Applications

Keras Applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning.

Weights are downloaded automatically when instantiating a model. They are stored at `~/.keras/models/`.

Available models

Models for image classification with weights trained on ImageNet:

- Xception
- VGG16
- VGG19
- ResNet50
- InceptionV3
- InceptionResNetV2
- MobileNet

All of these architectures (except Xception and MobileNet) are compatible with both TensorFlow and Theano, and upon instantiation the models will be built according to the image data format set in your Keras configuration file at `~/.keras/keras.json`. For instance, if you have set `image_data_format=channels_last`, then any model loaded from this repository will get built according to the TensorFlow data format convention, "Height-Width-Depth".

The Xception model is only available for TensorFlow, due to its reliance on `SeparableConvolution` layers. The MobileNet model is only available for TensorFlow, due to its reliance on `DepthwiseConvolution` layers.

Usage examples for image classification models

Classify ImageNet classes with ResNet50

```
from keras.applications.resnet50 import ResNet50
from keras.preprocessing import image
from keras.applications.resnet50 import preprocess_input, decode_predictions
import numpy as np

model = ResNet50(weights='imagenet')

img_path = 'elephant.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

preds = model.predict(x)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
print('Predicted:', decode_predictions(preds, top=3)[0])
# Predicted: [(u'n02504013', u'Indian_elephant', 0.82658225), (u'n01871265', u'tuske
```

Extract features with VGG16

```
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
import numpy as np

model = VGG16(weights='imagenet', include_top=False)

img_path = 'elephant.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

features = model.predict(x)
```

Extract features from an arbitrary intermediate layer with VGG19

```
from keras.applications.vgg19 import VGG19
from keras.preprocessing import image
from keras.applications.vgg19 import preprocess_input
from keras.models import Model
import numpy as np

base_model = VGG19(weights='imagenet')
model = Model(inputs=base_model.input, outputs=base_model.get_layer('block4_pool').o

img_path = 'elephant.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

block4_pool_features = model.predict(x)
```

Fine-tune InceptionV3 on a new set of classes

```
from keras.applications.inception_v3 import InceptionV3
from keras.preprocessing import image
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras import backend as K

# create the base pre-trained model
base_model = InceptionV3(weights='imagenet', include_top=False)

# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
# let's add a fully-connected layer
x = Dense(1024, activation='relu')(x)
# and a logistic layer -- let's say we have 200 classes
predictions = Dense(200, activation='softmax')(x)

# this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)

# first: train only the top layers (which were randomly initialized)
# i.e. freeze all convolutional InceptionV3 layers
for layer in base_model.layers:
    layer.trainable = False

# compile the model (should be done *after* setting layers to non-trainable)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')

# train the model on the new data for a few epochs
model.fit_generator(...)

# at this point, the top layers are well trained and we can start fine-tuning
# convolutional layers from inception V3. We will freeze the bottom N layers
# and train the remaining top layers.

# let's visualize layer names and layer indices to see how many layers
# we should freeze:
for i, layer in enumerate(base_model.layers):
   print(i, layer.name)

# we chose to train the top 2 inception blocks, i.e. we will freeze
# the first 249 layers and unfreeze the rest:
for layer in model.layers[:249]:
   layer.trainable = False
for layer in model.layers[249:]:
   layer.trainable = True

# we need to recompile the model for these modifications to take effect
# we use SGD with a low learning rate
from keras.optimizers import SGD
model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy

# we train our model again (this time fine-tuning the top 2 inception blocks
# alongside the top Dense layers
model.fit_generator(...)
```

Build InceptionV3 over a custom input tensor

```
from keras.applications.inception_v3 import InceptionV3
from keras.layers import Input

# this could also be the output a different Keras model or layer
input_tensor = Input(shape=(224, 224, 3))  # this assumes K.image_data_format() == '

model = InceptionV3(input_tensor=input_tensor, weights='imagenet', include_top=True)
```

Documentation for individual models

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.715	0.901	138,357,544	23
VGG19	549 MB	0.727	0.910	143,667,240	26
ResNet50	99 MB	0.759	0.929	25,636,712	168
InceptionV3	92 MB	0.788	0.944	23,851,784	159
InceptionResNetV2	215 MB	0.804	0.953	55,873,736	572
MobileNet	17 MB	0.665	0.871	4,253,864	88

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

Xception

```
keras.applications.xception.Xception(include_top=True, weights='imagenet', input_ten
```

Xception V1 model, with weights pre-trained on ImageNet.

On ImageNet, this model gets to a top-1 validation accuracy of 0.790 and a top-5 validation accuracy of 0.945.

Note that this model is only available for the TensorFlow backend, due to its reliance on `SeparableConvolution` layers. Additionally it only supports the data format `'channels_last'` (height, width, channels).

The default input size for this model is 299x299.

Arguments

- include_top: whether to include the fully-connected layer at the top of the network.
- weights: one of `None` (random initialization) or `'imagenet'` (pre-training on ImageNet).
- input_tensor: optional Keras tensor (i.e. output of `layers.Input()`) to use as image input for the model.
- input_shape: optional shape tuple, only to be specified if `include_top` is `False` (otherwise the input shape has to be `(299, 299, 3)`. It should have exactly 3 inputs channels, and width and height should be no smaller than 71. E.g. `(150, 150, 3)` would be one valid value.
- pooling: Optional pooling mode for feature extraction when `include_top` is `False`.

    - `None` means that the output of the model will be the 4D tensor output of the last convolutional layer.
    - `'avg'` means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor.
    - `'max'` means that global max pooling will be applied.
- classes: optional number of classes to classify images into, only to be specified if `include_top` is `True`, and if no `weights` argument is specified.

Returns

A Keras `Model` instance.

References

- Xception: Deep Learning with Depthwise Separable Convolutions

License

These weights are trained by ourselves and are released under the MIT license.

VGG16

```
keras.applications.vgg16.VGG16(include_top=True, weights='imagenet', input_tensor=No
```

VGG16 model, with weights pre-trained on ImageNet.

This model is available for both the Theano and TensorFlow backend, and can be built both with `'channels_first'` data format (channels, height, width) or `'channels_last'` data format (height, width, channels).

The default input size for this model is 224x224.

Arguments

- include_top: whether to include the 3 fully-connected layers at the top of the network.
- weights: one of `None` (random initialization) or `'imagenet'` (pre-training on ImageNet).
- input_tensor: optional Keras tensor (i.e. output of `layers.Input()`) to use as image input for the model.
- input_shape: optional shape tuple, only to be specified if `include_top` is `False` (otherwise the input shape has to be `(224, 224, 3)` (with `'channels_last'` data format) or `(3, 224, 224)` (with `'channels_first'` data format). It should have exactly 3 inputs channels, and width and height should be no smaller than 48. E.g. `(200, 200, 3)` would be one valid value.
- pooling: Optional pooling mode for feature extraction when `include_top` is `False`.

    - `None` means that the output of the model will be the 4D tensor output of the last convolutional layer.
    - `'avg'` means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor.
    - `'max'` means that global max pooling will be applied.
- classes: optional number of classes to classify images into, only to be specified if `include_top` is `True`, and if no `weights` argument is specified.

Returns

A Keras `Model` instance.

References

- Very Deep Convolutional Networks for Large-Scale Image Recognition: please cite this paper if you use the VGG models in your work.

License

These weights are ported from the ones released by VGG at Oxford under the Creative Commons Attribution License.

VGG19

```
keras.applications.vgg19.VGG19(include_top=True, weights='imagenet', input_tensor=No
```

VGG19 model, with weights pre-trained on ImageNet.

This model is available for both the Theano and TensorFlow backend, and can be built both with `'channels_first'` data format (channels, height, width) or `'channels_last'` data format (height, width, channels).

The default input size for this model is 224x224.

Arguments

- include_top: whether to include the 3 fully-connected layers at the top of the network.
- weights: one of `None` (random initialization) or `'imagenet'` (pre-training on ImageNet).
- input_tensor: optional Keras tensor (i.e. output of `layers.Input()`) to use as image input for the model.
- input_shape: optional shape tuple, only to be specified if `include_top` is `False` (otherwise the input shape has to be `(224, 224, 3)` (with `'channels_last'` data format) or `(3, 224, 224)` (with `'channels_first'` data format). It should have exactly 3 inputs channels, and width and height should be no smaller than 48. E.g. `(200, 200, 3)` would be one valid value.

- pooling: Optional pooling mode for feature extraction when `include_top` is `False`.
  - `None` means that the output of the model will be the 4D tensor output of the last convolutional layer.
  - `'avg'` means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor.
  - `'max'` means that global max pooling will be applied.
- classes: optional number of classes to classify images into, only to be specified if `include_top` is `True`, and if no `weights` argument is specified.

Returns

A Keras `Model` instance.

References

- Very Deep Convolutional Networks for Large-Scale Image Recognition

License

These weights are ported from the ones released by VGG at Oxford under the Creative Commons Attribution License.

ResNet50

```
keras.applications.resnet50.ResNet50(include_top=True, weights='imagenet', input_ten
```

ResNet50 model, with weights pre-trained on ImageNet.

This model is available for both the Theano and TensorFlow backend, and can be built both with `'channels_first'` data format (channels, height, width) or `'channels_last'` data format (height, width, channels).

The default input size for this model is 224x224.

Arguments

- include_top: whether to include the fully-connected layer at the top of the network.
- weights: one of `None` (random initialization) or `'imagenet'` (pre-training on ImageNet).
- input_tensor: optional Keras tensor (i.e. output of `layers.Input()`) to use as image input for the model.
- input_shape: optional shape tuple, only to be specified if `include_top` is `False` (otherwise the input shape has to be `(224, 224, 3)` (with `'channels_last'` data format) or `(3, 224, 224)` (with `'channels_first'` data format). It should have exactly 3 inputs channels, and width and height should be no smaller than 197. E.g. `(200, 200, 3)` would be one valid value.

- pooling: Optional pooling mode for feature extraction when `include_top` is `False`.
  - `None` means that the output of the model will be the 4D tensor output of the last convolutional layer.
  - `'avg'` means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor.
  - `'max'` means that global max pooling will be applied.
- classes: optional number of classes to classify images into, only to be specified if `include_top` is `True`, and if no `weights` argument is specified.

Returns

A Keras `Model` instance.

References

- Deep Residual Learning for Image Recognition

License

These weights are ported from the ones released by Kaiming He under the MIT license.

InceptionV3

```
keras.applications.inception_v3.InceptionV3(include_top=True, weights='imagenet', in
```

Inception V3 model, with weights pre-trained on ImageNet.

This model is available for both the Theano and TensorFlow backend, and can be built both with `'channels_first'` data format (channels, height, width) or `'channels_last'` data format (height, width, channels).

The default input size for this model is 299x299.

Arguments

- include_top: whether to include the fully-connected layer at the top of the network.
- weights: one of `None` (random initialization) or `'imagenet'` (pre-training on ImageNet).
- input_tensor: optional Keras tensor (i.e. output of `layers.Input()`) to use as image input for the model.
- input_shape: optional shape tuple, only to be specified if `include_top` is `False` (otherwise the input shape has to be `(299, 299, 3)` (with `'channels_last'` data format) or `(3, 299, 299)` (with `'channels_first'` data format). It should have exactly 3 inputs channels, and width and height should be no smaller than 139. E.g. `(150, 150, 3)` would be one valid value.
- pooling: Optional pooling mode for feature extraction when `include_top` is `False`.

    - `None` means that the output of the model will be the 4D tensor output of the last convolutional layer.
    - `'avg'` means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor.
    - `'max'` means that global max pooling will be applied.
- classes: optional number of classes to classify images into, only to be specified if `include_top` is `True`, and if no `weights` argument is specified.

Returns

A Keras `Model` instance.

References

- Rethinking the Inception Architecture for Computer Vision

License

These weights are released under the Apache License.

InceptionResNetV2

```
keras.applications.inception_resnet_v2.InceptionResNetV2(include_top=True, weights='
```

Inception-ResNet V2 model, with weights pre-trained on ImageNet.

This model is available for both the Theano and TensorFlow backend (but not CNTK), and can be built both with `'channels_first'` data format (channels, height, width) or `'channels_last'` data format (height, width, channels).

The default input size for this model is 299x299.

Arguments

- include_top: whether to include the fully-connected layer at the top of the network.
- weights: one of `None` (random initialization) or `'imagenet'` (pre-training on ImageNet).
- input_tensor: optional Keras tensor (i.e. output of `layers.Input()`) to use as image input for the model.
- input_shape: optional shape tuple, only to be specified if `include_top` is `False` (otherwise the input shape has to be `(299, 299, 3)` (with `'channels_last'` data format) or `(3, 299, 299)` (with `'channels_first'` data format). It should have exactly 3 inputs channels, and width and height should be no smaller than 139. E.g. `(150, 150, 3)` would be one valid value.
- pooling: Optional pooling mode for feature extraction when `include_top` is `False`.

    - `None` means that the output of the model will be the 4D tensor output of the last convolutional layer.
    - `'avg'` means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor.
    - `'max'` means that global max pooling will be applied.
- classes: optional number of classes to classify images into, only to be specified if `include_top` is `True`, and if no `weights` argument is specified.

Returns

A Keras `Model` instance.

References

- Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning

License

These weights are released under the Apache License.

MobileNet

```
keras.applications.mobilenet.MobileNet(input_shape=None, alpha=1.0, depth_multiplier
```

MobileNet model, with weights pre-trained on ImageNet.

Note that only TensorFlow is supported for now, therefore it only works with the data format `image_data_format='channels_last'` in your Keras config at `~/.keras/keras.json`. To load a MobileNet model via `load_model`, import the custom objects `relu6` and `DepthwiseConv2D` and pass them to the `custom_objects` parameter.

E.g.

```
model = load_model('mobilenet.h5', custom_objects={
                   'relu6': mobilenet.relu6,
                   'DepthwiseConv2D': mobilenet.DepthwiseConv2D})
```

The default input size for this model is 224x224.

Arguments

- input_shape: optional shape tuple, only to be specified if `include_top` is `False` (otherwise the input shape has to be `(224, 224, 3)` (with `'channels_last'` data format) or (3, 224, 224) (with `'channels_first'` data format). It should have exactly 3 inputs channels, and width and height should be no smaller than 32. E.g. `(200, 200, 3)` would be one valid value.

- alpha: controls the width of the network.
  - If `alpha` < 1.0, proportionally decreases the number of filters in each layer.
  - If `alpha` > 1.0, proportionally increases the number of filters in each layer.
  - If `alpha` = 1, default number of filters from the paper are used at each layer.
- depth_multiplier: depth multiplier for depthwise convolution (also called the resolution multiplier)
- dropout: dropout rate
- include_top: whether to include the fully-connected layer at the top of the network.
- weights: `None` (random initialization) or `'imagenet'` (ImageNet weights)
- input_tensor: optional Keras tensor (i.e. output of `layers.Input()`) to use as image input for the model.
- pooling: Optional pooling mode for feature extraction when `include_top` is `False`.
  - `None` means that the output of the model will be the 4D tensor output of the last convolutional layer.
  - `'avg'` means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor.
  - `'max'` means that global max pooling will be applied.
- classes: optional number of classes to classify images into, only to be specified if `include_top` is `True`, and if no `weights` argument is specified.

Returns

A Keras `Model` instance.

References

- MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

License