Transfer Learning

Print to PDF

Contents

Es una técnica de aprendizaje automático en la que un modelo previamente entrenado en un conjunto de datos grande y general se utiliza para una tarea similar. Las características aprendidas por la red preentrenada pueden funcionar como un modelo genérico, lo que las hace útiles para resolver problemas relacionados.

El transfer learning aprovecha los conocimientos adquiridos por el modelo preentrenado, optimizando tiempo y recursos, además de mejorar el rendimiento, especialmente en tareas con conjuntos de datos pequeños.

Python proporciona varios modelos preentrenados a través del módulo `tf.keras.applications` en TensorFlow. Este módulo incluye arquitecturas populares de redes neuronales entrenadas en conjuntos de datos grandes, como ImageNet (con 1.4 millones de imágenes etiquetadas y 1000 clases diferentes). Algunos ejemplos son:

- VGG16 y VGG19
- ResNet (Residual Networks)
- Inception (GoogleNet)
- Xception
- MobileNet

📌 Documentación tf.keras.applications

✅ **Ventajas del Transfer Learning**

- Permite aprovechar modelos previamente entrenados en tareas similares o relacionadas.
- Ideal cuando se tienen pocos datos etiquetados en la nueva tarea.
- Reduce significativamente el tiempo de entrenamiento y la demanda computacional comparado con entrenar desde cero.
- Proporciona una buena línea base de rendimiento desde el inicio.
- Muy útil en áreas donde el etiquetado es costoso o escaso (por ejemplo, medicina, imágenes satelitales).

❌ **Desventajas del Transfer Learning**

- Si el dominio origen y destino son muy distintos, el modelo puede no transferir bien el conocimiento.
- Riesgo de sobreajuste (overfitting),especialmente si se hace fine-tuning con pocos datos.
- Muchos modelos preentrenados tienen millones de parámetros, lo que puede dificultar su uso en producción o en dispositivos con pocos recursos.

Métodos en Transfer Learning

Este método consiste en utilizar las representaciones aprendidas por un modelo preentrenado para extraer características de nuevas muestras.Posterior, estas características se procesan a través de un clasificador o regresor, que se entrena desde cero.

En particular, las redes convolucionales tienen dos partes principales:

- Base convolucional: Conformada por las capas convolucionales, encargadas de la extracción de patrones, como bordes, texturas y formas.
- Parte de decisión: Compuesta por capas densas que realizan la clasificación o regresión basada en las características extraídas.

En este método, la base convolucional se congela para evitar modificar sus pesos, ya que estas capas contienen características generales útiles que pueden ser reutilizadas en diversas tareas.

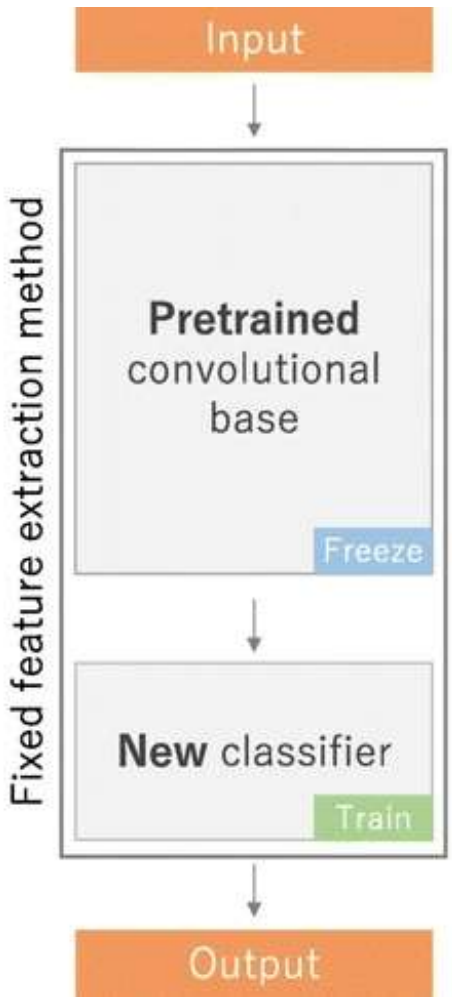

Fig.1. Imagen tomada de researchgate.net

Cargar el modelo preentrenado VGG16

Vamos a cargar la red VGG16 preentrenada en ImageNet, utilizando keras.applications.VGG16. Elegimos esta arquitectura porque es uno de los modelos con menos capas en comparación con otras redes más profundas como ResNet o EfficientNet.

Al cargarla, pasamos tres argumentos clave al constructor:

weights='imagenet'Indica que queremos cargar los pesos preentrenados en el dataset ImageNet,

include_top=`True` or `False` incluir o excluir el clasificador denso final del modelo

input_shape=(altura, ancho, canales) Define el tamaño de entrada de nuestras imágenes, por ejemplo (224, 224, 3) para imágenes RGB.

```
from tensorflow.keras.applications import VGG16

vgg16 = VGG16(weights='imagenet',
                  include_top=True,
                  input_shape=(224, 224, 3))
vgg16.summary()
```

```
Model: "vgg16"
```

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080

block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102,764,544
fc2 (Dense)	(None, 4096)	16,781,312
predictions (Dense)	(None, 1000)	4,097,000

Total params: 138,357,544 (527.79 MB)

Trainable params: 138,357,544 (527.79 MB)

Non-trainable params: 0 (0.00 B)

Cuando se carga el modelo VGG16 con `include_top=False`, lo que estamos haciendo es excluir la parte superior del modelo, es decir, todo lo que viene después de la última capa convolucional. Esto incluye la capa Flatten, que normalmente transforma los mapas de características bidimensionales en un vector unidimensional, y también las capas densas (Dense) que funcionan como clasificador final

```
if vgg16 != None:
    del vgg16

from tensorflow.keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(224, 224, 3))
conv_base.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808

block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0

Total params: 14,714,688 (56.13 MB)

Trainable params: 14,714,688 (56.13 MB)

Non-trainable params: 0 (0.00 B)

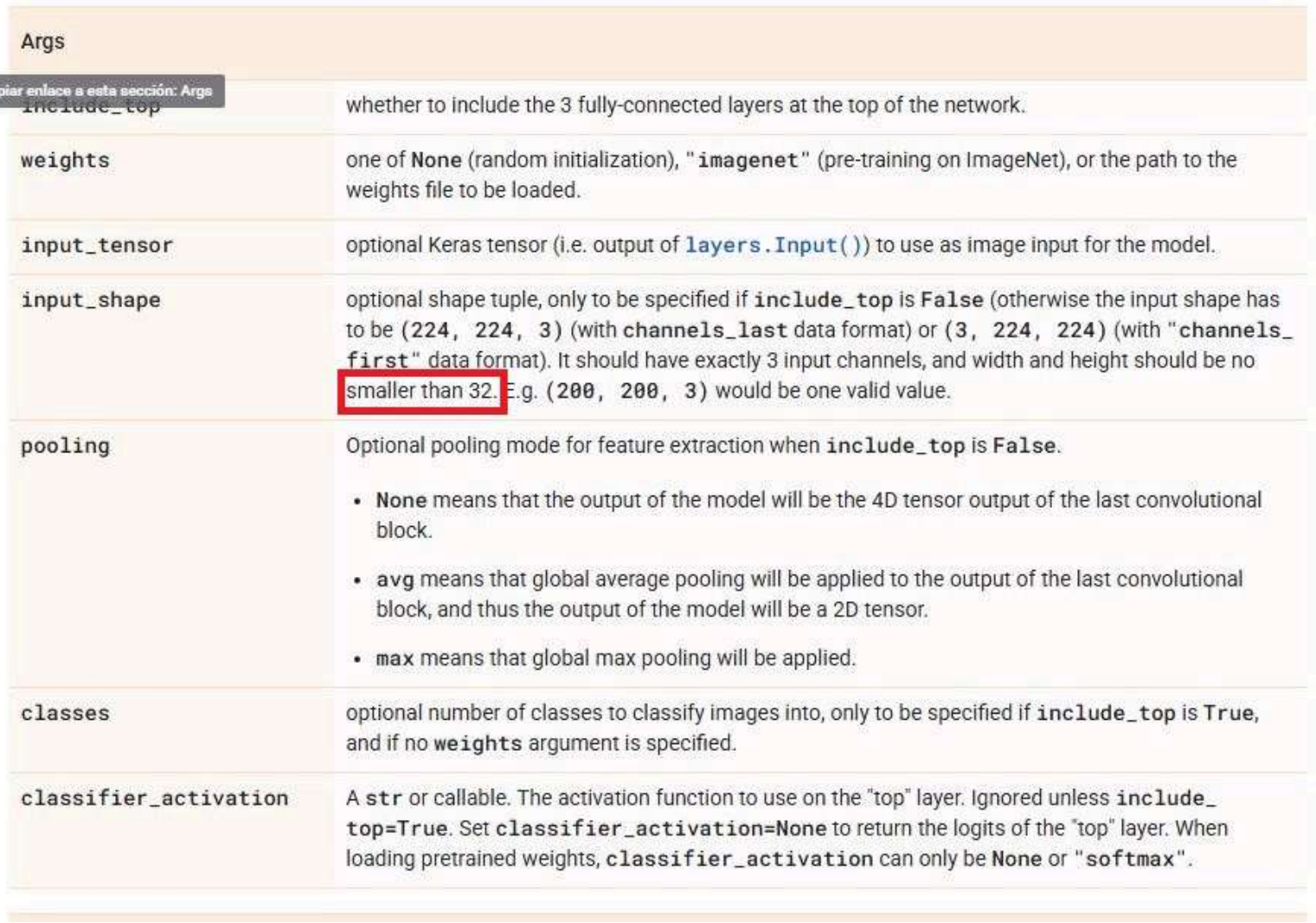

Fig.2. Imagen tomada de Tensorflow.org

Es importante revisar los argumentos de los modelos preentrenados porque tienen requisitos específicos de entrada. Por ejemplo, VGG16 fue entrenado con imágenes de 224x224 y 3 canales , pero puede usarse con imágenes de mínimo 32x32 porque al pasar por las capas convolucionales y de max pooling, el tamaño de la imagen se reduce, lo que podría hacerla inválida antes de llegar al final del modelo.

> **¿Por qué solo reutilizar la base convolucional?**
>
> - La base convolucional de una red extrae mapas de características que capturan patrones visuales generales como bordes, texturas, formas y estructuras. Estos mapas de características son útiles en muchas tareas visuales, incluso cuando cambian los objetos o contextos (por ejemplo, de autos a edificios o de fotos a imágenes satelitales).
>
> En cambio, las capas Fully Connected (FC) o densas al final del modelo aprenden representaciones muy específicas a las clases del conjunto de datos original.

Ahora, se usa la extracción de características de la base convolucional preentrenada y se conectamos un clasificador binario final sin modificar las capas convolucionales. Sin embargo, si simplemente la insertamos en nuestra arquitectura y la dejamos entrenar desde cero junto con las nuevas capas que agregamos, estamos permitiendo que sus parámetros se modifiquen a partir de nuestros propios datos. Esto podría parecer razonable, pero existe un riesgo significativo: nuestros datos son limitados y, en muchos casos, muy distintos a los del conjunto original ImageNet. Como resultado, el modelo podría comenzar a “olvidar” aquello que había aprendido, un fenómeno conocido como catastrophic forgetting.

Para evitar esto, recurrimos a una estrategia conocida como congelar la base convolucional, mediante `conv_base.trainable = False`

```python
from tensorflow.keras.applications import VGG16
from tensorflow.keras import layers, models


conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(32, 32, 3))

conv_base.trainable = False  # Congelar capas


model = models.Sequential()
model.add(layers.Input(shape=(32,32, 3)))
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.summary()
```

Model: "sequential_23"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 1, 1, 512)	14,714,688
flatten_20 (Flatten)	(None, 512)	0
dense_44 (Dense)	(None, 256)	131,328
dense_45 (Dense)	(None, 1)	257

Total params: 14,846,273 (56.63 MB)

Trainable params: 131,585 (514.00 KB)

Non-trainable params: 14,714,688 (56.13 MB)

Análisis de parámetros al combinar VGG16 con un clasificador

Cuando se utiliza VGG16 como extractor de características, su bloque convolucional contiene **14.714.688** parámetros entrenables. Sin embargo, al agregar un clasificador denso y congelar las capas de VGG16 (conv_base.trainable = False), los parámetros totales del modelo aumentan a **14.846.273**, pero solo **131.585** pertenecen al clasificador y son entrenables

Ajuste fino (Fine-Tuning)

Consiste en descongelar algunas de las capas superiores de la base convolucional para permitir que sus pesos sean ajustados durante el entrenamiento, adaptándolos de manera más específica al nuevo problema.

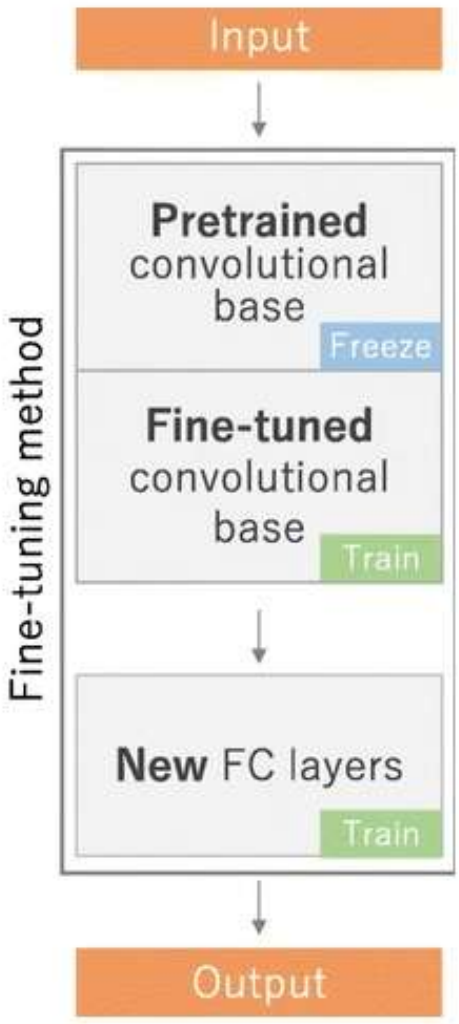

Fig.3. Imagen tomada de researchgate.net

Ahora, en lugar de mantener congelada toda la base convolucional de VGG16, se descongela parcialmente a partir de una capa específica (por ejemplo, block5_conv1). Esto permite realizar ajuste fino, es decir, entrenar solo las capas más profundas de la red para que adapten sus representaciones a la nueva tarea, mientras que las capas anteriores conservan el conocimiento general aprendido en ImageNet.

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0

Fig.4. block5_conv1

```
# Permitimos que se puedan entrenar las capas
conv_base.trainable = True

set_trainable = False
for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

Este bloque de código activa el entrenamiento solo para las capas más profundas de la red VGG16. Primero, se permite que toda la base convolucional sea modificable, pero luego, mediante un recorrido capa por capa, se congelan las primeras capas

y se activa el entrenamiento únicamente a partir de la capa `block5_conv1`.

```
modelf = models.Sequential()
modelf.add(layers.Input(shape=(32,32, 3)))
modelf.add(conv_base)
modelf.add(layers.Flatten())
modelf.add(layers.Dense(256, activation='relu'))
modelf.add(layers.Dense(1, activation='sigmoid'))

modelf.summary()
```

Model: "sequential_24"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 1, 1, 512)	14,714,688
flatten_21 (Flatten)	(None, 512)	0
dense_46 (Dense)	(None, 256)	131,328
dense_47 (Dense)	(None, 1)	257

Total params: 14,846,273 (56.63 MB)

Trainable params: 7,211,009 (27.51 MB)

Non-trainable params: 7,635,264 (29.13 MB)

La figura 5. compara tres estrategias de entrenamiento en modelos preentrenados: entrenamiento completo, congelamiento parcial y fine-tuning. Cuando todos los parámetros son entrenables, el modelo tiene mayor capacidad de adaptación, pero también mayor riesgo de sobreajuste y demanda computacional. Al congelar el bloque convolucional y entrenar solo el clasificador, se reduce drásticamente la cantidad de parámetros ajustables, lo que mejora la eficiencia y evita el sobreajuste, aunque limita la capacidad de adaptación del modelo. Finalmente, el fine-tuning parcial ofrece un equilibrio entre ambas estrategias, permitiendo ajustar parte de la red convolucional para adaptarse mejor a nuevas tareas sin requerir un entrenamiento completo desde cero

Fig.5.Comparación de parámetros entrenables y no entrenables en diferentes configuraciones del modelo

Conclusiones

- Transfer Learning permite reutilizar modelos preentrenados en tareas similares, adaptándolos a nuevas tareas con menos datos y en menor tiempo. Esta tecnica es útil cuando se dispone de conjuntos de datos limitados o cuando entrenar un modelo desde cero sería costoso en términos de tiempo y recursos.
- En el contexto de imágenes satelitales, Transfer Learning facilita la clasificación y análisis de grandes volúmenes de datos. Modelos preentrenados como VGG16, ResNet o Vision Transformer pueden adaptarse para identificar características específicas en imágenes satelitales.

Referencias

- IBM. (s.f.). Transfer learning. IBM Think. https://www.ibm.com/think/topics/transfer-learning
- Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., & He, Q. (2021). A comprehensive survey on transfer learning. Proceedings of the IEEE, 109(1), 43–76. https://doi.org/10.1109/JPROC.2020.3004555