DEEP LEARNING con Python

François Chollet

ANAYA MULTIMEDIA

Índice de contenidos

Agradecimientos 5
Sobre el autor 6

Introducción 15

Sobre este libro...16
- Quién debería leer este libro 16
- Estructura del libro 17
- Requisitos de software/hardware 18

Código fuente 18
Foro del libro 19
Sobre la imagen de cubierta...19

Parte 1. Fundamentos del *deep learning* 25

Capítulo 1. ¿Qué es el *deep learning*? 23

1.1. Inteligencia Artificial, *machine learning* y *deep learning* 23
- 1.1.1. Inteligencia Artificial 24
- 1.1.2. *Machine learning*... 24
- 1.1.3. Aprender representaciones de datos.. 26
- 1.1.4. El "*deep*" en el *deep learning*.. 28
- 1.1.5. Entender cómo funciona el *deep learning* en tres figuras............... 29
- 1.1.6. ¿Qué ha conseguido el *deep learning* hasta ahora? 32

1.1.7. No crea en el bombo publicitario a corto plazo ... 32
1.1.8. La promesa de la IA ... 33
1.2. Antes del *deep learning*: una breve historia del *machine learning* ... 34
1.2.1. Modelo probabilístico ... 35
1.2.2. Primeras redes neuronales ... 35
1.2.3. Métodos *kernel* ... 36
1.2.4. Árboles de decisiones, *random forests* y potenciación de gradiente ... 37
1.2.5. Regreso a las redes neuronales ... 38
1.2.6. Qué diferencia al *deep learning* ... 39
1.2.7. El paisaje moderno del *machine learning* ... 40
1.3. ¿Por qué el *deep learning*? ¿Por qué ahora? ... 40
1.3.1. Hardware ... 41
1.3.2. Datos ... 42
1.3.3. Algoritmos ... 43
1.3.4. Una nueva oleada de inversión ... 43
1.3.5. La democratización del *deep learning* ... 44
1.3.6. ¿Durará? ... 44

Capítulo 2. Los bloques de construcción matemáticos de las redes neuronales 47

2.1. Un primer vistazo a una red neuronal ... 47
2.2. Representaciones de datos para redes neuronales ... 51
2.2.1. Escalares (tensores 0D) ... 51
2.2.2. Vectores (tensores 1D) ... 52
2.2.3. Matrices (tensores 2D) ... 52
2.2.4. Tensores 3D y tensores con más dimensiones ... 52
2.2.5. Atributos clave ... 53
2.2.6. Manipular tensores en Numpy ... 54
2.2.7. La noción de lotes de datos ... 55
2.2.8. Ejemplos de tensores de datos en el mundo real ... 55
2.2.9. Datos vectoriales ... 55
2.2.10. Datos de series temporales o de secuencia ... 56
2.2.11. Datos de imagen ... 57
2.2.12. Datos de vídeo ... 57
2.3. Los engranajes de las redes neuronales: operaciones con tensores ... 58
2.3.1. Operaciones elemento a elemento ... 58
2.3.2. *Broadcasting* ... 59
2.3.3. Producto tensorial ... 60

2.3.4. Cambiar la forma de tensores ... 62
2.3.5. Interpretación geométrica de las operaciones con tensores ... 63
2.3.6. Interpretación geométrica del *deep learning* ... 64
2.4. El motor de las redes neuronales: optimización basada en gradiente ... 65
2.4.1. ¿Qué es una derivada? ... 66
2.4.2. Derivada de una operación con tensores: el gradiente ... 67
2.4.3. Descenso de gradiente estocástico ... 68
2.4.4. Encadenar derivadas: el algoritmo de retropropagación ... 71
2.5. Volviendo al primer ejemplo ... 72

Capítulo 3. Iniciarse en las redes neuronales 75

3.1. Anatomía de una red neuronal ... 76
3.1.1. Capas: los bloques de construcción del *deep learning* ... 76
3.1.2. Modelos: redes de capas ... 77
3.1.3. Funciones de pérdida y optimizadores: claves para configurar el proceso de aprendizaje ... 78
3.2. Introducción a Keras ... 79
3.2.1. Keras, TensorFlow, Theano y CNTK ... 80
3.2.2. Desarrollo con Keras: resumen general ... 81
3.3. Configurar una estación de trabajo de *deep learning* ... 82
3.3.1. *Notebooks* de Jupyter: la forma preferida de ejecutar experimentos de *deep learning* ... 83
3.3.2. Poner Keras en marcha: dos opciones ... 83
3.3.3. Ejecutar trabajos de *deep learning* en la nube: pros y contras ... 84
3.3.4. ¿Cuál es la mejor GPU para el *deep learning*? ... 84
3.4. Clasificar críticas de películas: ejemplo de clasificación binaria ... 84
3.4.1. El conjunto de datos de IMDB ... 85
3.4.2. Preparar los datos ... 86
3.4.3. Crear la red ... 87
3.4.4. Validación de nuestro enfoque ... 90
3.4.5. Utilizar una red entrenada para generar predicciones con datos nuevos ... 93
3.4.6. Más experimentos ... 94
3.4.7. Resumen ... 94
3.5. Clasificar noticias: ejemplo de clasificación multiclase ... 94
3.5.1. El conjunto de datos de Reuters ... 95
3.5.2. Preparar los datos ... 96
3.5.3. Crear la red ... 96
3.5.4. Validar el enfoque ... 97

3.5.5. Generar predicciones en datos nuevos.. 100
3.5.6. Una forma diferente de manejar las etiquetas y la pérdida 100
3.5.7. La importancia de tener capas intermedias
lo bastante grandes.. 101
3.5.8. Más experimentos 101
3.5.9. Resumen..102
3.6. Predecir precios de casas: ejemplo de regresión..................................102
3.6.1. El conjunto de datos de los precios de las casas de Boston..........102
3.6.2. Preparar los datos 103
3.6.3. Crear la red 104
3.6.4. Validar nuestro enfoque utilizando validación
de K iteraciones.. 104
3.6.5. Resumen... 109

Capítulo 4. Fundamentos del *machine learning* *111*

4.1. Cuatro ramas de *machine learning*..111
4.1.1. Aprendizaje supervisado.. 112
4.1.2. Aprendizaje no supervisado.. 112
4.1.3. Aprendizaje autosupervisado 112
4.1.4. Aprendizaje por refuerzo 113
4.2. Evaluación de modelos de *machine learning* 114
4.2.1. Conjuntos de entrenamiento, validación y prueba......................... 115
4.2.2. A tener en cuenta 118
4.3. Procesamiento de datos, ingeniería de características
y aprendizaje de características.. 119
4.3.1. Procesamiento de datos para redes neuronales........................... 119
4.3.2. Ingeniería de características 120
4.4. Sobreajuste y subajuste...122
4.4.1. Reducir el tamaño de la red...123
4.4.2. Añadir regularización de peso 125
4.4.3. Añadir *dropout* 127
4.5. El flujo de trabajo universal del *machine learning*.....................................129
4.5.1. Definir el problema y montar un conjunto de datos......................129
4.5.2. Elegir una medida del éxito 130
4.5.3. Decidir un protocolo de evaluación...130
4.5.4. Preparación de los datos.. 131
4.5.5. Desarrollar un modelo que lo haga mejor
que un modelo de referencia 131
4.5.6. Ampliación: desarrollar un modelo con sobreajuste 132
4.5.7. Regularización del modelo y ajuste de los hiperparámetros.........133

Parte 2. *Deep learning* en la práctica 25

Capítulo 5. *Deep learning* para visión por ordenador 137

5.1. Introducción a las *convnets*..137
5.1.1. La operación de convolución 140
5.1.2. La operación *max pooling*..145
5.2. Entrenar una *convnet* desde cero con un conjunto
de datos pequeño 147
5.2.1. La relevancia del *deep learning* para problemas
con pocos datos 148
5.2.2. Descargar los datos...149
5.2.3. Crear la red.. 151
5.2.4. Procesamiento de datos 153
5.2.5. Utilizar el aumento de datos..157
5.3. Utilizar una *convnet* preentrenada.. 161
5.3.1. Extracción de características...162
5.3.2. Ajuste fino 171
5.3.3. Resumen... 177
5.4. Visualizar lo que aprenden las *convnets* 178
5.4.1. Visualizar activaciones intermedias 178
5.4.2. Visualizar filtros de *convnets* 185
5.4.3. Visualizar mapas de calor de activación de clase 190

Capítulo 6. *Deep learning* para texto y secuencias 197

6.1. Trabajar con datos de texto...198
6.1.1. Codificación *one-hot* de palabras y caracteres.................................199
6.1.2. Utilizar *embeddings* de palabras 201
6.1.3. Juntar todas las piezas: de texto bruto a *embeddings*
de palabras 207
6.1.4. Resumen..213
6.2. Entender las redes neuronales recurrentes 214
6.2.1. Una capa recurrente en Keras 216
6.2.2. Comprender las capas LSTM y GRU 220
6.2.3. Ejemplo concreto de LSTM en Keras 223
6.2.4. Resumen 225
6.3. Uso avanzado de las redes neuronales recurrentes.................................. 225
6.3.1. Un problema de predicción de temperatura.................................... 225
6.3.2. Preparar los datos 228
6.3.3. Un punto de referencia de sentido común
sin *machine learning*...230

6.3.4. Una aproximación básica con *machine learning* 231
6.3.5. Una primera referencia recurrente 233
6.3.6. Usar el *dropout* recurrente para combatir el sobreajuste 235
6.3.7. Apilar capas recurrentes 236
6.3.8. Utilizar RNR bidireccionales 238
6.3.9. Ir más allá 241
6.3.10. Resumen 242
6.4. Procesamiento de secuencias con *convnets* 243
6.4.1. Comprender la convolución 1D para datos secuenciales 243
6.4.2. *Pooling* 1D para datos secuenciales 244
6.4.3. Implementación de una *convnet* 1D 244
6.4.4. Combinar RNC y RNR para procesar secuencias largas 246
6.4.5. Resumen 250

Capítulo 7. Prácticas adecuadas de *deep learning* avanzado 253

7.1. Más allá del modelo Sequential: la API funcional de Keras 253
7.1.1. Introducción a la API funcional 257
7.1.2. Modelos con múltiples entradas 258
7.1.3. Modelos con múltiples salidas 260
7.1.4. Grafos acíclicos dirigidos de capas 262
7.1.5. Compartir pesos de capas 266
7.1.6. Modelos como capas 267
7.1.7. Resumen 268
7.2. Inspeccionar y monitorizar modelos de *deep learning* utilizando retrollamadas de Keras y TensorBoard 269
7.2.1. Utilizar retrollamadas para actuar en un modelo durante el entrenamiento 269
7.2.2. Introducción a TensorBoard: el *framework* de visualización de TensorFlow 272
7.2.3. Resumen 277
7.3. Sacar el máximo partido a nuestros modelos 279
7.3.1. Patrones de arquitectura avanzados 279
7.3.2. Optimización de hiperparámetros 282
7.3.3. Ensamblaje de modelos 284
7.3.4. Resumen 286

Capítulo 8. *Deep learning* generativo 289

8.1. Generación de texto con LSTM 291
8.1.1. Breve historia de las redes recurrentes generativas 291
8.1.2. ¿Cómo se generan los datos secuenciales? 292

8.1.3. La importancia de la estrategia de muestreo 292
8.1.4. Implementación de generación de texto con LSTM a nivel de carácter 294
8.1.5. Resumen 299
8.2. DeepDream 299
8.2.1. Implementación de DeepDream en Keras 300
8.2.2. Resumen 306
8.3. Transferencia de estilo neuronal 306
8.3.1. La pérdida de contenido 307
8.3.2. La pérdida de estilo 307
8.3.3. Transferencia de estilo neuronal en Keras 308
8.3.4. Resumen 314
8.4. Generar imágenes con autocodificadores variacionales 314
8.4.1. Muestreo de espacios latentes de imágenes 315
8.4.2. Vectores conceptuales para la edición de imágenes 315
8.4.3. Autocodificadores variacionales 317
8.4.4. Resumen 322
8.5. Introducción a las redes generativas antagónicas 323
8.5.1. Implementación esquemática de una GAN 324
8.5.2. Un saco de trucos 325
8.5.3. La generadora 327
8.5.4. La discriminadora 327
8.5.5. La red antagonista 328
8.5.6. Cómo entrenar la DCGAN 329
8.5.7. Resumen 330

Capítulo 9. Conclusiones 333

9.1. Conceptos clave para revisar 333
9.1.1. Varios enfoques de la IA 334
9.1.2. Qué hace al *deep learning* especial dentro del campo del *machine learning* 334
9.1.3. Cómo pensar en el *deep learning* 335
9.1.4. Tecnologías habilitadoras clave 336
9.1.5. El flujo de trabajo universal del *machine learning* 337
9.1.6. Arquitecturas de red clave 338
9.1.7. El espacio de posibilidades 342
9.2. Las limitaciones del *deep learning* 343
9.2.1. El riesgo de antropomorfizar los modelos de *machine learning* 344
9.2.2. Generalización local frente a generalización extrema 346
9.2.3. Resumen 348

9.3. El futuro del *deep learning* 348
9.3.1. Modelos como programas 349
9.3.2. Más allá de la retropropagación y las capas diferenciables 351
9.3.3. *Machine learning* automatizado 351
9.3.4. Aprendizaje permanente y reutilización de subrutinas modulares 352
9.3.5. La visión a largo plazo 354
9.4. Mantenerse al día en un campo que avanza deprisa 355
9.4.1. Practicar con problemas del mundo real utilizando Kaggle 355
9.4.2. Leer sobre los últimos avances en arXiv 356
9.4.3. Explorar el ecosistema Keras 356
9.5. Despedida 357

Parte 3. Apéndices **359**

Apéndice A. Instalar Keras y sus dependencias en Ubuntu **361**

A.1. Instalar la *suite* científica de Python 362
A.2. Configurar el soporte para GPU 363
A.3. Instalar Theano (opcional) 364
A.4. Instalar Keras 364

Apéndice B. Ejecutar *notebooks* de Jupyter en una instancia de GPU en EC2 **367**

B.1. ¿Qué son los *notebooks* de Jupyter? ¿Por qué ejecutar *notebooks* de Jupyter en GPU en AWS? 367
B.2. ¿Por qué no querríamos utilizar Jupyter en AWS para *deep learning*? 368
B.3. Configurar una instancia de GPU en AWS 368
B.3.1. Configurar Jupyter 371
B.4. Instalar Keras 372
B.5. Configurar la redirección del puerto local 372
B.6. Utilizar Jupyter desde su navegador local 372

Índice alfabético **375**

4 Fundamentos del *machine learning*

En este capítulo:

- Formas de *machine learning* más allá de la clasificación y la regresión.
- Procedimientos de evaluación formal para modelos de aprendizaje automático.
- Preparación de datos para el *deep learning*.
- Ingeniería de características.
- Resolución del sobreajuste.
- El flujo de trabajo universal para abordar problemas de aprendizaje automático.

Tras los tres ejemplos prácticos del capítulo 3, debería empezar a sentirse familiarizado con el uso de redes neuronales para abordar problemas de clasificación y regresión. Ya ha sido testigo del principal problema del *machine learning*: el sobreajuste. Este capítulo convertirá parte de su nueva intuición en un marco conceptual sólido para afrontar y resolver problemas de *deep learning*. Consolidaremos todos estos conceptos (evaluación de modelos, procesamiento de datos e ingeniería de características, y resolución del sobreajuste) en un flujo de trabajo detallado de siete pasos para sacar adelante cualquier tarea de aprendizaje automático.

4.1. Cuatro ramas de *machine learning*

En nuestros ejemplos anteriores, se ha familiarizado con tres tipos específicos de problemas de *machine learning*: clasificación binaria, clasificación multiclase y regresión escalar. Los tres son ejemplos de aprendizaje supervisado, donde el objetivo es aprender la relación entre las entradas de entrenamiento y los objetivos de entrenamiento.

El aprendizaje supervisado es solo la punta del iceberg: el *machine learning* es un campo muy vasto con una taxonomía de subcampos compleja. Los algoritmos de *machine learning* suelen entrar en cuatro grandes categorías, que describiremos en las próximas secciones.

mapa de características de entrada y aplica la misma transformación a todas esas porciones, produciendo un mapa de características de salida. Dicho mapa todavía es un tensor 3D: tiene una anchura y una altura. Su profundidad puede ser arbitraria, porque la profundidad de salida es un parámetro de la capa, y los diferentes canales en ese eje de profundidad ya no representan colores específicos como en la entrada RGB, sino que representan filtros. Los filtros codifican aspectos específicos de los datos de entrada: en un nivel alto, un solo filtro podría codificar el concepto "presencia de una cara en la entrada", por ejemplo.

En el ejemplo de MNIST, la primera capa de convolución toma un mapa de características de tamaño `(28, 28, 1)` y genera como salida un mapa de características de tamaño `(26, 26, 32)`: computa 32 filtros sobre su entrada. Cada uno de estos 32 canales de salida contiene una cuadrícula de valores de 26 × 26, que es un mapa de respuestas del filtro sobre la entrada, indicando la respuesta de ese patrón de filtro en diferentes ubicaciones en la entrada (véase la figura 5.3). Eso es lo que significa el término "mapa de características": cada dimensión en el eje de profundidad es una característica (o filtro) y el tensor 2D `output[:, :, n]` es el mapa espacial 2D de la respuesta de este filtro sobre la entrada.

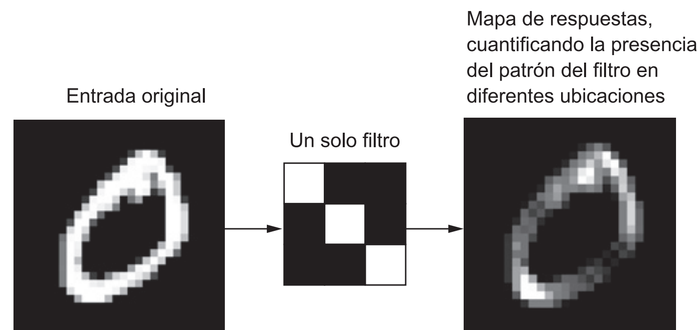

Figura 5.3. El concepto de un mapa de respuestas: un mapa 2D de la presencia de un patrón en diferentes ubicaciones en una entrada.

Las convoluciones se definen por dos parámetros clave:

- **Tamaño de las porciones extraídas de las entradas:** Suelen ser de 3 × 3 o de 5 × 5. En el ejemplo, eran de 3 × 3, que es una opción habitual.
- **Profundidad del mapa de características de salida:** El número de filtros computados por la convolución. El ejemplo empezaba con una profundidad de 32 y terminaba con una profundidad de 64.

En capas `Conv2D` de Keras, estos parámetros son los primeros argumentos que se pasan a la capa: `Conv2D(profundidad_salida, (altura_ventana, anchura_ventana))`.

Un convolución funciona "deslizando" estas ventanas de tamaño 3 × 3 o 5 × 5 sobre el mapa de características de entrada 3D, deteniéndose en cada ubicación posible y extrayendo la porción 3D de las características circundantes (con la forma `(altura_ventana, anchura_ventana, profundidad_salida)`). Después, cada una de esas porciones 3D se transforma (mediante un producto tensorial con la misma matriz de pesos aprendida, llamado "*kernel* de convolución") en un vector 1D con la forma `(profundidad_salida,)`. Todos estos vectores se reagrupan entonces espacialmente formando un mapa de salida 3D con la forma `(altura, anchura, profundidad_salida)`. Cada ubicación espacial del mapa de características de salida se corresponde con la misma ubicación en el mapa de características de entrada (por ejemplo, la esquina inferior derecha de la salida contiene información sobre la esquina inferior derecha de la entrada). Por ejemplo, con ventanas de 3 × 3, el vector `output[i, j, :]` viene de la porción 3D `input[i-1:i+1, j-1:j+1, :]`. El proceso completo se detalla en la figura 5.4.

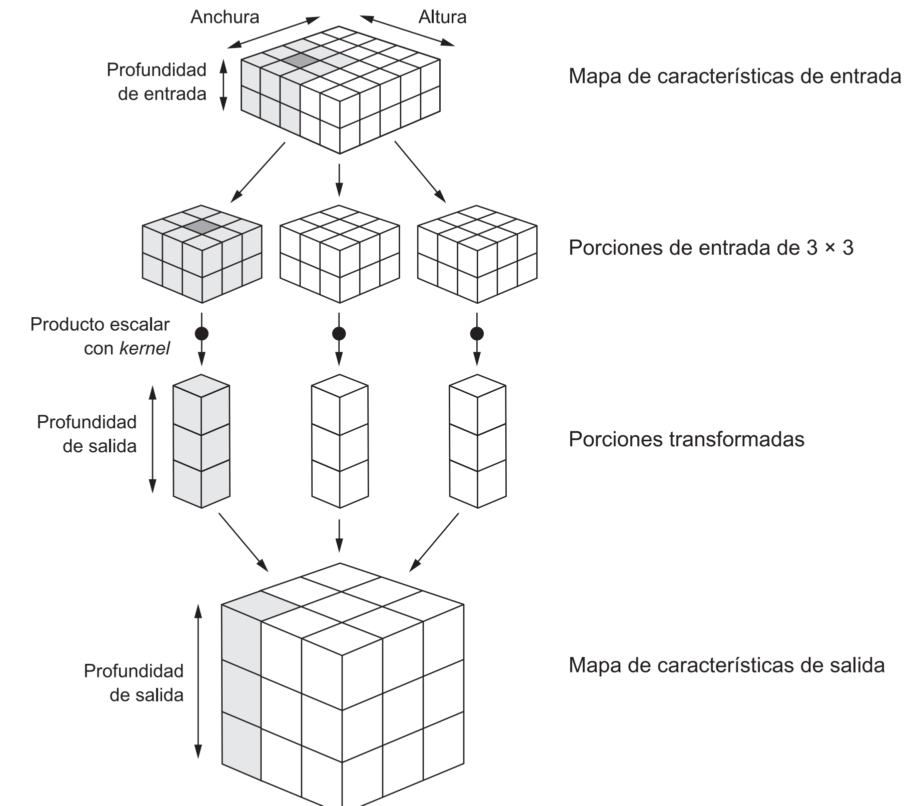

Figura 5.4. Funcionamiento de la convolución.

Tenga en cuenta que la anchura y la altura de salida pueden ser diferentes a la anchura y la altura de entrada. Esto puede deberse a dos razones:

- Efectos de los bordes, que pueden contrarrestarse rellenando el mapa de características de entrada.
- El uso de "pasos de avance", que definiremos enseguida.

Vamos a ver estas nociones con más detalle.

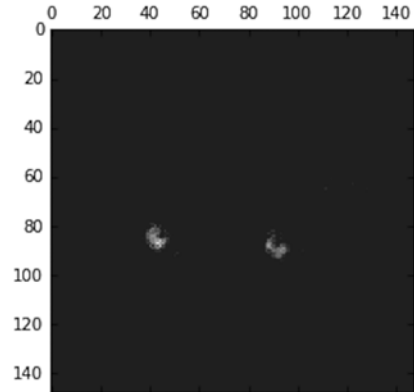

Figura 5.26. Séptimo canal de la activación de la primera capa en la imagen del gato de prueba.

Este parece un detector de "puntos verdes brillantes", útil para codificar ojos de gato. En este punto, vamos a trazar una visualización completa de todas las activaciones de la red (véase la figura 5.27). Vamos a extraer y trazar todos los canales en cada uno de los ocho mapas de activaciones y apilaremos los resultados en un tensor de imagen grande, con canales apilados unos junto a otros.

Listado 5.31. Visualización de todos los canales en cada activación intermedia.

```
layer_names = []
for layer in model.layers[:8]:
   layer_names.append(layer.name)

images_per_row = 16

for layer_name, layer_activation in zip(layer_names, activations):
   n_features = layer_activation.shape[-1]

   size = layer_activation.shape[1]

   n_cols = n_features // images_per_row
   display_grid = np.zeros((size * n_cols, images_per_row * size))

   for col in range(n_cols):
      for row in range(images_per_row):
         channel_image = layer_activation[0,
                                          :, :,
                                          col * images_per_row + row]
         channel_image -= channel_image.mean()
         channel_image /= channel_image.std()
         channel_image *= 64
         channel_image += 128
         channel_image = np.clip(channel_image, 0, 255).astype('uint8')
         display_grid[col * size : (col + 1) * size,
                      row * size : (row + 1) * size] = channel_image
```

- Nombres de las capas para que los tenga como parte de su trazado (`layer_names = []` … `layer_names.append(layer.name)`)
- Muestra el mapa de características (`for layer_name, layer_activation in zip(layer_names, activations):`)
- Número de características en el mapa de características (`n_features = layer_activation.shape[-1]`)
- El mapa de características tiene la forma (1, size, size, n_features) (`size = layer_activation.shape[1]`)
- Tesela los canales de activación de esta matriz (`n_cols = …`, `display_grid = …`)
- Tesela cada filtro en una gran cuadrícula horizontal (`for col in range(n_cols):`)
- Postprocesa la característica para hacerla aceptable visualmente (`channel_image -= channel_image.mean()`)
- Muestra la cuadrícula (`display_grid[col * size : (col + 1) * size,`)


```
scale = 1. / size
plt.figure(figsize=(scale * display_grid.shape[1],
                    scale * display_grid.shape[0]))
plt.title(layer_name)
plt.grid(False)
plt.imshow(display_grid, aspect='auto', cmap='viridis')
```

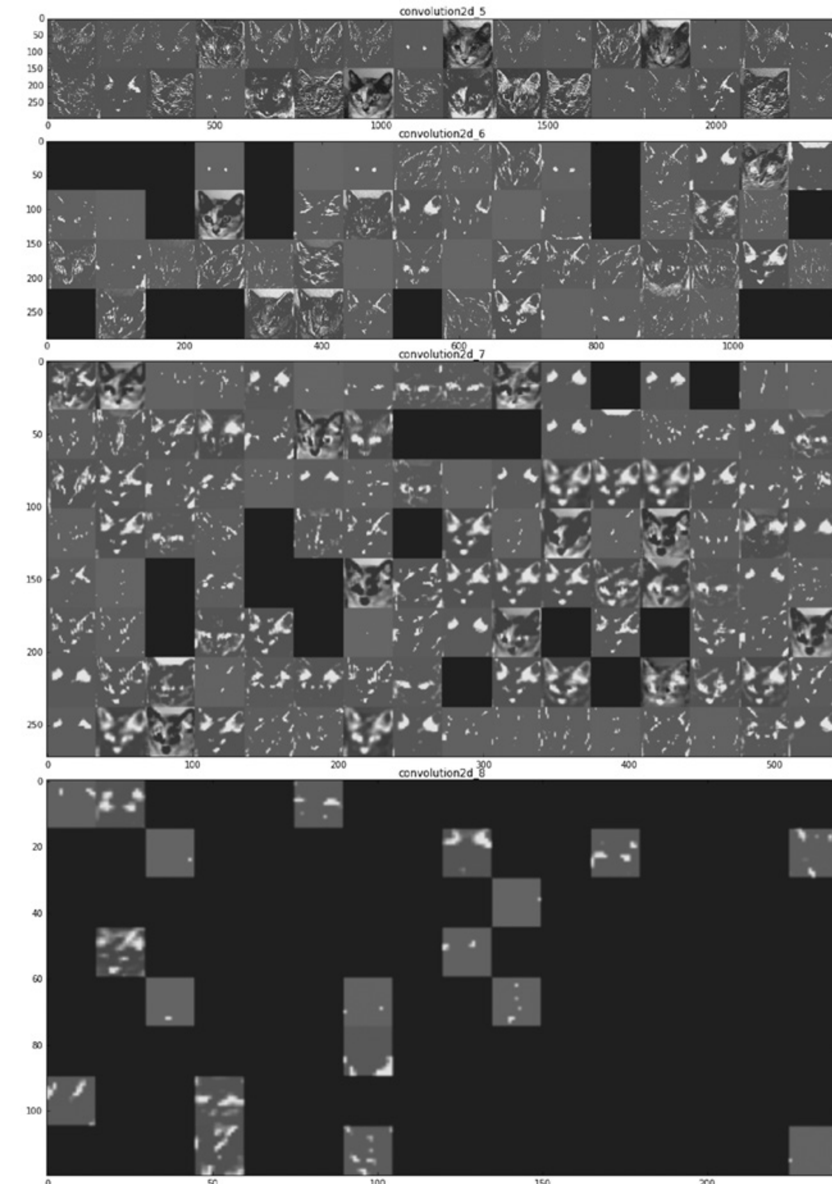

Figura 5.27. Todos los canales de cada activación de la capa en la imagen del gato de prueba.

Prácticas adecuadas de *deep learning* avanzado

En este capítulo:

- La API funcional de Keras.
- Utilizar retrollamadas en Keras.
- Trabajar con la herramienta de visualización TensorBoard.
- Prácticas adecuadas importantes para desarrollar modelos de última generación.

Este capítulo explora varias herramientas potentes que nos acercarán a poder desarrollar modelos de última generación para problemas difíciles. Al utilizar la API funcional de Keras, podemos crear modelos de tipo grafo, compartir una capa a través de diferentes entradas y utilizar modelos de Keras igual que funciones de Python. Las retrollamadas en Keras y la herramienta de visualización basada en el navegador TensorBoard nos permite monitorizar modelos durante el entrenamiento. También hablaremos de otras prácticas adecuadas, como la normalización de lotes, las conexiones residuales, la optimización de hiperparámetros y ensamblaje de modelos.

7.1. Más allá del modelo Sequential: la API funcional de Keras

Hasta ahora, todas las redes neuronales presentadas en este libro se han implementado utilizando el modelo `Sequential`. El modelo `Sequential` asume que la red tiene exactamente una entrada y exactamente una salida y que consiste en una pila lineal de capas (véase la figura 7.1).

Se trata de una suposición que se verifica con frecuencia; la configuración es tan común que hemos podido tratar muchos temas y aplicaciones prácticas en las páginas anteriores utilizando solo la clase de modelo `Sequential`. Pero este conjunto de suposiciones es demasiado

```
    factors = (1,
              float(size[0]) / img.shape[1],
              float(size[1]) / img.shape[2],
              1)
    return scipy.ndimage.zoom(img, factors, order=1)

def save_img(img, fname):
    pil_img = deprocess_image(np.copy(img))
    scipy.misc.imsave(fname, pil_img)

def preprocess_image(image_path):       <- Función útil para abrir, redimensionar y dar formato a
                                           imágenes como tensores que pueda procesar Inception V3
    img = image.load_img(image_path)
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = inception_v3.preprocess_input(img)
    return img

def deprocess_image(x):       <- Función útil para convertir un tensor en una imagen válida
    if K.image_data_format() == 'channels_first':
        x = x.reshape((3, x.shape[2], x.shape[3]))
        x = x.transpose((1, 2, 0))
    else:
        x = x.reshape((x.shape[1], x.shape[2], 3))       <- Deshace el preprocesamiento realizado
                                                            por inception_v3.preprocess_input
    x /= 2.
    x += 0.5
    x *= 255.
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

Nota: Como la red Inception V3 original estaba entrenada para reconocer conceptos en imágenes de tamaño 299 × 299 y dado que el proceso implica reducir las imágenes en un factor considerable, la implementación de DeepDream produce resultados mucho mejores con imágenes de entre 300 × 300 y 400 × 400. Independientemente de eso, puede ejecutar el mismo código con imágenes de cualquier tamaño y relación.

Partiendo de una fotografía tomada en las pequeñas colinas entre la bahía de San Francisco y el campus de Google, obtuvimos DeepDream, que muestra la figura 8.5.

Le recomiendo encarecidamente explorar lo que se puede hacer ajustando las capas que use en su pérdida. Las capas que están más abajo en la red contienen representaciones más locales y menos abstractas y conducen a patrones de sueños con un aspecto más geométrico. Las capas que están más arriba conducen a patrones visuales más reconocibles basados en los objetos más habituales en ImageNet, como ojos de perro, plumas de pájaro, etc. Puede usar la generación aleatoria de los parámetros en el diccionario `layer_contributions` para explorar rápidamente muchas combinaciones de capas diferentes. La figura 8.6 muestra distintos resultados obtenidos usando configuraciones de capa diferentes, a partir de una imagen de un delicioso hojaldre casero.

Figura 8.5. Ejecución del código DeepDream con una imagen de ejemplo.

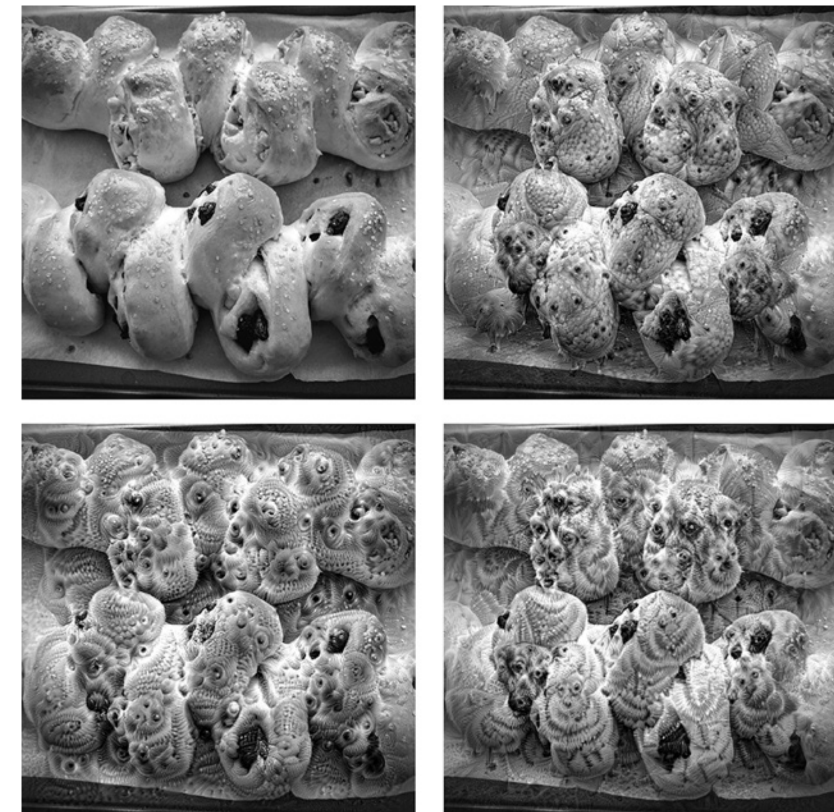

Figura 8.6. Prueba de distintas configuraciones de DeepDream con una imagen de ejemplo.

de descripciones en inglés de las características de un producto de software, escrito por un gerente de producción, además del correspondiente código fuente desarrollado por un equipo de ingenieros para cumplir estos requisitos. Incluso con estos datos, no podríamos entrenar un modelo de *deep learning* para que leyese una descripción del producto y generase la base de código apropiada. Este es solo un ejemplo entre muchos. En general, cualquier cosa que requiera razonamiento (como programar o aplicar el método científico), planificación a largo plazo y manipulación de datos algorítmicos está fuera del alcance de los modelos de *deep learning*, no importa cuántos datos introduzcamos. Incluso aprender un algoritmo de ordenación con una red neuronal profunda es muy difícil.

Esto se debe a que un modelo de *deep learning* es solo una cadena de transformaciones geométricas simples continuas que asignan un espacio de vectores a otro. Lo único que puede hacer es asignar una cantidad de datos X a otra cantidad Y, asumiendo la existencia de una transformación continua de X a Y que pueda aprenderse. Un modelo de *deep learning* puede interpretarse como un tipo de programa, pero, a la inversa, la mayoría de programas no pueden expresarse como modelos de *deep learning*; para la mayoría de las tareas, o no existe una red neuronal profunda correspondiente que la resuelva, o, si existe, puede que no sea aprendible: la transformación geométrica correspondiente puede ser demasiado compleja o es posible que no haya datos apropiados disponibles para el aprendizaje.

Aumentar la escala de las técnicas actuales de *deep learning* apilando más capas y usando más datos de entrenamiento puede paliar, aunque superficialmente, algunos de estos problemas, pero no resolverá los problemas fundamentales como que los modelos de *deep learning* están limitados en lo que pueden representar y que la mayoría de los programas que nos interesa aprender no pueden expresarse como una transformación geométrica continua de una variedad de datos.

9.2.1. El riesgo de antropomorfizar los modelos de *machine learning*

Un riesgo real de la IA contemporánea es malinterpretar lo que hacen los modelos de *deep learning* y sobrestimar sus capacidades. Un rasgo fundamental de los humanos es nuestra teoría de la mente: nuestra tendencia a proyectar intenciones, creencias y conocimientos en las cosas que nos rodean. Dibujar una cara sonriente en una roca de repente hace que esté "contenta" (en nuestras mentes). Aplicado al *deep learning*, esto significa que, por ejemplo, cuando somos capaces de entrenar con éxito de algún modo un modelo para generar pies de foto para describir imágenes, acabamos pensando que el modelo "entiende" los contenidos de las imágenes y los pies de foto que genera. Después, nos llevamos una sorpresa cuando cualquier tipo de desviación leve del tipo de imágenes presente en los datos de entrenamiento hace que el modelo genere pies de foto completamente absurdos (véase la figura 9.1).

En particular, esto resulta más llamativo con los ejemplos antagonistas, que son muestras introducidas en una red de *deep learning* que están diseñadas para engañar al modelo y que este las clasifique mal. Ya sabemos que, por ejemplo, es posible utilizar ascenso de gradiente en un espacio de entrada para generar entradas que maximicen la activación de algunos filtros de *convnets*; esta es la base de la técnica de visualización de filtros explicada en el capítulo 5 y también del algoritmo DeepDream del capítulo 8. De manera similar, a través del ascenso de gradiente, podemos modificar ligeramente una imagen para maximizar la predicción de clase para una clase determinada. Al tomar una imagen de un panda y añadirle un gradiente de gibón, podemos conseguir que una red neuronal clasifique al panda como gibón (véase la figura 9.2). Esto demuestra tanto la fragilidad de estos modelos como la diferencia profunda entre su asignación entrada-a-salida y nuestra percepción humana.

El niño sostiene un bate de béisbol.

Figura 9.1. Fallo de un sistema de colocación de pies de foto en imágenes basado en el *deep learning*.

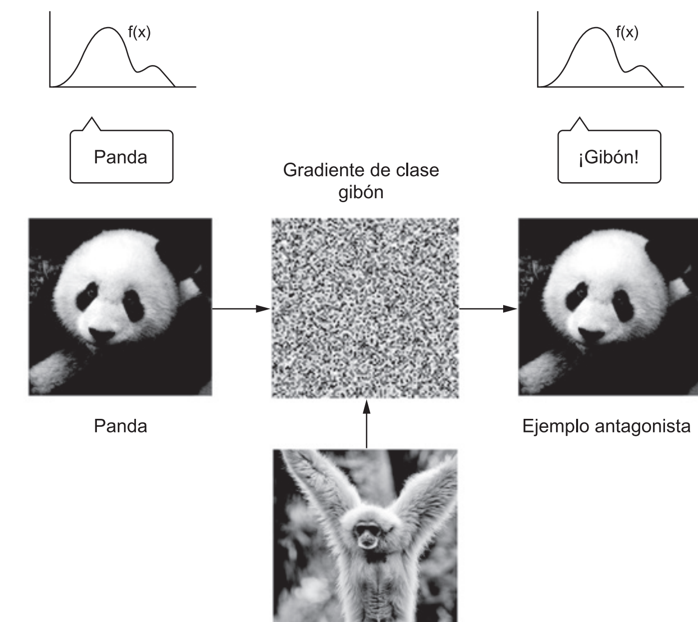

Figura 9.2. Un ejemplo antagonista: cambios imperceptibles en una imagen pueden alterar por completo la clasificación que hace el modelo de la imagen.

Ejecutar *notebooks* de Jupyter en una instancia de GPU en EC2

Este apéndice ofrece una guía paso a paso para la ejecución de *notebooks* de Jupyter de *deep learning* en una instancia de GPU en AWS y la edición de los *notebooks* desde cualquier parte del navegador. Esta es la opción perfecta para la investigación de *deep learning* si no tenemos una GPU en nuestra máquina local. La versión original (y actualizada) de esta guía puede encontrarse en `https://blog.keras.io`.

B.1. ¿Qué son los *notebooks* de Jupyter? ¿Por qué ejecutar *notebooks* de Jupyter en GPU en AWS?

Un *notebook* de Jupyter es una aplicación web que nos permite escribir y anotar código Python de manera interactiva. Es una forma estupenda de experimentar, investigar y compartir aquello en lo que estamos trabajando.

Muchas aplicaciones de *deep learning* son muy intensivas a nivel computacional y pueden tardar horas o incluso días si se ejecutan en núcleos de la CPU de un portátil. Ejecutarlas en una GPU puede acelerar de manera considerable el entrenamiento y la inferencia (a menudo, a una velocidad entre 5 y 10 veces mayor, cuando se pasa de una CPU moderna a una sola GPU moderna). Pero es posible que no tengamos acceso a una GPU en nuestra máquina local. Ejecutar *notebooks* de Jupyter en AWS nos proporciona la misma experiencia que trabajar en nuestra máquina local, al mismo tiempo que nos permite utilizar una o más GPU en AWS. Además, solo pagamos por lo que utilizamos, lo que resulta favorable en comparación con invertir en nuestra propia GPU (o más de una) si solo utilizamos *deep learning* de forma ocasional.

Deep Learning con Python

François Chollet

El aprendizaje automático ha progresado de manera notable en los últimos años. Hemos pasado del discurso casi inutilizable y el reconocimiento de imágenes a una precisión casi humana. Hemos pasado de máquinas que no podían ganar a un jugador de go decente a derrotar al campeón del mundo. Tras este progreso se encuentra el deep learning, una combinación de avances en ingeniería, prácticas adecuadas y teoría que permite crear una gran abundancia de aplicaciones inteligentes que antes eran imposibles.

Deep Learning con Python presenta el campo del deep learning utilizando el lenguaje Python y la potente biblioteca Keras. Escrito por François Chollet, creador de Keras e investigador de Google AI, este libro desarrolla su comprensión mediante explicaciones intuitivas y ejemplos prácticos. Explorará conceptos complicados y practicará con aplicaciones en visión por ordenador (visión artificial), procesamiento de lenguaje natural y modelos generativos. Para cuando acabe, tendrá el conocimiento y las habilidades prácticas para aplicar el deep learning a sus propios proyectos.

Qué incluye

- Deep learning desde los principios básicos
- Configuración de su propio entorno de deep learning
- Modelos de clasificación de imágenes
- Deep learning para texto y secuencias
- Transferencia de estilo neuronal, generación de texto y generación de imágenes

Los lectores necesitan tener conocimientos intermedios de Python. No se requiere experiencia previa con Keras, TensorFlow ni machine learning.

François Chollet es investigador en inteligencia artificial en el Google Brain Team y autor de la biblioteca de deep learning Keras.

"La explicación más clara del deep learning que me he encontrado… Da gusto leerlo."
—Richard Tobias, Cephasonics

"Un excelente título práctico introductorio, con gran profundidad y amplitud."
—David Blumenthal-Barby
Babbel

"Llena el hueco entre el bombo publicitario y el sistema de deep learning en funcionamiento."
—Peter Rabinovitch, Akamai

"El mejor recurso para convertirse en un maestro de Keras y del deep learning."
—Claudio Rodriguez
Cox Media Group

ISBN 978-84-415-4225-9
2315153
9 788441 542259

ANAYA MULTIMEDIA
www.anayamultimedia.es