PRÁCTICA 1: VGG16 en Keras

- Autor: Francisco Herrera Barajas
- Fechas: 21 / 04 / 2025

1. Programar en Keras la red VGG16 para clasificación binaria

Importamos las bibliotecas básicas

In [ ]:
```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import os
import requests
import zipfile
import urllib.request
import shutil
from PIL import Image
from tensorflow.keras.callbacks import Callback
from sklearn.metrics import confusion_matrix
from scipy.stats import chi2_contingency
import warnings
warnings.filterwarnings('ignore')
```

Para entrenar a la red nos conectaremos a un entorno con GPU. Verficamos que tengamos una GPU disponible.

In [2]:
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  1
2025-04-25 18:08:50.347791: I external/local_xla/xla/stream_executor/cuda/cuda_execu
tor.cc:901] successful NUMA node read from SysFS had negative value (-1), but there
must be at least one NUMA node, so returning NUMA node zero. See more at https://git
hub.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-pci#L344-L355
2025-04-25 18:08:50.638204: I external/local_xla/xla/stream_executor/cuda/cuda_execu
tor.cc:901] successful NUMA node read from SysFS had negative value (-1), but there
must be at least one NUMA node, so returning NUMA node zero. See more at https://git
hub.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-pci#L344-L355
2025-04-25 18:08:50.638469: I external/local_xla/xla/stream_executor/cuda/cuda_execu
tor.cc:901] successful NUMA node read from SysFS had negative value (-1), but there
must be at least one NUMA node, so returning NUMA node zero. See more at https://git
hub.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-pci#L344-L355
```

Para gestionar mejor el uso de la memoria de la GPU utilizamos

In [3]:
```
physical_devices = tf.config.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(physical_devices[0], True)
```

Procedemos a descargarnos la base de datos

In [4]:
```
# URL de la base de datos
dataset_url = "https://download.microsoft.com/download/3/e/1/3e1c3f21-ecdb-4869-836
zip_path = "kagglecatsanddogs_5340.zip"
extract_path = "cats_and_dogs"

# Descargar el archivo si no existe
if not os.path.exists(zip_path):
    print("Descargando dataset...")
    urllib.request.urlretrieve(dataset_url, zip_path)
    print("Descarga completa.")

# Extraer el contenido
if not os.path.exists(extract_path):
    print("Extrayendo archivos...")
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(extract_path)
    print("Extracción completa.")
else:
    print("Los archivos ya han sido extraídos.")
```

```
Los archivos ya han sido extraídos.
```

Creamos nuevos directorios para organizar la base de datos

In [5]:
```
# Rutas centrales
original_dir = "cats_and_dogs/PetImages"  # Dataset original sin procesar
base_dir = "data/cats_and_dogs_filtered"  # Dataset organizado
train_dir = os.path.join(base_dir, 'train')
val_dir = os.path.join(base_dir, 'validation')

# Ruta orginales de las imégenes
cat_dir = os.path.join(original_dir, 'Cat')
dog_dir = os.path.join(original_dir, 'Dog')

# Ruta creadas para los gatos
train_cat_dir = os.path.join(train_dir, 'cats')
val_cat_dir = os.path.join(val_dir, 'cats')

# Rutas creadas para los perros
train_dog_dir = os.path.join(train_dir, 'dogs')
val_dog_dir = os.path.join(val_dir, 'dogs')
```

Obtenemos los nombres de las imágenes y las metememos en listas

In [6]:
```
# Obtener listas de las imágenes
all_cats = [f for f in os.listdir(cat_dir) if f.endswith('.jpg')]
all_dogs = [f for f in os.listdir(dog_dir) if f.endswith('.jpg')]
```

Separar las imágenes en conjuntos de entrenamiento (1000) y validación (250).

In [7]:
```
# Separar train/val
train_cats = all_cats[:2000]
val_cats = all_cats[2000:2500]

train_dogs = all_dogs[:2000]
val_dogs = all_dogs[2000:2500]
```

Función para limpiar previamente las carpetas

In [8]:
```
def limpiar_dataset(base_dir):
    subdirs = [
        os.path.join(base_dir, 'train', 'cats'),
        os.path.join(base_dir, 'train', 'dogs'),
        os.path.join(base_dir, 'validation', 'cats'),
        os.path.join(base_dir, 'validation', 'dogs')
    ]

    for folder in subdirs:
        if os.path.exists(folder):
            for f in os.listdir(folder):
                file_path = os.path.join(folder, f)
                try:
                    if os.path.isfile(file_path):
                        os.remove(file_path)
                except Exception as e:
                    print(f"No se pudo eliminar {file_path}: {e}")
        else:
            os.makedirs(folder)

    print("Carpetas de entrenamiento y validación limpiadas.")
```

Limpiamos las carpetas

In [9]:
```
limpiar_dataset("data/cats_and_dogs_filtered")
```

```
Carpetas de entrenamiento y validación limpiadas.
```

Función para copiar las imágenes y moverlas a los nuevos directorios, también la redimensionamos para poder meterlas en la red

In [10]:
```
def mover_imagenes_desde_listas(lista_train, lista_val, clase, origen, train_dest,
    os.makedirs(train_dest, exist_ok=True)
    os.makedirs(val_dest, exist_ok=True)

    # Mover imágenes de entrenamiento
    for i, fname in enumerate(lista_train):
        src_path = os.path.join(origen, fname)
        try:
            with Image.open(src_path) as img:
                img = img.convert("RGB")
                img = img.resize(target_size)
                newname = f"{clase}_train_{i}.jpg"
```

```
                img.save(os.path.join(train_dest, newname))
        except Exception as e:
            print(f"Fallo en {fname}: {e}")
            continue

    # Mover imágenes de validación
    for i, fname in enumerate(lista_val):
        src_path = os.path.join(origen, fname)
        try:
            with Image.open(src_path) as img:
                img = img.convert("RGB")
                img = img.resize(target_size)
                newname = f"{clase}_val_{i}.jpg"
                img.save(os.path.join(val_dest, newname))
        except Exception as e:
            print(f" Fallo en {fname}: {e}")
            continue

    print(f"{clase}: {len(lista_train)} entrenamiento, {len(lista_val)} validación"
```

Aplicamos la función

In [11]:
```
# Ejecutar
mover_imagenes_desde_listas(train_cats, val_cats, 'cat', cat_dir, train_cat_dir, va
mover_imagenes_desde_listas(train_dogs, val_dogs, 'dog', dog_dir, train_dog_dir, va
```

```
cat: 2000 entrenamiento, 500 validación
Fallo en 11702.jpg: cannot identify image file 'cats_and_dogs/PetImages/Dog/11702.jp
g'
dog: 2000 entrenamiento, 500 validación
```

Aumentación para el entrenamiento

In [35]:
```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

Normalización para la validación

In [36]:
```
val_datagen = ImageDataGenerator(rescale=1./255)
```

Generadores de las imágenes

In [32]:
```
train_generator = train_datagen.flow_from_directory(
    'data/cats_and_dogs_filtered/train',
    target_size=(128, 128),
    batch_size=64,
    class_mode='binary',
```

```
    shuffle= False
)

validation_generator = val_datagen.flow_from_directory(
    'data/cats_and_dogs_filtered/validation',
    target_size=(128, 128),
    batch_size=64,
    class_mode='binary',
    shuffle= False
)
```

```
Found 3999 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

In [23]:
```
# Dataset de entrenamiento
train_dataset = tf.data.Dataset.from_generator(
    lambda: train_generator,
    output_signature=(
        tf.TensorSpec(shape=(None, 128, 128, 3), dtype=tf.float32),
        tf.TensorSpec(shape=(None,), dtype=tf.float32)
    )
).prefetch(tf.data.AUTOTUNE)

# Dataset de validación
val_dataset = tf.data.Dataset.from_generator(
    lambda: validation_generator,
    output_signature=(
        tf.TensorSpec(shape=(None, 128, 128, 3), dtype=tf.float32),
        tf.TensorSpec(shape=(None,), dtype=tf.float32)
    )
).prefetch(tf.data.AUTOTUNE)
```

Red VGG16

In [47]:
```
# Solo Dropout en Dense y en el último bloque convolucional
model = Sequential()

# Bloque 1
model.add(Conv2D(32, (3,3), padding="same", activation="relu", input_shape=(128,128
model.add(Conv2D(32, (3,3), padding="same", activation="relu", name="block1_conv2")
model.add(MaxPool2D((2,2), name="block1_pool"))

# Bloque 2
model.add(Conv2D(64, (3,3), padding="same", activation="relu"))
model.add(Conv2D(64, (3,3), padding="same", activation="relu"))
model.add(MaxPool2D((2,2)))

# Bloque 3
model.add(Conv2D(128, (3,3), padding="same", activation="relu"))
model.add(Conv2D(128, (3,3), padding="same", activation="relu"))
model.add(MaxPool2D((2,2)))

# Clasificación
model.add(Flatten())
model.add(Dense(512, activation="relu"))
```

```
model.add(Dropout(0.3))
model.add(Dense(1, activation="sigmoid",dtype="float32"))
```

In [48]:
```
from keras.optimizers import Adam

# Compilación:
model.compile(
    optimizer=Adam(learning_rate=0.00001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)
```

In [49]:
```
class GuardarPesosPrimeraYUltima(Callback):
    def __init__(self, total_epocas, carpeta="pesos"):
        super().__init__()
        self.epoca_inicio = 1
        self.epoca_final = total_epocas
        self.carpeta = carpeta
        os.makedirs(self.carpeta, exist_ok=True)

    def on_epoch_end(self, epoch, logs=None):
        epoca_actual = epoch + 1
        if epoca_actual == self.epoca_inicio:
            ruta = os.path.join(self.carpeta, f"pesos_iniciales_epoca_{epoca_actual
            self.model.save_weights(ruta)
            print(f"Guardados pesos INICIALES en: {ruta}")
        elif epoca_actual == self.epoca_final:
            ruta = os.path.join(self.carpeta, f"pesos_finales_epoca_{epoca_actual:0
            self.model.save_weights(ruta)
            print(f"Guardados pesos FINALES en: {ruta}")
```

In [50]:
```
custom_metrics = CustomMetricsCheckpointFull(
    train_gen=train_generator,
    val_gen=validation_generator,
    save_path="resultados/metricas"
)

callback_guardar_pesos = GuardarPesosPrimeraYUltima(total_epocas=200, carpeta="peso
```

In [51]:

```
from tensorflow.keras.mixed_precision import set_global_policy
set_global_policy('mixed_float16')

steps_per_epoch = len(train_generator)
validation_steps = len(validation_generator)

# ## Entrenamiento en GPU ##
with tf.device('/GPU:0'):
    history = model.fit(
        train_dataset,
        epochs=200,
        validation_data=val_dataset,
        steps_per_epoch=steps_per_epoch,
        validation_steps=validation_steps,
        callbacks=[ callback_guardar_pesos]
    )
```

```
Epoch 1/200
61/63 [============================>.] - ETA: 0s - loss: 1.1993 - accuracy: 0.4699Gu
ardados pesos INICIALES en: pesos_guardados/pesos_iniciales_epoca_001.h5
63/63 [==============================] - 19s 247ms/step - loss: 1.1829 - accuracy:
0.4779 - val_loss: 0.6926 - val_accuracy: 0.5500
Epoch 2/200
63/63 [==============================] - 11s 172ms/step - loss: 0.7005 - accuracy:
0.3443 - val_loss: 0.6927 - val_accuracy: 0.5030
Epoch 3/200
63/63 [==============================] - 10s 166ms/step - loss: 0.6960 - accuracy:
0.3563 - val_loss: 0.6928 - val_accuracy: 0.5010
Epoch 4/200
63/63 [==============================] - 11s 174ms/step - loss: 0.6980 - accuracy:
0.2641 - val_loss: 0.6931 - val_accuracy: 0.5470
Epoch 5/200
63/63 [==============================] - 10s 168ms/step - loss: 0.6932 - accuracy:
0.4126 - val_loss: 0.6931 - val_accuracy: 0.5600
Epoch 6/200
63/63 [==============================] - 10s 166ms/step - loss: 0.6925 - accuracy:
0.4394 - val_loss: 0.6927 - val_accuracy: 0.5000
Epoch 7/200
63/63 [==============================] - 10s 164ms/step - loss: 0.6970 - accuracy:
0.4039 - val_loss: 0.6931 - val_accuracy: 0.5640
Epoch 8/200
63/63 [==============================] - 12s 197ms/step - loss: 0.6931 - accuracy:
0.4399 - val_loss: 0.6931 - val_accuracy: 0.5660
Epoch 9/200
63/63 [==============================] - 10s 168ms/step - loss: 0.6932 - accuracy:
0.4949 - val_loss: 0.6931 - val_accuracy: 0.5690
Epoch 10/200
63/63 [==============================] - 10s 167ms/step - loss: 0.6932 - accuracy:
0.4881 - val_loss: 0.6931 - val_accuracy: 0.5630
Epoch 11/200
63/63 [==============================] - 11s 170ms/step - loss: 0.6996 - accuracy:
0.3648 - val_loss: 0.6929 - val_accuracy: 0.5030
Epoch 12/200
63/63 [==============================] - 11s 171ms/step - loss: 0.6948 - accuracy:
0.3053 - val_loss: 0.6931 - val_accuracy: 0.5590
Epoch 13/200
63/63 [==============================] - 12s 187ms/step - loss: 0.6927 - accuracy:
0.5174 - val_loss: 0.6928 - val_accuracy: 0.5010
Epoch 14/200
63/63 [==============================] - 12s 194ms/step - loss: 0.6960 - accuracy:
0.3913 - val_loss: 0.6931 - val_accuracy: 0.5610
Epoch 15/200
63/63 [==============================] - 10s 167ms/step - loss: 0.6932 - accuracy:
0.4824 - val_loss: 0.6930 - val_accuracy: 0.5510
Epoch 16/200
63/63 [==============================] - 10s 167ms/step - loss: 0.6932 - accuracy:
0.4829 - val_loss: 0.6931 - val_accuracy: 0.5620
Epoch 17/200
63/63 [==============================] - 11s 169ms/step - loss: 0.6931 - accuracy:
0.5036 - val_loss: 0.6931 - val_accuracy: 0.5650
Epoch 18/200
63/63 [==============================] - 10s 167ms/step - loss: 0.6931 - accuracy:
0.4929 - val_loss: 0.6931 - val_accuracy: 0.5590
```

```
Epoch 19/200
63/63 [==============================] - 14s 220ms/step - loss: 0.6915 - accuracy:
0.5074 - val_loss: 0.6925 - val_accuracy: 0.5000
Epoch 20/200
63/63 [==============================] - 12s 193ms/step - loss: 0.6963 - accuracy:
0.4706 - val_loss: 0.6926 - val_accuracy: 0.5000
Epoch 21/200
63/63 [==============================] - 10s 162ms/step - loss: 0.6957 - accuracy:
0.4551 - val_loss: 0.6926 - val_accuracy: 0.5000
Epoch 22/200
63/63 [==============================] - 10s 165ms/step - loss: 0.6946 - accuracy:
0.4706 - val_loss: 0.6926 - val_accuracy: 0.5000
Epoch 23/200
63/63 [==============================] - 11s 175ms/step - loss: 0.6943 - accuracy:
0.4736 - val_loss: 0.6926 - val_accuracy: 0.5010
Epoch 24/200
63/63 [==============================] - 9s 153ms/step - loss: 0.6941 - accuracy: 0.
4861 - val_loss: 0.6925 - val_accuracy: 0.5010
Epoch 25/200
63/63 [==============================] - 13s 201ms/step - loss: 0.6942 - accuracy:
0.4916 - val_loss: 0.6925 - val_accuracy: 0.5010
Epoch 26/200
63/63 [==============================] - 12s 192ms/step - loss: 0.6940 - accuracy:
0.4914 - val_loss: 0.6925 - val_accuracy: 0.5010
Epoch 27/200
63/63 [==============================] - 10s 166ms/step - loss: 0.6939 - accuracy:
0.4934 - val_loss: 0.6925 - val_accuracy: 0.5010
Epoch 28/200
63/63 [==============================] - 10s 164ms/step - loss: 0.6939 - accuracy:
0.4914 - val_loss: 0.6925 - val_accuracy: 0.5010
Epoch 29/200
63/63 [==============================] - 10s 164ms/step - loss: 0.6940 - accuracy:
0.4906 - val_loss: 0.6925 - val_accuracy: 0.5010
Epoch 30/200
63/63 [==============================] - 11s 176ms/step - loss: 0.6938 - accuracy:
0.4894 - val_loss: 0.6925 - val_accuracy: 0.5010
Epoch 31/200
63/63 [==============================] - 13s 213ms/step - loss: 0.6938 - accuracy:
0.4849 - val_loss: 0.6925 - val_accuracy: 0.5010
Epoch 32/200
63/63 [==============================] - 12s 191ms/step - loss: 0.6938 - accuracy:
0.4926 - val_loss: 0.6925 - val_accuracy: 0.5010
Epoch 33/200
63/63 [==============================] - 10s 165ms/step - loss: 0.6960 - accuracy:
0.4116 - val_loss: 0.6924 - val_accuracy: 0.5000
Epoch 34/200
63/63 [==============================] - 10s 167ms/step - loss: 0.6943 - accuracy:
0.4854 - val_loss: 0.6925 - val_accuracy: 0.5010
Epoch 35/200
63/63 [==============================] - 10s 165ms/step - loss: 0.6935 - accuracy:
0.4974 - val_loss: 0.6925 - val_accuracy: 0.5010
Epoch 36/200
63/63 [==============================] - 10s 163ms/step - loss: 0.6937 - accuracy:
0.4934 - val_loss: 0.6924 - val_accuracy: 0.5010
Epoch 37/200
63/63 [==============================] - 12s 196ms/step - loss: 0.6940 - accuracy:
```

```
0.4826 - val_loss: 0.6924 - val_accuracy: 0.5010
Epoch 38/200
63/63 [==============================] - 13s 204ms/step - loss: 0.6936 - accuracy:
0.4879 - val_loss: 0.6924 - val_accuracy: 0.5010
Epoch 39/200
63/63 [==============================] - 10s 164ms/step - loss: 0.6938 - accuracy:
0.4791 - val_loss: 0.6924 - val_accuracy: 0.5010
Epoch 40/200
63/63 [==============================] - 10s 157ms/step - loss: 0.6936 - accuracy:
0.4879 - val_loss: 0.6924 - val_accuracy: 0.5010
Epoch 41/200
63/63 [==============================] - 11s 174ms/step - loss: 0.7011 - accuracy:
0.3448 - val_loss: 0.6925 - val_accuracy: 0.5050
Epoch 42/200
63/63 [==============================] - 10s 154ms/step - loss: 0.6975 - accuracy:
0.3786 - val_loss: 0.6922 - val_accuracy: 0.5000
Epoch 43/200
63/63 [==============================] - 13s 204ms/step - loss: 0.6956 - accuracy:
0.4706 - val_loss: 0.6921 - val_accuracy: 0.5000
Epoch 44/200
63/63 [==============================] - 12s 200ms/step - loss: 0.7044 - accuracy:
0.2521 - val_loss: 0.6928 - val_accuracy: 0.5350
Epoch 45/200
63/63 [==============================] - 10s 169ms/step - loss: 0.6978 - accuracy:
0.4079 - val_loss: 0.6925 - val_accuracy: 0.5030
Epoch 46/200
63/63 [==============================] - 10s 166ms/step - loss: 0.6961 - accuracy:
0.3993 - val_loss: 0.6921 - val_accuracy: 0.5000
Epoch 47/200
63/63 [==============================] - 11s 172ms/step - loss: 0.7008 - accuracy:
0.3696 - val_loss: 0.6929 - val_accuracy: 0.5000
Epoch 48/200
63/63 [==============================] - 10s 167ms/step - loss: 0.7006 - accuracy:
0.4006 - val_loss: 0.6926 - val_accuracy: 0.5010
Epoch 49/200
63/63 [==============================] - 12s 198ms/step - loss: 0.6944 - accuracy:
0.4756 - val_loss: 0.6928 - val_accuracy: 0.5060
Epoch 50/200
63/63 [==============================] - 13s 211ms/step - loss: 0.6935 - accuracy:
0.4861 - val_loss: 0.6928 - val_accuracy: 0.5090
Epoch 51/200
63/63 [==============================] - 10s 156ms/step - loss: 0.6935 - accuracy:
0.4859 - val_loss: 0.6927 - val_accuracy: 0.5110
Epoch 52/200
63/63 [==============================] - 10s 168ms/step - loss: 0.6934 - accuracy:
0.4926 - val_loss: 0.6927 - val_accuracy: 0.5120
Epoch 53/200
63/63 [==============================] - 10s 169ms/step - loss: 0.6934 - accuracy:
0.4941 - val_loss: 0.6926 - val_accuracy: 0.5140
Epoch 54/200
63/63 [==============================] - 10s 167ms/step - loss: 0.6933 - accuracy:
0.4964 - val_loss: 0.6926 - val_accuracy: 0.5220
Epoch 55/200
63/63 [==============================] - 14s 226ms/step - loss: 0.6933 - accuracy:
0.4949 - val_loss: 0.6925 - val_accuracy: 0.5270
Epoch 56/200
```

```
63/63 [==============================] - 12s 190ms/step - loss: 0.6933 - accuracy:
0.4991 - val_loss: 0.6924 - val_accuracy: 0.5260
Epoch 57/200
63/63 [==============================] - 11s 172ms/step - loss: 0.6932 - accuracy:
0.5009 - val_loss: 0.6923 - val_accuracy: 0.5300
Epoch 58/200
63/63 [==============================] - 10s 166ms/step - loss: 0.6932 - accuracy:
0.4979 - val_loss: 0.6923 - val_accuracy: 0.5330
Epoch 59/200
63/63 [==============================] - 11s 170ms/step - loss: 0.6931 - accuracy:
0.4989 - val_loss: 0.6921 - val_accuracy: 0.5370
Epoch 60/200
63/63 [==============================] - 11s 169ms/step - loss: 0.6931 - accuracy:
0.5036 - val_loss: 0.6920 - val_accuracy: 0.5420
Epoch 61/200
63/63 [==============================] - 12s 198ms/step - loss: 0.6930 - accuracy:
0.5074 - val_loss: 0.6917 - val_accuracy: 0.5470
Epoch 62/200
63/63 [==============================] - 13s 211ms/step - loss: 0.6931 - accuracy:
0.4956 - val_loss: 0.6917 - val_accuracy: 0.5510
Epoch 63/200
63/63 [==============================] - 10s 169ms/step - loss: 0.6928 - accuracy:
0.5119 - val_loss: 0.6910 - val_accuracy: 0.5480
Epoch 64/200
63/63 [==============================] - 11s 171ms/step - loss: 0.6931 - accuracy:
0.5104 - val_loss: 0.6913 - val_accuracy: 0.5640
Epoch 65/200
63/63 [==============================] - 11s 172ms/step - loss: 0.6929 - accuracy:
0.5051 - val_loss: 0.6913 - val_accuracy: 0.5430
Epoch 66/200
63/63 [==============================] - 10s 168ms/step - loss: 0.6926 - accuracy:
0.5121 - val_loss: 0.6895 - val_accuracy: 0.5470
Epoch 67/200
63/63 [==============================] - 11s 184ms/step - loss: 0.6932 - accuracy:
0.4954 - val_loss: 0.6908 - val_accuracy: 0.5790
Epoch 68/200
63/63 [==============================] - 13s 203ms/step - loss: 0.6913 - accuracy:
0.5294 - val_loss: 0.6875 - val_accuracy: 0.5410
Epoch 69/200
63/63 [==============================] - 10s 165ms/step - loss: 0.6947 - accuracy:
0.4836 - val_loss: 0.6905 - val_accuracy: 0.5700
Epoch 70/200
63/63 [==============================] - 10s 168ms/step - loss: 0.6919 - accuracy:
0.5179 - val_loss: 0.6883 - val_accuracy: 0.5850
Epoch 71/200
63/63 [==============================] - 11s 181ms/step - loss: 0.6920 - accuracy:
0.5106 - val_loss: 0.6894 - val_accuracy: 0.5500
Epoch 72/200
63/63 [==============================] - 10s 164ms/step - loss: 0.6924 - accuracy:
0.5126 - val_loss: 0.6868 - val_accuracy: 0.5750
Epoch 73/200
63/63 [==============================] - 12s 190ms/step - loss: 0.6923 - accuracy:
0.5131 - val_loss: 0.6880 - val_accuracy: 0.5770
Epoch 74/200
63/63 [==============================] - 13s 203ms/step - loss: 0.6896 - accuracy:
0.5419 - val_loss: 0.6835 - val_accuracy: 0.5810
```

```
Epoch 75/200
63/63 [==============================] - 10s 163ms/step - loss: 0.6927 - accuracy:
0.5164 - val_loss: 0.6882 - val_accuracy: 0.5690
Epoch 76/200
63/63 [==============================] - 10s 168ms/step - loss: 0.6910 - accuracy:
0.5324 - val_loss: 0.6850 - val_accuracy: 0.5940
Epoch 77/200
63/63 [==============================] - 11s 170ms/step - loss: 0.6882 - accuracy:
0.5559 - val_loss: 0.6808 - val_accuracy: 0.5840
Epoch 78/200
63/63 [==============================] - 11s 175ms/step - loss: 0.6896 - accuracy:
0.5249 - val_loss: 0.6830 - val_accuracy: 0.5990
Epoch 79/200
63/63 [==============================] - 11s 181ms/step - loss: 0.6859 - accuracy:
0.5616 - val_loss: 0.6758 - val_accuracy: 0.5900
Epoch 80/200
63/63 [==============================] - 12s 191ms/step - loss: 0.6890 - accuracy:
0.5239 - val_loss: 0.6804 - val_accuracy: 0.5880
Epoch 81/200
63/63 [==============================] - 10s 166ms/step - loss: 0.6855 - accuracy:
0.5599 - val_loss: 0.6742 - val_accuracy: 0.5970
Epoch 82/200
63/63 [==============================] - 11s 169ms/step - loss: 0.6873 - accuracy:
0.5369 - val_loss: 0.6758 - val_accuracy: 0.5860
Epoch 83/200
63/63 [==============================] - 11s 169ms/step - loss: 0.6868 - accuracy:
0.5479 - val_loss: 0.6758 - val_accuracy: 0.6070
Epoch 84/200
63/63 [==============================] - 11s 179ms/step - loss: 0.6803 - accuracy:
0.5761 - val_loss: 0.6701 - val_accuracy: 0.5790
Epoch 85/200
63/63 [==============================] - 11s 172ms/step - loss: 0.6887 - accuracy:
0.5389 - val_loss: 0.6754 - val_accuracy: 0.6010
Epoch 86/200
63/63 [==============================] - 12s 194ms/step - loss: 0.6770 - accuracy:
0.5811 - val_loss: 0.6654 - val_accuracy: 0.5960
Epoch 87/200
63/63 [==============================] - 10s 166ms/step - loss: 0.6842 - accuracy:
0.5589 - val_loss: 0.6700 - val_accuracy: 0.6100
Epoch 88/200
63/63 [==============================] - 11s 171ms/step - loss: 0.6760 - accuracy:
0.5854 - val_loss: 0.6607 - val_accuracy: 0.5990
Epoch 89/200
63/63 [==============================] - 11s 170ms/step - loss: 0.6808 - accuracy:
0.5659 - val_loss: 0.6646 - val_accuracy: 0.6160
Epoch 90/200
63/63 [==============================] - 11s 179ms/step - loss: 0.6732 - accuracy:
0.5834 - val_loss: 0.6582 - val_accuracy: 0.6040
Epoch 91/200
63/63 [==============================] - 12s 190ms/step - loss: 0.6769 - accuracy:
0.5751 - val_loss: 0.6605 - val_accuracy: 0.6110
Epoch 92/200
63/63 [==============================] - 12s 194ms/step - loss: 0.6756 - accuracy:
0.5729 - val_loss: 0.6670 - val_accuracy: 0.5990
Epoch 93/200
63/63 [==============================] - 10s 165ms/step - loss: 0.6786 - accuracy:
```

```
0.6119 - val_loss: 1.1716 - val_accuracy: 0.5040
Epoch 94/200
63/63 [==============================] - 10s 166ms/step - loss: 0.8391 - accuracy:
0.4944 - val_loss: 0.6687 - val_accuracy: 0.5730
Epoch 95/200
63/63 [==============================] - 10s 168ms/step - loss: 0.6880 - accuracy:
0.5649 - val_loss: 0.6628 - val_accuracy: 0.6350
Epoch 96/200
63/63 [==============================] - 11s 181ms/step - loss: 0.6633 - accuracy:
0.6212 - val_loss: 0.6538 - val_accuracy: 0.6350
Epoch 97/200
63/63 [==============================] - 11s 176ms/step - loss: 0.6693 - accuracy:
0.5969 - val_loss: 0.6528 - val_accuracy: 0.6400
Epoch 98/200
63/63 [==============================] - 12s 187ms/step - loss: 0.6628 - accuracy:
0.6202 - val_loss: 0.6454 - val_accuracy: 0.6330
Epoch 99/200
63/63 [==============================] - 10s 165ms/step - loss: 0.6644 - accuracy:
0.6069 - val_loss: 0.6436 - val_accuracy: 0.6340
Epoch 100/200
63/63 [==============================] - 10s 167ms/step - loss: 0.6609 - accuracy:
0.6097 - val_loss: 0.6374 - val_accuracy: 0.6310
Epoch 101/200
63/63 [==============================] - 10s 169ms/step - loss: 0.6598 - accuracy:
0.6057 - val_loss: 0.6368 - val_accuracy: 0.6350
Epoch 102/200
63/63 [==============================] - 11s 176ms/step - loss: 0.6572 - accuracy:
0.6169 - val_loss: 0.6337 - val_accuracy: 0.6360
Epoch 103/200
63/63 [==============================] - 11s 175ms/step - loss: 0.6579 - accuracy:
0.6074 - val_loss: 0.6286 - val_accuracy: 0.6400
Epoch 104/200
63/63 [==============================] - 12s 194ms/step - loss: 0.6553 - accuracy:
0.6107 - val_loss: 0.6294 - val_accuracy: 0.6290
Epoch 105/200
63/63 [==============================] - 10s 165ms/step - loss: 0.6513 - accuracy:
0.6177 - val_loss: 0.6531 - val_accuracy: 0.6060
Epoch 106/200
63/63 [==============================] - 10s 168ms/step - loss: 0.6722 - accuracy:
0.5936 - val_loss: 0.6280 - val_accuracy: 0.6350
Epoch 107/200
63/63 [==============================] - 10s 168ms/step - loss: 0.6503 - accuracy:
0.6217 - val_loss: 0.6256 - val_accuracy: 0.6300
Epoch 108/200
63/63 [==============================] - 12s 191ms/step - loss: 0.6507 - accuracy:
0.6187 - val_loss: 0.6187 - val_accuracy: 0.6460
Epoch 109/200
63/63 [==============================] - 11s 181ms/step - loss: 0.6446 - accuracy:
0.6202 - val_loss: 0.6199 - val_accuracy: 0.6380
Epoch 110/200
63/63 [==============================] - 12s 197ms/step - loss: 0.6499 - accuracy:
0.6132 - val_loss: 0.6166 - val_accuracy: 0.6570
Epoch 111/200
63/63 [==============================] - 10s 167ms/step - loss: 0.6424 - accuracy:
0.6249 - val_loss: 0.6094 - val_accuracy: 0.6510
Epoch 112/200
```

```
63/63 [==============================] - 11s 169ms/step - loss: 0.6360 - accuracy:
0.6362 - val_loss: 0.6071 - val_accuracy: 0.6280
Epoch 113/200
63/63 [==============================] - 11s 170ms/step - loss: 0.6362 - accuracy:
0.6547 - val_loss: 0.8882 - val_accuracy: 0.5480
Epoch 114/200
63/63 [==============================] - 11s 178ms/step - loss: 0.7165 - accuracy:
0.5769 - val_loss: 0.6216 - val_accuracy: 0.6450
Epoch 115/200
63/63 [==============================] - 12s 190ms/step - loss: 0.6477 - accuracy:
0.6352 - val_loss: 0.6125 - val_accuracy: 0.6560
Epoch 116/200
63/63 [==============================] - 14s 224ms/step - loss: 0.6295 - accuracy:
0.6419 - val_loss: 0.6046 - val_accuracy: 0.6580
Epoch 117/200
63/63 [==============================] - 10s 159ms/step - loss: 0.6322 - accuracy:
0.6379 - val_loss: 0.6010 - val_accuracy: 0.6590
Epoch 118/200
63/63 [==============================] - 11s 177ms/step - loss: 0.6310 - accuracy:
0.6339 - val_loss: 0.5990 - val_accuracy: 0.6590
Epoch 119/200
63/63 [==============================] - 11s 181ms/step - loss: 0.6288 - accuracy:
0.6434 - val_loss: 0.5955 - val_accuracy: 0.6570
Epoch 120/200
63/63 [==============================] - 11s 173ms/step - loss: 0.6282 - accuracy:
0.6382 - val_loss: 0.6019 - val_accuracy: 0.6700
Epoch 121/200
63/63 [==============================] - 11s 173ms/step - loss: 0.6229 - accuracy:
0.6477 - val_loss: 0.6024 - val_accuracy: 0.6640
Epoch 122/200
63/63 [==============================] - 12s 189ms/step - loss: 0.6282 - accuracy:
0.6304 - val_loss: 0.6059 - val_accuracy: 0.6590
Epoch 123/200
63/63 [==============================] - 10s 167ms/step - loss: 0.6261 - accuracy:
0.6394 - val_loss: 0.5915 - val_accuracy: 0.6590
Epoch 124/200
63/63 [==============================] - 11s 173ms/step - loss: 0.6174 - accuracy:
0.6562 - val_loss: 0.6069 - val_accuracy: 0.6300
Epoch 125/200
63/63 [==============================] - 11s 176ms/step - loss: 0.6147 - accuracy:
0.6594 - val_loss: 0.6454 - val_accuracy: 0.6280
Epoch 126/200
63/63 [==============================] - 11s 183ms/step - loss: 0.6438 - accuracy:
0.6219 - val_loss: 0.5869 - val_accuracy: 0.6720
Epoch 127/200
63/63 [==============================] - 11s 169ms/step - loss: 0.6125 - accuracy:
0.6579 - val_loss: 0.5863 - val_accuracy: 0.6740
Epoch 128/200
63/63 [==============================] - 12s 191ms/step - loss: 0.6110 - accuracy:
0.6599 - val_loss: 0.5863 - val_accuracy: 0.6750
Epoch 129/200
63/63 [==============================] - 11s 177ms/step - loss: 0.6122 - accuracy:
0.6629 - val_loss: 0.5861 - val_accuracy: 0.6850
Epoch 130/200
63/63 [==============================] - 10s 155ms/step - loss: 0.6123 - accuracy:
0.6527 - val_loss: 0.5926 - val_accuracy: 0.6730
```

```
Epoch 131/200
63/63 [==============================] - 11s 174ms/step - loss: 0.6158 - accuracy:
0.6572 - val_loss: 0.5849 - val_accuracy: 0.6890
Epoch 132/200
63/63 [==============================] - 11s 183ms/step - loss: 0.6142 - accuracy:
0.6522 - val_loss: 0.5813 - val_accuracy: 0.6840
Epoch 133/200
63/63 [==============================] - 11s 171ms/step - loss: 0.6052 - accuracy:
0.6602 - val_loss: 0.5850 - val_accuracy: 0.6900
Epoch 134/200
63/63 [==============================] - 12s 192ms/step - loss: 0.6058 - accuracy:
0.6642 - val_loss: 0.5886 - val_accuracy: 0.6780
Epoch 135/200
63/63 [==============================] - 10s 168ms/step - loss: 0.6148 - accuracy:
0.6534 - val_loss: 0.5970 - val_accuracy: 0.6640
Epoch 136/200
63/63 [==============================] - 11s 169ms/step - loss: 0.6130 - accuracy:
0.6474 - val_loss: 0.5952 - val_accuracy: 0.6630
Epoch 137/200
63/63 [==============================] - 11s 175ms/step - loss: 0.6063 - accuracy:
0.6717 - val_loss: 0.6739 - val_accuracy: 0.6240
Epoch 138/200
63/63 [==============================] - 12s 191ms/step - loss: 0.6219 - accuracy:
0.6519 - val_loss: 0.5819 - val_accuracy: 0.6810
Epoch 139/200
63/63 [==============================] - 10s 166ms/step - loss: 0.6047 - accuracy:
0.6697 - val_loss: 0.5743 - val_accuracy: 0.6940
Epoch 140/200
63/63 [==============================] - 12s 197ms/step - loss: 0.5931 - accuracy:
0.6812 - val_loss: 0.5762 - val_accuracy: 0.6870
Epoch 141/200
63/63 [==============================] - 11s 170ms/step - loss: 0.5998 - accuracy:
0.6729 - val_loss: 0.5723 - val_accuracy: 0.6910
Epoch 142/200
63/63 [==============================] - 11s 171ms/step - loss: 0.5909 - accuracy:
0.6797 - val_loss: 0.5745 - val_accuracy: 0.6890
Epoch 143/200
63/63 [==============================] - 12s 186ms/step - loss: 0.5982 - accuracy:
0.6712 - val_loss: 0.5710 - val_accuracy: 0.6930
Epoch 144/200
63/63 [==============================] - 11s 178ms/step - loss: 0.5890 - accuracy:
0.6857 - val_loss: 0.5729 - val_accuracy: 0.6890
Epoch 145/200
63/63 [==============================] - 11s 175ms/step - loss: 0.5991 - accuracy:
0.6754 - val_loss: 0.5771 - val_accuracy: 0.6960
Epoch 146/200
63/63 [==============================] - 12s 197ms/step - loss: 0.6009 - accuracy:
0.6749 - val_loss: 0.5829 - val_accuracy: 0.6830
Epoch 147/200
63/63 [==============================] - 11s 179ms/step - loss: 0.5978 - accuracy:
0.6714 - val_loss: 0.5778 - val_accuracy: 0.6850
Epoch 148/200
63/63 [==============================] - 10s 159ms/step - loss: 0.5931 - accuracy:
0.6764 - val_loss: 0.5905 - val_accuracy: 0.6700
Epoch 149/200
63/63 [==============================] - 11s 183ms/step - loss: 0.5942 - accuracy:
```

```
0.6769 - val_loss: 0.6264 - val_accuracy: 0.6440
Epoch 150/200
63/63 [==============================] - 11s 179ms/step - loss: 0.5970 - accuracy:
0.6759 - val_loss: 0.5972 - val_accuracy: 0.6630
Epoch 151/200
63/63 [==============================] - 11s 183ms/step - loss: 0.5930 - accuracy:
0.6684 - val_loss: 0.5753 - val_accuracy: 0.6930
Epoch 152/200
63/63 [==============================] - 13s 202ms/step - loss: 0.5802 - accuracy:
0.6812 - val_loss: 0.5697 - val_accuracy: 0.7010
Epoch 153/200
63/63 [==============================] - 11s 174ms/step - loss: 0.5827 - accuracy:
0.6857 - val_loss: 0.5691 - val_accuracy: 0.7000
Epoch 154/200
63/63 [==============================] - 11s 169ms/step - loss: 0.5866 - accuracy:
0.6749 - val_loss: 0.5674 - val_accuracy: 0.7020
Epoch 155/200
63/63 [==============================] - 12s 186ms/step - loss: 0.5788 - accuracy:
0.6854 - val_loss: 0.6029 - val_accuracy: 0.6640
Epoch 156/200
63/63 [==============================] - 11s 174ms/step - loss: 0.5860 - accuracy:
0.6804 - val_loss: 0.5937 - val_accuracy: 0.6730
Epoch 157/200
63/63 [==============================] - 11s 176ms/step - loss: 0.5880 - accuracy:
0.6774 - val_loss: 0.6182 - val_accuracy: 0.6580
Epoch 158/200
63/63 [==============================] - 12s 187ms/step - loss: 0.5902 - accuracy:
0.6774 - val_loss: 0.5679 - val_accuracy: 0.6980
Epoch 159/200
63/63 [==============================] - 11s 169ms/step - loss: 0.5768 - accuracy:
0.6902 - val_loss: 0.5655 - val_accuracy: 0.7040
Epoch 160/200
63/63 [==============================] - 11s 169ms/step - loss: 0.5779 - accuracy:
0.6932 - val_loss: 0.5630 - val_accuracy: 0.7080
Epoch 161/200
63/63 [==============================] - 11s 172ms/step - loss: 0.5738 - accuracy:
0.6914 - val_loss: 0.5637 - val_accuracy: 0.7060
Epoch 162/200
63/63 [==============================] - 11s 179ms/step - loss: 0.5785 - accuracy:
0.6812 - val_loss: 0.5650 - val_accuracy: 0.6970
Epoch 163/200
63/63 [==============================] - 11s 178ms/step - loss: 0.5809 - accuracy:
0.6822 - val_loss: 0.5722 - val_accuracy: 0.6960
Epoch 164/200
63/63 [==============================] - 12s 196ms/step - loss: 0.5909 - accuracy:
0.6769 - val_loss: 0.5626 - val_accuracy: 0.7020
Epoch 165/200
63/63 [==============================] - 11s 171ms/step - loss: 0.5764 - accuracy:
0.6862 - val_loss: 0.5609 - val_accuracy: 0.7000
Epoch 166/200
63/63 [==============================] - 11s 170ms/step - loss: 0.5772 - accuracy:
0.6879 - val_loss: 0.5580 - val_accuracy: 0.7040
Epoch 167/200
63/63 [==============================] - 11s 181ms/step - loss: 0.5715 - accuracy:
0.6922 - val_loss: 0.5618 - val_accuracy: 0.6990
Epoch 168/200
```

```
63/63 [==============================] - 10s 165ms/step - loss: 0.5742 - accuracy:
0.6924 - val_loss: 0.5666 - val_accuracy: 0.6950
Epoch 169/200
63/63 [==============================] - 11s 173ms/step - loss: 0.5788 - accuracy:
0.6857 - val_loss: 0.5638 - val_accuracy: 0.6980
Epoch 170/200
63/63 [==============================] - 11s 184ms/step - loss: 0.5761 - accuracy:
0.6912 - val_loss: 0.5609 - val_accuracy: 0.6970
Epoch 171/200
63/63 [==============================] - 10s 167ms/step - loss: 0.5768 - accuracy:
0.6864 - val_loss: 0.5594 - val_accuracy: 0.6960
Epoch 172/200
63/63 [==============================] - 11s 173ms/step - loss: 0.5747 - accuracy:
0.6969 - val_loss: 0.5636 - val_accuracy: 0.6960
Epoch 173/200
63/63 [==============================] - 11s 183ms/step - loss: 0.5808 - accuracy:
0.6777 - val_loss: 0.5548 - val_accuracy: 0.7050
Epoch 174/200
63/63 [==============================] - 10s 165ms/step - loss: 0.5642 - accuracy:
0.7009 - val_loss: 0.5593 - val_accuracy: 0.7070
Epoch 175/200
63/63 [==============================] - 11s 171ms/step - loss: 0.5696 - accuracy:
0.6884 - val_loss: 0.5591 - val_accuracy: 0.7000
Epoch 176/200
63/63 [==============================] - 13s 202ms/step - loss: 0.5678 - accuracy:
0.6894 - val_loss: 0.5536 - val_accuracy: 0.7120
Epoch 177/200
63/63 [==============================] - 10s 158ms/step - loss: 0.5632 - accuracy:
0.6959 - val_loss: 0.5535 - val_accuracy: 0.7080
Epoch 178/200
63/63 [==============================] - 10s 167ms/step - loss: 0.5636 - accuracy:
0.7027 - val_loss: 0.5521 - val_accuracy: 0.7060
Epoch 179/200
63/63 [==============================] - 11s 173ms/step - loss: 0.5646 - accuracy:
0.7009 - val_loss: 0.5596 - val_accuracy: 0.6980
Epoch 180/200
63/63 [==============================] - 11s 173ms/step - loss: 0.5654 - accuracy:
0.6964 - val_loss: 0.5606 - val_accuracy: 0.6960
Epoch 181/200
63/63 [==============================] - 11s 179ms/step - loss: 0.5668 - accuracy:
0.6927 - val_loss: 0.5608 - val_accuracy: 0.6980
Epoch 182/200
63/63 [==============================] - 11s 184ms/step - loss: 0.5706 - accuracy:
0.6944 - val_loss: 0.5527 - val_accuracy: 0.7050
Epoch 183/200
63/63 [==============================] - 11s 181ms/step - loss: 0.5647 - accuracy:
0.6957 - val_loss: 0.5530 - val_accuracy: 0.7030
Epoch 184/200
63/63 [==============================] - 10s 160ms/step - loss: 0.5624 - accuracy:
0.7009 - val_loss: 0.5488 - val_accuracy: 0.7080
Epoch 185/200
63/63 [==============================] - 11s 174ms/step - loss: 0.5598 - accuracy:
0.7007 - val_loss: 0.5638 - val_accuracy: 0.6970
Epoch 186/200
63/63 [==============================] - 11s 172ms/step - loss: 0.5713 - accuracy:
0.6917 - val_loss: 0.5634 - val_accuracy: 0.6970
```

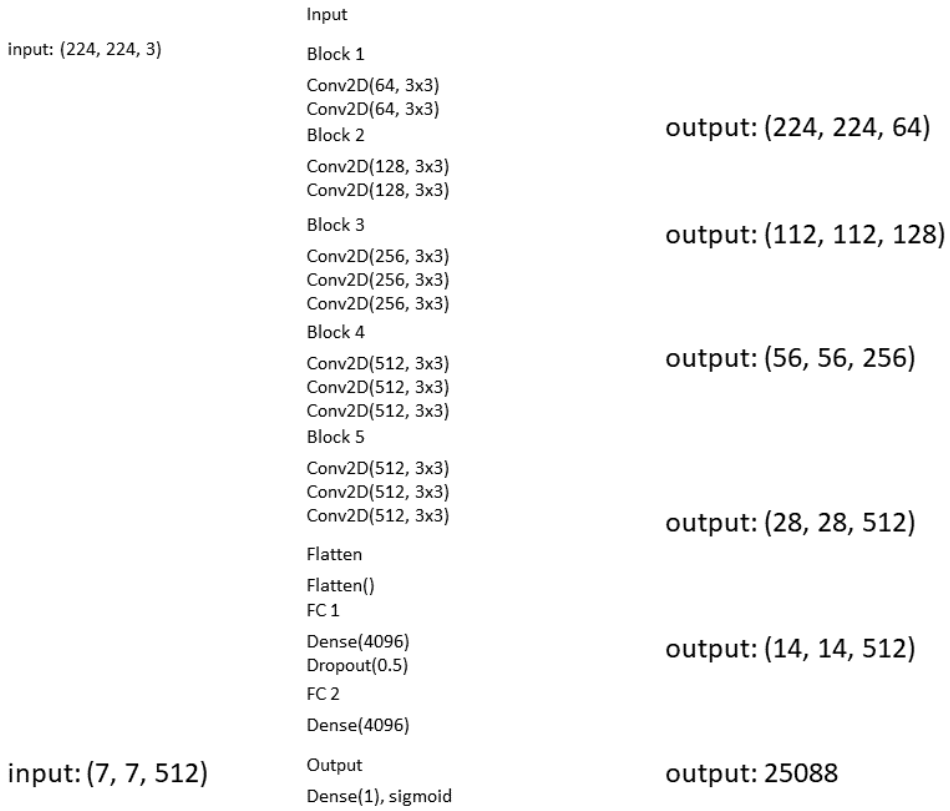
```
Epoch 187/200
63/63 [==============================] - 11s 174ms/step - loss: 0.5631 - accuracy:
0.6932 - val_loss: 0.5735 - val_accuracy: 0.6960
Epoch 188/200
63/63 [==============================] - 12s 194ms/step - loss: 0.5619 - accuracy:
0.6964 - val_loss: 0.6639 - val_accuracy: 0.6630
Epoch 189/200
63/63 [==============================] - 10s 160ms/step - loss: 0.5734 - accuracy:
0.6939 - val_loss: 0.5540 - val_accuracy: 0.7030
Epoch 190/200
63/63 [==============================] - 11s 179ms/step - loss: 0.5519 - accuracy:
0.7047 - val_loss: 0.5489 - val_accuracy: 0.7180
Epoch 191/200
63/63 [==============================] - 11s 174ms/step - loss: 0.5594 - accuracy:
0.7007 - val_loss: 0.5503 - val_accuracy: 0.7140
Epoch 192/200
63/63 [==============================] - 11s 179ms/step - loss: 0.5480 - accuracy:
0.7099 - val_loss: 0.5691 - val_accuracy: 0.6970
Epoch 193/200
63/63 [==============================] - 11s 185ms/step - loss: 0.5506 - accuracy:
0.7067 - val_loss: 0.5567 - val_accuracy: 0.7120
Epoch 194/200
63/63 [==============================] - 11s 183ms/step - loss: 0.5500 - accuracy:
0.7047 - val_loss: 0.5686 - val_accuracy: 0.7010
Epoch 195/200
63/63 [==============================] - 10s 166ms/step - loss: 0.5475 - accuracy:
0.7067 - val_loss: 0.6087 - val_accuracy: 0.6910
Epoch 196/200
63/63 [==============================] - 11s 173ms/step - loss: 0.5571 - accuracy:
0.6984 - val_loss: 0.5940 - val_accuracy: 0.6920
Epoch 197/200
63/63 [==============================] - 13s 209ms/step - loss: 0.5545 - accuracy:
0.7059 - val_loss: 0.5609 - val_accuracy: 0.7030
Epoch 198/200
63/63 [==============================] - 11s 170ms/step - loss: 0.5524 - accuracy:
0.7092 - val_loss: 0.5432 - val_accuracy: 0.7140
Epoch 199/200
63/63 [==============================] - 11s 183ms/step - loss: 0.5472 - accuracy:
0.7052 - val_loss: 0.5442 - val_accuracy: 0.7150
Epoch 200/200
63/63 [==============================] - ETA: 0s - loss: 0.5498 - accuracy: 0.7024Gu
ardados pesos FINALES en: pesos_guardados/pesos_finales_epoca_200.h5
63/63 [==============================] - 12s 200ms/step - loss: 0.5498 - accuracy:
0.7024 - val_loss: 0.5462 - val_accuracy: 0.7080
```

2. Descripción gráfica de la red VGG16

El modelo VGG16 orignal:

In [70]:
```
from IPython.display import Image
Image(filename="Diapositiva1.PNG")
```

Out[70]:



Mi modelo VGG 16 modificado para posibilitar su entrenamiento:

In [72]:
```
model.summary()
```

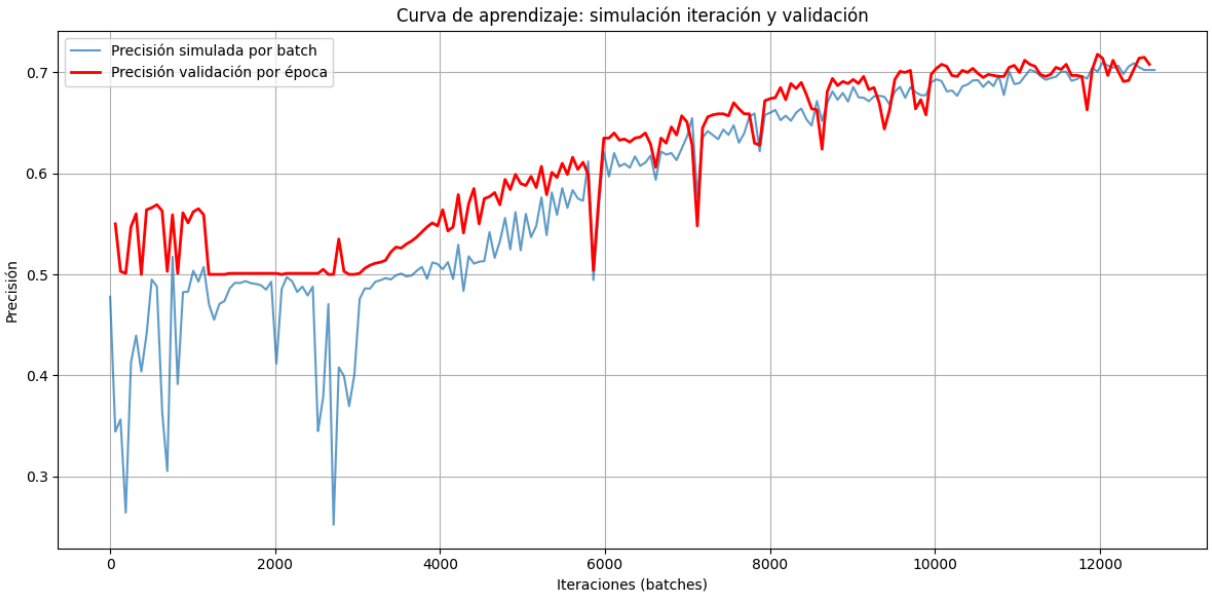
```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 block1_conv1 (Conv2D)       (None, 128, 128, 32)      896

 block1_conv2 (Conv2D)       (None, 128, 128, 32)      9248

 block1_pool (MaxPooling2D)  (None, 64, 64, 32)        0

 conv2d_28 (Conv2D)          (None, 64, 64, 64)        18496

 conv2d_29 (Conv2D)          (None, 64, 64, 64)        36928

 max_pooling2d_14 (MaxPooli  (None, 32, 32, 64)        0
 ng2D)

 conv2d_30 (Conv2D)          (None, 32, 32, 128)       73856

 conv2d_31 (Conv2D)          (None, 32, 32, 128)       147584

 max_pooling2d_15 (MaxPooli  (None, 16, 16, 128)       0
 ng2D)

 flatten_3 (Flatten)         (None, 32768)             0

 dense_8 (Dense)             (None, 512)               16777728

 dropout_7 (Dropout)         (None, 512)               0

 dense_9 (Dense)             (None, 1)                 513

=================================================================
Total params: 17065249 (65.10 MB)
Trainable params: 17065249 (65.10 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

3. Representar en una sola figura la evolución de la precisión (curva de aprendizaje) durante el entrenamiento

A) La precisión en la submuestra de entrenamiento, para cada iteración.

In [81]:
```python
import matplotlib.pyplot as plt

# Datos
accuracy = history.history['accuracy']        # Precisión entrenamiento por época
val_accuracy = history.history['val_accuracy'] # Precisión validación por época
epochs = range(1, len(accuracy)+1)

# Número de batches por época (aproximado)
```

```
batches_per_epoch = steps_per_epoch

# Simulación de ieeración
simulated_iter_accuracy = []

for i in range(len(accuracy)-1):
    start_acc = accuracy[i]
    end_acc = accuracy[i+1]
    # Linearly interpolate between the two epochs
    interpolated = np.linspace(start_acc, end_acc, batches_per_epoch, endpoint=Fals
    simulated_iter_accuracy.extend(interpolated)

# Para la última época: mantenemos la última accuracy fija
simulated_iter_accuracy.extend([accuracy[-1]] * batches_per_epoch)

# Ejes de batch
batch_steps = range(len(simulated_batch_accuracy))
epoch_steps = [i * batches_per_epoch for i in epochs]

# Dibujamos
plt.figure(figsize=(12,6))

plt.plot(batch_steps, simulated_batch_accuracy, label='Precisión simulada por batch
plt.plot(epoch_steps, val_accuracy, label='Precisión validación por época', color='

plt.title('Curva de aprendizaje: simulación iteración y validación')
plt.xlabel('Iteraciones (batches)')
plt.ylabel('Precisión')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('curva_aprendizaje_simulada.png')
plt.show()
```

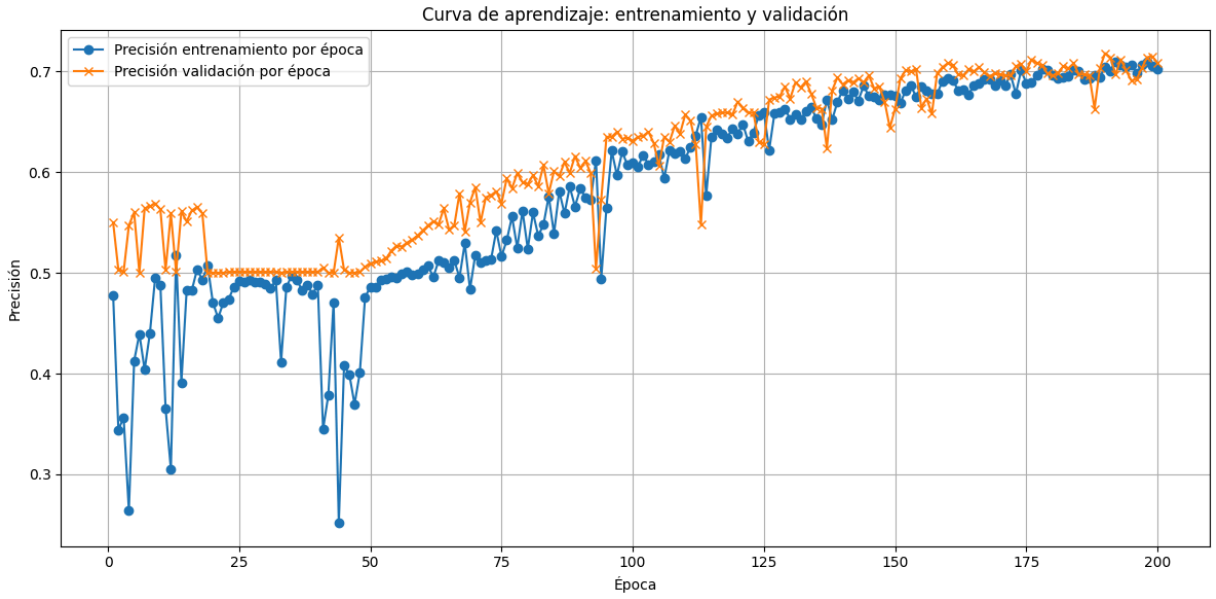


Como no he guardado la precisión por iteración la he simulado interpolando. En este gráfico podemos visualizar como el modelo tiene en general una baja precisión con grandes

oscilaciones. No se observa sobreajuste. que sería que la precisión por validación callese o se mantuviera estable mientras la precisión en el entramiento siguiese subiendo.

B) La precisión en la submuestra de test, para cada época.

In [79]:
```
accuracy = history.history['accuracy']        # Precisión en entrenamiento por époc
val_accuracy = history.history['val_accuracy'] # Precisión en validación por época
epochs = range(1, len(accuracy)+1)

plt.figure(figsize=(12,6))
plt.plot(epochs, accuracy, label='Precisión entrenamiento por época', marker='o')
plt.plot(epochs, val_accuracy, label='Precisión validación por época', marker='x')

plt.title('Curva de aprendizaje: entrenamiento y validación')
plt.xlabel('Época')
plt.ylabel('Precisión')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.savefig('curva_aprendizaje_epocas.png')
plt.show()
```



Como en el anterior gráfico el modelo en general tiene baja precisión y con muchas oscilaciones de esta. En las últimas épocas la precisión en el entrenamiento se acerca mucho a la de validación pudiendo ser un indicio de sobreajuste.

4. Visualizar núcleos y activaciones

In [63]:
```
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np

def preparar_imagen_perro(ruta_imagen):
    img = load_img(ruta_imagen, target_size=(128, 128))  # ← Ajusta si tu input es
```

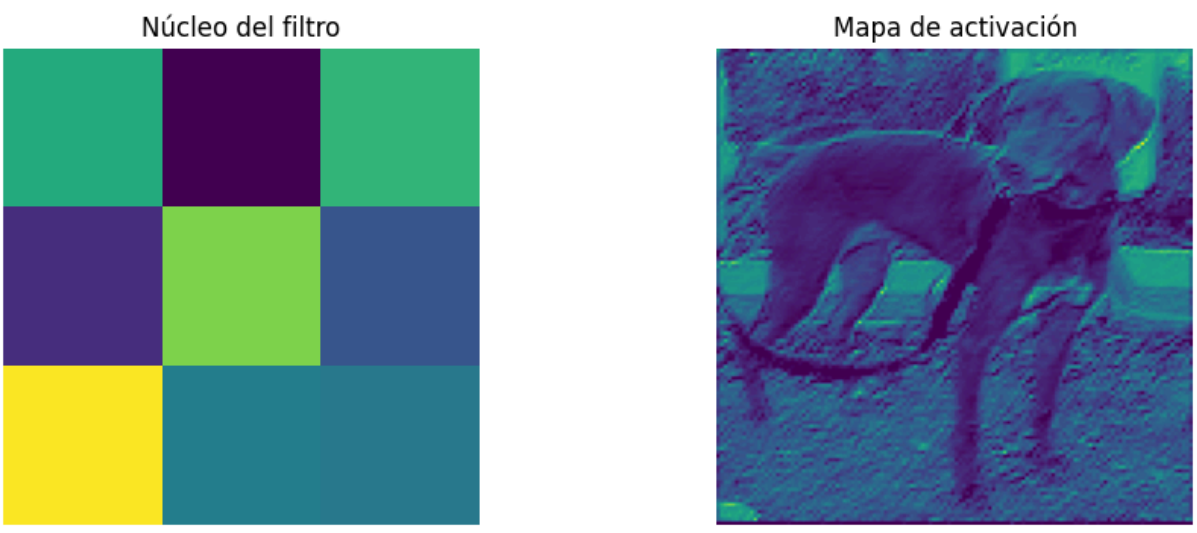
```
    x = img_to_array(img) / 255.0
    x = np.expand_dims(x, axis=0)
    return x
```

In [64]:
```
import matplotlib.pyplot as plt
from tensorflow.keras.models import Model

def visualizar_filtro_y_activacion(modelo, capa_nombre, imagen_tensor, filtro_idx,
    capa = modelo.get_layer(name=capa_nombre)
    submodelo = Model(inputs=modelo.input, outputs=capa.output)
    activacion = submodelo.predict(imagen_tensor)[0, :, :, filtro_idx]
    kernel = capa.get_weights()[0][:, :, :, filtro_idx]

    fig, axes = plt.subplots(1, 2, figsize=(10, 4))
    axes[0].imshow(kernel[:, :, 0], cmap="viridis")
    axes[0].set_title("Núcleo del filtro")
    axes[1].imshow(activacion, cmap="viridis")
    axes[1].set_title("Mapa de activación")
    for ax in axes: ax.axis("off")
    plt.suptitle(titulo)
    plt.tight_layout()
    plt.savefig(f"resultados/{titulo.replace(' ', '_')}.png")  # Guarda la figura
    plt.show()
```

A) El núcleo de la convolución y el correspondiente mapa de activación ante la imagen de un perro para el primer filtro de la primera capa convolucional al iniciarse el proceso de aprendizaje

In [73]:
```
imagen_tensor = preparar_imagen_perro("data/cats_and_dogs_filtered/train/dogs/dog_t

# Activación al inicio del aprendizaje (cargar pesos iniciales)
model.load_weights("pesos_guardados/pesos_iniciales_epoca_001.h5")
visualizar_filtro_y_activacion(model, "block1_conv1", imagen_tensor, 0, "inicio_blo
```

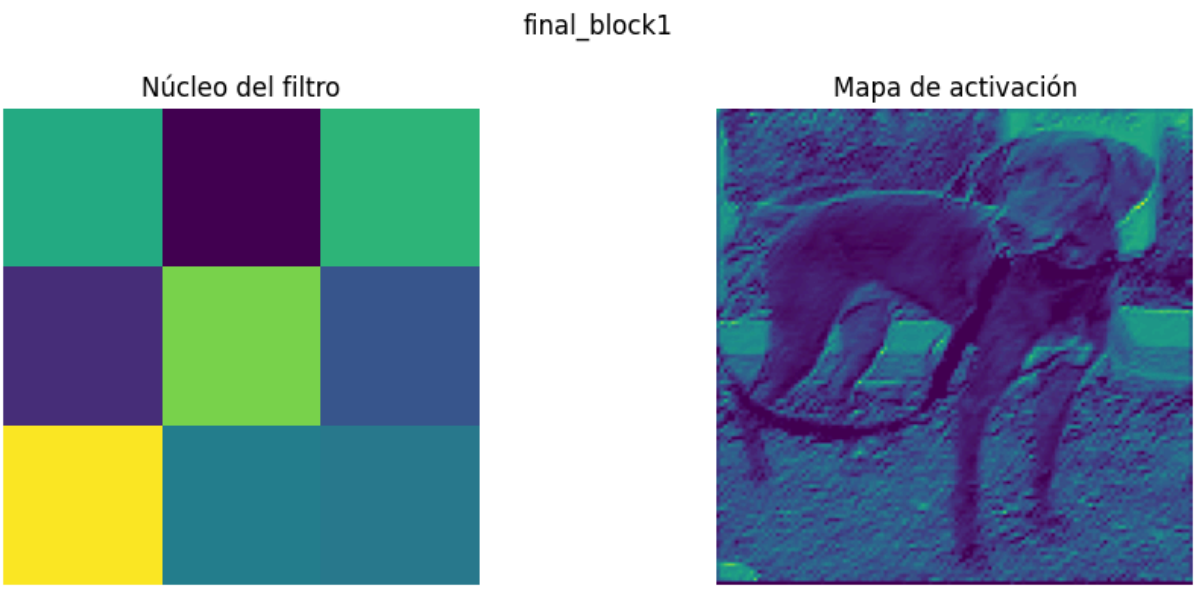
```
1/1 [==============================] - 0s 30ms/step
```



B) El núcleo de la convolución y el correspondiente mapa de activación ante la imagen del mismo perro para el primer filtro de la primera capa convolucional tras finalizar el proceso de aprendizaje

In [75]:
```
model.load_weights("pesos_guardados/pesos_finales_epoca_200.h5")

# Primer filtro de la primera capa tras el entrenamiento
visualizar_filtro_y_activacion(model, "block1_conv1", imagen_tensor, 0, "final_bloc
```

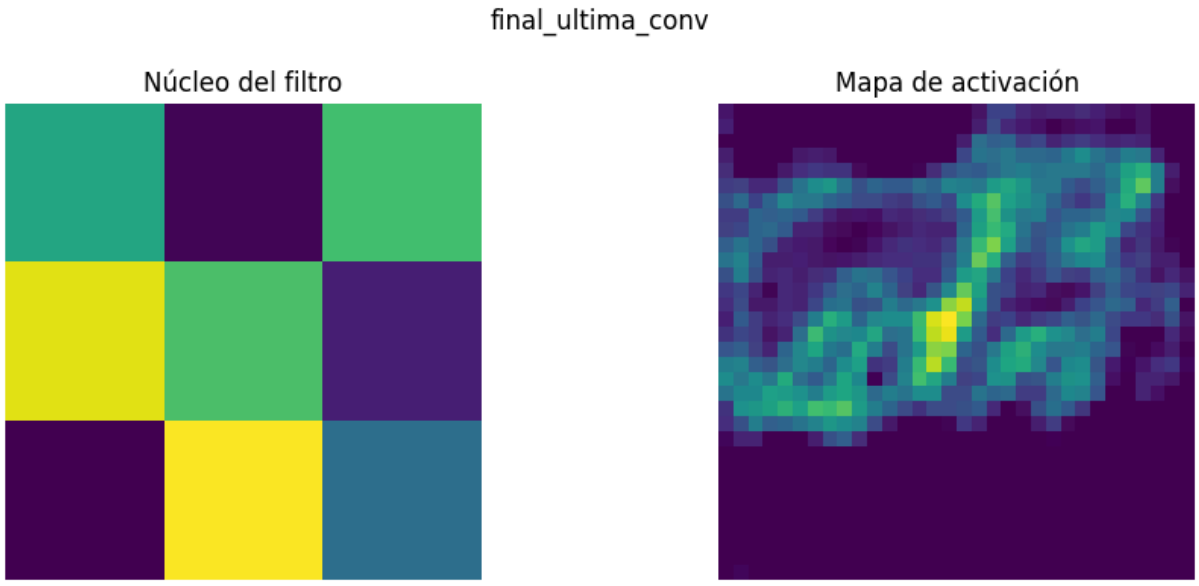
```
1/1 [==============================] - 0s 31ms/step
```



C) El núcleo de la convolución y el correspondiente mapa de activación ante la imagen del mismo perro para el primer filtro de la última capa convolucional tras finalizar el proceso de aprendizaje.

In [76]:
```
# Primer filtro de la última capa convolucional (aquí conv2d_4 es la última capa co
visualizar_filtro_y_activacion(model, "conv2d_31", imagen_tensor, 0, "final_ultima_
```

```
1/1 [==============================] - 0s 61ms/step
```



La red ve igual tanto en el primer bloque tanto al principio como al final de entrenamiento. En este primer bloque detecta partes más salientes de las imágenes. Por otro lado, en la última capa detecta formas más específicas, en nuestro caso parece una imagen bastante abstracta.

5. Sobreajuste

In [77]:
```
from sklearn.metrics import accuracy_score
from scipy.stats import chi2_contingency
import numpy as np

# Obtener verdaderas etiquetas
y_train_true = train_generator.classes
y_val_true = validation_generator.classes

# Predicciones
y_train_pred = (model.predict(train_generator) > 0.5).astype(int).flatten()
y_val_pred = (model.predict(validation_generator) > 0.5).astype(int).flatten()

# Aciertos y errores
train_correct = np.sum(y_train_pred == y_train_true)
train_incorrect = len(y_train_true) - train_correct

val_correct = np.sum(y_val_pred == y_val_true)
val_incorrect = len(y_val_true) - val_correct

# Tabla de contingencia
tabla = np.array([
    [train_correct, train_incorrect],
    [val_correct, val_incorrect]
])

# Chi-cuadrado
chi2, p, _, _ = chi2_contingency(tabla)
```

```
# Resultados
print("Tabla de contingencia:")
print(tabla)
print(f"\nChi² = {chi2:.3f}, p = {p:.4f}")

if p < 0.05:
    print("Existe evidencia estadística de sobreajuste.")
else:
    print("No hay evidencia clara de sobreajuste.")
```

```
63/63 [==============================] - 11s 167ms/step
16/16 [==============================] - 1s 39ms/step
Tabla de contingencia:
[[2924 1075]
 [ 708  292]]

Chi² = 2.049, p = 0.1523
No hay evidencia clara de sobreajuste.
```

Para detectar el sobre ajuste he utilizado una prueba Ji-cuadrado de independencia, creeando primeramente una tabala de contigencia

ANEXO Programas utilizados

He entrenado el modelo en Paperspace utilizando una máquina virtual A-4000. Asimismo, he utilizado power point para al arquitectura de la red VGG 16. Finalmente, he usado Chatgpt como consulta a la hora de programar

In [ ]:

In [1]:
```
!jupyter nbconvert --to webpdf CNN.ipynb
```

```
[NbConvertApp] Converting notebook CNN.ipynb to webpdf
[NbConvertApp] WARNING | Alternative text is missing on 6 image(s).
[NbConvertApp] Building PDF
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 719910 bytes to CNN.pdf
```