# Transfer Learning in Keras with Computer Vision Models

by Jason Brownlee on May 15, 2019 in Deep Learning for Computer Vision

Last Updated on September 3, 2019

Deep convolutional neural network models may take days or even weeks to train on very large datasets.

A way to short-cut this process is to re-use the model weights from pre-trained models that were developed for standard computer vision benchmark datasets, such as the ImageNet image recognition tasks. Top performing models can be downloaded and used directly, or integrated into a new model for your own computer vision problems.

In this post, you will discover how to use transfer learning when developing convolutional neural networks for computer vision applications.

After reading this post, you will know:

- Transfer learning involves using models trained on one problem as a starting point on a related problem.
- Transfer learning is flexible, allowing the use of pre-trained models directly, as feature extraction preprocessing, and integrated into entirely new models.
- Keras provides convenient access to many top performing models on the ImageNet image recognition tasks such as VGG, Inception, and ResNet.

Discover how to build models for photo classification, object detection, face recognition, and more in my new computer vision book, with 30 step-by-ste

Let's get started.

How to Use Transfer Learning when Developing Convolutional Neural Network Models
Photo by GoToVan, some rights reserved.

## Overview

This tutorial is divided into five parts; they are:

1. What Is Transfer Learning?
2. Transfer Learning for Image Recognition
3. How to Use Pre-Trained Models
4. Models for Transfer Learning
5. Examples of Using Pre-Trained Models

## What Is Transfer Learning?

Transfer learning generally refers to a process where a model trained on one problem is used in some way on a second related problem.

In deep learning, transfer learning is a technique whereby a neural network model is first trained on a problem similar to the problem that is being solved. One or more layers from the trained model are then used in a new model trained on the problem of interest.

> *This is typically understood in a supervised*
> *the target may be of a different nature. For e*
> *categories, such as cats and dogs, in the fi*
> *categories, such as ants and wasps, in the*

— Page 536, Deep Learning, 2016.

Transfer learning has the benefit of decreasing the
result in lower generalization error.

The weights in re-used layers may be used as the
response to the new problem. This usage treats tra
scheme. This may be useful when the first related problem has a lot more labeled data than the problem of interest and the similarity in the structure of the problem may be useful in both contexts.

> *… the objective is to take advantage of data from the first setting to extract information that may be useful when learning or even when directly making predictions in the second setting.*

— Page 538, Deep Learning, 2016.

## Transfer Learning for Image Recognition

A range of high-performing models have been developed for image classification and demonstrated on the annual ImageNet Large Scale Visual Recognition Challenge, or ILSVRC.

This challenge, often referred to simply as ImageNet, given the source of the image used in the competition, has resulted in a number of innovations in the architecture and training of convolutional neural networks. In addition, many of the models used in the competitions have been released under a permissive license.

These models can be used as the basis for transfer learning in computer vision applications.

This is desirable for a number of reasons, not least:

- **Useful Learned Features**: The models have learned how to detect generic features from photographs, given that they were trained on more than 1,000,000 images for 1,000 categories.
- **State-of-the-Art Performance**: The models achieved state of the art performance and remain effective on the specific image recognition task for which they were developed.
- **Easily Accessible**: The model weights are provided as free downloadable files and many libraries provide convenient APIs to download and use the models directly.

The model weights can be downloaded and used in the same model architecture using a range of different deep learning libraries, including Keras.

## How to Use Pre-Trained Models

The use of a pre-trained model is limited only by your creativity.

For example, a model may be downloaded and used as-is, such as embedded into an application and used to classify new photographs.

Alternately, models may be downloaded and use as feature extraction models. Here, the output of the model from a layer prior to the output layer of the model is used as input to a new classifier model.

Recall that convolutional layers closer to the input layer of the model learn low-level features such as lines, that layers in the middle of the layer learn complex abstract features that combine the lower level features extracted from the input, and layers closer to the output interpret the extracted features in the context of a classification task.

Armed with this understanding, a level of detail for feature extraction from an existing pre-trained model can be chosen. For example, if a new task is quite different from classifying objects in photographs (e.g. different to ImageNet), then perhaps the output of the pre-trained model after the few layers would be appropriate. If a new task is quite similar to the task of classifying objects in photographs, then perhaps the output from layers much deeper in the model can be used, or even the output of the fully connected layer prior to the output layer can be used.

The pre-trained model can be used as a separate feature extraction program, in which case input can be pre-processed by the model or portion of the model to a given an output (e.g. vector of numbers) for each input image, that can then use as input when training a new model.

Alternately, the pre-trained model or desired portion of the model can be integrated directly into a new neural network model. In this usage, the weights of the pre-trained can be frozen so that they are not updated as the new model is trained. Alternately, the weights may be updated during the training of the new model, perhaps with a lower learning rate, allowing the pre-trained model to act like a weight initialization scheme when training the new model.

We can summarize some of these usage patterns as follows:

- **Classifier**: The pre-trained model is used directly to classify new images.
- **Standalone Feature Extractor**: The pre-trained model, or some portion of the model, is used to pre-process images and extract relevant features.
- **Integrated Feature Extractor**: The pre-trained model, or some portion of the model, is integrated into a new model, but layers of the pre-trained model are frozen during training.
- **Weight Initialization**: The pre-trained model, or some portion of the model, is integrated into a new model, and the layers of the pre-trained model are trained in concert with the new model.

Each approach can be effective and save significant time in developing and training a deep convolutional neural network model.

It may not be clear as to which usage of the pre-trained model may yield the best results on your new computer vision task, therefore some experimentation may be required.

## Models for Transfer Learning

There are perhaps a dozen or more top-performing models for image recognition that can be downloaded and used as the basis for image recognition and related computer vision tasks.

Perhaps three of the more popular models are as follows:

- VGG (e.g. VGG16 or VGG19).
- GoogLeNet (e.g. InceptionV3).
- Residual Network (e.g. ResNet50).

These models are both widely used for transfer learning both because of their performance, but also because they were examples that introduced specific architectural innovations, namely consistent and repeating structures (VGG), inception modules (GoogLeNet), and residual modules (ResNet).

Keras provides access to a number of top-performing pre-trained models that were developed for image recognition tasks.

They are available via the Applications API, and include functions to load a model with or without the pre-trained weights, and prepare data in a way that a given model may expect (e.g. scaling of size and pixel values).

The first time a pre-trained model is loaded, Keras will download the required model weights, which may take some time given the speed of your internet connection. Weights are stored in the *.keras/models/* directory under your home directory and will be loaded from this location the next time that they are used.

When loading a given model, the "*include_top*" argument can be set to *False*, in which case the fully-connected output layers of the model used to make predictions is not loaded, allowing a new output layer to be added and trained. For example:

```
...
# load model without output layer
model = VGG16(include_top=False)
```

Additionally, when the "*include_top*" argument is *False*, the "*input_tensor*" argument must be specified, allowing the expected fixed-sized input of the model to be changed. For example:

```
...
# load model and specify a new input shape for images
new_input = Input(shape=(640, 480, 3))
model = VGG16(include_top=False, input_tensor=new_input)
```

A model without a top will output activations from the last convolutional or pooling layer directly. One approach to summarizing these activations for thier use in a classifier or as a feature vector representation of input is to add a global pooling layer, such as a max global pooling or average global pooling. The result is a vector that can be used as a feature descriptor for an input. Keras provides this capability directly via the '*pooling*' argument that can be set to '*avg*' or '*max*'. For example:

```
...
# load model and specify a new input shape for images and avg pooling output
new_input = Input(shape=(640, 480, 3))
model = VGG16(include_top=False, input_tensor=new_input, pooling='avg')
```

Images can be prepared for a given model using the *preprocess_input()* function; e.g., pixel scaling is performed in a way that was performed to images in the training dataset when the model was developed. For example:

```
...
# prepare an image
from keras.applications.vgg16 import preprocess_input
images = ...
prepared_images = preprocess_input(images)
```

Finally, you may wish to use a model architecture on your dataset, but not use the pre-trained weights, and instead initialize the model with random weights and train the model from scratch.

This can be achieved by setting the '*weights*' argument to None instead of the default '*imagenet*'. Additionally, the '*classes*' argument can be set to define the number of classes in your dataset, which will then be configured for you in the output layer of the model. For example:

```
...
# define a new model with random weights and 10 classes
new_input = Input(shape=(640, 480, 3))
model = VGG16(weights=None, input_tensor=new_input, classes=10)
```

Now that we are familiar with the API, let's take a look at loading three models using the Keras Applications API.

## Load the VGG16 Pre-trained Model

The VGG16 model was developed by the Visual Graphics Group (VGG) at Oxford and was described in the 2014 paper titled "Very Deep Convolutional Networks for Large-Scale Image Recognition."

By default, the model expects color input images to be rescaled to the size of 224×224 squares.

The model can be loaded as follows:

```
# example of loading the vgg16 model
from keras.applications.vgg16 import VGG16
# load model
model = VGG16()
# summarize the model
model.summary()
```

Running the example will load the VGG16 model and download the model weights if required.

The model can then be used directly to classify a photograph into one of 1,000 classes. In this case, the model architecture is summarized to confirm that it was loaded correctly.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
```

## Load the InceptionV3 Pre-Trained Model

The InceptionV3 is the third iteration of the inception architecture, first developed for the GoogLeNet model.

This model was developed by researchers at Google and described in the 2015 paper titled "Rethinking the Inception Architecture for Computer Vision."

The model expects color images to have the square shape 299×299.

The model can be loaded as follows:

```
# example of loading the inception v3 model
from keras.applications.inception_v3 import InceptionV3
# load model
model = InceptionV3()
# summarize the model
model.summary()
```

Running the example will load the model, downloading the weights if required, and then summarize the model architecture to confirm it was loaded correctly.

The output is omitted in this case for brevity, as it is a deep model with many layers.

## Load the ResNet50 Pre-trained Model

The Residual Network, or ResNet for short, is a model that makes use of the residual module involving shortcut connections.

It was developed by researchers at Microsoft and described in the 2015 paper titled "Deep Residual Learning for Image Recognition."

The model expects color images to have the square shape 224×224.

```
# example of loading the resnet50 model
from keras.applications.resnet50 import ResNet50
# load model
model = ResNet50()
# summarize the model
model.summary()
```

Running the example will load the model, downloading the weights if required, and then summarize the model architecture to confirm it was loaded correctly.

The output is omitted in this case for brevity, as it is a deep model.

## Examples of Using Pre-Trained Models

Now that we are familiar with how to load pre-trained models in Keras, let's look at some examples of how they might be used in practice.

In these examples, we will work with the VGG16 model as it is a relatively straightforward model to use and a simple model architecture to understand.

We also need a photograph to work with in these examples. Below is a photograph of a dog, taken by Justin Morgan and made available under a permissive license.

Photograph of a Dog

Download the photograph and place it in your current working directory with the filename '*dog.jpg*'.

- Photograph of a Dog (dog.jpg)

## Pre-Trained Model as Classifier

A pre-trained model can be used directly to classify new photographs as one of the 1,000 known classes in the image classification task in the ILSVRC.

We will use the VGG16 model to classify new images.

First, the photograph needs to loaded and reshaped to a 224×224 square, expected by the model, and the pixel values scaled in the way expected by the model. The model operates on an array of samples, therefore the dimensions of a loaded image need to be expanded by 1, for one image with 224×224 pixels and three channels.

```
# load an image from file
image = load_img('dog.jpg', target_size=(224, 224))
# convert the image pixels to a numpy array
image = img_to_array(image)
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
# prepare the image for the VGG model
image = preprocess_input(image)
```

Next, the model can be loaded and a prediction made.

This means that a predicted probability of the photo belonging to each of the 1,000 classes is made. In this example, we are only concerned with the most likely class, so we can decode the predictions and retrieve the label or name of the class with the highest probability.

```
# predict the probability across all output classes
yhat = model.predict(image)
# convert the probabilities to class labels
label = decode_predictions(yhat)
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
```

Tying all of this together, the complete example below loads a new photograph and predicts the most likely class.

```python
# example of using a pre-trained model as a classifier
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions
from keras.applications.vgg16 import VGG16
# load an image from file
image = load_img('dog.jpg', target_size=(224, 224))
# convert the image pixels to a numpy array
image = img_to_array(image)
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
# prepare the image for the VGG model
image = preprocess_input(image)
# load the model
model = VGG16()
# predict the probability across all output classes
yhat = model.predict(image)
# convert the probabilities to class labels
label = decode_predictions(yhat)
# retrieve the most likely result, e.g. highest probability
label = label[0][0]
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

Running the example predicts more than just dog; it also predicts the specific breed of '*Doberman*' with a probability of 33.59%, which may, in fact, be correct.

```
Doberman (33.59%)
```

## Pre-Trained Model as Feature Extractor Preprocessor

The pre-trained model may be used as a standalone program to extract features from new photographs.

Specifically, the extracted features of a photograph may be a vector of numbers that the model will use to describe the specific features in a photograph. These features can then be used as input in the development of a new model.

The last few layers of the VGG16 model are fully connected layers prior to the output layer. These layers will provide a complex set of features to describe a given input image and may provide useful input when training a new model for image classification or related computer vision task.

The image can be loaded and prepared for the model, as we did before in the previous example.

We will load the model with the classifier output part of the model, but manually remove the final output layer. This means that the second last fully connected layer with 4,096 nodes will be the new output layer.

```
# load model
model = VGG16()
# remove the output layer
model.layers.pop()
model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
```

This vector of 4,096 numbers will be used to represent the complex features of a given input image that can then be saved to file to be loaded later and used as input to train a new model. We can save it as a pickle file.

```
# get extracted features
features = model.predict(image)
print(features.shape)
# save to file
dump(features, open('dog.pkl', 'wb'))
```

Tying all of this together, the complete example of using the model as a standalone feature extraction model is listed below.

```
# example of using the vgg16 model as a feature extraction model
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions
from keras.applications.vgg16 import VGG16
from keras.models import Model
from pickle import dump
# load an image from file
image = load_img('dog.jpg', target_size=(224, 224))
# convert the image pixels to a numpy array
image = img_to_array(image)
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
# prepare the image for the VGG model
image = preprocess_input(image)
# load model
model = VGG16()
# remove the output layer
model.layers.pop()
model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
# get extracted features
features = model.predict(image)
print(features.shape)
# save to file
dump(features, open('dog.pkl', 'wb'))
```

Running the example loads the photograph, then prepares the model as a feature extraction model.

The features are extracted from the loaded photo and the shape of the feature vector is printed, showing it has 4,096 numbers. This feature is then saved to a new file *dog.pkl* in the current working directory.

```
(1, 4096)
```

This process could be repeated for each photo in a new training dataset.

## Pre-Trained Model as Feature Extractor in Model

We can use some or all of the layers in a pre-trained model as a feature extraction component of a new model directly.

This can be achieved by loading the model, then simply adding new layers. This may involve adding new convolutional and pooling layers to expand upon the feature extraction capabilities of the model or adding new fully connected classifier type layers to learn how to interpret the extracted features on a new dataset, or some combination.

For example, we can load the VGG16 models without the classifier part of the model by specifying the "*include_top*" argument to "*False*", and specify the preferred shape of the images in our new dataset as 300×300.

```
# load model without classifier layers
model = VGG16(include_top=False, input_shape=(300, 300, 3))
```

We can then use the Keras function API to add a new Flatten layer after the last pooling layer in the VGG16 model, then define a new classifier model with a Dense fully connected layer and an output layer that will predict the probability for 10 classes.

```
# add new classifier layers
flat1 = Flatten()(model.outputs)
class1 = Dense(1024, activation='relu')(flat1)
output = Dense(10, activation='softmax')(class1)
# define new model
model = Model(inputs=model.inputs, outputs=output)
```

An alternative approach to adding a Flatten layer would be to define the VGG16 model with an average pooling layer, and then add fully connected layers. Perhaps try both approaches on your application and see which results in the best performance.

The weights of the VGG16 model and the weights for the new model will all be trained together on the new dataset.

The complete example is listed below.

```
# example of tending the vgg16 model
from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.layers import Dense
from keras.layers import Flatten
# load model without classifier layers
model = VGG16(include_top=False, input_shape=(300, 300, 3))
# add new classifier layers
flat1 = Flatten()(model.outputs)
class1 = Dense(1024, activation='relu')(flat1)
output = Dense(10, activation='softmax')(class1)
# define new model
model = Model(inputs=model.inputs, outputs=output)
```

```
# summarize
model.summary()
# ...
```

Running the example defines the new model ready for training and summarizes the model architecture.

We can see that we have flattened the output of the last pooling layer and added our new fully connected layers.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 300, 300, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 300, 300, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 300, 300, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 150, 150, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 150, 150, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 150, 150, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 75, 75, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 75, 75, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 75, 75, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 75, 75, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 37, 37, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 37, 37, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 37, 37, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 37, 37, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 18, 18, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 41472)             0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              42468352
_________________________________________________________________
dense_2 (Dense)              (None, 10)                10250
=================================================================
Total params: 57,193,290
Trainable params: 57,193,290
Non-trainable params: 0
```

```
_________________________________________________________________
```

Alternately, we may wish to use the VGG16 model layers, but train the new layers of the model without updating the weights of the VGG16 layers. This will allow the new output layers to learn to interpret the learned features of the VGG16 model.

This can be achieved by setting the "*trainable*" property on each of the layers in the loaded VGG model to False prior to training. For example:

```
# load model without classifier layers
model = VGG16(include_top=False, input_shape=(300, 300, 3))
# mark loaded layers as not trainable
for layer in model.layers:
    layer.trainable = False
...
```

You can pick and choose which layers are trainable.

For example, perhaps you want to retrain some of the convolutional layers deep in the model, but none of the layers earlier in the model. For example:

```
# load model without classifier layers
model = VGG16(include_top=False, input_shape=(300, 300, 3))
# mark some layers as not trainable
model.get_layer('block1_conv1').trainable = False
model.get_layer('block1_conv2').trainable = False
model.get_layer('block2_conv1').trainable = False
model.get_layer('block2_conv2').trainable = False
...
```

## Further Reading

This section provides more resources on the topic if you are looking to go deeper.

### Posts

- How to Improve Performance With Transfer Learning for Deep Learning Neural Networks
- A Gentle Introduction to Transfer Learning for Deep Learning
- How to Use The Pre-Trained VGG Model to Classify Objects in Photographs

### Books

- Deep Learning, 2016.

### Papers

- A Survey on Transfer Learning, 2010.
- How transferable are features in deep neural networks?, 2014.
- CNN features off-the-shelf: An astounding baseline for recognition, 2014.

### APIs

- Keras Applications API

### Articles

- Transfer Learning, Wikipedia.
- Transfer Learning – Machine Learning's Next Frontier, 2017.

## Summary

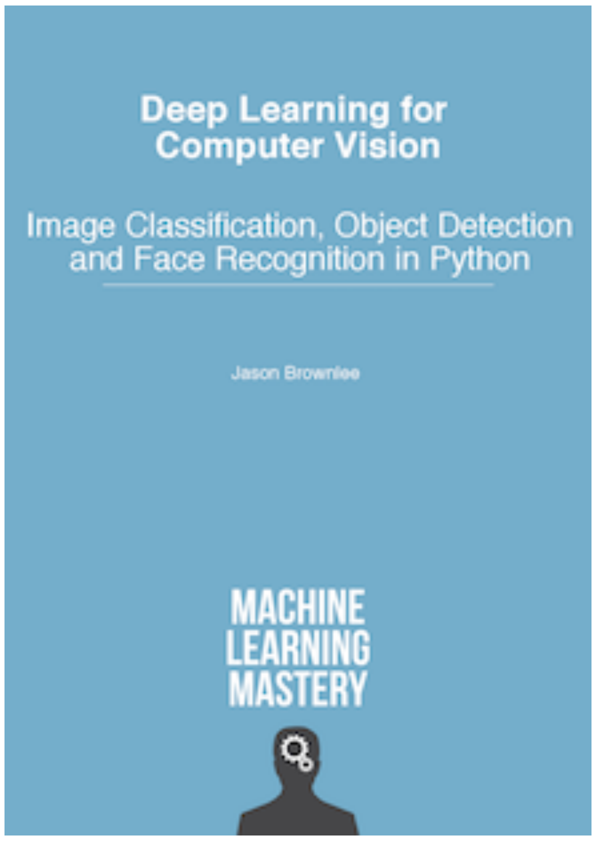
In this post, you discovered how to use transfer learning when developing convolutional neural networks for computer vision applications.

Specifically, you learned:

- Transfer learning involves using models trained on one problem as a starting point on a related problem.
- Transfer learning is flexible, allowing the use of pre-trained models directly as feature extraction preprocessing and integrated into entirely new models.
- Keras provides convenient access to many top performing models on the ImageNet image recognition tasks such as VGG, Inception, and ResNet.

Do you have any questions?
Ask your questions in the comments below and I will do my best to answer.

---


Deep Learning for Computer Vision
Image Classification, Object Detection and Face Recognition in Python
Jason Brownlee
MACHINE LEARNING MASTERY

Tweet Share Share

**About Jason Brownlee**

Jason Brownlee, PhD is a machine learning specialist who teaches developers how to get results with modern machine learning methods via hands-on tutorials.

View all posts by Jason Brownlee →

‹ How to Save and Load Your Keras Deep Learning Model

How to Classify Photos of Dogs and Cats (with 97% accuracy) ›

## 35 Responses to *Transfer Learning in Keras with Computer Vision Models*

**Ludwig** May 15, 2019 at 8:37 am # REPLY

amazing post! thank you.

**Jason Brownlee** May 15, 2019 at 2:41 pm # REPLY

Thanks, I'm glad it helped!

**Andre** October 1, 2019 at 9:15 pm #

REPLY

can I train one of the models with rectangular input image?, eg :input_shape=(100, 150, 3)

**Jason Brownlee** October 2, 2019 at 7:59 am #

REPLY

Yes.

**Sujit Nalawade** May 17, 2019 at 11:13 pm #

REPLY

best post i've read on transfer learning

**Jason Brownlee** May 18, 2019 at 7:38 am #

REPLY

Thank you very much!

**halima** May 21, 2019 at 8:21 am #

REPLY

hello . is transfer learning just for image recognition?
can i use it in another apps? . for example i have models using classification methods of machine learning but not deep learning or neural networks ,
can i use transfer learning to have the knowledge from one model and use it as feature in another model ?
and if i can how i will do this ?
thanks a lot.

**Jason Brownlee** May 21, 2019 at 2:41 pm #

REPLY

No, transfer learning is also very common for NLP e.g. text data.

You could use it for tabular data too, if you had enough data and pre-trained models.

**Atefeh** July 22, 2019 at 6:27 pm #

REPLY

Hello Mr.Brownlee
After saving the feature vector gained from the above example,is it possible to use these vectors directly as the inputs for a LSTM?

**Jason Brownlee** July 23, 2019 at 7:58 am #

REPLY

Sure.

The photo captioning model does this from memory, for example:
https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python/

**Rakesh** July 27, 2019 at 9:40 pm #

REPLY

well written, super helpful!

Just a small typo – you may want to change to (300,300,3) in this line (first snippet under "Pre-Trained Model as Feature Extractor in Model"):
model = VGG16(include_top=False, input_shape=(224, 224, 3))

**Jason Brownlee** July 28, 2019 at 6:44 am #

REPLY

Thanks, fixed!

**Gautam KIshore Shahi** August 29, 2019 at 11:59 pm #

REPLY

Hello,

I am writing to use the concept and the code mentioned for my data, data is of binary class after loading the image, when I click on the predict, I get the error, Uncaught (in promise) Error: Error when checking: expected tfjs@0.13.5.2 flatten_1_input to have shape [null,7,7,512] but got array with shape [1,224,224,3].

Can you please help me to solve this issue?

**Jason Brownlee** August 30, 2019 at 6:25 am # REPLY

Sorry, I have not seen this issue.

Perhaps try posting your code and error to stackoverflow?

**Jacques Thibodeau** September 27, 2019 at 6:36 am # REPLY

Does it make sense to use convolutional layers in the model you build on top of the pre-trained model? I'm looking for examples of this, but only really find fully connected layers after our pre-trained model. If we can, how should we go about it? If we can't, why not?

**Jason Brownlee** September 27, 2019 at 8:06 am # REPLY

It may, if you want to interpret the learned features by another model without messing them up via adjusting their weights.

I may have an example, perhaps check some of the tutorials for image classification here:
https://machinelearningmastery.com/start-here/#dlfcv

**Jacques Thibodeau** September 27, 2019 at 11:02 pm # REPLY

Yes, that makes sense!

In the context of building a more accurate transfer learning model, it would make less sense? At the moment I'm working on the pneumonia Kaggle dataset while using InceptionV3 as the pre-trained model. The dataset and number of classes are quite small compared to imagenet. Would retraining some of the layers within Inception be a good idea?

(I'm getting the impression that in this context I should just train the model, and then tune it based on what I get instead of getting too theoretical.)

**Jason Brownlee** September 28, 2019 at 6:18 am # REPLY

I think starting with a pre-trained model is almost always the way to go, and tuning the output layers or adding some new layers and tuning them should be tried.

**Theekshana** October 15, 2019 at 5:57 pm #

REPLY

Hi Jason,

I'm planning to use Transfer learning for ECG signal related classification project. I'm thinking of using existing models in Keras API (since it's difficult to find a signal related pre-trained model.)

Talking about the data set, I have only 1000 signal samples. Therefore, now the transfer learning problem narrows down to "target dataset is small and different from the base training dataset" problem.

Can you please provide me with some suggestions to approach this (books, research papers).

Big fan of your website. Thanks.

Theekshana.

**Jason Brownlee** October 16, 2019 at 7:58 am #

REPLY

Perhaps you can use a model from a related time series classification problem for transfer learning?

**sneh** October 25, 2019 at 6:15 pm #

REPLY

hi.
can we use vgg16 which is trained on image dataset for an pima-Indian diabetes dataset?
or can a image based Pretrained model can be used for non-image dataset?

**Jason Brownlee** October 26, 2019 at 4:36 am #

REPLY

Not at all.

**Dave** October 27, 2019 at 9:31 pm #

REPLY

So with this approach, how would one go about training multiple classes?

**Dave** October 27, 2019 at 9:33 pm #

REPLY

Sorry, wrong article, please ignore…

**Jason Brownlee** October 28, 2019 at 6:03 am #

REPLY

No problem.

**Atefeh** November 8, 2019 at 5:55 pm #

REPLY

Hello Mr.Brownlee
In the last code(Pre-Trained Model as Feature Extractor in Model),where does the feature vectors has been saved?
I mean how can i use the feature vectors?
for example if i have 3000 images which are from 10 classes(each class has 300 images),how can i have the feature vectors of each image in a class?

in your last code, if we have 3000 images as the network input,then do we have 3000 feature vectors? and so if it is right, how we can access to those vectors?

thank you

**Jason Brownlee** November 9, 2019 at 6:11 am #

REPLY

You can pass an image through the model to get the feature vectors for the image.

You can repeat this for all your images and save the array of vectors to file if you wish.

**MonaST** November 16, 2019 at 11:10 pm #

REPLY

Hi,
I tried to fit the following model but there are an error and I did not know how to fix it. Could you help me , please.

InvalidArgumentError Traceback (most recent call last)
in
1 # train
—-> 2 history=model.fit(X_train, y_train, batch_size=32, epochs=1,validation_data=

```
(X_test,y_test),shuffle=False,verbose=2)

C:\Anaconda64\lib\site-packages\keras\engine\training.py in fit(self, x, y, batch_size, epochs, verbose,
callbacks, validation_split, validation_data, shuffle, class_weight, sample_weight, initial_epoch,
steps_per_epoch, validation_steps, validation_freq, max_queue_size, workers, use_multiprocessing,
**kwargs)
1237 steps_per_epoch=steps_per_epoch,
1238 validation_steps=validation_steps,
-> 1239 validation_freq=validation_freq)
1240
1241 def evaluate(self,

C:\Anaconda64\lib\site-packages\keras\engine\training_arrays.py in fit_loop(model, fit_function,
fit_inputs, out_labels, batch_size, epochs, verbose, callbacks, val_function, val_inputs, shuffle,
initial_epoch, steps_per_epoch, validation_steps, validation_freq)
194 ins_batch[i] = ins_batch[i].toarray()
195
–> 196 outs = fit_function(ins_batch)
197 outs = to_list(outs)
198 for l, o in zip(out_labels, outs):

C:\Anaconda64\lib\site-packages\tensorflow_core\python\keras\backend.py in __call__(self, inputs)
3738 value = math_ops.cast(value, tensor.dtype)
3739 converted_inputs.append(value)
-> 3740 outputs = self._graph_fn(*converted_inputs)
3741
3742 # EagerTensor.numpy() will often make a copy to ensure memory safety.

C:\Anaconda64\lib\site-packages\tensorflow_core\python\eager\function.py in __call__(self, *args,
**kwargs)
1079 TypeError: For invalid positional/keyword argument combinations.
1080 “””
-> 1081 return self._call_impl(args, kwargs)
1082
1083 def _call_impl(self, args, kwargs, cancellation_manager=None):

C:\Anaconda64\lib\site-packages\tensorflow_core\python\eager\function.py in _call_impl(self, args,
kwargs, cancellation_manager)
1119 raise TypeError(“Keyword arguments {} unknown. Expected {}.”.format(
1120 list(kwargs.keys()), list(self._arg_keywords)))
-> 1121 return self._call_flat(args, self.captured_inputs, cancellation_manager)
1122
1123 def _filtered_call(self, args, kwargs):

C:\Anaconda64\lib\site-packages\tensorflow_core\python\eager\function.py in _call_flat(self, args,
captured_inputs, cancellation_manager)
```

```
1222 if executing_eagerly:
1223 flat_outputs = forward_function.call(
-> 1224 ctx, args, cancellation_manager=cancellation_manager)
1225 else:
1226 gradient_name = self._delayed_rewrite_functions.register()

C:\Anaconda64\lib\site-packages\tensorflow_core\python\eager\function.py in call(self, ctx, args,
cancellation_manager)
509 inputs=args,
510 attrs=("executor_type", executor_type, "config_proto", config),
–> 511 ctx=ctx)
512 else:
513 outputs = execute.execute_with_cancellation(

C:\Anaconda64\lib\site-packages\tensorflow_core\python\eager\execute.py in quick_execute(op_name,
num_outputs, inputs, attrs, ctx, name)
65 else:
66 message = e.message
—> 67 six.raise_from(core._status_to_exception(e.code, message), None)
68 except TypeError as e:
69 keras_symbolic_tensors = [

C:\Anaconda64\lib\site-packages\six.py in raise_from(value, from_value)

InvalidArgumentError: Matrix size-incompatible: In[0]: [1,802816], In[1]: [25088,1024]
[[node dense_1/Relu (defined at C:\Anaconda64\lib\site-
packages\tensorflow_core\python\framework\ops.py:1751) ]]
[Op:__inference_keras_scratch_graph_1052]

Function call stack:
keras_scratch_graph
```

My model is:

```
load model without classifier layers without change imput size
modelVGG = VGG16(include_top=False, input_shape=(224, 224, 3))
# add new classifier layers
flat1 = Flatten()(modelVGG.outputs)
class1 = Dense(1024, activation='relu')(flat1)
output = Dense(49, activation='sigmoid')(class1)
# define new model
model = Model(inputs=modelVGG.inputs, outputs=output)
# mark loaded layers as not trainable
for layer in modelVGG.layers:
layer.trainable = Falsee

print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)

(1452, 224, 224, 3)
(1452, 49)
(435, 224, 224, 3)
(435, 49)
```

**Jason Brownlee** November 17, 2019 at 7:14 am # REPLY

Sorry to hear that, this may help:
https://machinelearningmastery.com/faq/single-faq/why-does-the-code-in-the-tutorial-not-work-for-me

**Hadi** November 19, 2019 at 8:13 pm # REPLY

amazing post!
Thank you for your time.

**Jason Brownlee** November 20, 2019 at 6:11 am # REPLY

Thanks!

**shazia saqib** November 22, 2019 at 1:23 am # REPLY

you are a great support

**Jason Brownlee** November 22, 2019 at 6:07 am # REPLY

Thanks!

**saif** November 24, 2019 at 6:57 am # REPLY can i use pretrained model to facial emotion recognition with fer2013 dataset?

**Jason Brownlee** November 24, 2019 at 9:24 am #

REPLY

I don't know, perhaps try it?

## Leave a Reply

Name (required)

Email (will not be published) (required)

Website

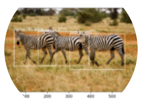
SUBMIT COMMENT

*Welcome!* I'm **Jason Brownlee** PhD and I help developers get results with machine learning.
Read More

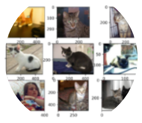
**Picked for you:**

How to Train an Object Detection Model with Keras

How to Develop a Face Recognition System Using FaceNet in Keras



How to Perform Object Detection With YOLOv3 in Keras



How to Classify Photos of Dogs and Cats (with 97% accuracy)

A Gentle Introduction to Transfer Learning for Deep Learning