E.T.S. de Ingeniería Industrial, Informática
y de Telecomunicación

Aplicación de filtros de concurrencias en redes neuronales convolucionales para la transferencia de estilos en imágenes.

Grado en Ingeniería Informática

Trabajo Fin de Grado

Alumno: Asier Andueza Ibarrola

Director: Miguel Pagola Barrio

Pamplona, 21 de enero de 2022

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Índice

1. Resumen

El style transfer es una técnica de relativamente novedosa y consiste en generar una imagen en base a una fotografía y a una imagen artística, resultando esa imagen en una mezcla del contenido de la fotografía y el estilo de la imagen artística. Esta técnica puede resultar útil en diferentes ámbitos, desde editores de foto y video hasta videojuegos y realidad virtual.

La manera más común de obtener un algoritmo de transferencia de estilos es mediante el uso de redes neuronales convolucionales, uno de los modelos de aprendizaje automático más potentes a la hora de tratar con imágenes, por su capacidad de extracción de determinadas características de las imágenes que les demos, la cual podemos modificar para extraer lo necesario de nuestras dos imágenes y obtener con ello la mezcla deseada. Añadido a este modelo, se va a tratar también con tensores de coocurrencias, que son herramientas muy potentes en el campo de la detección de imágenes (o image retrieval).

El objetivo primario de este trabajo va a ser obtener un algoritmo de style transfer modificado frente al original incluyendo el concepto del tensor de coocurrencias y comparar sus resultados con el algoritmo original mediante la generación de transferencias de estilos con un grupo de imágenes dadas.

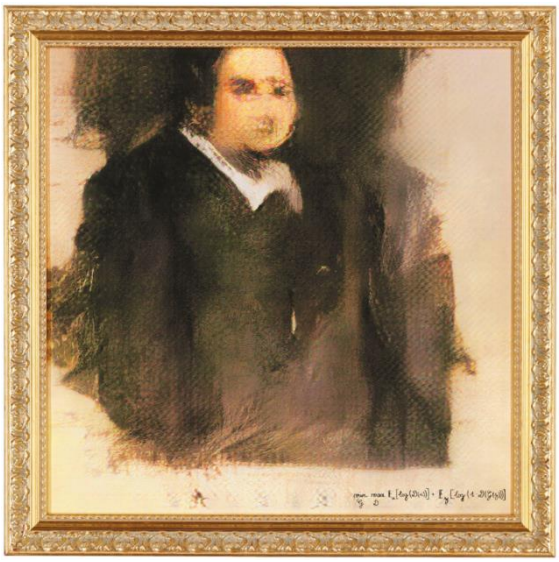
Palabras Clave: Red neuronal convolucional, Tensor de coocurrencias, Transferencia de estilos, aprendizaje automático

2. Introducción

La visión artificial es uno de los campos de investigación en auge con más capacidad de exploración y expansión del momento, hoy en día se genera cada día más contenido digital que nunca y a éste se le pueden dar aplicaciones muy útiles, pero para ello se requiere de técnicas potentes y ágiles en el procesamiento de este contenido. Solo en el campo que a imágenes se refiere puedes mencionar un gran número de usos que se les puede dar a las mismas mediante técnicas de visión artificial y aprendizaje automático que podemos utilizar diariamente: métodos de reconocimiento facial para, por ejemplo el desbloqueo de un teléfono inteligente, maneras de generar y recuperar el texto que puede contener una imagen, múltiples algoritmos que asisten en la detección de enfermedades, etc.

En este trabajo, por ejemplo, aplicamos procesos de visión artificial y de machine learning en una técnica relativamente novedosa llamada 'Style Transfer' (o transferencia de estilos), la cual consiste en mezclar el contenido de una fotografía o imagen de tu elección con una imagen artística de tu elección también (un cuadro, un dibujo, incluso una textura), con el objetivo de generar una imagen con los elementos clave de la primera imagen pero representados de una manera semejante al estilo artístico que se ha proporcionado con la segunda imagen y a su vez intentar obtener otros modelos viables introduciendo técnicas de búsqueda de imágenes como los tensores de coocurrencias.

A priori, en comparación con los ejemplos de aplicaciones de visión artificial que se han mencionado anteriormente, puede parecer una técnica intrascendental, pero aplicaciones de visión artificial como esta en el mundo del entretenimiento cada día están ganando más y más tracción. Sin ir más lejos, el 25 de octubre de 2018, el 'Retrato de Edmond Belamy', se subastó y vendió por más de 435.000 dólares, retrato compuesto y generado en su totalidad por una red neuronal generativa entrenada en esa tarea mediante el procesamiento de 15000 retratos pintados entre el siglo XIV y el siglo XX [1].



Retrato "Edmond de Belamy, from La Famille de Belamy", generado mediante una red neuronal generativa adversaria. Fuente: https://www.christies.com/lot/lot-edmond-de-belamy-from-la-famille-de-6166184/?from=salesummery&intobjectid=6166184&sid=18abf70b-239c-41f7-bf78-99c5a4370bc7

Acontecimientos como este dejan claro que la revolución de la inteligencia artificial en el mundo del arte y del entretenimiento es cuestión de tiempo, si no ha llegado ya. Técnicas como el style transfer, en su versión más simple, es una herramienta fantástica para que artistas presten su visión creativa a gente no tan capacitada en esos campos para experimentar con sus propias capacidades creativas. Además, si queremos llevar esta técnica más lejos, desarrollada lo suficiente, puede ser aplicable a muchos campos. El más simple puede ser su aplicación en editores de fotos y vídeos hasta ámbitos más complejos de creación que incluyan procesos artísticos como pueden ser videojuegos o incluso realidad virtual.

En este último campo, compañías como Facebook (ahora Meta) buscan de estas técnicas para mejorar lo que ellos llaman 'visual storytelling' (o narración visual) en realidad virtual, [2] tema del que se va a hablar ahora más y más con este mundo virtual o 'metaverso' que buscan desarrollar y promocionar como la revolución social definitiva.

Está claro que en el panorama social actual el entretenimiento es rey y más en el estado del arte en estos momentos, con esta visión global, en este trabajo lo que se busca es aplicar estas técnicas de visión artificial y aprendizaje automático aprendidas durante el grado de ingeniería informática e incluir y modificar técnicas más avanzadas como el style transfer en redes neuronales convolucionales, de maneras que se definirán en más detalle en los objetivos de este trabajo.

3. Objetivos

Este trabajo nace con la idea de aprender y profundizar en un tipo de red que abre y evoluciona la investigación en imágenes, que son las redes neuronales convolucionales, apoyado en un conocimiento ya existente en redes neuronales aprendido durante el grado. Estas redes son las más potentes en el campo de visión artificial y las que más avances han llegado a aportar en técnicas de extracción de características en imágenes, hasta llegar a simular comportamientos muy similares a las neuronas de un cerebro biológico correspondientes a la corteza visual primaria.

El objetivo principal de este trabajo es obtener un modelo capaz de realizar la transferencia de estilos mediante redes neuronales convolucionales e incorporarle un tensor de coocurrencias, (técnica de búsqueda de imagen la cual se profundizará en su concepto junto con todos los demás más adelante en el trabajo) con la idea de comparar los resultados de ambos modelos para un mismo grupo de imágenes de contenido y estilo con tal de tratar de determinar cuál de ellos es mejor.

Con esto en mente, este trabajo se va a dividir en cinco partes marcadas. La primera definir el marco teórico de los conceptos que se van a tratar en este trabajo, empezando desde conceptos más simples como el perceptrón, evolucionarlos hacia la red neuronal, definir conceptos y procesos intrínsecos de la misma, mencionar tipos de redes neuronales y profundizar en la red neuronal convolucional y conceptos relacionados con la misma, seguido por su proceso de ejecución y finalmente terminar esta explicación teórica hablando del artículo que define el style transfer y el que define el tensor de coocurrencias. Más adelante se hablará de la propuesta de cambio que se le va a dar al style transfer aplicando tensores de coocurrencias y finalmente se hablará de las herramientas y la implementación realizada en este trabajo, así como presentar y explicar el porqué de los resultados finales obtenidos.

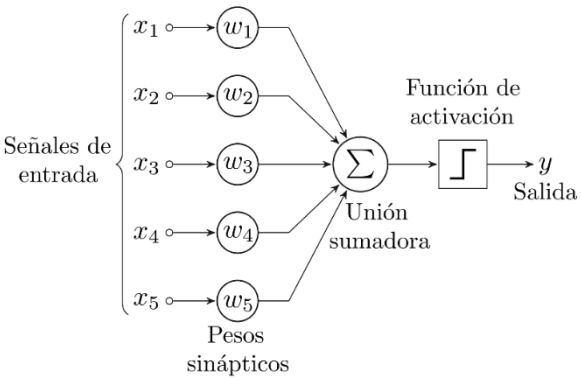
Entrando en detalle en esta última parte, se va a hablar de cómo se genera un modelo a partir de la arquitectura de una red muy conocida para la extracción de características de una imagen, la red vgg19, y de qué manera se introducen los módulos necesarios para adaptar la red para aplicar el style transfer. Mas allá se empieza con las diferentes maneras de obtener los resultados y la explicación del porqué de las transferencias de estilos obtenidas y acabando con una explicación más en detalle del rol que toma uno de los componentes más importantes en el style transfer, la matriz de Gram, así como el rol que toma un tensor de coocurrencias a la hora de extraer características con respecto a la matriz de Gram.

4. Teoría

4.1 Introducción teórica. El perceptrón y la Red Neuronal

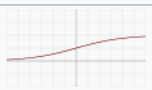
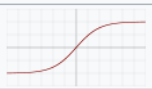
En este trabajo se van a trabajar con redes neuronales convolucionales (CNN en adelante), pero antes de definir una CNN vamos a construir sobre los conceptos más básicos de una red neuronal, comenzando por el perceptrón.

El perceptrón nace de adoptar el comportamiento de una neurona biológica al ámbito informático, y es el componente básico de cualquier red neuronal. Un perceptrón está formado por una serie de señales de entrada, a cada cual se le asigna una importancia en forma de peso sináptico, todas estas señales ponderadas se agregan junto a un error (o bias) y en base a ese resultado, ayudándose de una función de activación, el perceptrón devuelve un valor. [3]



Esquema de un perceptrón con 5 entradas. Fuente: https://es.wikipedia.org/wiki/Perceptrón

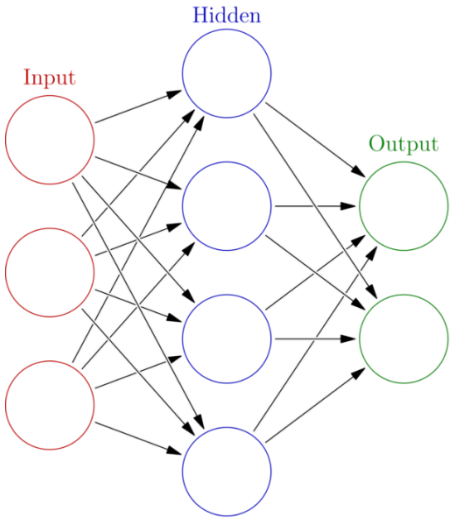
Respecto a las funciones de activación, éstas son "criterios" que aplicamos al perceptrón para dar la salida. Por ejemplo si buscamos clasificar nuestro ejemplo como 'verdadero o falso', podemos simplemente aplicar una función "*binary step*", si buscamos recibir una probabilidad de ocurrencia, en vez de un verdadero o falso, podemos utilizar una función de activación sigmoidea. Existen otras como la activación *"ReLU"* (Rectified Linear Unit), la cual ha demostrado ofrecer los mejores resultados en la extracción de características visuales, y es la que vamos a utilizar en este trabajo. [4]

Name	Plot	Function, $f(x)$
Identity		x
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$
Logistic, sigmoid, or soft step		$\sigma(x) = \frac{1}{1+e^{-x}}$
Hyperbolic tangent (tanh)		$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$
Rectified linear unit (ReLU)[10]		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max\{0, x\} = x\mathbf{1}_{x>0}$

Algunas funciones de activación, sus representaciones gráficas y sus definiciones matemáticas, sea x el resultado de la suma ponderada de nuestro perceptrón. Fuente: https://en.wikipedia.org/wiki/Activation_function

Una vez definido el perceptrón, podemos definir una red neuronal como una colección o conjunto de perceptrones definidos anteriormente, intentando simular una estructura neural en un cerebro biológico en la que cada neurona (perceptrón) puede recibir señales, procesarlas y transmitir la nueva señal al resto de neuronas conectadas.

Por lo general una red neuronal está compuesta por una serie de neuronas de entrada, podemos llamarlas capa de entrada de ahora en adelante, las cuales envían la información procesada con esas neuronas (o perceptrones) a capas intermedias de la red, las que llamamos capas ocultas. Las capas ocultas envían la información procesada por sus perceptrones hacia adelante a la siguiente capa. Cada conexión entre neuronas tiene un peso, como se ha definido en los perceptrones. La información avanza entre las capas, hasta llegar a la última capa, la capa de salida.



3. Representación de una red neuronal con tres perceptrones de entrada, una capa oculta con 4 perceptrones y una salida de dos perceptrones. Fuente: https://www.norwegiancreations.com/2019/04/introduction-to-neural-network/

Estas redes pueden entrenarse con conjuntos (previamente definidos) de elementos de entrada y su salida esperada, lo que clasifica las redes neuronales como algoritmos de aprendizaje supervisado, cuyo objetivo es dar salidas correctas a entradas relacionadas con el conjunto inicial de elementos. Para ello las redes se entrenan ajustándose los pesos mencionados anteriormente (las conexiones entre las neuronas), de manera que se aciertan el mayor número de escenarios posibles. [5]

Por otra parte, podemos mencionar también el aprendizaje no supervisado, definido según Wikipedia como un método de Aprendizaje Automático donde un modelo se ajusta a las observaciones. Se distingue del Aprendizaje supervisado por el hecho de que no hay un conocimiento a priori. En el aprendizaje no supervisado, un conjunto de datos de objetos de entrada es tratado. Así, el aprendizaje no supervisado típicamente trata los objetos de entrada como un conjunto de variables aleatorias, siendo construido un modelo de densidad para el conjunto de datos. [6]

Cabe mencionar también la función de salida, que es la función que se encarga de determinar el valor que se va a pasar a las siguientes neuronas. Si la función de activación está por debajo de un umbral determinado, ninguna salida se pasa a la neurona subsiguiente. Normalmente, no cualquier valor es permitido como una entrada para una neurona, por lo tanto, los valores de salida están comprendidos en el rango [0, 1] o [-1, 1]. También pueden ser binarios {0, 1} o {-1, 1}. La función que más se utiliza es la más sencilla además, cuya salida es la misma que la entrada, ésta se conoce como función de identidad. [7]

Una vez explicado lo básico, vamos a profundizar un poco en el cómo se ajustan esos pesos y en cómo aprende una red neuronal, para ello vamos a definir el algoritmo de backpropagation, que es el más famoso en estos casos. [8]

Este algoritmo busca encontrar valores de pesos óptimos para resolver el problema con un error mínimo y comienza aplicando propagación hacia adelante lo primero. Este proceso comienza con la inicialización de pesos de manera aleatoria. De esta manera, se puede calcular una salida y compararla con la salida real, y la diferencia puede reflejarse hacia los pesos como un error.

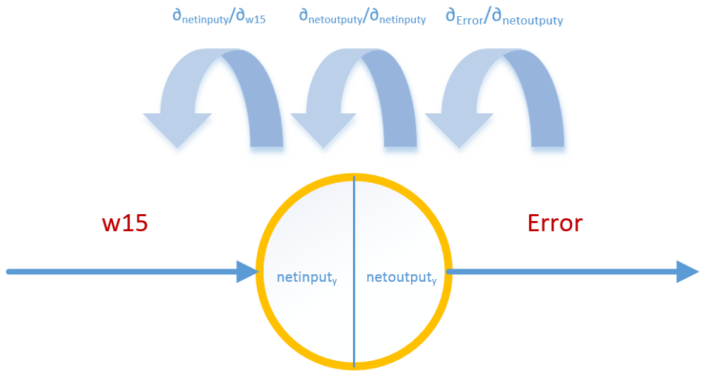
Una vez tenemos los pesos, el forward propagation o propagación hacia adelante, consiste en calcular las salidas de la red mediante el recorrido hacia adelante de salidas de las neuronas como entradas de las siguientes… Recorriendo la red hasta llegar al final y calcular esa salida. Si la salida es la esperada, implica que los pesos son óptimos y no debe de haber cambios, si no lo es, se procedería a cambiar la red.

Suponiendo que sea necesario cambiar los pesos de la red para mejorar la predicción, surge el cálculo del error para representar lo cerca o lejos que se está del resultado deseado. La opción más popular y sencilla es la utilización del error cuadrático medio (MSE por sus siglas en inglés), cuya fórmula se muestra a continuación:

$$MSE = \frac{1}{n}\sum_{i=0}^{n}(\bar{x}_i - x_i)^2$$

Siendo n el número de elementos, y los otros dos operadores el vector de las salidas obtenidas y la salidas reales o deseadas.

Una vez tengamos el error de la red, se van a modificar utilizando la técnica de propagación hacia atrás (o backpropagation) que básicamente es una técnica que busca los pesos óptimos basados en experiencias previas, por ello el error calculado en una iteración anterior se refleja en todos los pesos. Vamos a poner un ejemplo detallado en función de la siguiente neurona, que vamos a decir, por hacer los cálculos más simples, que se encuentra en la última capa de la red (la capa que está directamente conectada a la salida o resultado del modelo). Suponiendo la neurona que se muestra a continuación de una red la que sea y la fórmula de cálculo de error anterior (MSE).



4. Neurona de una red con un peso de entrada w15 y un error de salida. Fuente: https://sefiks.com/2017/01/21/the-math-behind-backpropagation/

Valiéndonos de la regla de la cadena, vamos a calcular cuánto ha aportado este elemento de la red, que recordemos que se haya en la última capa para simplificar el

ejemplo, en el error total del modelo, calculamos la derivada parcial del error en función de ese peso como se muestra a continuación:

$$\frac{\delta Error}{\delta w_{15}} = \frac{\delta Error}{\delta netoutput_y} * \frac{\delta netoutput_y}{\delta netinput_y} * \frac{\delta netinput_y}{\delta w_{15}}$$

Una vez tenemos $\frac{\delta Error}{\delta w_{15}}$, lo que podemos establecer como lo que contribuye ese peso al error total obtenido, vamos a modificar dicho peso. El resultado de la derivada parcial nos indicará si va a ser un cambio grande o pequeño y sumado a esto, vamos a introducir un nuevo concepto, el de la tasa de aprendizaje.

La tasa de aprendizaje es un valor λ, que nos indica cómo de rápido va a modificarse la red, esto significa que con una tasa pequeña damos menor posibilidad de variación del nuevo valor respecto al antiguo, pudiendo resultar en modelos más costosos de ajustar, mientras que una tasa grande cambiará más rápido pero con el riesgo de caer en mínimos locales.

Esta tasa de aprendizaje es importante es importante elegirla bien, ya que como se ha dicho antes, la derivada parcial del error respecto de ese peso puede dar un gran cambio a la red o no. De cualquier manera, calculamos el nuevo peso de la siguiente manera:

$$w'_{15} = w_{15} - \lambda * \frac{\delta Error}{\delta w_{15}}$$

Este proceso ha de ser repetido en todas y cada una de las neuronas que conforman la red, aumentando el cálculo añadiendo términos a la derivada parcial en función de la capa que nos encontremos conforme nos vayamos acercando más y más a la entrada de la red (por ello se ha representado el ejemplo con una neurona de la capa final). Ahora solo queda modificar los pesos hasta que la salida del modelo no pueda mejorarse más, lo que implica que el error ha sido reducido lo máximo posible.

Una vez definido el proceso que toma una red neuronal para aprender, cabe destacar que existen múltiples tipos de redes neuronales, cada una con una arquitectura base similar pero con cambios que las hacen útiles en determinadas situaciones:

- Red neuronal profunda: Una red neuronal con todas las capas ocultas completamente interconectadas
- Red neuronal generativa: Generalmente una agrupación de dos redes neuronales profundas para generar contenido nuevo inventado en base a contenido previo real
- Red neuronal bayesiana: mapean la relación entre eventos en términos de probabilidad. Muestra cómo la ocurrencia de ciertos eventos influye en la probabilidad de que ocurran otros eventos.
- Red neuronal convolucional: son el tipo de red con el que vamos a trabajar y se explicará con más profundidad a continuación.

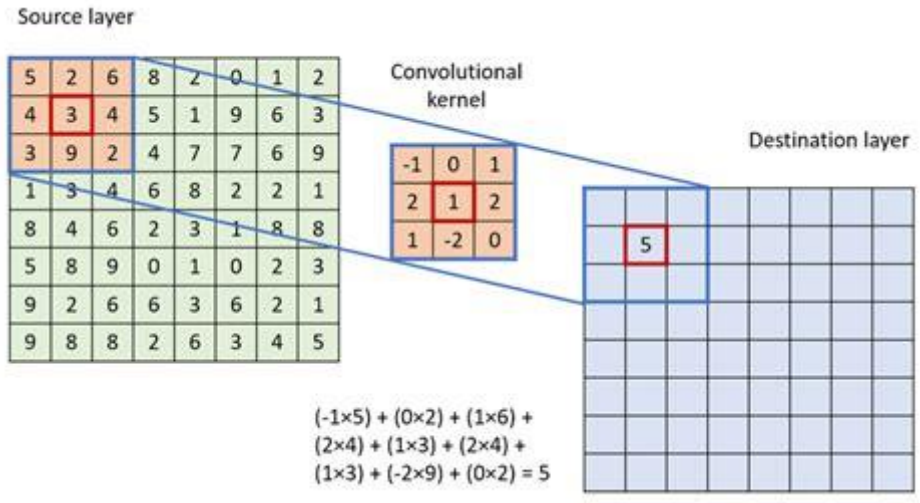
4.2. Explicación CNN

Son la clase de redes neuronales profundas más poderosas para trabajar en tareas de procesamiento de imagen. Puede decirse que las neuronas corresponden a campos receptivos de una manera muy similar a las neuronas de un cerebro biológico correspondientes a la corteza visual primaria. Su arquitectura consiste en múltiples capas de filtros convolucionales de una o más dimensiones, seguido por una reducción por muestreo y unas capas de perceptrones más sencillos al final para realizar la clasificación final, si fuera necesario. Vamos a entrar en detalle con estas fases a continuación.

4.2.1. Operación de convolución.

Partimos de la base de que la red recibe en la capa de entrada todos los píxeles de una imagen, por ejemplo, para una imagen de 100x100, será necesaria una capa inicial que tenga 10000 neuronas, suponiendo que sea una imagen a escala de grises, si tuviésemos una imagen a color tendríamos que triplicar la cantidad de neuronas. Sobre la imagen que introducimos realizamos la operación de convolución correspondiente.

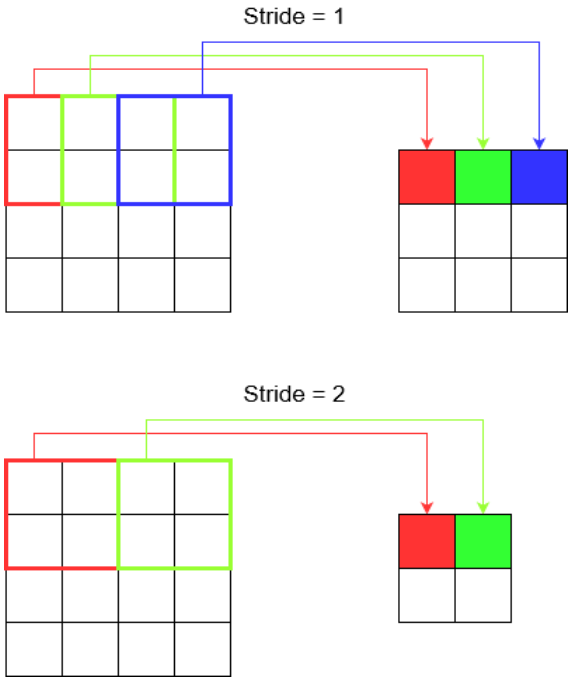
Una operación de convolución consiste en, para cada pixel, tomar una ventana de vecinos a su alrededor, con unos valores predefinidos, de un radio determinado, e ir operando el producto escalar entre los valores de la imagen original dentro de esa ventana y los valores predefinidos que tiene la ventana, a esta ventana se le conoce como kernel. [9]



4. Esquema de una operación convolucional. Fuente: https://www.researchgate.net/figure/Schematic-illustration-of-a-convolutional-operation-The-convolutional-kernel-shifts-over_fig2_332190148

4.2.2. Stride.

Esta ventana, o kernel convolucional, necesita de una variable conocida como 'stride', que se conoce como el número de 'pixeles' o elementos de la matriz que se mueven tras realizar cada operación convolucional. Un stride de 1 implica pasar la ventana de convolución elemento a elemento, un stride de 2 implica hacerla en un valor sí, otro no, etc.



5. Operación de convolución con diferentes valores de stride, con un stride 1 se puede observar que no hay pixeles con los que se realiza la convolución, mientras que con un stride de 2, hay algunos pixeles que no realizan su operación con todos sus vecinos.

Fuente: https://i.stack.imgur.com/XD2O4.png

4.2.3. Tipos de kernel

Existen muchos tipos de kernel, éstos, según los valores que contengan, sirven para extraer ciertas características de la imagen, pueden utilizarse para enfocar imágenes, hacerlas borrosas, realzarlas, perfilarlas, detectar bordes…

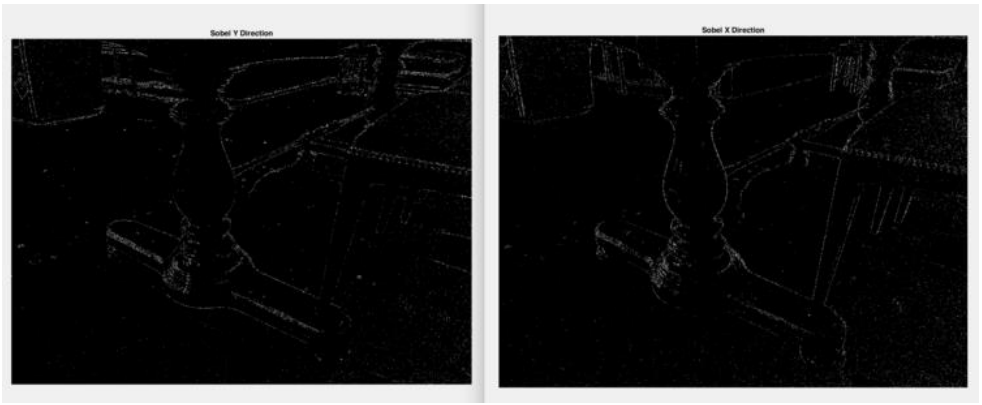
A continuación vamos a mostrar el ejemplo de lo que sería un par de operaciones de convolución para detectar bordes en una imagen en blanco y negro. Se hacen las dos convoluciones por separado para hallar los bordes verticales y horizontales de la imagen y luego se agregan ambos resultados

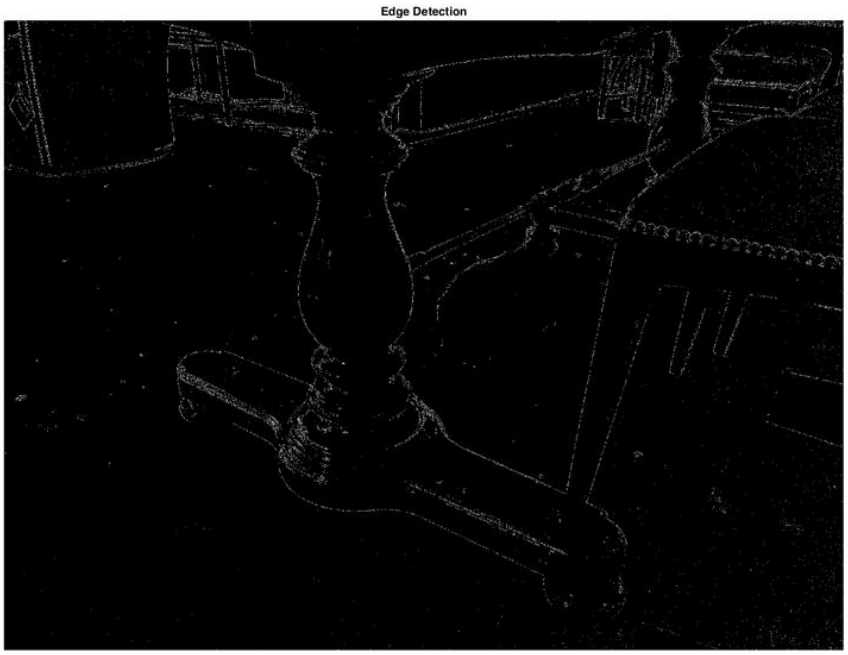
X – Direction Kernel

-1	0	1
-2	0	2
-1	0	1

Y – Direction Kernel

-1	-2	-1
0	0	0
1	2	1



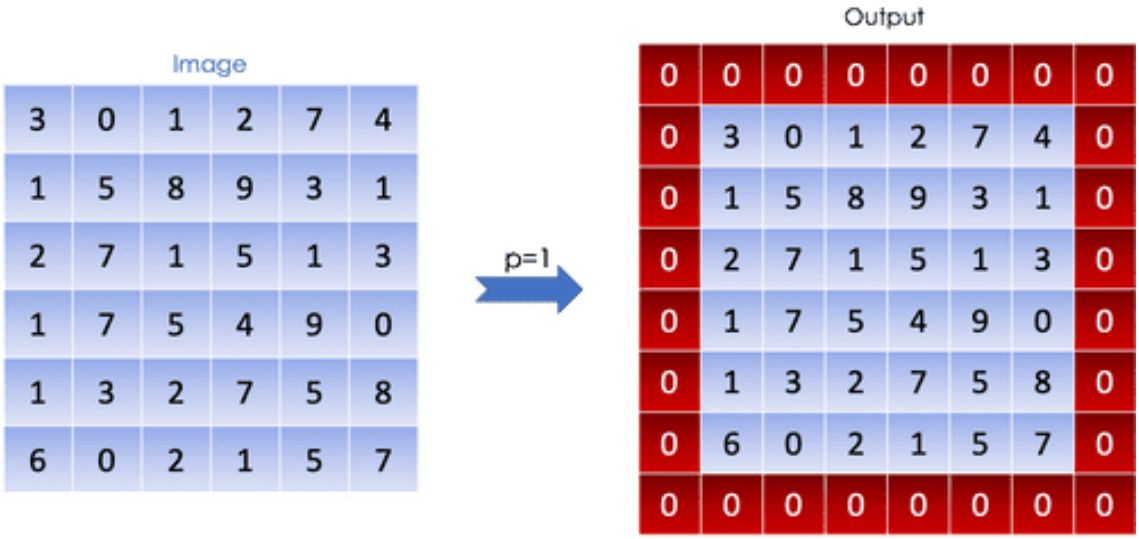


6. Kernels de Sobel para la detección de bordes verticales y horizontales para una imagen en blanco y negro y más tarde agregados un ejemplo. Fuente: https://www.projectrhea.org/rhea/index.php/An_Implementation_of_Sobel_Edge_Detection

4.2.4. Padding. Técnicas de padding

En la imagen que muestra la operación de convolución, podemos observar cómo se realiza la operación de convolución para el valor de la matriz rodeado en rojo, pero ¿Qué pasaría si se hiciese en uno de las filas o columnas que representan los bordes de la matriz? Puesto que el kernel convolucional no tiene valores para multiplicar los valores que se 'queden fuera' de la matiz al rodear ese elemento, para buscar una solución a este problema, surge la técnica de 'padding'.

Hay muchas maneras de realizar esta técnica, pero por lo general consiste en añadir o quitar las filas y columnas que resulten un problema en la imagen. Por ejemplo podemos ignorar los elementos de la matriz que no tengan suficientes vecinos como para realizar una convolución propiamente. O bien añadir ceros a los bordes de la matriz equivalentes al radio del kernel convolucional y empezar a deslizar la ventana por todos los valores de la imagen original. Esta última es la más conocida y se conoce como 'zero-padding'.



6. Ejemplo de Zero padding para una matriz, suponiendo un kernel 3x3 (radio/p = 1): Fuente: https://www.ismailmebsout.com/Convolutional%20Neural%20Network%20-%20Part%201/

4.2.5. Función de activación ReLU

Ahora bien, una vez que sabemos lo que es un kernel, llamamos filtro convolucional a un conjunto de kernels, cada capa convolucional de la red contiene una serie de filtros, que a su vez está formado de diferentes kernels y cada uno de ellos extrae una característica en concreto de la imagen, lo que significa que la salida de una capa específica consiste en diferentes imágenes filtradas de la imagen de entrada, lo que llamamos mapas de características.

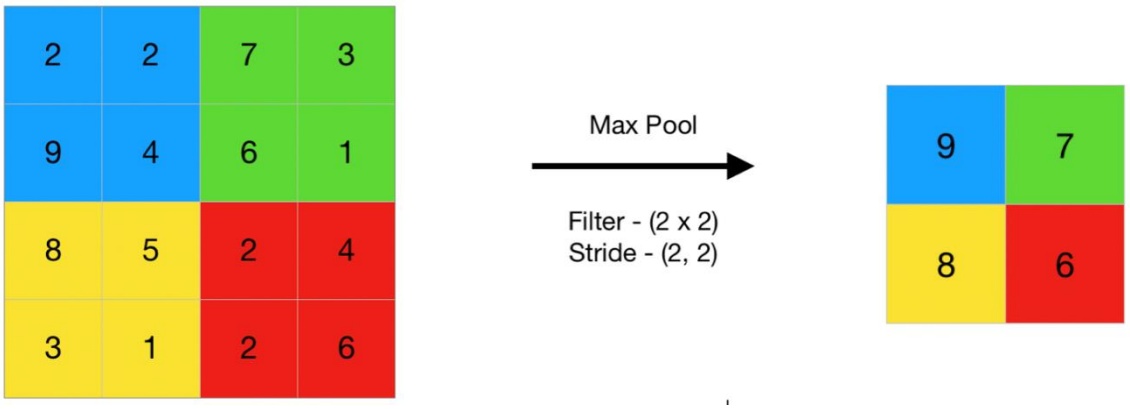
Una vez obtenemos un mapa de características, aplicamos lo que hemos definido anteriormente como función de activación, en este caso aplicando la función ReLU, que consiste en coger los valores negativos de nuestro mapa y hacerlos cero, lo cual ayuda bastante en trabajos de procesamiento de imagen.

4.2.6. Pooling. Tipos de pooling

Una vez hemos realizado las operaciones que hayamos incluido en la capa de convolución, vamos a tomar las neuronas más representativas, lo cual es necesario porque tras cada capa aumentamos muchísimo las neuronas necesarias para procesar la siguiente capa, y puede llegar a ser computacionalmente inviable, por lo que cogiendo las neuronas más representativas, preservamos las características más importantes que detectó cada filtro. Hay diferentes maneras de hacer este muestreo, voy a repasar dos, max pooling y average pooling. [13]

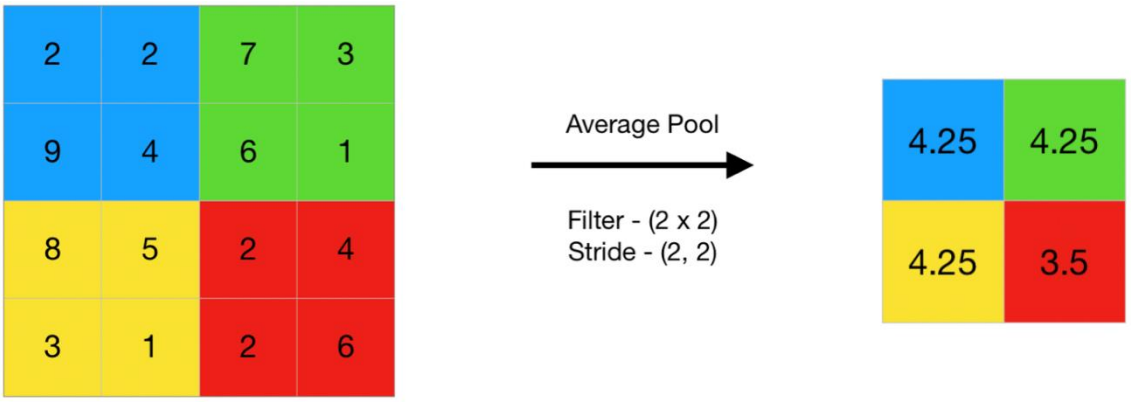
Esencialmente son bastante similares, el principio básico es dividir la imagen en secciones dado un "stride" (generalmente de 2, lo que implica agrupar la imagen de salida en grupos de 2x2), siguiendo con este ejemplo:

Con el max pooling, cogemos el valor más alto de entre esos grupos de 4 pixeles:



7. Ejemplo de max pooling para una matriz de 4x4. Fuente: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/

Por otra parte, el avg pooling realiza la media entre los valores de esos grupos de 4 píxeles:



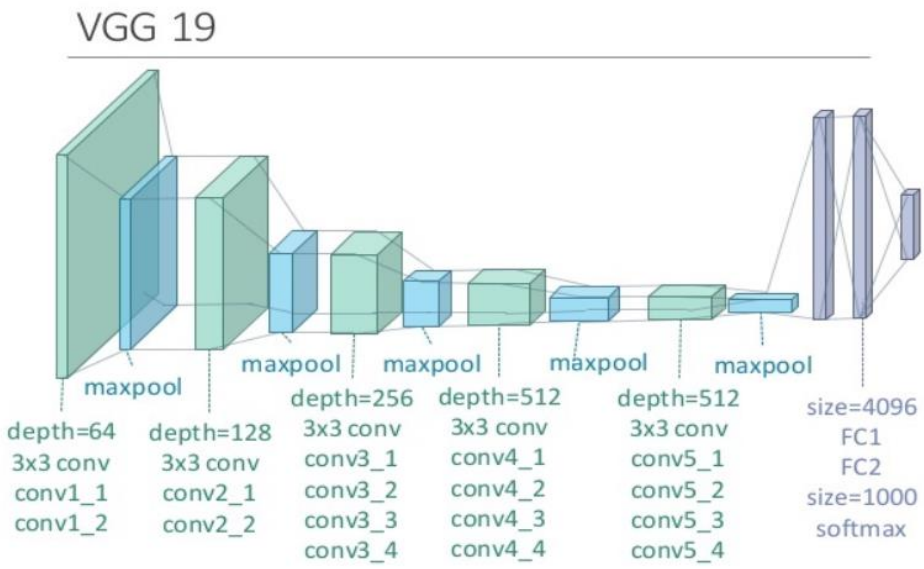
8. Ejemplo de avg pooling para una matriz de 4x4. Fuente: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/

4.2.7. Proceso de ejecución de una CNN

Una vez definidos los diferentes procesos y operaciones por los que pasa una imagen en una red neuronal convolucional, vamos a detallar el proceso de ejecución de esta, para ello, vamos a utilizar la arquitectura vgg19, una red neuronal convolucional pre-entrenada para clasificar imágenes, cuyos pesos han sido entrenados en la base de imágenes imagenet, que contiene más de 14.197.122 imágenes, utilizadas para ayudar a investigadores, estudiantes y demás en el campo de estudio de imagen y visión artificial.

La arquitectura de dicha red está formada por 16 capas convolucionales, tres capas totalmente conectadas, 5 capas de max pooling y una capa softmax para comprimir los valores entre 0 y 1, en concreto, y por orden: Dos capas Conv3x3 (64) + ReLu, MaxPool, dos capas Conv3x3 (128) + ReLU, MaxPool, 4 capas Conv3x3 (256) + ReLU, MaxPool,

4 capas Conv3x3 (512) + ReLU, MaxPool, 4 capas Conv3x3 (512) + ReLU, MaxPool, tres fully connected (4096) y una SoftMax final. [14]



7. Detalle de la arquitectura vgg19. Fuente: https://user-images.githubusercontent.com/46887077/54683836-5e4fd080-4b1b-11e9-8ae0-5d58c6ef63b4.png

Ya sabemos la forma que toma la red que vamos a utilizar, ahora vamos a ver que proceso sigue una imagen de entrada. En primer lugar hay que tener en cuenta el formato de la imagen introducida. Por ejemplo para una imagen de dimensiones 100x100, la representación en escala de grises deja una matriz de 10000 elementos, mientas que si se trata de una imagen a color, esto significa que cada pixel tiene tres elementos, un valor de intensidad para el rojo, otro para el verde y otro para el azul, lo que triplicaría el tamaño de esta matriz (100x100x3).

Una vez tenemos la matriz, comenzamos su recorrido por la primera capa convolucinal, donde se realiza las operaciones de convolución pertinentes por dos filtros diferentes, conv1 y conv 2, cabe destacar también que tras salir de cada filtro se aplica la función de activación ReLU, como se ha indicado al principio en la estructura de un perceptrón. Pasado el segundo filtro, tenemos como salida un primer vector (o mapa) de características interesantes de la imagen, éste se introduce ahora en una capa de pooling para reducir la información poco importante, como se ha definido previamente, dando como resultado una imagen de dimensiones mucho menores en altura y anchura, pero de una profundidad mucho mayor. La profundidad de esta representación se le conoce como ‘canales’, y cada uno de estos representa la probabilidad de ocurrencia de cierto fenómeno extraido en los filtros anteriores,dentro de la imagen (bordes verticales, horizontales, profundidad de elementos, objetos, etc..).

Se va a repetir este proceso en las siguientes capas, pasando por un proceso de pooling tras cada una de ellas que resulta en información condensada de menor tamaño pero de mayor profundidad y tras pasar por todas ellas, implica que la red ha extraido todas las características que podia extraer de la imagen y pasamos a la fase de clasificación.

La manera que tiene de clasificar esta red es mediante dos capas totalmente conectadas (cada neurona de la primera capa está conectada a todas las de la capa siguiente) que conecta el aprendizaje de características con la clasificación de la imagen. La información obtenida en nuestra fase de extracción de características se aplana en un vector y se introduce en esta nueva capa totalmente conectada, una primera capa de 4096 neuronas una segunda del mismo tamaño y una tercera de 1000. Finalmente se pasa el resultado por la función softmax para dejar los valores entre 0 y 1.

Pese a que se ha explicado este ultimo paso de clasificaicón, en este trabajo sólo se va a usar la capacidad de extraer características de la red. También cabe destacar que en una red neuronal convolucional, en la parte de extracción de características en concreto, los pesos que conectan las neuronas consisten en los kerneles convolucionales que se han definido anteriormente, y que de necesitarse, son los que se modifican para mejorar la clasificación final, modificando también de esta manera el algoritmo de backtracking que se ha explicado anteriormente. [15]

4.3 Descripción del artículo

En el artículo se detalla el proceso que hay que seguir para obtener la transferencia de estilos dadas dos imágenes, una que nos aporta el contenido y otra el estilo de la fusión final. [16]

En primer lugar vamos a definir dos imágenes que introduciremos al algoritmo, la primera a la que denominaremos imagen de contenido, la cual va a definir el contenido de nuestra imagen final y una imagen de estilo, la cual va a definir el estilo de nuestra imagen final. Con respecto a la imagen final, vamos a generarla a partir de una imagen de ruido aleatorio.

Entrando en más detalle del cómo llegamos a un resultado óptimo a partir de una imagen aleatoria, vamos a introducir una de nuestras imágenes de entrada por nuestra CNN (por ejemplo la imagen de contenido), con el objetivo de codificar esa imagen como una serie de mapas de características, la cual cada elemento consiste en las respuestas de la imagen al filtro (o filtros) determinado de esa capa de la red. Específicamente, una capa de nuestra red CNN con N_i filtros distintos, devuelve N_i mapas de características, cada uno de tamaño M_i, siendo éste el tamaño (altura x anchura) de nuestro mapa de características. Si queremos obtener los resultados de una capa de la red, tenemos que devolver una matriz con tantas filas como filtros y cuyas columnas sean las respuestas de ese filtro en esa posición.

Concretamente una matriz de características F^l de dimensiones NxM ,$F^l_{i,j}$ donde j es la activación (respuesta del filtro) del filtro i de la capa l.

Para generar nuestra imagen final vamos a generarla a partir de aplicar el descenso por gradiente a partir de dicha imagen aleatoria para encontrar una imagen que coincida con las respuestas de los mapas de características obtenidos al introducir nuestra imagen

de entrada en la red CNN. Con esto tenemos que calcular la pérdida o error en el contenido de la imagen generada con nuestra imagen de contenido de entrada.

Para calcular el error, definimos el error cuadrático entra las dos representaciones de características para una capa l, nuestra imagen de entrada $\vec{p}$ y nuestra imagen aleatoria a generar $\vec{x}$:

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2}\sum_{i,j}\left(F_{i,j}^{l} - P_{i,j}^{l}\right)^{2}$$

Siendo:

- $F_{i,j}^{l}$: la respuesta en la posición j del filtro i en la capa l de la imagen aleatoria.
- $P_{i,j}^{l}$: la respuesta en la posición j del filtro i en la capa l de la imagen de contenido.
- Calculamos el cuadrado de la diferencia entre las respuestas a los filtros de la red.

El resultado de esta ecuación consiste en una nueva matriz de dimensiones NxM, siendo N y M las definidas anteriormente, con las diferencias entre la imagen aleatoria y la imagen de contenido, las cuales queremos disminuir lo máximo posible.

Ahora necesitamos servirnos de la derivada en función de las activaciones de filtros para una capa l para calcular el gradiente con respecto a nuestra imagen $\vec{x}$ definida anteriormente. Definimos la derivada del error en el contenido para una capa l como:

$$\frac{\partial \mathcal{L}_{content}}{\partial F_{i,j}^{l}} = \begin{cases} (F^{l} - P^{l})_{i,j}, & F_{i,j}^{l} > 0 \\ 0, & F_{i,j}^{l} < 0 \end{cases}$$

Siendo:

- $(F^{l} - P^{l})_{i,j}$: la diferencia entre las respuestas en la posición j del filtro i en la capa l de la imagen aleatoriamente generada y de la imagen de contenido.

Con esto podemos cambiar la imagen aleatoria generada inicialmente hacia una imagen que produzca la misma respuesta en una capa dada de nuestra red CNN que nuestra imagen de contenido $\vec{p}$.

En este artículo se muestran las reconstrucciones de la imagen de contenido tras los resultados obtenidos tras las capas de la red VGG original ‘conv1_1’, ‘conv2_1’, ‘conv3_1’, ‘conv4_1’ y ‘conv5_1’.

Paralelamente se van a construir representaciones de estilo que calculen las correlaciones en diferentes respuestas de filtros. Estas correlaciones vienen dadas por la matriz de Gram G de dimensiones N_l x N_l mapas de características, G^{l} para denotar la matriz de Gram para una capa l de nuestra CNN, donde $G_{i,j}^{l}$ es el producto vectorial entre los mapas de características vectorizados i y j para una capa l, lo que definido matemáticamente queda:

$$G_{i,j}^{l} = \sum_{k} F_{i,k}^{l} * F_{j,k}^{l}$$

Siendo:

- $F_{i,k}^{l}$: La matriz de características explicada anteriormente solo que "aplanada" para transformarse en una matriz de dimensiones 1 x (N x M).
- $F_{j,k}^{l}$: Es técnicamente la matriz traspuesta de la mencionada anteriormente, partiendo de la misma matriz de características, y cambiándole la forma a un vector de (N x M) x 1.

Una vez definida la matriz de Gram, vamos a hacer uso de esta para generar texturas que coincidan con el estilo de nuestra imagen de estilo de entrada, para ello una vez más vamos a utilizar el descenso por gradiente de una imagen generada aleatoriamente (la que se convertirá en nuestra imagen final) para llegar a una imagen cuyo estilo coincida con el estilo de nuestra imagen de estilo de entrada. En resumen, utilizamos la matriz de Gram para calcular las correlaciones entre los mapas de características aplanados.

Para lograr esto vamos a buscar minimizar la distancia media al cuadrado entre los elementos de la matriz de Gram de nuestra imagen original y de nuestra imagen generada, definimos $\vec{a}$ como nuestra imagen de estilo de entrada y $\vec{x}$ la imagen que generamos aleatoriamente, calculamos este error para una capa l de nuestra CNN de la siguiente manera:

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{i,j}^{l} - A_{i,j}^{l}\right)^2$$

Siendo:

- N_l y M_l son el alto y el ancho de nuestra matriz de características en la capa l de nuestra CNN
- A^l es la representación de estilo (o matriz de Gram) de nuestra imagen de estilo de entrada
- G^l es la representación de estilo (o matriz de Gram) de nuestra imagen generada aleatoriamente

Con esta operación lo que conseguimos es calcular el error o pérdida que aporta la capa l a la pérdida total. Ahora necesitamos sumar todos los errores parciales de estilo por cada capa de nuestra CNN, pero incluyendo un nuevo peso w_l para poder marcar el cuánto contribuye cada capa en la definición del estilo de nuestra imagen final, de manera que podemos formalizar la pérdida de estilo como:

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

Siendo:

- w_l el peso asignado previamente a la capa l, en este caso es recomendable que la asignación de todos los pesos distintos de cero sume 1
- E_l el error de estilo para la capa l definido anteriormente

Ahora necesitamos calcular los gradientes de E_l con respecto a las activaciones en las capas más bajas de nuestra red CNN mediante "backpropagation", vamos a definir la derivada del error en estilo por capa:

$$\frac{\partial E_l}{\partial F_{i,j}^l} = \begin{cases} ((F^l)^T (G^l - A^l))_{i,j}, & F_{i,j}^l > 0 \\ 0, & F_{i,j}^l < 0 \end{cases}$$

Siendo:

- $(F^l)^T$ la matriz traspuesta de nuestra matriz de características de la imagen generada aleatoriamente en la capa l
- $(G^l - A^l)$ la diferencia entre los elementos de las matrices de Gram en la capa l de la imagen generada aleatoriamente y la matriz de Gram de nuestra imagen de estilo de entrada.

En este artículo se muestran las reconstrucciones de la imagen de estilo tras los resultados obtenidos tras las capas de la red VGG original {'conv1_1'}, {'conv1_1' y 'conv2_1'}, {'conv1_1', 'conv2_1' y 'conv3_1'}, {'conv1_1', 'conv2_1', 'conv3_1' y 'conv4_1'} y {'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1' y 'conv5_1'}.

Una vez definidos los cálculos de la perdida de estilo y de contenido, tenemos todo lo necesario para generar nuestra imagen final, para la que necesitamos minimizar conjuntamente la distancia de la imagen aleatoria generada al principio con la representación del contenido de la imagen de contenido de entrada en una capa de nuestra red CNN ('conv4_2' en nuestro caso) y la distancia de la imagen aleatoria generada con la representación del estilo de la imagen del estilo de entrada en numerosas capas de nuestra red CNN (en nuestro caso 'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1' y 'conv5_1').

Por último nos queda calcular ese error o distancia que queremos minimizar, para lo cual vamos antes a valernos de un par de parámetros α y β, cuyo ratio (α/β) va a determinar el cómo de estilizada va a estar la imagen final, o cómo de definido o fiel al original va a estar el contenido de la imagen generada respecto a la original.

7. Reconstrucciones de la imagen final en función de la capa convolucional y del ratio α/β. Menor ratio implica mayor claridad en el contenido de la imagen. Fuente: https://arxiv.org/pdf/1508.06576v2.pdf

Incluyendo estos parámetros al cálculo final, lo definimos de la siguiente manera:

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$

Siendo:

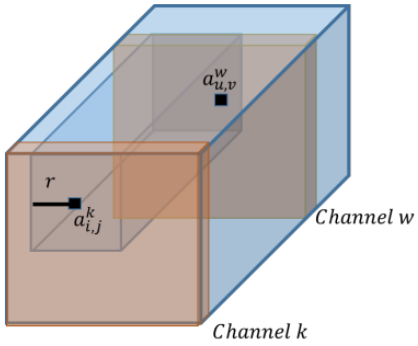
- α y β los pesos asignados al contenido y al estilo respectivamente, conforme aumenta el ratio, más estilizada está la imagen, en este caso se utiliza un $\alpha = 1$ y se utiliza β para conseguir el ratio deseado, con un $\beta = 10^2$ o $\beta = 10^3$ se le da más importancia al contenido mientras que usar valores superiores de β (10^4 o 10^5) difuminan en gran medida el contenido
- $\mathcal{L}_{content}(\vec{p}, \vec{x})$ es el cálculo de la pérdida del contenido de nuestra imagen de contenido y nuestra imagen generada (nótese que ignoramos el parámetro de la capa l ya que hemos definido anteriormente que se va a usar la capa conv4_2)
- $\mathcal{L}_{style}(\vec{a}, \vec{x})$ es el cálculo de la pérdida del estilo de nuestra imagen de estilo y nuestra imagen generada (nótese que ignoramos el parámetro de la capa l ya

que hemos definido anteriormente que se van a usar la capas conv1_1, conv2_1, conv3_1, conv4_1 y conv5_1)

4.4. Coocurrencias. El filtro de coocurrencias y el tensor de coocurrencias.

Otra de las cosas que se van a utilizar en este trabajo son las coocurrencias. Éstas se definen como la coincidencia entre un par de elementos. Las coocurrencias aplicadas a las redes neuronales convolucionales pueden dar lugar a la obtención de tensores de coocurrencias que se van a explicar a continuación.

Supón que extraemos nuestra matriz tridimensional de características en cualquier punto de la ejecución de una imagen la que sea en nuestra red neuronal convolucional. Ahora bien, definimos una coocurrencia en este contexto, cuando el valor de dos activaciones en diferentes canales, dentro de una región dada, son ambos mayores que un umbral predefinido t, como se muestra a continuación [17]:



Tenemos una coocurrencia cuando dos activaciones, en este caso para el canal k, centrado en el elemento $a^{k}_{i,j}$ y dado un radio r a su alrededor, encuentra en cualquier otro canal, en este caso el canal w, un valor $a^{w}_{u,v}$ que está por encima, junto a $a^{k}_{i,j}$, de un umbral dado t.

Fuente: https://arxiv.org/pdf/2003.13827.pdf

Ahora bien, sabiendo el concepto de coocurrencia, pasamos a definir los filtros de coocurrencias, lo que nos va a llevar a crear un tensor (o matriz) de 4 dimensiones, lo que significa que es un conjunto de matrices de tres dimensiones. Esto es necesario porque para obtener nuestro tensor de coocurrencias necesitamos calcular la coocurrencia de un canal i con todos los demás, pero nunca consigo mismo, lo que nos lleva a crear (suponiendo un número n de canales) n matrices (o tensores) tridimensionales del mismo tamaño que el tensor original, y con todos los elementos puestos a 1, menos los elementos

correspondientes al canal i en el que se pretende calcular la coocurrencia con el resto, dejándolos a 0.

Una vez tenemos el conjunto de tensores, al que llamaremos filtro de coocurrencias, vamos a calcular el tensor de coocurrencias.

Para ello, dado un mapa de características convolucional, una distancia r y un umbral t, definimos una coocurrencia positiva entre dos activaciones $a^{k}_{i,j}$ y $a^{w}_{u,v}$ como:

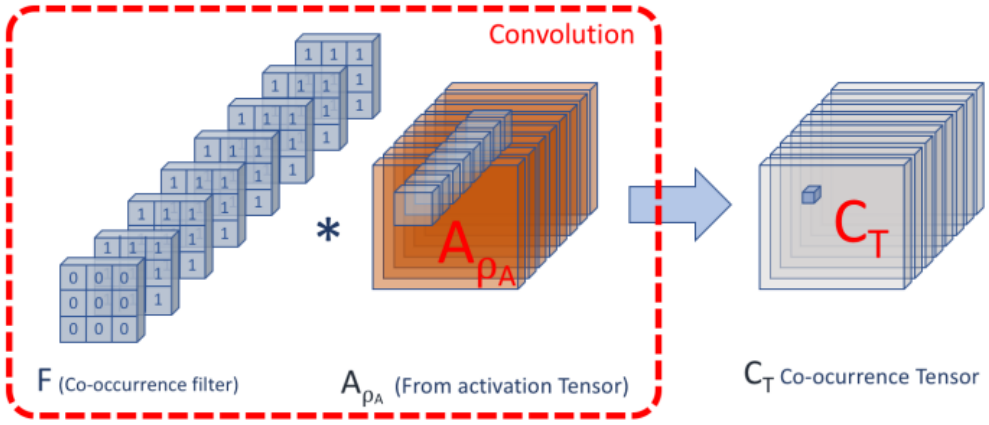
$$\rho\left(a^{k}_{i,j}, a^{w}_{u,v}\right) = \begin{cases} 1, si\ |i-u|\ \leq r\ y\ |j-v|\ \leq r\ \ y\ a^{k}_{i,j}\ < t\ \ y\ a^{w}_{u,v}\ < t \\ 0\ en\ otro\ caso \end{cases}$$

Con esto podemos saber los valores que nos dan una coocurrencia entre canales y señalarlos, y poner a cero los que no, con esto describimos el siguiente proceso para un tensor A:

- Sacamos el filtro (o conjunto de matrices) de unos en función de sus dimensiones.
- Sacamos una máscara dado un umbral t para el tensor A ($\rho_A = A > t$).
- Con este tensor booleano que indica los valores que cumplen el primer requisito indispensable para la coocurrencia, sacamos los valores que lo cumplen ($A_{\rho A} = A * \rho_A$).
- Con este nuevo tensor de elementos, el radio de coocurrencia y el número de canales D, realizamos la convolución para agregar todos los valores de los diferentes canales que cumplen la condición de coocurrencia, y pasa a ser el elemento del tensor de coocurrencias para ese valor ($C_T = conv2d(A_{\rho A}, filtros, padding = r)/(D-1)$ Teniendo en cuenta una

dimensión menos ya que no contamos el canal desde el que se está calculando la coocurrencia con el resto

- Por último se multiplica al tensor obtenido tras la convolución por la máscara ρ_A para quitar los valores que originalmente no cumplían la condición del umbral y han sido reintroducidos por cómo funciona la convolución



1 Ejemplo del filtro de coocurrencias F (para el primer canal) convolucionado con el tensor filtrado por umbral para obtener un elemento del tensor de coocurrencias. Fuente: https://arxiv.org/pdf/2003.13827.pdf

5. Propuesta de cambio. Coocurrencias y Style Transfer.

Llegados a este punto hemos definido ya tanto el artículo que se va a implementar y la teoría de los modelos implicados en la implementación. El objetivo de este trabajo es mejorar el resultado del algoritmo de style transfer cambiando la manera de generar el error de estilo.

Actualmente, nos servimos de la matriz de Gram para calcular el error de la representación de estilo de la imagen, que se basa en calcular la correlación de cada canal con el resto de canales. Una vez saquemos resultados con este método, vamos a realizar una segunda implementación que sustituya el método de la matriz de Gram con el tensor de coocurrencias definido anteriormente y sacar más resultados y compararlos entre sí.

La implementación de estos dos modelos diferentes se explicará más adelante, pero la idea principal es que debido a que el tensor de coocurrencias es bueno detectando similitudes en las características de la imagen entre canales para regiones concretas, puede llegar a ofrecer buenos resultados frente a la matriz de Gram que tiene una base teórica similar.

6. Herramientas e implementación.

6.1. Introducción y desarrollo de la implementación:

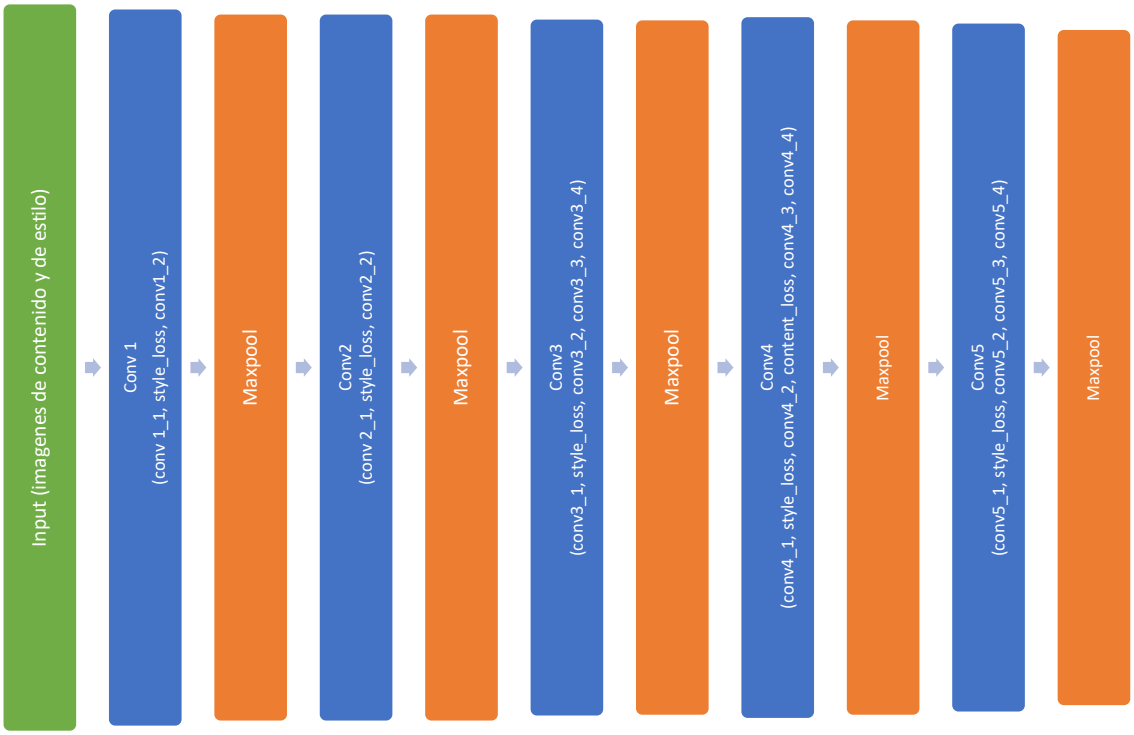
Con el objetivo de realizar una transferencia de estilos, se va a entrenar una red para la transferencia de una imagen utilizando el método original basado en la matriz de Gram y otra modificando (o cambiando más bien), la matriz de Gram por un tensor de coocurrencias, para intentar comprobar si la segunda implementación mejora el resultado con respecto a la implementación original.

La implementación de este trabajo se ha realizado en un cuaderno jupyter y utilizando el lenguaje de programación Python 3, mientras que las imágenes de contenido y de estilo que se van a utilizar has sido descargadas desde el buscador de Google imágenes, algunas de ellas recomendadas en el artículo original y otras buscadas por mi cuenta.

Se ha construido un modelo de red neuronal en el que partimos de la base de la vgg19 mencionada anteriormente, de la cual importamos los pesos, y sobre ella se han realizado diversas modificaciones para adecuarlas a la obtención de transferencias de estilos.

En el artículo se mencionan una serie de capas de la vgg19 de las que se obtienen diversos cálculos, en concreto para las 5 capas convolucionales se calcula un error de estilo a la salida de cada una de ellas (tras pasar la función de activación ReLU), mientras que tras la salida de la cuarta capa convolucional (conv4_2) únicamente se sacará el error de la representación de contenido. Por otra parte vamos a ignorar la parte de clasificación puesto que no va a ser necesario clasificar nuestro resultado final.

Estos cálculos de errores se han implementado mediante módulos que se intercalarán en la arquitectura de la red vgg, como se muestra a continuación:



Arquitectura del modelo utilizado para realizar el style transfer. Nótese que tras cada mención de filtro 'conv', se realiza la activación ReLU del mapa de características pese a que no se muestra por motivos visuales.

El primer problema que surge es el tamaño de las imágenes. A priori no debería de importar el tamaño puesto que ni el error cuadrático medio ni la matriz de Gram (los dos cálculos matemáticos que se realizan en el cálculo del style_loss, y únicamente el error cuadrático medio en el cálculo del content_loss). El conflicto surge en cuanto se va a cambiar la manera de computar el error de estilo introduciendo el tensor de coocurrencias, por lo que fijamos un tamaño de imagen de 512x380, que son de tamaño lo suficientemente reducido para aligerar la carga computacional que lleva ejecutar el modelo en una maquina local y lo suficientemente grande como para apreciar los resultados obtenidos. Por otra parte, el resto de los parámetros (pesos de los errores de contenido e imagen, pesos de los filtros, etc.) se han usado los recomendados en el artículo original.

Cabe destacar que en los módulos 'style_loss' que se introducen en el modelo, se da la posibilidad de cambiar entre los modos de cálculo de error de estilo mediante la matriz de Gram o el cálculo del tensor de coocurrencias. Además, se puede dar el tensor de coocurrencias de diferentes maneras:

- Se puede utilizar como la matriz 'en bruto' (tensor tridimensional, cada canal representa la coocurrencia de ese canal con el resto de canales.
- Puede utilizarse como una matriz 'bidimensional' de altura*anchura, en la que cada elemento es la media de todas las coocurrencias por canal de dicho elemento
- Por último puede representarse como un vector de canales en el que cada elemento representa la media de todas las coocurrencias de dicho canal

Se tratará de obtener resultados de estas maneras y compararlos entre sí para buscar el mejor resultado posible.

Se introducen dos imágenes, una de contenido y otra de estilo, se fija un tamaño de imagen y se decide si la imagen objetivo se va a obtener partiendo de una copia de la imagen de contenido, o de una imagen de ruido generada aleatoriamente. Se preprocesan las imágenes de entrada y se pasan por la red para obtener los mapas de características. Una vez obtenidos comenzamos a iterar sobre la imagen objetivo, fijamos las iteraciones que den un buen resultado, en este caso se obtiene un buen resultado utilizando 1000.

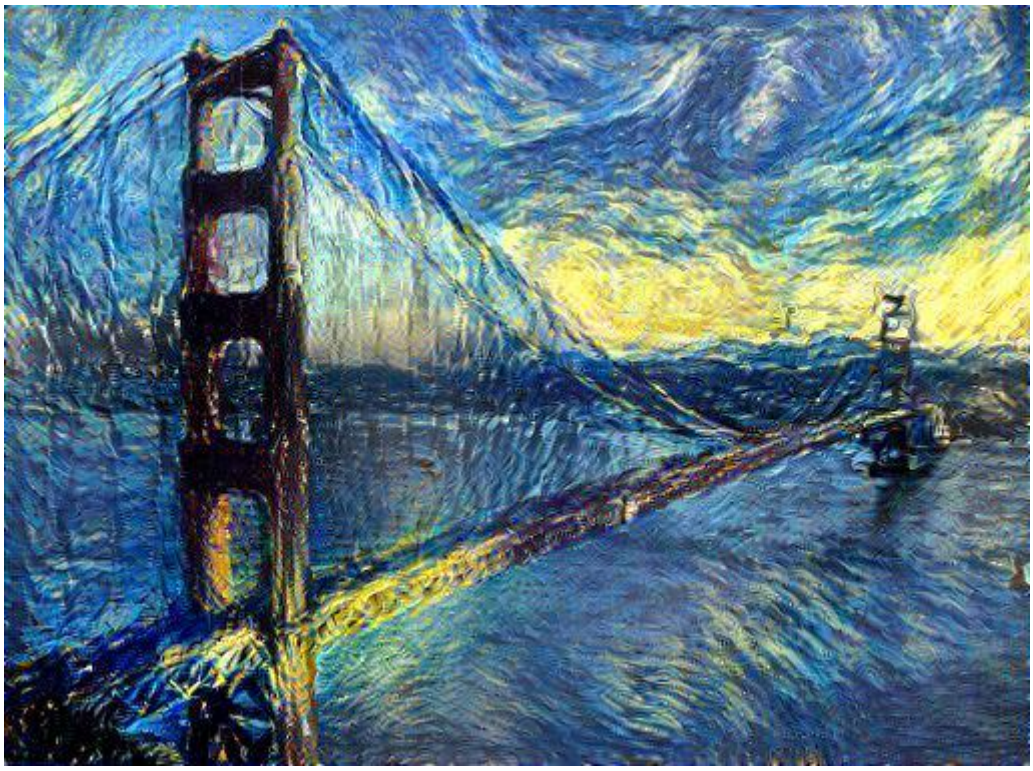
7. Obtención de resultados

Vamos a realizar una ejecución del algoritmo, utilizando las dos fotos que se muestran a continuación

Rep. de contenido. Golden Gate de San Francisco. Fuente: Google imágenes

Rep. de estilo. Noche estrellada de Vincent Van Gogh. Fuente: Google imágenes

Primero vamos a ejecutar el algoritmo utilizando la matriz de Gram, como se ha explicado anteriormente, utilizando un ratio de pesos de contenido y estilo de 10^{-2}, 1000 iteraciones y un tamaño de 512x380 para obtener lo siguiente:



Resultado final del style transfer con matriz de Gram

Vamos ahora a ejecutar el algoritmo con los mismos parámetros pero utilizando la técnica del tensor de coocurrencias, transformándolo, por ejemplo, en su forma de vector (resultante de hacer la media de elementos por cada canal):

Resultado final del style transfer con coocurrencias

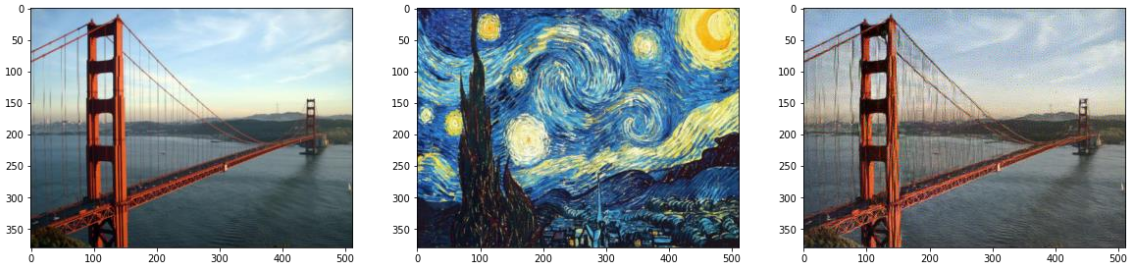
Claramente se puede observar que hay algo que no va bien, puesto que a priori solo se muestra la representación de contenido de la imagen (obtenida a partir de una imagen de ruido aleatorio). Esto se puede confirmar debido a que mostrando el tensor resultante del módulo del cálculo de estilo, los valores andan por debajo del orden de 10^{-9}, lo que debe de hacerlos insignificantes a la hora de calcular el resultado final. Cabe destacar también que pese a que se ha mostrado una ejecución obteniendo el vector de coocurrencias, el resultado utilizando el tensor de coocurrencias tal cual y utilizando la matriz (media de canales por elemento), obtiene un resultado, a primera vista, similar.

Tras esta observación, el primer instinto es buscar una manera de hacer los valores, sin desmerecer el método de obtención de los mismos, más grandes.

En primer lugar vamos a intentar aumentar el peso del estilo, por ejemplo aumentarlo a 10^{10}, para obtener el mismo resultado.

Resultado final del style transfer usando un peso de $1e^{15}$, no se obtienen resultados válidos como se puede observar por la tercera imagen

Vamos a intentar otros enfoques para intentar un buen resultado con esta técnica. Debido a que como se ha observado anteriormente los valores de los tensores de módulos son muy bajos, vamos a normalizar el tensor, de manera que los valores del tensor sumen 1. Esto último da un resultado similar al primer intento que hemos realizado al aplicar coocurrencias.



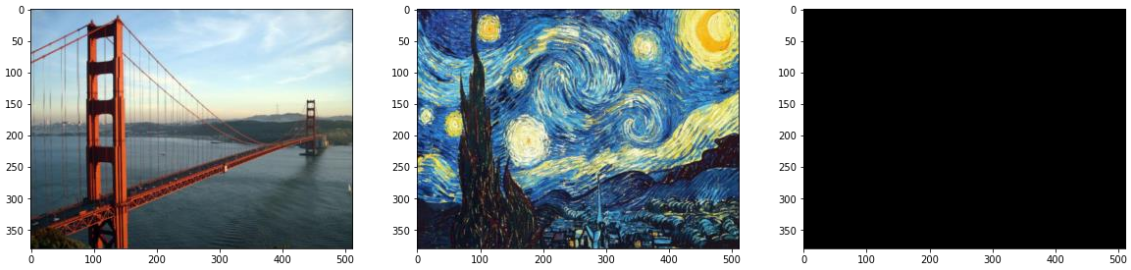
Intento de ejecución de style transfer con matriz normalizada

Por último, debido a que modificar los parámetros per se del style transfer, vamos a intentar ir más allá y calcular la matriz de Gram entre los tensores de coocurrencias de dos maneras. La primera es calcularla sobre la matriz de medias de canales por elemento, (la forma de representar el tensor de coocurrencias como una matriz que se ha explicado anteriormente) y la segunda transformando esa matriz en un vector de dimensiones 1x(canales x altura x anchura) y calculando la matriz de Gram con la misma.

Se muestran los resultados a continuación:

Ejecución de style transfer utilizando el método de obtención de la matriz de gran pero usando el tensor de coocurrencias representada como la matriz de medias de canales por elemento

Es evidente que el resultado sigue sin ser el adecuado, el propio tensor no debe de aportar la información necesaria de la que se sirve la matriz de gran para calcular la representación de estilo.



Representación de estilo utilizando el tensor de coocurrencias representado como un vector y utilizado para el cálculo de la matriz de Gram

Aquí podemos observar otro problema, y es que, llegado cierto punto en la ejecución del algoritmo, se llegan a ciertos valores que el programa ya considera infinitos, debido a la multiplicación constante de los valores del vector de características utilizado para el cálculo, como se puede observar en la imagen de a continuación:

```
Output exceeds the size limit. Open the full output data in a text editor
tensor([[122.1151]], device='cuda:0', grad_fn=<DivBackward0>)
tensor([[2068.9717]], device='cuda:0', grad_fn=<DivBackward0>)
tensor([[1306.1200]], device='cuda:0', grad_fn=<DivBackward0>)
tensor([[25961.1797]], device='cuda:0', grad_fn=<DivBackward0>)
tensor([[35.4072]], device='cuda:0', grad_fn=<DivBackward0>)
tensor([[122.1078]], device='cuda:0', grad_fn=<DivBackward0>)
tensor([[2068.9924]], device='cuda:0', grad_fn=<DivBackward0>)
tensor([[1306.0518]], device='cuda:0', grad_fn=<DivBackward0>)
tensor([[25959.4570]], device='cuda:0', grad_fn=<DivBackward0>)
tensor([[35.4082]], device='cuda:0', grad_fn=<DivBackward0>)
tensor([[1.4830e+24]], device='cuda:0', grad_fn=<DivBackward0>)
tensor([[6.3385e+25]], device='cuda:0', grad_fn=<DivBackward0>)
tensor([[1.0472e+26]], device='cuda:0', grad_fn=<DivBackward0>)
tensor([[7.6062e+26]], device='cuda:0', grad_fn=<DivBackward0>)
tensor([[4.5184e+23]], device='cuda:0', grad_fn=<DivBackward0>)
tensor([[nan]], device='cuda:0', grad_fn=<DivBackward0>)
tensor([[nan]], device='cuda:0', grad_fn=<DivBackward0>)
tensor([[nan]], device='cuda:0', grad_fn=<DivBackward0>)
tensor([[nan]], device='cuda:0', grad_fn=<DivBackward0>)
tensor([[nan]], device='cuda:0', grad_fn=<DivBackward0>)
tensor([[nan]], device='cuda:0', grad_fn=<DivBackward0>)
```

Visualización del vector de salida del módulo de cálculo del estilo, se observa que llega un punto en el que la multiplicación de valores simplemente es muy grande

Este es un problema que podemos solucionar introduciendo unos pesos de contenido y de estilo negativos, de manera que el ratio entre ellos se mantenga pero que hagan los valores de esta representación de estilos más manejables. Por ejemplo vamos a utilizar un peso de contenido de $5e^{-9}$ y un peso de estilo de $1e^{-7}$.

Se muestra el resultado de esta modificación a continuación:

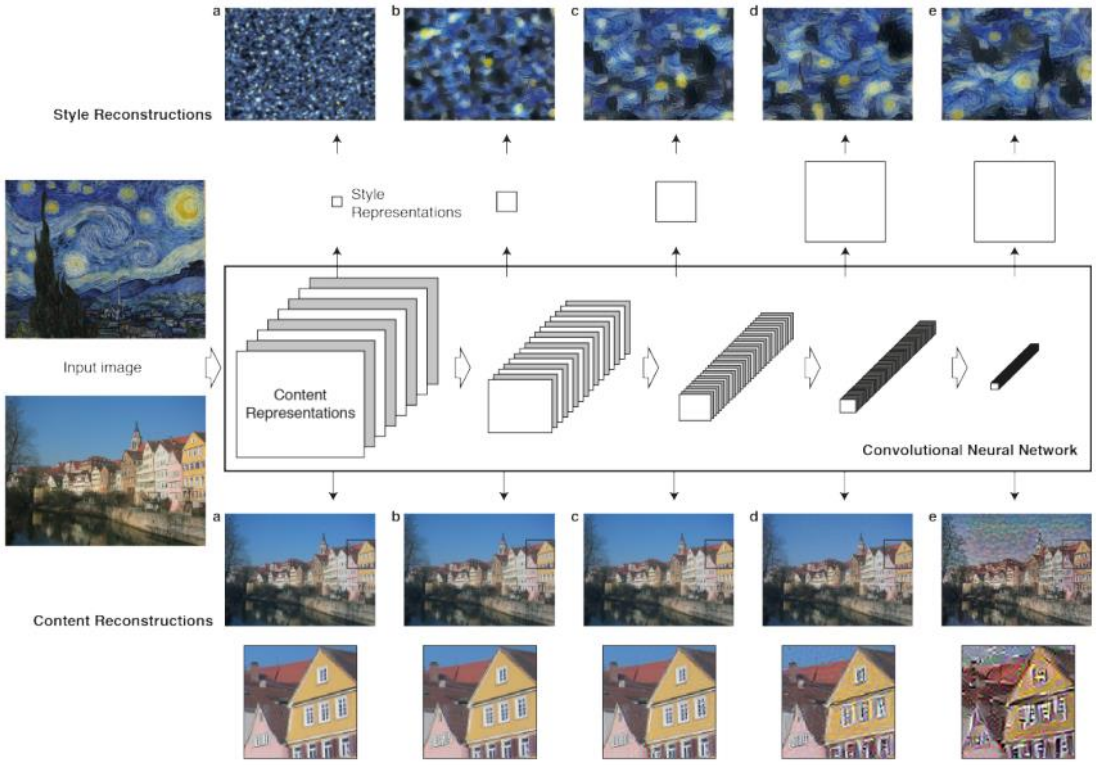
Representación de estilo utilizando el tensor de coocurrencias representado como un vector y utilizado para el cálculo de la matriz de Gram, pero con los pesos reducidos para evitar el problema de los valores muy grandes

Obtenidos estos resultados y sin mucho más que probar, paso a centrarme en explicar por qué este acercamiento al problema no funciona. Para ello vamos a profundizar un poco en las características de la red vgg19 y porque la matriz de Gram es utilizada y funciona bien en la técnica de style transfer, y en qué se diferencia con el tensor de coocurrencias propuesto anteriormente como para no dar buenos resultados en esta técnica.

7.1 La extracción de características en la red VGG19 y la matriz de Gram

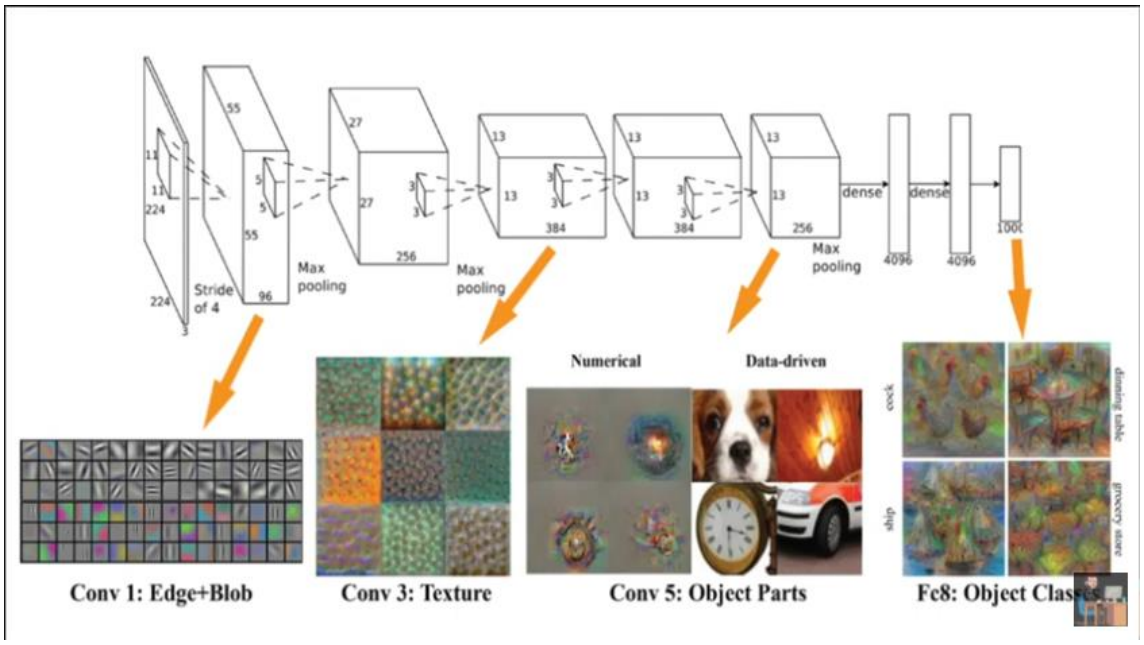
Se ha mencionado antes la estructura de la red VGG19, pero va a ser interesante saber un poco más en detalle en que se 'especializa' en detectar nuestro modelo en cada fase para saber luego como trata esos datos la matriz de Gram.

Vamos para ello a sacar una imagen del artículo explicado anteriormente en el que se pueden apreciar varias cosas [16]. En primer lugar, en lo que a contenido se respecta, se puede observar en la imagen, que a la hora de reconstruir la imagen de contenido introducida, las capas más bajas de la red reconstruyen la imagen prácticamente perfecta, a partir de la respuesta de únicamente esa capa. Este fenómeno se puede observar en las tres primeras capas, mientras que en las capas finales, la representación de contenido se muestra más influenciada por los contenidos de más alto nivel que detecta la red, como se va a mencionar más tarde, a coste de perder detalle en pixeles concretos.



Reconstrucciones de estilo y contenido dadas dos imágenes de entrada durante la ejecución de la red VGG19. Fuente: https://arxiv.org/pdf/1508.06576v2.pdf

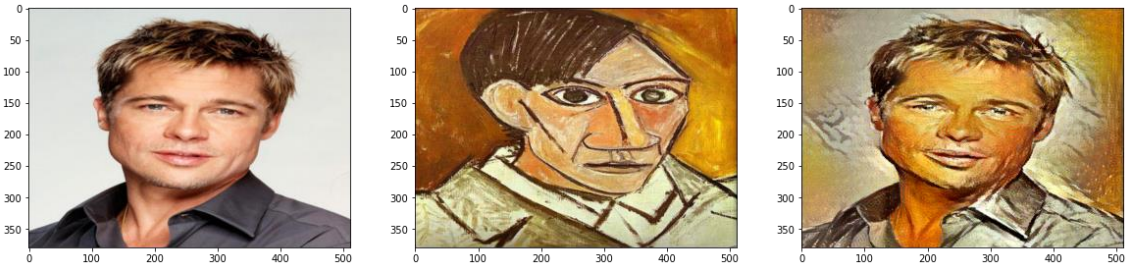
En lo que a la representación de estilo se refiere, que es aquí donde entra en juego nuestra matriz de Gram, la cual se ha definido su comportamiento anteriormente. Ayudándonos de esta, calculamos las correlaciones entre las diferentes características captadas por el modelo modificado de la VGG19. Las características que puede llegar a extraer la red van aumentando en complejidad conforme se avanza de capa convolucional, comenzando por conceptos más generales como bordes, curvas, líneas, puntos, etc. y acabando en la identificación de texturas o partes de objetos. [18]



Características que extraen algunas capas de la red vgg19. Fuente: https://www.youtube.com/watch?v=Elxnzxk-AUk

Al obtener los mapas de características agrupados en un tensor, lo que hacemos es convertir cada canal, es decir cada mapa de características de una característica en concreto, en un vector, y apilarlos uno encima del otro, lo que nos deja con una matriz con tantas filas como canales y tantas columnas como el alto por el ancho de los mapas de características a transformar.

Con esta matriz y su producto por su traspuesta lo que conseguimos es calcular en qué manera cada par de características 'coinciden' para cada punto del mapa de características, con la distinción de que para calcular el estilo lo que más necesitaremos es medir esas correlaciones entre texturas, objetos, trazos, distribuciones de colores, etc. Elementos cuyas ocurrencias hacen depender entre ellos. Vamos a generar un ejemplo para explicar más en detalle esto último.



Style transfer resultante del primer plano del actor Leonardo DiCaprio y el Autorretrato 1907 de Pablo Picasso, utilizando el método original de la matriz de Gram

Debido a la naturaleza del producto matricial, nuestras filas de la matriz que vamos a multiplicar por la traspuesta de sí misma, cada una de las filas representa la

ocurrencia de una característica en la imagen, y al multiplicarse por la traspuesta se calcula la correlación de esa característica con cada una de las otras características.

En el caso de la imagen anterior vamos a fijarnos por ejemplo en la camisa de la representación de contenido. Nuestro modelo extraerá en capas más altas las características de la textura blanquecina de la camisa, y detectará parte del objeto también por las líneas anguladas que separan la prenda de ropa del resto, ahora pues, al tener una alta correlación, (la aparición de la textura suele llevar la aparición del objeto 'cuello de camisa' y viceversa) implica generar esa textura con sus trazos de borde característicos en la detección de ese objeto.

Además, si nos fijamos en el pelo, la representación de estilo da una alta correlación entre esa textura y el trazo del pelo, y al detectar en la imagen de contenido un tono de pelo parecido al color del estilo, se relaciona con ese color detectado en el estilo, y su correlación con ese trazo, pues representas el estilo del pelo, que en este caso se ve menos uniforme por el cambio de tonalidad en la imagen de contenido.

Volviendo a cuestiones más teóricas, en la publicación 'demistifying style transfer' [19], se describe la diferencia entre matrices de Gram como un método equivalente a minimizar una forma de MMD (Maximum Mean Discrepancy), que es una métrica para medir y minimizar la diferencia entre un 'source' dado y un objetivo. Esto es el componente clave del área conocida como 'Domain Adaptation' cuyo objetivo es transferir el modelo que se aprende en un dominio de entrada a un dominio objetivo.

Siendo MMD un test estadístico para medir la diferencia entre dos distribuciones, al poder desarrollarse matemáticamente la formula del cálculo de la pérdida de estilo como un problema de MMD con kernel gaussiano, como se muestra en el artículo [19], se puede decir que intrínsicamente el style transfer es un proceso de alinear distribuciones de activaciones de la red entre imágenes.

7.2. El rol del tensor de coocurrencias

Sabiendo por qué funciona la matriz de Gram en la transferencia de estilos, y si recordamos la implementación teórica del mismo, tenemos un problema ya que ha quedado claro que la red 'aprende' estilos mediante correlaciones de características, y esto es algo que el tensor de coocurrencias no puede ofrecernos. Este tensor representa las coocurrencias de cada elemento con el resto de canales para ese mismo elemento (y sus elementos cercanos), lo que no nos da esa definición de estilo global como la correlación entre diferentes elementos de la red de alto nivel. Citando el artículo en el que se define este tensor, el objetivo de la representación mediante coocurrencias es caracterizar la dependencia espacial de características de imagen [17], lo que contradice la idea de estilo global que ofrece el acercamiento de la matriz de Gram.

8. Conclusiones y trabajos futuros

En vista de los resultados presentados, está claro que no son satisfactorios ya que no se ha conseguido una salida válida con el método de coocurrencias propuesto, aun así, se pueden extraer algunas conclusiones claras del trabajo.

En primer lugar, la suposición de que un tensor de coocurrencias iba a dar resultados válidos en la transferencia de estilos es claramente errónea. Pese a pensar que podía dar resultados debido a que, sin estudiar profundamente los conceptos de matriz de Gram y de tensor de coocurrencias, se relacionaba el concepto de la detección de estilo de la matriz de Gram con la detección de texturas que puede ofrecer el tensor, sin tener en cuenta el factor de la definición más general de estilo que abarca la matriz de Gram, frente a la limitación más regional que tiene el tensor de coocurrencias.

En segundo lugar, ha quedado claro las maneras en las que se puede modificar el algoritmo del style transfer, al definirse más detalladamente la manera en la que se calcula el error de estilos, mediante la diferencia de las matrices de Gram, la que se extrae en un primer lugar al haber pasado la imagen de estilo por la red y obtenido su mapa de características y la segunda de la imagen objetivo, cuya distancia entre ellas se busca minimizar. Al definirse el cálculo de error de estilo como un problema de MMD con kernel gaussiano, el cambio de ese kernel por otros compatibles puede llevar a otras maneras de generar representaciones de style transfer, que puede ser interesante implementar y comparar en trabajos futuros.

Con respecto a los resultados obtenidos mediante el uso de la matriz de Gram, es muy difícil dar con una combinación de parámetros dentro del algoritmo que nos deje el mejor resultado, ya que es algo puramente subjetivo. Por ejemplo, se ha mostrado anteriormente cómo simplemente el cambio del ratio entre el error de estilo y el error de contenido da representaciones desde más fieles al contenido hasta más ‘artísticas’, se muestra a continuación un ejemplo con tres ratios de contenido y estilo diferentes generados por el algoritmo:

Style transfer de un pulpo y el cuadro 'El naufragio del minotauro', de Joseph Mallord Wiliam Turner, con un ratio de contenido/estilo de $1e^{-2}$ (2.29 minutos de ejecución)

Style transfer de comparación con un ratio de contenido/estilo de $1e^{-3}$ (2.34 minutos de ejecución)

Style transfer de comparación con un ratio de contenido/estilo de $1e^{-4}$ (2.28 minutos de ejecución)

Como se puede observar con estos tres resultados, no hay uno necesariamente mejor o peor, son simplemente diferentes. Se ha incluido además en los pies de foto respectivamente, los tiempos de ejecución de cada uno, que muestra también que no hay ventajas de tiempo de ejecución conforme se aumenta o disminuye el ratio de contenido y estilo.

Lo mismo ocurre con el tipo de pooling, se cuentan alrededor de 2.30 minutos para calcular una transferencia de estilos tanto utilizando un pooling calculado por el elemento máximo u otro calculado mediante la media (con una resolución de imagen de 512 x 380). Puede llegar a ser posible que surjan diferencias de tiempo de ejecución a la hora de operar para generar imágenes de resoluciones mayores que esta (la cual es bastante baja por limitaciones de hardware), hasta llegar a suponer un factor a tener en cuenta, suposición que se menciona ya que se remarca en el artículo original que el 'avg pooling' ayudaba al flujo del gradiente frente al 'max pooling' [16], pese a que aquí no se ha notado mucha diferencia. A continuación se muestran un par de ejemplos siguiendo con los inputs anteriores, con un mismo ratio de contenido/estilo de $1e^{-3}$:

Style transfer utilizando max pooling (mismo ejemplo que la imagen anterior, todas ellas han sido obtenidas utilizando max pooling por preferencia personal)

Style transfer utilizando avg pooling

Una vez más, a la vista de lo obtenido queda a la opinión del lector cuál de los dos resultados es el mejor.

Todo esto no quita el hecho de que el objetivo principal del trabajo era obtener una estrategia de style transfer viable utilizando los tensores de coocurrencias, y fijando una serie de parámetros, comparar los resultados intentando cuantificar, de alguna manera (por ejemplo mediante encuestas pidiendo un ranking de los resultados para un mismo set de imágenes obtenidas con métodos diferentes), cuál de los dos métodos era el mejor, y este objetivo evidentemente no se ha cumplido.

Aun así la experiencia en este trabajo ha sido satisfactoria, se ha profundizado tanto en las redes neuronales convolucionales como en esta técnica de style transfer, que aunque a priori puede parecer meramente anecdótica, puede llegar a tener aplicaciones muy interesantes, refinada la técnica lo máximo posible. Por otra parte con respecto a los tensores de coocurrencias, elegidos para sustituir a la matriz de Gram por su capacidad de detectar texturas, son muy potentes para tareas, como por ejemplo para la detección de imágenes (Image Retrieval) por sus características, y que al ser capaces de detectar esas coocurrencias de características en imágenes, tiene que poder incorporarse en un algoritmo de este estilo que no requiera de esa matriz de correlaciones más global que busca la matriz de Gram, que quizá pueda buscarse en trabajos futuros.

9. Bibliografía

[1] Is artificial intelligence set to become art's next medium? https://www.christies.com/features/A-collaboration-between-two-artists-one-human-one-a-machine-9332-1.aspx

[2] Using AI for new visual storytelling techniques in VR https://engineering.fb.com/2017/07/26/virtual-reality/using-ai-for-new-visual-storytelling-techniques-in-vr/

[3] Perceptrón https://es.wikipedia.org/wiki/Perceptr%C3%B3n

[4] Funciones de activación https://en.wikipedia.org/wiki/Activation_function

[5] Introducción a la Red Neuronal https://www.norwegiancreations.com/2019/04/introduction-to-neural-network/

[6] Aprendizaje no supervisado https://es.wikipedia.org/wiki/Aprendizaje_no_supervisado

[7] Redes Neuronales: Conceptos Básicos y Aplicaciones. Carlos Alberto Ruiz, Marta Susana Basualdo y Damián Jorge Matich https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monograias/matich-redesneuronales.pdf

[8] The math behind backpropagation. Sefik Ilkin Serengil https://sefiks.com/2017/01/21/the-math-behind-backpropagation/

[9] Convolutional Neural Networks (CNN) (opengenus.org) https://iq.opengenus.org/convolutional-neural-networks/

[10] What is "stride" in Convolutional Neural Network? https://medium.com/machine-learning-algorithms/what-is-stride-in-convolutional-neural-network-e3b4ae9baedb

[11] An Implementation of Sobel Edge Detection. Sean Sodha https://www.projectrhea.org/rhea/index.php/An_Implementation_of_Sobel_Edge_Detection

[12] Convolutional Neural Networks - Part 1 https://www.ismailmebsout.com/Convolutional%20Neural%20Network%20-%20Part%201/

[13] CNN | Introduction to Pooling Layer https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/

[14] Understanding the VGG19 Architecture https://iq.opengenus.org/vgg19-architecture/

[15] Derivation of Backpropagation in Convolutional Neural Network (CNN). Zhifei Zhang 2016.10 - Derivation of Backpropagation in Convolutional Neural Network (CNN).pdf (zzutk.github.io)

[16] A Neural Algorithm of Artistic Style. Leon A. Gatys, Alexander S. Ecker, Matthias Bethge https://arxiv.org/pdf/1508.06576v2.pdf

[17] Co-Occurrence of Deep Convolutional Features for Image Search https://arxiv.org/pdf/2003.13827.pdf

[18] The gram matrix in neural style transfer. https://www.youtube.com/watch?v=Elxnzxk-AUk

[19] Demystifying Neural Style Transfer. Yanghao Li. Naiyan Wang. Jiaying Liu. Xiaodi Hou. https://arxiv.org/pdf/1701.01036.pdf