

Redes Neuronales y Aprendizaje Profundo Parte 2

Jorge Linde Díaz

CUNEF

December 17, 2021



Redes Neuronales y Aprendizaje Profundo

1 Optimización del aprendizaje

- Técnicas de regularización
- Técnicas de optimización

2 Desarrollo de aplicaciones

- Análisis del problema
- Preparación del dataset
- Modelado y entrenamiento
- Fine Tuning y Evaluación
- Transfer Learning

3 Arquitecturas específicas

- Entrenamientos Personalizados
- ResNet
- Autoencoders
- U-Net
- GANs
- Super Resolution

Índice

1 Optimización del aprendizaje

- Técnicas de regularización
- Técnicas de optimización

2 Desarrollo de aplicaciones

- Análisis del problema
- Preparación del dataset
- Modelado y entrenamiento
- Fine Tuning y Evaluación
- Transfer Learning

3 Arquitecturas específicas

- Entrenamientos Personalizados
- ResNet
- Autoencoders
- U-Net
- GANs
- Super Resolution

Optimización del aprendizaje

Resumen

En este tema veremos qué técnicas podemos utilizar para hacer un aprendizaje más efectivo y robusto. Al ser modelos con una capacidad de aprendizaje tan alta debemos restringir dicha capacidad sin perder rendimiento. Además, buscaremos modelos eficientes desde el punto de vista computacional.

Índice

1 Optimización del aprendizaje

- Técnicas de regularización
- Técnicas de optimización

2 Desarrollo de aplicaciones

- Análisis del problema
- Preparación del dataset
- Modelado y entrenamiento
- Fine Tuning y Evaluación
- Transfer Learning

3 Arquitecturas específicas

- Entrenamientos Personalizados
- ResNet
- Autoencoders
- U-Net
- GANs
- Super Resolution

Técnicas de regularización

Overfitting

El overfitting se produce cuando un modelo funciona mucho mejor en el train dataset que en el validation dataset. Esto se produce cuando la capacidad de entrenamiento del modelo o su complejidad es "mucho mayor" que el tamaño del dataset y hacemos un sobreentrenamiento.

De esta manera el modelo aprende de "memoria" las relaciones $x \longleftrightarrow y$ del train dataset y no generaliza correctamente.

El modelo no aprende a resolver el problema, se ha aprendido el dataset.

Técnicas de regularización

Overfitting

El overfitting se produce cuando un modelo funciona mucho mejor en el train dataset que en el validation dataset. Esto se produce cuando la capacidad de entrenamiento del modelo o su complejidad es "mucho mayor" que el tamaño del dataset y hacemos un sobreentrenamiento.

De esta manera el modelo aprende de "memoria" las relaciones $x \longleftrightarrow y$ del train dataset y no generaliza correctamente.

El modelo no aprende a resolver el problema, se ha aprendido el dataset.

Overfitting

El overfitting se produce cuando un modelo funciona mucho mejor en el train dataset que en el validation dataset. Esto se produce cuando la capacidad de entrenamiento del modelo o su complejidad es "mucho mayor" que el tamaño del dataset y hacemos un sobreentrenamiento.

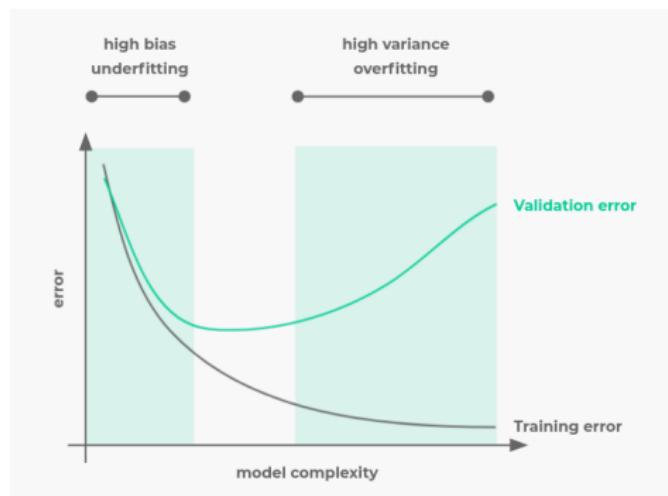
De esta manera el modelo aprende de "memoria" las relaciones $x \longleftrightarrow y$ del train dataset y no generaliza correctamente.

El modelo no aprende a resolver el problema, se ha aprendido el dataset.

Técnicas de regularización

Overfitting

Podemos decir que la complejidad de un modelo es la dimensión del espacio vectorial donde viven sus parámetros o pesos. Las NN tienen una dimensión mucho mayor que los modelos de ML clásicos, como por ejemplo el SVM (?). Por eso es tan fácil hacer overfitting con una NN si el tamaño del dataset no es grande.



Técnicas de regularización

¿Cómo podemos mitigar el overfitting?

Veamos las posibles técnicas de regularización que podemos aplicar a una NN. No es una lista cerrada, pero aquí presentamos las más relevantes

Dataset más grande

Utilizando un dataset más grande durante el entrenamiento evitamos la aparición de overfitting.

Reducir el tamaño del modelo

Así reducimos la capacidad de aprendizaje del modelo y evitamos el aprendizaje 1 a 1 de los elementos del dataset. La idea es entrenar con el tamaño de modelo óptimo. Empezaremos por un modelo pequeño e iremos subiendo hasta que aparezca el overfitting.

Técnicas de regularización

¿Cómo podemos mitigar el overfitting?

Veamos las posibles técnicas de regularización que podemos aplicar a una NN. No es una lista cerrada, pero aquí presentamos las más relevantes

Dataset más grande

Utilizando un dataset más grande durante el entrenamiento evitamos la aparición de overfitting.

Reducir el tamaño del modelo

Así reducimos la capacidad de aprendizaje del modelo y evitamos el aprendizaje 1 a 1 de los elementos del dataset. La idea es entrenar con el tamaño de modelo óptimo. Empezaremos por un modelo pequeño e iremos subiendo hasta que aparezca el overfitting.

Técnicas de regularización

¿Cómo podemos mitigar el overfitting?

Veamos las posibles técnicas de regularización que podemos aplicar a una NN. No es una lista cerrada, pero aquí presentamos las más relevantes

Dataset más grande

Utilizando un dataset más grande durante el entrenamiento evitamos la aparición de overfitting.

Reducir el tamaño del modelo

Así reducimos la capacidad de aprendizaje del modelo y evitamos el aprendizaje 1 a 1 de los elementos del dataset. La idea es entrenar con el tamaño de modelo óptimo. Empezaremos por un modelo pequeño e iremos subiendo hasta que aparezca el overfitting.

Técnicas de regularización

Early Stopping

Esta técnica detiene automáticamente el entrenamiento cuando se empieza a producir el overfitting.

Regularización de los pesos

Introduciremos ciertas limitaciones en los pesos o parámetros de la red para reducir la capacidad de aprendizaje. Podemos limitar el tamaño de los pesos usando, por ejemplo, la L2-regularización.

$$L' = L + \alpha \|\theta\|^2$$

Esta modificación en el loss solo se añade durante el training, por lo que puede producir valores del loss inesperados.

Técnicas de regularización

Early Stopping

Esta técnica detiene automáticamente el entrenamiento cuando se empieza a producir el overfitting.

Regularización de los pesos

Introduciremos ciertas limitaciones en los pesos o parámetros de la red para reducir la capacidad de aprendizaje. Podemos limitar el tamaño de los pesos usando, por ejemplo, la L2-regularización.

$$L' = L + \alpha \|\theta\|^2$$

Esta modificación en el loss solo se añade durante el training, por lo que puede producir valores del loss inesperados.

Técnicas de regularización

Early Stopping

Esta técnica detiene automáticamente el entrenamiento cuando se empieza a producir el overfitting.

Regularización de los pesos

Introduciremos ciertas limitaciones en los pesos o parámetros de la red para reducir la capacidad de aprendizaje. Podemos limitar el tamaño de los pesos usando, por ejemplo, la L2-regularización.

$$L' = L + \alpha \|\theta\|^2$$

Esta modificación en el loss solo se añade durante el training, por lo que puede producir valores del loss inesperados.

Técnicas de regularización

Dropout

El dropout es una técnica indirecta de regularización. La idea es apagar de manera aleatoria en cada step de entrenamiento un porcentaje (`dropout_rate`) de las neuronas, evitando el overfitting ya que la red es cambiante.

Este proceso solo se produce durante el training, por lo que hay que compensar el hecho de que ciertas neuronas no estén activadas. Esto se consigue escalando el output de cada capa de la siguiente manera:

$$\text{output} \rightarrow \frac{\text{output}}{1 - \text{dropout_rate}}$$

Podemos verlo como un ensemble de mini redes, ya que muchas sub redes deberían ponerse de acuerdo para hacer overfitting.

Técnicas de regularización

Dropout

El dropout es una técnica indirecta de regularización. La idea es apagar de manera aleatoria en cada step de entrenamiento un porcentaje (`dropout_rate`) de las neuronas, evitando el overfitting ya que la red es cambiante.

Este proceso solo se produce durante el training, por lo que hay que compensar el hecho de que ciertas neuronas no estén activadas. Esto se consigue escalando el output de cada capa de la siguiente manera:

$$\text{output} \rightarrow \frac{\text{output}}{1 - \text{dropout_rate}}$$

Podemos verlo como un ensemble de mini redes, ya que muchas sub redes deberían ponerse de acuerdo para hacer overfitting.

Técnicas de regularización

Dropout

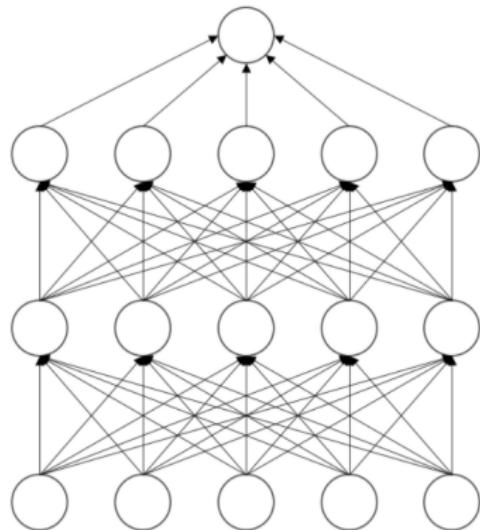
El dropout es una técnica indirecta de regularización. La idea es apagar de manera aleatoria en cada step de entrenamiento un porcentaje (`dropout_rate`) de las neuronas, evitando el overfitting ya que la red es cambiante.

Este proceso solo se produce durante el `training`, por lo que hay que compensar el hecho de que ciertas neuronas no estén activadas. Esto se consigue escalando el `output` de cada capa de la siguiente manera:

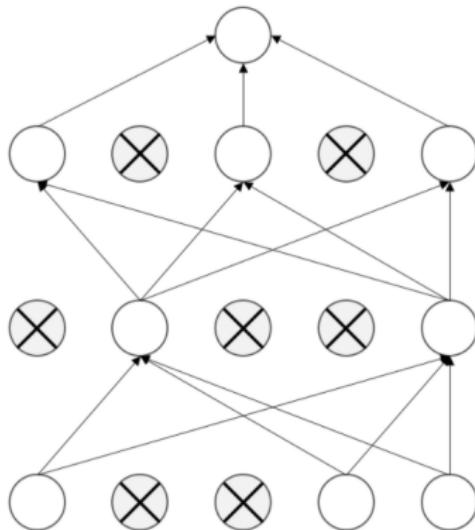
$$\text{output} \rightarrow \frac{\text{output}}{1 - \text{dropout_rate}}$$

Podemos verlo como un ensemble de mini redes, ya que muchas sub redes deberían ponerse de acuerdo para hacer overfitting.

Técnicas de regularización



Standard Neural Net



After applying dropout

Figure: Dropout

Data Augmentation

Esta técnica es una de las más usadas y efectivas, con la que se consiguen modelos robustos frente a cambios en los datos respecto del dataset de entrenamiento. Consiste en la generación aleatoria de datos de training mediante la perturbación de los datos originales del dataset.

Habría que destacar dos tipos de transformaciones para llevar a cabo el data augmentation.

- Las transformaciones que dejan invariantes las etiquetas: Este tipo de transformaciones se consiguen mediante aplicaciones afines, recortes, zoom, rellenado, cambio de luminosidad, etc. Con dichas transformaciones conseguimos aumentar virtualmente el tamaño del dataset.

Data Augmentation

Esta técnica es una de las más usadas y efectivas, con la que se consiguen modelos robustos frente a cambios en los datos respecto del dataset de entrenamiento. Consiste en la generación aleatoria de datos de training mediante la perturbación de los datos originales del dataset.

Habría que destacar dos tipos de transformaciones para llevar a cabo el data augmentation.

- Las transformaciones que dejan invariantes las etiquetas: Este tipo de transformaciones se consiguen mediante aplicaciones afines, recortes, zoom, rellenado, cambio de luminosidad, etc. Con dichas transformaciones conseguimos aumentar virtualmente el tamaño del dataset.

Técnicas de regularización

Data Augmentation

Esta técnica es una de las más usadas y efectivas, con la que se consiguen modelos robustos frente a cambios en los datos respecto del dataset de entrenamiento. Consiste en la generación aleatoria de datos de training mediante la perturbación de los datos originales del dataset.

Habría que destacar dos tipos de transformaciones para llevar a cabo el data augmentation.

- Las transformaciones que dejan invariantes las etiquetas: Este tipo de transformaciones se consiguen mediante aplicaciones afines, recortes, zoom, rellenado, cambio de luminosidad, etc. Con dichas transformaciones conseguimos aumentar virtualmente el tamaño del dataset.

Técnicas de regularización

Data Augmentation

- Las transformaciones que "modifican" ligeramente las etiquetas: Estas transformaciones son más agresivas que las del primer tipo, por ejemplo introduciendo ruido, blur, niebla, modificación ligera de los datos de entrada o de las etiquetas, etc. Con esto conseguimos aumentar el bias y reducir la varianza del modelo.

Existen distintas librerías para implementar el data augmentation, como por ejemplo: Data Augmentation Repo

Técnicas de regularización

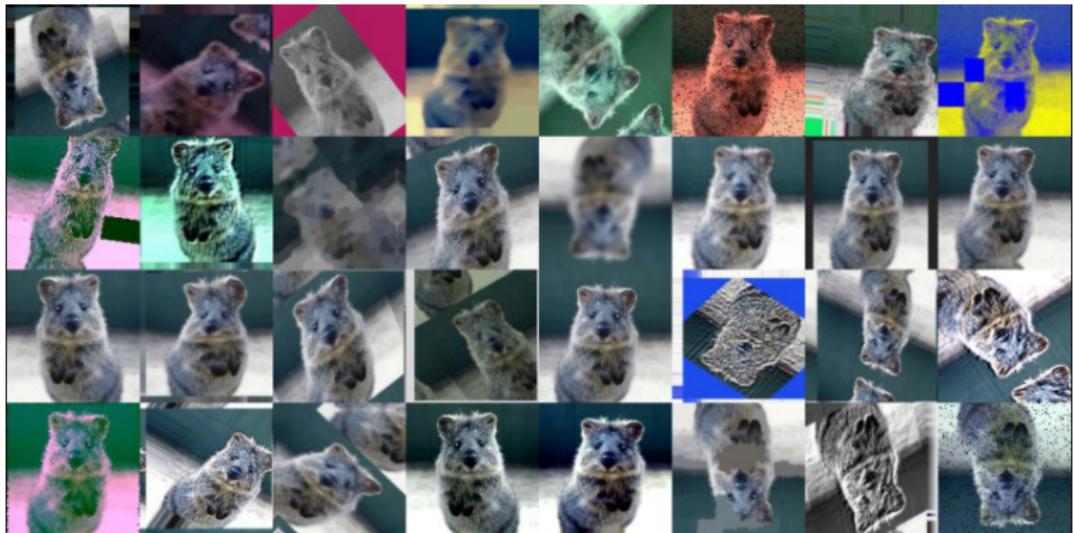


Figure: Data Augmentation

Índice

1 Optimización del aprendizaje

- Técnicas de regularización
- Técnicas de optimización

2 Desarrollo de aplicaciones

- Análisis del problema
- Preparación del dataset
- Modelado y entrenamiento
- Fine Tuning y Evaluación
- Transfer Learning

3 Arquitecturas específicas

- Entrenamientos Personalizados
- ResNet
- Autoencoders
- U-Net
- GANs
- Super Resolution

Técnicas de optimización

Optimización

En esta sección veremos distintas técnicas que mejoran la velocidad de aprendizaje o la velocidad de predicción. En principio estas mejoras no deberían incrementar el performance de nuestro modelo, aunque indirectamente, en muchas ocasiones, sí se produce este incremento.

Batch size correcto

Como ya se ha visto, es muy importante la selección del tamaño del batch size. Hay que tener en cuenta que existe un trade-off entre el tamaño del batch size y el score, pero en general con un batch size relativamente pequeño obtenemos un buen score con unos tiempos de entrenamiento muy reducidos.

Técnicas de optimización

Optimización

En esta sección veremos distintas técnicas que mejoran la velocidad de aprendizaje o la velocidad de predicción. En principio estas mejoras no deberían incrementar el performance de nuestro modelo, aunque indirectamente, en muchas ocasiones, sí se produce este incremento.

Batch size correcto

Como ya se ha visto, es muy importante la selección del tamaño del batch size. Hay que tener en cuenta que existe un trade-off entre el tamaño del batch size y el score, pero en general con un batch size relativamente pequeño obtenemos un buen score con unos tiempos de entrenamiento muy reducidos.

Técnicas de optimización

Batch Normalization

El Batch Normalization es una de las técnicas más potentes para reducir los tiempos de entrenamiento. Gracias a ella es posible el entrenamiento de NN muy profundas (>20 layers).

La normalización sirve para homogeneizar los datos y hacerlos menos dependientes del problema en cuestión. Con ello conseguimos que ciertos modelos se entrenen más rápido. La técnica más habitual de normalización es la transformación:

$$x \rightarrow \frac{x - \mu}{\sigma}$$

El BN es muy similar conceptualmente, normaliza el input de cada layer para que los valores estén en el rango óptimo del entrenamiento (¿?). Esta normalización se hace en cada step del entrenamiento con los datos de media y varianza de ese batch. Con ello conseguimos mejorar la propagación de ∇L a lo largo de la red.

Técnicas de optimización

Batch Normalization

El Batch Normalization es una de las técnicas más potentes para reducir los tiempos de entrenamiento. Gracias a ella es posible el entrenamiento de NN muy profundas (>20 layers).

La normalización sirve para homogeneizar los datos y hacerlos menos dependientes del problema en cuestión. Con ello conseguimos que ciertos modelos se entrenen más rápido. La técnica más habitual de normalización es la transformación:

$$x \rightarrow \frac{x - \mu}{\sigma}$$

El BN es muy similar conceptualmente, normaliza el input de cada layer para que los valores estén en el rango óptimo del entrenamiento (¿?). Esta normalización se hace en cada step del entrenamiento con los datos de media y varianza de ese batch. Con ello conseguimos mejorar la propagación de ∇L a lo largo de la red.

Técnicas de optimización

Batch Normalization

El Batch Normalization es una de las técnicas más potentes para reducir los tiempos de entrenamiento. Gracias a ella es posible el entrenamiento de NN muy profundas (>20 layers).

La normalización sirve para homogeneizar los datos y hacerlos menos dependientes del problema en cuestión. Con ello conseguimos que ciertos modelos se entrenen más rápido. La técnica más habitual de normalización es la transformación:

$$x \rightarrow \frac{x - \mu}{\sigma}$$

El BN es muy similar conceptualmente, normaliza el input de cada layer para que los valores estén en el rango óptimo del entrenamiento (¿?).

Esta normalización se hace en cada step del entrenamiento con los datos de media y varianza de ese batch. Con ello conseguimos mejorar la propagación de ∇L a lo largo de la red.

Técnicas de optimización

Batch Normalization

El Batch Normalization es una de las técnicas más potentes para reducir los tiempos de entrenamiento. Gracias a ella es posible el entrenamiento de NN muy profundas (>20 layers).

La normalización sirve para homogeneizar los datos y hacerlos menos dependientes del problema en cuestión. Con ello conseguimos que ciertos modelos se entrenen más rápido. La técnica más habitual de normalización es la transformación:

$$x \rightarrow \frac{x - \mu}{\sigma}$$

El BN es muy similar conceptualmente, normaliza el input de cada layer para que los valores estén en el rango óptimo del entrenamiento (?). Esta normalización se hace en cada step del entrenamiento con los datos de media y varianza de ese batch. Con ello conseguimos mejorar la propagación de ∇L a lo largo de la red.

Técnicas de optimización

Batch Normalization

Cuando la red es muy grande normalmente se da el Vanishing/Exploding Gradients problem que impide el entrenamiento de la red. Este problema puede ser resuelto usando BN u otras técnicas como el skip connections.

En general se deberá implementar el BN en la mayoría de las capas, pero es muy importante no usarlo en la última.

Durante el test time el BN se congela y la normalización se realiza utilizando la media de las medias y varianzas utilizadas durante el entrenamiento. Por ello es aconsejable esperar a que el modelo converja para que las medias se estabilicen y el performance en test sea el correcto.

Técnicas de optimización

Batch Normalization

Cuando la red es muy grande normalmente se da el Vanishing/Exploding Gradients problem que impide el entrenamiento de la red. Este problema puede ser resuelto usando BN u otras técnicas como el skip connections.

En general se deberá implementar el BN en la mayoría de las capas, pero es muy importante no usarlo en la última.

Durante el test time el BN se congela y la normalización se realiza utilizando la media de las medias y varianzas utilizadas durante el entrenamiento. Por ello es aconsejable esperar a que el modelo converja para que las medias se estabilicen y el performance en test sea el correcto.

Técnicas de optimización

Batch Normalization

Cuando la red es muy grande normalmente se da el Vanishing/Exploding Gradients problem que impide el entrenamiento de la red. Este problema puede ser resuelto usando BN u otras técnicas como el skip connections.

En general se deberá implementar el BN en la mayoría de las capas, pero es muy importante no usarlo en la última.

Durante el test time el BN se congela y la normalización se realiza utilizando la media de las medias y varianzas utilizadas durante el entrenamiento. Por ello es aconsejable esperar a que el modelo converja para que las medias se estabilicen y el performance en test sea el correcto.

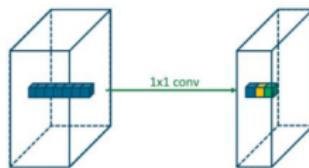
Técnicas de optimización

1x1 convs

Las capas convolucionales con `kernel_size=(1,1)` fueron introducidas por Google en su red GoogLeNet para la modificación de la feature dimension. Este tipo de capas convolucionales pueden usarse para reducir la dimensión de las features y obtener un modelo con menos parámetros sin disminuir su eficiencia. Con ello obtendremos un modelo más eficiente computacionalmente.

Las 1x1 convs pueden verse como una FCNN a nivel de pixel que procesa las features y no mezcla información espacial. Se separa el aprendizaje de features con el espacial.

Es recomendable su uso antes de la parte de clasificación para no obtener un flatten de decenas de miles de neuronas.



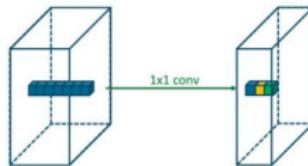
Técnicas de optimización

1x1 convs

Las capas convolucionales con `kernel_size=(1,1)` fueron introducidas por Google en su red GoogLeNet para la modificación de la feature dimension. Este tipo de capas convolucionales pueden usarse para reducir la dimensión de las features y obtener un modelo con menos parámetros sin disminuir su eficiencia. Con ello obtendremos un modelo más eficiente computacionalmente.

Las 1x1 convs pueden verse como una FCNN a nivel de pixel que procesa las features y no mezcla información espacial. Se separa el aprendizaje de features con el espacial.

Es recomendable su uso antes de la parte de clasificación para no obtener un flatten de decenas de miles de neuronas.



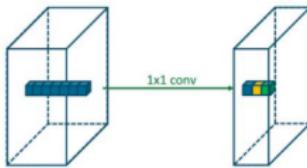
Técnicas de optimización

1x1 convs

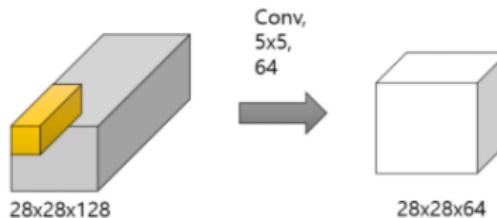
Las capas convolucionales con `kernel_size=(1,1)` fueron introducidas por Google en su red GoogLeNet para la modificación de la feature dimension. Este tipo de capas convolucionales pueden usarse para reducir la dimensión de las features y obtener un modelo con menos parámetros sin disminuir su eficiencia. Con ello obtendremos un modelo más eficiente computacionalmente.

Las 1x1 convs pueden verse como una FCNN a nivel de pixel que procesa las features y no mezcla información espacial. Se separa el aprendizaje de features con el espacial.

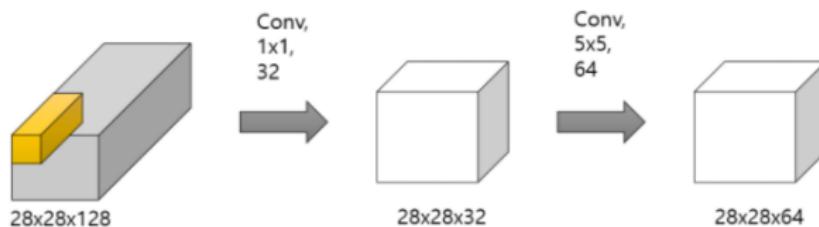
Es recomendable su uso antes de la parte de clasificación para no obtener un flatten de decenas de miles de neuronas.



Técnicas de optimización



$$\#params = 28 \times 28 \times 64 \times 5 \times 5 \times 128 = 160M$$



$$\#params = 28 \times 28 \times 32 \times 128 \times 1 \times 1 = 4.8M$$

$$\#params = 28 \times 28 \times 64 \times 5 \times 5 \times 32 = 40M$$

$$\#total = 44.8M$$

Figure: 1×1 convolutions

Técnicas de optimización

Selección correcta del tamaño del modelo

Como ya hemos visto, el tamaño del modelo está relacionado con su score. Además también lo está con la velocidad de entrenamiento y predicción del mismo.

Dependiendo del caso de uso, puede ser preferible reducir un 1% o 2% el score del modelo si con ello ganamos un 2x o 3x de rendimiento (?). Es muy importante tener esto en cuenta cuando se diseña el modelo para casos de uso de IoT, Drones, dispositivos en el edge, etc.

Feature Engineering

En principio no es necesario hacer feature engineering en modelos de deep learning, ya que no aumentará el score. Sin embargo, si se conoce el caso de uso se puede hacer feature engineering ya que en ciertos casos acelerará el aprendizaje. Por ejemplo normalizando, usando indice NDVI, HSV, etc.

Técnicas de optimización

Selección correcta del tamaño del modelo

Como ya hemos visto, el tamaño del modelo está relacionado con su score. Además también lo está con la velocidad de entrenamiento y predicción del mismo.

Dependiendo del caso de uso, puede ser preferible reducir un 1% o 2% el score del modelo si con ello ganamos un 2x o 3x de rendimiento (?). Es muy importante tener esto en cuenta cuando se diseña el modelo para casos de uso de IoT, Drones, dispositivos en el edge, etc.

Feature Engineering

En principio no es necesario hacer feature engineering en modelos de deep learning, ya que no aumentará el score. Sin embargo, si se conoce el caso de uso se puede hacer feature engineering ya que en ciertos casos acelerará el aprendizaje. Por ejemplo normalizando, usando indice NDVI, HSV, etc.

Técnicas de optimización

Selección correcta del tamaño del modelo

Como ya hemos visto, el tamaño del modelo está relacionado con su score. Además también lo está con la velocidad de entrenamiento y predicción del mismo.

Dependiendo del caso de uso, puede ser preferible reducir un 1% o 2% el score del modelo si con ello ganamos un 2x o 3x de rendimiento (?). Es muy importante tener esto en cuenta cuando se diseña el modelo para casos de uso de IoT, Drones, dispositivos en el edge, etc.

Feature Engineering

En principio no es necesario hacer feature engineering en modelos de deep learning, ya que no aumentará el score. Sin embargo, si se conoce el caso de uso se puede hacer feature engineering ya que en ciertos casos acelerará el aprendizaje. Por ejemplo normalizando, usando indice NDVI, HSV, etc.

Técnicas de optimización

Callbacks

Es recomendable el uso de callbacks como EarlyStopping o ReduceLROnPlateau para hacer entrenamientos más eficientes. Gracias a ellos podremos terminar antes el entrenamiento y acelerar la convergencia del modelo.

TensorFlow Optimization

TensorFlow dispone (beta) de varias optimizaciones para los modelos de deep learning. Dichas optimizaciones reducen la latencia, el tamaño del modelo, agrupan operaciones, etc. Las más relevantes son TF-Lite, Quantization, Pruning, Tensor-RT.

Si vamos a trabajar con dispositivos con una capacidad de cómputo muy limitada es recomendable utilizar estas optimizaciones. [TensorFlow Optimizations Link](#)

Técnicas de optimización

Callbacks

Es recomendable el uso de callbacks como EarlyStopping o ReduceLROnPlateau para hacer entrenamientos más eficientes. Gracias a ellos podremos terminar antes el entrenamiento y acelerar la convergencia del modelo.

TensorFlow Optimization

TensorFlow dispone (beta) de varias optimizaciones para los modelos de deep learning. Dichas optimizaciones reducen la latencia, el tamaño del modelo, agrupan operaciones, etc. Las más relevantes son TF-Lite, Quantization, Pruning, Tensor-RT.

Si vamos a trabajar con dispositivos con una capacidad de cómputo muy limitada es recomendable utilizar estas optimizaciones. TensorFlow Optimizations Link

Técnicas de optimización

Callbacks

Es recomendable el uso de callbacks como EarlyStopping o ReduceLROnPlateau para hacer entrenamientos más eficientes. Gracias a ellos podremos terminar antes el entrenamiento y acelerar la convergencia del modelo.

TensorFlow Optimization

TensorFlow dispone (beta) de varias optimizaciones para los modelos de deep learning. Dichas optimizaciones reducen la latencia, el tamaño del modelo, agrupan operaciones, etc. Las más relevantes son TF-Lite, Quantization, Pruning, Tensor-RT.

Si vamos a trabajar con dispositivos con una capacidad de cómputo muy limitada es recomendable utilizar estas optimizaciones. [TensorFlow Optimizations Link](#)

Índice

1 Optimización del aprendizaje

- Técnicas de regularización
- Técnicas de optimización

2 Desarrollo de aplicaciones

- Análisis del problema
- Preparación del dataset
- Modelado y entrenamiento
- Fine Tuning y Evaluación
- Transfer Learning

3 Arquitecturas específicas

- Entrenamientos Personalizados
- ResNet
- Autoencoders
- U-Net
- GANs
- Super Resolution

Desarrollo de aplicaciones

Resumen

En este tema veremos una serie de pasos generales a tener en cuenta cuando desarrollemos aplicaciones de DL. Cada proyecto es un mundo y, por tanto, habrá que analizar cada uno de manera individual.

Índice

1 Optimización del aprendizaje

- Técnicas de regularización
- Técnicas de optimización

2 Desarrollo de aplicaciones

- Análisis del problema
- Preparación del dataset
- Modelado y entrenamiento
- Fine Tuning y Evaluación
- Transfer Learning

3 Arquitecturas específicas

- Entrenamientos Personalizados
- ResNet
- Autoencoders
- U-Net
- GANs
- Super Resolution

Análisis del problema

Evaluación del método a usar

El primer paso es evaluar cuál es la técnica que mejor se adapta al problema. En función del dataset y los recursos computacionales, deberemos seleccionar o probar un algoritmo clásico, un modelo de ML o uno de DL.

Ejemplo: Detección de falsificación de un DNI usando LBP o Convolucionales.

Análisis del problema

Análisis del dataset

Es muy importante analizar en detalle la calidad del dataset, tanto de los datos como de las etiquetas, si las hubiera. La calidad del dataset marcará una cota superior del score máximo que podremos obtener. Aunque sea trivial la observación, es importante que los datos tengan las features necesarias para poder predecir las etiquetas esperadas.

Ejemplo: LULC, urban vs industrial.

Como ya se ha visto, es muy importante que el tamaño del dataset sea lo suficientemente grande y esté correctamente distribuido. Si no fuera el caso, es recomendable buscar en internet datasets similares que puedan ser incorporados al que ya tenemos.

Análisis del problema

Análisis del dataset

Es muy importante analizar en detalle la calidad del dataset, tanto de los datos como de las etiquetas, si las hubiera. La calidad del dataset marcará una cota superior del score máximo que podremos obtener. Aunque sea trivial la observación, es importante que los datos tengan las features necesarias para poder predecir las etiquetas esperadas.

Ejemplo: LULC, urban vs industrial.

Como ya se ha visto, es muy importante que el tamaño del dataset sea lo suficientemente grande y esté correctamente distribuido. Si no fuera el caso, es recomendable buscar en internet datasets similares que puedan ser incorporados al que ya tenemos.

Análisis del problema

No reinventar la rueda

Existe una gran comunidad de DL, por lo que es altamente recomendable buscar distintos artículos o proyectos ya realizados que nos puedan servir de base o guía. En la mayoría de los casos lo que vamos a hacer ya lo ha hecho alguien, por lo que es preferible invertir tiempo en mejorar algo en vez de construirlo desde cero.

Análisis del problema

Prototipado

Llegados a este punto es recomendable hacer una prueba de concepto con un modelo simple para ver si el problema está bien definido y hay una relación entre x e y . Para ver dicha relación nos podremos ayudar de un Feature Importance, de un Random Forest u otra técnica más avanzada de explicabilidad.

Esta prueba de concepto será satisfactoria si obtenemos un score razonable y el workflow funciona correctamente. Con ello podremos contrastar las ideas iniciales y discernir la viabilidad del proyecto.

Prototipado

Llegados a este punto es recomendable hacer una prueba de concepto con un modelo simple para ver si el problema está bien definido y hay una relación entre x e y . Para ver dicha relación nos podremos ayudar de un Feature Importance, de un Random Forest u otra técnica más avanzada de explicabilidad.

Esta prueba de concepto será satisfactoria si obtenemos un score razonable y el workflow funciona correctamente. Con ello podremos contrastar las ideas iniciales y discernir la viabilidad del proyecto.

Índice

1 Optimización del aprendizaje

- Técnicas de regularización
- Técnicas de optimización

2 Desarrollo de aplicaciones

- Análisis del problema
- Preparación del dataset**
- Modelado y entrenamiento
- Fine Tuning y Evaluación
- Transfer Learning

3 Arquitecturas específicas

- Entrenamientos Personalizados
- ResNet
- Autoencoders
- U-Net
- GANs
- Super Resolution

Preparación del dataset

Limpieza y formato

Al igual que con los modelos de ML, es necesario hacer una limpieza y homogeneización de los datos para que no falten valores o la misma feature venga expresada de varias formas distintas.

Además, tendremos que dar un formato específico a los datos que sea compatible con la entrada de una red neuronal. Por ejemplo, podemos usar un word embedding para pasar palabras a un vector de longitud fija o reescalar las imágenes de nuestro dataset para que todas tengan la misma dimensión.

Preparación del dataset

Limpieza y formato

Al igual que con los modelos de ML, es necesario hacer una limpieza y homogeneización de los datos para que no falten valores o la misma feature venga expresada de varias formas distintas.

Además, tendremos que dar un formato específico a los datos que sea compatible con la entrada de una red neuronal. Por ejemplo, podemos usar un word embedding para pasar palabras a un vector de longitud fija o reescalar las imágenes de nuestro dataset para que todas tengan la misma dimensión.

Preparación del dataset

Tratamiento del dato

Es recomendable normalizar los datos de entrada entre $[0, 1]$ o $[-1, 1]$ para acelerar el aprendizaje ¿?. Es muy importante, si tenemos features de distinta naturaleza (RGB vs HSV), que la normalización se lleve a cabo en cada feature de manera independiente ¿?.

También es recomendable, para mejorar la velocidad de aprendizaje, incluir todas las features relevantes y trabajar en un espacio de representación adecuado.

Es importante que cualquier transformación que se haga a los datos de train o validation también se haga a los datos de test para que la distribución y features en dichos conjuntos sean similares. Por ejemplo: Coordenadas UTM vs geográficas.

Preparación del dataset

Tratamiento del dato

Es recomendable normalizar los datos de entrada entre $[0, 1]$ o $[-1, 1]$ para acelerar el aprendizaje ¿?. Es muy importante, si tenemos features de distinta naturaleza (RGB vs HSV), que la normalización se lleve a cabo en cada feature de manera independiente ¿?.

También es recomendable, para mejorar la velocidad de aprendizaje, incluir todas las features relevantes y trabajar en un espacio de representación adecuado.

Es importante que cualquier transformación que se haga a los datos de train o validation también se haga a los datos de test para que la distribución y features en dichos conjuntos sean similares. Por ejemplo: Coordenadas UTM vs geográficas.

Preparación del dataset

Tratamiento del dato

Es recomendable normalizar los datos de entrada entre $[0, 1]$ o $[-1, 1]$ para acelerar el aprendizaje ¿?. Es muy importante, si tenemos features de distinta naturaleza (RGB vs HSV), que la normalización se lleve a cabo en cada feature de manera independiente ¿?.

También es recomendable, para mejorar la velocidad de aprendizaje, incluir todas las features relevantes y trabajar en un espacio de representación adecuado.

Es importante que cualquier transformación que se haga a los datos de train o validation también se haga a los datos de test para que la distribución y features en dichos conjuntos sean similares. Por ejemplo: Coordenadas UTM vs geográficas.

Preparación del dataset

Especificar el dtype

Para una optimización de los recursos, muchas veces es necesario seleccionar un dtype específico para los datos. Por ejemplo:

- Bool: 0,1
- uint8: 0,...,255
- int16: -32768,...,32767
- float32: -3.4028235e+38,...,3.4028235e+38

Índice

1 Optimización del aprendizaje

- Técnicas de regularización
- Técnicas de optimización

2 Desarrollo de aplicaciones

- Análisis del problema
- Preparación del dataset
- Modelado y entrenamiento**
- Fine Tuning y Evaluación
- Transfer Learning

3 Arquitecturas específicas

- Entrenamientos Personalizados
- ResNet
- Autoencoders
- U-Net
- GANs
- Super Resolution

Selección de la arquitectura

Tendremos que seleccionar un tipo de NN específica que se adapte mejor a nuestro problema. Por ejemplo:

- Datos tabulares: FCNN, CNN
- Serie temporal: RNN, CNN, FCNN
- Imágenes: CNN
- Vídeo: CNN, RNN

Una vez seleccionemos el tipo de red tendremos que elegir o construir una arquitectura en particular. Podemos considerar la arquitectura de la red como otro hiperparámetro más. Hay que tener en cuenta que, en función de la arquitectura, la red trabajará de forma distinta.

Selección de la arquitectura

Tendremos que seleccionar un tipo de NN específica que se adapte mejor a nuestro problema. Por ejemplo:

- Datos tabulares: FCNN, CNN
- Serie temporal: RNN, CNN, FCNN
- Imágenes: CNN
- Vídeo: CNN, RNN

Una vez seleccionemos el tipo de red tendremos que elegir o construir una arquitectura en particular. Podemos considerar la arquitectura de la red como otro hiperparámetro más. Hay que tener en cuenta que, en función de la arquitectura, la red trabajará de forma distinta.

Modelado y entrenamiento

Entrenamiento

Una vez seleccionada la arquitectura tendremos que definir las funciones necesarias para entrenar y validar la red.

- ① La métrica nos determinará el rendimiento de nuestro modelo y deberá ser definida acorde a los requisitos del proyecto.
- ② La función loss la usaremos para entrenar el modelo y tendrá que tener cierta monotonía con la métrica. Podemos seleccionar una función predefinida o crearnos una ad-hoc.
- ③ Seleccionaremos el optimizador que mejor se adapte a nuestro problema (en general Adam) y, además, podremos seleccionar una estrategia de descenso del Learning Rate.

Además se deberá hacer una correcta selección de los parámetros de entrenamiento Batch size, epochs y Learning Rate inicial, ya que con ellos podemos modificar la velocidad y calidad del entrenamiento.

Modelado y entrenamiento

Entrenamiento

Una vez seleccionada la arquitectura tendremos que definir las funciones necesarias para entrenar y validar la red.

- ① La métrica nos determinará el rendimiento de nuestro modelo y deberá ser definida acorde a los requisitos del proyecto.
- ② La función loss la usaremos para entrenar el modelo y tendrá que tener cierta monotonía con la métrica. Podemos seleccionar una función predefinida o crearnos una ad-hoc.
- ③ Seleccionaremos el optimizador que mejor se adapte a nuestro problema (en general Adam) y, además, podremos seleccionar una estrategia de descenso del Learning Rate.

Además se deberá hacer una correcta selección de los parámetros de entrenamiento Batch size, epochs y Learning Rate inicial, ya que con ellos podemos modificar la velocidad y calidad del entrenamiento.

Modelado y entrenamiento

Entrenamiento

Una vez seleccionada la arquitectura tendremos que definir las funciones necesarias para entrenar y validar la red.

- ① La métrica nos determinará el rendimiento de nuestro modelo y deberá ser definida acorde a los requisitos del proyecto.
- ② La función loss la usaremos para entrenar el modelo y tendrá que tener cierta monotonía con la métrica. Podemos seleccionar una función predefinida o crearnos una ad-hoc.
- ③ Seleccionaremos el optimizador que mejor se adapte a nuestro problema (en general Adam) y, además, podremos seleccionar una estrategia de descenso del Learning Rate.

Además se deberá hacer una correcta selección de los parámetros de entrenamiento Batch size, epochs y Learning Rate inicial, ya que con ellos podemos modificar la velocidad y calidad del entrenamiento.

Modelado y entrenamiento

Entrenamiento

Una vez seleccionada la arquitectura tendremos que definir las funciones necesarias para entrenar y validar la red.

- ① La métrica nos determinará el rendimiento de nuestro modelo y deberá ser definida acorde a los requisitos del proyecto.
- ② La función loss la usaremos para entrenar el modelo y tendrá que tener cierta monotonía con la métrica. Podemos seleccionar una función predefinida o crearnos una ad-hoc.
- ③ Seleccionaremos el optimizador que mejor se adapte a nuestro problema (en general Adam) y, además, podremos seleccionar una estrategia de descenso del Learning Rate.

Además se deberá hacer una correcta selección de los parámetros de entrenamiento Batch size, epochs y Learning Rate inicial, ya que con ellos podemos modificar la velocidad y calidad del entrenamiento.

Modelado y entrenamiento

Entrenamiento

Una vez seleccionada la arquitectura tendremos que definir las funciones necesarias para entrenar y validar la red.

- ① La métrica nos determinará el rendimiento de nuestro modelo y deberá ser definida acorde a los requisitos del proyecto.
- ② La función loss la usaremos para entrenar el modelo y tendrá que tener cierta monotonía con la métrica. Podemos seleccionar una función predefinida o crearnos una ad-hoc.
- ③ Seleccionaremos el optimizador que mejor se adapte a nuestro problema (en general Adam) y, además, podremos seleccionar una estrategia de descenso del Learning Rate.

Además se deberá hacer una correcta selección de los parámetros de entrenamiento Batch size, epochs y Learning Rate inicial, ya que con ellos podemos modificar la velocidad y calidad del entrenamiento.

Índice

1 Optimización del aprendizaje

- Técnicas de regularización
- Técnicas de optimización

2 Desarrollo de aplicaciones

- Análisis del problema
- Preparación del dataset
- Modelado y entrenamiento
- Fine Tuning y Evaluación**
- Transfer Learning

3 Arquitecturas específicas

- Entrenamientos Personalizados
- ResNet
- Autoencoders
- U-Net
- GANs
- Super Resolution

Fine Tuning y Evaluación

Fine Tuning

Una vez tengamos nuestro modelo entrenado y sepamos que el workflow es el correcto, deberemos mejorarlo al máximo optimizando los hiperparámetros sobre el conjunto de validación. Podremos usar la técnica k-fold cross-validation para reducir la varianza del score¿?.

Deberemos optimizar las capas, neuronas, funciones, LR...

Como el espacio de hiperparámetros tiene una dimensión grande y la métrica no es diferenciable sobre ellos, es muy difícil obtener el óptimo. Lo mejor es buscar una región sub-óptima con un random search y luego optimizar en esa región con un grid search.

Leaks de información

Durante esta etapa es muy importante no tener leaks de información entre los datasets train, validation y test, ya que supondría un falseamiento del score. Por ejemplo, hay que tener cuidado con "arrow of time", datos redundantes y datasets no aleatorizados.

Fine Tuning y Evaluación

Fine Tuning

Una vez tengamos nuestro modelo entrenado y sepamos que el workflow es el correcto, deberemos mejorarlo al máximo optimizando los hiperparámetros sobre el conjunto de validación. Podremos usar la técnica k-fold cross-validation para reducir la varianza del score¿?.

Deberemos optimizar las capas, neuronas, funciones, LR...

Como el espacio de hiperparámetros tiene una dimensión grande y la métrica no es diferenciable sobre ellos, es muy difícil obtener el óptimo. Lo mejor es buscar una región sub-óptima con un random search y luego optimizar en esa región con un grid search.

Leaks de información

Durante esta etapa es muy importante no tener leaks de información entre los datasets train, validation y test, ya que supondría un falseamiento del score. Por ejemplo, hay que tener cuidado con "arrow of time", datos redundantes y datasets no aleatorizados.

Fine Tuning y Evaluación

Fine Tuning

Una vez tengamos nuestro modelo entrenado y sepamos que el workflow es el correcto, deberemos mejorarlo al máximo optimizando los hiperparámetros sobre el conjunto de validación. Podremos usar la técnica k-fold cross-validation para reducir la varianza del score¿?.

Deberemos optimizar las capas, neuronas, funciones, LR...

Como el espacio de hiperparámetros tiene una dimensión grande y la métrica no es diferenciable sobre ellos, es muy difícil obtener el óptimo. Lo mejor es buscar una región sub-óptima con un random search y luego optimizar en esa región con un grid search.

Leaks de información

Durante esta etapa es muy importante no tener leaks de información entre los datasets train, validation y test, ya que supondría un falseamiento del score. Por ejemplo, hay que tener cuidado con "arrow of time", datos redundantes y datasets no aleatorizados.

Fine Tuning y Evaluación

Metodología de evaluación

Para que el score obtenido de la métrica sea lo más representativo posible, es necesario que el proceso de evaluación esté bien definido y sea consistente.

Por lo tanto:

- El procedimiento de evaluación ha de ser fijo.
- La métrica no puede cambiar.
- El dataset sobre el que se evalúa ha de ser siempre el mismo.

De no ser así se producirían perturbaciones en el score y se falsearía los resultados.

Fine Tuning y Evaluación

Metodología de evaluación

Para que el score obtenido de la métrica sea lo más representativo posible, es necesario que el proceso de evaluación esté bien definido y sea consistente.

Por lo tanto:

- El procedimiento de evaluación ha de ser fijo.
- La métrica no puede cambiar.
- El dataset sobre el que se evalúa ha de ser siempre el mismo.

De no ser así se producirían perturbaciones en el score y se falsearía los resultados.

Fine Tuning y Evaluación

Evaluación final

Una vez hayamos terminado con el fine tuning, procederemos a evaluar nuestro modelo sobre el conjunto de test, exclusivamente una sola vez y luego se descartará dicho dataset.

Ciclo de vida

Puede ser que el score del modelo no sea el deseado después de este procedimiento. En tal caso se producirá un ciclo de mejora iterativo en el se vayan añadiendo cada vez mejoras en las features, arquitectura y dataset.

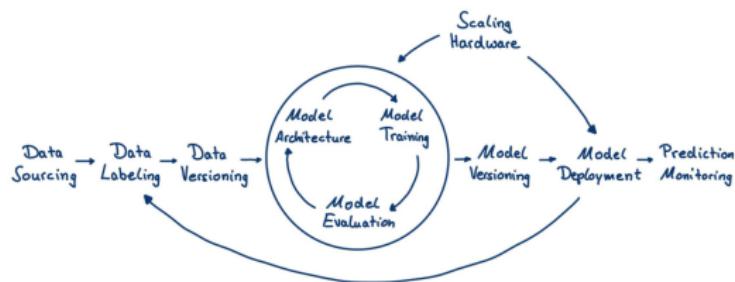


Figure: Ciclo de vida

Fine Tuning y Evaluación

Evaluación final

Una vez hayamos terminado con el fine tuning, procederemos a evaluar nuestro modelo sobre el conjunto de test, exclusivamente una sola vez y luego se descartará dicho dataset.

Ciclo de vida

Puede ser que el score del modelo no sea el deseado después de este procedimiento. En tal caso se producirá un ciclo de mejora iterativo en el se vayan añadiendo cada vez mejoras en las features, arquitectura y dataset.

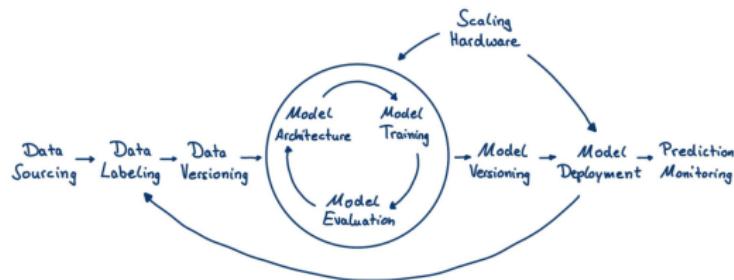


Figure: Ciclo de vida

Índice

1 Optimización del aprendizaje

- Técnicas de regularización
- Técnicas de optimización

2 Desarrollo de aplicaciones

- Análisis del problema
- Preparación del dataset
- Modelado y entrenamiento
- Fine Tuning y Evaluación
- Transfer Learning

3 Arquitecturas específicas

- Entrenamientos Personalizados
- ResNet
- Autoencoders
- U-Net
- GANs
- Super Resolution

Transfer Learning

Transfer Learning

Como ya se ha visto, para entrenar una NN se necesita una gran cantidad de datos y recursos computacionales. Cuando no disponemos de alguno de ellos podemos recurrir al Transfer Learning, con el cual podremos adaptar un modelo, ya entrenado en un problema similar, para resolver nuestro problema.

'Un problema similar' se refiere a que los datasets usados para resolver dichos problemas tengan features de bajo nivel muy similares. Esta técnica es muy usada en problemas relacionados con imágenes y texto.

Transfer Learning

Transfer Learning

Como ya se ha visto, para entrenar una NN se necesita una gran cantidad de datos y recursos computacionales. Cuando no disponemos de alguno de ellos podemos recurrir al Transfer Learning, con el cual podremos adaptar un modelo, ya entrenado en un problema similar, para resolver nuestro problema.

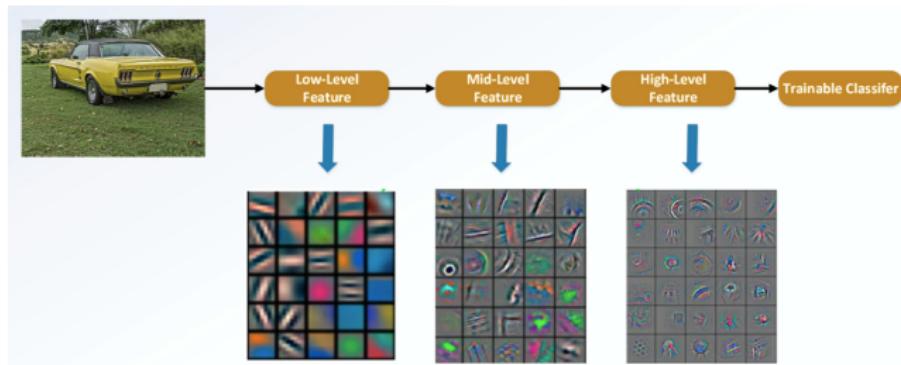
'Un problema similar' se refiere a que los datasets usados para resolver dichos problemas tengan features de bajo nivel muy similares. Esta técnica es muy usada en problemas relacionados con imágenes y texto.

Transfer Learning

Levels of features

La idea detrás del Transfer Learning es que un modelo entrenado en un dataset grande y variado puede ser usado como extractor de features, las cuales deberían ser representativas en problemas similares. Por ejemplo, podríamos usar un modelo entrenado en ImageNet para detectar tipos de coches.

Esta característica de extracción de features, para problemas similares, es una diferencia muy relevante del DL frente a modelos Shallow-Learning (features predefinidas).

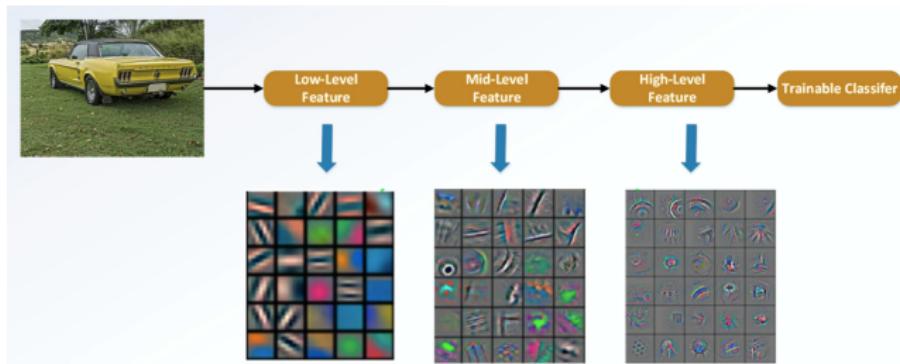


Transfer Learning

Levels of features

La idea detrás del Transfer Learning es que un modelo entrenado en un dataset grande y variado puede ser usado como extractor de features, las cuales deberían ser representativas en problemas similares. Por ejemplo, podríamos usar un modelo entrenado en ImageNet para detectar tipos de coches.

Esta característica de extracción de features, para problemas similares, es una diferencia muy relevante del DL frente a modelos Shallow-Learning (features predefinidas).



Transfer Learning

Tipos de Transfer Learning

Principalmente existen dos maneras de usar el transfer learning.

- ① Extractor de features: Utilizaremos una parte de la red para obtener features representativas.
- ② Fine Tuning: Modificaremos una parte de la red para adaptarla a nuestro modelo.

Es importante, en cualquier caso, que el modelo que vayamos a usar esté totalmente entrenado. Con ello podemos asumir que la parte de feature extraction de la red es buena y el error que introduce no es grande.

En general solo vamos a modificar las capas superiores de la red, que son las que se encargan de resolver el problema específico. Además, cuantas menos capas modifiquemos más pequeño va a ser el dataset necesario para entrenar el modelo y menos overfitting se producirá.

Transfer Learning

Tipos de Transfer Learning

Principalmente existen dos maneras de usar el transfer learning.

- ① Extractor de features: Utilizaremos una parte de la red para obtener features representativas.
- ② Fine Tuning: Modificaremos una parte de la red para adaptarla a nuestro modelo.

Es importante, en cualquier caso, que el modelo que vayamos a usar esté totalmente entrenado. Con ello podemos asumir que la parte de feature extraction de la red es buena y el error que introduce no es grande.

En general solo vamos a modificar las capas superiores de la red, que son las que se encargan de resolver el problema específico. Además, cuantas menos capas modifiquemos más pequeño va a ser el dataset necesario para entrenar el modelo y menos overfitting se producirá.

Transfer Learning

Tipos de Transfer Learning

Principalmente existen dos maneras de usar el transfer learning.

- ① Extractor de features: Utilizaremos una parte de la red para obtener features representativas.
- ② Fine Tuning: Modificaremos una parte de la red para adaptarla a nuestro modelo.

Es importante, en cualquier caso, que el modelo que vayamos a usar esté totalmente entrenado. Con ello podemos asumir que la parte de feature extraction de la red es buena y el error que introduce no es grande.

En general solo vamos a modificar las capas superiores de la red, que son las que se encargan de resolver el problema específico. Además, cuantas menos capas modifiquemos más pequeño va a ser el dataset necesario para entrenar el modelo y menos overfitting se producirá.

Transfer Learning

Extractor de features

En este caso utilizaremos las features generadas por un modelo preentrenado como input para un nuevo modelo. En este caso solo tendríamos que entrenar una red pequeña en nuestro dataset.

El número de capas con las que nos quedaremos variará en función de la similitud de los problemas.

Podemos implementar este método de dos maneras.

Transfer Learning

Extractor de features

En este caso utilizaremos las features generadas por un modelo preentrenado como input para un nuevo modelo. En este caso solo tendríamos que entrenar una red pequeña en nuestro dataset. El número de capas con las que nos quedaremos variará en función de la similitud de los problemas.

Podemos implementar este método de dos maneras.

Transfer Learning

Extractor de features

En este caso utilizaremos las features generadas por un modelo preentrenado como input para un nuevo modelo. En este caso solo tendríamos que entrenar una red pequeña en nuestro dataset. El número de capas con las que nos quedaremos variará en función de la similitud de los problemas.

Podemos implementar este método de dos maneras.

Transfer Learning

Extractor de features: Método 1

Guardamos las features (output de la capa seleccionada) en numpy's y usamos esos datos como input para entrenar nuestro modelo en vez del input original x . Con ello conseguimos que el entrenamiento sea más rápido, ya que no tiene que procesar los datos, aunque el tiempo en predicción es el mismo.

Este método tiene dos inconvenientes:

- ① No podemos utilizar técnicas de data augmentation.
- ② Puede ser que las features ocupen mucho más espacio que el input original.

Transfer Learning

Extractor de features: Método 1

Guardamos las features (output de la capa seleccionada) en numpy's y usamos esos datos como input para entrenar nuestro modelo en vez del input original x . Con ello conseguimos que el entrenamiento sea más rápido, ya que no tiene que procesar los datos, aunque el tiempo en predicción es el mismo.

Este método tiene dos inconvenientes:

- ① No podemos utilizar técnicas de data augmentation.
- ② Puede ser que las features ocupen mucho más espacio que el input original.

Transfer Learning

Extractor de features: Método 2

Este método es mucho más elegante y recomendable. Consiste en poner nuestra nueva red encima de la red inicial, congelando las capas de esta última para que no se modifiquen. Utilizaremos esta nueva red como una red normal, donde el proceso de predicción y entrenamiento sería el habitual.

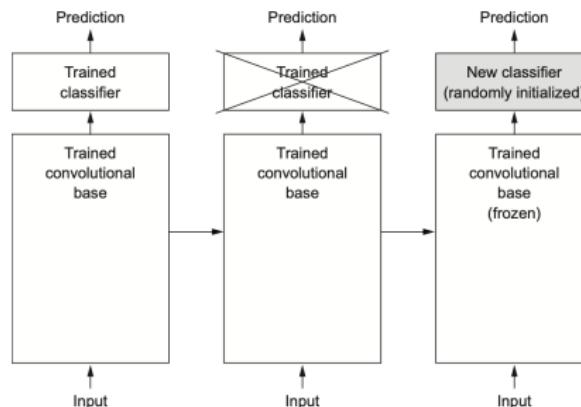


Figure: Extractor de features

Transfer Learning

Fine Tuning

Esta técnica se aplica cuando el problema es el mismo y se quiere reentrenar en otro dataset con una distribución distinta de los datos. En este caso congelaremos las primeras capas de la red y las últimas capas las dejaremos intactas. Finalmente entrenaríamos el modelo en el nuevo dataset.

Con ello conseguimos reutilizar un modelo entrenado previamente para obtener otro especializado en un nuevo dataset y con un coste mucho menor.

Transfer Learning

Fine Tuning

Esta técnica se aplica cuando el problema es el mismo y se quiere reentrenar en otro dataset con una distribución distinta de los datos. En este caso congelaremos las primeras capas de la red y las últimas capas las dejaremos intactas. Finalmente entrenaríamos el modelo en el nuevo dataset. Con ello conseguimos reutilizar un modelo entrenado previamente para obtener otro especializado en un nuevo dataset y con un coste mucho menor.

Transfer Learning

Modelos preentrenados

En Github se pueden encontrar una gran cantidad de modelos entrenados que pueden ser reutilizados. Además, en la siguiente página se puede encontrar una gran variedad de modelos entrenados por tensorflow.

[TensorFlow Hub Link](#)

Índice

1 Optimización del aprendizaje

- Técnicas de regularización
- Técnicas de optimización

2 Desarrollo de aplicaciones

- Análisis del problema
- Preparación del dataset
- Modelado y entrenamiento
- Fine Tuning y Evaluación
- Transfer Learning

3 Arquitecturas específicas

- Entrenamientos Personalizados
- ResNet
- Autoencoders
- U-Net
- GANs
- Super Resolution

Arquitecturas específicas

Resumen

En este tema veremos varias arquitecturas específicas diseñadas para la solución de problemas concretos. Este tipo de arquitecturas se han diseñado Ad-hoc usando los componentes vistos previamente.

Índice

1 Optimización del aprendizaje

- Técnicas de regularización
- Técnicas de optimización

2 Desarrollo de aplicaciones

- Análisis del problema
- Preparación del dataset
- Modelado y entrenamiento
- Fine Tuning y Evaluación
- Transfer Learning

3 Arquitecturas específicas

• Entrenamientos Personalizados

- ResNet
- Autoencoders
- U-Net
- GANs
- Super Resolution

Entrenamientos Personalizados

Entrenamientos Personalizados

Dependiendo del problema puede ser que no queramos hacer un entrenamiento estándar, por ejemplo:

- ① Entrenamiento de solo una parte de la red en modelos multi-input/multi-output.
- ② Entrenar con una loss dinámica.
- ③ Modificar los pesos de la red trabajando directamente con el gradiente.

Entrenamientos Personalizados

Entrenamientos Personalizados

Dependiendo del problema puede ser que no queramos hacer un entrenamiento estándar, por ejemplo:

- ① Entrenamiento de solo una parte de la red en modelos multi-input/multi-output.
- ② Entrenar con una loss dinámica.
- ③ Modificar los pesos de la red trabajando directamente con el gradiente.

Entrenamientos Personalizados

Entrenamientos Personalizados

Dependiendo del problema puede ser que no queramos hacer un entrenamiento estándar, por ejemplo:

- ① Entrenamiento de solo una parte de la red en modelos multi-input/multi-output.
- ② Entrenar con una loss dinámica.
- ③ Modificar los pesos de la red trabajando directamente con el gradiente.

Entrenamientos Personalizados

Entrenamientos Personalizados

Dependiendo del problema puede ser que no queramos hacer un entrenamiento estándar, por ejemplo:

- ① Entrenamiento de solo una parte de la red en modelos multi-input/multi-output.
- ② Entrenar con una loss dinámica.
- ③ Modificar los pesos de la red trabajando directamente con el gradiente.

Entrenamientos Personalizados

Entrenamientos Personalizados

Para ello necesitamos crear un bloque de código que produzca un step de entrenamiento. La estructura típica sería:

```
with tf.GradientTape() as tape:  
    output = model(input, training=True)  
    loss = loss(labels, output)
```

```
gradients = tape.gradient(loss, model.trainable_variables)  
optimizer.apply_gradients(zip(gradients, model.trainable_variables))
```

Entrenamientos Personalizados

Entrenamientos Personalizados

Con este custom training loop seremos capaces de entrenar modelos más complejos que requieran modificaciones específicas en cada momento. Este proceso de entrenamiento será necesario para ciertas arquitecturas.

Índice

1 Optimización del aprendizaje

- Técnicas de regularización
- Técnicas de optimización

2 Desarrollo de aplicaciones

- Análisis del problema
- Preparación del dataset
- Modelado y entrenamiento
- Fine Tuning y Evaluación
- Transfer Learning

3 Arquitecturas específicas

- Entrenamientos Personalizados
- **ResNet**
- Autoencoders
- U-Net
- GANs
- Super Resolution

ResNet

ResNet

ResNet (Residual Neural Network) es un tipo de arquitectura que se diseñó para el campeonato ImageNet y que, de hecho, lo ganó. ResNet utiliza una sucesión de módulos con skip connections para solventar dos problemas que aparecen cuando se entrena redes neuronales muy profundas, los cuales son:

- Vanishing Gradient Problem
- Representation Bottleneck

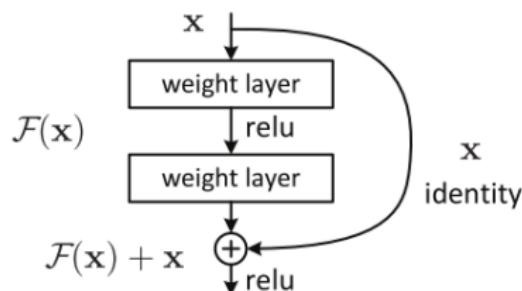


Figure: Módulo ResNet

Vanishing Gradient Problem

Para entrenar una NN debemos propagar el error por todas las capas de la red utilizando el ∇L . Si la red tiene muchas capas puede ser que el error no se propague correctamente porque, en ciertas capas, ocurra que

$$\nabla L \simeq 0$$

Esto produce un entrenamiento muy lento o incluso nulo en ciertos casos.

Representation Bottleneck

Si tenemos un modelo secuencial, el input de una capa será el output de la anterior. Por tanto, si una capa "destruye" parte de la información las capas sucesivas se verán afectadas.

Cuando la red tiene muchas capas más probabilidades habrá de que se produzca esta destrucción y más grande será la acumulación de errores.

Vanishing Gradient Problem

Para entrenar una NN debemos propagar el error por todas las capas de la red utilizando el ∇L . Si la red tiene muchas capas puede ser que el error no se propague correctamente porque, en ciertas capas, ocurra que

$$\nabla L \approx 0$$

Esto produce un entrenamiento muy lento o incluso nulo en ciertos casos.

Representation Bottleneck

Si tenemos un modelo secuencial, el input de una capa será el output de la anterior. Por tanto, si una capa "destruye" parte de la información las capas sucesivas se verán afectadas.

Cuando la red tiene muchas capas más probabilidades habrá de que se produzca esta destrucción y más grande será la acumulación de errores.

Vanishing Gradient Problem

Para entrenar una NN debemos propagar el error por todas las capas de la red utilizando el ∇L . Si la red tiene muchas capas puede ser que el error no se propague correctamente porque, en ciertas capas, ocurra que

$$\nabla L \simeq 0$$

Esto produce un entrenamiento muy lento o incluso nulo en ciertos casos.

Representation Bottleneck

Si tenemos un modelo secuencial, el input de una capa será el output de la anterior. Por tanto, si una capa "destruye" parte de la información las capas sucesivas se verán afectadas.

Cuando la red tiene muchas capas más probabilidades habrá de que se produzca esta destrucción y más grande será la acumulación de errores.

Vanishing Gradient Problem

Para entrenar una NN debemos propagar el error por todas las capas de la red utilizando el ∇L . Si la red tiene muchas capas puede ser que el error no se propague correctamente porque, en ciertas capas, ocurra que

$$\nabla L \simeq 0$$

Esto produce un entrenamiento muy lento o incluso nulo en ciertos casos.

Representation Bottleneck

Si tenemos un modelo secuencial, el input de una capa será el output de la anterior. Por tanto, si una capa "destruye" parte de la información las capas sucesivas se verán afectadas.

Cuando la red tiene muchas capas más probabilidades habrá de que se produzca esta destrucción y más grande será la acumulación de errores.

ResNet

Arquitectura ResNet

Una ResNet está compuesta por una sucesión de módulos con skip connections. En general esta arquitectura incorpora también BatchNormalization y funciones de activación ReLu. Estos saltos reintroducen la información en capas posteriores haciendo que "virtualmente" la red tenga menos capas pero con una capacidad de aprendizaje similar.

Esto produce una mitigación de los dos problemas vistos anteriormente. Es recomendable usar este tipo de módulos en redes de más de 10 capas.

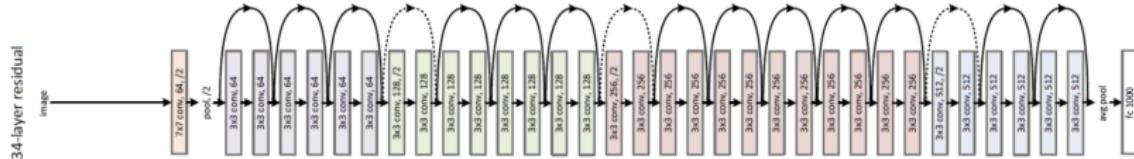


Figure: Arquitectura ResNet

ResNet

Arquitectura ResNet

Una ResNet está compuesta por una sucesión de módulos con skip connections. En general esta arquitectura incorpora también BatchNormalization y funciones de activación ReLu. Estos saltos reintroducen la información en capas posteriores haciendo que "virtualmente" la red tenga menos capas pero con una capacidad de aprendizaje similar.

Esto produce una mitigación de los dos problemas vistos anteriormente. Es recomendable usar este tipo de módulos en redes de más de 10 capas.

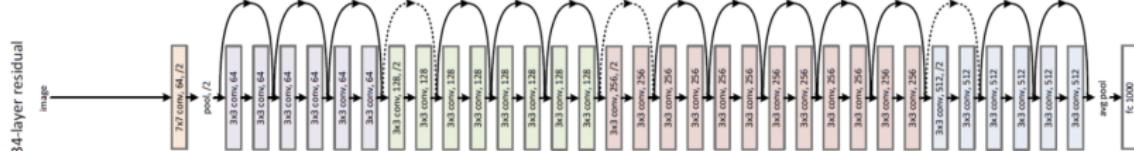


Figure: Arquitectura ResNet

Arquitectura ResNet

Si escribimos las ecuaciones de cada capa podemos ver cómo las modificaciones se hacen poco a poco introduciendo residuos.

ResNet: $x_0 \rightarrow x_1 = x_0 + f_0(x_0) \rightarrow x_2 = x_0 + f_0(x_0) + f_1(x_1) \rightarrow \dots$

En cambio, si lo comparamos con una red secuencial estándar vemos la dificultad de la propagación del gradiente, así como la dependencia del modelo a que todas las capas funcionen correctamente.

Secuencial: $x_0 \rightarrow x_1 = f_0(x_0) \rightarrow x_2 = f_1(f_0(x_0)) \rightarrow \dots$

Arquitectura ResNet

Si escribimos las ecuaciones de cada capa podemos ver cómo las modificaciones se hacen poco a poco introduciendo residuos.

ResNet: $x_0 \rightarrow x_1 = x_0 + f_0(x_0) \rightarrow x_2 = x_0 + f_0(x_0) + f_1(x_1) \rightarrow \dots$

En cambio, si lo comparamos con una red secuencial estándar vemos la dificultad de la propagación del gradiente, así como la dependencia del modelo a que todas las capas funcionen correctamente.

Secuencial: $x_0 \rightarrow x_1 = f_0(x_0) \rightarrow x_2 = f_1(f_0(x_0)) \rightarrow \dots$

Índice

1 Optimización del aprendizaje

- Técnicas de regularización
- Técnicas de optimización

2 Desarrollo de aplicaciones

- Análisis del problema
- Preparación del dataset
- Modelado y entrenamiento
- Fine Tuning y Evaluación
- Transfer Learning

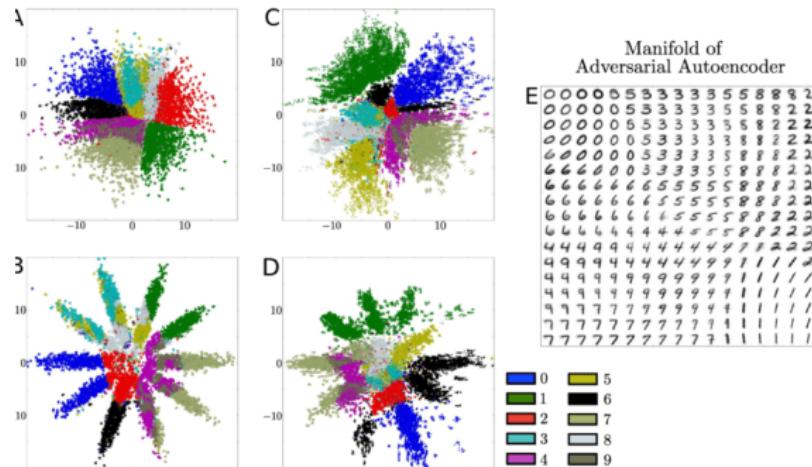
3 Arquitecturas específicas

- Entrenamientos Personalizados
- ResNet
- Autoencoders**
- U-Net
- GANs
- Super Resolution

Autoencoders

Autoencoders

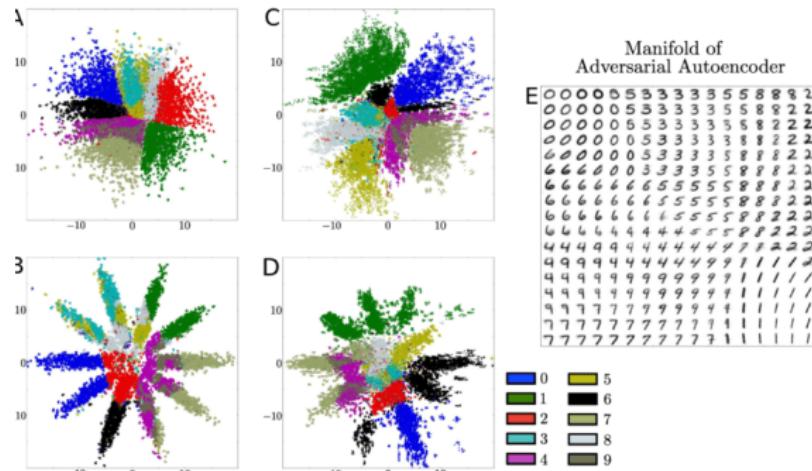
Un autoencoder sirve para encontrar un espacio representativo o espacio latente de nuestros datos. En principio este espacio tiene una dimensión menor de la original y nos aporta ciertas características de nuestro dataset. Con ello podemos obtener modelos más eficientes al poder trabajar con un dataset más pequeño, un modelo más pequeño y más rápido. Ya que va a ser mucho más eficiente trabajar sobre el espacio latente.



Autoencoders

Autoencoders

Un autoencoder sirve para encontrar un espacio representativo o espacio latente de nuestros datos. En principio este espacio tiene una dimensión menor de la original y nos aporta ciertas características de nuestro dataset. Con ello podemos obtener modelos más eficientes al poder trabajar con un dataset más pequeño, un modelo más pequeño y más rápido. Ya que va a ser mucho más eficiente trabajar sobre el espacio latente.



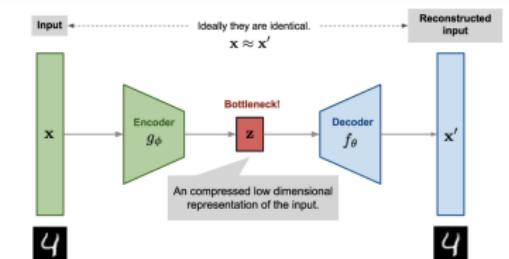
Autoencoders

Estructura de los Autoencoders

Los autoencoders constan de dos partes:

- ① Codificador: reduce la dimensión de la entrada x generando la proyección z de x en el espacio latente.
- ② Decodificador: dada una proyección z del espacio latente intenta recuperar la preimagen x sin pérdida de información.

La idea es proyectar y calcular la preimagen del punto x , reduciendo la dimensión de los datos entre medias. Con ello forzamos a que el modelo encuentre un espacio representativo donde, a partir de una proyección, podamos obtener x sin pérdida de información.



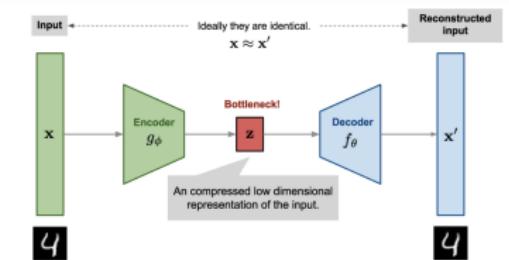
Autoencoders

Estructura de los Autoencoders

Los autoencoders constan de dos partes:

- ① Codificador: reduce la dimensión de la entrada x generando la proyección z de x en el espacio latente.
- ② Decodificador: dada una proyección z del espacio latente intenta recuperar la preimagen x sin pérdida de información.

La idea es proyectar y calcular la preimagen del punto x , reduciendo la dimensión de los datos entre medias. Con ello forzamos a que el modelo encuentre un espacio representativo donde, a partir de una proyección, podamos obtener x sin pérdida de información.



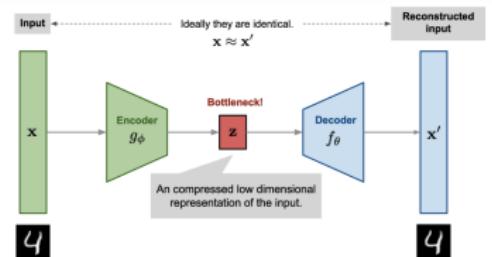
Autoencoders

Estructura de los Autoencoders

Los autoencoders constan de dos partes:

- ① Codificador: reduce la dimensión de la entrada x generando la proyección z de x en el espacio latente.
- ② Decodificador: dada una proyección z del espacio latente intenta recuperar la preimagen x sin pérdida de información.

La idea es proyectar y calcular la preimagen del punto x , reduciendo la dimensión de los datos entre medias. Con ello forzamos a que el modelo encuentre un espacio representativo donde, a partir de una proyección, podamos obtener x sin pérdida de información.



Autoencoders

Beneficios de los autoencoders

Gracias a los autoencoders podemos representar datos complejos en dos dimensiones para su visualización. Lo bueno de esta reducción de dimensionalidad es que las dependencias encontradas pueden ser complejas (vs PCA) y manejar una gran cantidad de datos.

Podemos usar los autoencoders con la técnica de transfer learning para entrenar modelos sobre dataset pequeños, ya que podemos ver el encoder como un extractor de features.

También podemos utilizar los autoencoders para eliminar ruido de nuestros datos, especialmente en imágenes.

Autoencoders

Beneficios de los autoencoders

Gracias a los autoencoders podemos representar datos complejos en dos dimensiones para su visualización. Lo bueno de esta reducción de dimensionalidad es que las dependencias encontradas pueden ser complejas (vs PCA) y manejar una gran cantidad de datos.

Podemos usar los autoencoders con la técnica de transfer learning para entrenar modelos sobre dataset pequeños, ya que podemos ver el encoder como un extractor de features.

También podemos utilizar los autoencoders para eliminar ruido de nuestros datos, especialmente en imágenes.

Autoencoders

Beneficios de los autoencoders

Gracias a los autoencoders podemos representar datos complejos en dos dimensiones para su visualización. Lo bueno de esta reducción de dimensionalidad es que las dependencias encontradas pueden ser complejas (vs PCA) y manejar una gran cantidad de datos.

Podemos usar los autoencoders con la técnica de transfer learning para entrenar modelos sobre dataset pequeños, ya que podemos ver el encoder como un extractor de features.

También podemos utilizar los autoencoders para eliminar ruido de nuestros datos, especialmente en imágenes.

Autoencoders + Transfer Learning

Si obtener etiquetas para nuestro dataset es costoso, podemos entrenar un autoencoder sobre nuestros datos sin etiquetas. Más tarde construiríamos un modelo que fuese la concatenación de nuestro encoder, con las capas congeladas, y una NN pequeña.

Finalmente entrenamos dicho modelo con las etiquetas disponibles. Con ello obtenemos un modelo que trabaja sobre features representativas de nuestros datos y que no requiere un gran dataset, ya que el modelo que realmente estamos entrenando es pequeño.

Autoencoders + Transfer Learning

Si obtener etiquetas para nuestro dataset es costoso, podemos entrenar un autoencoder sobre nuestros datos sin etiquetas. Más tarde construiríamos un modelo que fuese la concatenación de nuestro encoder, con las capas congeladas, y una NN pequeña.

Finalmente entrenamos dicho modelo con las etiquetas disponibles. Con ello obtenemos un modelo que trabaja sobre features representativas de nuestros datos y que no requiere un gran dataset, ya que el modelo que realmente estamos entrenando es pequeño.

Autoencoders

Eliminación de ruido

Podemos crear un autoencoder que elimine el ruido de los datos. La idea es agregar una capa al principio del autoencoder para que añada ruido a los datos y entrenar el autoencoder para que recupere el dato inicial sin ruido. Para eliminar el ruido de los datos simplemente tendríamos que aplicar el autoencoder, sin la primera capa, a los datos.

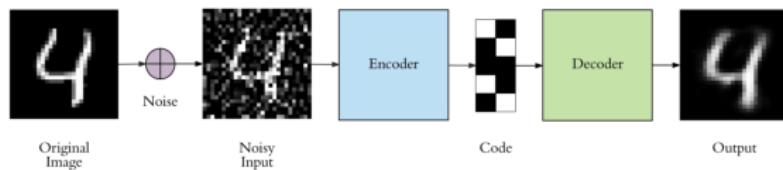


Figure: *Eliminación de ruido con Autoencoders*

Autoencoders

Eliminación de ruido

Podemos crear un autoencoder que elimine el ruido de los datos. La idea es agregar una capa al principio del autoencoder para que añada ruido a los datos y entrenar el autoencoder para que recupere el dato inicial sin ruido. Para eliminar el ruido de los datos simplemente tendríamos que aplicar el autoencoder, sin la primera capa, a los datos.

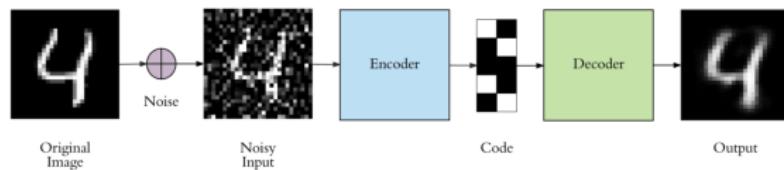


Figure: *Eliminación de ruido con Autoencoders*

Índice

1 Optimización del aprendizaje

- Técnicas de regularización
- Técnicas de optimización

2 Desarrollo de aplicaciones

- Análisis del problema
- Preparación del dataset
- Modelado y entrenamiento
- Fine Tuning y Evaluación
- Transfer Learning

3 Arquitecturas específicas

- Entrenamientos Personalizados
- ResNet
- Autoencoders
- U-Net**
- GANs
- Super Resolution

Problemas de Segmentación

La segmentación es una clasificación pixel a pixel de la imagen.

Para resolver un problema de segmentación necesitamos un modelo cuyos input y outputs tengan dimensiones $n \times n \times n_canales$ y $n \times n \times n_clases$ respectivamente. Dicho modelo lo entrenaríamos como un modelo de clasificación, usando como loss la media de las losses en cada pixel.

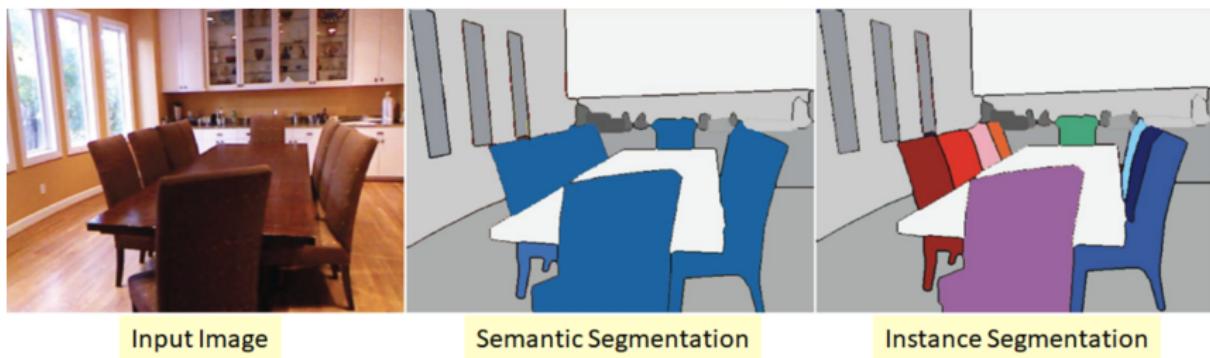


Figure: Semantic Segmentation vs Instance Segmentation

U-Net

Problemas de Segmentación

La segmentación es una clasificación pixel a pixel de la imagen. Para resolver un problema de segmentación necesitamos un modelo cuyos input y outputs tengan dimensiones $n \times n \times n_canales$ y $n \times n \times n_clases$ respectivamente. Dicho modelo lo entrenaríamos como un modelo de clasificación, usando como loss la media de las losses en cada pixel.

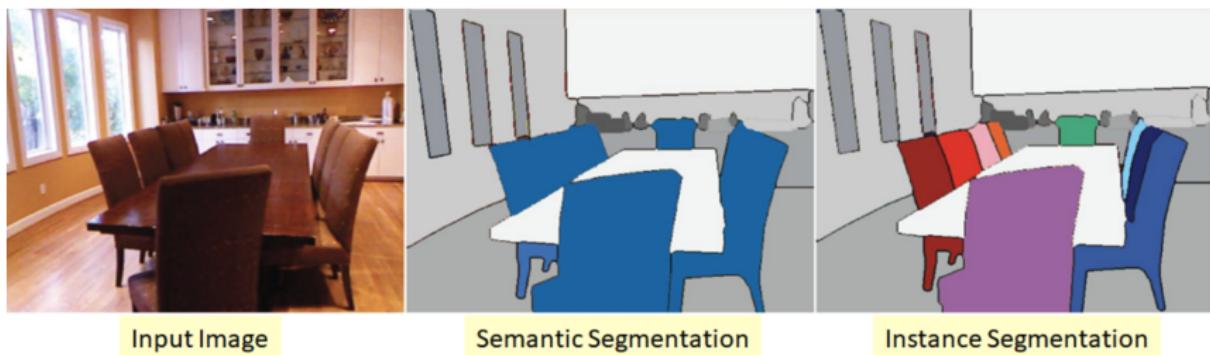


Figure: Semantic Segmentation vs Instance Segmentation

Problemas con modelos secuenciales convolucionales

Podríamos generar una red secuencial convolucional para resolver problemas de segmentación.

Usando una arquitectura secuencial, la clasificación por pixel se hará utilizando features locales de bajo nivel por lo que el modelo no tendrá una noción global de los patrones presentes en la imagen. Esto es un problema ya que muchas veces la clasificación de un pixel depende de otros pixeles alejados y de los patrones complejos de la propia imagen.

Por otra parte, si usásemos una arquitectura de encoder/decoder, el modelo sí tendría esta información, pero en cambio perdería mucha información de bajo nivel y, por tanto, la máscara no sería detallada.

Problemas con modelos secuenciales convolucionales

Podríamos generar una red secuencial convolucional para resolver problemas de segmentación.

Usando una arquitectura secuencial, la clasificación por pixel se hará utilizando features locales de bajo nivel por lo que el modelo no tendrá una noción global de los patrones presentes en la imagen. Esto es un problema ya que muchas veces la clasificación de un pixel depende de otros pixeles alejados y de los patrones complejos de la propia imagen.

Por otra parte, si usásemos una arquitectura de encoder/decoder, el modelo sí tendría esta información, pero en cambio perdería mucha información de bajo nivel y, por tanto, la máscara no sería detallada.

Problemas con modelos secuenciales convolucionales

Podríamos generar una red secuencial convolucional para resolver problemas de segmentación.

Usando una arquitectura secuencial, la clasificación por pixel se hará utilizando features locales de bajo nivel por lo que el modelo no tendrá una noción global de los patrones presentes en la imagen. Esto es un problema ya que muchas veces la clasificación de un pixel depende de otros pixeles alejados y de los patrones complejos de la propia imagen.

Por otra parte, si usásemos una arquitectura de encoder/decoder, el modelo sí tendría esta información, pero en cambio perdería mucha información de bajo nivel y, por tanto, la máscara no sería detallada.

U-Net

Arquitectura U-Net

Las redes U-Net tienen un tipo de arquitectura diseñado expresamente para resolver problemas de segmentación en imágenes. Son redes del tipo encoder/decoder, sin pérdida de información que, además, van acompañadas de módulos ResNet, funciones ReLu y BatchNormalization.

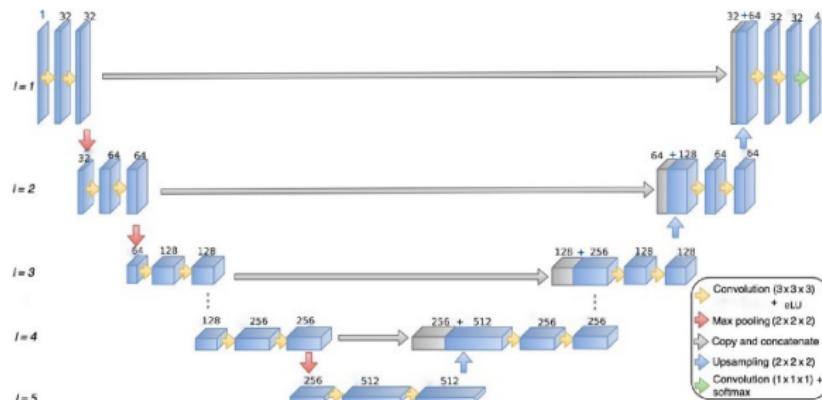


Figure: Arquitectura U-Net

Arquitectura U-Net

Los primeros niveles de la red analizan y procesan las features de bajo nivel, mientras que los últimos procesan features de alto nivel. Con ello conseguimos un análisis de la imagen tanto global como detallada.

Además con las skip connections conseguimos no perder información en ninguna parte de la red.

Como procesamos la imagen a distintas escalas conseguimos que la segmentación sea muy detallada, pudiendo detectar objetos de distintos tamaños y topologías.

Arquitectura U-Net

Los primeros niveles de la red analizan y procesan las features de bajo nivel, mientras que los últimos procesan features de alto nivel. Con ello conseguimos un análisis de la imagen tanto global como detallada. Además con las skip connections conseguimos no perder información en ninguna parte de la red.

Como procesamos la imagen a distintas escalas conseguimos que la segmentación sea muy detallada, pudiendo detectar objetos de distintos tamaños y topologías.

Arquitectura U-Net

Los primeros niveles de la red analizan y procesan las features de bajo nivel, mientras que los últimos procesan features de alto nivel. Con ello conseguimos un análisis de la imagen tanto global como detallada. Además con las skip connections conseguimos no perder información en ninguna parte de la red.

Como procesamos la imagen a distintas escalas conseguimos que la segmentación sea muy detallada, pudiendo detectar objetos de distintos tamaños y topologías.

Índice

1 Optimización del aprendizaje

- Técnicas de regularización
- Técnicas de optimización

2 Desarrollo de aplicaciones

- Análisis del problema
- Preparación del dataset
- Modelado y entrenamiento
- Fine Tuning y Evaluación
- Transfer Learning

3 Arquitecturas específicas

- Entrenamientos Personalizados
- ResNet
- Autoencoders
- U-Net
- GANs**
- Super Resolution

GANs

GANs

Las GANs (Generative Adversarial Networks) se introdujeron en 2014 para la generación sintética y realista de imágenes. Hoy en día se usan para generación sintética de cualquier tipo de dato. La idea es hacer que los datos generados sean "indistinguibles", desde el punto de vista estadístico, de los datos reales.

Intuitivamente las GANs pueden verse como un sistema que, de manera iterativa, compite contra sí mismo para obtener mejores resultados.

GANs

GANs

Las GANs (Generative Adversarial Networks) se introdujeron en 2014 para la generación sintética y realista de imágenes. Hoy en día se usan para generación sintética de cualquier tipo de dato. La idea es hacer que los datos generados sean "indistinguibles", desde el punto de vista estadístico, de los datos reales.

Intuitivamente las GANs pueden verse como un sistema que, de manera iterativa, compite contra sí mismo para obtener mejores resultados.

GANs

Generative Adversarial Network

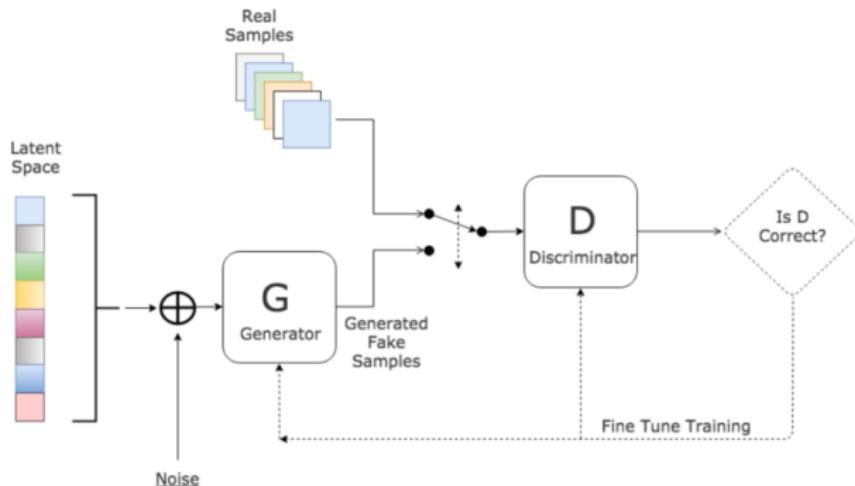


Figure: Arquitectura de las GANs

Funcionamiento

El generador producirá los datos sintéticos y se entrenará para hacer que el discriminador falle. En cambio el discriminador se entrenará para distinguir entre datos reales y datos sintéticos.

Llegará un momento en el que el sistema se estabilice, donde los datos creados por el generador tengan unas "características" muy similares a los datos reales y el discriminador sea incapaz de mejorar para distinguirlos.

Es un proceso lento e iterativo de competición entre ambas redes en el que no se busca un mínimo, sino un punto de equilibrio. Si una de las dos redes es mucho mejor que la otra el sistema no funcionará correctamente.

Funcionamiento

El generador producirá los datos sintéticos y se entrenará para hacer que el discriminador falle. En cambio el discriminador se entrenará para distinguir entre datos reales y datos sintéticos.

Llegará un momento en el que el sistema se estabilice, donde los datos creados por el generador tengan unas "características" muy similares a los datos reales y el discriminador sea incapaz de mejorar para distinguirlos.

Es un proceso lento e iterativo de competición entre ambas redes en el que no se busca un mínimo, sino un punto de equilibrio. Si una de las dos redes es mucho mejor que la otra el sistema no funcionará correctamente.

Funcionamiento

El generador producirá los datos sintéticos y se entrenará para hacer que el discriminador falle. En cambio el discriminador se entrenará para distinguir entre datos reales y datos sintéticos.

Llegará un momento en el que el sistema se estabilice, donde los datos creados por el generador tengan unas "características" muy similares a los datos reales y el discriminador sea incapaz de mejorar para distinguirlos. Es un proceso lento e iterativo de competición entre ambas redes en el que no se busca un mínimo, sino un punto de equilibrio. Si una de las dos redes es mucho mejor que la otra el sistema no funcionará correctamente.

Entrenamiento

El discriminador $D(y)$ lo entrenaremos como un modelo de clasificación entre imágenes reales y fakes.

Para entrenar el generador hay que tener un poco más de cuidado. Como lo que queremos es que el generador engañe al discriminador, tenemos que modificar los pesos de G para que $z = D(G(x))$ nos dé la etiqueta "real". Por tanto, optimizaremos $z = D(G(x))$ únicamente respecto de los pesos de G , usando

$$\nabla_{\theta_G} D \circ G$$

Entrenamiento

El discriminador $D(y)$ lo entrenaremos como un modelo de clasificación entre imágenes reales y fakes.

Para entrenar el generador hay que tener un poco más de cuidado. Como lo que queremos es que el generador engañe al discriminador, tenemos que modificar los pesos de G para que $z = D(G(x))$ nos dé la etiqueta "real". Por tanto, optimizaremos $z = D(G(x))$ únicamente respecto de los pesos de G , usando

$$\nabla_{\theta_G} D \circ G$$

Entrenamiento

Para entrenar el generador tenemos dos opciones:

- ① Entrenar el modelo $D(G(x))$, congelando las capas asociadas a D .
- ② Utilizar un entrenamiento personalizado para obtener $\nabla_{\theta_G} D \circ G$.

Es un entrenamiento lento el cual requiere muchos ajustes de los parámetros de configuración del sistema.

Entrenamiento

Para entrenar el generador tenemos dos opciones:

- ① Entrenar el modelo $D(G(x))$, congelando las capas asociadas a D .
- ② Utilizar un entrenamiento personalizado para obtener $\nabla_{\theta_G} D \circ G$.

Es un entrenamiento lento el cual requiere muchos ajustes de los parámetros de configuración del sistema.

Índice

1 Optimización del aprendizaje

- Técnicas de regularización
- Técnicas de optimización

2 Desarrollo de aplicaciones

- Análisis del problema
- Preparación del dataset
- Modelado y entrenamiento
- Fine Tuning y Evaluación
- Transfer Learning

3 Arquitecturas específicas

- Entrenamientos Personalizados
- ResNet
- Autoencoders
- U-Net
- GANs
- Super Resolution

Super Resolution

Super Resolution

La Super Resolution es una técnica que consigue aumentar la resolución y la calidad de una imagen. Hay algoritmos clásicos como por ejemplo el bicúbico que son capaces de aumentar la resolución pero con una mala calidad.

Con esta técnica conseguimos que la imagen "parezca" real después de aumentar su resolución.

La arquitectura utilizada para esta técnica se basa en las GANs.

Super Resolution

Super Resolution

La Super Resolution es una técnica que consigue aumentar la resolución y la calidad de una imagen. Hay algoritmos clásicos como por ejemplo el bicúbico que son capaces de aumentar la resolución pero con una mala calidad.

Con esta técnica conseguimos que la imagen "parezca" real después de aumentar su resolución.

La arquitectura utilizada para esta técnica se basa en las GANs.

Super Resolution

Super Resolution

La Super Resolution es una técnica que consigue aumentar la resolución y la calidad de una imagen. Hay algoritmos clásicos como por ejemplo el bicúbico que son capaces de aumentar la resolución pero con una mala calidad.

Con esta técnica conseguimos que la imagen "parezca" real después de aumentar su resolución.

La arquitectura utilizada para esta técnica se basa en las GANs.

Super Resolution

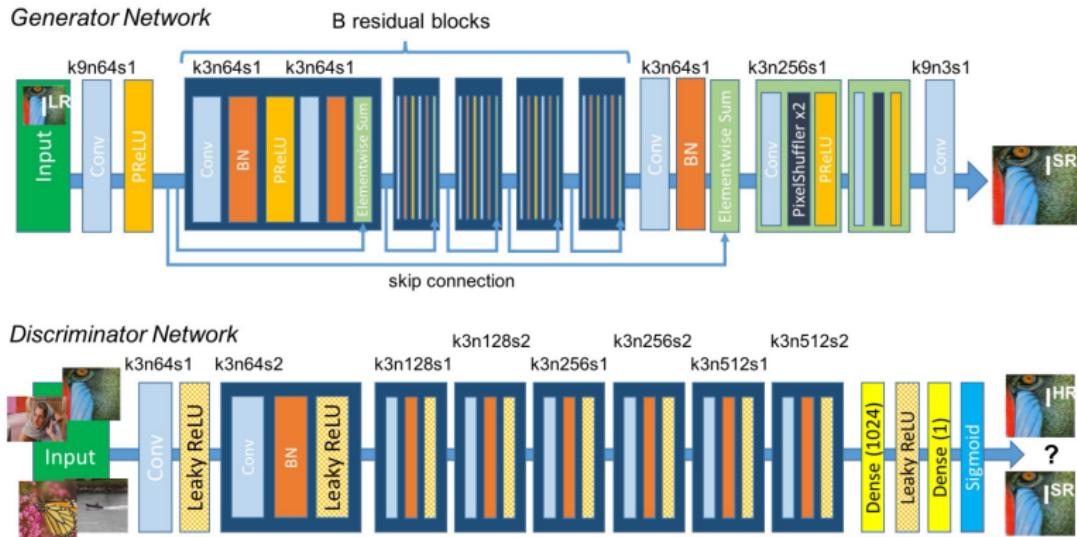


Figure: Arquitectura de Super Resolution

Super Resolution

Arquitectura

El sistema consta de dos partes:

- ① El generador: Se encarga de transformar la imagen de low resolution en high resolution y para ello utiliza capas deconvolucionales o de super sampling. Como es habitual en NN profundas, se utilizarán módulos ResNet para procesar la imagen.
- ② El discriminador: Se encargará de distinguir entre imágenes de alta resolución reales o generadas sintéticamente por el generador. Su función es ayudar al generador a producir imágenes con mejor calidad.

Arquitectura

El sistema consta de dos partes:

- ① El generador: Se encarga de transformar la imagen de low resolution en high resolution y para ello utiliza capas deconvolucionales o de super sampling. Como es habitual en NN profundas, se utilizarán módulos ResNet para procesar la imagen.
- ② El discriminador: Se encargará de distinguir entre imágenes de alta resolución reales o generadas sintéticamente por el generador. Su función es ayudar al generador a producir imágenes con mejor calidad.

Arquitectura

El sistema consta de dos partes:

- ① El generador: Se encarga de transformar la imagen de low resolution en high resolution y para ello utiliza capas deconvolucionales o de super sampling. Como es habitual en NN profundas, se utilizarán módulos ResNet para procesar la imagen.
- ② El discriminador: Se encargará de distinguir entre imágenes de alta resolución reales o generadas sintéticamente por el generador. Su función es ayudar al generador a producir imágenes con mejor calidad.

Super Resolution

Funcionamiento

Lo interesante de este sistema es su entrenamiento. Al igual que con las GANs, el objetivo es llegar a un punto de equilibrio entre ambos modelos.

Lo más difícil del sistema es definir la función loss para el generador, ya que el discriminador es un modelo de clasificación. Si usásemos solo

$$\text{loss}(y, \tilde{y}) = \text{MSE}(y, \tilde{y})$$

para entrenar el generador, las imágenes no serían realistas o naturales, ya que estamos trabajando a nivel de píxel. ¿Cómo podemos discernir si dos imágenes son similares?

Aquí es donde aparece el concepto de Perceptual Loss, una loss global que nos ayuda a comparar dos imágenes.

Super Resolution

Funcionamiento

Lo interesante de este sistema es su entrenamiento. Al igual que con las GANs, el objetivo es llegar a un punto de equilibrio entre ambos modelos. Lo más difícil del sistema es definir la función loss para el generador, ya que el discriminador es un modelo de clasificación. Si usásemos solo

$$\text{loss}(y, \tilde{y}) = \text{MSE}(y, \tilde{y})$$

para entrenar el generador, las imágenes no serían realistas o naturales, ya que estamos trabajando a nivel de píxel. ¿Cómo podemos discernir si dos imágenes son similares?

Aquí es donde aparece el concepto de Perceptual Loss, una loss global que nos ayuda a comparar dos imágenes.

Super Resolution

Funcionamiento

Lo interesante de este sistema es su entrenamiento. Al igual que con las GANs, el objetivo es llegar a un punto de equilibrio entre ambos modelos. Lo más difícil del sistema es definir la función loss para el generador, ya que el discriminador es un modelo de clasificación. Si usásemos solo

$$\text{loss}(y, \tilde{y}) = \text{MSE}(y, \tilde{y})$$

para entrenar el generador, las imágenes no serían realistas o naturales, ya que estamos trabajando a nivel de píxel. ¿Cómo podemos discernir si dos imágenes son similares?

Aquí es donde aparece el concepto de Perceptual Loss, una loss global que nos ayuda a comparar dos imágenes.

Super Resolution

Perceptual Loss

El Perceptual Loss está diseñada para medir la similitud entre dos imágenes. Consta de tres partes y cada una de ellas mide una característica de la imagen. El loss del generador será una suma ponderada de:

- ① MSE loss: mide el error a nivel de píxel.

$$MSE(y, \tilde{y})$$

- ② Content loss: Mide la diferencia entre las features de alto nivel de la imagen. Para ello usa los outputs intermedios de una red preentrenada, como la VGG.

$$MSE(VGG_m(y), VGG_m(\tilde{y}))$$

- ③ Adversarial loss: Mide el realismo de la imagen usando el error del discriminador.

$$\log[1 - D(G(x))]$$

Super Resolution

Perceptual Loss

El Perceptual Loss está diseñada para medir la similitud entre dos imágenes. Consta de tres partes y cada una de ellas mide una característica de la imagen. El loss del generador será una suma ponderada de:

- ① MSE loss: mide el error a nivel de píxel.

$$MSE(y, \tilde{y})$$

- ② Content loss: Mide la diferencia entre las features de alto nivel de la imagen. Para ello usa los outputs intermedios de una red preentrenada, como la VGG.

$$MSE(VGG_m(y), VGG_m(\tilde{y}))$$

- ③ Adversarial loss: Mide el realismo de la imagen usando el error del discriminador.

$$\log[1 - D(G(x))]$$

Super Resolution

Perceptual Loss

El Perceptual Loss está diseñada para medir la similitud entre dos imágenes. Consta de tres partes y cada una de ellas mide una característica de la imagen. El loss del generador será una suma ponderada de:

- ① MSE loss: mide el error a nivel de píxel.

$$MSE(y, \tilde{y})$$

- ② Content loss: Mide la diferencia entre las features de alto nivel de la imagen. Para ello usa los outputs intermedios de una red preentrenada, como la VGG.

$$MSE(VGG_m(y), VGG_m(\tilde{y}))$$

- ③ Adversarial loss: Mide el realismo de la imagen usando el error del discriminador.

$$\log[1 - D(G(x))]$$

Super Resolution

Perceptual Loss

El Perceptual Loss está diseñada para medir la similitud entre dos imágenes. Consta de tres partes y cada una de ellas mide una característica de la imagen. El loss del generador será una suma ponderada de:

- ① MSE loss: mide el error a nivel de píxel.

$$MSE(y, \tilde{y})$$

- ② Content loss: Mide la diferencia entre las features de alto nivel de la imagen. Para ello usa los outputs intermedios de una red preentrenada, como la VGG.

$$MSE(VGG_m(y), VGG_m(\tilde{y}))$$

- ③ Adversarial loss: Mide el realismo de la imagen usando el error del discriminador.

$$\log[1 - D(G(x))]$$

Super Resolution



Figure: Ejemplo comparativo entre distintas losses

Super Resolution



Figure: HR vs LR vs SuperResolution vs Bicubic