Chapter 14 – Deep Computer Vision Using Convolutional Neural Networks

This notebook contains all the sample code in chapter 14.

Run in Google Colab (https://colab.research.google.com/github/ageron/handson-ml2/blob/master/14_deep_computer_vision_with_cnns.ipynb)

Setup

First, let's import a few common modules, ensure MatplotLib plots figures inline and prepare a function to save the figures. We also check that Python 3.5 or later is installed (although Python 2.x may work, it is deprecated so we strongly recommend you use Python 3 instead), as well as Scikit-Learn ≥0.20 and TensorFlow ≥2.0.

In [1]:
```
# Python ≥3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Scikit-Learn ≥0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

try:
    # %tensorflow_version only exists in Colab.
    %tensorflow_version 2.x
    IS_COLAB = True
except Exception:
    IS_COLAB = False

# TensorFlow ≥2.0 is required
import tensorflow as tf
from tensorflow import keras
assert tf.__version__ >= "2.0"

if not tf.config.list_physical_devices('GPU'):
    print("No GPU was detected. CNNs can be very slow without a GPU.")
    if IS_COLAB:
        print("Go to Runtime > Change runtime and select a GPU hardware acce
lerator.")

# Common imports
import numpy as np
import os

# to make this notebook's output stable across runs
np.random.seed(42)
tf.random.set_seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "cnn"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=30
0):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

```
No GPU was detected. CNNs can be very slow without a GPU.
```

A couple utility functions to plot grayscale and RGB images:

In [2]:
```
def plot_image(image):
    plt.imshow(image, cmap="gray", interpolation="nearest")
    plt.axis("off")

def plot_color_image(image):
    plt.imshow(image, interpolation="nearest")
    plt.axis("off")
```

What is a Convolution?

In [3]:
```
import numpy as np
from sklearn.datasets import load_sample_image

# Load sample images
china = load_sample_image("china.jpg") / 255
flower = load_sample_image("flower.jpg") / 255
images = np.array([china, flower])
batch_size, height, width, channels = images.shape

# Create 2 filters
filters = np.zeros(shape=(7, 7, channels, 2), dtype=np.float32)
filters[:, 3, :, 0] = 1  # vertical line
filters[3, :, :, 1] = 1  # horizontal line

outputs = tf.nn.conv2d(images, filters, strides=1, padding="SAME")

plt.imshow(outputs[0, :, :, 1], cmap="gray") # plot 1st image's 2nd feature
map
plt.axis("off") # Not shown in the book
plt.show()
```

In [4]:
```
for image_index in (0, 1):
    for feature_map_index in (0, 1):
        plt.subplot(2, 2, image_index * 2 + feature_map_index + 1)
        plot_image(outputs[image_index, :, :, feature_map_index])

plt.show()
```

In [5]:
```
def crop(images):
    return images[150:220, 130:250]
```

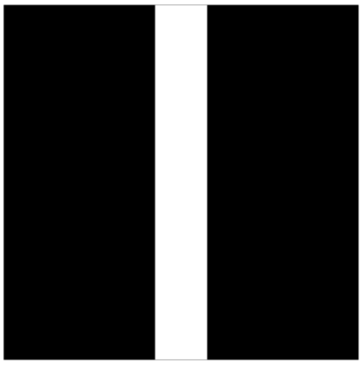
In [6]:
```
plot_image(crop(images[0, :, :, 0]))
save_fig("china_original", tight_layout=False)
plt.show()

for feature_map_index, filename in enumerate(["china_vertical", "china_horiz
ontal"]):
    plot_image(crop(outputs[0, :, :, feature_map_index]))
    save_fig(filename, tight_layout=False)
    plt.show()
```

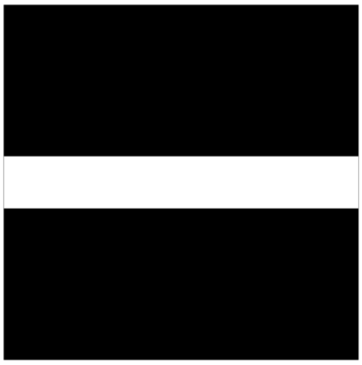
```
Saving figure china_original
```

```
Saving figure china_vertical
```

```
Saving figure china_horizontal
```

In [7]:
```
plot_image(filters[:, :, 0, 0])
plt.show()
plot_image(filters[:, :, 0, 1])
plt.show()
```





Convolutional Layer

Using `keras.layers.Conv2D()` :

In [8]:
```
conv = keras.layers.Conv2D(filters=32, kernel_size=3, strides=1,
                           padding="SAME", activation="relu")
```

VALID vs SAME padding

Confusingly, `"VALID"` padding means no padding at all.

In [10]:
```
kernel_size = 7
strides = 2

conv_valid = keras.layers.Conv2D(filters=1, kernel_size=kernel_size, strides
=strides, padding="VALID")
conv_same = keras.layers.Conv2D(filters=1, kernel_size=kernel_size, strides=
strides, padding="SAME")
```

Pooling layer

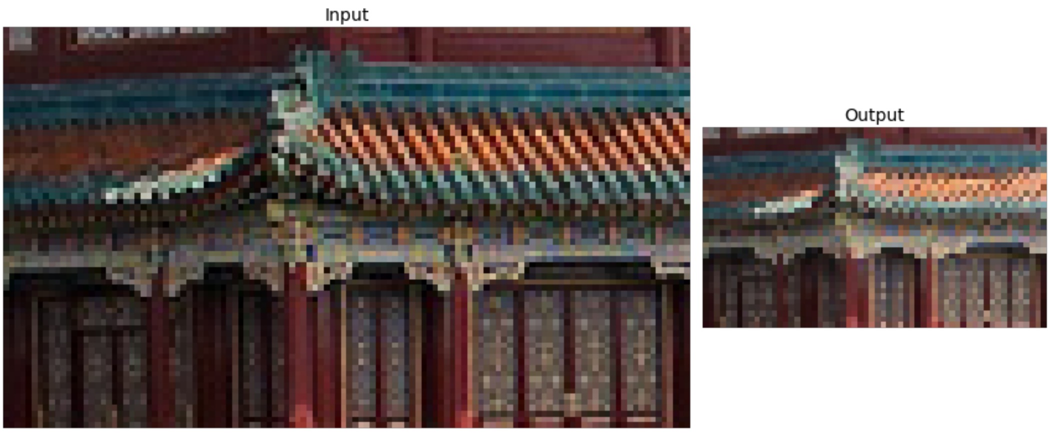
Max pooling

In [11]:
```
max_pool = keras.layers.MaxPool2D(pool_size=2)
```

In [12]:
```
cropped_images = np.array([crop(image) for image in images], dtype=np.float3
2)
output = max_pool(cropped_images)
```

In [13]:
```
fig = plt.figure(figsize=(12, 8))
gs = mpl.gridspec.GridSpec(nrows=1, ncols=2, width_ratios=[2, 1])

ax1 = fig.add_subplot(gs[0, 0])
ax1.set_title("Input", fontsize=14)
ax1.imshow(cropped_images[0])  # plot the 1st image
ax1.axis("off")
ax2 = fig.add_subplot(gs[0, 1])
ax2.set_title("Output", fontsize=14)
ax2.imshow(output[0])  # plot the output for the 1st image
ax2.axis("off")
save_fig("china_max_pooling")
plt.show()
```

```
Saving figure china_max_pooling
```



Average pooling

In [14]:
```
avg_pool = keras.layers.AvgPool2D(pool_size=2)
```

In [15]:
```
output_avg = avg_pool(cropped_images)
```

In [16]:
```
fig = plt.figure(figsize=(12, 8))
gs = mpl.gridspec.GridSpec(nrows=1, ncols=2, width_ratios=[2, 1])

ax1 = fig.add_subplot(gs[0, 0])
ax1.set_title("Input", fontsize=14)
ax1.imshow(cropped_images[0])  # plot the 1st image
ax1.axis("off")
ax2 = fig.add_subplot(gs[0, 1])
ax2.set_title("Output", fontsize=14)
ax2.imshow(output_avg[0])  # plot the output for the 1st image
ax2.axis("off")
plt.show()
```

Input

Output

Tackling Fashion MNIST With a CNN

In [17]:
```
(X_train_full, y_train_full), (X_test, y_test) = keras.datasets.fashion_mnis
t.load_data()
X_train, X_valid = X_train_full[:-5000], X_train_full[-5000:]
y_train, y_valid = y_train_full[:-5000], y_train_full[-5000:]

X_mean = X_train.mean(axis=0, keepdims=True)
X_std = X_train.std(axis=0, keepdims=True) + 1e-7
X_train = (X_train - X_mean) / X_std
X_valid = (X_valid - X_mean) / X_std
X_test = (X_test - X_mean) / X_std

X_train = X_train[..., np.newaxis]
X_valid = X_valid[..., np.newaxis]
X_test = X_test[..., np.newaxis]
```

Note : Partial functions allow one to derive a function with x parameters to a function with fewer parameters and fixed values set for the more limited function.

In [18]:
```
from functools import partial

DefaultConv2D = partial(keras.layers.Conv2D,
                        kernel_size=3, activation='relu', padding="SAME")

model = keras.models.Sequential([
    DefaultConv2D(filters=64, kernel_size=7, input_shape=[28, 28, 1]),
    keras.layers.MaxPooling2D(pool_size=2),
    DefaultConv2D(filters=128),
    DefaultConv2D(filters=128),
    keras.layers.MaxPooling2D(pool_size=2),
    DefaultConv2D(filters=256),
    DefaultConv2D(filters=256),
    keras.layers.MaxPooling2D(pool_size=2),
    keras.layers.Flatten(),
    keras.layers.Dense(units=128, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(units=64, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(units=10, activation='softmax'),
])
```

In [19]:
```
model.compile(loss="sparse_categorical_crossentropy", optimizer="nadam", met
rics=["accuracy"])
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_valid, y
_valid))
score = model.evaluate(X_test, y_test)
X_new = X_test[:10] # pretend we have new images
y_pred = model.predict(X_new)
```

```
Train on 55000 samples, validate on 5000 samples
Epoch 1/10
55000/55000 [==============================] - 435s 8ms/sample - loss: 0.7362
- accuracy: 0.7460 - val_loss: 0.3829 - val_accuracy: 0.8654
Epoch 2/10
55000/55000 [==============================] - 492s 9ms/sample - loss: 0.4270
- accuracy: 0.8576 - val_loss: 0.3241 - val_accuracy: 0.8802
Epoch 3/10
55000/55000 [==============================] - 498s 9ms/sample - loss: 0.3709
- accuracy: 0.8749 - val_loss: 0.3086 - val_accuracy: 0.8888
Epoch 4/10
55000/55000 [==============================] - 477s 9ms/sample - loss: 0.3346
- accuracy: 0.8892 - val_loss: 0.2978 - val_accuracy: 0.8894
Epoch 5/10
55000/55000 [==============================] - 477s 9ms/sample - loss: 0.3108
- accuracy: 0.8948 - val_loss: 0.2948 - val_accuracy: 0.8946
Epoch 6/10
55000/55000 [==============================] - 471s 9ms/sample - loss: 0.2981
- accuracy: 0.9005 - val_loss: 0.2871 - val_accuracy: 0.9024
Epoch 7/10
55000/55000 [==============================] - 470s 9ms/sample - loss: 0.2887
- accuracy: 0.9039 - val_loss: 0.2801 - val_accuracy: 0.8980
Epoch 8/10
55000/55000 [==============================] - 551s 10ms/sample - loss: 0.272
6 - accuracy: 0.9074 - val_loss: 0.2889 - val_accuracy: 0.9016
Epoch 9/10
55000/55000 [==============================] - 497s 9ms/sample - loss: 0.2634
- accuracy: 0.9116 - val_loss: 0.2937 - val_accuracy: 0.9000
Epoch 10/10
55000/55000 [==============================] - 534s 10ms/sample - loss: 0.252
9 - accuracy: 0.9153 - val_loss: 0.2959 - val_accuracy: 0.8956
10000/10000 [==============================] - 25s 2ms/sample - loss: 0.3124
- accuracy: 0.8961
```

Using a Pretrained Model

In [32]:
```
model = keras.applications.resnet50.ResNet50(weights="imagenet")
```

In [33]:
```
images_resized = tf.image.resize(images, [224, 224])
plot_color_image(images_resized[0])
plt.show()
```

In [34]:
```
images_resized = tf.image.resize_with_pad(images, 224, 224, antialias=True)
plot_color_image(images_resized[0])
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for f
loats or [0..255] for integers).
```

In [35]:
```
images_resized = tf.image.resize_with_crop_or_pad(images, 224, 224)
plot_color_image(images_resized[0])
plt.show()
```

In [36]:
```
china_box = [0, 0.03, 1, 0.68]
flower_box = [0.19, 0.26, 0.86, 0.7]
images_resized = tf.image.crop_and_resize(images, [china_box, flower_box],
[0, 1], [224, 224])
plot_color_image(images_resized[0])
plt.show()
plot_color_image(images_resized[1])
plt.show()
```

In [37]:
```
inputs = keras.applications.resnet50.preprocess_input(images_resized * 255)
Y_proba = model.predict(inputs)
```

In [38]:
```
Y_proba.shape
```

Out[38]:
```
(2, 1000)
```

In [39]:
```
top_K = keras.applications.resnet50.decode_predictions(Y_proba, top=3)
for image_index in range(len(images)):
    print("Image #{}".format(image_index))
    for class_id, name, y_proba in top_K[image_index]:
        print("  {} - {:12s} {:.2f}%".format(class_id, name, y_proba * 100))
    print()
```

```
Image #0
  n03877845 - palace       43.39%
  n02825657 - bell_cote    43.08%
  n03781244 - monastery    11.69%

Image #1
  n04522168 - vase         53.97%
  n07930864 - cup          9.52%
  n11939491 - daisy        4.96%
```

In [ ]: