Automatización del proceso de obtención de imágenes híbridas

Daniel García Pérez
danigarcia14@correo.ugr.es

David Muñoz Sánchez
dmunozs14@correo.ugr.es

Carlos Lao Vizcaíno
carloslao@correo.ugr.es

Hugo Gálvez Ureña
hugogalvez@correo.ugr.es

Abstract

Tras la realización de la Práctica 1 de la asignatura, se quiso ahondar más en la obtención de imágenes híbridas. Tras la detenida lectura del artículo [3] , parecía adecuado probar la obtención de imágenes híbridas con Redes Neuronales Convolucionales, para así dejar de lado los análisis hechos con Fourier y no tener en cuenta los conceptos asociados a este, que quedan más lejos del conocimiento del público general.

El problema se puede abstraer en determinar a qué imagen se le aplicará un filtro de paso alto, y qué parejas de sigmas definirán los dos filtros. Para abordar este nuevo planteamiento, la manera de proceder ha sido ir mejorando un modelo simple inicial, además de un modelo basado en redes siamesas.

1. Introducción

Tal y como se ha comentado en el apartado anterior, la principal motivación para este trabajo es dejar de lado el Análisis de Fourier para el proceso de obtención de imágenes híbridas, y también mejorar la forma que se aplicó en la Práctica 1 de ensayo y error.

Uno de los primeros problemas existentes, fue la no existencia de ningún DataSet debidamente etiquetado con el que poder entrenar nuestro modelo, por tanto, se tuvo que hacer un conjunto de entrenamiento completamente de cero y etiquetarlo manualmente según el criterio visual de uno de los integrantes para que la red aprendiera a hibridar de forma parecida a una persona, y no mezclar criterios de todos los integrantes. De cara a una posible implementación del modelo más avanzada y con los suficientes medios, como un DataSet más extenso, sí que sería conveniente una mayor diversidad en el criterio de hibridación, pero entendemos que en esta primera resolución del problema, simplificar el criterio a modelizar es lo adecuado. Para asegurar que las parejas de imágenes a hibridar no tuvieran derechos de autor, además de poder elegir imágenes que fueran fácilmente hibridables, se optó por utilizar Stable Diffusion, un modelo de aprendizaje automático para generar imágenes digitales de alta calidad a partir de descripciones en lenguaje natural.

Otro de los problemas fue la librería usada para la construcción y entrenamiento de las redes. FastAI resultó ser demasiado opaca y limitada de posibilidades para entrenar una red con varias salidas y causó bastante dolores de cabeza ya desde la declaración de un simple DataBlock. Es por ello que se optó por hacer todo desde cero haciendo uso de PyTorch, que es sin duda la librería más usada para tareas de Visión por Computador y que cuenta con una gran comunidad en la red para posibles problemas.

Como se irá detallando en esta memoria, los resultados han sido buenos para la regresión (obtención de sigmas para las convoluciones con la imagen de altas frecuencias y bajas frecuencias), pero no tanto para la clasificación (decisión de a cuál de las dos imágenes se aplicará un filtro paso bajo y otro paso alto).

2. Background

Para comprender el desarrollo del trabajo es indispensable conocer las siguientes tecnologías y conceptos.

2.1. Stable Diffusion

Stable Diffusion [1] es un modelo de aprendizaje automático desarrollado por Runway y LMU Múnich para generar imágenes digitales de alta calidad a partir de descripciones en lenguaje natural o estímulos (prompts en inglés).

El modelo se basa en la técnica de difusión, que consiste en empezar con una imagen aleatoria y, a continuación, ir añadiendo detalles gradualmente hasta llegar a la imagen deseada. En el caso de Stable Diffusion, la imagen aleatoria es una imagen de ruido blanco, y los detalles se añaden a través de un proceso de aprendizaje automático.

Para generar una imagen a partir de una descripción en lenguaje natural, Stable Diffusion utiliza la CNN para transformar la descripción en un conjunto de instrucciones. Estas instrucciones se utilizan a continuación para añadir detalles a la imagen de ruido blanco.

El proceso de difusión es un proceso iterativo, lo que significa que se repite varias veces. En cada iteración, la imagen se modifica ligeramente para que se acerque más a la imagen deseada. De esta forma, se ha generado la primera imagen de la pareja que servirá como plantilla para la segunda, consiguiendo así que la forma de la segunda imagen se parezca lo suficiente a la primera para una posterior hibridación adecuada. Para este proceso, es necesario realizar prueba y error, refinando diversos parámetros y añadiendo prompts negativos para obtener una pareja final con la máxima calidad posible.

2.2. PyTorch

PyTorch [2] es un marco de trabajo de código abierto para el desarrollo de modelos de aprendizaje profundo. Fue desarrollado por el grupo de investigación de inteligencia artificial de Facebook y se ha convertido en una herramienta popular en la comunidad de aprendizaje profundo debido a su flexibilidad y facilidad de uso.

Como se ha comentado en la introducción, la poca transparencia de FastAI ha conducido al uso de PyTorch, que además es la librería en la que se basa FastAI. Dado que PyTorch sí cuenta con esta transparencia y está implementado a más bajo nivel, se ha gozado de una mayor flexibilidad para realizar el proyecto.

2.3. Hibridación de imágenes

La hibridación de imágenes se trata del proceso de generar una nueva imagen en la que para la percepción visual ambas imágenes coinciden, imponiéndose a mayor distancia la elegida para las bajas frecuencias y a menor distancia para las altas. La razón de este efecto reside en el hecho de que el sistema visual humano descarta las bajas frecuencias a menor distancia y las altas a mayor.

Esto se consigue utilizando filtros Gaussianos para el suavizado, en el caso de la imagen de bajas frecuencias, y la Laplaciana de la Gaussiana, la cual se obtiene haciendo la diferencia entre la imagen original y su correspondiente suavizada (esto se considera una buena aproximación de la Laplaciana), para la de altas.

Destacar que los sigmas que definen ambos filtros no tienen por qué coincidir, es más, el problema que se aborda es la determinación de estos dos valores distintos, además de la elección de la imagen a suavizar.

3. Trabajos relacionados

No se ha encontrado ninguna referencia donde se haga algo parecido a lo que se ha hecho en este proyecto, es decir, automatizar la creación de imágenes híbridas a partir de un conjunto de datos debidamente etiquetado según la percepción humana.

No obstante, existe un artículo llamado *Hybrid Images* [3], citado según Google Académico por unos 201 artículos, donde se define lo qué son las imágenes híbridas y se relaciona su obtención con la frecuencia espacial de una imagen.

El objetivo del artículo era explorar una técnica para crear imágenes híbridas que pudieran percibirse de manera diferente a diferentes distancias.

En el artículo, los autores proponen combinar las frecuencias espaciales de dos imágenes diferentes para crear una imagen híbrida. La idea es que a una distancia cercana, el observador percibe las frecuencias espaciales de alta energía, mientras que a una distancia más lejana, las frecuencias de baja energía dominarían la percepción visual. Este fenómeno se conoce como *hybrid images* o imágenes híbridas, y ha demostrado ser una técnica interesante para ilustrar la importancia de las frecuencias espaciales en la percepción visual.

La técnica de imágenes híbridas ha sido utilizada en diversas áreas, como la investigación en percepción visual y la creación de ilusiones visuales.

En términos de dominio de Fourier, las imágenes híbridas se generan manipulando las transformadas de Fourier de las imágenes originales. El dominio de Fourier es una representación matemática que descompone una señal en sus componentes de frecuencia. En el contexto de las imágenes, la transformada de Fourier permite analizar la distribución de frecuencias espaciales presentes en una imagen.

Al combinar las transformadas de Fourier de dos imágenes, una de alta frecuencia y otra de baja frecuencia, y aplicar la transformada inversa de Fourier al resultado, se obtiene una imagen híbrida que puede percibirse de manera diferente a diferentes distancias. Las frecuencias altas se asocian con detalles finos que son más evidentes cuando se observa la imagen de cerca, mientras que las frecuencias bajas contribuyen a la percepción general y son más notables cuando se ve la imagen desde lejos.

En el artículo, no se explora la idea de automatizar este proceso, llegando solo a la evaluación de hibridaciones ya creadas, y es por ello, que se decidió probar el desempeño de una red neuronal en esta tarea.

4. Métodos

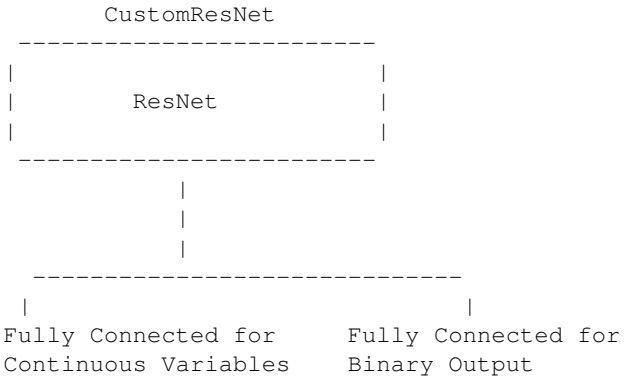
Como el problema a resolver se trata de una regresión junto con una clasificación y se ha creado un DataSet con imágenes hibridadas etiquetadas se abordará haciendo uso de redes neuronales convolucionales. En este caso, es lógico utilizar una red preentrenada, siendo la elección para este trabajo, ResNet-18, que obtiene buenos resultados en el DataSet ImageNet. Esta red se utiliza para un problema de clasificación con más de dos etiquetas, por lo que para nuestro enfoque habrá que modificar la cabecera de la red, manteniendo el cuerpo.

A continuación, se describen los dos enfoques tomados para implementar esta red.

4.1. Red Neuronal Convolucional basada en ResNet-18

En este primer modelo de la red, se utiliza ResNet-18, modificando la cabecera para obtener las salidas deseadas, como hemos adelantado. Es por ello, que al mantener el cuerpo de ResNet-18, debemos adecuar las entradas a esta red, que cuenta con un único input.

Como nuestro problema cuenta con dos imágenes a hibridar, una forma plausible de introducir la pareja a la red sería crear una imagen con dos canales, cada uno representando cada imagen de la pareja. Queda resuelto así con este enfoque la adaptación de las entradas a la red planteada.



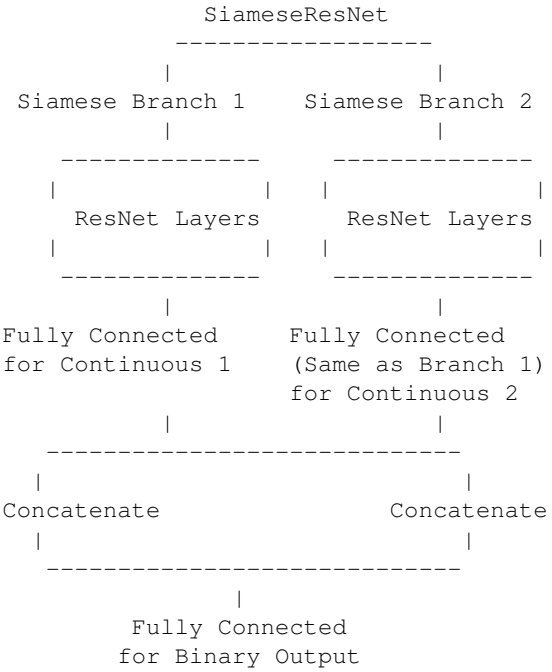
ResNet son las capas de ResNet-18 sin contar la última donde clasifica. Las fully connected para las variables continuas es una capa linear de 512 a 2. La última capa para la salida binaria se trata de una capa linear de 512 a 1.

4.2. Red Siamesa basada en ResNet-18

Una red siamesa es un tipo de arquitectura de red neuronal utilizada en tareas de comparación y reconocimiento de patrones. La idea central detrás de una red siamesa es tener dos ramas idénticas (o "gemelas") que comparten los mismos pesos y arquitectura, y cuyas salidas se comparan para realizar alguna tarea específica.

Visto así, esta arquitectura puede parecer, a priori, que se ajusta bien al problema que tratamos de resolver, la única diferencia, es que, en última instancia, nuestra aproximación deberá devolver dos variables continuas, una por cada rama de la red siamesa (que serían los sigmas de las imágenes). El atributo binario (0 si a la primera imagen se le aplica un filtro paso bajo y 1 si se le aplica el filtro paso bajo a la segunda), se predice concatenando los dos atributos continuos y devolviendo un binario.

El esquema de esta red sería el siguiente:



Resnet Layers se refiere a las capas de ResNet-18 sin contar la última donde clasifica. Las fully connected para las variables continuas son capas lineares de 512 a 1. La última capa para la salida binaria se trata de una capa linear de 1024 a 1.

5. Experimentos

A continuación, vamos a explicar los DataSets utilizados, los distintos experimentos realizados durante la implementación de los modelos determinados y se irá indicando como se ha procedido en la práctica para escoger el mejor modelo.

5.1. DataSet

Para el entrenamiento de los modelos, se ha tenido que crear de cero el DataSet, lo cual implica la creación de las parejas de imágenes hibridables y su posterior etiquetado.

Para la creación de las imágenes, como se ha indicado en la sección 2.1, se ha usado StableDiffusion y para el etiquetado, un integrante del grupo ha hibridado las imágenes en un txt aparte, siendo la línea i las etiquetas de la pareja de imágenes i. Cada pareja de imágenes se identifica porque el nombre de cada imagen empieza por el mismo número.

Si atendemos a las etiquetas, aparecen en el siguiente orden:

```
binaria continua1 continua2
```

Siendo la etiqueta binaria un 0 ó 1, 0 significa que a la primera imagen se le aplica un filtro paso bajo y 1 significa que este filtro se le aplica a la segunda imagen.

Las continuas definen los sigmas de la primera y la segunda imagen respectivamente.

El DataSet de entrenamiento varía ligeramente según nos refiramos a la red explicada en 4.1 o a la red detallada en 4.2. Esto se debe a que la red basada en ResNet-18 necesita una única entrada y la red siamesa necesita una entrada por rama.

Antes de explicar ambos conjuntos de datos por separado, notar que, se parte en todo momento de una carpeta con todas las parejas. Además, en ambos casos se aplica una partición del conjunto de items, reservando el 20 % para la validación y el 80 % restante para entrenamiento.

5.1.1 DataSet para 4.1

Partiendo de la carpeta mencionada anteriormente, se crea una nueva carpeta en tiempo de ejecución (si ya está se borra y se vuelve a crear) y se introducen las parejas como una única imagen en blanco y negro, representando el canal primero a la primera imagen de la pareja y el segundo a la segunda.

Además, para obtener más ejemplos de entrenamiento, se ha invertido el orden de los canales por cada pareja, es decir, en otro ejemplo, el segundo canal ha pasado a ser el primero y viceversa. Esto provoca que haya que cambiar también las etiquetas.

Si la binaria es 0 pasa a 1 y al contrario, la continua1 cambia por la continua2 y al revés de igual manera.

La primera columna de este DataSet es, por tanto, una ruta a la imagen con dos canales que se ha creado en tiempo de ejecución.

5.1.2 DataSet para 4.2

Para este segundo DataSet no se crea ninguna carpeta en ejecución, ya que se puede utilizar directamente las imágenes de la carpeta inicial como entradas a la red, luego el DataSet se genera directamente. a partir de esta carpeta.

El patrón de las etiquetas coincide con el modelo anterior, sustituyendo la columna con la ruta a la imagen de dos canales, por dos columnas con las rutas a las dos imágenes de la pareja en cuestión (en color).

De manera análoga a la gestión de una imagen inversa por cada imagen de dos canales, en este caso las entradas se duplican intercambiando el orden de las imágenes de cada entrada y aplicando el mismo cambio de etiquetas descrito en la subsección anterior.

5.1.3 DataSet para Test

Para el testeo del considerado tras los experimentos como mejor modelo, se ha creado una carpeta a parte siguiendo la forma de proceder para 5.1.1. Solo se ha testeado el modelo para el enfoque 4.1, dado que como se explicará posteriormente, el 4.2 no ha obtenido buenos resultados.

En total, se forman unas 29 parejas, lo cual pareció suficiente para el testeo del modelo.

Para finalizar, indicar que en Train, ambas clases de la etiqueta binaria están igual representadas por la operación de cálculo de la hibridación inversa. Al hacer el split de Validación, se ha comprobado que ambos conjuntos resultantes siguen manteniendo una proporción parecida de 0 y 1.

5.1.4 Transformaciones de los datos

Tanto en validación, train y test, se realizan una serie de transformaciones comunes que son necesarias para el correcto funcionamiento de la red.

- Resize a 224x224. Los modelos se basan en ResNet-18, que requiere que las imágenes sean de ese tamaño.
- Se transforman las imágenes a tensores, para poder operar con PyTorch.
- Las imágenes se normalizan según las estadísticas de ImageNet.

Además, en train, se añade Data Augmentation, con vista a mejorar la generalización del modelo. Además, como contamos con un conjunto de entrenamiento no muy numeroso, esto no será perjudicial. Hay que procurar introducir transformaciones que no necesiten de un cambio de sigmas para hibridar, es decir, no sería conveniente, por ejemplo, añadir un cambio de brillo.

Las transformaciones que se realizan son:

- RandomResizedCrop(224): esta transformación realiza un recorte aleatorio y redimensiona la imagen a un tamaño específico. En este caso, la imagen se recorta de manera aleatoria a una resolución de 224x224 píxeles. Este tipo de transformación es útil para introducir variabilidad en la posición y escala de los objetos en las imágenes durante el entrenamiento.
- RandomHorizontalFlip(): esta transformación voltea horizontalmente la imagen de manera aleatoria con una probabilidad del 50 %. Esto ayuda al modelo a aprender invariantes espaciales en relación con las reflexiones horizontales y mejora la generalización del modelo.
- RandomRotation(45): esta transformación rota la imagen de manera aleatoria en un ángulo entre -45 y 45 grados. Esto proporciona robustez al modelo ante la variabilidad en la orientación de los objetos en las imágenes.

5.2. Enfoque 4.1

Antes de comentar los distintos experimentos, conviene decir los elementos comunes que comparten todos ellos, que en este caso, únicamente, son las funciones de pérdida.

Como tenemos dos problemas, uno de regresión y otro de clasificación, naturalmente, tendrán que ser dos distintas. Se usan MSELoss y BCEWithLogitsLoss.

La función de pérdida MSELoss se utiliza comúnmente en problemas de regresión en redes neuronales. MSE significa Error Cuadrático Medio (Mean Squared Error en inglés), y esta función de pérdida calcula la media de las diferencias al cuadrado entre los valores predichos y los valores reales.

La función de pérdida BCEWithLogitsLoss en PyTorch se utiliza comúnmente en problemas de clasificación binaria. El nombre de la función indica dos componentes clave: *BCE* se refiere a la Entropía Cruzada Binaria (Binary Cross-Entropy), y *WithLogits* indica que la función opera sobre las salidas logits (antes de aplicar la función de activación como la sigmoide).

En problemas de clasificación binaria, donde cada ejemplo puede pertenecer a una de las dos clases, esta función de pérdida es especialmente útil. La BCEWithLogitsLoss combina la aplicación de la función logística (sigmoide) y la Entropía Cruzada Binaria en una sola operación eficiente. Esto simplifica el cómputo y mejora la estabilidad numérica en comparación con realizar estas operaciones por separado.

La función toma como entrada los logits producidos por la última capa de la red, que representan la puntuación antes de aplicar la función de activación. Estos logits pueden ser interpretados como medidas de confianza o evidencia a favor de una clase en particular.

La BCEWithLogitsLoss realiza dos pasos principales. Primero, aplica la función logística (sigmoide) a los logits, convirtiéndolos en valores entre 0 y 1, que pueden interpretarse como probabilidades. Luego, calcula la Entropía Cruzada Binaria entre estas probabilidades y las etiquetas reales.

A continuación, se irán detallando los distintos experimentos en el orden en el que se han hecho en la práctica. Más concretamente, se hablará acerca de los distintos optimizadores probados. Notar que estos optimizadores se han testado con las capas de ResNet-18 sin preentrenar y sin usar Fine Tune.

5.2.1 Optimizadores

- ADAM: Es el algoritmo de optimizador más popular que combina ideas de RMSPROP y SGD. El algoritmo adapta automáticamente las tasas de aprendizaje de cada parámetro, dándoles valores diferentes según la magnitud de los momentos acumulados.
- SGD: Es la base de otros muchos optimizadores más avanzados. La elección de la tasa de aprendizaje es crítica en SGD, ya que una tasa demasiado grande puede hacer que el algoritmo diverja o salte alrededor del mínimo, mientras que una tasa demasiado pequeña puede hacer que la convergencia sea lenta o quede atrapado en mínimos locales.
- ADAGRAD: Como ADAM es un método optimizador adaptativo. Se basa en ajustar la tasa de aprendizaje de cada parámetro de manera individual basándose en su historial de gradientes. Tiene riesgo de provocar una convergencia lenta.
- RMSPROP: Particularmente útil en situaciones donde la magnitud de los gradientes puede variar significativamente a lo largo del entrenamiento. Al adaptar la tasa de aprendizaje de manera más suave, RMSprop puede ser más robusto y converger eficientemente en una variedad de problemas.
- ADADELTA: Trata de abordar la disminución excesiva de la tasa de aprendizaje en ADAGRAD, eliminando la necesidad de una tasa de aprendizaje global al dividir el cambio en los parámetros por la raíz cuadrada del promedio móvil de los cuadrados de los gradientes recientes.
- ADAMW: AdamW es una variante de Adam que incorpora una corrección de peso de decaimiento L2 directamente en la regla de actualización de pesos. Esta corrección tiene como objetivo mitigar el sobreajuste al penalizar explícitamente los pesos grandes.
- ADAMAX: La introducción del máximo acumulado en lugar de la media móvil exponencial de los cuadrados de los gradientes en Adamax es una característica distintiva de este optimizador. Se argumenta que el uso del máximo absoluto puede ser más robusto en algunas situaciones.

Todas las implementaciones de estos optimizadores en PyTorch (menos Adam), permiten indicar un valor de weight decay (también conocido como regularización L2) es una técnica utilizada en el entrenamiento de modelos de aprendizaje automático para evitar el sobreajuste (overfitting). El sobreajuste ocurre cuando un modelo se ajusta demasiado a los datos de entrenamiento y, como resultado, no generaliza bien a datos no vistos, lo que lleva a un rendimiento deficiente en conjunto de prueba o en situaciones del mundo real.

La idea básica detrás del weight decay es penalizar los valores grandes de los pesos (parámetros) del modelo durante el proceso de entrenamiento. Esto se logra agregando un término adicional a la función de pérdida del modelo. La

función de pérdida regularizada L2 se define como la suma de los cuadrados de todos los pesos del modelo multiplicados por un factor de regularización.

Estos valores se han establecido haciendo pruebas y tomando el que mejor curva de aprendizaje obtuviera.

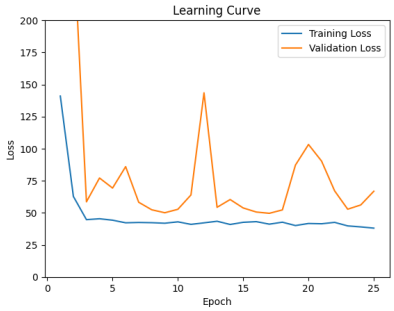


Figura 1. Curva aprendizaje Adam.

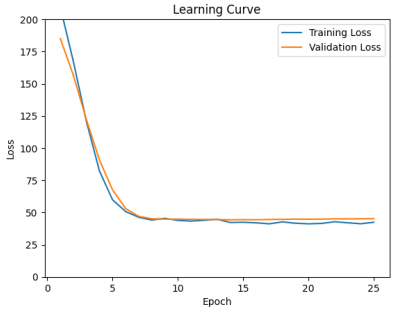


Figura 2. Curva aprendizaje SGD.

5.2.2 Fine Tune

Tras realizar ejecuciones con los distintos optimizadores, se determinó que los dos que mejor perfomaban eran RMSProp y SGD. A partir de ese momento, se optó por realizar el entrenamiento con estos dos optimizadores, siguiendo el enfoque FineTune.

Para ello, se realizan dos etapas de entrenamiento, una con las capas de ResNet-18 (que ahora sí se toma preentrenada) congeladas y otra sin congelar. A la etapa congelada se le asignan 3 épocas y a la otra 30.

A partir de estos dos experimentos, se obtuvieron dos curvas de aprendizaje, ambas entorno a 45 de pérdida:

- La curva de SGD obtiene un mínimo mayor que la de RMSProp, pero sin embargo presenta un comportamiento estable. El modelo pasa a aprender con menor velocidad a partir de la época 7 pero sigue bajando ligeramente.
- La curva de RMSProp obtiene un mínimo menor pero se observa overfit a partir de la época 10.

Finalmente, se decide proseguir con Fine Tune para los siguientes experimentos, porque, si se compara con la ejecución de SGD con ResNet-18 sin preentrenar, a pesar de que esta obtiene una pérdida 4 puntos por debajo (que al tratarse de MSE no sería una diferencia notable en regresión), el loss en train se minimiza más lentamente y la curva se aplana mucho, haciendo sospechar que si se le añadieran más épocas no mejoraría. En la curva de Fine Tune, a pesar de que se aplana también, lo hace en menor medida, y sí que se podría lograr más rendimiento.

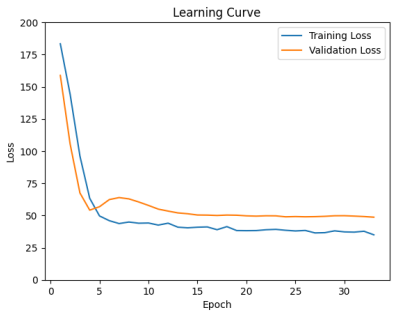


Figura 3. Curva aprendizaje FINE TUNE SGD.

5.2.3 Inicialización de parámetros en las capas FC

Como hasta ahora, se había usado una inicialización random para las capa completamente conectadas de salida del modelo (las demás ya están inicializadas porque se toma ResNet-18 preentrenada), se probó con dos inicializaciones distintas para estas capas:

- Inicialización de Glorot: También conocida como Xavier Initialization, Glorot Initialization está diseñada para funciones de activación que son lineales en el rango cercano a cero, como la tangente hiperbólica (tanh) y la función logística.
- Inicialización He: He Initialization está diseñada para funciones de activación rectificadas lineales (ReLU) y sus variantes (ReLU, Leaky ReLU, Parametric ReLU, etc.).

No se observó ninguna mejora con respecto a las curvas de aprendizaje de la ejecución con la inicialización aleatoria, así que se optó por no incluir ninguna de las dos inicializaciones en los siguientes experimentos.

5.2.4 One Cycle Policy

En Pytorch, se puede introducir esta política mediante un scheduler. A continuación, todas las variables que se nombren se hacen en términos de los parámetros de este.

La política de ciclo único fue introducida por Leslie N. Smith y otros en ”Super-Convergence: Very Fast Training

of Neural Networks Using Large Learning Rates" [4]. Esta política programa la tasa de aprendizaje con un enfriamiento coseno desde lr_{max}/div hasta lr_{max}, y luego hasta lr_{max}/div_{final} (puedes pasar un conjunto de valores a lr_{max} si deseas utilizar tasas de aprendizaje diferenciales). También programa el momento con un recocido coseno según los valores proporcionados en los momentums.

En el enfoque Fine Tune, para hacerlo lo más parecido posible a como está implementada la función Fine Tune en FastAI, se necesitan dos schedulers, uno para la fase de entrenamiento con las capas congeladas y otro para la siguiente fase. La principal características es que el segundo scheduler tiene un lr_{max} de la mitad de valor que el primero.

Tras realizar dos experimentos (uno con las capas de ResNet-18 sin preentrenar y otro usando Fine Tune), se determinó que no merecía la pena mantener esta política, puesto que el primero resultó en una pérdida en validación peor que el modelo que se ha ido escogiendo y el segundo mejoró un poco el loss pero la curva se aplanaba a partir de la época 10, es decir, optimizaba más lentamente. No se aplicará en siguientes experimentos.

5.2.5 Red simple

También se probó con una red simple en las capas las determinamos desde 0. La arquitectura se tomó de la práctica 2 del compañero que mejor resultados tenía.

No se consiguieron unos resultados decentes, ni en valor de pérdida ni en estabilidad de las curvas en train y validation, por lo que de nuevo decidimos descartar este nuevo camino estudiado.

5.2.6 Pruebas para el Batch Size y Learning Rate

Por último, se optó por elegir los mejores valores posibles tanto para el batch size como para el learning rate, a partir de los sucesivos entrenamientos del modelo para los siguientes parámetros propuestos:

- Batch Size = 32 y LR = {0.00001, 0.0001, 0.00002, 0.00004, 0.000005}
- Batch Size = 24 y LR = {0.00001, 0.0001}
- Batch Size = 16 y LR = {0.00001, 0.0001, 0.00002, 0.000005}
- Batch Size = 12 y LR = {0.00001}
- Batch Size = 8 y LR = {0.00001}

Se propuso encontrar una pareja de valores que generara una pérdida en validación baja pero, sobre todo, una gráfica muy estable, de cara a poder ampliar este trabajo con diversas técnicas que no hemos considerado en esta ocasión, para lo que tal estabilidad se vuelve primordial. Elegir la pareja de valores con una menor pérdida asociada, pero acompañada esta de una gráfica muy inestable (con mucho sobreajuste por ejemplo) no sería una buena decisión, volviendo nuestro modelo poco escalable.

Tras los experimentos realizados, llegamos a la conclusión de que la pareja de Batch Size - Learning Rate (de entre los valores estudiados) que minimiza la pérdida en validación es batch size = 32 y learning rate = 0.00004, obteniendo una pérdida de 42.03. Sin embargo, la gráfica presenta una serie de inestabilidades, que nos han llevado a buscar otra pareja de valores que las corrija, sin aumentar demasiado la pérdida.

Para los valores batch size = 16 y learning rate = 0.000005 obtenemos una pérdida en validacón de 46.03 pero con una gráfica muy estable. No consideramos excesiva la diferencia de 4 puntos entre las dos pérdidas mencionadas, teniendo en cuenta que estos valores proceden de la suma de cuadrados, lo que a escalas de pérdidas tan altas crecen significativamente con pequeñas diferencias en las pérdidas de cada etiqueta por separado.

En definitiva, elegimos un batch size de 16 y un learning rate de 0.000005 para continuar con nuestro mejor modelo. A continuación, se muestra las curvas de aprendizaje del modelo elegido.

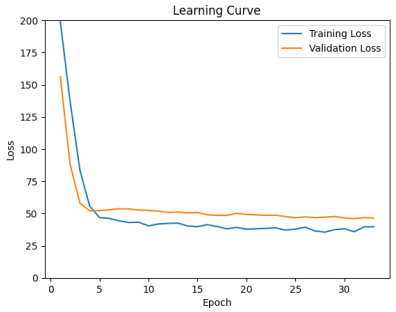


Figura 4. Curva aprendizaje lr = 0.000005 BS = 16.

5.3. Enfoque 4.2

Para la red siamesa únicamente se han realizado dos experimentos haciendo uso de ResNet-18 sin preentrenar y sin el enfoque Fine Tune. Con Adam los resultados no eran muy buenos pero con SGD eran prácticamente iguales que con el enfoque anterior, con loss parecidas tanto para las etiquetas de regresión como para la binaria. Además, se aplana después de aprender más lentamente entorno a 42 de loss, no muy distante de el mejor enfoque que hemos tomado anteriormente, con 46 de loss. Por ello, se decidió no seguir probando ni experimentando con esta arquitectura.

5.4. Testeo del modelo

Para las etiquetas continuas, a diferencia de para el cálculo de la función de pérdida, en el test se ha utilizado el MAE (Mean Absolute Error), siendo esta medida mucho más interpretable a la hora de ver cómo de bien funciona el modelo.

El MAE medio obtenido entre las dos etiquetas continuas tras las pruebas realizadas sobre el conjunto de test ha sido 3.38 (aproximadamente 3.11 para el primer sigma y 3.64 para el segundo). Se consideran válidos estos valores, teniendo en cuenta que a la hora hibridar las imágenes, no se perciben grandes cambios con errores de esta magnitud. En el colab asociado se han añadido unos gráficos de barras que comparan las etiquetas continuas reales con las predicciones del modelo, tanto en los casos en los que acertó la etiqueta binaria como en los que no.

Para la etiqueta binaria se usa el accuracy. Para calcular las predicciones se aplica la sigmoide a la salida binaria y todos los valores mayores a 0.5 se etiquetan como 1 y los menores o iguales como 0. El accuracy logrado ha sido de 0.55, lo cual no dista mucho de un clasificador que etiquetara todas las parejas como 0 o como 1, por lo que no es un gran resultado. Esto se preveía durante el entrenamiento, ya que la pérdida asociada a la etiqueta binaria no se comportaba lo suficientemente bien.

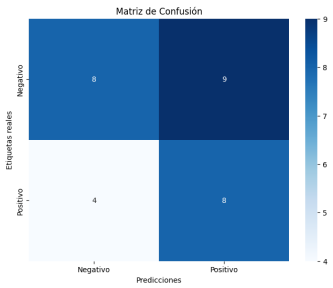


Figura 5. Matriz de confusión Test.

Figura 6. Hibridación con las etiquetas reales de Test.

Se pueden observar dos ejemplos de hibridaciones en estas imágenes, lográndose un buen efecto en la Figura 7 (etiquetas predichas por el modelo). Se ha optado por visualizar las imágenes en color, a pesar de que el modelo testeado opera en blanco y negro. Esto se justifica en el hecho de que los sigmas no varían demasiado entre un mismo par de imágenes en color o en blanco negro.

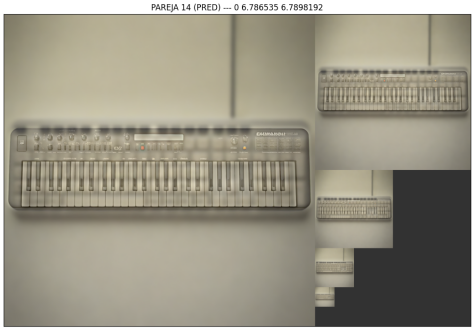


Figura 7. Hibridación con las etiquetas predichas de Test.

6. Conclusiones

Con este trabajo, se ha logrado una aproximación bastante buena de un automatizador de hibridación de imágenes.

Los cálculos de los sigmas no es un problema para el modelo final elegido, siendo el principal inconveniente, la determinación de a qué imagen se le aplica un filtro paso alto y a cuál un filtro paso bajo.

Se podría considerar un futuro proyecto ahondando más en la predicción de etiquetas binarias y considerando una profunda ampliación de las imágenes para entrenamiento. Además, todas las curvas de aprendizaje muestran una ralentización del entrenamiento en épocas cercanas, por lo que sería otra vía de mejora para un nuevo trabajo.

Por último, todos los experimentos se podrían hacer igual con la siamesa, que sería bueno para otro proyecto, aunque por los resultados obtenidos y por el uso típico de las siamesas (comparación de imágenes), probablemente, no se conseguiría mejorar el enfoque elegido para hacer las predicciones en test.

Referencias

[1] GitHub - CompVis/stable-diffusion: A latent text-to-image diffusion model — github.com. https://github.com/CompVis/stable-diffusion. 1

[2] PyTorch — pytorch.org. https://pytorch.org/. 2

[3] Aude Oliva, Antonio Torralba, and Philippe G Schyns. Hybrid images. *ACM Transactions on Graphics (TOG)*, 25(3):527–532, 2006. 1, 2

[4] Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial intelligence and machine learning for multi-domain operations applications*, volume 11006, pages 369–386. SPIE, 2019. 7