % de acierto fácilmente, mientras que algoritmos más tradicionales de ML ya rondan un 97 % de efectividad.
- Se ha usado en exceso: Ian Goodfellow, una eminencia en el mundo de deep learning, hizo un llamado a la comunidad en 2017 para que se dejara de usar.
- No representa tareas actuales de visión por computador: Otra de las figuras de deep learning, François Chollet, manifestó que MNIST ya no es representativo de las tareas, actividades y desafíos de la visión por computador moderna.

Así las cosas, fue propuesto un nuevo conjunto de datos, protagonista de nuestro estudio, Fashion MNIST. Las principales características de este conjunto, algunas heredadas de su padre, son:

- Al igual que MNIST, Fashion-MNIST se compone de imágenes en escala de grises, de 28×28 píxeles cada una.
- La base de datos se compone de 60.000 imágenes para entrenamiento, y unas 10.000 para pruebas. Todas las clases/categorías (10) poseen 6.000 imágenes, lo que implica que el dataset está balanceado.
- A diferencia del MNIST tradicional, las categorías no son números del 0 al 9 escritos a mano alzada, sino prendas de vestir, como se puede suponer por su nombre. En particular, estas son las categorías:
  - T-shirt/top (Camiseta o top).
  - Trouser/pants (Pantalones).
  - Pullover shirt (Pullover).
  - Dress (Vestido).
  - Coat (Abrigo).
  - Sandal (Sandalias).
  - Shirt (Camisa).
  - Sneaker (Zapatos deportivos).
  - Bag (Bolso o maleta)
  - Ankle boot (Botines).

En la figura 1 se muestran tres filas de versiones diferentes de cada prenda de la colección:

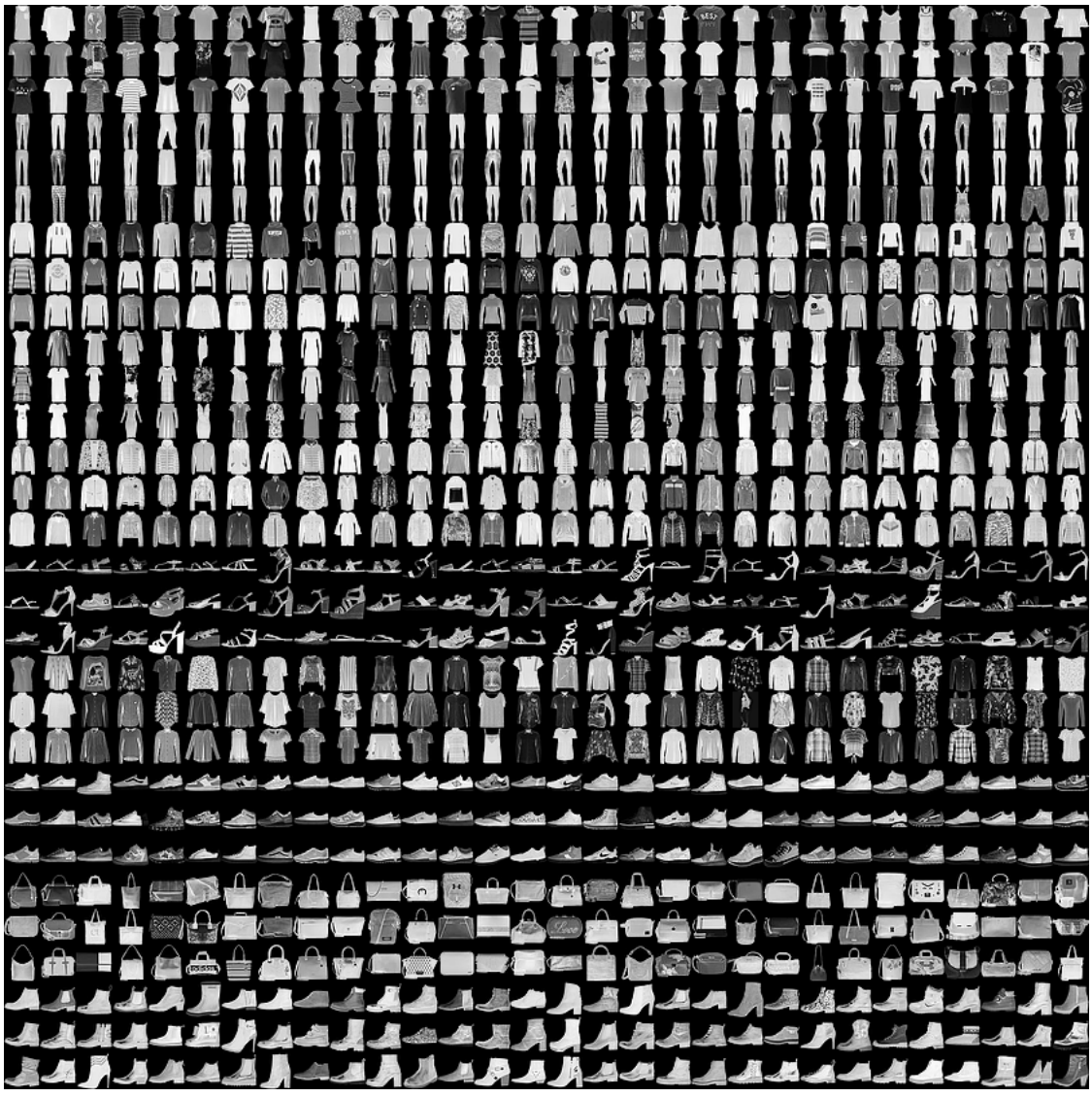


Figura 1: Muestra de prendas de la base de datos que nos ocupa.

1.1 Estado del arte sobre CNN

En el artículo científico [8] proponen un estado del arte para el problema Fashion MNIST basado en CNN. De las tres propuestas, que van en orden de complejidad, la **primera** ya la hemos implementado nosotros, y se corresponde, a falta de ver algunos valores concretos de parámetros del modelo (tamaño de kernels, de pooling, etc) con nuestro Experimento 2, el primero de la serie CNN.

El **segundo modelo** que proponen resulta muy interesante. Es un ajuste del modelo previo, consistente en aplicar una normalización a las imágenes que van entrado a cada capa convolucional, antes de que se aplique la correspondiente convolución (algo conocido como ***Batch normalization***).

Durante el proceso de entrenamiento, se normalizan las entradas de cada capa de convolución utilizando la media y la varianza de los valores en el fragmento (mini-batch) que se estaba procesando. La entrada de ese mini-batch generalmente se modifica para tener una media cero y una varianza unitaria.

Esta técnica permite entrenar la red más rápido con tasas de aprendizaje más altas. Con Batch Normalization, los autores indican que llegaron a lograr el mismo valor de pérdida (*loss function*) después de 10 repeticiones (*épocas*) que se logró después de 40 repeticiones sin aplicar *Batch Normalization*.

Por otra parte, en la referencia [10] aportan una motivación para utilizar esta técnica cuando se construyen modelos CNN mucho más complejos (con más capas).

La **tercera** y última **estrategia** se centra en aprovechar algunos "*skip connections*" o saltos de conexiones. Tal y como se indica en el artículo, cuando se entrena una red neuronal mediante *backpropagation*, el gradiente, que es el que permite optimizar y ajustar los pesos, debe propagarse hacia atrás también desde la capa de salida más alta hasta la capa de entrada más baja para garantizar que los parámetros se actualicen (se ajusten) correctamente (recordar en la figura 2). Esto se puede conseguir de forma sencilla usando conexiones de salto residuales (*residual skip connections*). En concreto, aquí se añaden la entrada anterior y el valor actual de la salida convolucionada para obtener el valor de salida final[1].

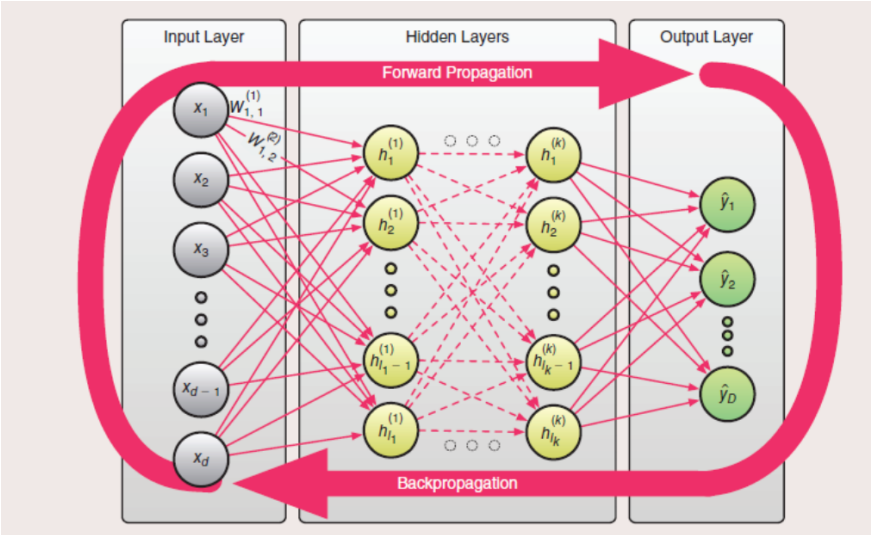


Figura 2: Concepto de *backpropagation*. Fuente: transparencias de clase.

A modo aclaratorio, en la figura 3, se muestran las tres arquitecturas propuestas por los autores en [8].

[1] En la referencia [11] se da una explicación más teórica y ampliada, por si es de interés.

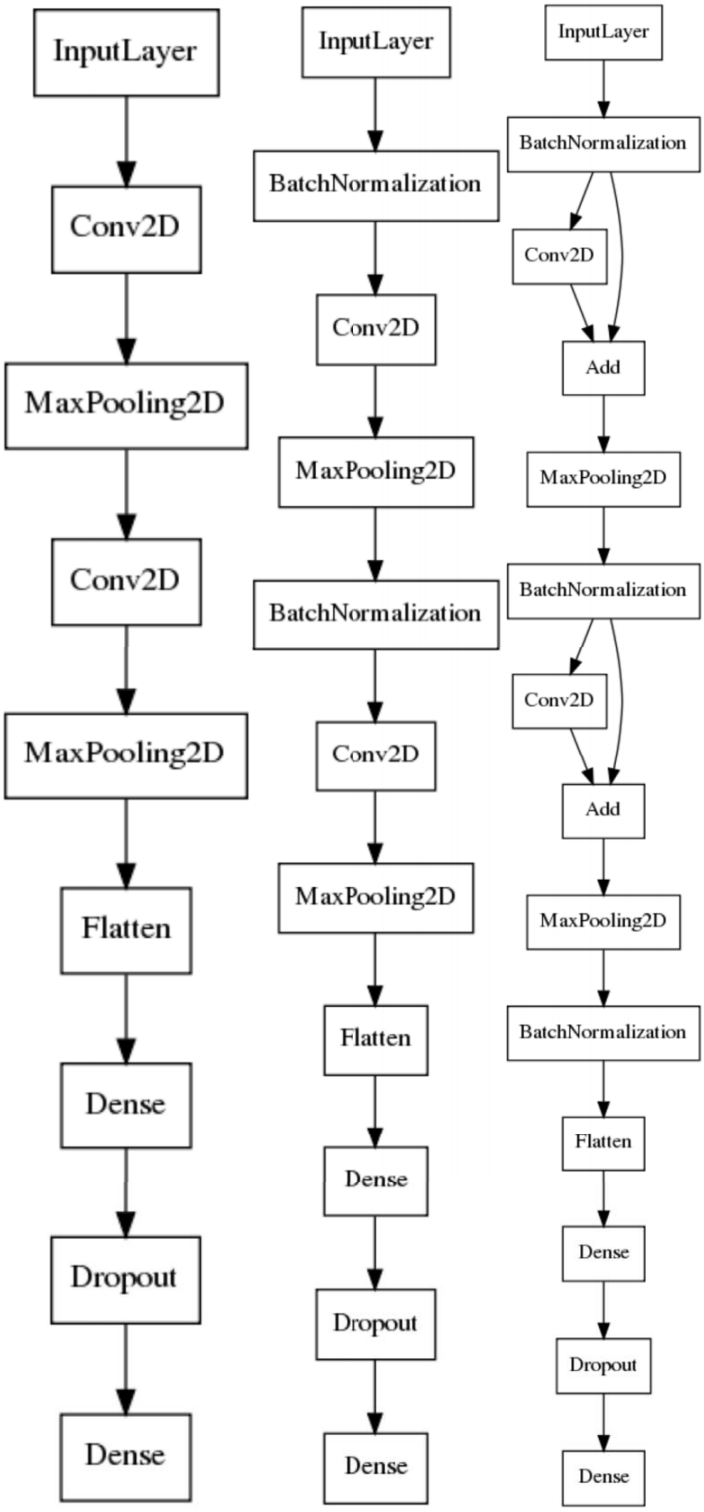


Figura 3: Arquitecturas propuestas en [8] para el *state-of-the-art* de Fashion MNIST en base a modelos CNN. Fuente: [8].

2 Descripción de los algoritmos

Del *benchmark* sobre modelos de *machine learning* que ofrecían en la página oficial, he intentado extraer, para que sea simplemente representativo, un modelo que fuese competitivo tanto en tiempo de ejecución como, al mismo tiempo, en bondad (tasa de acierto, en nuestro caso). De esta forma, he escogido una configuración del célebre Random Forest (o árboles aleatorios), cuyas ideas clave describo más adelante. Posteriormente, me centraré en la segunda propuesta del enunciado del problema: algoritmos de DL, en concreto, las redes convolucionales, en cuya explicación me detendré algo más debido a que no lo hemos trabajado en prácticas (más allá de un seminario "de ampliación", que refiero aquí porque, personalmente, me fue de gran ayuda para entender –creo que correctamente- esta estrategia y lo que la rodea.

Detallo ahora los algoritmos que he usado, y que darán lugar a los diversos experimentos que elaboraré posteriormente:

1. Clasificador Random Forest.
2. Redes neuronales convolucionales (CNN).
3. Red preentrenada a partir de la red VGG16.

2.1 Algoritmo *Random Forest*

Los modelos Random Forest están formados por un conjunto de árboles de decisión independientes, cada uno entrenado con una muestra distinta de los datos de entrenamiento. La muestra es generada mediante bootstrapping. La predicción de una nueva observación se produce por agregación (voto de la mayoría, generalmente) de las predicciones de todos los árboles individuales que conforman el modelo. Se trata de una estrategia de bagging, es decir, de combinación de los resultados obtenidos por clasificadores más simples (árboles de decisión) para obtener una única predicción común aportada por todos los clasificadores, y no por uno sólo.

Es una estrategia muy recomendada en la bibliografía para clasificación multiclase. Algunos parámetros que merece la pena comentar, y que han sido usados en nuestro experimento son:

- *Criterion*: define el criterio para medir la calidad de una partición del dataset asignada a cada clasificador.
- Profundidad máxima (*max_depth*): la profundidad de cada árbol clasificador. En nuestros experimentos lo hemos limitado a 100, para evitar posibles efectos de sobreaprendizaje.
- *Número de clasificadores*: este parámetro es en general menos susceptible de sobreaprendizaje, pero si el número de clasificadores es excesivo, nuestro clasificador puede ser demasiado lento, debido a que se vuelve más pesado.
- *Otros* parámetros, algo más específicos y "delicados", controlan, por ejemplo, el mínimo número de muestras para partir un nodo en un árbol o tareas de paralelización (n_jobs).

Para implementar el modelo de Random Forest he usado la librería Scikit-Learn, que conocemos de prácticas, y que ofrece, desde mi punto de vista, muchas posibilidades de ajuste del modelo debido a la gran cantidad de parámetros que ofrece.

La figura 4 ilustra de forma muy sencilla la "arquitectura" de un modelo de predicción Random Forest.

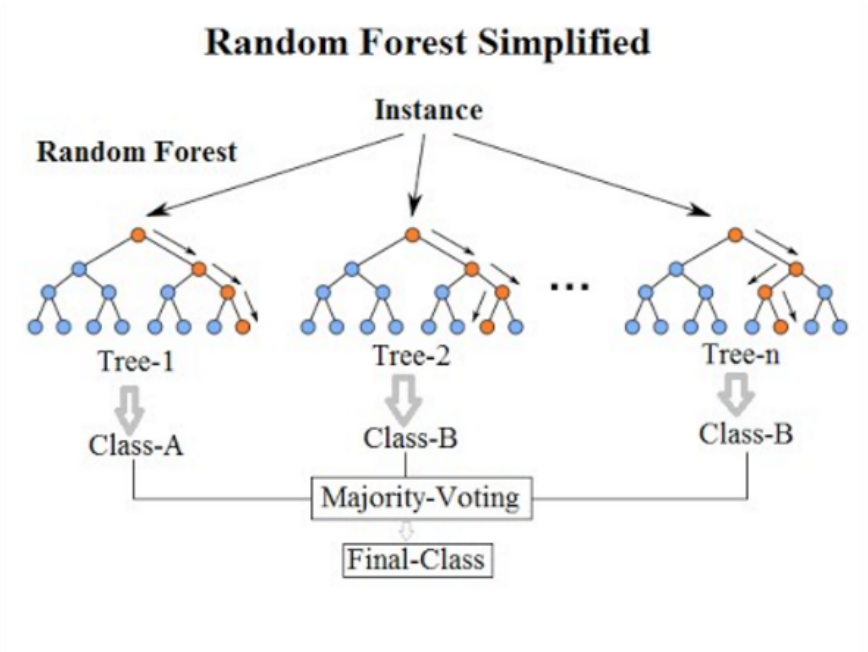


Figura 4: Estructura del algoritmo Random Forest. Fuente: Wikipedia.

2.2 Algoritmo CNN (redes neuronales convolucionales)

El algoritmo CNN es un tipo de Red Neuronal Artificial con aprendizaje supervisado que procesa sus capas imitando al cortex visual del ojo humano para identificar distintas características en las entradas. Dichas entradas (imágenes en escala de grises de ropa, en nuestro caso) permiten que la máquina pueda identificar objetos y, en definitiva, poder "ver". Para ello, la CNN contiene varias capas especializadas con una cierta jerarquía, basada en su capacidad de detección de elementos en la imagen: las primeras capas pueden detectar, por ejemplo, líneas o curvas, y a partir de aquí se van especializando hasta llegar a capas más profundas que reconocen formas complejas como un rostro o la silueta de un animal.

En definitiva, los elementos típicos de estas arquitecturas son:

1. **Entrada**: Serán los pixeles de la imagen. Representa al alto, ancho y profundidad. En nuestro caso, las imágenes de la BD son en escala de grises, luego se representará un solo color (una sola dimensión).

2. **Capa de Convolución**: procesará la salida de neuronas que están conectadas en ≪zonas locales≫ de entrada (es decir, píxeles cercanos), calculando el producto escalar entre sus pesos (valor de pixel) y una pequeña zona a la que están conectados en el volumen de entrada. Si usamos, por ejemplo, 32 filtros, este será nuestro volumen de salida -32 *feature maps*.

3. La función de activación se aplicará sobre la matriz resultante de la convolución. La función RELU, la que se utiliza en los dos experimentos de CNN, tiene como objetivo evitar que el modelo aprenda imágenes con características lineales, algo que no es característico de las imágenes, en general.

4. ***Pooling***: Hará una reducción de la dimensión de los mapas de características obtenidos anteriormente tomando fragmentos. La estrategia más común es la de selección del máximo de la región

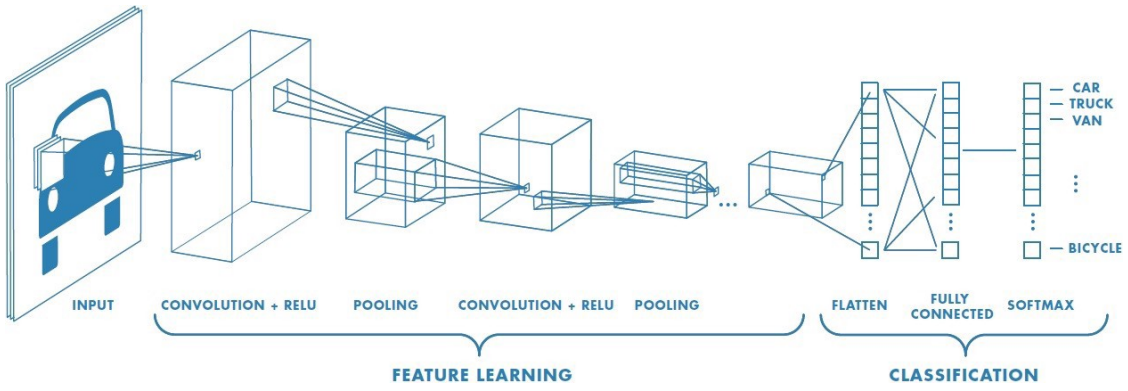


Figura 5: Arquitectura típica de un modelo CNN. Fuente: [4]

5. **Capa densa (red totalmente conectada - *FC*)**: red de neuronas feedforward que conectará con la última capa de *subsampling* y finalizará con la cantidad de neuronas que queremos clasificar. Es «tradición» añadirla (casi) antes de la clasificación final.

La figura 5 ilustra la estructura típica de un modelo de *deep learning* basado en CNN.

2.3 Red preentrenada a partir de VGG16

La técnica de redes neuronales preentrenadas consiste en la utilización de modelos neuronales que ya han sido entrenados con conjuntos de datos más genéricos que el problema en cuestión que se quiere resolver. Además, son de una enorme complejidad en comparación con los que aquí podamos proponer. Estos modelos, que podríamos llamarlos desde un punto de vista práctico «metamodelos», se adaptan de una forma muy sencilla a nuestro problema concreto, siempre y cuando conozcamos el proceso, que ya adelantamos en sencillo.

Ejemplos de redes preentrenadas son AlexNet o VGG16. En este caso vamos a escoger el modelo VGG16, ampliamente conocido, y que se ha entrenado además sobre un conjunto de imágenes, ImageNet, con lo cual puede resultarnos, a priori, positivo.

Aunque no considero excesivamente relevante conocer, a nuestro nivel, los detalles de estas complejas redes, sí podemos echar un vistazo a su arquitectura desde el punto de vista de la elaboración de un modelo basado en CNN. Esta arquitectura se muestra en la figura 6, y en ella vemos dos grandes fases: las convoluciones (que son las que extraen y preparan las características de las imágenes de entrada) y la fase de clasificación (capas *Dense*, como sabemos y cabe esperar).

En la subsección correspondiente al experimento asociado a este «algoritmo» daremos algunos detalles más concretos sobre cómo la hemos utilizado y por qué.

3 Estudio experimental

En esta sección presento un breve estudio experimental con diversas configuraciones de los algoritmos expuestos anteriormente.

Siguiendo el «guión» explicativo aportado a los alumnos, empezaré con un ejemplo de RandomForest, seguiré con dos modelos de CNN y finalizaré con una red preentrenada basada en la red VGG16, disponible en Tensorflow.

Los resultados de los experimentos se muestran al final, para poder hacer el análisis (comparativo) más cómodamente y facilitar su comprensión.

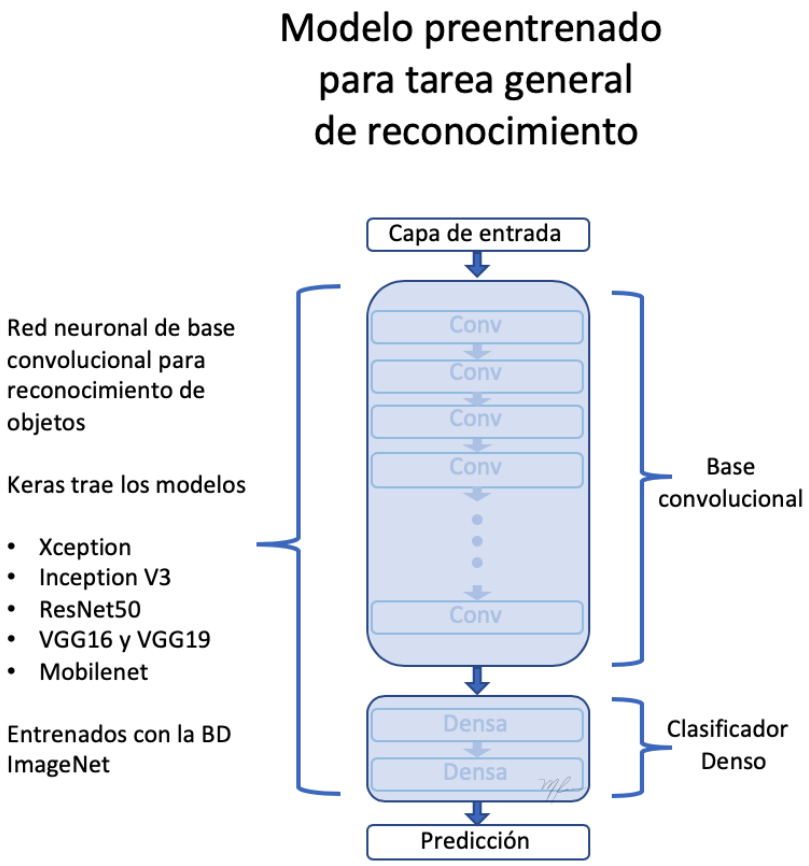


Figura 6: Arquitectura *intuitiva* de VGG16. Fuente: [7]

3.1 Experimento 1: Random Forest

Este experimento se basa en uno de los expuestos en el benchmark de la web oficial del proyecto Fashion MNIST de Zalando [1].

Se ha escogido una configuración de parámetros concreta de *Random Forest*, por su escaso tiempo de ejecución, porque además ha ofrecido, dentro de esta primera familia de algoritmos clásicos de ML, una de las mejores tasas de acierto, y porque sabemos que en general es un algoritmo de *ensemble* que suele ofrecer buenos resultados en muchas situaciones.

Los detalles de código que merece la pena reseñar son:

- No se ha aplicado ninguna estrategia de preprocesado.
- La generación del modelo se ha realizado así:

```
clf = RandomForestClassifier(criterion='gini',max_depth=100,n_estimators=100)
    .fit(X_train, y_train)
```

  donde se aprecia que:

  - El criterio para el peso asociado a una partición es el por defecto (*gini*).
  - La profundidad máxima está limitada, seguramente para evitar el sobreaprendizaje.
  - El número de clasificadores no es muy elevado, lo que implicará que el tiempo de ejecución no sea excesivamente alto.

3.2 Experimento 2: Red neuronal convolucional con 2 capas + *Pooling*

Siguiendo las recomendaciones del documento explicativo de la práctica, aplicamos en este segundo experimento un modelo de redes neuronales convolucionales (CNN). Este modelo, a diferencia del resto, no ha sido extraído de un autor externo, si bien he tomado como guía, y modificando ligeramente su arquitectura, el de la referencia [5].

Las principales características del modelo se explican básicamente observando su arquitectura, y teniendo como referencia la figura explicativa de una CNN "genérica" (revisar figura 6):

1. Capa convolucional de 32 'feature maps' con kernel de tamaño 5x5
2. Capa de pooling usando la estrategia max de tamaño 2x2
3. Capa convolucional de 64 'feature maps' con kernel de tamaño 5x5
4. Capa de pooling usando la estrategia max de tamaño 2x2
5. Capa dropout para "apagar" algunas neuronas (el 25 %) y evitar un posible sobreaprendizaje.
6. Capa Flatten para convertir los resultados anteriores a formado 1D (feature maps)
7. Una capa Dense (red totalmente conectada)
8. Una capa Dropout
9. Una capa Dense (para clasificación final)

Se han tomado tamaños del kernel usuales, en base a la literatura (3x3), también para el *pooling* (2x2).

En concreto, el código para generar esta red ha sido:

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
    input_shape=(img_rows, img_cols, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(n_classes, activation='softmax')) # capa clasificación
```

3.3 Experimento 3: Red neuronal convolucional con 2 capas + *Pooling*

Este experimento es similar al anterior, pero ahora se ha añadido una capa convolucional extra para observar el efecto que tiene añadir mayor capacidad de «visión» a nuestro modelo, y cómo afecta esto. El modelo se inspira en el propuesto en [6], y también cabe destacar que el tamaño del kernel es mayor (5×5) que en el experimento 2.

Como ya es preceptivo, mostramos el código de generación del modelo:

```
model = Sequential()
model.add(Conv2D(32, (5, 5), input_shape=(img_rows, img_cols, 1),
       padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))

model.add(Conv2D(64, (5, 5), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))

model.add(Conv2D(128, (1, 1), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))

model.add(Flatten())

model.add(Dense(1024, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax')) # capa de clasificación
```

3.4 Experimento 4: Red preentrenada a partir de VGG16

A partir de lo explicado en la subsección «2.3 Red preentrenada a partir de VGG16», se ha desarrollado un ejemplo simple de construcción de un modelo de red neuronal convolucional (CNN) a partir de la red genérica preentrenada VGG16. El procedimiento se llama *transferencia de conocimiento* y ofrece como principal logro el poder aprovechar el potencial de modelos ya entrenados con sofisticadas capas convolucionales para poder adaptarlo a nuestro problema concreto. Ahora bien, ¿cómo se produce esta adaptación?

Yo he empleado, para esta adaptación, el enfoque que considero es más simple e intuitivo. Se basa en contruir un modelo CNN como hasta ahora, solo que en lugar de añadir capas convolucionales, cargamos como primera capa el modelo preentrenado VGG16. A continuación, añadiríamos

una capa de tipo *Flatten*() para asegurarnos que las características salen con el formato correcto y posteriormente, como habíamos hecho en los experimentos anteriores y la arquitectura CNN refiere, añadiríamos las capas para la parte de clasificación (capas Dense, en esencia).

En la figura 7-derecha muestro la arquitectura del nuevo modelo que subyace de lo que acabamos de plantear, y a la izquierda, la arquitectura de VGG16 (ya mostrada en la figura 6). En la arquitectura de la derecha apreciamos lo que estábamos comentando: una base de capas convolucionales (VGG16) ya «empaquetadas» y configuradas, y una segunda etapa con el clasificador entrenable a partir del conjunto de datos de nuestro problema (Fashion MNIST).

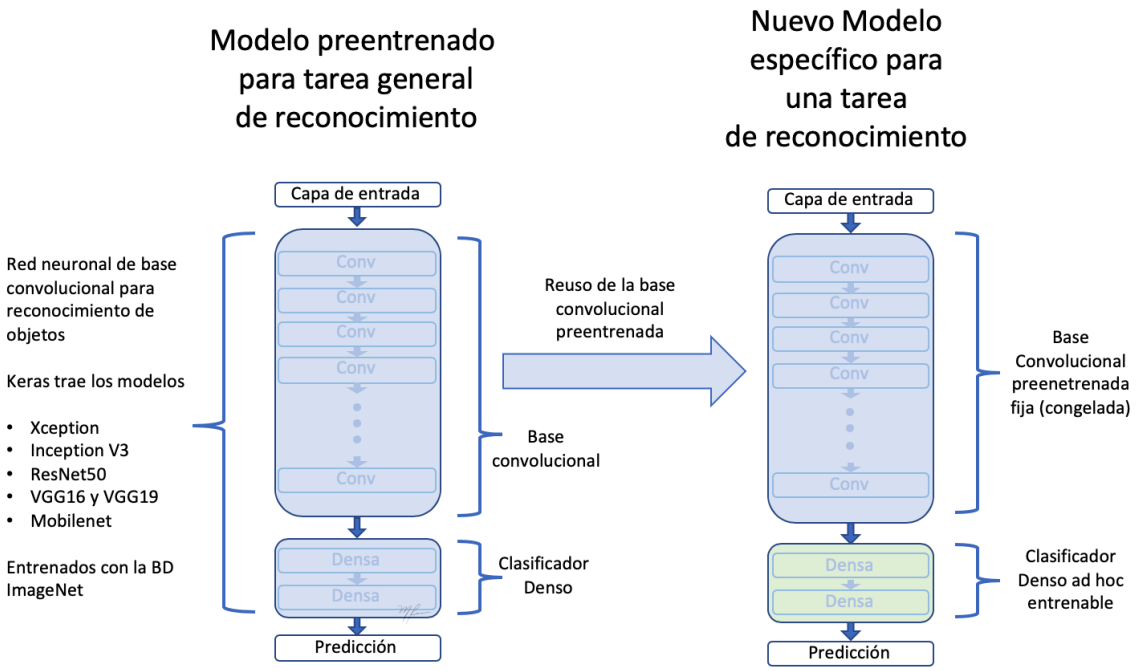


Figura 7: Arquitectura *intuitiva* del modelo de red preentrenada (derecha) junto con el VGG16 (izquierda). Fuente: [7]

En el código asociado a este experimento se realizan las siguientes tareas básicas[2]:

1. Importar los conjuntos de entrenamiento y prueba.
2. Convertir a 3 canales (R-G-B) las imágenes para ser aceptadas por VGG16 (requisito).
3. Reestructurar el formato de las imágenes, como ya habíamos hecho en el resto de experimentos.
4. Redimensionar las imágenes a tamaño aceptable por VGG16 (otro requisito de este modelo, ya que no lo hemos elaborado nosotros).
5. Normalizar las imágenes y reconvertir las listas de etiquetas (*y_train* e *y_test*) (como ya habíamos hecho en el resto de experimentos de CNN.
6. Se carga el modelo VGG16:

[2]Algunas funcionalidades auxiliares para hacer el *resize* de imágenes, manipular formatos, etc están basadas en las usadas por [9].

```
conv_base = VGG16(weights='./models/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5',
    include_top=False,
    input_shape=(IMG_HEIGHT, IMG_WIDTH, IMG_DEPTH) )
```

7. Creamos nuestra red a partir del modelo VGG16 ya cargado y dos capas *Dense*, siendo la última, como sabemos, con la función *softmax* habitual para clasificación.

```
model = Sequential()
model.add(conv_base)      # modelo base agregado como una capa!
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

8. El resto ya son tareas conocidas: compilar el modelo y hacer el entrenamiento (*fit*).

Y con esto, ya tenemos nuestra red preentrenada adaptada al problema Fashion MNIST.

3.5 Análisis (comparativo) de resultados

En este apartado vamos a analizar brevemente los resultados obtenidos en los distintos experimentos, junto a algunas visualizaciones.

En primer lugar, mostramos la tabla comparativa de resultados de los experimentos realizados (ver tabla 1).

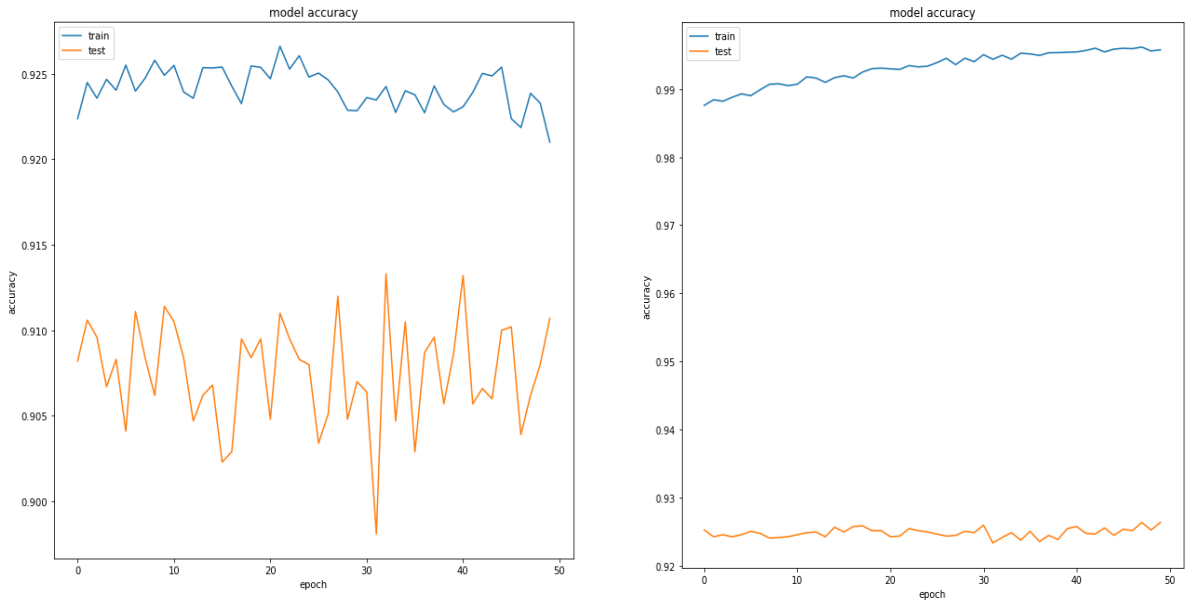
Experimento#	¿Preprocesado?	Acierto en train	Acierto en test/validación
1	No	0,881	0,884
2	Sí, normalización CNN	0,9359	0,9105
3	Sí, normalización CNN	0,9802	0,93
4	Sí.	0.93127	0.9086

Cuadro 1: Tabla de resultados de los experimentos realizados.

Lo que podemos ver es que hemos seguido una estrategia progresiva de mejora. En el modelo de Random Forest teníamos una tasa de acierto de 0,88 aproximadamente sobre el conjunto de test (unas 10.000 imágenes). Posteriormente, con el primer experimento de redes convolucionales, que incluía dos convoluciones y pooling tras cada una de ellas, la tasa de acierto, como era de esperar, mejoró: de 0,88 a 0,91. Finalmente, la segunda versión de CNN, que ya incluía una nueva capa convolucional (lo que, en principio, va a aportar más capacidad de "visión" al modelo) ha seguido mejorando el acierto, hasta el 0,93. Como vemos, no son aumentos demasiado bruscos, sobre todo entre los dos experimentos de CNN, porque la única gran diferencia entre ellos es que el segundo tiene una capa más que el primero, y que se han cambiado algunos tamaños de kernel y pooling, básicamente. Sí que hay más diferencias con respecto al *RandomForest*, como ya anunciábamos.

En cuanto a los resultados de la *red preentrenada* (Experimento 4), vemos que no los ha ofrecido mejores que hasta ahora. Es posible que el número de épocas o incluso el tamaño de las particiones (*batch*) haya influido negativamente, si bien las tasas de acierto han estado muy cerca de los experimentos basados en CNN previos (0,9 tanto en train como en validación/test). También hay que tener en cuenta que no hemos eliminado neuronas antes de la capa de clasificación final, lo cual puede haber provocado algún efecto de sobreaprendizaje.

A continuación mostramos en la Figura 8 una gráfica para cada uno de los dos experimentos de CNN realizados (Experimentos 2 y 3).



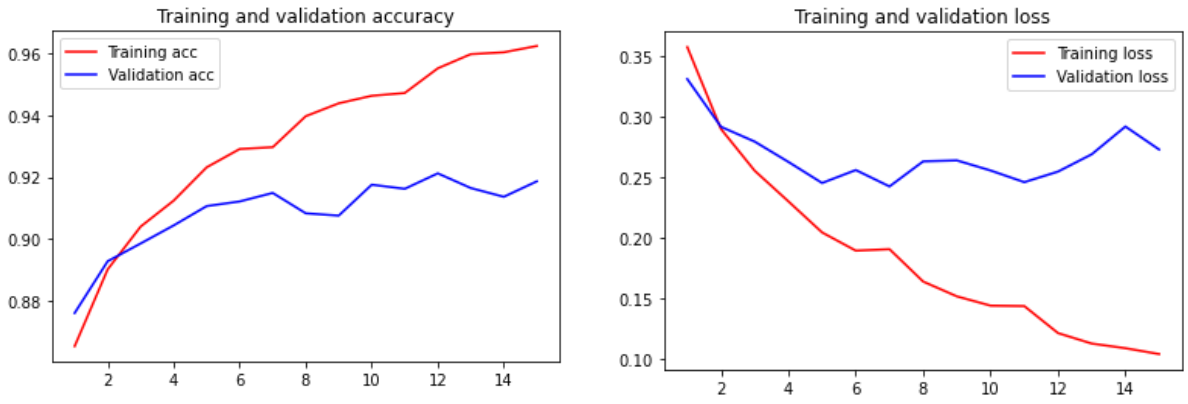
(A) Experimento primero de CNN (B) Experimento segundo de CNN

Figura 8: Comparativa de gráficas de *accuracy* en los diferentes *epochs* para los dos experimentos de CNN.

En la gráfica (A), correspondiente al primer modelo CNN (2 convoluciones), vemos que las diferencias entre el acierto en la validación y el test no han sido demasiado altas, a pesar de la sensación que da la gráfica; de hecho, si nos fijamos en el eje Y, vemos que la diferencia entre ellas, de media, está en las centésimas (0,92 y 0,91 aproximadamente). Esto indica que no ha habido prácticamente sobreaprendizaje, si bien vemos oscilaciones más pronunciadas (altibajos) en la gráfica de *test* (naranja) que de *train* (azul).

Por su parte, en la gráfica (B), que se corresponde con el modelo CNN con 3 capas convolucionales y una capa *Dense* adicional, la evolución del acierto ha sido mucho más homogénea, tanto en el conjunto de validación como en la evaluación (test), pues no apreciamos apenas ≪picos≫ en ambas curvas. Lo que sí se aprecia es que en este caso el modelo se ha comportado algo peor prediciendo el test que el conjunto de validación. A pesar de todo, la tasa de acierto sobre el conjunto de test del modelo último (B) es, como ya veíamos en la tabla 1, mejor que el modelo (A).

Mostramos también a continuación, en la figura 9, las gráficas de evolución del *accuracy* (acierto) y el *loss* (error) a lo largo de los diferentes *epochs* en el train y la validación de forma comparada. En ellas vemos que conforme se van ejecutando las épocas, va creciendo el acierto y va disminuyendo el acierto. Pero lo interesante es ver que esta tendencia, para el caso del train se mantiene constante, sin momentos de descenso reseñables, mientras que para el conjunto de validación llega un momento en que el crecimiento de la tasa de acierto y el descenso del error se estabilizan ambas dos, lo cual indica que deja de mejorar en ese punto. A pesar de esto, me quedo con lo ≪positivo≫ de esta red preentrenada, que hemos conseguido un acierto del 90,86 % en el conjunto de validación (y 93,13 % en train) y que posiblemente con algún pequeño ajuste (añadiendo la capa Dropout antes de la capa final *Dense*, por ejemplo, para evitar un posible sobreaprendizaje) podamos superar de lejos al resto de experimentos realizados.



(A) Evolución de la tasa de acierto

(B) Evolución del error (loss)

Figura 9: Comparativa de gráficas de *accuracy* y *loss* en los diferentes *epochs* para la red preentrenada (Experimento 4).

4 Planteamiento de futuro

Si visualizamos globalmente el contenido de este proyecto, nos damos cuenta de algo: las redes neuronales convolucionales (CNN) han acaparado buena parte de nuestro tiempo. Empezábamos en el experimento número 2 diseñando un modelo CNN básico, con dos capas, seguíamos en el tercero añadiendo más capas y cambiando algunos valores, y en el último experimento con la red preentrenada, que también se fundamentaba en CNN. Además, en el artículo donde se proponía un estado del arte para Fashion MNIST también se centraban en modelos de CNN. Así las cosas, ¿por qué no seguir en esta línea?

En la referencia [12] se hace una propuesta interesante, sobre todo para nosotros, por el contenido de la asignatura. En ella, se plantea sustituir el criterio de clasificación habitual de la última capa de CNN, softmax, por un algoritmo conocido: las Máquinas de Soporte Vectorial, debido a unos resultados obtenidos en otro artículo científico [13], más genérico, donde el nuevo modelo CNN-SVM (CNN con el clasificador de SVM) ofrecía resultados prometedores sobre problemas similares que el conocido CNN-softmax (es decir, con el clasificador *típico* de CNN).

De esta forma, los autores intentaron comprobar sobre MNIST y sobre Fashion MNIST si realmente podía producirse esa mejora. Elaboraron un modelo CNN sencillo con dos capas y lo probaron sobre ambos conjuntos con un criterio de clasificadón distinto cada uno, los dos a comparar:

- CNN-softmax: 2 capas convolucionales y una clasificación con criterio *softmax*.
- CNN-SVM: 2 capas convolucionales y una clasificación basada en SVM.

Pero los resultados no fueron satisfactorios, y el rendimiento que les produjo CNN-SVM fue inferior al validarlo sobre el conjunto de *train* y *test* (de ambos), aunque las diferencias no fueron demasiado significativas (ver la gráfica de la figura 10).

Entonces, **¿cuál es la propuesta de trabajo?** Partiendo de la base de que los modelos que se usaban en este artículo eran demasiado simples, tal y como reconocían sus propios autores, podemos implementar modelos algo más sofisticados en base a los conocimientos que se han adquirido durante la asignatura, y especialmente durante el desarrollo del presente trabajo y al trabajo sobre la literatura. Recordemos que disponemos de técnicas como la validación cruzada,

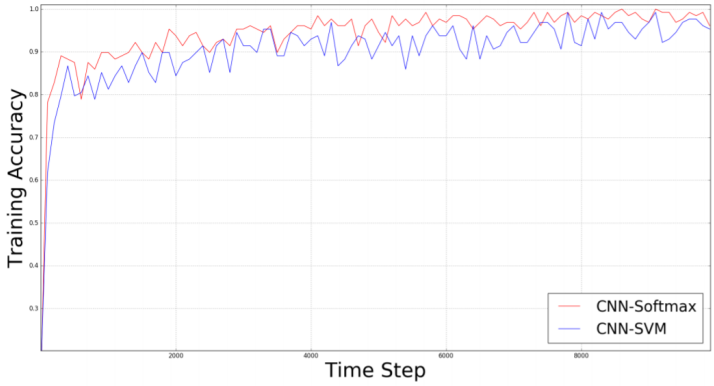


Figura 10: Gráfica de accuracy sobre test y train en los experimentos de referencia de [12]. Fuente: [12].

que se ha usado en este trabajo, para evaluar los modelos que vamos generando partiendo de un conjunto de entrenamiento y un conjunto de validación.

Además, en este documento se ha presentado un ejemplo (simple, pero ilustrativo) de adaptación de redes preentrenadas, las cuales sabemos que son muy sofisticadas y han sido entrenadas sobre grandes conjuntos de datos, con lo que se nos abre aquí otra línea de indagación y de experimentación a la búsqueda de esa mejora.

Los autores han puesto el código abierto en la plataforma GitHub [14] muy bien documentado, lo que puede ser una buena referencia para comenzar.

Referencias

[1] Página oficial de Github para Fashion MNIST: `https://github.com/zalandoresearch/fashion-mnist`

[2] `https://datasmarts.net/es/que-es-fashion-mnist/`

[3] Página sobre los fundamentos de la práctica: `https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/`

[4] `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networ`

[5] `https://github.com/abelusha/MNIST-Fashion-CNN`

[6] `https://github.com/umbertogriffo/Fashion-mnist-cnn-keras/`

[7] `http://personal.cimat.mx:8181/~mrivera/cursos/aprendizaje_profundo/preentrenadas/preentrenadas.html`

[8] S. Bhatnagar, D. Ghosal and M. H. Kolekar, «Classification of fashion article images using convolutional neural networks», 2017 *Fourth International Conference on Image Information Processing* (ICIIP), Shimla, 2017, pp. 1-6, doi: 10.1109/ICIIP.2017.8313740.

[9] `https://www.kaggle.com/anandad/classify-fashion-mnist-with-vgg16`

[10] `https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural`

[11] https://stats.stackexchange.com/questions/56950/neural-network-with-skip-layer-connections

[12] A. Fred, «An Architecture Combining Convolutional Neural Network (CNN) and Support Vector Machine (SVM) for Image Classification», 2019. Accesible en: https://arxiv.org/abs/1712.03541

[13] Y. Tang, «Deep Learning using Linear Support Vector Machines». Accesible en https://arxiv.org/pdf/1306.0239.pdf

[14] https://github.com/AFAgarap/cnn-svm