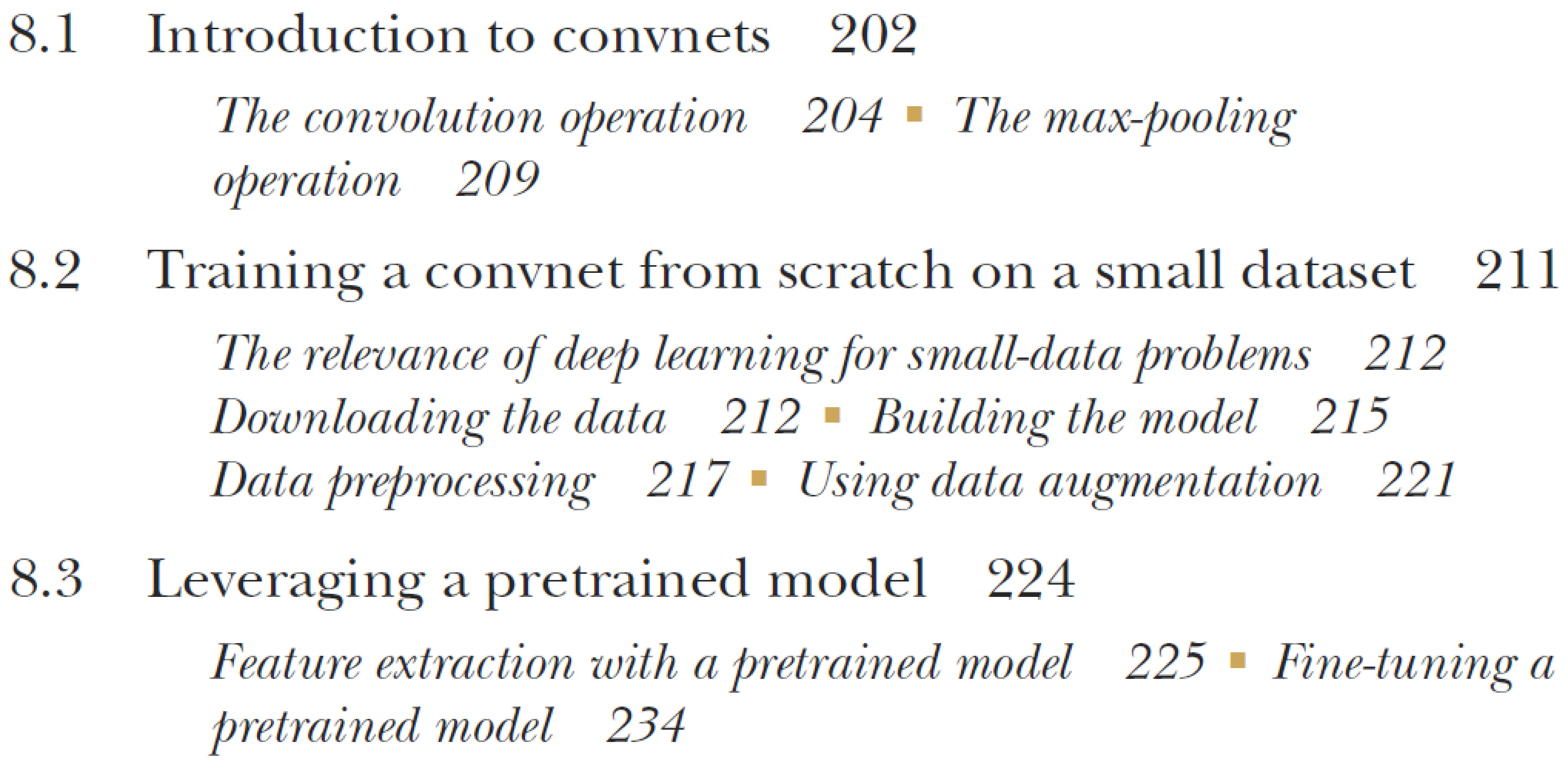
Convolutional Networks (ConvNets): Part II

November 3, 2022

ddebarr@uw.edu

http://cross-entropy.net/ml530/Deep_Learning_3.pdf

Deep Learning for Computer Vision



Instantiating a Small ConvNet

```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

Conv2D

- A convolution filter is the neuron of the convolution layer, deriving features by matching the filter to a subregion of the image [generating “local” (translation invariant) features]
- Most common: kernel_size = (3, 3) and strides = (1, 1)
- Output shape
  - padding = ‘valid’ (i.e. no padding)

    output_shape = math.ceil((input_shape - filter_size + 1) / strides)
  - padding = ‘same’

    output_shape = math.ceil(input_shape / strides)

    https://github.com/tensorflow/tensorflow/blob/v2.9.0/tensorflow/python/ops/nn_ops.py#L17

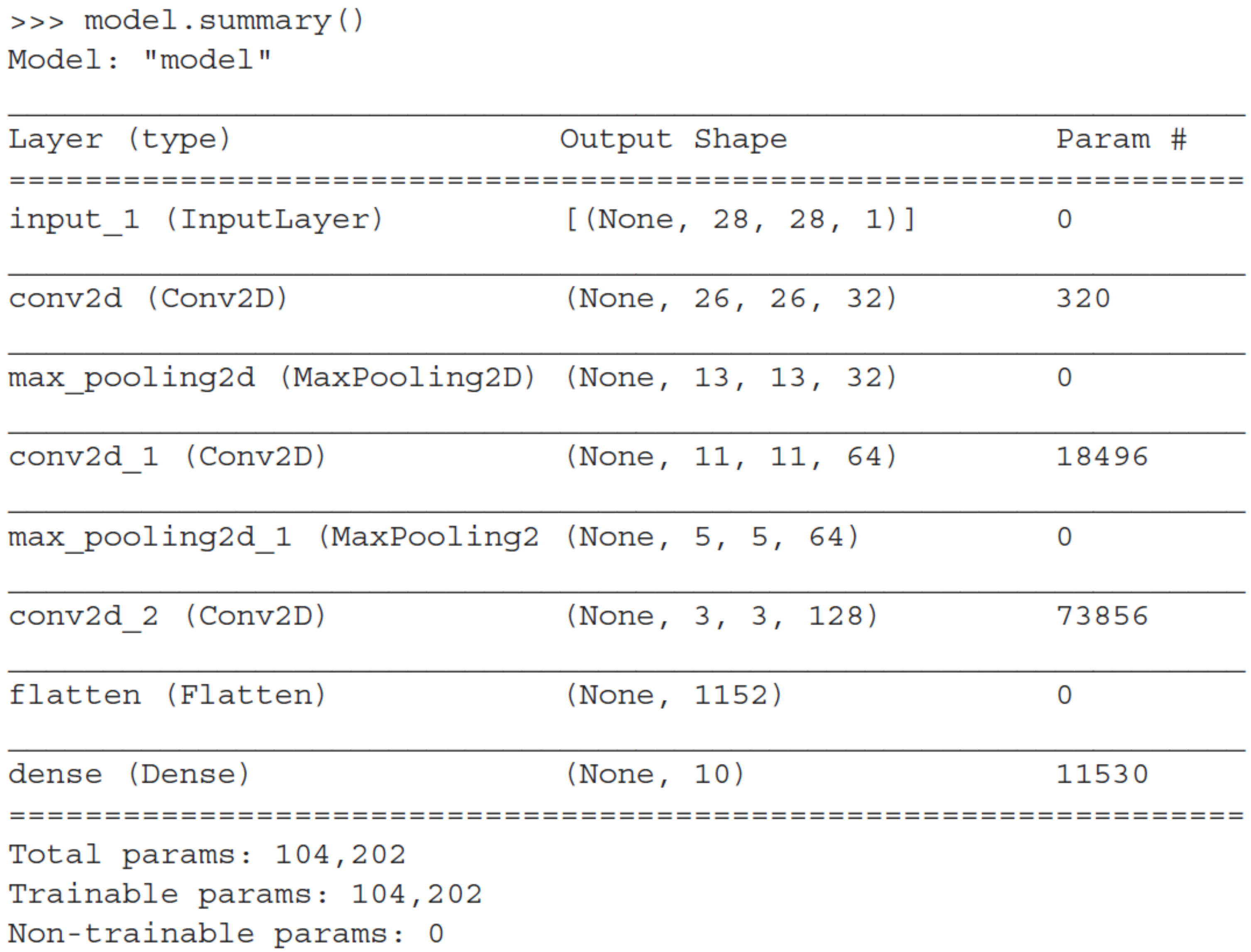
Pooling2D

- MaxPooling2D or AvgPooling2D
- Changes the resolution (height and width) of the feature maps, changing the scale of a picture element (pixel)
- Most common: pool_size = (2, 2) and strides = (2, 2)
- Output shape
    - padding = “valid” (the default)

      output_shape = math.floor((input_shape – pool_size) / strides) + 1
    - padding = “same”

      output_shape = math.floor((input_shape – 1) / strides) + 1

ConvNet Summary



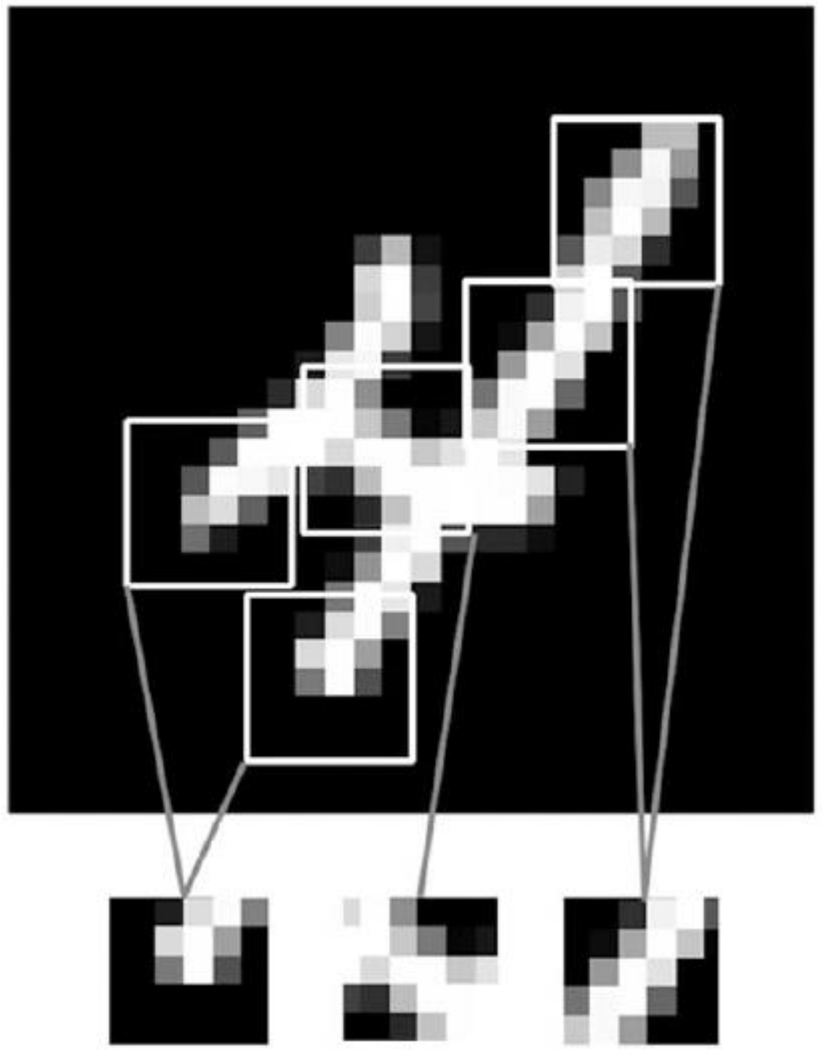
```
>>> model.summary()
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 128)         73856
_________________________________________________________________
flatten (Flatten)            (None, 1152)              0
_________________________________________________________________
dense (Dense)                (None, 10)                11530
=================================================================
Total params: 104,202
Trainable params: 104,202
Non-trainable params: 0
```

Training and Evaluating the ConvNet

```
from tensorflow.keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype("float32") / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype("float32") / 255
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

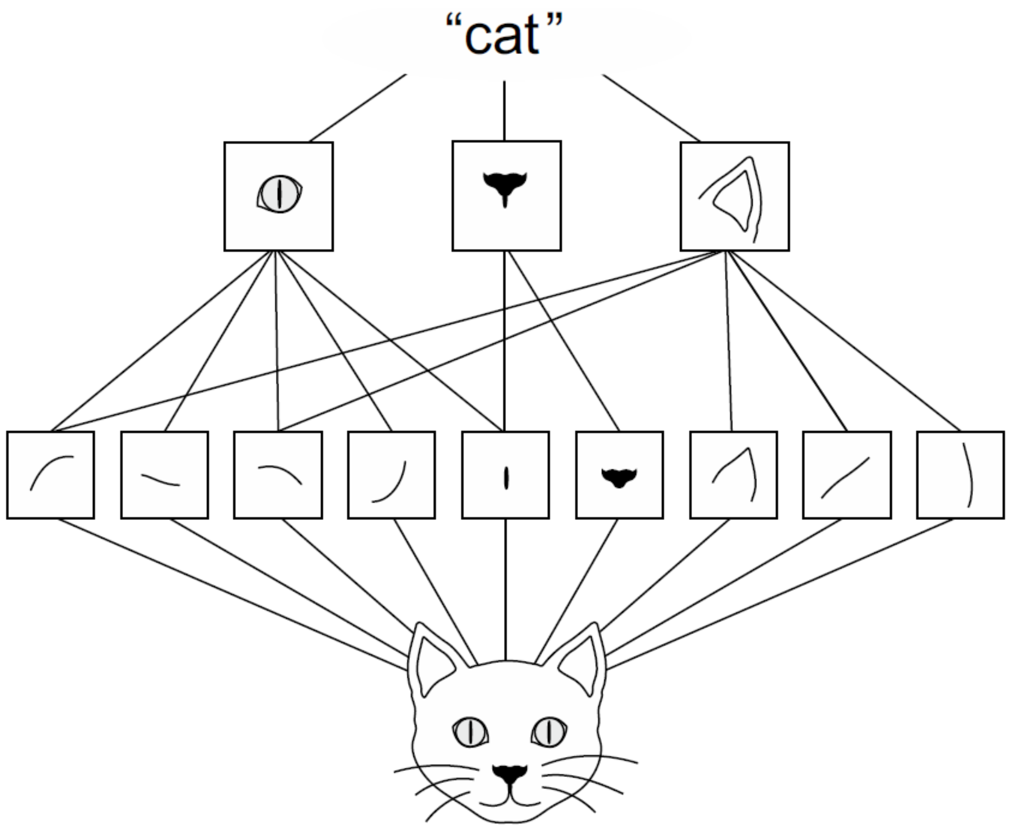
```
>>> test_loss, test_acc = model.evaluate(test_images, test_labels)
>>> print(f"Test accuracy: {test_acc:.3f}")
Test accuracy: 0.991
```

Images can be Broken into Local Patterns

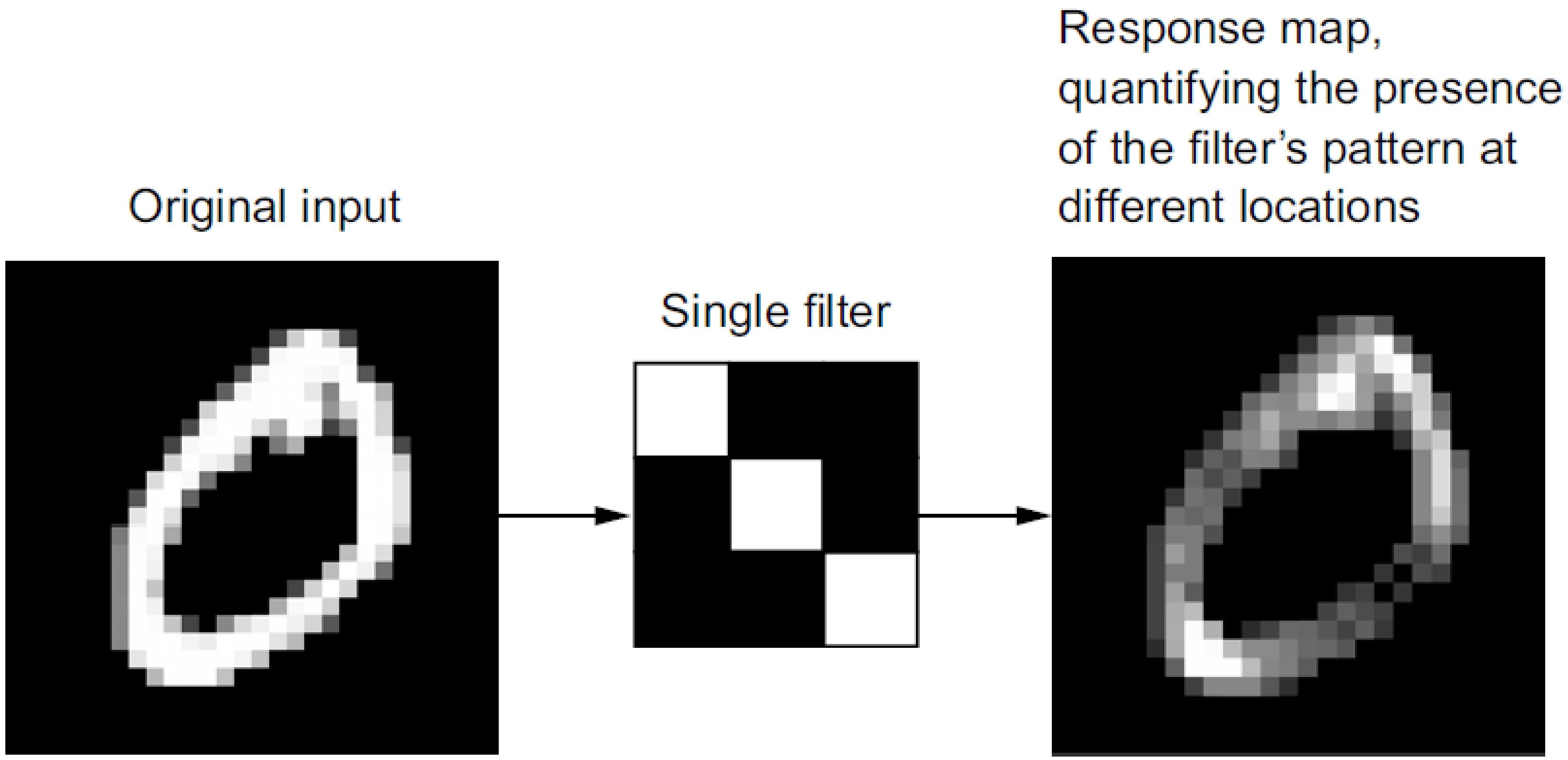


ConvNet Properties

- Learned patterns are translation-invariant
- Learned patterns can include spatial hierarchies of patterns



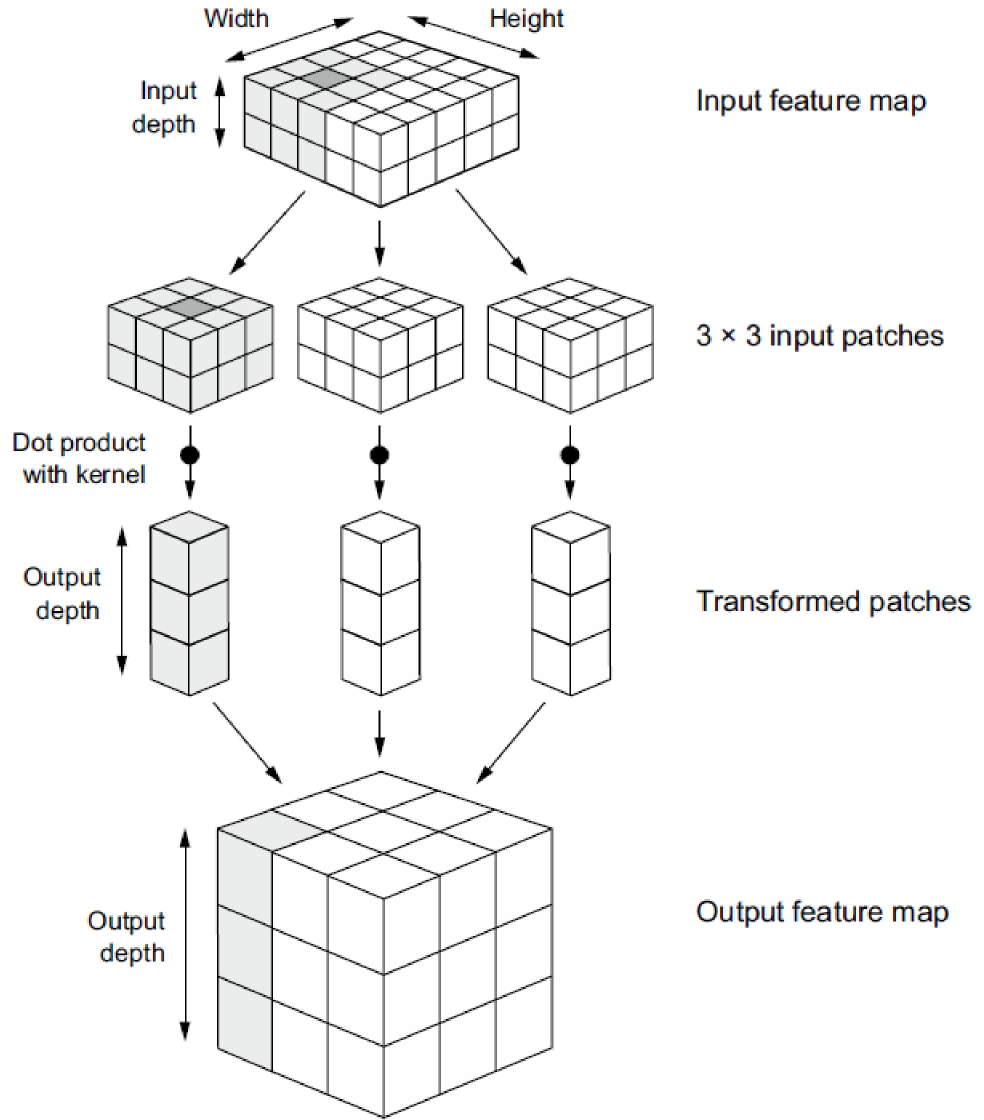
Response Map (aka Feature Map)


Original input
Single filter
Response map, quantifying the presence of the filter's pattern at different locations

Convolution Parameters

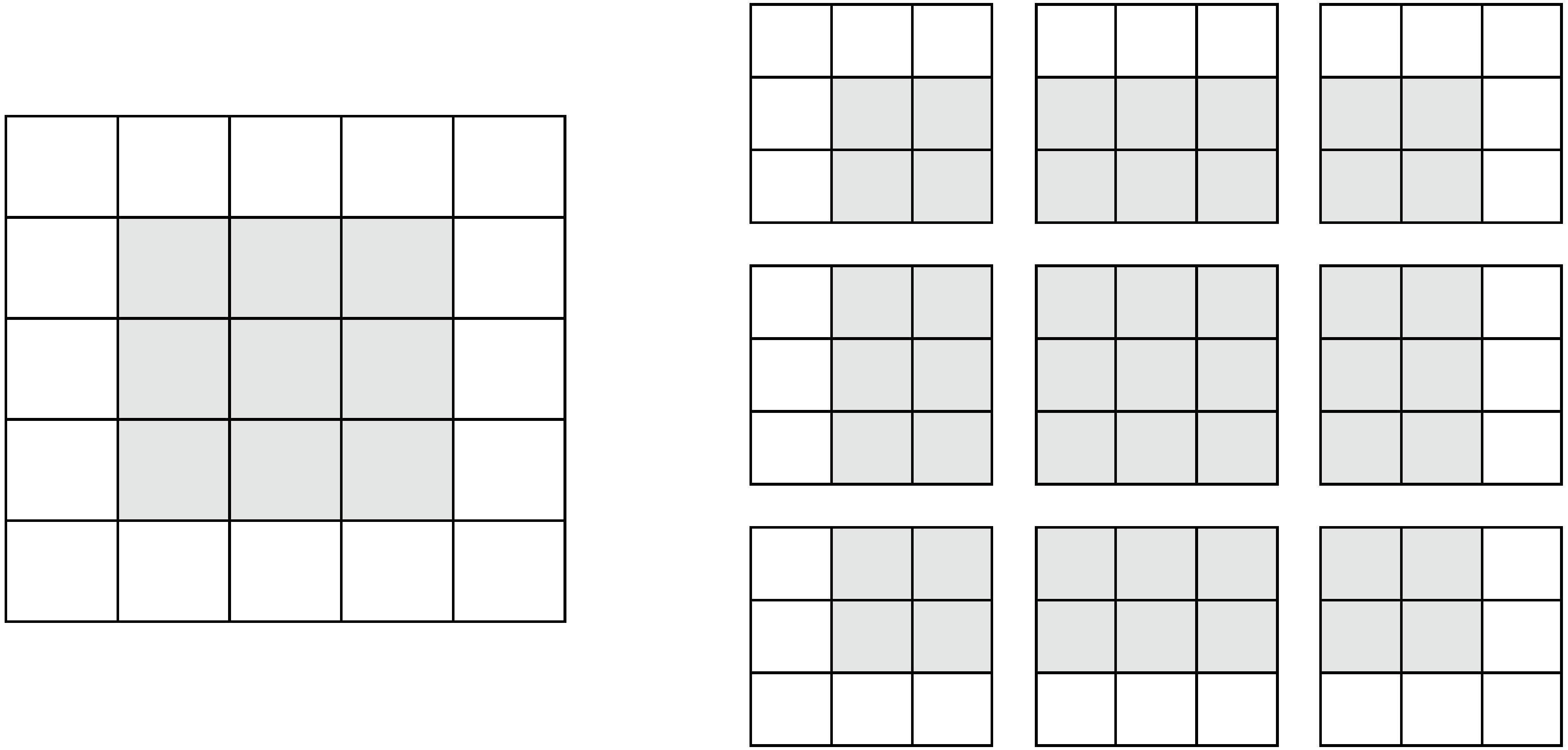
- Size of the patch used to generate a feature value
- Depth of the output feature map (one feature map per filter/kernel/neuron)

How Convolution Works

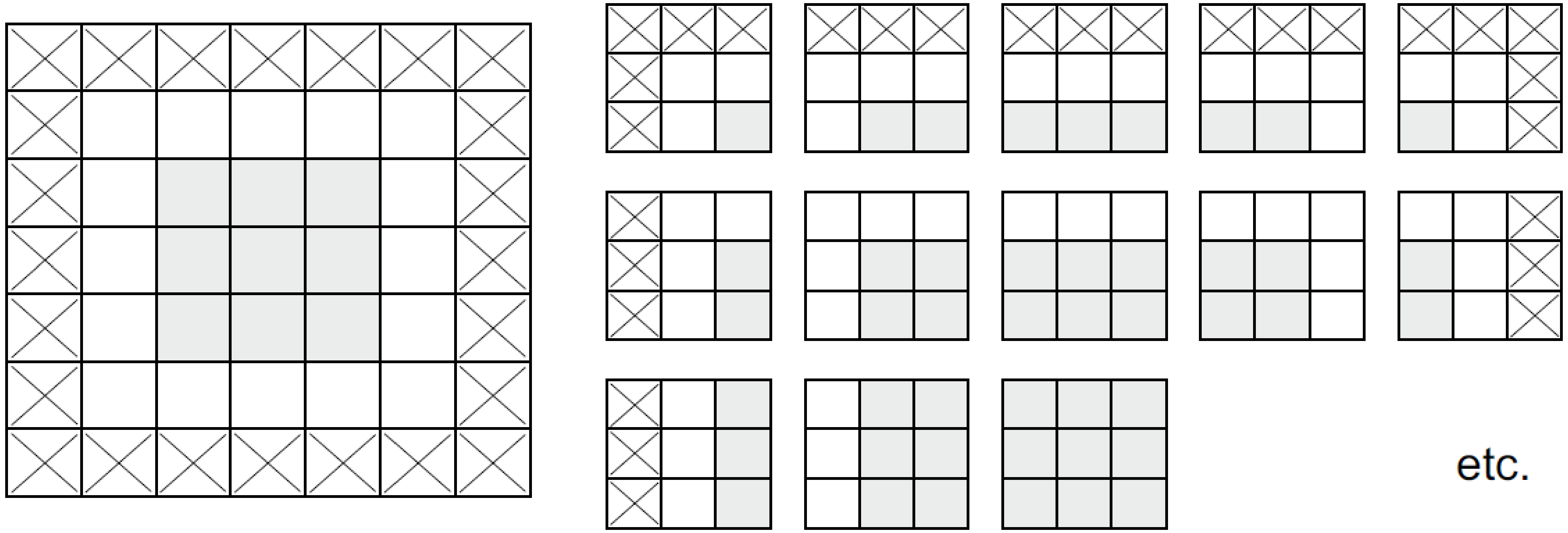


input_shape: (5, 5)
filters = 3
kernel_size = (3, 3)
padding = ‘valid’
output_shape: (3, 3)

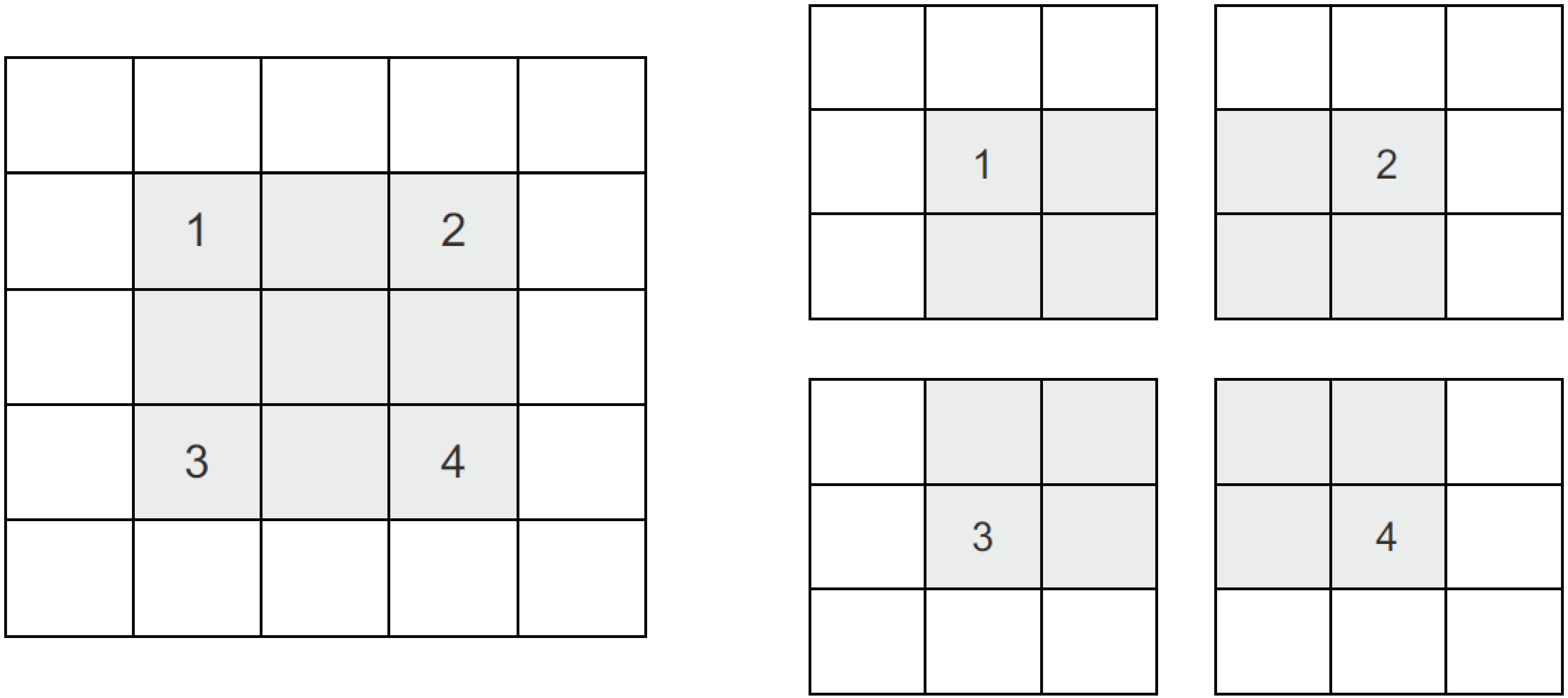
3x3 Patches for a 5x5 Input: padding = ‘valid’



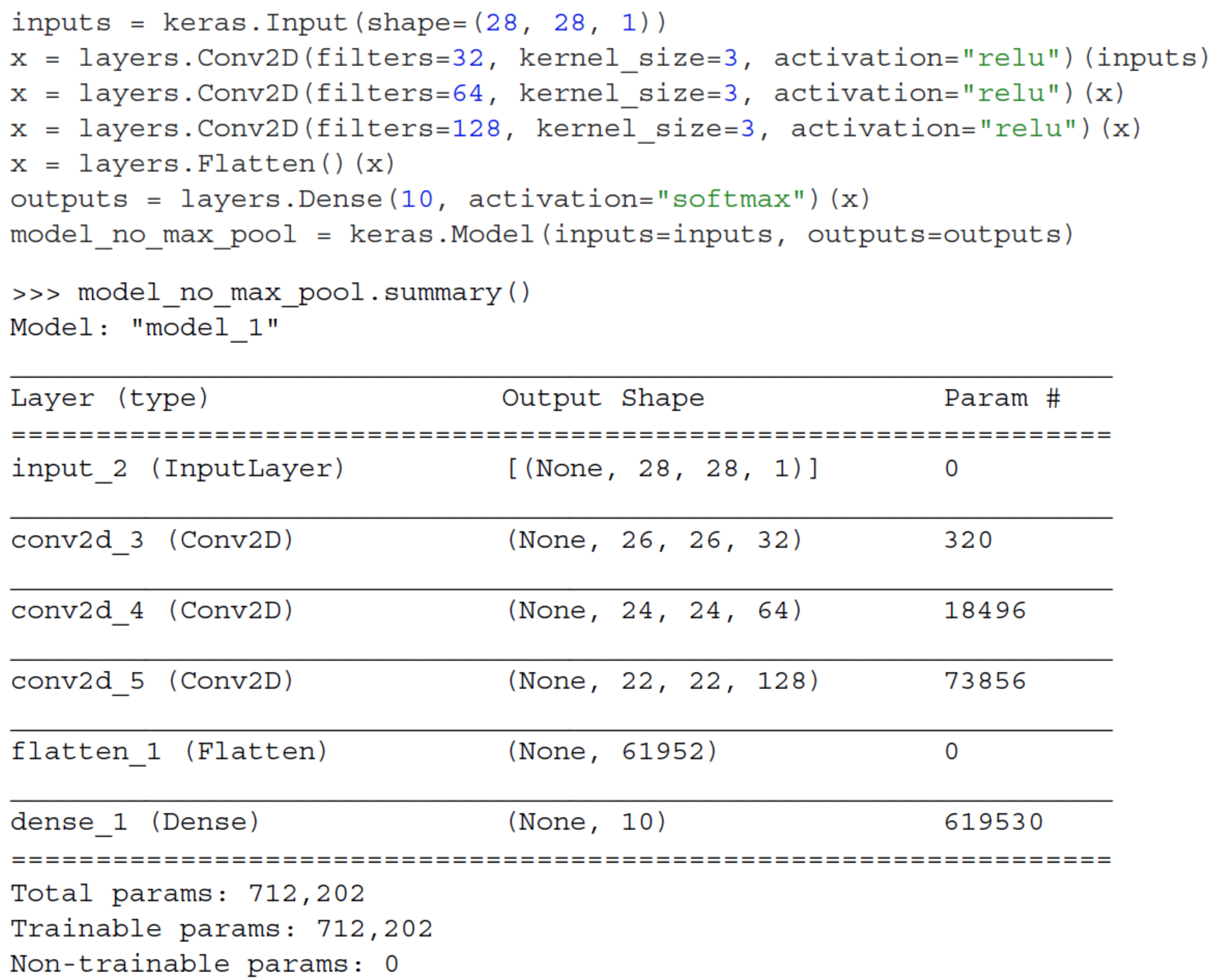
3x3 Patches for a 5x5 Input: padding = ‘same’



3x3 Patches for 5x5 Input: strides = 2 (‘valid’)



Convolution Without Pooling (or Strides > 1)



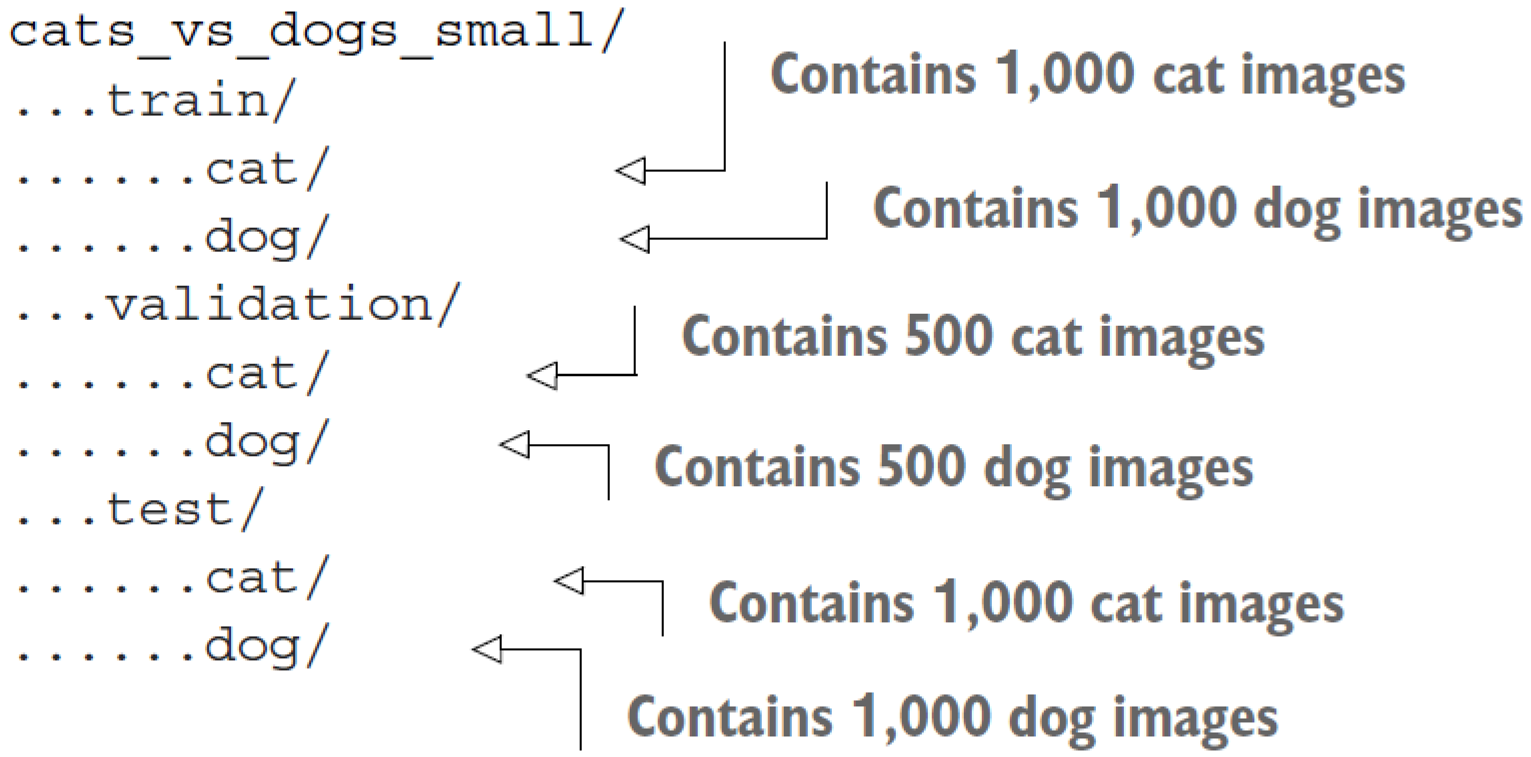
```
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model_no_max_pool = keras.Model(inputs=inputs, outputs=outputs)
```

```
>>> model_no_max_pool.summary()
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 22, 22, 128)       73856
_________________________________________________________________
flatten_1 (Flatten)          (None, 61952)             0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                619530
=================================================================
Total params: 712,202
Trainable params: 712,202
Non-trainable params: 0
```

- Limited scale for features
  - conv2d_3: (3, 3)
  - conv2d_4: (5, 5)
  - conv2d_5: (7, 7)
- Added dense layer is large, because the feature map resolution is large

Dogs vs Cats Dataset

cats_vs_dogs_small Directory Structure



```
cats_vs_dogs_small/
...train/
......cat/
......dog/
...validation/
......cat/
......dog/
...test/
......cat/
......dog/
```

Contains 1,000 cat images
Contains 1,000 dog images
Contains 500 cat images
Contains 500 dog images
Contains 1,000 cat images
Contains 1,000 dog images

Copying Images

```
import os, shutil, pathlib

original_dir = pathlib.Path("train")
new_base_dir = pathlib.Path("cats_vs_dogs_small")

def make_subset(subset_name, start_index, end_index):
    for category in ("cat", "dog"):
        dir = new_base_dir / subset_name / category
        os.makedirs(dir)
        fnames = [f"{category}.{i}.jpg"
                  for i in range(start_index, end_index)]
        for fname in fnames:
            shutil.copyfile(src=original_dir / fname,
                            dst=dir / fname)

make_subset("train", start_index=0, end_index=1000)
make_subset("validation", start_index=1000, end_index=1500)
make_subset("test", start_index=1500, end_index=2500)
```

Path to the directory where the original dataset was uncompressed

Directory where we will store our smaller dataset

Utility function to copy cat (and dog) images from index start_index to index end_index to the subdirectory new_base_dir/{subset_name}/cat (and /dog). The "subset_name" will be either "train", "validation", or "test".

Create the training subset with the first 1,000 images of each category.

Create the validation subset with the next 500 images of each category.

Create the test subset with the next 1,000 images of each category.

Building the Model

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

The model expects RGB images of size 180 × 180.

Rescale inputs to the [0, 1] range by dividing them by 255.

Model Summary

```
>>> model.summary()
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 180, 180, 3)]     0
_________________________________________________________________
rescaling (Rescaling)        (None, 180, 180, 3)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 178, 178, 32)      896
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 89, 89, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 87, 87, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 43, 43, 64)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 41, 41, 128)       73856
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 20, 20, 128)       0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 18, 18, 256)       295168
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 9, 9, 256)         0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 7, 7, 256)         590080
_________________________________________________________________
flatten_2 (Flatten)          (None, 12544)             0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 12545
=================================================================
Total params: 991,041
Trainable params: 991,041
Non-trainable params: 0
```

Configuring the Model

```
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

Preprocessing the Data

- Read the picture files
- Decode the JPEG content to RGB grids of pixels
- Convert these into floating-point tensors
- Resize them to a shared size (we’ll use 180 × 180)
- Pack them into batches (we’ll use batches of 32 images)

image_dataset_from_directory

```
from tensorflow.keras.utils import image_dataset_from_directory

train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(180, 180),
    batch_size=32)
validation_dataset = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(180, 180),
    batch_size=32)
test_dataset = image_dataset_from_directory(
    new_base_dir / "test",
    image_size=(180, 180),
    batch_size=32)
```

Tensorflow Datasets

The from_tensor_slices() class method can be used to create a Dataset from a NumPy array, or a tuple or dict of NumPy arrays.

```
import numpy as np
import tensorflow as tf
random_numbers = np.random.normal(size=(1000, 16))
dataset = tf.data.Dataset.from_tensor_slices(random_numbers)
```

```
>>> for i, element in enumerate(dataset):
>>>     print(element.shape)
>>>     if i >= 2:
>>>         break
(16,)
(16,)
(16,)
```

```
>>> batched_dataset = dataset.batch(32)
>>> for i, element in enumerate(batched_dataset):
>>>     print(element.shape)
>>>     if i >= 2:
>>>         break
(32, 16)
(32, 16)
(32, 16)
```

```
>>> reshaped_dataset = dataset.map(lambda x: tf.reshape(x, (4, 4)))
>>> for i, element in enumerate(reshaped_dataset):
>>>     print(element.shape)
>>>     if i >= 2:
>>>         break
(4, 4)
(4, 4)
(4, 4)
```

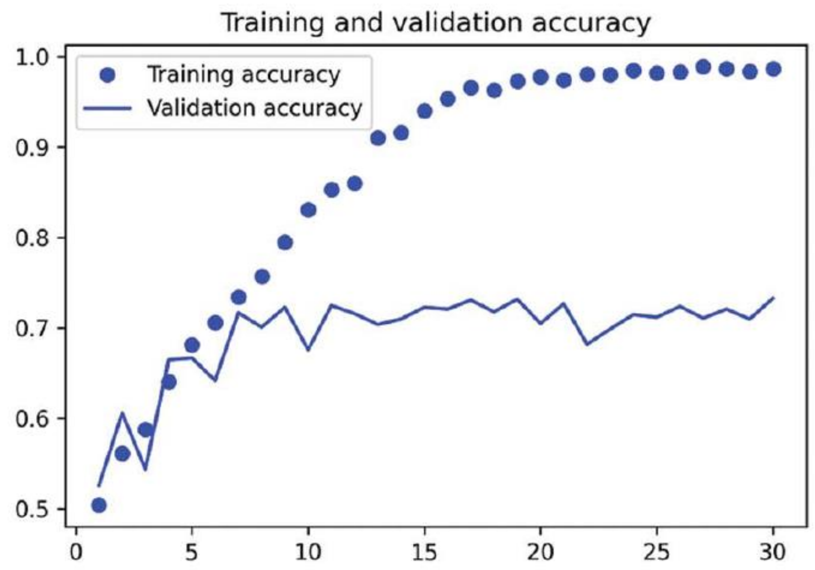
Displaying Shapes of Data and Labels

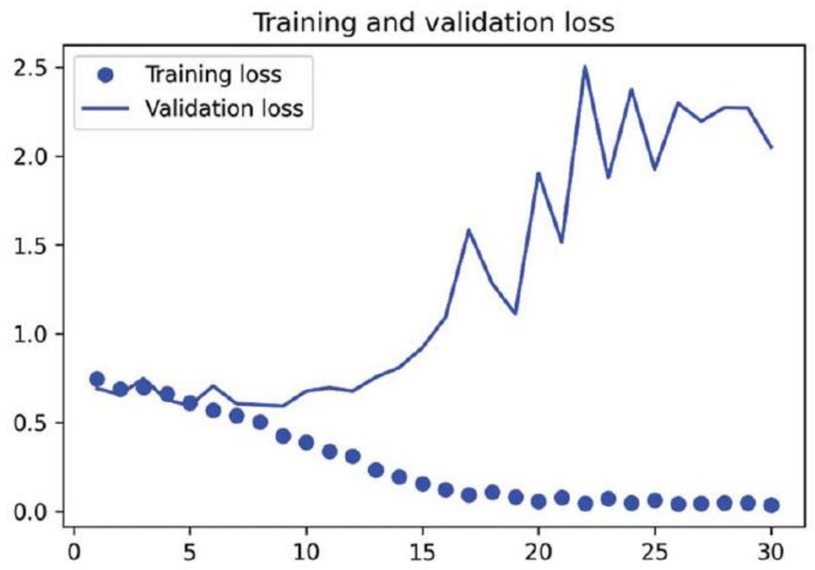
```
>>> for data_batch, labels_batch in train_dataset:
>>>     print("data batch shape:", data_batch.shape)
>>>     print("labels batch shape:", labels_batch.shape)
>>>     break
data batch shape: (32, 180, 180, 3)
labels batch shape: (32,)
```

Fitting the Model

```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]

history = model.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

Tst Acc:
69.5%





Data Augmentation

```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)
```

- `RandomFlip("horizontal")`—Applies horizontal flipping to a random 50% of the images that go through it
- `RandomRotation(0.1)`—Rotates the input images by a random value in the range [–10%, +10%] (these are fractions of a full circle—in degrees, the range would be [–36 degrees, +36 degrees])
- `RandomZoom(0.2)`—Zooms in or out of the image by a random factor in the range [-20%, +20%]

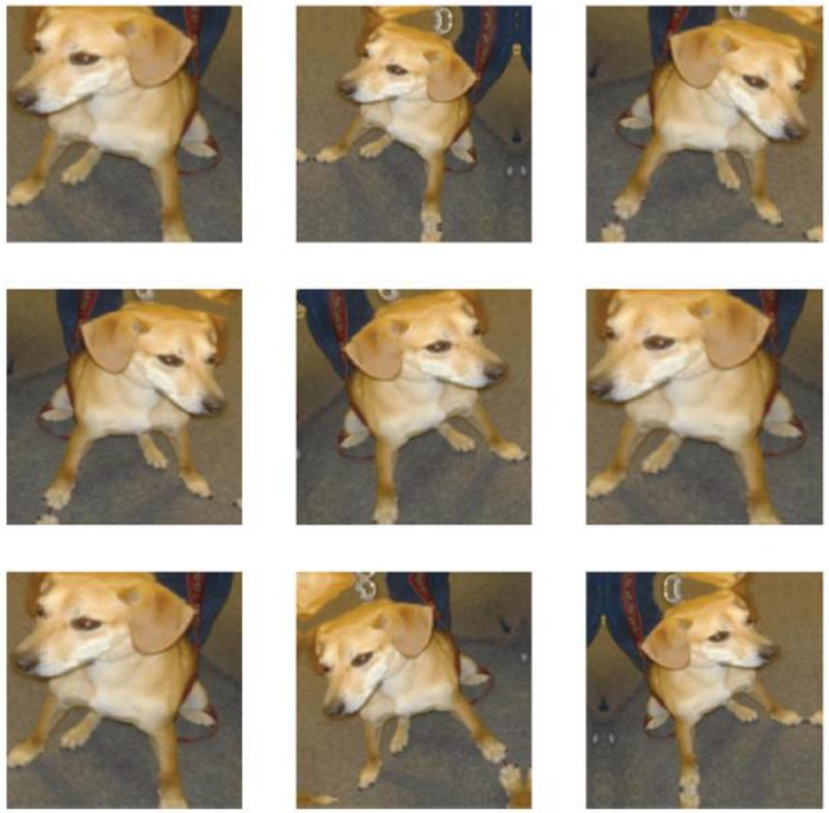
Displaying Randomly Augmented Images

```
plt.figure(figsize=(10, 10))
for images, _ in train_dataset.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")
```

We can use take(N) to only sample *N* batches from the dataset. This is equivalent to inserting a break in the loop after the *N*th batch.

Apply the augmentation stage to the batch of images.

Display the first image in the output batch. For each of the nine iterations, this is a different augmentation of the same image.



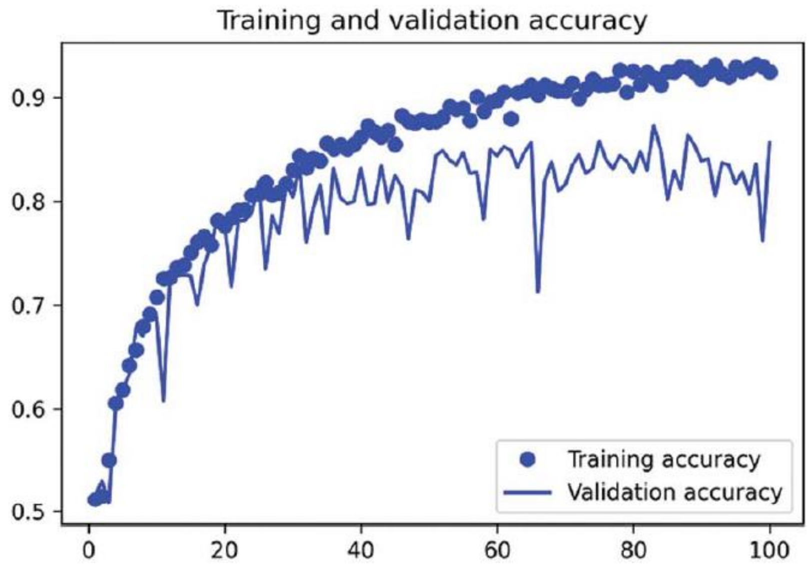
Adding Augmentations Model

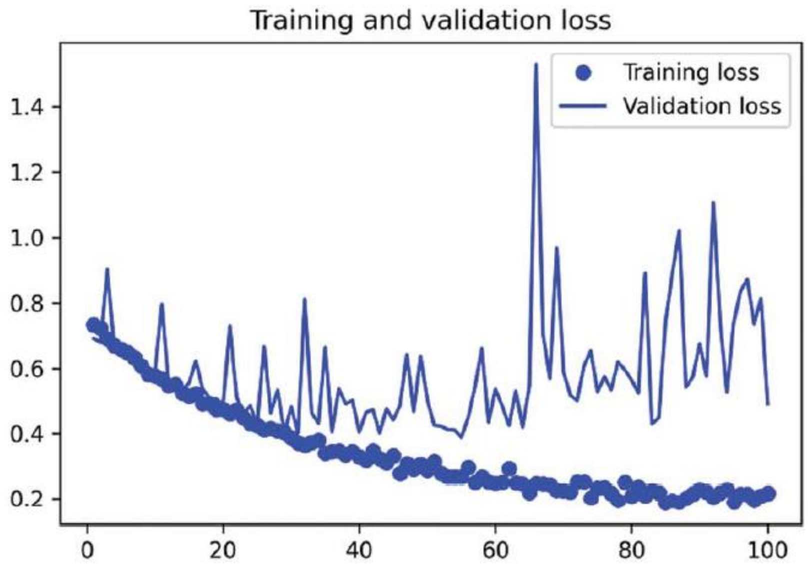
```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)

model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

Fitting the Model

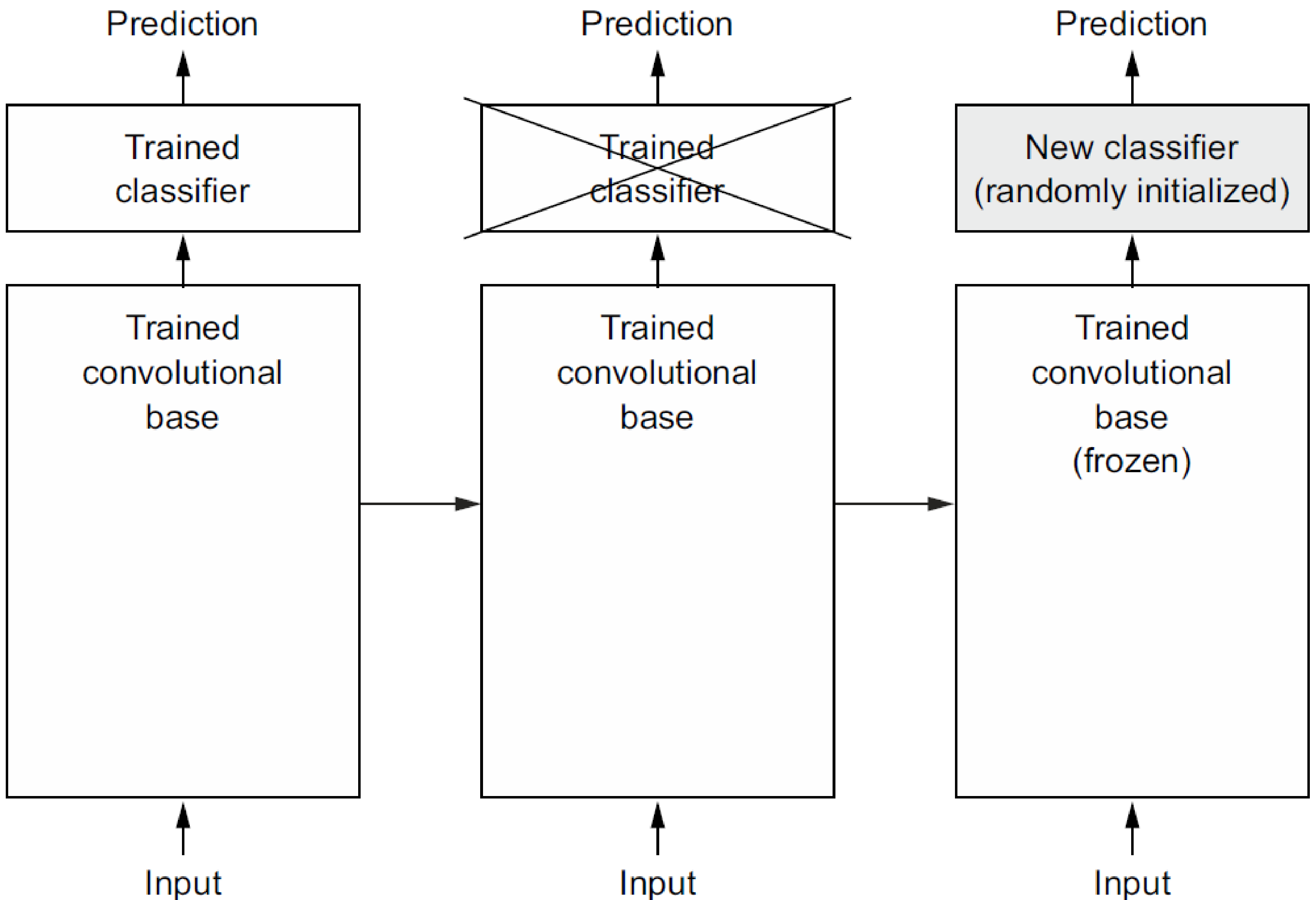
```
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch_with_augmentation.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=100,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

Tst Acc:
83.5%
(was 69.5%)





Using a Frozen Pretrained Model


Prediction
Trained classifier
Trained convolutional base
Input
Prediction
~~Trained classifier~~
Trained convolutional base
Input
Prediction
New classifier (randomly initialized)
Trained convolutional base (frozen)
Input

- Xception
- ResNet
- MobileNet
- EfficientNet
- DenseNet
- etc.

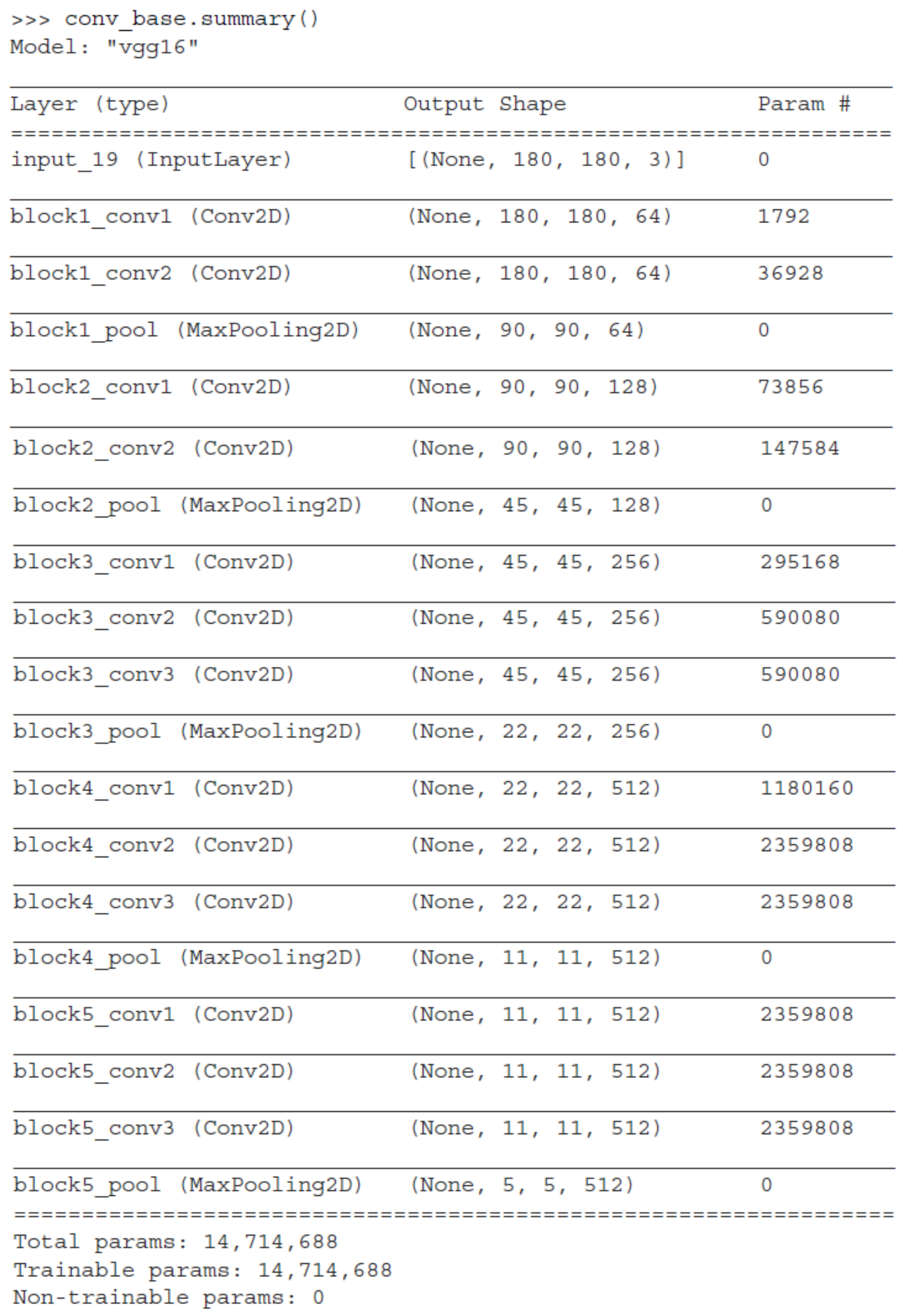
Instantiating VGG16

```
conv_base = keras.applications.vgg16.VGG16(
    weights="imagenet",
    include_top=False,
    input_shape=(180, 180, 3))
```

- weights specifies the weight checkpoint from which to initialize the model.
- include_top refers to including (or not) the densely connected classifier on top of the network. By default, this densely connected classifier corresponds to the 1,000 classes from ImageNet. Because we intend to use our own densely connected classifier (with only two classes: cat and dog), we don't need to include it.
- input_shape is the shape of the image tensors that we'll feed to the network. This argument is purely optional: if we don't pass it, the network will be able to process inputs of any size. Here we pass it so that we can visualize (in the following summary) how the size of the feature maps shrinks with each new convolution and pooling layer.

Pretrained

VGG16 Summary



```
>>> conv_base.summary()
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_19 (InputLayer)        [(None, 180, 180, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 180, 180, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 180, 180, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 90, 90, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 90, 90, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 90, 90, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 45, 45, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 45, 45, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 45, 45, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 45, 45, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 22, 22, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 22, 22, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 22, 22, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 22, 22, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 11, 11, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 11, 11, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 11, 11, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 11, 11, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 5, 5, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

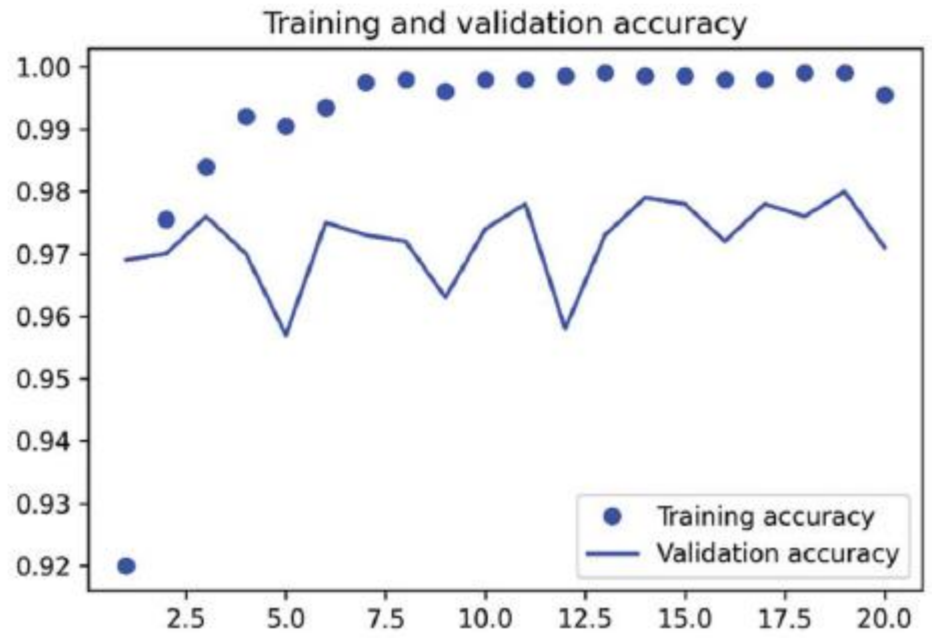
Using a Separate Model

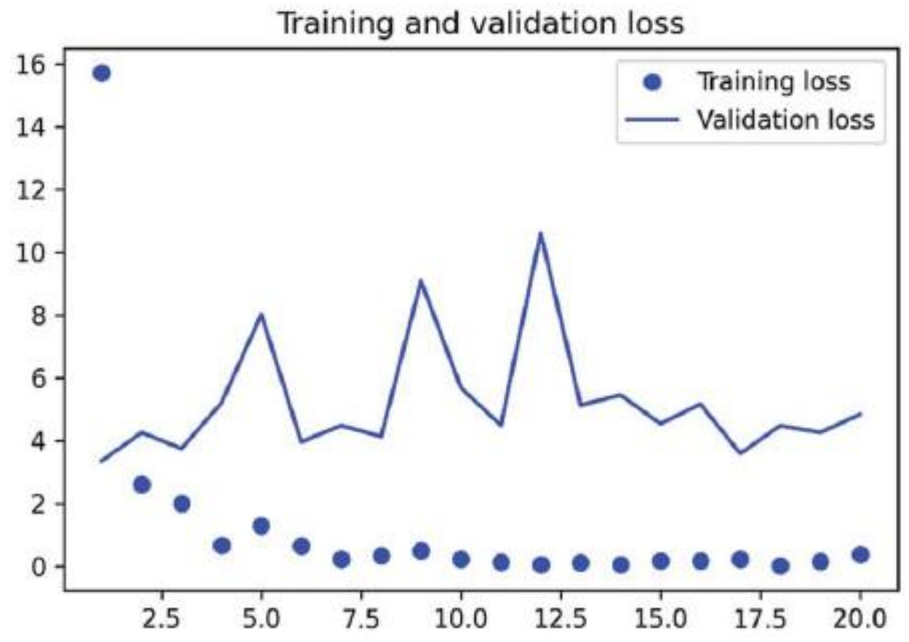
```
import numpy as np

def get_features_and_labels(dataset):
    all_features = []
    all_labels = []
    for images, labels in dataset:
        preprocessed_images = keras.applications.vgg16.preprocess_input(images)
        features = conv_base.predict(preprocessed_images)
        all_features.append(features)
        all_labels.append(labels)
    return np.concatenate(all_features), np.concatenate(all_labels)

train_features, train_labels =  get_features_and_labels(train_dataset)
val_features, val_labels =  get_features_and_labels(validation_dataset)
test_features, test_labels =  get_features_and_labels(test_dataset)

inputs = keras.Input(shape=(5, 5, 512))
x = layers.Flatten()(inputs)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
```

Note the use of the Flatten layer before passing the features to a Dense layer.

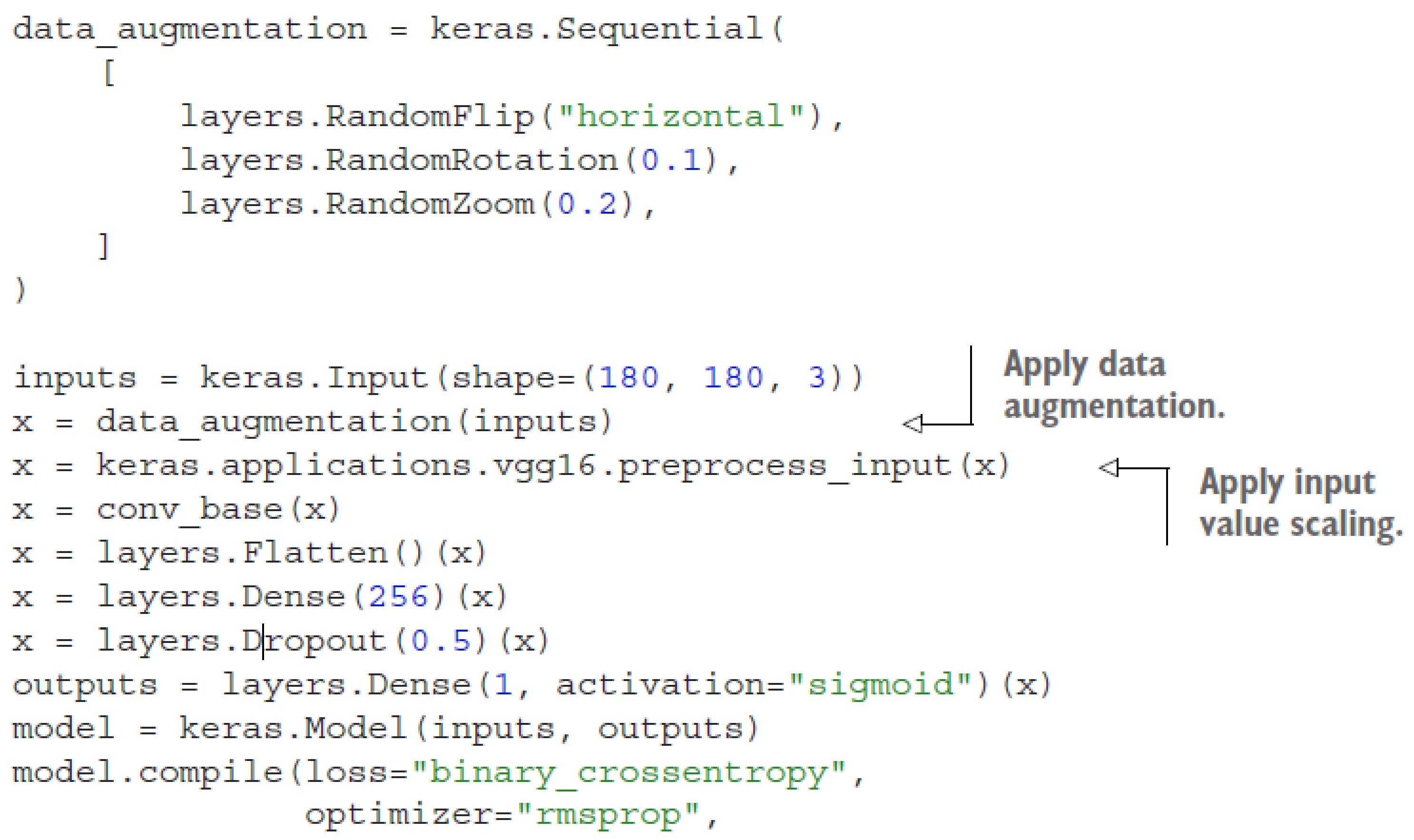
Result for Using a Separate Model

Val Acc: around 97% (was Tst Acc: { 69.5%, 83.5% })





Extending the Model with Data Augmentation



```
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.2),
    ]
)

inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = keras.applications.vgg16.preprocess_input(x)
x = conv_base(x)
x = layers.Flatten()(x)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
```

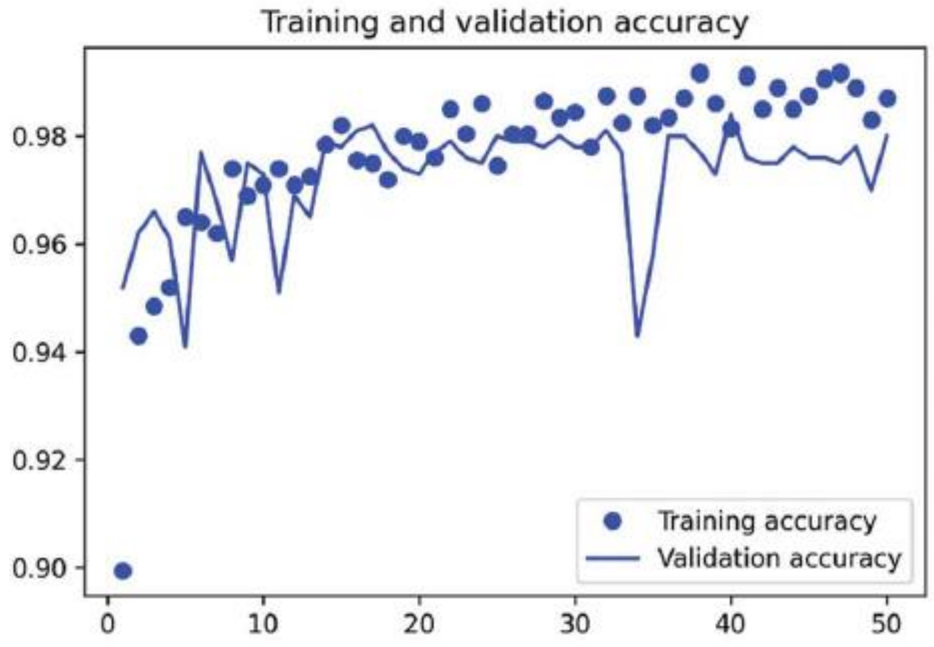
Apply data augmentation.

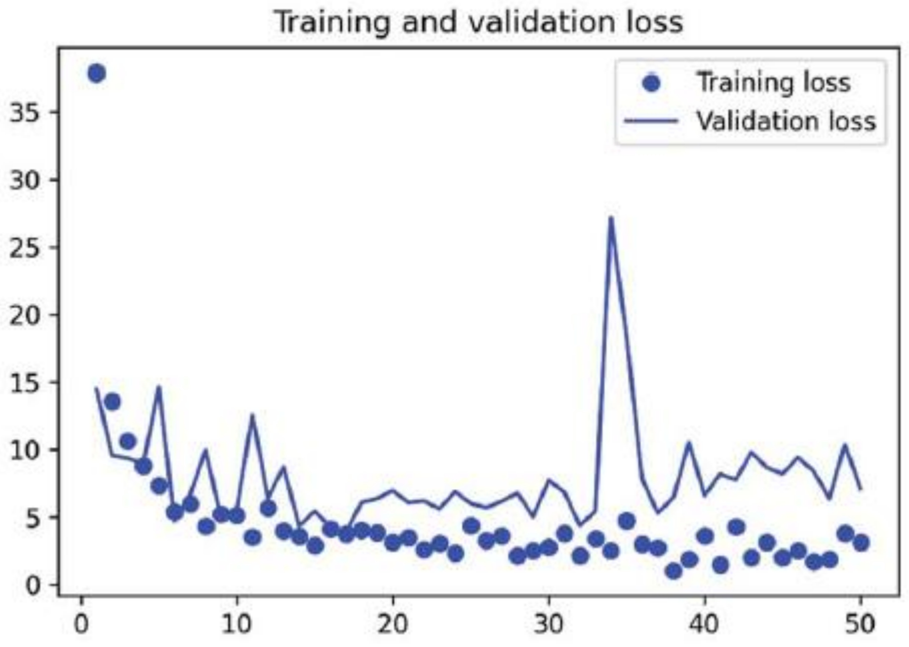
Apply input value scaling.

Pretrained

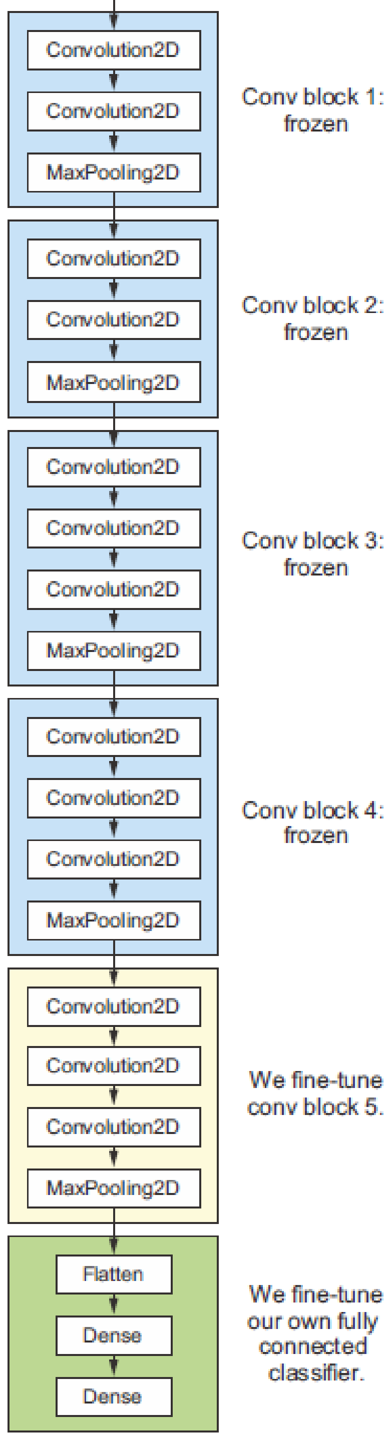
Result for Extending the Model

Tst Acc: 97.5% (was Val Acc: around 97%; Tst Acc: { 69.5%, 83.5% })





Finetuning


Convolution2D
Convolution2D
MaxPooling2D
Conv block 1: frozen
Convolution2D
Convolution2D
MaxPooling2D
Conv block 2: frozen
Convolution2D
Convolution2D
Convolution2D
MaxPooling2D
Conv block 3: frozen
Convolution2D
Convolution2D
Convolution2D
MaxPooling2D
Conv block 4: frozen
Convolution2D
Convolution2D
Convolution2D
MaxPooling2D
We fine-tune conv block 5.
Flatten
Dense
Dense
We fine-tune our own fully connected classifier.

Unfreezing Last Block

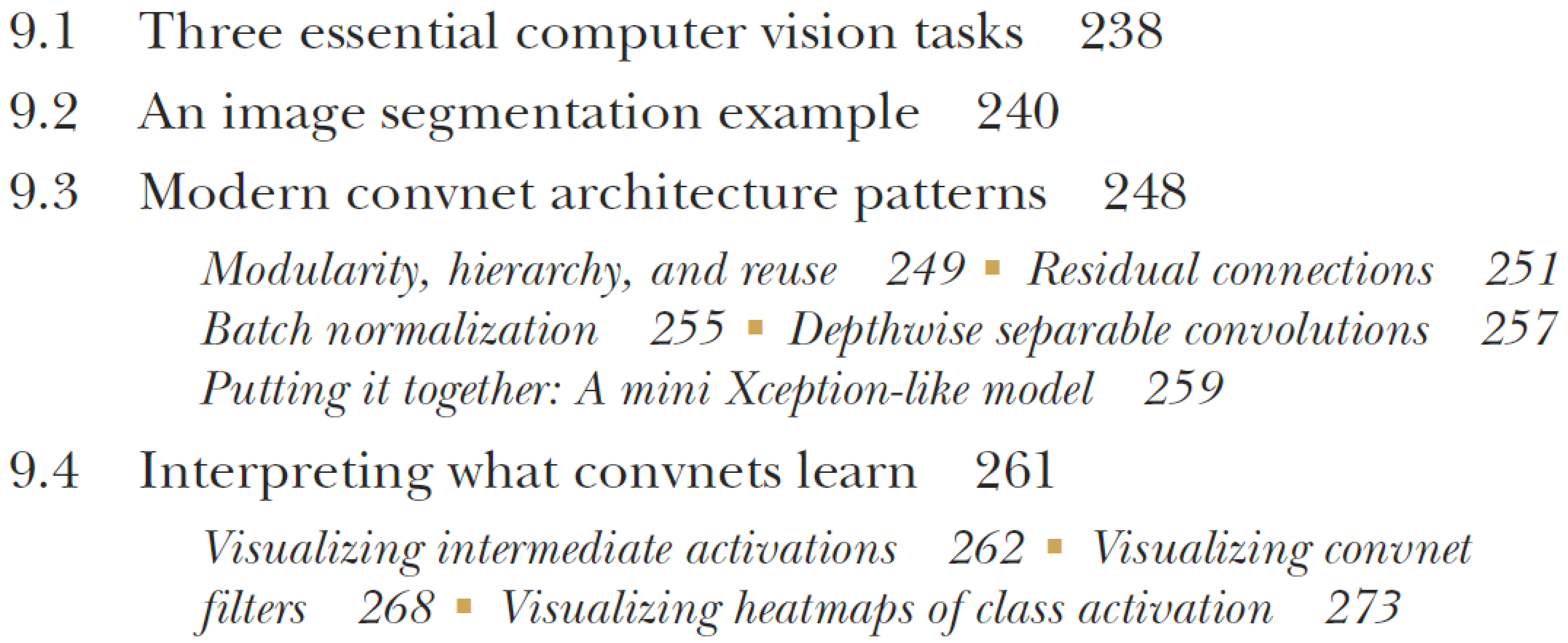
- block5: block5_conv1, block5_conv2, block5_conv3, block5_pool
- Finetuning: Tst Acc = 98.5%
  - Previous
    - Small ConvNet: Tst Acc = 69.5%
    - Data Augmentations: Tst Acc = 83.5%
    - Frozen without Augmentations: Val Acc around 97%
    - Frozen with Augmentations: Tst Acc: 97.5%

```
conv_base.trainable = True
for layer in conv_base.layers[:-4]:
    layer.trainable = False
```

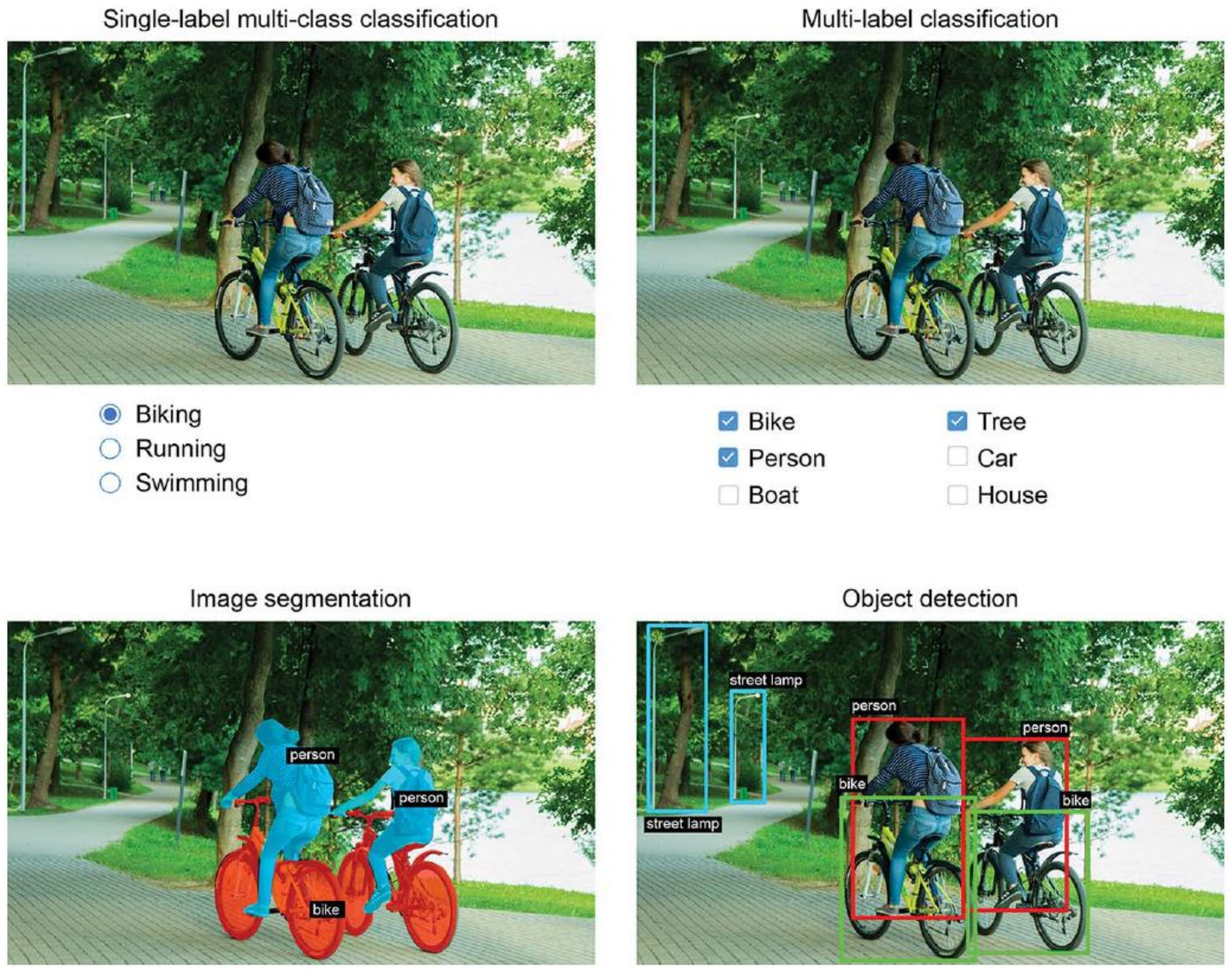
Summary

- Convnets are the best type of machine learning models for computer vision tasks. It's possible to train one from scratch even on a very small dataset, with decent results.
- Convnets work by learning a hierarchy of modular patterns and concepts to represent the visual world.
- On a small dataset, overfitting will be the main issue. Data augmentation is a powerful way to fight overfitting when you're working with image data.
- It's easy to reuse an existing convnet on a new dataset via feature extraction. This is a valuable technique for working with small image datasets.
- As a complement to feature extraction, you can use fine-tuning, which adapts to a new problem some of the representations previously learned by an existing model. This pushes performance a bit further.

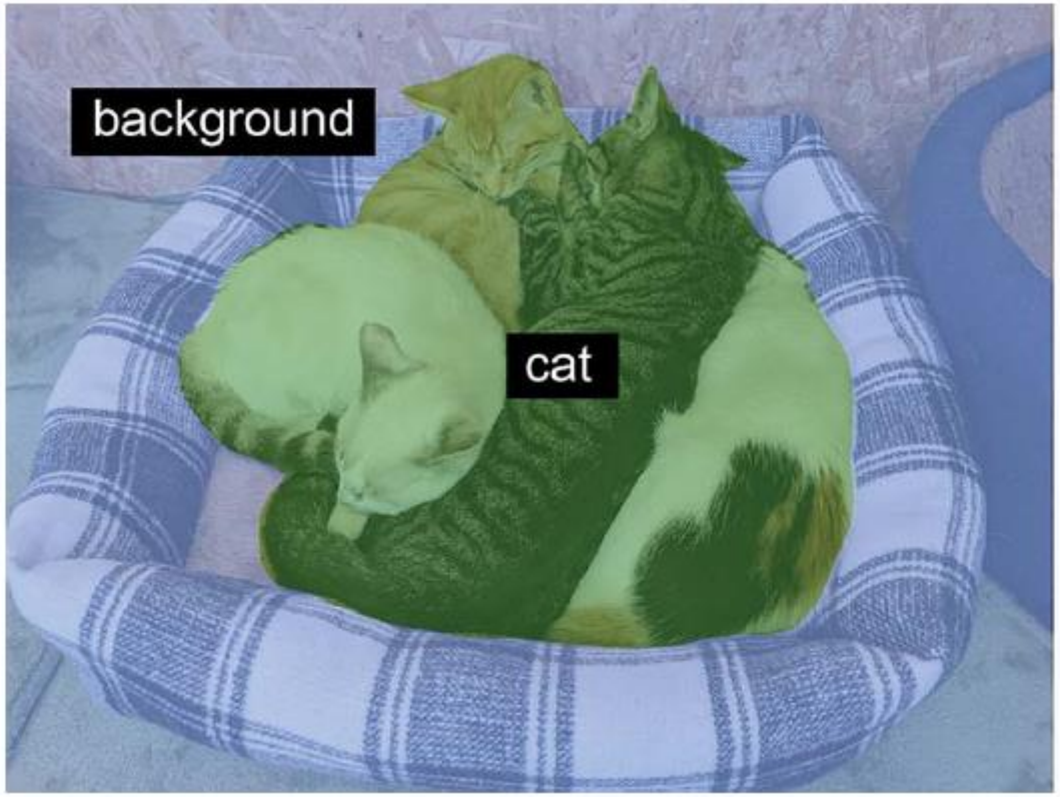
Advanced Deep Learning for Computer Vision

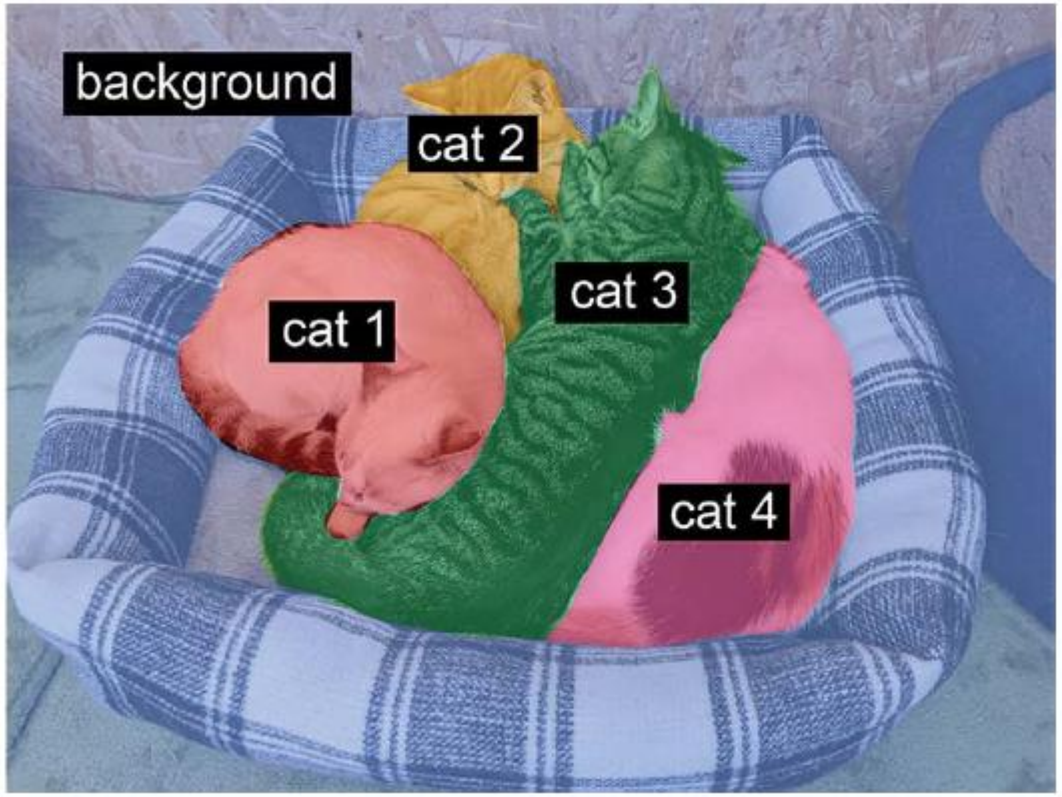


Computer Vision Tasks



Semantic vs Instance Segmentation





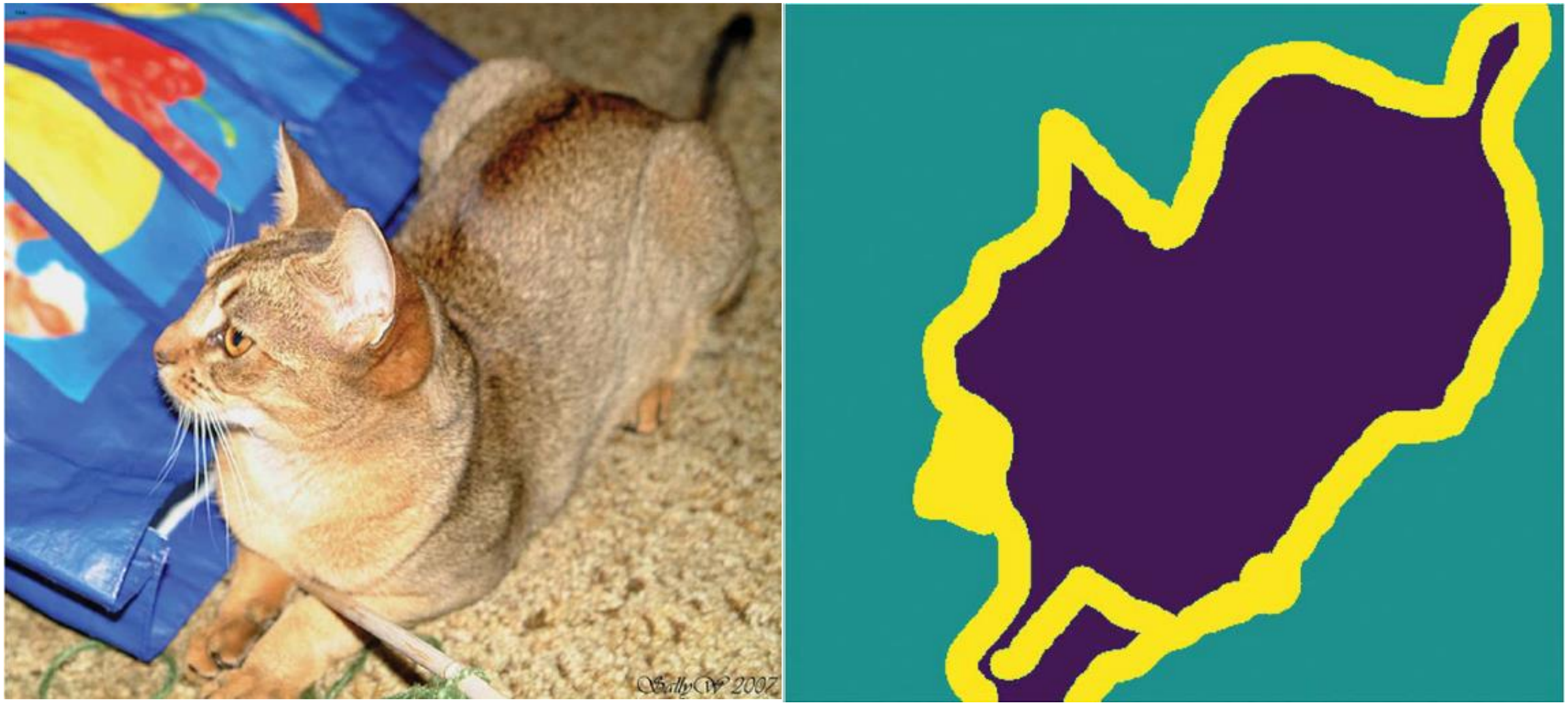
Semantic Segmentation Data

```
!wget http://www.robots.ox.ac.uk/~vgg/data/pets/data/images.tar.gz
!wget http://www.robots.ox.ac.uk/~vgg/data/pets/data/annotations.tar.gz
!tar -xf images.tar.gz
!tar -xf annotations.tar.gz
```

```
import os

input_dir = "images/"
target_dir = "annotations/trimaps/"

input_img_paths = sorted(
    [os.path.join(input_dir, fname)
     for fname in os.listdir(input_dir)
     if fname.endswith(".jpg")])

target_paths = sorted(
    [os.path.join(target_dir, fname)
     for fname in os.listdir(target_dir)
     if fname.endswith(".png") and not fname.startswith(".")])
```

Semantic Segmentation Example

1 (foreground)
2 (background)
3 (contour)



Partitioning the Data

```
import numpy as np
import random

img_size = (200, 200)
num_imgs = len(input_img_paths)

random.Random(1337).shuffle(input_img_paths)
random.Random(1337).shuffle(target_paths)

def path_to_input_image(path):
    return img_to_array(load_img(path, target_size=img_size))

def path_to_target(path):
    img = img_to_array(
        load_img(path, target_size=img_size, color_mode="grayscale"))
    img = img.astype("uint8") - 1
    return img

input_imgs = np.zeros((num_imgs,) + img_size + (3,), dtype="float32")
targets = np.zeros((num_imgs,) + img_size + (1,), dtype="uint8")
for i in range(num_imgs):
    input_imgs[i] = path_to_input_image(input_img_paths[i])
    targets[i] = path_to_target(target_paths[i])

num_val_samples = 1000
train_input_imgs = input_imgs[:-num_val_samples]
train_targets = targets[:-num_val_samples]
val_input_imgs = input_imgs[-num_val_samples:]
val_targets = targets[-num_val_samples:]
```

We resize everything to 200 × 200.

Total number of samples in the data

Shuffle the file paths (they were originally sorted by breed). We use the same seed (1337) in both statements to ensure that the input paths and target paths stay in the same order.

Subtract 1 so that our labels become 0, 1, and 2.

Load all images in the input_imgs float32 array and their masks in the targets uint8 array (same order). The inputs have three channels (RBG values) and the targets have a single channel (which contains integer labels).

Reserve 1,000 samples for validation.

Split the data into a training and a validation set.

Building the Model

```
from tensorflow import keras
from tensorflow.keras import layers

def get_model(img_size, num_classes):
    inputs = keras.Input(shape=img_size + (3,))
    x = layers.Rescaling(1./255)(inputs)

    x = layers.Conv2D(64, 3, strides=2, activation="relu", padding="same")(x)
    x = layers.Conv2D(64, 3, activation="relu", padding="same")(x)
    x = layers.Conv2D(128, 3, strides=2, activation="relu", padding="same")(x)
    x = layers.Conv2D(128, 3, activation="relu", padding="same")(x)
    x = layers.Conv2D(256, 3, strides=2, padding="same", activation="relu")(x)
    x = layers.Conv2D(256, 3, activation="relu", padding="same")(x)

    x = layers.Conv2DTranspose(256, 3, activation="relu", padding="same")(x)
    x = layers.Conv2DTranspose(
        256, 3, activation="relu", padding="same", strides=2)(x)
    x = layers.Conv2DTranspose(128, 3, activation="relu", padding="same")(x)
    x = layers.Conv2DTranspose(
        128, 3, activation="relu", padding="same", strides=2)(x)
    x = layers.Conv2DTranspose(64, 3, activation="relu", padding="same")(x)
    x = layers.Conv2DTranspose(
        64, 3, activation="relu", padding="same", strides=2)(x)

    outputs = layers.Conv2D(num_classes, 3, activation="softmax",
     padding="same")(x)

    model = keras.Model(inputs, outputs)
    return model

model = get_model(img_size=img_size, num_classes=3)
model.summary()
```

Don't forget to rescale input images to the [0-1] range.

Note how we use padding="same" everywhere to avoid the influence of border padding on feature map size.

We end the model with a per-pixel three-way softmax to classify each output pixel into one of our three categories.

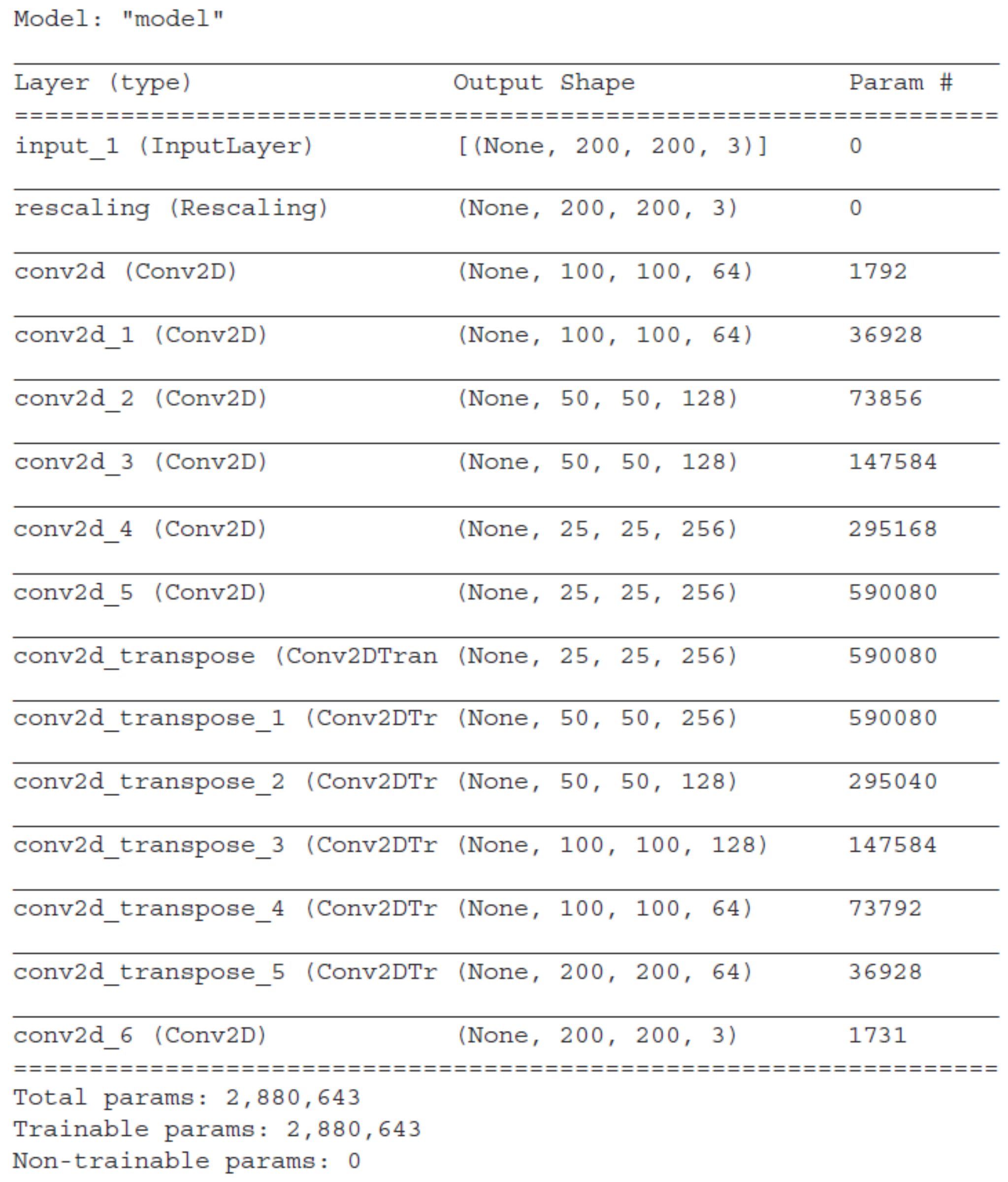
Transposed Convolution

```
import numpy as np
filter = np.array([ [ .1, .2, .3 ], [ .4, .5, .6 ], [ .7, .8, .9 ] ], dtype = np.float32)    # kernel_size = (3, 3)
empty_row = [ .0, .0, .0, .0, .0 ]    # input_shape: (5, 5)
W = np.array([    # notice how each output cell gets its own row below
   [ .1, .2, .3, .0, .0, .4, .5, .6, .0, .0, .7, .8, .9, .0, .0 ] + empty_row + empty_row,
   [ .0, .1, .2, .3, .0, .0, .4, .5, .6, .0, .0, .7, .8, .9, .0 ] + empty_row + empty_row,
   [ .0, .0, .1, .2, .3, .0, .0, .4, .5, .6, .0, .0, .7, .8, .9 ] + empty_row + empty_row,
   empty_row + [ .1, .2, .3, .0, .0, .4, .5, .6, .0, .0, .7, .8, .9, .0, .0 ] + empty_row,
   empty_row + [ .0, .1, .2, .3, .0, .0, .4, .5, .6, .0, .0, .7, .8, .9, .0 ] + empty_row,
   empty_row + [ .0, .0, .1, .2, .3, .0, .0, .4, .5, .6, .0, .0, .7, .8, .9 ] + empty_row,
   empty_row + empty_row + [ .1, .2, .3, .0, .0, .4, .5, .6, .0, .0, .7, .8, .9, .0, .0 ],
   empty_row + empty_row + [ .0, .1, .2, .3, .0, .0, .4, .5, .6, .0, .0, .7, .8, .9, .0 ],
   empty_row + empty_row + [ .0, .0, .1, .2, .3, .0, .0, .4, .5, .6, .0, .0, .7, .8, .9 ]], dtype = np.float32).transpose()
```

To go from (flattened/reshaped) input shape to output shape, we have: 1x25 * 25x9 = 1x9   # reshape to get 3x3 output

To go from (flattened/reshaped) output shape to input shape, we have to transpose the convolution matrix: 1x9 * 9x25 = 1x25

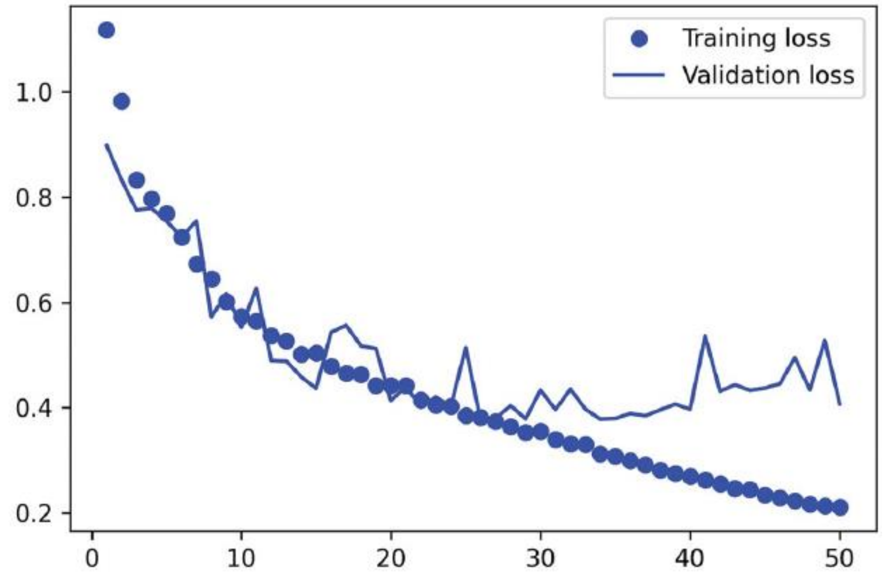
Model Summary



```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 200, 200, 3)]     0
_________________________________________________________________
rescaling (Rescaling)        (None, 200, 200, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 100, 100, 64)      1792
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 100, 100, 64)      36928
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 50, 50, 128)       73856
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 50, 50, 128)       147584
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 25, 25, 256)       295168
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 25, 25, 256)       590080
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 25, 25, 256)       590080
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 50, 50, 256)       590080
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 50, 50, 128)       295040
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 100, 100, 128)     147584
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 100, 100, 64)      73792
_________________________________________________________________
conv2d_transpose_5 (Conv2DTr (None, 200, 200, 64)      36928
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 200, 200, 3)       1731
=================================================================
Total params: 2,880,643
Trainable params: 2,880,643
Non-trainable params: 0
```

Fitting the Model

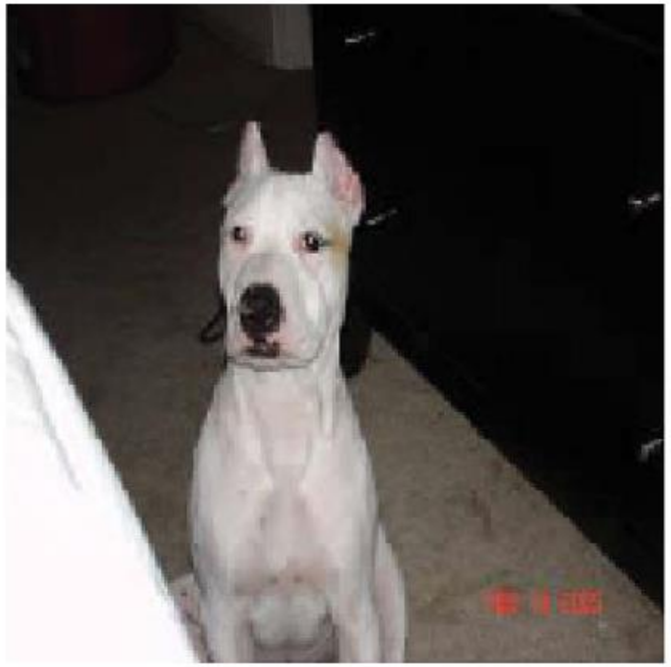
```
model.compile(optimizer="rmsprop", loss="sparse_categorical_crossentropy")

callbacks = [
    keras.callbacks.ModelCheckpoint("oxford_segmentation.keras",
                                    save_best_only=True)
]

history = model.fit(train_input_imgs, train_targets,
                    epochs=50,
                    callbacks=callbacks,
                    batch_size=64,
                    validation_data=(val_input_imgs, val_targets))
```

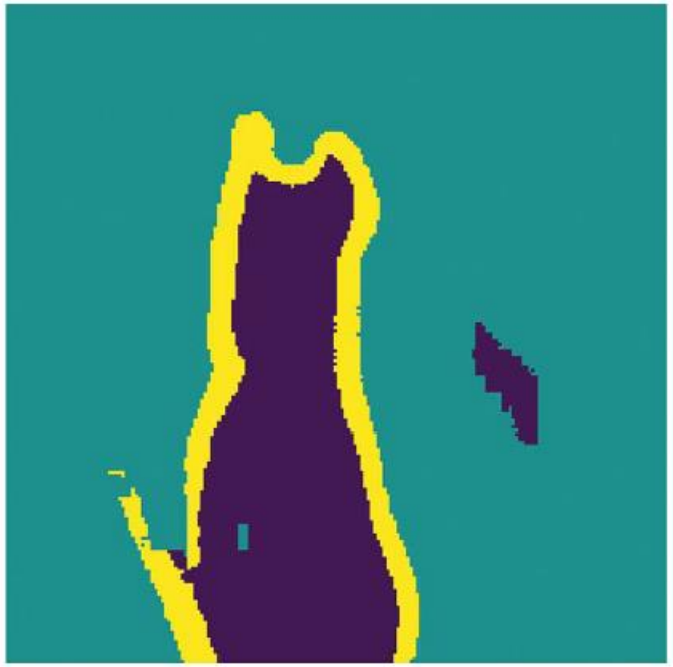


Example Prediction

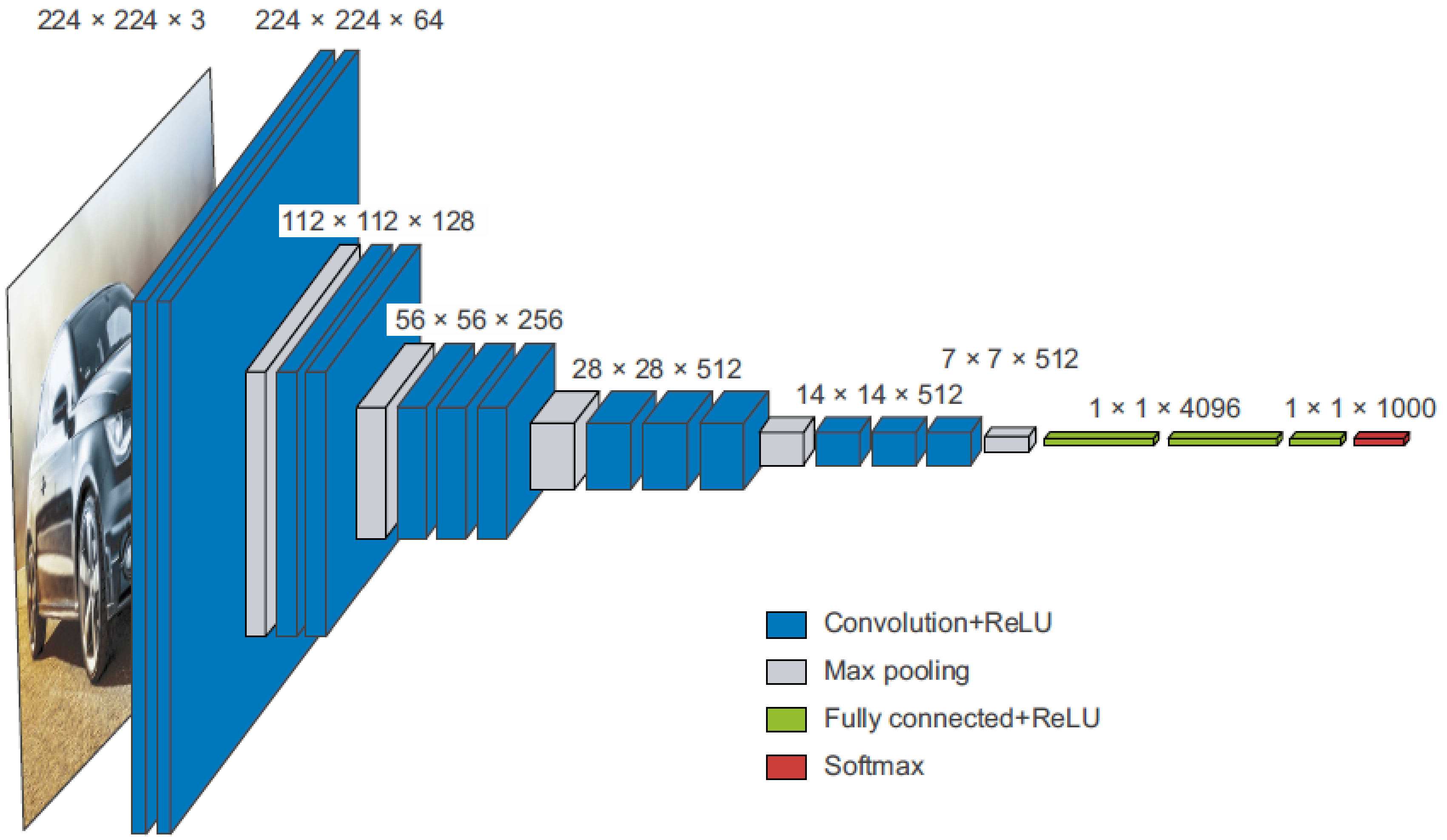
```
mask = model.predict(np.expand_dims(test_image, 0))[0]

def display_mask(pred):
    mask = np.argmax(pred, axis=-1)
    mask *= 127
    plt.axis("off")
    plt.imshow(mask)

display_mask(mask)
```

Utility to display a model's prediction





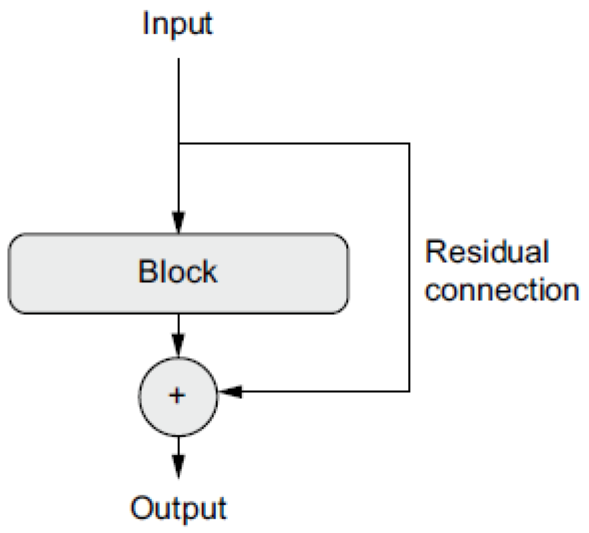
VGG16

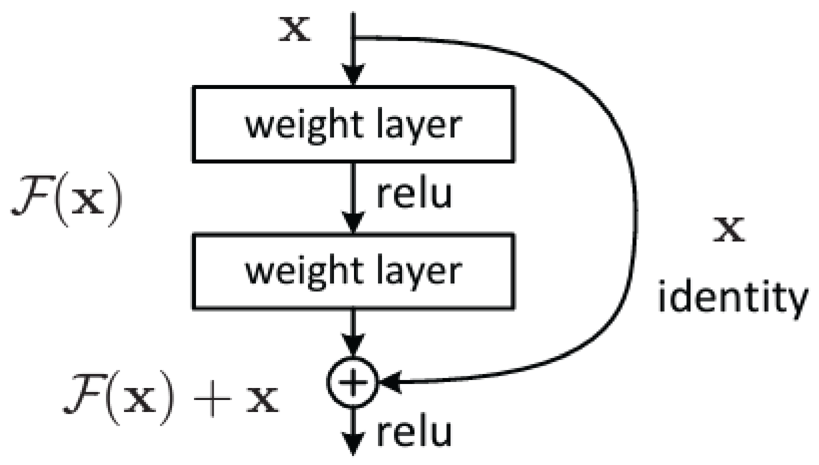

224 × 224 × 3
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
🟦 Convolution+ReLU
⬜ Max pooling
🟩 Fully connected+ReLU
🟥 Softmax

Residual Network (ResNet) Block

I prefer the interpretation from the original paper; i.e. residual = block(x)

https://arxiv.org/abs/1512.03385





Some input tensor

Save a pointer to the original input. This is called the residual.

```
x = ...
residual = x
x = block(x)
x = add([x, residual])
```

This computation block can potentially be destructive or noisy, and that's fine.

Add the original input to the layer's output: the final output will thus always preserve full information about the original input.

A building block is shown in Fig. 2. Formally, in this paper we consider a building block defined as:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}. \qquad (1)$$

Here $\mathbf{x}$ and $\mathbf{y}$ are the input and output vectors of the layers considered. The function $\mathcal{F}(\mathbf{x}, \{W_i\})$ represents the residual mapping to be learned. For the example in Fig. 2 that has two layers, $\mathcal{F} = W_2\sigma(W_1\mathbf{x})$ in which σ denotes

Simple ResNet

```
inputs = keras.Input(shape=(32, 32, 3))
x = layers.Rescaling(1./255)(inputs)

def residual_block(x, filters, pooling=False):
    residual = x
    x = layers.Conv2D(filters, 3, activation="relu", padding="same")(x)
    x = layers.Conv2D(filters, 3, activation="relu", padding="same")(x)
    if pooling:
        x = layers.MaxPooling2D(2, padding="same")(x)
        residual = layers.Conv2D(filters, 1, strides=2)(residual)
    elif filters != residual.shape[-1]:
        residual = layers.Conv2D(filters, 1)(residual)
    x = layers.add([x, residual])
    return x

x = residual_block(x, filters=32, pooling=True)
x = residual_block(x, filters=64, pooling=True)
x = residual_block(x, filters=128, pooling=False)

x = layers.GlobalAveragePooling2D()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
model.summary()
```

Utility function to apply a convolutional block with a residual connection, with an option to add max pooling

If we use max pooling, we add a strided convolution to project the residual to the expected shape.

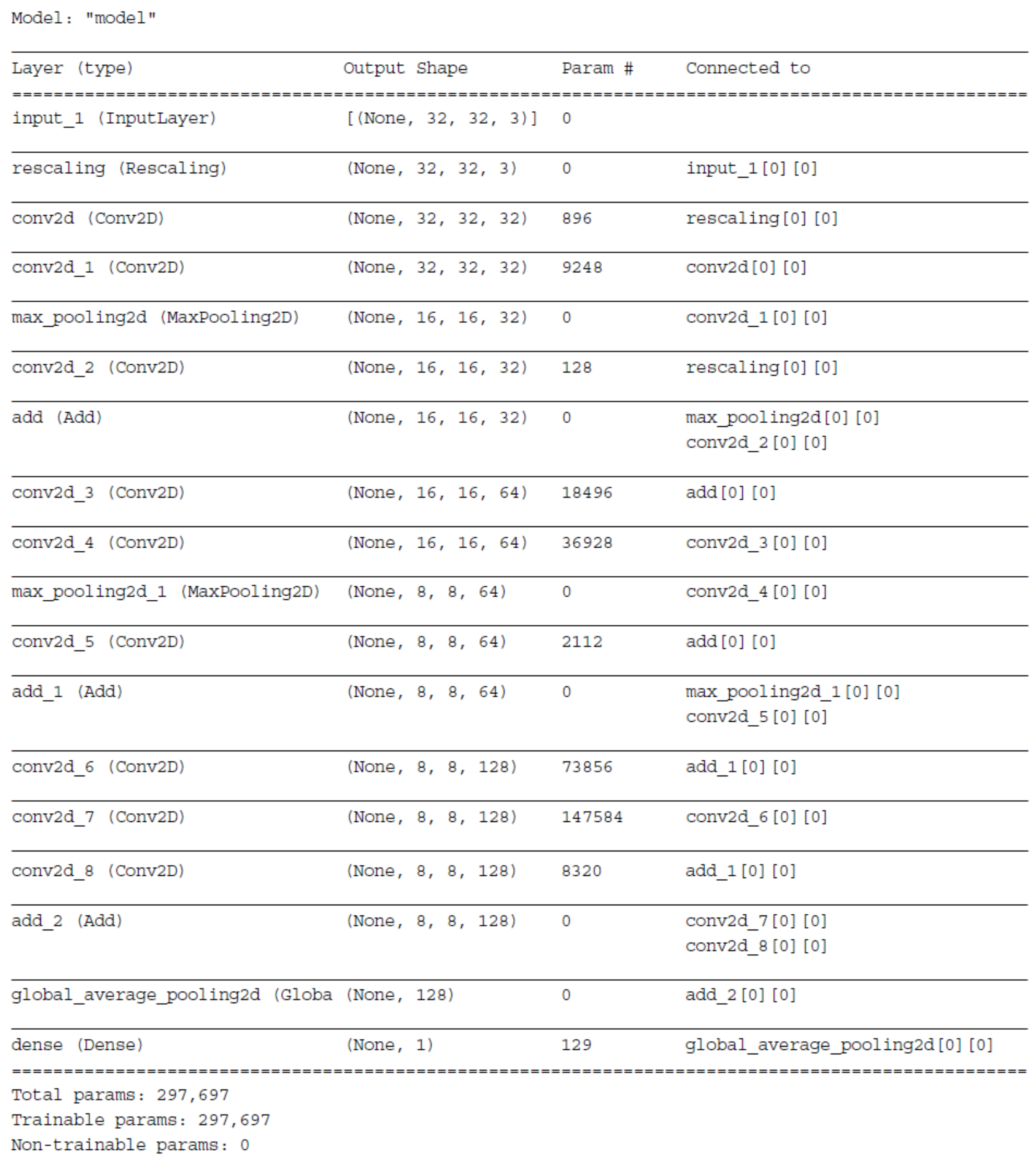
If we don't use max pooling, we only project the residual if the number of channels has changed.

First block

Second block; note the increasing filter count in each block.

The last block doesn't need a max pooling layer, since we will apply global average pooling right after it.

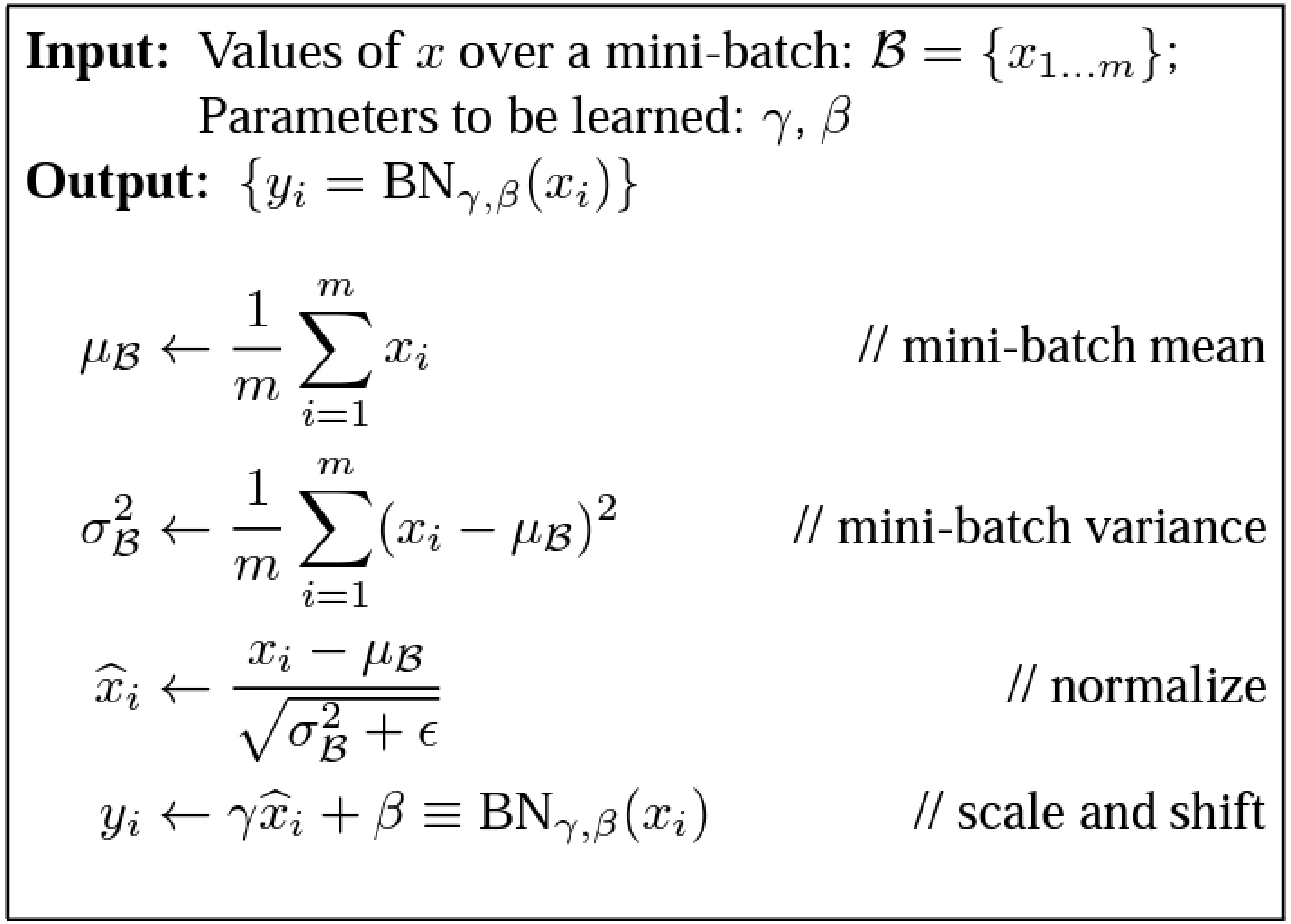
ResNet Summary



```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)           [(None, 32, 32, 3)]  0
__________________________________________________________________________________________________
rescaling (Rescaling)          (None, 32, 32, 3)    0           input_1[0][0]
__________________________________________________________________________________________________
conv2d (Conv2D)                (None, 32, 32, 32)   896         rescaling[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)              (None, 32, 32, 32)   9248        conv2d[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)   (None, 16, 16, 32)   0           conv2d_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)              (None, 16, 16, 32)   128         rescaling[0][0]
__________________________________________________________________________________________________
add (Add)                      (None, 16, 16, 32)   0           max_pooling2d[0][0]
                                                                 conv2d_2[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)              (None, 16, 16, 64)   18496       add[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)              (None, 16, 16, 64)   36928       conv2d_3[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D) (None, 8, 8, 64)     0           conv2d_4[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)              (None, 8, 8, 64)     2112        add[0][0]
__________________________________________________________________________________________________
add_1 (Add)                    (None, 8, 8, 64)     0           max_pooling2d_1[0][0]
                                                                 conv2d_5[0][0]
__________________________________________________________________________________________________
conv2d_6 (Conv2D)              (None, 8, 8, 128)    73856       add_1[0][0]
__________________________________________________________________________________________________
conv2d_7 (Conv2D)              (None, 8, 8, 128)    147584      conv2d_6[0][0]
__________________________________________________________________________________________________
conv2d_8 (Conv2D)              (None, 8, 8, 128)    8320        add_1[0][0]
__________________________________________________________________________________________________
add_2 (Add)                    (None, 8, 8, 128)    0           conv2d_7[0][0]
                                                                 conv2d_8[0][0]
__________________________________________________________________________________________________
global_average_pooling2d (Globa (None, 128)         0           add_2[0][0]
__________________________________________________________________________________________________
dense (Dense)                  (None, 1)            129         global_average_pooling2d[0][0]
==================================================================================================
Total params: 297,697
Trainable params: 297,697
Non-trainable params: 0
```

https://arxiv.org/abs/1502.03167

Batch Normalization



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

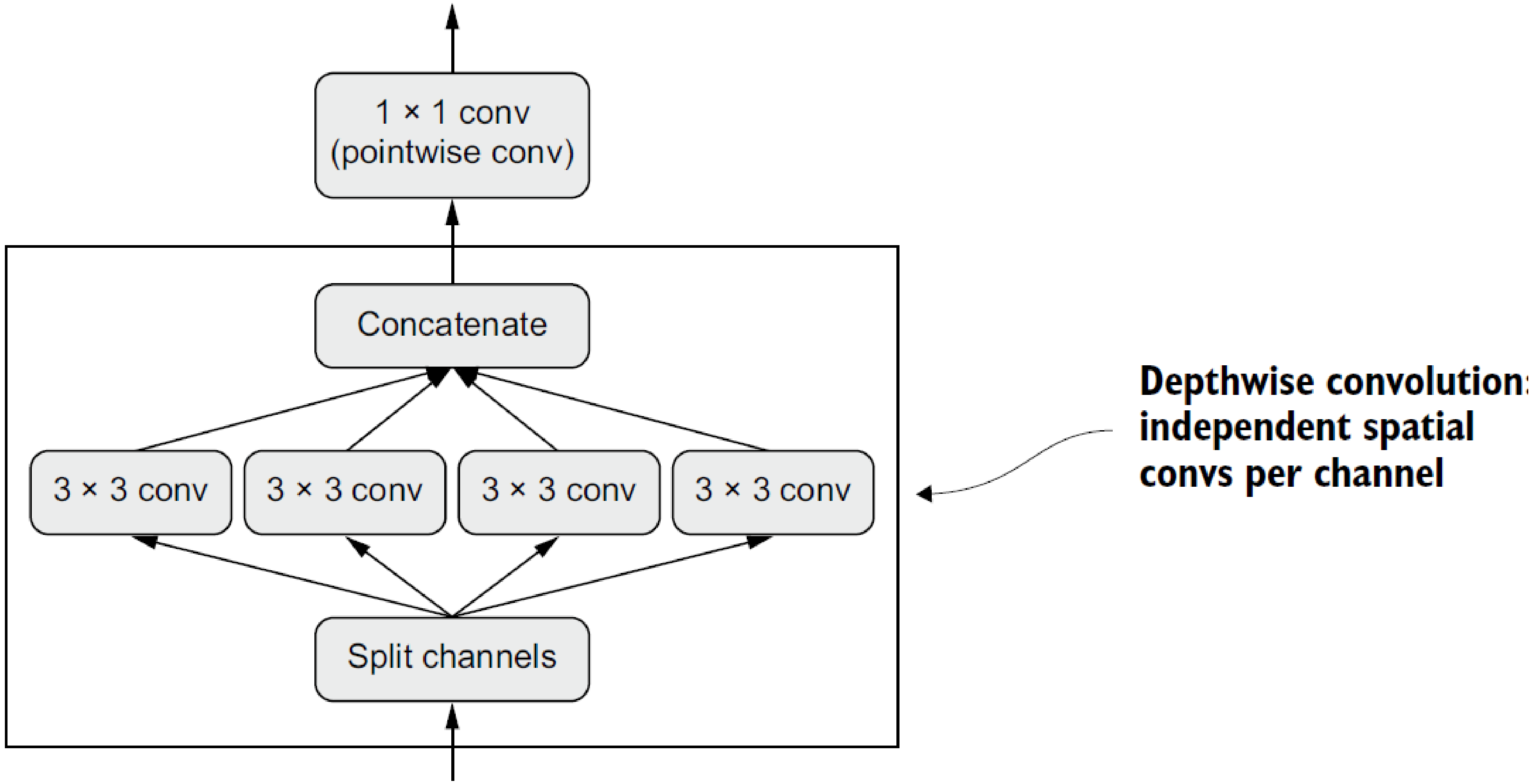
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Depthwise Separable Convolution

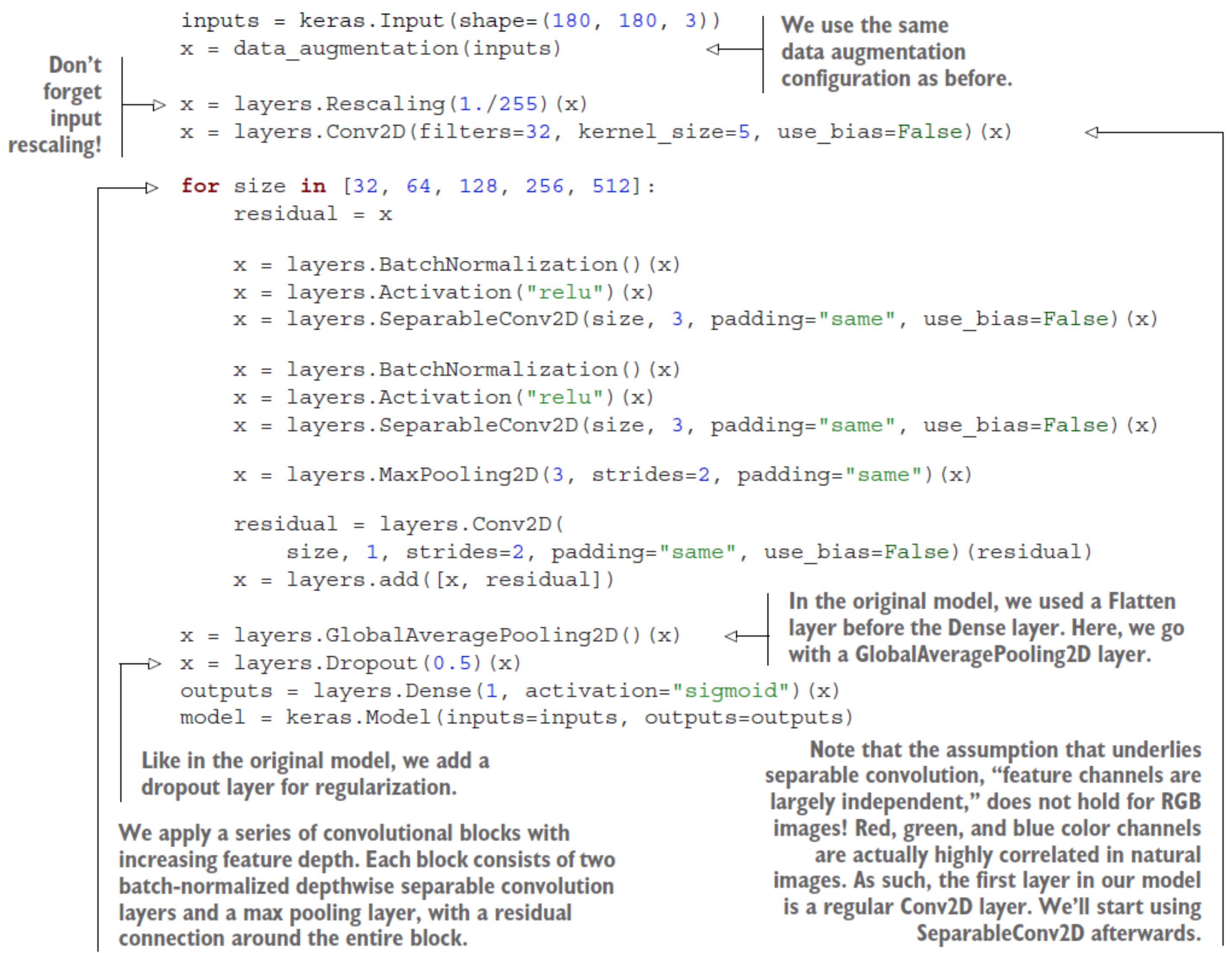
One filter per input channel, followed by 1x1 “pointwise” convolution:



ConvNet Architecture Principles

- Your model should be organized into repeated *blocks* of layers, usually made of multiple convolution layers and a max pooling layer.
- The number of filters in your layers should increase as the size of the spatial feature maps decreases.
- Deep and narrow is better than broad and shallow.
- Introducing residual connections around blocks of layers helps you train deeper networks.
- It can be beneficial to introduce batch normalization layers after your convolution layers.
- It can be beneficial to replace `Conv2D` layers with `SeparableConv2D` layers, which are more parameter-efficient.

A Mini Xception-like Model



```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)

x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=5, use_bias=False)(x)

for size in [32, 64, 128, 256, 512]:
    residual = x

    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)
    x = layers.SeparableConv2D(size, 3, padding="same", use_bias=False)(x)

    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)
    x = layers.SeparableConv2D(size, 3, padding="same", use_bias=False)(x)

    x = layers.MaxPooling2D(3, strides=2, padding="same")(x)

    residual = layers.Conv2D(
        size, 1, strides=2, padding="same", use_bias=False)(residual)
    x = layers.add([x, residual])

x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

We use the same data augmentation configuration as before.

Don't forget input rescaling!

In the original model, we used a Flatten layer before the Dense layer. Here, we go with a GlobalAveragePooling2D layer.

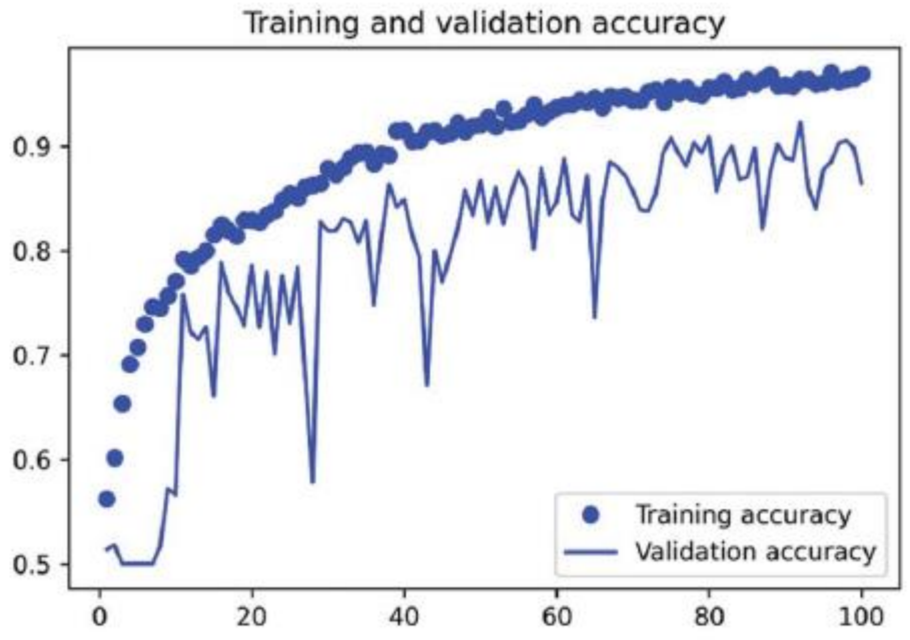
Like in the original model, we add a dropout layer for regularization.

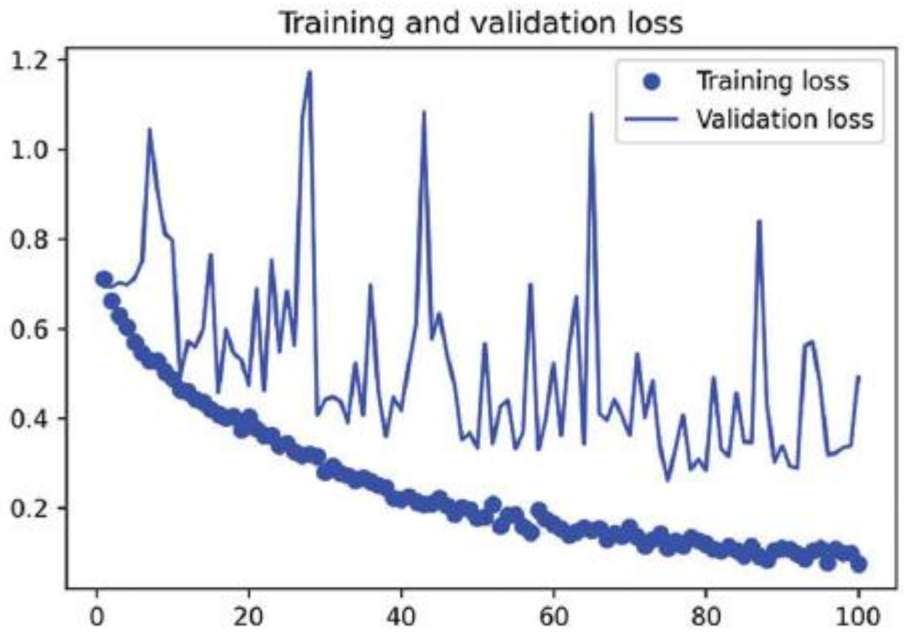
We apply a series of convolutional blocks with increasing feature depth. Each block consists of two batch-normalized depthwise separable convolution layers and a max pooling layer, with a residual connection around the entire block.

Note that the assumption that underlies separable convolution, "feature channels are largely independent," does not hold for RGB images! Red, green, and blue color channels are actually highly correlated in natural images. As such, the first layer in our model is a regular Conv2D layer. We'll start using SeparableConv2D afterwards.

Mini Xception-like Model on Cats vs Dogs

From scratch, without data augmentation …





Code for Visualizing Convolution Activations

```
from tensorflow.keras import layers

layer_outputs = []
layer_names = []
for layer in model.layers:
    if isinstance(layer, (layers.Conv2D, layers.MaxPooling2D)):
        layer_outputs.append(layer.output)
        layer_names.append(layer.name)
activation_model = keras.Model(inputs=model.input, outputs=layer_outputs)
```

Extract the outputs of all Conv2D and MaxPooling2D layers and put them in a list.

Save the layer names for later.

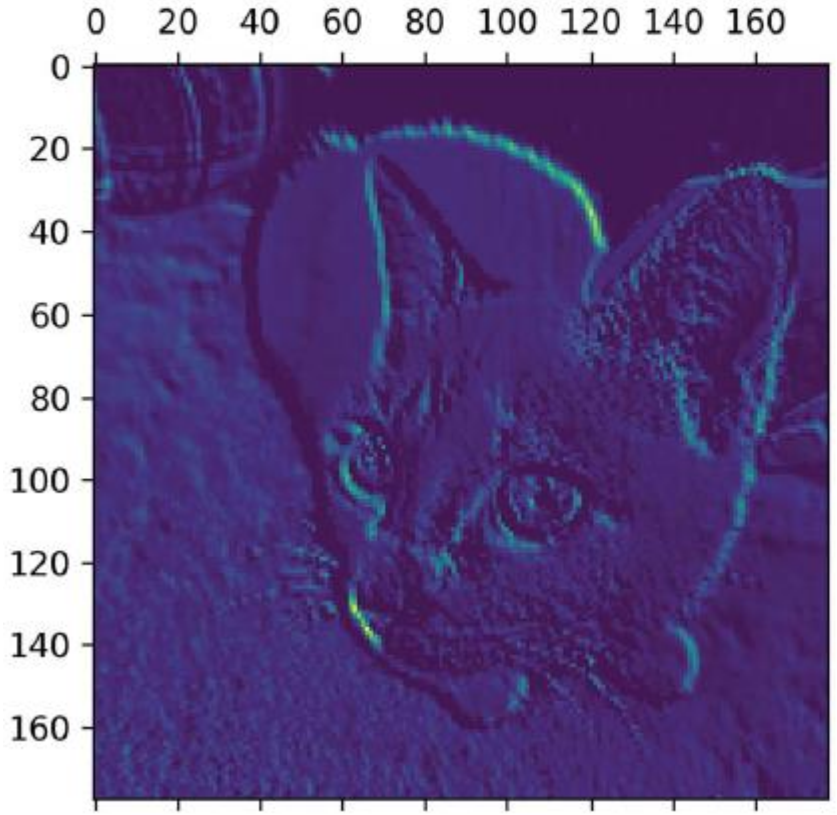
Create a model that will return these outputs, given the model input.

```
activations = activation_model.predict(img_tensor)
```

Return a list of nine NumPy arrays: one array per layer activation.

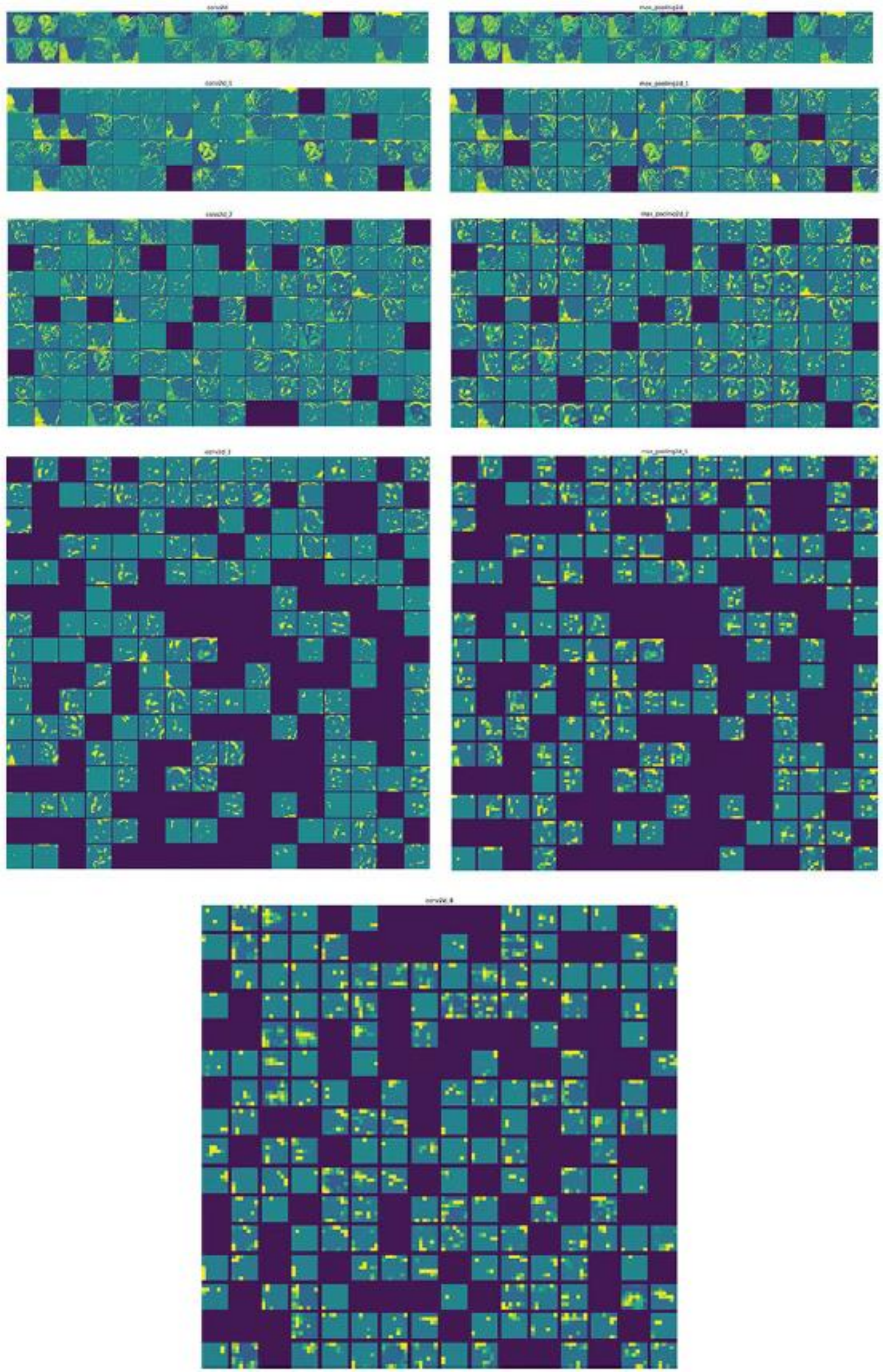
```
import matplotlib.pyplot as plt
plt.matshow(first_layer_activation[0, :, :, 5], cmap="viridis")
```

Visualizing Convolution Activations



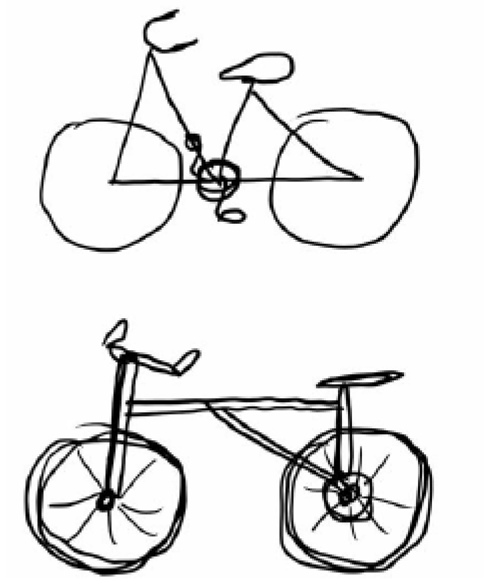
Visualizations

```
>>> from tensorflow import keras
>>> model = keras.models.load_model(
    "convnet_from_scratch_with_augmentation.keras")
>>> model.summary()
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 180, 180, 3)]     0
_________________________________________________________________
sequential (Sequential)      (None, 180, 180, 3)       0
_________________________________________________________________
rescaling_1 (Rescaling)      (None, 180, 180, 3)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 178, 178, 32)      896
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 89, 89, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 87, 87, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 43, 43, 64)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 41, 41, 128)       73856
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 20, 20, 128)       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 18, 18, 256)       295168
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 9, 9, 256)         0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 7, 7, 256)         590080
_________________________________________________________________
flatten_1 (Flatten)          (None, 12544)             0
_________________________________________________________________
dropout (Dropout)            (None, 12544)             0
_________________________________________________________________
dense 1 (Dense)              (None, 1)                 12545
```

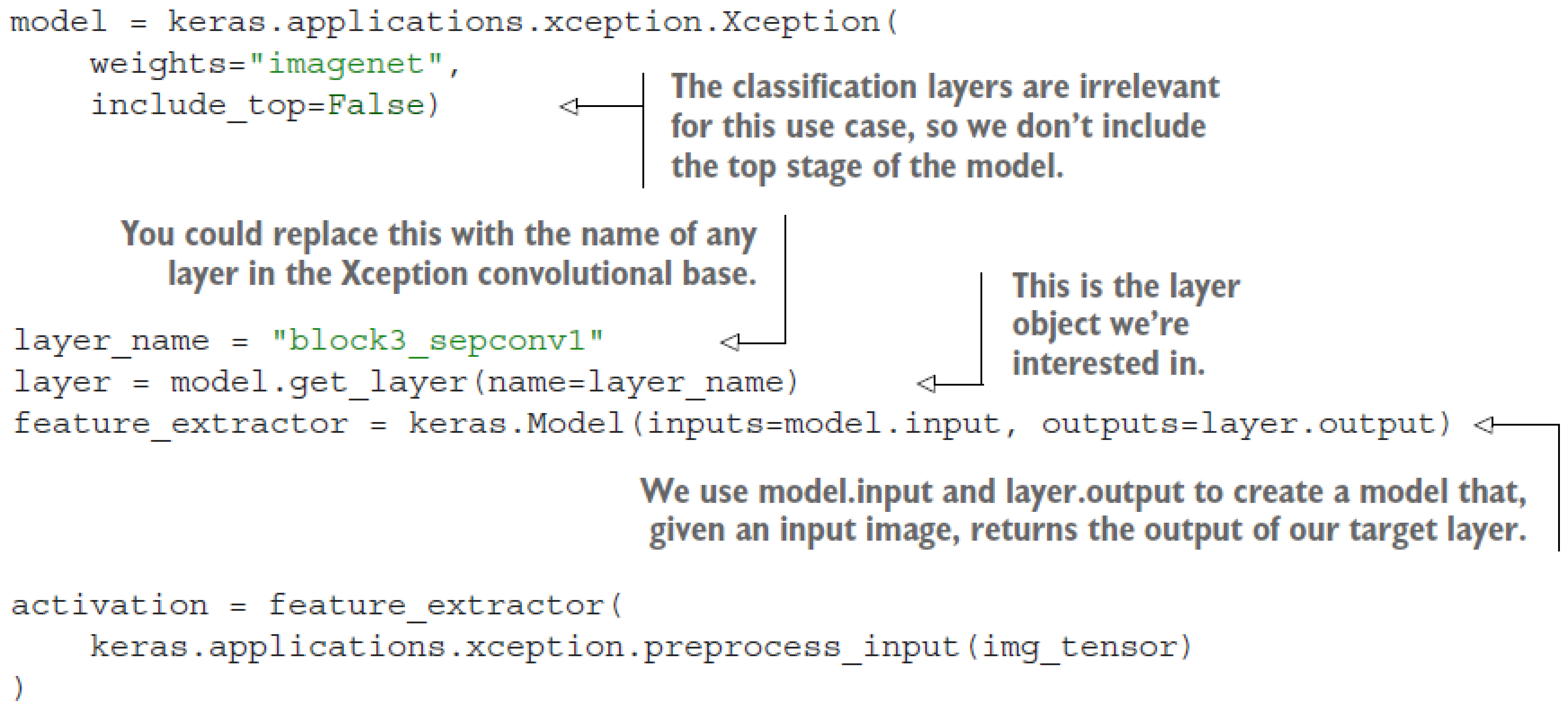


Higher-level Abstractions

"after observing a scene for a few seconds, a human can remember which abstract objects were present in it (bicycle, tree) but can’t remember the specific appearance of these objects"



Computing Activations



```
model = keras.applications.xception.Xception(
    weights="imagenet",
    include_top=False)
```

The classification layers are irrelevant for this use case, so we don't include the top stage of the model.

You could replace this with the name of any layer in the Xception convolutional base.

This is the layer object we're interested in.

```
layer_name = "block3_sepconv1"
layer = model.get_layer(name=layer_name)
feature_extractor = keras.Model(inputs=model.input, outputs=layer.output)
```

We use model.input and layer.output to create a model that, given an input image, returns the output of our target layer.

```
activation = feature_extractor(
    keras.applications.xception.preprocess_input(img_tensor)
)
```

"Loss" (Mean Filter Activation)

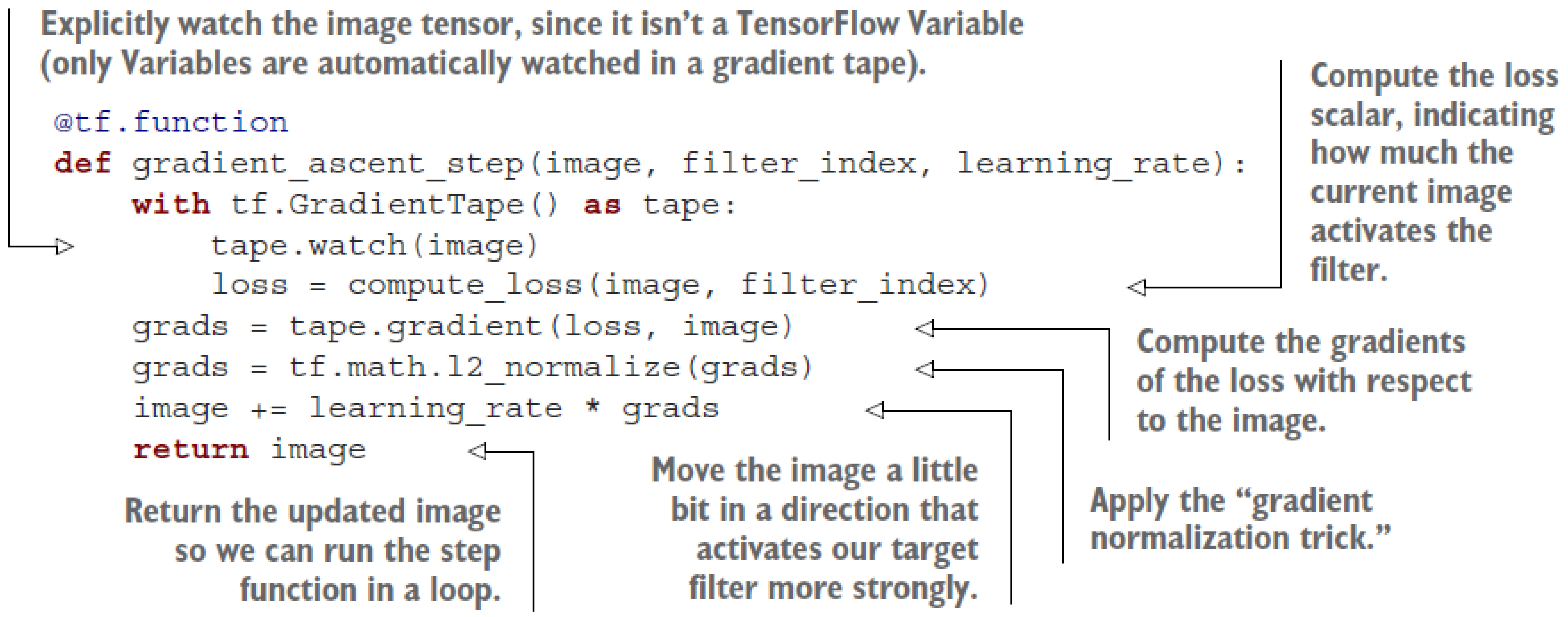
```
import tensorflow as tf

def compute_loss(image, filter_index):
    activation = feature_extractor(image)
    filter_activation = activation[:, 2:-2, 2:-2, filter_index]
    return tf.reduce_mean(filter_activation)
```

The loss function takes an image tensor and the index of the filter we are considering (an integer).

Note that we avoid border artifacts by only involving non-border pixels in the loss; we discard the first two pixels along the sides of the activation.

Return the mean of the activation values for the filter.

Gradient Ascent



Explicitly watch the image tensor, since it isn't a TensorFlow Variable (only Variables are automatically watched in a gradient tape).

```
@tf.function
def gradient_ascent_step(image, filter_index, learning_rate):
    with tf.GradientTape() as tape:
        tape.watch(image)
        loss = compute_loss(image, filter_index)
    grads = tape.gradient(loss, image)
    grads = tf.math.l2_normalize(grads)
    image += learning_rate * grads
    return image
```

Compute the loss scalar, indicating how much the current image activates the filter.

Compute the gradients of the loss with respect to the image.

Apply the "gradient normalization trick."

Move the image a little bit in a direction that activates our target filter more strongly.

Return the updated image so we can run the step function in a loop.

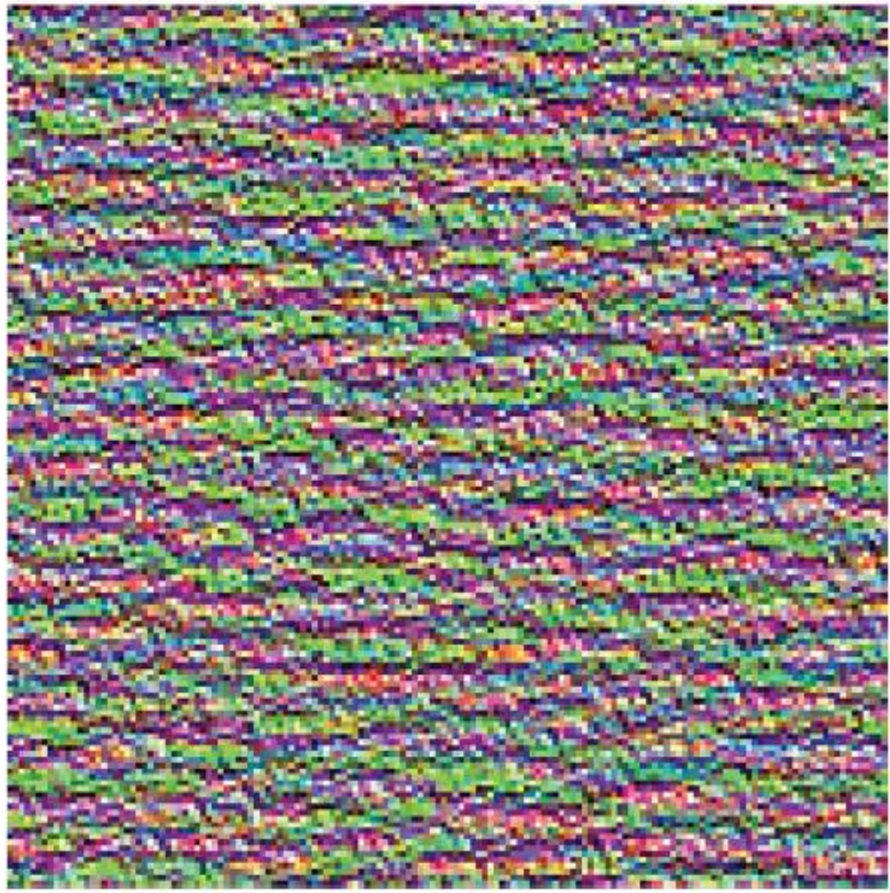
The “Party Worms” Filter

```
>>> plt.axis("off")
>>> plt.imshow(deprocess_image(generate_filter_pattern(filter_index=2)))
```

```
def deprocess_image(image):
    image -= image.mean()
    image /= image.std()
    image *= 64
    image += 128
    image = np.clip(image, 0, 255).astype("uint8")
    image = image[25:-25, 25:-25, :]
    return image
```

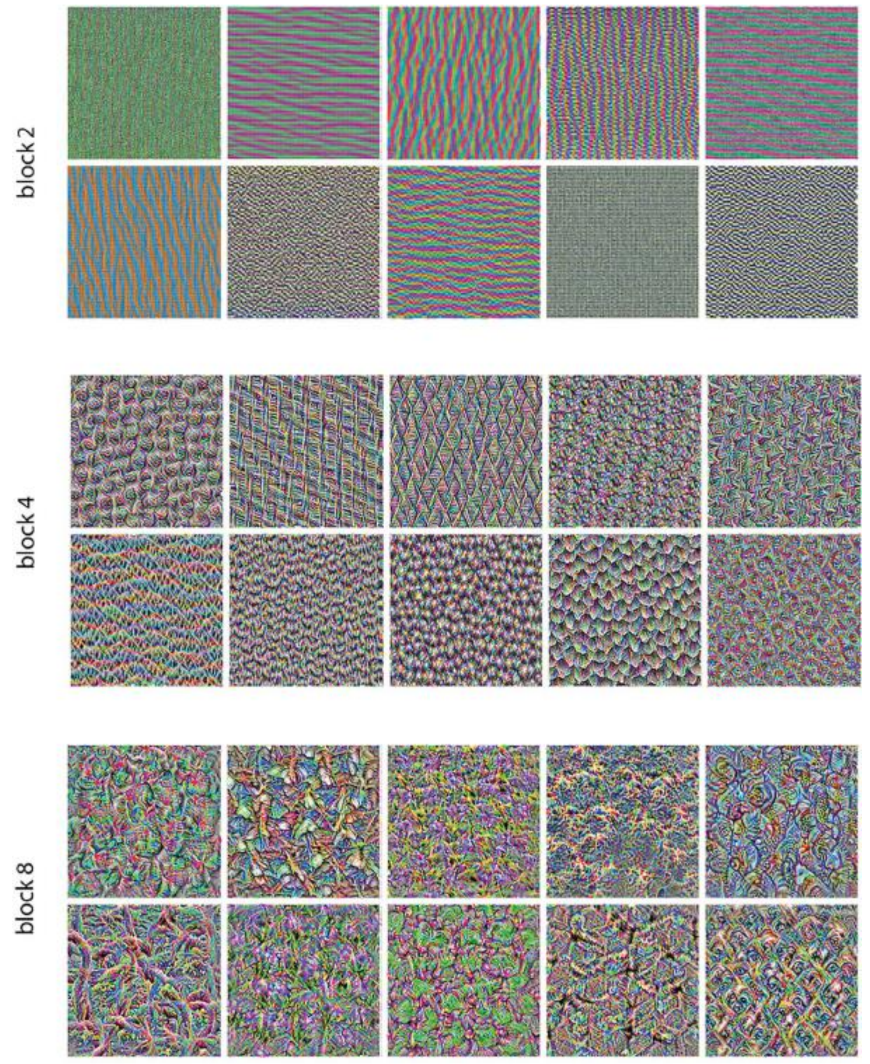
Normalize image values within the [0, 255] range.

Center crop to avoid border artifacts.



Presumably we’re mapping values from [-2, 2] to [0, 255]

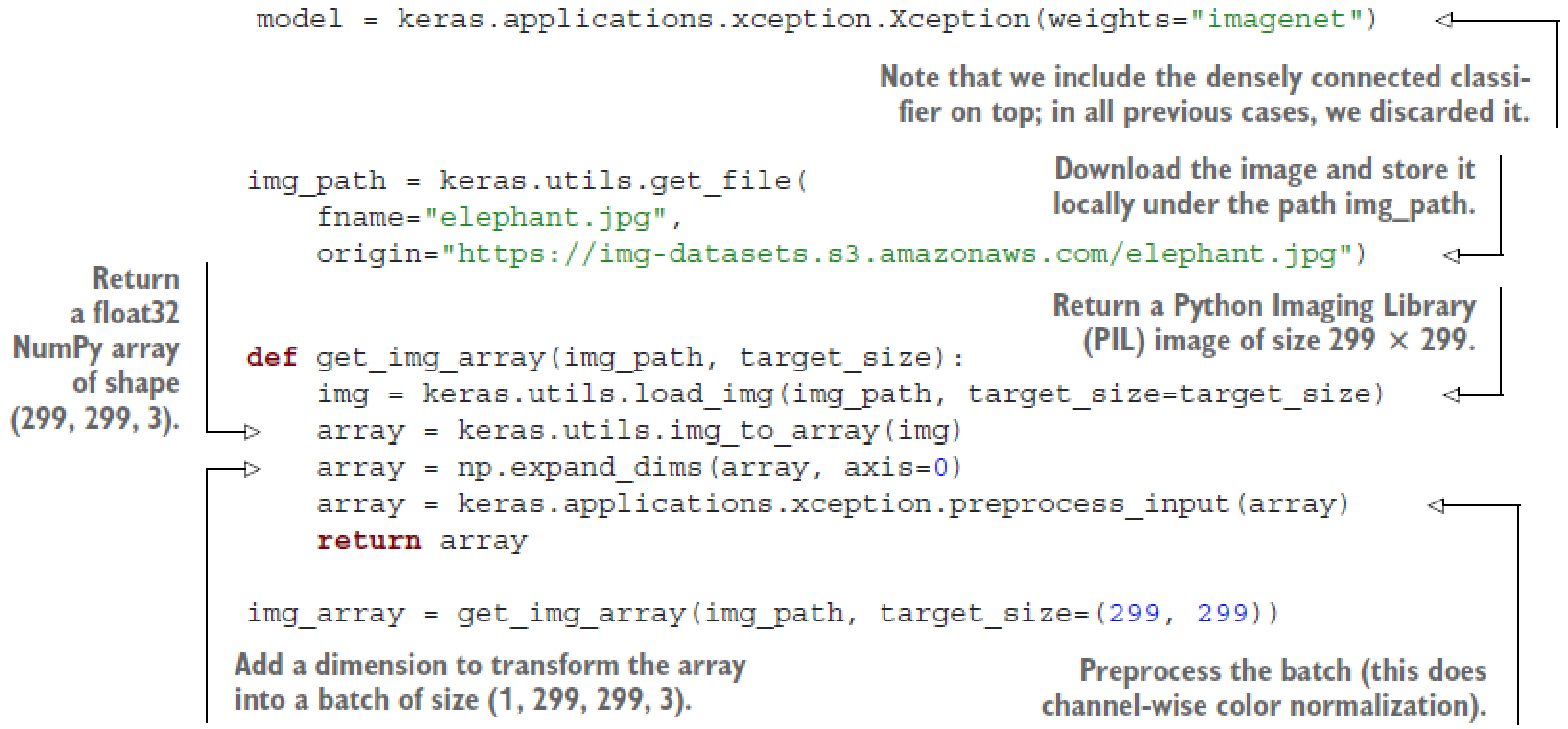
Optimal Filter Inputs for Sample Filters



Class Activation Map Heatmap

- Imagine that we're weighting a spatial map of "how intensely the input image activates different channels" (convolution activations) by "how important each channel is with regard to the class" (gradient of class activation with respect to convolution activation)
- Resulting in a spatial map of "how intensely the input image activates the class"

Loading Xception Model and an Image



```python
model = keras.applications.xception.Xception(weights="imagenet")

img_path = keras.utils.get_file(
    fname="elephant.jpg",
    origin="https://img-datasets.s3.amazonaws.com/elephant.jpg")

def get_img_array(img_path, target_size):
    img = keras.utils.load_img(img_path, target_size=target_size)
    array = keras.utils.img_to_array(img)
    array = np.expand_dims(array, axis=0)
    array = keras.applications.xception.preprocess_input(array)
    return array

img_array = get_img_array(img_path, target_size=(299, 299))
```

Note that we include the densely connected classifier on top; in all previous cases, we discarded it.

Download the image and store it locally under the path img_path.

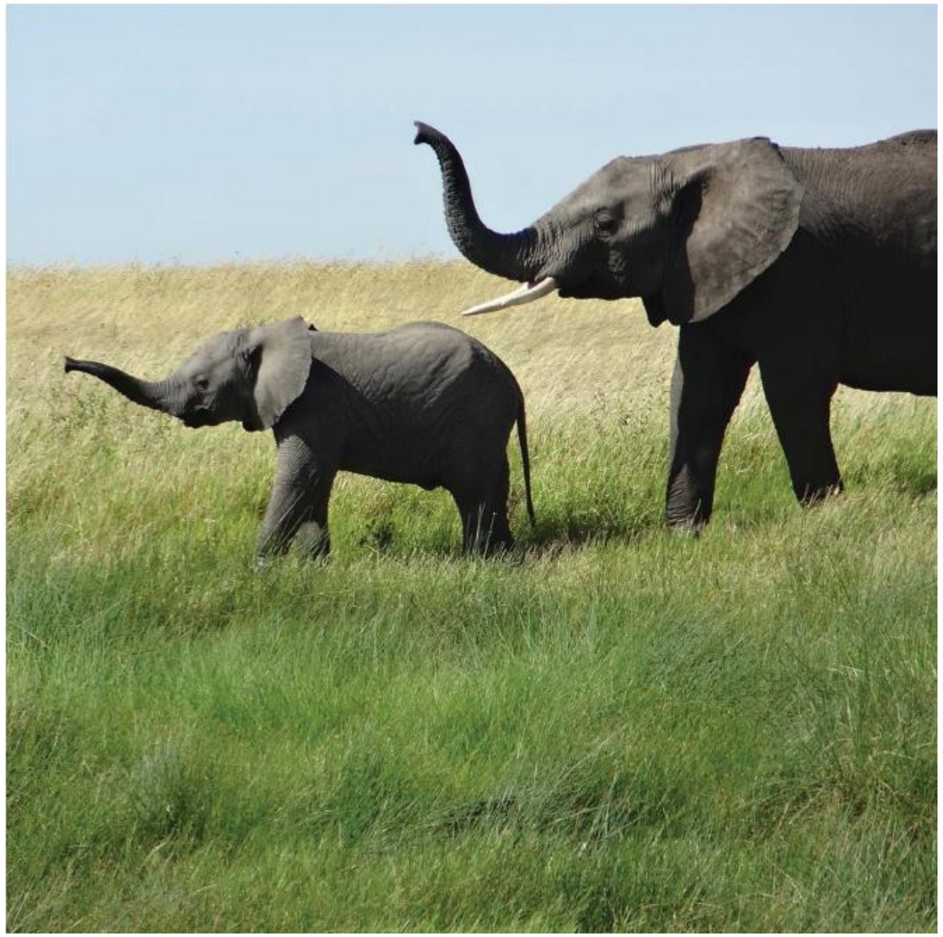
Return a Python Imaging Library (PIL) image of size 299 × 299.

Return a float32 NumPy array of shape (299, 299, 3).

Add a dimension to transform the array into a batch of size (1, 299, 299, 3).

Preprocess the batch (this does channel-wise color normalization).

African Elephant Image



```
>>> preds = model.predict(img_array)
>>> print(keras.applications.xception.decode_predictions(preds, top=3)[0])
[("n02504458", "African_elephant", 0.8699266),
 ("n01871265", "tusker", 0.076968715),
 ("n02504013", "Indian_elephant", 0.02353728)]
```

Computing the Gradients

```
last_conv_layer_name = "block14_sepconv2_act"
classifier_layer_names = [
    "avg_pool",
    "predictions",
]
last_conv_layer = model.get_layer(last_conv_layer_name)
last_conv_layer_model = keras.Model(model.inputs, last_conv_layer.output)

classifier_input = keras.Input(shape=last_conv_layer.output.shape[1:])
x = classifier_input
for layer_name in classifier_layer_names:
    x = model.get_layer(layer_name)(x)
classifier_model = keras.Model(classifier_input, x)

import tensorflow as tf

with tf.GradientTape() as tape:
    last_conv_layer_output = last_conv_layer_model(img_array)
    tape.watch(last_conv_layer_output)
    preds = classifier_model(last_conv_layer_output)
    top_pred_index = tf.argmax(preds[0])
    top_class_channel = preds[:, top_pred_index]

grads = tape.gradient(top_class_channel, last_conv_layer_output)
```

Compute activations of the last conv layer and make the tape watch it.

Retrieve the activation channel corresponding to the top predicted class.

This is the gradient of the top predicted class with regard to the output feature map of the last convolutional layer.

Standalone Class Activation Heatmap

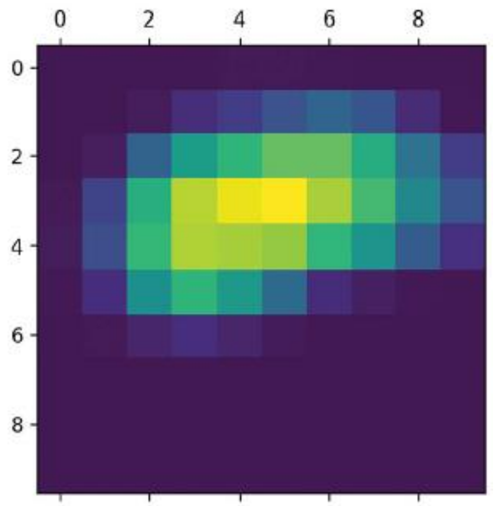
This is a vector where each entry is the mean intensity of the gradient for a given channel. It quantifies the importance of each channel with regard to the top predicted class.

```
pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2)).numpy()
last_conv_layer_output = last_conv_layer_output.numpy()[0]
for i in range(pooled_grads.shape[-1]):
    last_conv_layer_output[:, :, i] *= pooled_grads[i]
heatmap = np.mean(last_conv_layer_output, axis=-1)
```

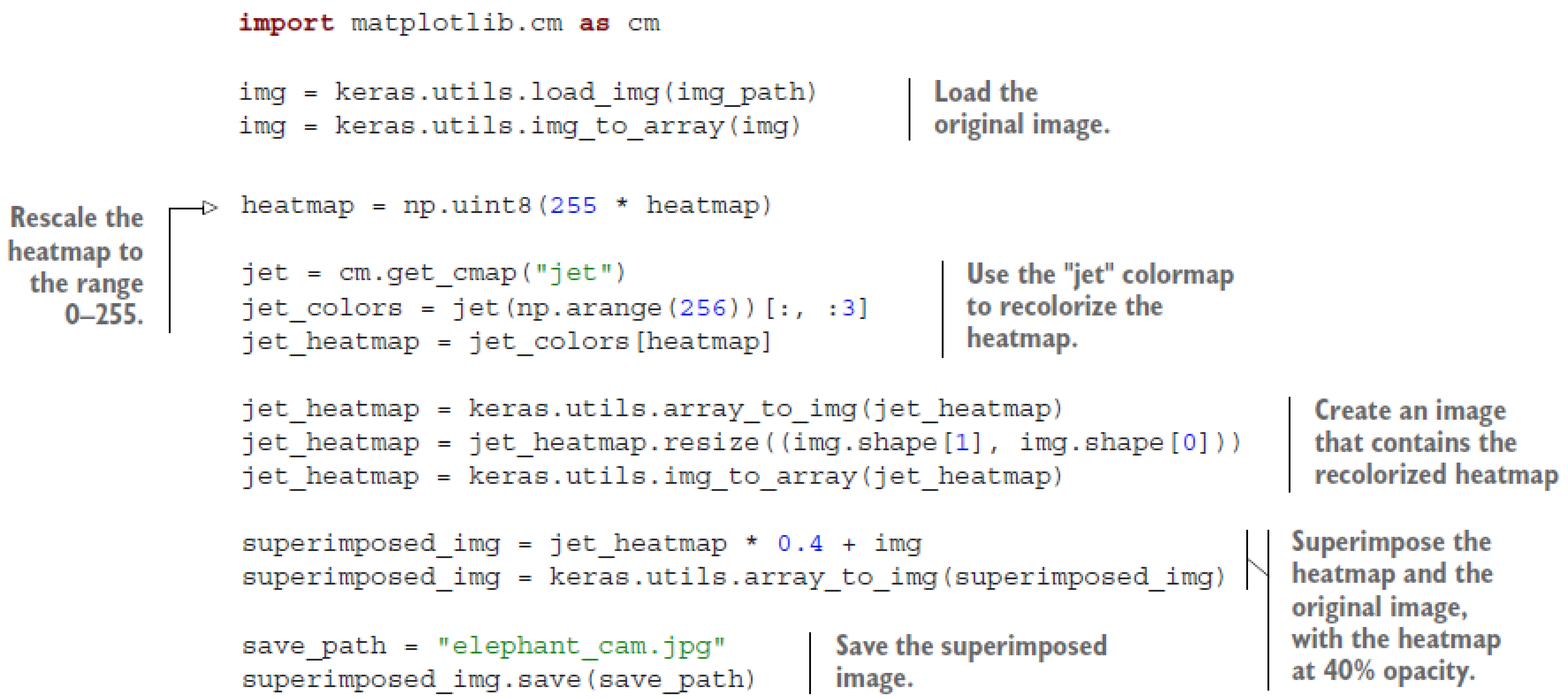
Multiply each channel in the output of the last convolutional layer by "how important this channel is."

The channel-wise mean of the resulting feature map is our heatmap of class activation.

```
heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
plt.matshow(heatmap)
```



Superimposing the Heatmap



```
import matplotlib.cm as cm

img = keras.utils.load_img(img_path)
img = keras.utils.img_to_array(img)

heatmap = np.uint8(255 * heatmap)

jet = cm.get_cmap("jet")
jet_colors = jet(np.arange(256))[:, :3]
jet_heatmap = jet_colors[heatmap]

jet_heatmap = keras.utils.array_to_img(jet_heatmap)
jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
jet_heatmap = keras.utils.img_to_array(jet_heatmap)

superimposed_img = jet_heatmap * 0.4 + img
superimposed_img = keras.utils.array_to_img(superimposed_img)

save_path = "elephant_cam.jpg"
superimposed_img.save(save_path)
```

Load the original image.

Rescale the heatmap to the range 0–255.

Use the "jet" colormap to recolorize the heatmap.

Create an image that contains the recolorized heatmap

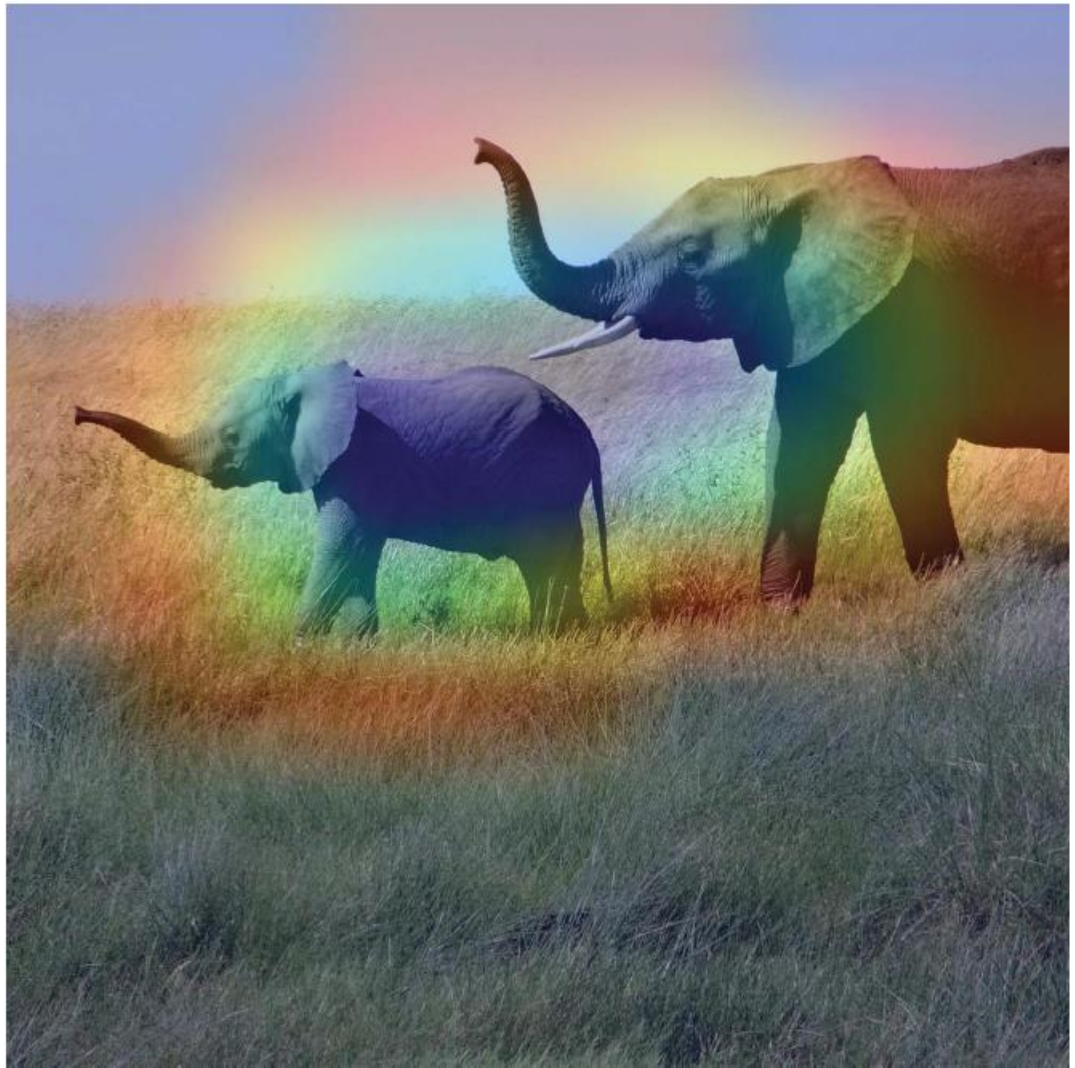
Superimpose the heatmap and the original image, with the heatmap at 40% opacity.

Save the superimposed image.

Class Activation Heatmap

This visualization technique answers two important questions:

- Why did the network think this image contained an African elephant?
- Where is the African elephant located in the picture?



Summary

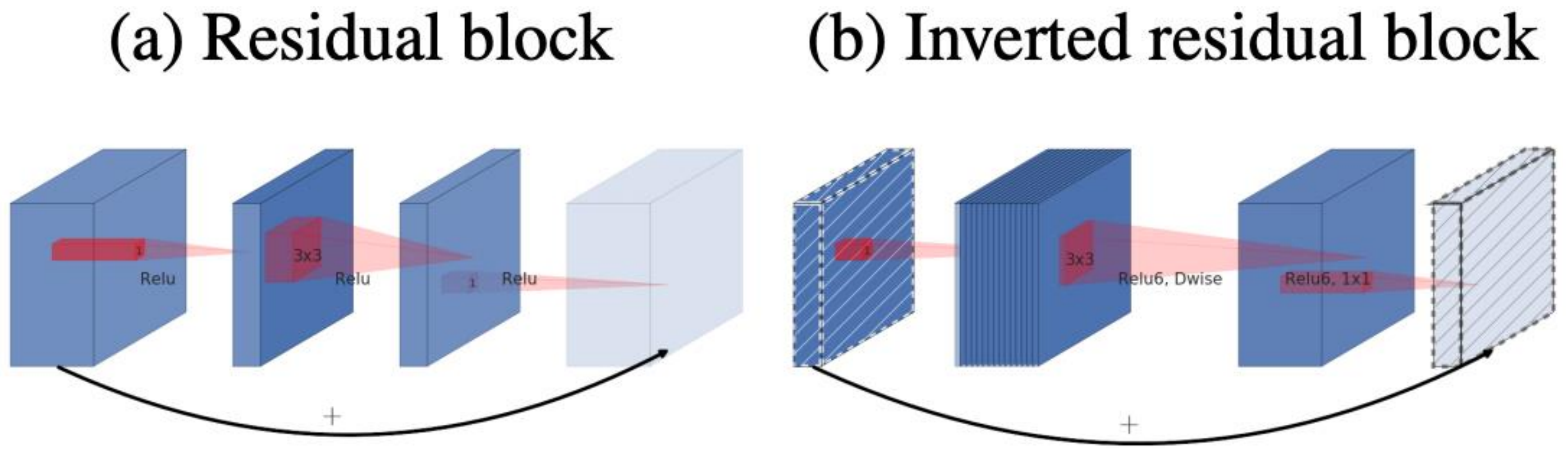
- There are three essential computer vision tasks you can do with deep learning: image classification, image segmentation, and object detection.
- Following modern convnet architecture best practices will help you get the most out of your models. Some of these best practices include using residual connections, batch normalization, and depthwise separable convolutions.
- The representations that convnets learn are easy to inspect—convnets are the opposite of black boxes!
- You can generate visualizations of the filters learned by your convnets, as well as heatmaps of class activity.

EfficientNet v2 Notes

https://paperswithcode.com/method/inverted-residual-block

Inverted Residual Block

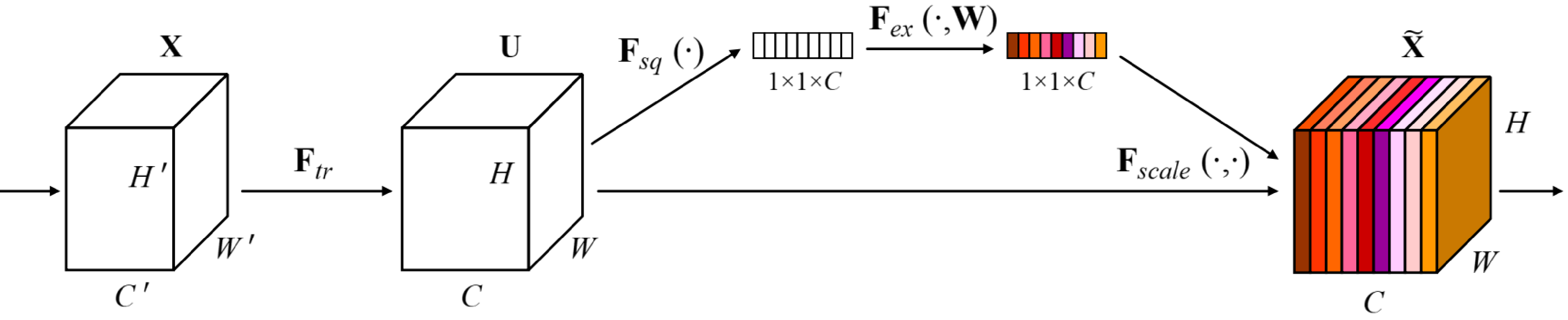
- Notice how the residual block has an identity connection between the wider layers (more channels), while the inverted residual block has an identity connection between the narrower layers (less channels)
- Sometimes called an MBConv block, because the inverted residual block structure was originally proposed for MobileNet version 2



https://github.com/keras-team/keras/blob/v2.9.0/keras/applications/efficientnet_v2.py#L607 https://paperswithcode.com/method/squeeze-and-excitation-block

Squeeze and Excitation Block

Sigmoid activation is applied to the colored 1x1xC features, which is then used to gate the feature maps (scaling features using coefficients in the interval [0, 1])



https://github.com/keras-team/keras/blob/v2.9.0/keras/applications/efficientnet_v2.py#L607