< Deep Learning - PART2 TF2 CNNs >

Ch 5. CNNs Workshop 5 - Transfer learning with TF Hub ¶

2021/10/01

[Reference] :

> 1. TensorFlow Core - Tutorials: **Transfer learning with TensorFlow Hub** https://www.tensorflow.org/tutorials/images/transfer_learning_with_hub?hl=zh_tw (https://www.tensorflow.org/tutorials/images/transfer_learning_with_hub?hl=zh_tw)
> 2. Keras Documentation - **Keras Applications** https://keras.io/applications/ (https://keras.io/applications/)
> 3. Andrew Ng - Coursera : **Transfer learning** https://zh-tw.coursera.org/lecture/machine-learning-projects/transfer-learning-WNPap (https://zh-tw.coursera.org/lecture/machine-learning-projects/transfer-learning-WNPap)

TensorFlow Hub (https://tfhub.dev/) is a repository of pre-trained TensorFlow models.

This tutorial demonstrates how to:

1. Use models from TensorFlow Hub with `tf.keras`
2. Use an image classification model from TensorFlow Hub
3. Do simple transfer learning to fine-tune a model for your own image classes

Setup

> On **`Anaconda Prompt`** , run :
>
> ```
> pip install tensorflow_hub
> ```

In [ ]:

```
import numpy as np
import time

import PIL.Image as Image
import matplotlib.pylab as plt

import tensorflow as tf
import tensorflow_hub as hub

import datetime

%load_ext tensorboard
```

An ImageNet classifier

You'll start by using a classifier model pre-trained on the [ImageNet (https://en.wikipedia.org/wiki/ImageNet)](https://en.wikipedia.org/wiki/ImageNet) benchmark dataset—no initial training required!

Download the classifier

Select a [MobileNetV2 (https://arxiv.org/abs/1801.04381)](https://arxiv.org/abs/1801.04381) pre-trained model [from TensorFlow Hub (https://tfhub.dev/google/tf2-preview/mobilenet_v2/classification/2)](https://tfhub.dev/google/tf2-preview/mobilenet_v2/classification/2) and wrap it as a Keras layer with [`hub.KerasLayer` (https://www.tensorflow.org/hub/api_docs/python/hub/KerasLayer)](https://www.tensorflow.org/hub/api_docs/python/hub/KerasLayer). Any [compatible image classifier model (https://tfhub.dev/s?q=tf2&module-type=image-classification/)](https://tfhub.dev/s?q=tf2&module-type=image-classification/) from TensorFlow Hub will work here, including the examples provided in the drop-down below.

In [4]:

```
mobilenet_v2 ="https://tfhub.dev/google/tf2-preview/mobilenet_v2/classifica
inception_v3 = "https://tfhub.dev/google/imagenet/inception_v3/classificati

classifier_model = mobilenet_v2 #@param ["mobilenet_v2", "inception_v3"] {t
```

In [5]:

```
IMAGE_SHAPE = (224, 224)

classifier = tf.keras.Sequential([
    hub.KerasLayer(classifier_model, input_shape=IMAGE_SHAPE+(3,))
])
```

```
WARNING:tensorflow:AutoGraph could not transform <bound method Ke
rasLayer.call of <tensorflow_hub.keras_layer.KerasLayer object at
0x000002247099E760>> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, s
et the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10
`) and attach the full output.
Cause: module 'gast' has no attribute 'Index'
To silence this warning, decorate the function with @tf.autograp
h.experimental.do_not_convert


WARNING:tensorflow:AutoGraph could not transform <bound method Ke
rasLayer.call of <tensorflow_hub.keras_layer.KerasLayer object at
0x000002247099E760>> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, s
et the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10
`) and attach the full output.
Cause: module 'gast' has no attribute 'Index'
To silence this warning, decorate the function with @tf.autograp
h.experimental.do_not_convert


WARNING: AutoGraph could not transform <bound method KerasLayer.c
all of <tensorflow_hub.keras_layer.KerasLayer object at 0x0000022
47099E760>> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, s
et the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10
`) and attach the full output.
Cause: module 'gast' has no attribute 'Index'
To silence this warning, decorate the function with @tf.autograp
h.experimental.do_not_convert
```

Run it on a single image

Download a single image to try the model on:

In [6]:

```
grace_hopper = tf.keras.utils.get_file('image.jpg','https://storage.googlea
grace_hopper = Image.open(grace_hopper).resize(IMAGE_SHAPE)
grace_hopper
```

Out[6]:

In [7]:

```
grace_hopper = np.array(grace_hopper)/255.0
grace_hopper.shape
```

Out[7]:

```
(224, 224, 3)
```

Add a batch dimension (with `np.newaxis`) and pass the image to the model:

In [8]:

```
result = classifier.predict(grace_hopper[np.newaxis, ...])
result.shape
```

Out[8]:

```
(1, 1001)
```

The result is a 1001-element vector of logits, rating the probability of each class for the image.

The top class ID can be found with `tf.math.argmax` :

In [9]:

```
predicted_class = tf.math.argmax(result[0], axis=-1)
predicted_class
```

Out[9]:

```
<tf.Tensor: shape=(), dtype=int64, numpy=653>
```

Decode the predictions

Take the `predicted_class` ID (such as `653`) and fetch the ImageNet dataset labels to decode the predictions:

In [10]:

```
labels_path = tf.keras.utils.get_file('ImageNetLabels.txt','https://storage
imagenet_labels = np.array(open(labels_path).read().splitlines())
```

In [11]:

```
plt.imshow(grace_hopper)
plt.axis('off')
predicted_class_name = imagenet_labels[predicted_class]
_ = plt.title("Prediction: " + predicted_class_name.title())
```

Prediction: Military Uniform

Simple transfer learning

But what if you want to create a custom classifier using your own dataset that has classes that aren't included in the original ImageNet dataset (that the pre-trained model was trained on)?

To do that, you can:

1. Select a pre-trained model from TensorFlow Hub; and
2. Retrain the top (last) layer to recognize the classes from your custom dataset.

Dataset

In this example, you will use the TensorFlow flowers dataset:

In [12]:

```
data_root = tf.keras.utils.get_file(
  'flower_photos',
  'https://storage.googleapis.com/download.tensorflow.org/example_images/fl
   untar=True)
```

First, load this data into the model using the image data off disk with `tf.keras.preprocessing.image_dataset_from_directory`, which will generate a `tf.data.Dataset`:

In [13]:

```
batch_size = 32
img_height = 224
img_width = 224

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  str(data_root),
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size
)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
  str(data_root),
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size
)
```

```
Found 3670 files belonging to 5 classes.
Using 2936 files for training.
Found 3670 files belonging to 5 classes.
Using 734 files for validation.
```

The flowers dataset has five classes:

In [14]:

```
class_names = np.array(train_ds.class_names)
print(class_names)
```

```
['daisy' 'dandelion' 'roses' 'sunflowers' 'tulips']
```

Second, because TensorFlow Hub's convention for image models is to expect float inputs in the `[0, 1]` range, use the `tf.keras.layers.experimental.preprocessing.Rescaling` layer to achieve this.

Note: You could also include the `tf.keras.layers.experimental.preprocessing.Rescaling` layer inside the model. Refer to the [Working with preprocessing layers (https://www.tensorflow.org/guide/keras/preprocessing_layers)](https://www.tensorflow.org/guide/keras/preprocessing_layers) guide for a discussion of the tradeoffs.

In [15]:

```
normalization_layer = tf.keras.layers.experimental.preprocessing.Rescaling(
train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y)) # Where x
val_ds = val_ds.map(lambda x, y: (normalization_layer(x), y)) # Where x—ima
```

Third, finish the input pipeline by using buffered prefetching with `Dataset.prefetch`, so you can yield the data from disk without I/O blocking issues.

These are some of the most important `tf.data` methods you should use when loading data. Interested readers can learn more about them, as well as how to cache data to disk and other techniques, in the [Better performance with the tf.data API (https://www.tensorflow.org/guide/data_performance#prefetching)](https://www.tensorflow.org/guide/data_performance#prefetching) guide.

In [16]:

```
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

In [17]:

```
for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

```
(32, 224, 224, 3)
(32,)
```

Run the classifier on a batch of images

Now, run the classifier on an image batch:

In [18]:

```
result_batch = classifier.predict(train_ds)
```

In [19]:

```
predicted_class_names = imagenet_labels[tf.math.argmax(result_batch, axis=-
predicted_class_names
```

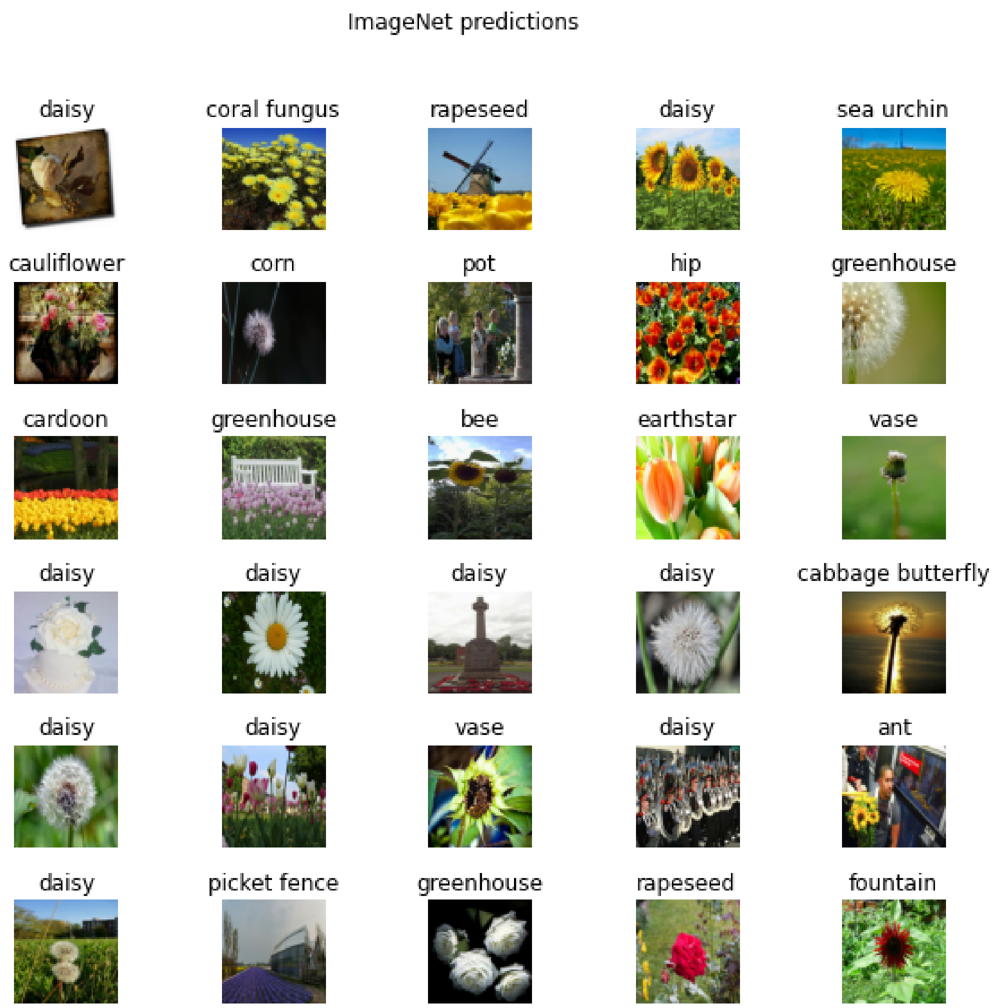
Out[19]:

```
array(['daisy', 'coral fungus', 'rapeseed', ..., 'daisy', 'dais
y',
       'birdhouse'], dtype='<U30')
```

Check how these predictions line up with the images:

In [20]:

```
plt.figure(figsize=(10,9))
plt.subplots_adjust(hspace=0.5)
for n in range(30):
  plt.subplot(6,5,n+1)
  plt.imshow(image_batch[n])
  plt.title(predicted_class_names[n])
  plt.axis('off')
_ = plt.suptitle("ImageNet predictions")
```



Note: all images are licensed CC-BY, creators are listed in the LICENSE.txt file.

The results are far from perfect, but reasonable considering that these are not the classes the model was trained for (except for "daisy").

Download the headless model

TensorFlow Hub also distributes models without the top classification layer. These can be used to easily perform transfer learning.

Select a [MobileNetV2 (https://arxiv.org/abs/1801.04381)](https://arxiv.org/abs/1801.04381) pre-trained model [from TensorFlow Hub (https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4)](https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4). Any [compatible image feature vector model (https://tfhub.dev/s?module-type=image-feature-vector&q=tf2)](https://tfhub.dev/s?module-type=image-feature-vector&q=tf2) from TensorFlow Hub will work here, including the examples from the drop-down menu.

In [21]:

```
mobilenet_v2 = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_v
inception_v3 = "https://tfhub.dev/google/tf2-preview/inception_v3/feature_v

feature_extractor_model = mobilenet_v2 #@param ["mobilenet_v2", "inception_
```

Create the feature extractor by wrapping the pre-trained model as a Keras layer with `hub.KerasLayer` [(https://www.tensorflow.org/hub/api_docs/python/hub/KerasLayer)](https://www.tensorflow.org/hub/api_docs/python/hub/KerasLayer). Use the `trainable=False` argument to freeze the variables, so that the training only modifies the new classifier layer:

In [22]:

```
feature_extractor_layer = hub.KerasLayer(
    feature_extractor_model,
    input_shape=(224, 224, 3),
    trainable=False)
```

The feature extractor returns a 1280-long vector for each image (the image batch size remains at 32 in this example):

In [23]:

```
feature_batch = feature_extractor_layer(image_batch)
print(feature_batch.shape)
```

```
(32, 1280)
```

Attach a classification head

To complete the model, wrap the feature extractor layer in a `tf.keras.Sequential` model and add a fully-connected layer for classification:

In [24]:

```
num_classes = len(class_names)

model = tf.keras.Sequential([
  feature_extractor_layer,
  tf.keras.layers.Dense(num_classes)
])

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer_1 (KerasLayer)   (None, 1280)              2257984
_________________________________________________________________
dense (Dense)                (None, 5)                 6405
=================================================================
Total params: 2,264,389
Trainable params: 6,405
Non-trainable params: 2,257,984
_________________________________________________________________
```

In [25]:

```
predictions = model(image_batch)
```

In [26]:

```
predictions.shape
```

Out[26]:

```
TensorShape([32, 5])
```

Train the model

Use `Model.compile` to configure the training process and add a `tf.keras.callbacks.TensorBoard` callback to create and store logs:

In [27]:

```python
model.compile(
  optimizer=tf.keras.optimizers.Adam(),
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['acc'])

log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir=log_dir,
    histogram_freq=1) # Enable histogram computation for every epoch.
```

Now use the `Model.fit` method to train the model.

To keep this example short, you'll be training for just 10 epochs. To visualize the training progress in TensorBoard later, create and store logs an a [TensorBoard callback (https://www.tensorflow.org/tensorboard/get_started#using_tensorboard_with_keras_modelfit)](https://www.tensorflow.org/tensorboard/get_started#using_tensorboard_with_keras_modelfit).

In [28]:

```
NUM_EPOCHS = 10

history = model.fit(train_ds,
                    validation_data=val_ds,
                    epochs=NUM_EPOCHS,
                    callbacks=tensorboard_callback)
```

```
Epoch 1/10
92/92 [==============================] - 67s 695ms/step - loss:
1.1070 - acc: 0.5653 - val_loss: 0.4672 - val_acc: 0.8474
Epoch 2/10
92/92 [==============================] - 71s 778ms/step - loss:
0.3976 - acc: 0.8619 - val_loss: 0.3764 - val_acc: 0.8706
Epoch 3/10
92/92 [==============================] - 82s 896ms/step - loss:
0.3058 - acc: 0.9010 - val_loss: 0.3406 - val_acc: 0.8842
Epoch 4/10
92/92 [==============================] - 96s 1s/step - loss: 0.25
21 - acc: 0.9248 - val_loss: 0.3221 - val_acc: 0.8924
Epoch 5/10
92/92 [==============================] - 104s 1s/step - loss: 0.2
146 - acc: 0.9380 - val_loss: 0.3108 - val_acc: 0.8951
Epoch 6/10
92/92 [==============================] - 111s 1s/step - loss: 0.1
863 - acc: 0.9463 - val_loss: 0.3032 - val_acc: 0.8978
Epoch 7/10
92/92 [==============================] - 105s 1s/step - loss: 0.1
641 - acc: 0.9551 - val_loss: 0.2979 - val_acc: 0.9005
Epoch 8/10
92/92 [==============================] - 103s 1s/step - loss: 0.1
460 - acc: 0.9643 - val_loss: 0.2939 - val_acc: 0.9005
Epoch 9/10
92/92 [==============================] - 101s 1s/step - loss: 0.1
308 - acc: 0.9717 - val_loss: 0.2909 - val_acc: 0.9019
Epoch 10/10
92/92 [==============================] - 105s 1s/step - loss: 0.1
179 - acc: 0.9739 - val_loss: 0.2885 - val_acc: 0.9046
```

Start the TensorBoard to view how the metrics change with each epoch and to track other scalar values:

In [ ]:

```
# %tensorboard --logdir logs/fit
```

To run TensorBoard, run the following command on `Anaconda (Powershell) Prompt` :

`tensorboard --logdir=` *path/to/log-directory*

- For instance, **`tensorboard --logdir logs/fit`**

Connecting to **`http://localhost:6006`**

Check the predictions

Obtain the ordered list of class names from the model predictions:

In [30]:

```
predicted_batch = model.predict(image_batch)
predicted_id = tf.math.argmax(predicted_batch, axis=-1)
predicted_label_batch = class_names[predicted_id]
print(predicted_label_batch)
```

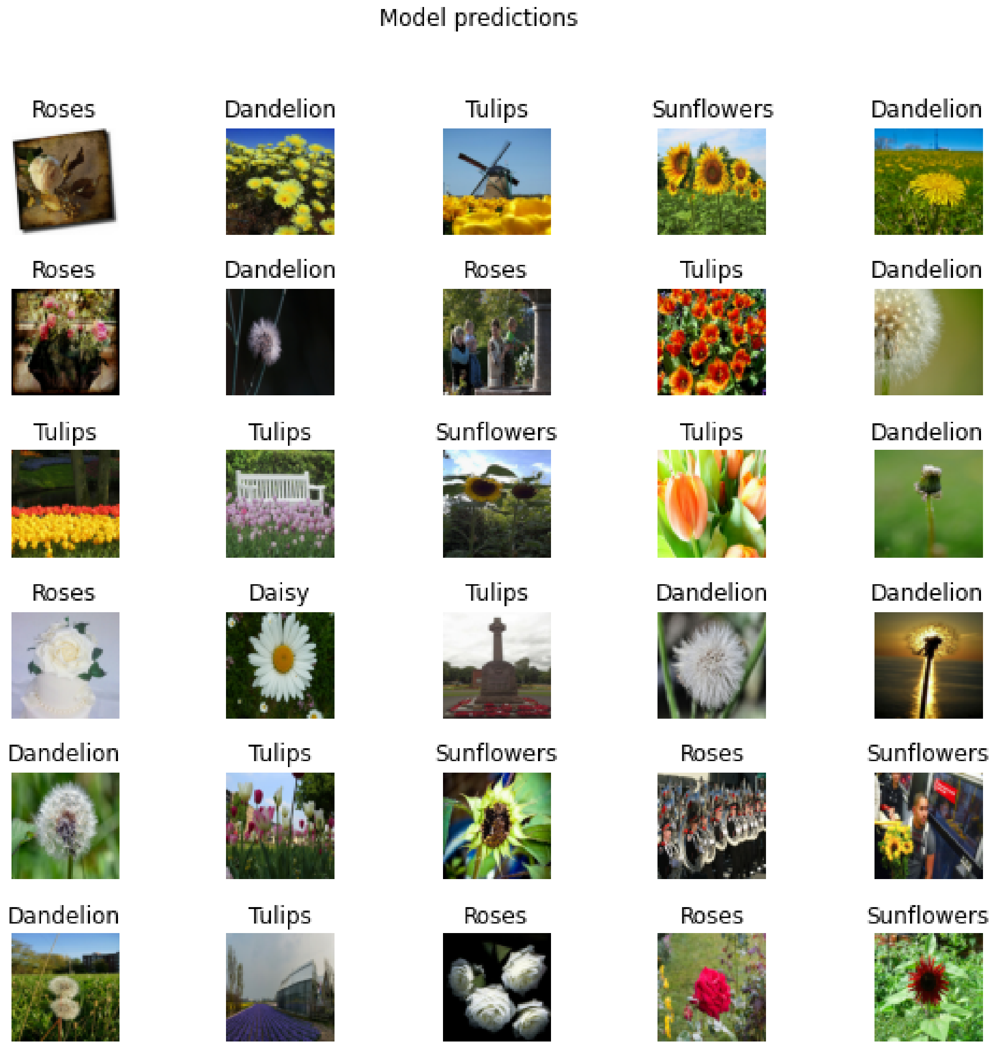
```
['roses' 'dandelion' 'tulips' 'sunflowers' 'dandelion' 'roses' 'd
andelion'
 'roses' 'tulips' 'dandelion' 'tulips' 'tulips' 'sunflowers' 'tul
ips'
 'dandelion' 'roses' 'daisy' 'tulips' 'dandelion' 'dandelion' 'da
ndelion'
 'tulips' 'sunflowers' 'roses' 'sunflowers' 'dandelion' 'tulips'
'roses'
 'roses' 'sunflowers' 'tulips' 'sunflowers']
```

Plot the model predictions:

In [31]:

```
plt.figure(figsize=(10,9))
plt.subplots_adjust(hspace=0.5)

for n in range(30):
  plt.subplot(6,5,n+1)
  plt.imshow(image_batch[n])
  plt.title(predicted_label_batch[n].title())
  plt.axis('off')
_ = plt.suptitle("Model predictions")
```



Export and reload your model

Now that you've trained the model, export it as a SavedModel for reusing it later.

In [32]:

```
t = time.time()

export_path = "/tmp/saved_models/{}".format(int(t))
model.save(export_path)

export_path
```

```
INFO:tensorflow:Assets written to: /tmp/saved_models/1633297160\a
ssets
```


```
INFO:tensorflow:Assets written to: /tmp/saved_models/1633297160\a
ssets
```


Out[32]:

```
'/tmp/saved_models/1633297160'
```

Confirm that you can reload the SavedModel and that the model is able to output the same results:

In [33]:

```
reloaded = tf.keras.models.load_model(export_path)
```

In [34]:

```
result_batch = model.predict(image_batch)
reloaded_result_batch = reloaded.predict(image_batch)
```

In [35]:

```
abs(reloaded_result_batch - result_batch).max()
```

Out[35]:

```
0.0
```

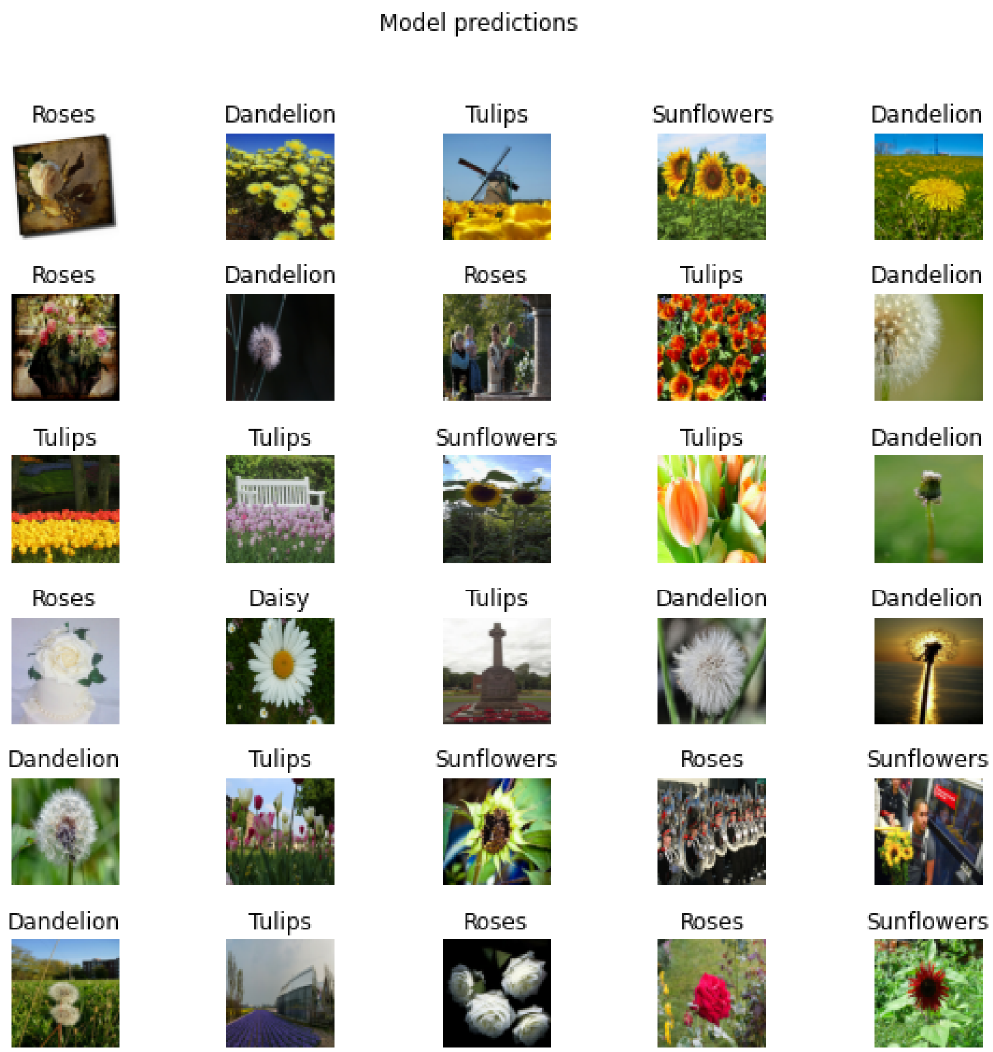
In [36]:

```
reloaded_predicted_id = tf.math.argmax(reloaded_result_batch, axis=-1)
reloaded_predicted_label_batch = class_names[reloaded_predicted_id]
print(reloaded_predicted_label_batch)
```

```
['roses' 'dandelion' 'tulips' 'sunflowers' 'dandelion' 'roses' 'd
andelion'
 'roses' 'tulips' 'dandelion' 'tulips' 'tulips' 'sunflowers' 'tul
ips'
 'dandelion' 'roses' 'daisy' 'tulips' 'dandelion' 'dandelion' 'da
ndelion'
 'tulips' 'sunflowers' 'roses' 'sunflowers' 'dandelion' 'tulips'
'roses'
 'roses' 'sunflowers' 'tulips' 'sunflowers']
```

In [37]:

```
plt.figure(figsize=(10,9))
plt.subplots_adjust(hspace=0.5)
for n in range(30):
  plt.subplot(6,5,n+1)
  plt.imshow(image_batch[n])
  plt.title(reloaded_predicted_label_batch[n].title())
  plt.axis('off')
_ = plt.suptitle("Model predictions")
```



Next steps

You can use the SavedModel to load for inference or convert it to a TensorFlow Lite (https://www.tensorflow.org/lite/convert/) model (for on-device machine learning) or a TensorFlow.js (https://www.tensorflow.org/js/tutorials#convert_pretrained_models_to_tensorflowjs) model (for machine learning in JavaScript).

Discover more tutorials (https://www.tensorflow.org/hub/tutorials) to learn how to use pre-trained models from TensorFlow Hub on image, text, audio, and video tasks.