Keras

Using the Keras Flatten Operation in CNN Models with Code Examples

This article explains how to use Keras to create a layer that flattens the output of convolutional neural network layers, in preparation for the fully connected layers that make a classification decision. Flattening is a key step in all Convolutional Neural Networks (CNN). If you're running multiple experiments in Keras, you can use MissingLink's deep learning platform to easily run, track, and manage all of your

In this article you will learn

X_test.reshape()
2. For class-based classification, one-hot encode the categories using to_categorical()
3. Build the model using the Sequential.add() function.
4. Add a convolutional layer, for example using Sequential.add(Conv2D(…)) – see our in-depth guide to Keras Conv2D layers.
5. Add a pooling layer, for example using the Sequential.add(MaxPooling2D(…)) function
6. Add a "flatten" layer which prepares a vector for the fully connected layers, for example using Sequential.add(Flatten()). *>> You are here. In this article, we explain the Keras flatten command, and the tf.layers.Flatten() function*
7. Add one or more fully connected layer using Sequential.add(Dense)), and if necessary a dropout layer.
8. Compile the model using model.compile()
9. Train the model using model.fit(), supplying X_train(),

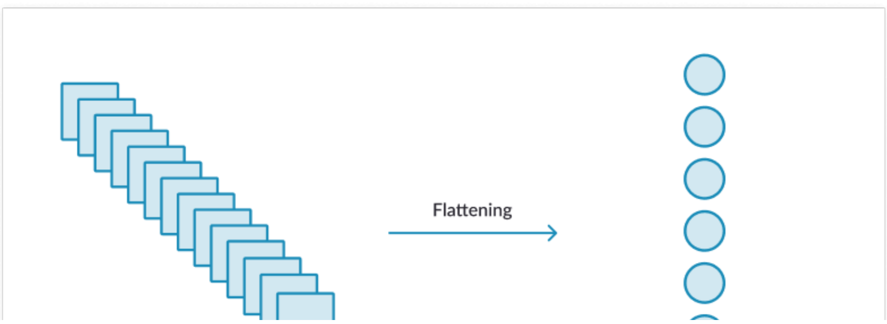
Role of the Flatten Layer in CNN Image Classification

A Convolutional Neural Network (CNN) architecture has three main parts:

- **A convolutional layer** that extracts features from a source image. Convolution helps with blurring, sharpening, edge detection, noise reduction, or other

- **A fully connected layer** also known as the dense layer, in which the results of the convolutional layers are fed through one or more neural layers to generate a prediction.

In between the convolutional layer and the fully connected layer, there is a 'Flatten' layer. Flattening transforms a two-dimensional matrix of features into a vector that can be fed into a fully connected neural network classifier.



Keras Flatten Examples and the tf.keras.layers.Flatten Class

In TensorFlow, you can perform the flatten operation using tf.keras.layers.Flatten() function.

```
keras.layers.Flatten(data_format=None)
```

Example 1: Flatten Operation Using Keras Sequential() Function

This example is based on a tutorial by Amal Nair. It shows how the flatten operation is performed as part of a model built using the Sequential() function which lets you sequentially add on layers to create your neural network model.

```
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
```

Initializing the network using the Sequential Class:

```
model = Sequential()
```



```
model.add(Convolution2D(filters = 32, kernel_size = (3, 3),
          input_shape = (64, 64, 3),
          activation = ‘relu’))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Convolution2D(32, 3, 3, activation = ‘relu’))
model.add(MaxPooling2D(pool_size = (2, 2)))
```

Flattening and adding two fully connected layers:

```
model.compile(optimiser = 'adam',
                        loss = 'binary_crossentropy',
                        metrics = ['accuracy'])
training_set =
train_datagen.flow_from_directory('dataset/training_set',
                        target_size = (64, 64),
                        batch_size = 32,
                        class_mode = 'binary')
test_set =
test_datagen.flow_from_directory('dataset/test_set',
                        target_size = (64, 64),
                        batch_size = 32,
                        class_mode = 'binary')
model.fit_generator(training_set,
                  samples_per_epoch = 2000,
                  nb_epoch = 15,
                  validation_data = test_set,
                  nb_val_samples = 200)
```

Example 2: Flatten Operation in a Simple CNN for

Jason Brownlee.

In this example, the model receives black and white 64×64 images as input, then has a sequence of two convolutional and pooling layers as feature extractors, followed by a flatten operation and a fully connected layer to interpret the features and an output layer with a sigmoid activation for two-class predictions. The flatten operation is highlighted.

```
from keras.utils import plot_model
from keras.models import Model
```

Input layer:

```
visible = Input(shape=(64,64,1))
```

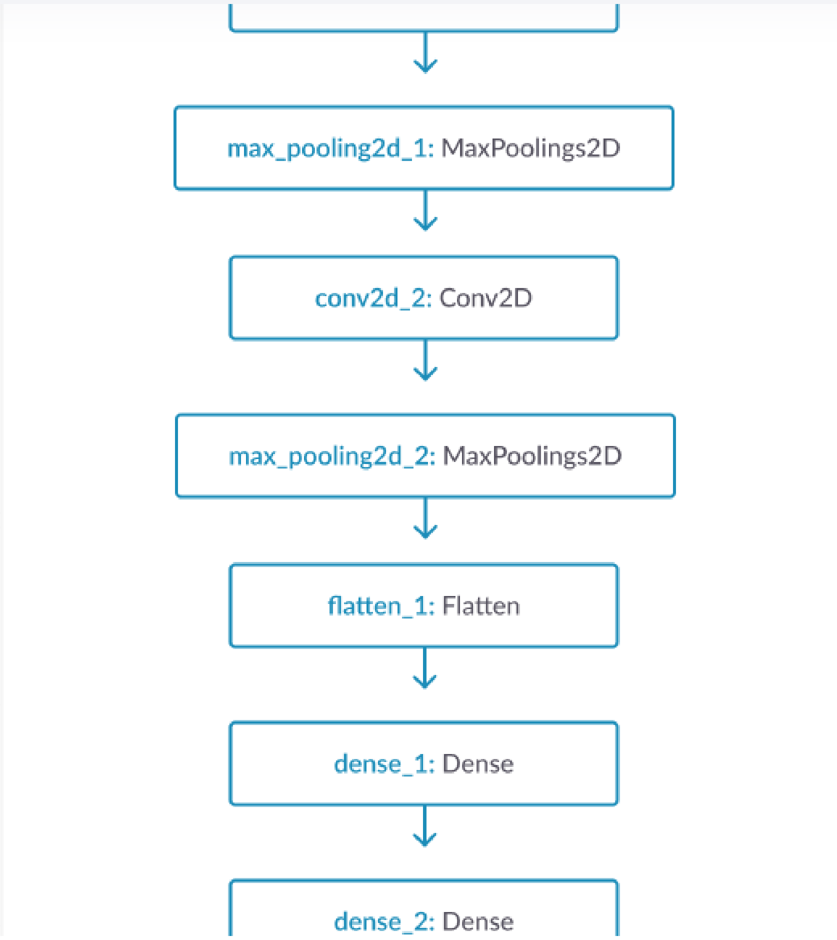
Convolution and pooling layers, with flatten operation performed after each one:

```
conv1 = Conv2D(32, kernel_size=4, activation='relu')(visible)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
conv2 = Conv2D(16, kernel_size=4, activation='relu')(pool1)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
flat = Flatten()(pool2)
```

Dense layer, prediction and displaying computational model:

```
hidden1 = Dense(10, activation='relu')(flat)
output = Dense(1, activation='sigmoid')(hidden1)
model = Model(inputs=visible, outputs=output)
```


A plot of the model graph is also created and saved to file.



Source: Machine Learning Mastery

Example 3: Flatten Operation in a CNN with a Shared Input Model

The model takes black and white images with size 64×64 pixels. There are two CNN feature extraction submodels that share this input. The first has a kernel size of 4 and the second a kernel size of 8.

```
from keras.layers import Input
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.pooling import MaxPooling2D
from keras.layers.merge import concatenate
```

Input layer:

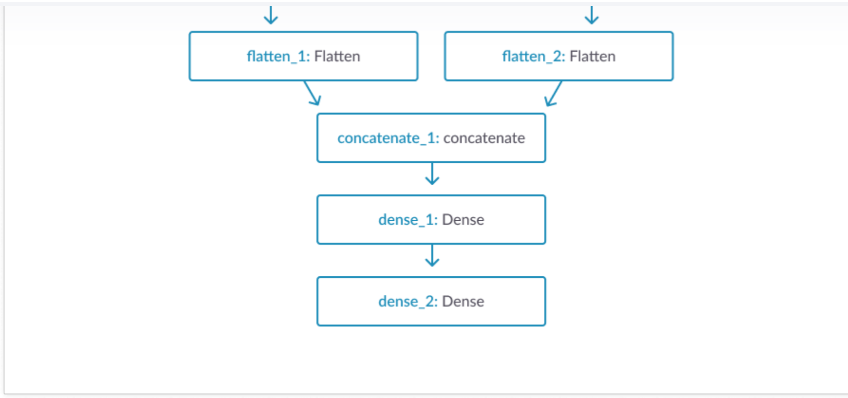
```
visible = Input(shape=(64,64,1))
```

Convolution and pooling layers, with flatten operation performed after each one:

```
conv1 = Conv2D(32, kernel_size=4, activation='relu')(visible)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
flat1 = Flatten()(pool1)
conv2 = Conv2D(16, kernel_size=8, activation='relu')(visible)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
```


Dense layer, prediction and displaying computational model:

```
hidden1 = Dense(10, activation='relu')(merge)
output = Dense(1, activation='sigmoid')(hidden1)
model = Model(inputs=visible, outputs=output)
print(model.summary())
plot_model(model, to_file='shared_input_layer.png')
```



Source: Machine Learning Mastery

Example 4: Flatten Operation in a CNN with a Multiple Input Model

This example shows an image classification model that takes two versions of the image as input, each of a different size. Specifically a black and white 64×64 version and a color 32×32 version. Separate feature extraction CNN models operate on each, then the results from both models are concatenated for interpretation and ultimate


```
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.pooling import MaxPooling2D
from keras.layers.merge import concatenate
```

Input layer, convolutions, pooling and flatten for first model:

```
visible1 = Input(shape=(64,64,1))
conv11 = Conv2D(32, kernel_size=4, activation='relu')
```

Input layer, convolutions, pooling and flatten for second model:

```
visible2 = Input(shape=(32,32,3))
conv21 = Conv2D(32, kernel_size=4, activation='relu')
(visible2)
pool21 = MaxPooling2D(pool_size=(2, 2))(conv21)
conv22 = Conv2D(16, kernel_size=4, activation='relu')(pool21)
pool22 = MaxPooling2D(pool_size=(2, 2))(conv22)
flat2 = Flatten()(pool22)
```

Merging the two models and applying fully connected layers:

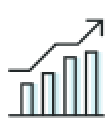
```
merge = concatenate([flat1, flat2])

hidden1 = Dense(10, activation='relu')(merge)
hidden2 = Dense(10, activation='relu')(hidden1)
output = Dense(1, activation='sigmoid')(hidden2)
model = Model(inputs=[visible1, visible2], outputs=output)
```

Source: Machine Learning Mastery

Running CNN at Scale on Keras with MissingLink

you'll run into some practical challenges:

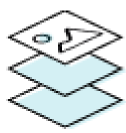


Tracking metrics and hyperparameters

The more experiments you run, the more difficult it will be to track what you ran, what colleagues on your team are running,

Running experiments on multiple machines

Computer vision deep learning projects are computationally intensive and models can take hours or even days or weeks to run. You will need to run CNNs on multiple GPUs and multiple machines; setting up these machines and distributing the work can be a burden.



Manage deep learning data

CNN projects with images or video can have very large training, evaluation and testing datasets. It's a hassle to copy data to each training machine, especially if it's in the cloud, figuring out which version of the data is on each machine, and managing updates.

for you, and lets you concentrate on building the most accurate model. Learn more to see how easy it is.

Learn More About Keras

Keras Conv2D: Working with CNN 2D Convolutions in Keras

Keras ResNet: Building, Training & Scaling Residual Nets on Keras missinglink.ai

Convolutional Neural Network: How to Build One in Keras & PyTorch

Resources

Blog

Product Features

Docs

Company

About us

missinglink.ai

Solutions

Deep Learning on AWS

Deep Learning on Google Cloud

Deep Learning on Azure

TensorFlow Hyperparameter Tuning

TensorFlow Visualization for Teams