05. CNNs

In this notebook you will learn how to build Convolutional Neural Networks (CNNs) for image processing.

Imports

In [1]:
```
%matplotlib inline
```

In [2]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import sklearn
import sys
import tensorflow as tf
from tensorflow import keras
import time
```

In [ ]:
```
print("python", sys.version)
for module in mpl, np, pd, sklearn, tf, keras:
    print(module.__name__, module.__version__)
```

In [3]:
```
assert sys.version_info >= (3, 5) # Python ≥3.5 required
assert tf.__version__ >= "2.0"    # TensorFlow ≥2.0 required
```

Exercise 1 – Simple CNN

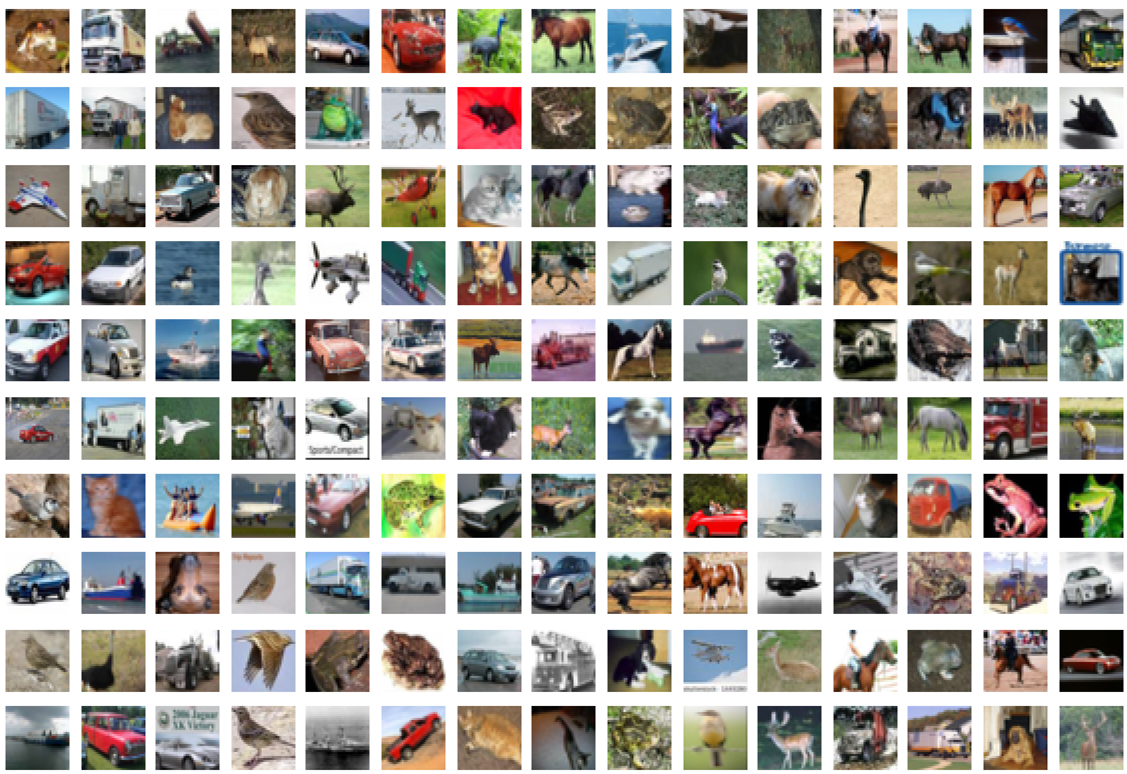
1.1)

Load CIFAR10 using `keras.datasets.cifar10.load_data()` , and split it into a training set (45,000 images), a validation set (5,000 images) and a test set (10,000 images). Make sure the pixel values range from 0 to 1. Visualize a few images using `plt.imshow()` .

In [4]:
```
classes = [
    "airplane",
    "automobile",
    "bird",
    "cat",
    "deer",
    "dog",
    "frog",
    "horse",
    "ship",
    "truck",
]
```

In [5]:
```
(X_train_full, y_train_full), (X_test, y_test) = keras.datasets.cifar10.load_d
ata()
X_train = X_train_full[:-5000] / 255
y_train = y_train_full[:-5000]
X_valid = X_train_full[-5000:] / 255
y_valid = y_train_full[-5000:]
X_test = X_test / 255
```

In [6]:
```
plt.figure(figsize=(10, 7))
n_rows, n_cols = 10, 15
for row in range(n_rows):
    for col in range(n_cols):
        i = row * n_cols + col
        plt.subplot(n_rows, n_cols, i + 1)
        plt.axis("off")
        plt.imshow(X_train[i])
```



Let's print the classes of the images in the first row:

In [7]:
```
for i in range(n_cols):
    print(classes[y_train[i][0]], end=" ")
```

```
frog truck truck deer automobile automobile bird horse ship cat deer horse hor
se bird truck
```

1.2)

Build and train a baseline model with a few dense layers, and plot the learning curves. Use the model's `summary()` method to count the number of parameters in this model.

Tip:

- Recall that to plot the learning curves, you can simply create a Pandas `DataFrame` with the `history.history` dict, then call its `plot()` method.

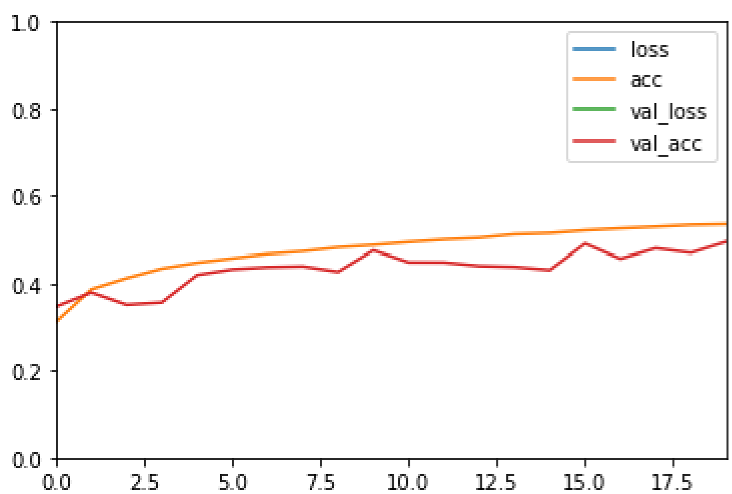
In [39]:

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[32, 32, 3]),
    keras.layers.Dense(64, activation="selu"),
    keras.layers.Dense(64, activation="selu"),
    keras.layers.Dense(64, activation="selu"),
    keras.layers.Dense(10, activation="softmax")
])
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=keras.optimizers.SGD(lr=0.01), metrics=["accuracy"])
history = model.fit(X_train, y_train, epochs=20,
                    validation_data=(X_valid, y_valid))
```

```
Train on 45000 samples, validate on 5000 samples
Epoch 1/20
45000/45000==============================] - 6s 130us/sample - loss: 1.9075 -
acc: 0.3125 - val_loss: 1.8396 - val_acc: 0.3474
Epoch 20/20
45000/45000==============================] - 5s 108us/sample - loss: 1.2987 -
acc: 0.5350 - val_loss: 1.4276 - val_acc: 0.4950
```

In [7]:

```
pd.DataFrame(history.history).plot()
plt.axis([0, 19, 0, 1])
plt.show()
```



In [8]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten (Flatten)            (None, 3072)              0         
_________________________________________________________________
dense (Dense)                (None, 64)                196672    
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4160      
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160      
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650       
=================================================================
Total params: 205,642
Trainable params: 205,642
Non-trainable params: 0
_________________________________________________________________
```

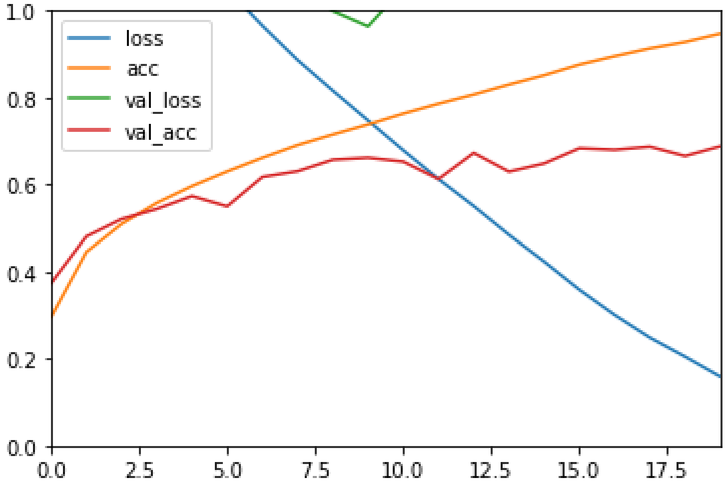
1.3)

Build and train a Convolutional Neural Network using a "classical" architecture: N * (Conv2D → Conv2D → Pool2D) → Flatten → Dense → Dense. Before you print the `summary()`, try to manually calculate the number of parameters in your model's architecture, as well as the shape of the inputs and outputs of each layer. Next, plot the learning curves and compare the performance with the previous model.

In [1]:
```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters=32, kernel_size=3, padding="same", activation=
"relu", input_shape=[32, 32, 3]),
    keras.layers.Conv2D(filters=32, kernel_size=3, padding="same", activation=
"relu"),
    keras.layers.MaxPool2D(pool_size=2),
    keras.layers.Conv2D(filters=64, kernel_size=3, padding="same", activation=
"relu"),
    keras.layers.Conv2D(filters=64, kernel_size=3, padding="same", activation=
"relu"),
    keras.layers.MaxPool2D(pool_size=2),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=keras.optimizers.SGD(lr=0.01), metrics=["accuracy"])
history = model.fit(X_train, y_train, epochs=20,
                    validation_data=(X_valid, y_valid))
```

```
Train on 45000 samples, validate on 5000 samples
Epoch 1/20
45000/45000==============================] - 9s 191us/sample - loss: 1.9518 -
acc: 0.2957 - val_loss: 1.7348 - val_acc: 0.3736
 ⋮
Epoch 20/20
45000/45000==============================] - 6s 125us/sample - loss: 0.1591 -
acc: 0.9471 - val_loss: 1.4888 - val_acc: 0.6878
```

In [10]:
```
pd.DataFrame(history.history).plot()
plt.axis([0, 19, 0, 1])
plt.show()
```



In [11]:
```
# Number of params in a convolutional layer =
# (kernel_width * kernel_height * channels_in + 1 for bias) * channels_out
(
    (3 * 3 * 3 + 1)  * 32  # in: 32x32x3   out: 32x32x32  Conv2D
  + (3 * 3 * 32 + 1) * 32  # in: 32x32x32  out: 32x32x32  Conv2D
  + 0                      # in: 32x32x32  out: 16x16x32  MaxPool2D
  + (3 * 3 * 32 + 1) * 64  # in: 16x16x32  out: 16x16x64  Conv2D
  + (3 * 3 * 64 + 1) * 64  # in: 16x16x64  out: 16x16x64  Conv2D
  + 0                      # in: 16x16x64  out: 8x8x64    MaxPool2D
  + 0                      # in: 8x8x64    out: 4096      Flatten
  + (4096 + 1) * 128       # in: 4096      out: 128       Dense
  + (128 + 1) * 10         # in: 128       out: 10        Dense
)
```

Out[11]:
```
591274
```

Let's check:

In [12]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896       
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248      
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496     
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        36928     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 4096)              0         
_________________________________________________________________
dense_4 (Dense)              (None, 128)               524416    
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290      
=================================================================
Total params: 591,274
Trainable params: 591,274
Non-trainable params: 0
_________________________________________________________________
```

1.4)

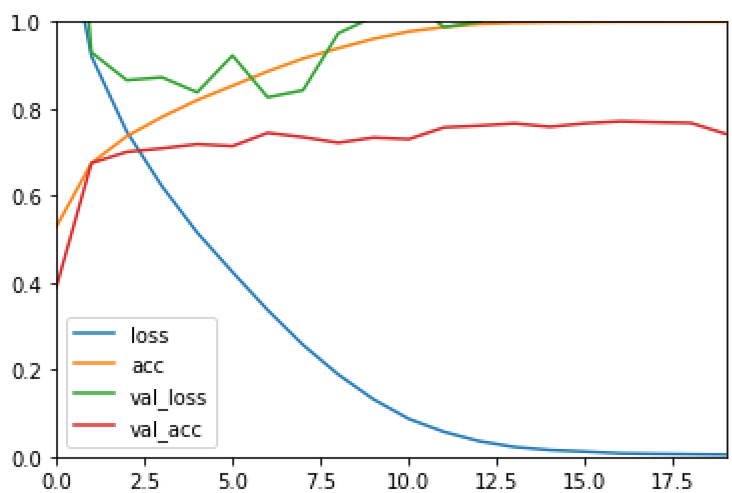
Looking at the learning curves, you can see that the model is overfitting. Add a Batch Normalization layer after each convolutional layer. Compare the model's performance and learning curves with the previous model.

Tip: there is no need for an activation function just before the pooling layers.

In [2]:
```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters=32, kernel_size=3, padding="same", activation=
"relu", input_shape=[32, 32, 3]),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=32, kernel_size=3, padding="same", activation=
"relu"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=2),
    keras.layers.Conv2D(filters=64, kernel_size=3, padding="same", activation=
"relu"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=64, kernel_size=3, padding="same", activation=
"relu"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=2),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=keras.optimizers.SGD(lr=0.01), metrics=["accuracy"])
history = model.fit(X_train, y_train, epochs=20,
                    validation_data=(X_valid, y_valid))
```

```
Train on 45000 samples, validate on 5000 samples
Epoch 1/20
45000/45000==============================] - 8s 170us/sample - loss: 1.3353 -
acc: 0.5280 - val_loss: 1.9303 - val_acc: 0.3884
 ⋮
Epoch 20/20
45000/45000==============================] - 7s 158us/sample - loss: 0.0045 -
acc: 1.0000 - val_loss: 1.3410 - val_acc: 0.7418
```

In [14]:
```
pd.DataFrame(history.history).plot()
plt.axis([0, 19, 0, 1])
plt.show()
```



Exercise 2 – Separable Convolutions

2.1)

Replace the `Conv2D` layers with `SeparableConv2D` layers (except the first one), fit your model and compare its performance and learning curves with the previous model.

In [42]:

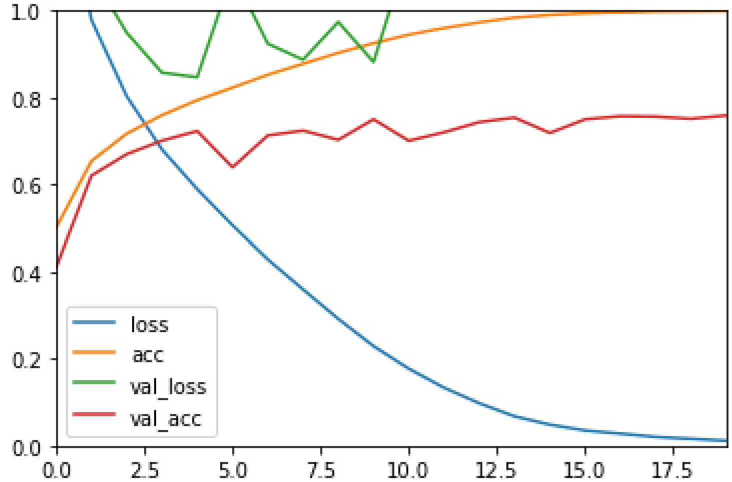
```

```

In [3]:

```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters=32, kernel_size=3, padding="same", activation=
"relu", input_shape=[32, 32, 3]),
    keras.layers.BatchNormalization(),
    keras.layers.SeparableConv2D(filters=32, kernel_size=3, padding="same", ac
tivation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=2),
    keras.layers.SeparableConv2D(filters=64, kernel_size=3, padding="same", ac
tivation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.SeparableConv2D(filters=64, kernel_size=3, padding="same", ac
tivation="relu"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=2),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
model.compile(loss="sparse_categorical_crossentropy",
              optimizer=keras.optimizers.SGD(lr=0.01), metrics=["accuracy"])
history = model.fit(X_train, y_train, epochs=20,
                    validation_data=(X_valid, y_valid))
```

```
Train on 45000 samples, validate on 5000 samples
Epoch 1/20
45000/45000==============================] - 8s 179us/sample - loss: 1.3953 -
acc: 0.5008 - val_loss: 1.7258 - val_acc: 0.4102
 ⋮
Epoch 20/20
45000/45000==============================] - 7s 163us/sample - loss: 0.0114 -
acc: 0.9995 - val_loss: 1.2084 - val_acc: 0.7588
```

In [16]:
```
pd.DataFrame(history.history).plot()
plt.axis([0, 19, 0, 1])
plt.show()
```



2.2)

Try to estimate the number of parameters in your network, then check your result with `model.summary()`.

Tip: the batch normalization layer adds two parameters for each feature map (the scale and bias).

In [19]:
```
# Number of params in a depthwise separable 2D convolution layer =
# kernel_width * kernel_height * channels_in + (channels_in + 1 for bias) * ch
annels_out
(
    (3 * 3 * 3 + 1) * 32        # in: 32x32x3   out: 32x32x32  Conv2D
  + 32 * 2                      # in: 32x32x32  out: 32x32x32  BN
  + 3 * 3 * 32 + (32 + 1) * 32  # in: 32x32x32  out: 32x32x32  SeparableConv2D
  + 32 * 2                      # in: 32x32x32  out: 32x32x32  BN
  + 0                           # in: 32x32x32  out: 16x16x32  MaxPool2D
  + 3 * 3 * 32 + (32 + 1) * 64  # in: 16x16x32  out: 16x16x64  SeparableConv2D
  + 64 * 2                      # in: 16x16x64  out: 16x16x64  BN
  + 3 * 3 * 64 + (64 + 1) * 64  # in: 16x16x64  out: 16x16x64  SeparableConv2D
  + 64 * 2                      # in: 16x16x64  out: 16x16x64  BN
  + 0                           # in: 16x16x64  out: 8x8x64    MaxPool2D
  + 0                           # in: 8x8x64    out: 4096      Flatten
  + (4096 + 1) * 128            # in: 4096      out: 128       Dense
  + (128 + 1) * 10              # in: 128       out: 10        Dense
)
```

Out[19]: 535466

Let's check:

In [20]:
```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_8 (Conv2D)            (None, 32, 32, 32)        896       
_________________________________________________________________
batch_normalization_v2_4 (Ba (None, 32, 32, 32)        128       
_________________________________________________________________
separable_conv2d (SeparableC (None, 32, 32, 32)        1344      
_________________________________________________________________
batch_normalization_v2_5 (Ba (None, 32, 32, 32)        128       
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 16, 16, 32)        0         
_________________________________________________________________
separable_conv2d_1 (Separabl (None, 16, 16, 64)        2400      
_________________________________________________________________
batch_normalization_v2_6 (Ba (None, 16, 16, 64)        256       
_________________________________________________________________
separable_conv2d_2 (Separabl (None, 16, 16, 64)        4736      
_________________________________________________________________
batch_normalization_v2_7 (Ba (None, 16, 16, 64)        256       
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 8, 8, 64)          0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 4096)              0         
_________________________________________________________________
dense_8 (Dense)              (None, 128)               524416    
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1290      
=================================================================
Total params: 535,850
Trainable params: 535,466
Non-trainable params: 384
_________________________________________________________________
```

Exercise 3 – Pretrained CNNs

3.1)

Using `keras.preprocessing.image.load_img()` followed by `keras.preprocessing.image.img_to_array()`, load one or more images (e.g., `fig.jpg` or `ostrich.jpg` in the `images` folder). You should set `target_size=(299, 299)` when calling `load_img()`, as this is the shape that the Xception network expects.

In [22]:
```
img_fig_path = os.path.join("images", "fig.jpg")
img_fig = keras.preprocessing.image.load_img(img_fig_path, target_size=(299, 2
99))
img_fig = keras.preprocessing.image.img_to_array(img_fig)
```

In [23]:
```
plt.imshow(img_fig / 255)
plt.axis("off")
plt.show()
img_fig.shape
```

Out[23]: (299, 299, 3)

In [24]:
```
img_ostrich_path = os.path.join("images", "ostrich.jpg")
img_ostrich = keras.preprocessing.image.load_img(img_ostrich_path, target_size
=(299, 299))
img_ostrich = keras.preprocessing.image.img_to_array(img_ostrich)
```

In [25]:
```
plt.imshow(img_ostrich / 255)
plt.axis("off")
plt.show()
img_ostrich.shape
```

Out[25]: (299, 299, 3)

3.2)

Create a batch containing the image(s) you just loaded, and preprocess this batch using `keras.applications.xception.preprocess_input()`. Verify that the features now vary from -1 to 1: this is what the Xception network expects.

In [26]:
```
X_batch = np.array([img_fig, img_ostrich])
X_preproc = keras.applications.xception.preprocess_input(X_batch)
```

In [27]:
```
X_preproc.min(), X_preproc.max()
```

Out[27]:
```
(-1.0, 1.0)
```

3.3)

Create an instance of the Xception model (`keras.applications.xception.Xception`) and use its `predict()` method to classify the images in the batch. You can use `keras.applications.resnet50.decode_predictions()` to convert the output matrix into a list of top-N predictions (with their corresponding class labels).

In [28]:
```
model = keras.applications.xception.Xception()
Y_proba = model.predict(X_preproc)
```

```
Downloading data from https://github.com/fchollet/deep-learning-models/release
s/download/v0.4/xception_weights_tf_dim_ordering_tf_kernels.h5
91889664/91884032==============================] - 3s 0us/step
```

In [29]:
```
Y_proba.shape
```

Out[29]:
```
(2, 1000)
```

In [30]:
```
np.argmax(Y_proba, axis=1)
```

Out[30]:
```
array([952,   9])
```

In [31]:
```
decoded_predictions = keras.applications.resnet50.decode_predictions(Y_proba)
for preds in decoded_predictions:
    for wordnet_id, name, proba in preds:
        print("{} ({}): {:.1f}%".format(name, wordnet_id, 100 * proba))
    print()
```

```
Downloading data from https://s3.amazonaws.com/deep-learning-models/image-mode
ls/imagenet_class_index.json
40960/35363==================================] - 0s 2us/step
fig (n07753113): 99.9%
grocery_store (n03461385): 0.0%
mushroom (n07734744): 0.0%
butternut_squash (n07717556): 0.0%
pomegranate (n07768694): 0.0%

ostrich (n01518878): 98.8%
bustard (n02018795): 0.1%
black_swan (n01860187): 0.0%
white_stork (n02002556): 0.0%
cock (n01514668): 0.0%
```

Exercise 4 – Data Augmentation and Transfer Learning

In this exercise you will reuse a pretrained Xception model to build a flower classifier.

First, let's download the dataset:

In [32]:
```
import tensorflow as tf
from tensorflow import keras
import os

flowers_url = "http://download.tensorflow.org/example_images/flower_photos.tg
z"
flowers_path = keras.utils.get_file("flowers.tgz", flowers_url, extract=True)
flowers_dir = os.path.join(os.path.dirname(flowers_path), "flower_photos")
```

```
Downloading data from http://download.tensorflow.org/example_images/flower_pho
tos.tgz
228818944/228813984==============================] - 4s 0us/step
```

In [33]:
```
for root, subdirs, files in os.walk(flowers_dir):
    print(root)
    for filename in files[:3]:
        print("    ", filename)
    if len(files) > 3:
        print("    ...")
```

```
/home/jupyter/.keras/datasets/flower_photos
    LICENSE.txt
/home/jupyter/.keras/datasets/flower_photos/roses
    3664842094_5fd60ee26b.jpg
    459042023_6273adc312_n.jpg
    5492988531_574cdc2bf0_n.jpg
    ...
/home/jupyter/.keras/datasets/flower_photos/dandelion
    515143813_b3afb08bf9.jpg
    14085038920_2ee4ce8a8d.jpg
    3696596109_4c4419128a_m.jpg
    ...
/home/jupyter/.keras/datasets/flower_photos/sunflowers
    6606823367_e89dc52a95_n.jpg
    5998488415_a6bacd9f83.jpg
    678714585_addc9aaaef.jpg
    ...
/home/jupyter/.keras/datasets/flower_photos/tulips
    16169741783_deeab1a679_m.jpg
    13539384593_23449f7332_n.jpg
    16711791713_e54bc9c1af_n.jpg
    ...
/home/jupyter/.keras/datasets/flower_photos/daisy
    9345273630_af3550031d.jpg
    14088053307_1a13a0bf91_n.jpg
    4746633946_23933c0810.jpg
    ...
```

4.1)

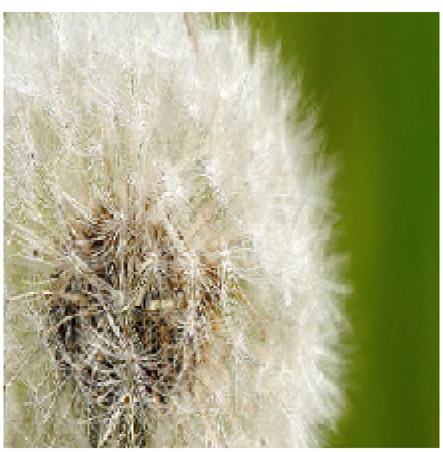
Build a `keras.preprocessing.image.ImageDataGenerator` that will preprocess the images and do some data augmentation (the documentation contains useful examples):

- It should at least perform horizontal flips and keep 10% of the data for validation, but you may also make it perform a bit of rotation, rescaling, etc.
- Also make sure to apply the Xception preprocessing function (using the `preprocessing_function` argument).
- Call this generator's `flow_from_directory()` method to get an iterator that will load and preprocess the flower photos from the `flower_photos` directory, setting the target size to (299, 299) and `subset` to `"training"`.
- Call this method again with the same parameters except `subset="validation"` to get a second iterator for validation.
- Get the next batch from the validation iterator and display the first image from the batch.

In [34]:
```
datagen = keras.preprocessing.image.ImageDataGenerator(
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.1,
    preprocessing_function=keras.applications.xception.preprocess_input)

train_generator = datagen.flow_from_directory(
        flowers_dir,
        target_size=(299, 299),
        batch_size=32,
        subset="training")

valid_generator = datagen.flow_from_directory(
        flowers_dir,
        target_size=(299, 299),
        batch_size=32,
        subset="validation")
```

```
Found 3306 images belonging to 5 classes.
Found 364 images belonging to 5 classes.
```

In [35]:
```
X_batch, y_batch = next(valid_generator)
plt.imshow((X_batch[0] + 1)/2)
plt.axis("off")
plt.show()
```



4.2)

Now let's build the model:

- Create a new `Xception` model, but this time set `include_top=False` to get the model without the top layer. **Tip**: you will need to access its `input` and `output` properties.
- Make all its layers non-trainable.
- Using the functional API, add a `GlobalAveragePooling2D` layer (feeding it the Xception model's output), and add a `Dense` layer with 5 neurons and the Softmax activation function.
- Compile the model. **Tip**: don't forget to add the `"accuracy"` metric.
- Fit your model using `fit_generator()`, passing it the training and validation iterators (and setting `steps_per_epoch` and `validation_steps` appropriately).

In [36]:
```
n_classes = 5

base_model = keras.applications.xception.Xception(include_top=False)

for layer in base_model.layers:
    layer.trainable = False

global_pool = keras.layers.GlobalAveragePooling2D()(base_model.output)
predictions = keras.layers.Dense(n_classes, activation='softmax')(global_pool)

model = keras.models.Model(base_model.input, predictions)

model.compile(loss="categorical_crossentropy",
              optimizer="sgd", metrics=["accuracy"])
```

```
Downloading data from https://github.com/fchollet/deep-learning-models/release
s/download/v0.4/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83689472/83683744==============================] - 3s 0us/step
```

In [38]:
```
history = model.fit_generator(
    train_generator,
    steps_per_epoch=3306 // 32,
    epochs=50,
    validation_data=valid_generator,
    validation_steps=364 // 32)
```

In [ ]:
```
pd.DataFrame(history.history).plot()
plt.axis([0, 19, 0, 1])
plt.show()
```

Object Detection Project

The Google [Street View House Numbers]() (SVHN) dataset contains pictures of digits in all shapes and colors, taken by the Google Street View cars. The goal is to classify and locate all the digits in large images.

- Train a Fully Convolutional Network on the 32x32 images.
- Use this FCN to build a digit detector in the large images.