In [1]:
```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt

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

In [2]:
```
# Importation de tensorflow
import tensorflow as tf
```

In [3]:
```
# Deux fonctions pour afficher des images à niveaux de gris et des images couleurs
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")
```

1 et 2 - Téléchargez le contenu des deux images (china.jpg et flower.jpg) et normalisation

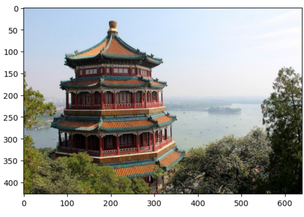
In [4]:
```
# Chargement de deux images en couleur à l'aide de
# la fonction load_sample_images() de Scikit-Learn
# Standardisation des images entre 0 et 1
# Et création d'un minilot d'entrée images
# Dans TensorFlow, chaque image d'entrée est
# représentée par un tenseur à 3 dimensions de forme
# [hauteur, largeur, nombre de canaux].
# Un mini-lot est représenté par un tenseur à quatre
# dimensions de forme [taille du mini-lot, hauteur,
# largeur, nombre de canaux].

import numpy as np
from sklearn.datasets import load_sample_image

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
```

3- Affichage et vérification des tailles des images

In [5]:
```
images.shape
```

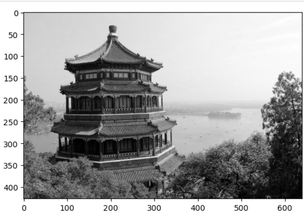
Out[5]:
```
(2, 427, 640, 3)
```

In [6]:
```
plt.imshow(china[:,:,:], cmap='gray')
plt.show()
```



In [7]:
```
plt.imshow(flower[:,:,:], cmap='gray')
plt.show()
```

In [8]:
```
plt.imshow(china[:,:,1], cmap='gray')
plt.show()
```



4 - Création des filtres

In [9]:
```
# Création de 2 filters 7*7, l'un avec une ligne blanche verticale
# centrale, l'autre avec une ligne blanche horizontale centrale
# filters correspond à l'ensemble des filtres à appliquer =
# un tenseur à quatre dimensions

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

5- Affichage du contenu des deux filtres

In [10]:
```
filters[:, :, 0, 0]
```

Out[10]:
```
array([[0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.]], dtype=float32)
```

In [11]:
```
filters[:, :, 0, 1]
```

Out[11]:
```
array([[0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0.],
       [1., 1., 1., 1., 1., 1., 1.],
       [0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

6-8 - Convolution

In [22]:
```
# images est le mini-lot d'entrée = le tenseur à quatre dimensions
# filters = l'ensemble des filtres à appliquer
# padding doit avoir pour valeur "SAME" ou "VALID"
# padding="SAME" : la couche de convolution ajoute une marge
# de zéros si nécessaire.
# padding="VALID" : la couche de convolution n'utilise pas
# de marge de zéros et peut ignorer certaines lignes et
# colonnes de l'images si nécessaire.
outputs = tf.nn.conv2d(images, filters, strides=1, padding="SAME")
```

In [19]:
```
outputs.shape
```

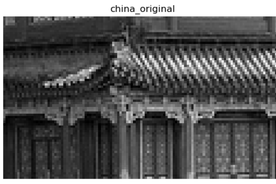
Out[19]:
```
TensorShape([2, 427, 640, 2])
```

9- Cartes de caractéristique

In [20]:
```
# La fonction crop est utilisée pour limiter l'analyse sur une zone
# et pour pouvoir visualiser l'impact du filtrage
def crop(images):
    return images[150:220, 130:250]
```

In [21]:
```
plot_image(crop(images[0, :, :, 0]))
plt.title("china_original")

plt.show()

for feature_map_index, filename in enumerate(["china_vertical", "china_horizontal"]):
    plot_image(crop(outputs[0, :, :, feature_map_index]))
    plt.show()
```

china_original



In [ ]:
```
```

In [ ]:
```
```