A Gentle Introduction to 1×1 Convolutions to Reduce the Complexity of Convolutional Neural Networks

by **Jason Brownlee** on April 29, 2019 in **Deep Learning for Computer Vision**

Pooling can be used to down sample the content of feature maps, reducing their width and height whilst maintaining their salient features.

A problem with deep convolutional neural networks is that the number of feature maps often increases with the depth of the network. This problem can result in a dramatic increase in the number of parameters and computation required when larger filter sizes are used, such as 5×5 and 7×7.

To address this problem, a 1×1 convolutional layer can be used that offers a channel-wise pooling, often called feature map pooling or a projection layer. This simple technique can be used for dimensionality reduction, decreasing the number of feature maps whilst retaining their salient features. It can also be used directly to create a one-to-one projection of the feature maps to pool features across channels or to increase the number of feature maps, such as after traditional pooling layers.

In this tutorial, you will discover how to use 1×1 filters to control the number of feature maps in a convolutional neural network.

After completing this tutorial, you will know:

- The 1×1 filter can be used to create a linear projection of a stack of feature maps.
- The projection created by a 1×1 can act like channel-wise pooling and be used for dimensionality reduction.
- The projection created by a 1×1 can also be used directly or be used to increase the number of feature maps in a model.

Let's get started.

A Gentle Introduction to 1×1 Convolutions to Reduce the Complexity of Convolutional Neural Networks

Tutorial Overview

This tutorial is divided into five parts; they are:

1. Convolutions Over Channels
2. Problem of Too Many Feature Maps
3. Downsample Feature Maps With 1×1 Filters
4. Examples of How to Use 1×1 Convolutions
5. Examples of 1×1 Filters in CNN Model Architectures

Convolutions Over Channels

Recall that a convolutional operation is a linear application of a smaller filter to a larger input that results in an output feature map.

A filter applied to an input image or input feature map always results in a single number. The systematic left-to-right and top-to-bottom application of the filter to the input results in a two-dimensional feature map. One filter creates one corresponding feature map.

A filter must have the same depth or number of channels as the input, yet, regardless of the depth of the input and the filter, the resulting output is a single number and one filter creates a feature map with a single channel.

Let's make this concrete with some examples:

- If the input has one channel such as a grayscale image, then a 3×3 filter will be applied in 3x3x1 blocks.
- If the input image has three channels for red, green, and blue, then a 3×3 filter will be applied in 3x3x3 blocks.
- If the input is a block of feature maps from another convolutional or pooling layer and has the depth of 64, then the 3×3 filter will be applied in 3x3x64 blocks to create the single values to make up the single output feature map.

The depth of the output of one convolutional layer is only defined by the number of parallel filters applied to the input.

Problem of Too Many Feature Maps

The depth of the input or number of filters used in convolutional layers often increases with the depth of the network, resulting in an increase in the number of resulting feature maps. It is a common model design pattern.

Further, some network architectures, such as the inception architecture, may also concatenate the output feature maps from multiple convolutional layers, which may also dramatically increase the depth of the input to subsequent convolutional layers.

A large number of feature maps in a convolutional neural network can cause a problem as a convolutional operation must be performed down through the depth of the input. This is a particular problem if the convolutional operation being performed is relatively large, such as 5×5 or 7×7 pixels, as it can result in considerably more parameters (weights) and, in turn, computation to perform the convolutional operations (large space and time complexity).

Pooling layers are designed to downscale feature maps and systematically halve the width and height of feature maps in the network. Nevertheless, pooling layers do not change the number of filters in the model, the depth, or number of channels.

Deep convolutional neural networks require a corresponding pooling type of layer that can downsample or reduce the depth or number of feature maps.

Downsample Feature Maps With 1×1 Filters

The solution is to use a 1×1 filter to down sample the depth or number of feature maps.

A 1×1 filter will only have a single parameter or weight for each channel in the input, and like the application of any filter results in a single output value. This structure allows the 1×1 filter to act like a single neuron with an input from the same position across each of the feature maps in the input. This single neuron can then be applied systematically with a stride of one, left-to-right and top-to-bottom without any need for padding, resulting in a feature map with the same width and height as the input.

The 1×1 filter is so simple that it does not involve any neighboring pixels in the input; it may not be considered a convolutional operation. Instead, it is a linear weighting or projection of the input. Further, a nonlinearity is used as with other convolutional layers, allowing the projection to perform non-trivial computation on the input feature maps.

This simple 1×1 filter provides a way to usefully summarize the input feature maps. The use of multiple 1×1 filters, in turn, allows the tuning of the number of summaries of the input feature maps to create, effectively allowing the depth of the feature maps to be increased or decreased as needed.

A convolutional layer with a 1×1 filter can, therefore, be used at any point in a convolutional neural network to control the number of feature maps. As such, it is often referred to as a projection operation or projection layer, or even a feature map or channel pooling layer.

Now that we know that we can control the number of feature maps with 1×1 filters, let's make it concrete with some examples.

Examples of How to Use 1×1 Convolutions

We can make the use of a 1×1 filter concrete with some examples.

Consider that we have a convolutional neural network that expected color images input with the square shape of 256x256x3 pixels.

These images then pass through a first hidden layer with 512 filters, each with the size of 3×3 with the same padding, followed by a ReLU activation function.

The example below demonstrates this simple model.

```
# example of simple cnn model
from keras.models import Sequential
from keras.layers import Conv2D
# create model
model = Sequential()
model.add(Conv2D(512, (3,3), padding='same', activation='relu', input_shape=(256, 256, 3)))
# summarize model
model.summary()
```

Running the example creates the model and summarizes the model architecture.

There are no surprises; the output of the first hidden layer is a block of feature maps with the three-dimensional shape of 256x256x512.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 256, 256, 512)     14336
=================================================================
Total params: 14,336
Trainable params: 14,336
Non-trainable params: 0
_________________________________________________________________
```

Example of Projecting Feature Maps

A 1×1 filter can be used to create a projection of the feature maps.

The number of feature maps created will be the same number and the effect may be a refinement of the features already extracted. This is often called channel-wise pooling, as opposed to traditional feature-wise pooling on each channel. It can be implemented as follows:

```
model.add(Conv2D(512, (1,1), activation='relu'))
```

We can see that we use the same number of features and still follow the application of the filter with a rectified linear activation function.

The complete example is listed below.

```
# example of a 1x1 filter for projection
from keras.models import Sequential
from keras.layers import Conv2D
# create model
model = Sequential()
model.add(Conv2D(512, (3,3), padding='same', activation='relu', input_shape=(256, 256, 3)))
model.add(Conv2D(512, (1,1), activation='relu'))
# summarize model
model.summary()
```

Running the example creates the model and summarizes the architecture.

We can see that no change is made to the width or height of the feature maps, and by design, the number of feature maps is kept constant with a simple projection operation applied.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 256, 256, 512)     14336
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 256, 256, 512)     262656
=================================================================
Total params: 276,992
Trainable params: 276,992
Non-trainable params: 0
_________________________________________________________________
```

Example of Decreasing Feature Maps

The 1×1 filter can be used to decrease the number of feature maps.

This is the most common application of this type of filter and in this way, the layer is often called a feature map pooling layer.

In this example, we can decrease the depth (or channels) from 512 to 64. This might be useful if the subsequent layer we were going to add to our model would be another convolutional layer with 7×7 filters. These filters would only be applied at a depth of 64 rather than 512.

```
model.add(Conv2D(64, (1,1), activation='relu'))
```

The composition of the 64 feature maps is not the same as the original 512, but contains a useful summary of dimensionality reduction that captures the salient features, such that the 7×7 operation may have a similar effect on the 64 feature maps as it might have on the original 512.

Further, a 7×7 convolutional layer with 64 filters itself applied to the 512 feature maps output by the first hidden layer would result in approximately one million parameters (weights). If the 1×1 filter is used to reduce the number of feature maps to 64 first, then the number of parameters required for the 7×7 layer is only approximately 200,000, an enormous difference.

The complete example of using a 1×1 filter for dimensionality reduction is listed below.

```
# example of a 1x1 filter for dimensionality reduction
from keras.models import Sequential
from keras.layers import Conv2D
# create model
model = Sequential()
model.add(Conv2D(512, (3,3), padding='same', activation='relu', input_shape=(256, 256, 3)))
model.add(Conv2D(64, (1,1), activation='relu'))
# summarize model
model.summary()
```

Running the example creates the model and summarizes its structure.

We can see that the width and height of the feature maps are unchanged, yet the number of feature maps was reduced from 512 to 64.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 256, 256, 512)     14336
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 256, 256, 64)      32832
=================================================================
Total params: 47,168
Trainable params: 47,168
Non-trainable params: 0
_________________________________________________________________
```

Example of Increasing Feature Maps

The 1×1 filter can be used to increase the number of feature maps.

This is a common operation used after a pooling layer prior to applying another convolutional layer.

The projection effect of the filter can be applied as many times as needed to the input, allowing the number of feature maps to be scaled up and yet have a composition that captures the salient features of the original.

We can increase the number of feature maps from 512 input from the first hidden layer to double the size at 1,024 feature maps.

```
model.add(Conv2D(1024, (1,1), activation='relu'))
```

The complete example is listed below.

```
# example of a 1x1 filter to increase dimensionality
from keras.models import Sequential
from keras.layers import Conv2D
# create model
model = Sequential()
model.add(Conv2D(512, (3,3), padding='same', activation='relu', input_shape=(256, 256, 3)))
model.add(Conv2D(1024, (1,1), activation='relu'))
# summarize model
model.summary()
```

Running the example creates the model and summarizes its structure.

We can see that the width and height of the feature maps are unchanged and that the number of feature maps was increased from 512 to double the size at 1,024.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 256, 256, 512)     14336
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 256, 256, 1024)    525312
=================================================================
Total params: 539,648
Trainable params: 539,648
Non-trainable params: 0
_________________________________________________________________
```

Now that we are familiar with how to use 1×1 filters, let's look at some examples where they have been used in the architecture of convolutional neural network models.

Examples of 1×1 Filters in CNN Model Architectures

In this section, we will highlight some important examples where 1×1 filters have been used as key elements in modern convolutional neural network model architectures.

Network in Network

The 1×1 filter was perhaps first described and popularized in the 2013 paper by Min Lin, et al. in their paper titled "Network In Network."

In the paper, the authors propose the need for an MLP convolutional layer and the need for cross-channel pooling to promote learning across channels.

> *This cascaded cross channel parametric pooling structure allows complex and learnable interactions of cross channel information.*

— Network In Network, 2013.

They describe a 1×1 convolutional layer as a specific implementation of cross-channel parametric pooling, which, in effect, that is exactly what a 1×1 filter achieves.

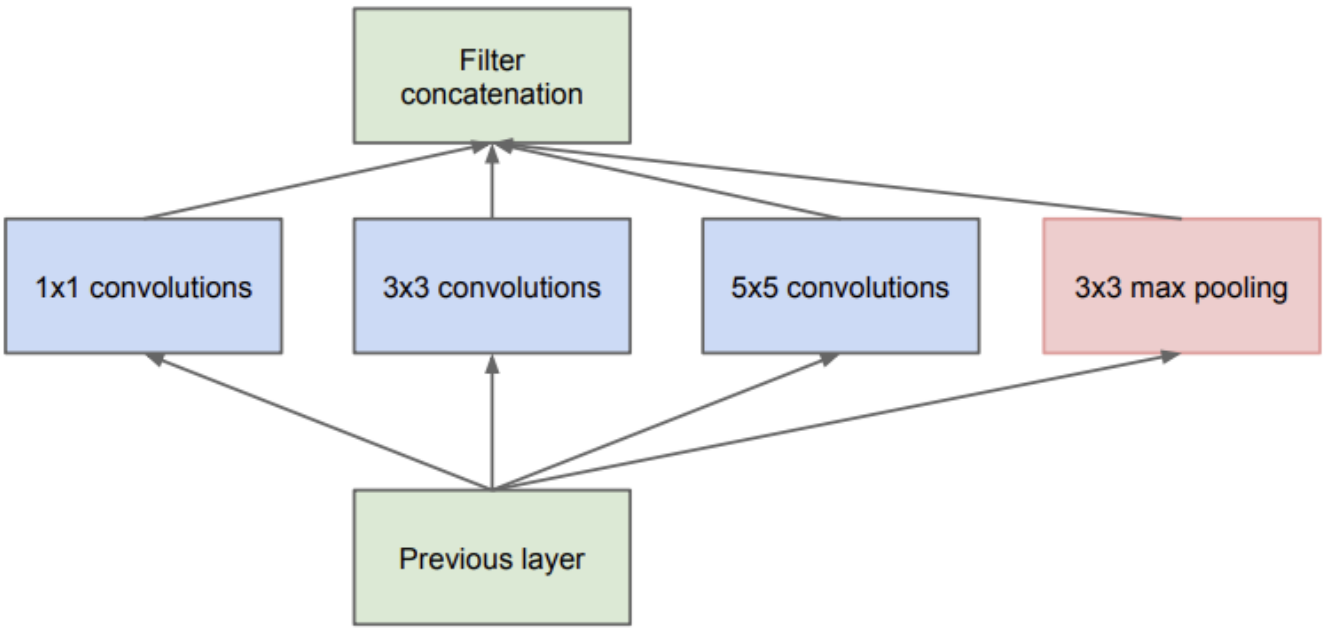
> *Each pooling layer performs weighted linear recombination on the input feature maps, which then go through a rectifier linear unit. […] The cross channel parametric pooling layer is also equivalent to a convolution layer with 1×1 convolution kernel.*

— Network In Network, 2013.

Inception Architecture

The 1×1 filter was used explicitly for dimensionality reduction and for increasing the dimensionality of feature maps after pooling in the design of the inception module, used in the GoogLeNet model by Christian Szegedy, et al. in their 2014 paper titled "Going Deeper with Convolutions."

The paper describes an "*inception module*" where an input block of feature maps is processed in parallel by different convolutional layers each with differently sized filters, where a 1×1 size filter is one of the layers used.



Example of the Naive Inception Module
Taken from Going Deeper with Convolutions, 2014.

The output of the parallel layers are then stacked, channel-wise, resulting in very deep stacks of convolutional layers to be processed by subsequent inception modules.

> *The merging of the output of the pooling layer with the outputs of convolutional layers would lead to an inevitable increase in the number of outputs from stage to stage. Even while this architecture might cover the optimal sparse structure, it would do it very inefficiently, leading to a computational blow up within a few stages.*

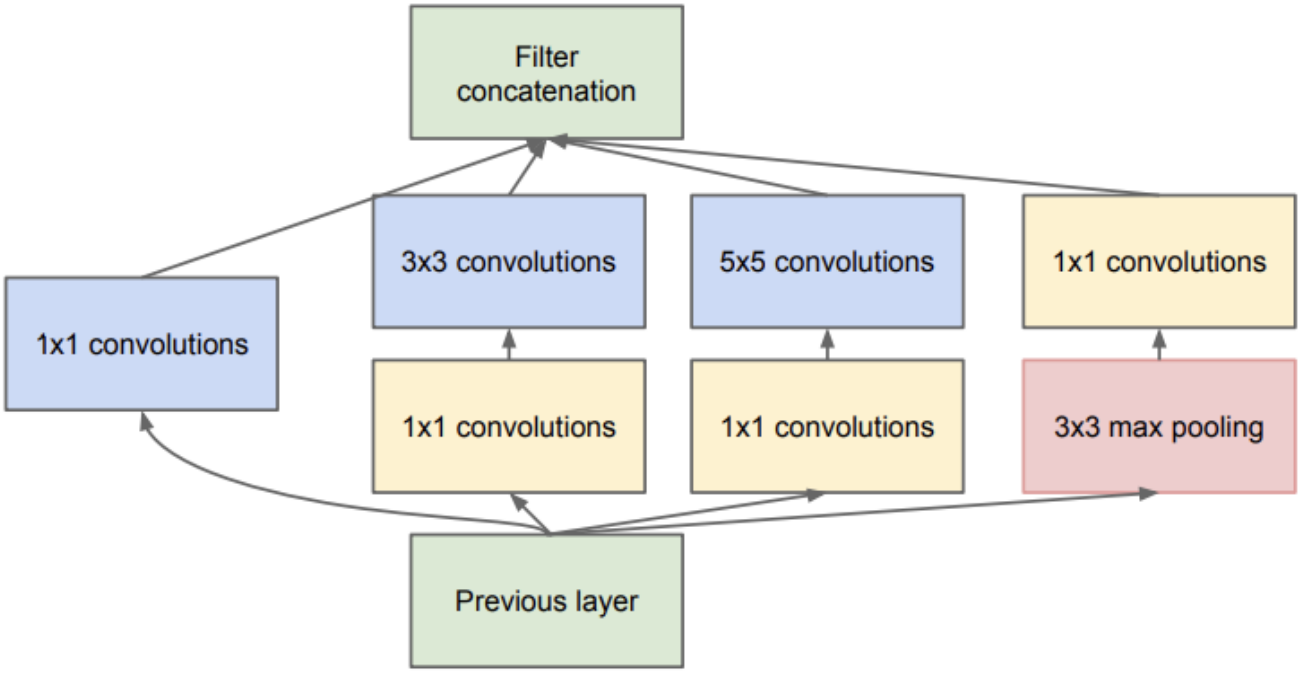
— Going Deeper with Convolutions, 2014.

The inception module is then redesigned to use 1×1 filters to reduce the number of feature maps prior to parallel convolutional layers with 5×5 and 7×7 sized filters.

> *This leads to the second idea of the proposed architecture: judiciously applying dimension reductions and projections wherever the computational requirements would increase too much otherwise. […] That is, 1×1 convolutions are used to compute reductions before the expensive 3×3 and 5×5 convolutions. Besides being used as reductions, they also include the use of rectified linear activation which makes them dual-purpose*

— Going Deeper with Convolutions, 2014.

The 1×1 filter is also used to increase the number of feature maps after pooling, artificially creating more projections of the downsampled feature map content.



Example of the Inception Module With Dimensionality Reduction
Taken from Going Deeper with Convolutions, 2014.

Residual Architecture

The 1×1 filter was used as a projection technique to match the number of filters of input to the output of residual modules in the design of the residual network by Kaiming He, et al. in their 2015 paper titled "Deep Residual Learning for Image Recognition."

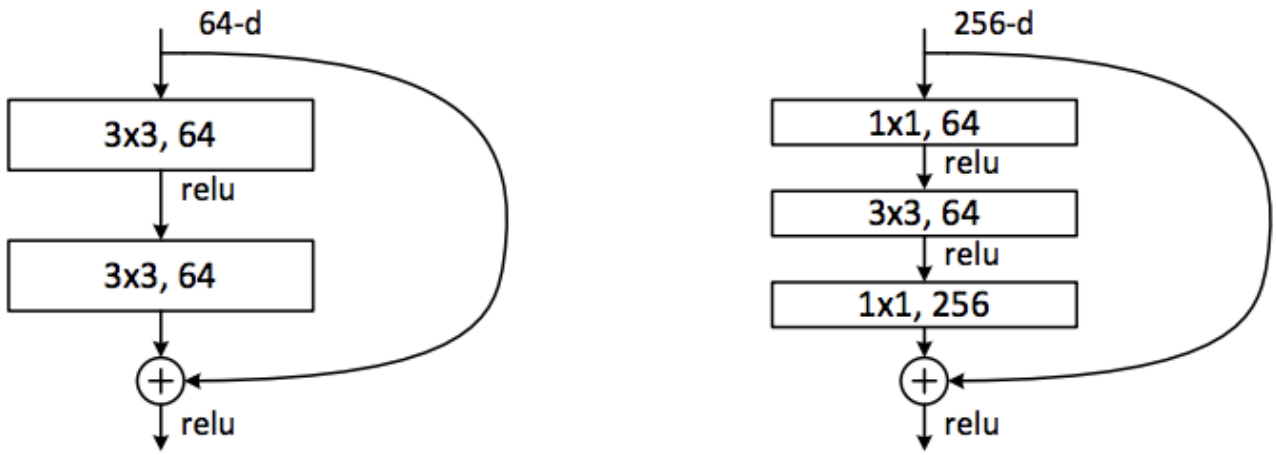
The authors describe an architecture comprised of "*residual modules*" where the input to a module is added to the output of the module in what is referred to as a shortcut connection.

Because the input is added to the output of the module, the dimensionality must match in terms of width, height, and depth. Width and height can be maintained via padding, although a 1×1 filter is used to change the depth of the input as needed so that it can be added with the output of the module. This type of connection is referred to as a projection shortcut connection.

Further, the residual modules use a bottleneck design with 1×1 filters to reduce the number of feature maps for computational efficiency reasons.

> *The three layers are 1×1, 3×3, and 1×1 convolutions, where the 1×1 layers are responsible for reducing and then increasing (restoring) dimensions, leaving the 3×3 layer a bottleneck with smaller input/output dimensions.*

— Deep Residual Learning for Image Recognition, 2015.



Example of a Normal and Bottleneck Residual Modules With Shortcut Connections
Taken from Deep Residual Learning for Image Recognition, 2015.

Further Reading

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

Papers

- Network In Network, 2013.
- Going Deeper with Convolutions, 2014.
- Deep Residual Learning for Image Recognition, 2015.

Articles

- One by One [1 x 1] Convolution – counter-intuitively useful, 2016.
- Yann LeCun on No Fully Connected Layers in CNN, 2015.
- Networks in Networks and 1×1 Convolutions, YouTube.

Summary

In this tutorial, you discovered how to use 1×1 filters to control the number of feature maps in a convolutional neural network.

Specifically, you learned:

- The 1×1 filter can be used to create a linear projection of a stack of feature maps.

- The projection created by a 1×1 can act like channel-wise pooling and be used for dimensionality reduction.
- The projection created by a 1×1 can also be used directly or be used to increase the number of feature maps in a model.

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

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 Implement VGG, Inception and ResNet Modules for Convolutional Neural Networks from Scratch
A Gentle Introduction to the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) ›

6 Responses to *A Gentle Introduction to 1×1 Convolutions to Reduce the Complexity of Convolutional Neural Networks*

Hu Chunfeng April 29, 2019 at 12:18 pm #

REPLY

Hi Jason,do you know how to implement temporal convolutional network(TCN) for multivariate time series forecasting.I don't know how to write the code.

Jason Brownlee April 29, 2019 at 2:05 pm #

REPLY

Sorry, I don't have a tutorial on that topic, I hope to cover it in the future.

Hu Chunfeng April 29, 2019 at 4:19 pm #

REPLY

Thank you! Your tutorials have helped me a lot! Looking forward to your tutorial about TCN.

Jason Brownlee April 30, 2019 at 6:47 am #

REPLY

Thanks.

yogi May 1, 2019 at 8:29 pm #

REPLY

Great sir, this method can we use for semantic segmentation ? for decoder

Jason Brownlee May 2, 2019 at 8:01 am #

REPLY

Typically a method such as Mask RCNN is used for semantic segmentation of images.

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

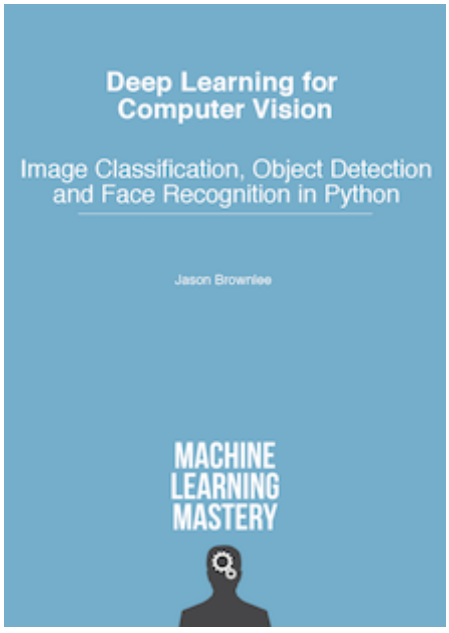
Welcome to Machine Learning Mastery!

Hi, I'm **Jason Brownlee**, PhD.
I write tutorials to help developers (*like you*) get results with machine learning.

Read More

Deep Learning for Computer Vision
Classify photos, Detect objects, Recognize faces.

Click to Get Started Now!

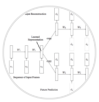


POPULAR

How to Develop LSTM Models for Multi-Step Time Series Forecasting of Household Power Consumption
OCTOBER 10, 2018

How to Develop LSTM Models for Time Series Forecasting
NOVEMBER 14, 2018

11 Classical Time Series Forecasting Methods in Python (Cheat Sheet)
AUGUST 6, 2018



A Gentle Introduction to LSTM Autoencoders
NOVEMBER 5, 2018

A Gentle Introduction to k-fold Cross-Validation
MAY 23, 2018

How to Develop RNN Models for Human Activity Recognition Time Series Classification
SEPTEMBER 24, 2018

How to Develop Convolutional Neural Network Models for Time Series Forecasting
NOVEMBER 12, 2018

Top beginner tutorials:

- How to Install Python for Machine Learning
- Your First Machine Learning Project in Python
- Your First Neural Network in Python
- Your First Classifier in Weka
- Your First Time Series Forecasting Project

Top project tutorials:

- Photo Captioning Project
- Neural Machine Translation Project
- Sentiment Analysis Project
- Power Forecasting Project