Need help with LSTMs in Python? Take the FREE Mini-Course.

CNN Long Short-Term Memory Networks

by **Jason Brownlee** on August 21, 2017 in **Long Short-Term Memory Networks**

Gentle introduction to CNN LSTM recurrent neural networks with example Python code.

Input with spatial structure, like images, cannot be modeled easily with the standard Vanilla LSTM.

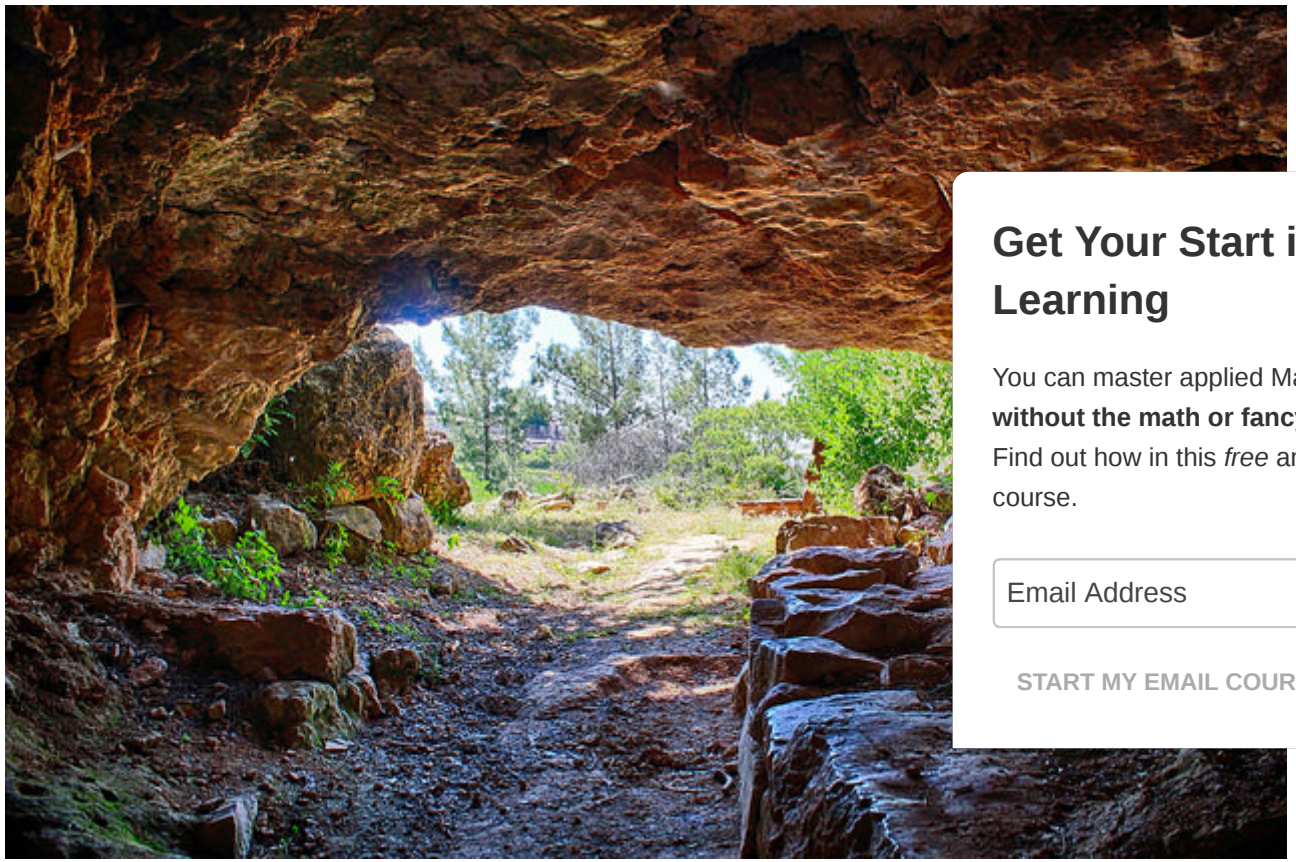
The CNN Long Short-Term Memory Network or CNN LSTM for short is an LSTM architecture specifically designed for sequence prediction problems with spatial inputs, like images or videos.

In this post, you will discover the CNN LSTM architecture for sequence prediction.

After completing this post, you will know:

- About the development of the CNN LSTM model architecture for sequence prediction.
- Examples of the types of problems to which the CNN LSTM model is suited.
- How to implement the CNN LSTM architecture in Python with Keras.

Let's get started.



Convolutional Neural Network Long Short-Term Memory Networks
Photo by Yair Aronshtam, some righs reserved.

CNN LSTM Architecture

Get Your Start in Machine Learning

The CNN LSTM architecture involves using Convolutional Neural Network (CNN) layers for feature extraction on input data combined with LSTMs to support sequence prediction.

CNN LSTMs were developed for visual time series prediction problems and the application of generating textual descriptions from sequences of images (e.g. videos). Specifically, the problems of:

- **Activity Recognition**: Generating a textual description of an activity demonstrated in a sequence of images.
- **Image Description**: Generating a textual description of a single image.
- **Video Description**: Generating a textual description of a sequence of images.

> *[CNN LSTMs are] a class of models that is both spatially and temporally deep, and has the flexibility to be applied to a variety of vision tasks involving sequential inputs and outputs*

— Long-term Recurrent Convolutional Networks for Visual Recognition and Description, 2015.

This architecture was originally referred to as a Long-term Recurrent Convolutional Network or LRCN, although we will use the more general name "CNN LSTM" to refer to LSTMs that use a CNN as a front end in this lesson.

This architecture is used for the task of generating textual descriptions of images. Key is the use of a CNN trained on an image classification task that is re-purposed as a feature extractor for the caption generating problem.

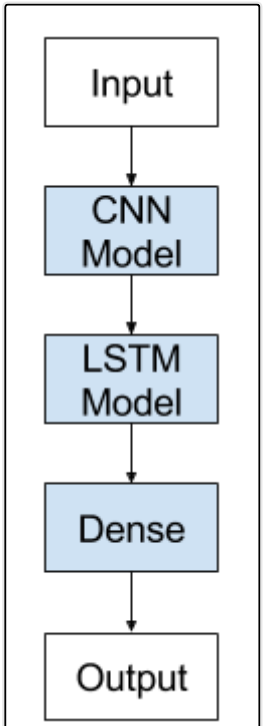
> *… it is natural to use a CNN as an image "encoder", by first pre-training it for an image classi… input to the RNN decoder that generates sentences*

— Show and Tell: A Neural Image Caption Generator, 2015.

This architecture has also been used on speech recognition and natural language processing problems where CNNs are used as feature extractors for the LSTMs on audio and textual input data.

This architecture is appropriate for problems that:

- Have spatial structure in their input such as the 2D structure or pixels in an image or the 1D structure of words in a sentence, paragraph, or document.
- Have a temporal structure in their input such as the order of images in a video or words in text, or require the generation of output with temporal structure such as words in a textual description.



Convolutional Neural Network Long Short-Term Memory Network Architecture

Need help with LSTMs for Sequence Prediction?

Take my free 7-day email course and discover 6 different LSTM architectures (with sample code).

Click to sign-up and also get a free PDF Ebook version of

Get Your Start in Machine Learning

Implement CNN LSTM in Keras

We can define a CNN LSTM model to be trained jointly in Keras.

A CNN LSTM can be defined by adding CNN layers on the front end followed by LSTM layers with a Dense layer on the output.

It is helpful to think of this architecture as defining two sub-models: the CNN Model for feature extraction and the LSTM Model for interpreting the features across time steps.

Let's take a look at both of these sub models in the context of a sequence of 2D inputs which we will

CNN Model

As a refresher, we can define a 2D convolutional network as comprised of Conv2D and MaxPooling2 h.

The Conv2D will interpret snapshots of the image (e.g. small squares) and the polling layers will con

For example, the snippet below expects to read in 10×10 pixel images with 1 channel (e.g. black and snapshots and output one new 10×10 interpretation of the image. The MaxPooling2D will pool the in a 5×5 consolidation. The Flatten layer will take the single 5×5 map and transform it into a 25-element v h as a Dense for outputting a prediction.

```
cnn = Sequential()
cnn.add(Conv2D(1, (2,2), activation='relu', padding='same', input_shape=(10,10,1)))
cnn.add(MaxPooling2D(pool_size=(2, 2)))
cnn.add(Flatten())
```

This makes sense for image classification and other computer vision tasks.

LSTM Model

The CNN model above is only capable of handling a single image, transforming it from input pixels into an internal matrix or vector representation.

We need to repeat this operation across multiple images and allow the LSTM to build up internal state and update weights using BPTT across a sequence of the internal vector representations of input images.

The CNN could be fixed in the case of using an existing pre-trained model like VGG for feature extraction from images. The CNN may not be trained, and we may wish to train it by backpropagating error from the LSTM across multiple input images to the CNN model.

In both of these cases, conceptually there is a single CNN model and a sequence of LSTM models, one for each time step. We want to apply the CNN model to each input image and pass on the output of each input image to the LSTM as a single time step.

We can achieve this by wrapping the entire CNN input model (one layer or more) in a TimeDistributed layer. This layer achieves the desired outcome of applying the same layer or layers multiple times. In this case, applying it multiple times to multiple inp
"image interpretations" or "image features" to the LSTM model to work on.

```
model.add(TimeDistributed(...))
model.add(LSTM(...))
model.add(Dense(...))
```

We now have the two elements of the model; let's put them together.

CNN LSTM Model

We can define a CNN LSTM model in Keras by first defining the CNN layer or layers, wrapping them
LSTM and output layers.

We have two ways to define the model that are equivalent and only differ as a matter of taste.

You can define the CNN model first, then add it to the LSTM model by wrapping the entire sequence of CNN layers in a TimeDistributed layer, as follows:

```
# define CNN model
cnn = Sequential()
cnn.add(Conv2D(...))
cnn.add(MaxPooling2D(...))
cnn.add(Flatten())
# define LSTM model
model = Sequential()
model.add(TimeDistributed(cnn, ...))
```

```
model.add(LSTM(..))
model.add(Dense(...))
```

An alternate, and perhaps easier to read, approach is to wrap each layer in the CNN model in a TimeDistributed layer when adding it to the main model.

```
model = Sequential()
# define CNN model
model.add(TimeDistributed(Conv2D(...))
model.add(TimeDistributed(MaxPooling2D(...)))
model.add(TimeDistributed(Flatten()))
# define LSTM model
model.add(LSTM(...))
model.add(Dense(...))
```

The benefit of this second approach is that all of the layers appear in the model summary and as such is preferred for now.

You can choose the method that you prefer.

Further Reading

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

Papers on CNN LSTM

- Long-term Recurrent Convolutional Networks for Visual Recognition and Description, 2015.
- Show and Tell: A Neural Image Caption Generator, 2015.
- Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks, 2015.
- Character-Aware Neural Language Models, 2015.
- Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting, 2015

Keras API

- Conv2D Keras API.
- MaxPooling2D Keras API.
- Flatten Keras API.
- TimeDistributed Keras API.

Posts

- Crash Course in Convolutional Neural Networks for Machine Learning
- Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras

Summary

In this post, you discovered the CNN LSTN model architecture.

Specifically, you learned:

- About the development of the CNN LSTM model architecture for sequence prediction.
- Examples of the types of problems to which the CNN LSTM model is suited.
- How to implement the CNN LSTM architecture in Python with Keras.

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

About Jason Brownlee

Dr. Jason Brownlee is a husband, proud father, academic researcher, author, professional develo to helping developers get started and get good at applied machine learning. Learn more.

View all posts by Jason Brownlee →

‹ Stacked Long Short-Term Memory Networks

Encoder-Decoder Long Short-Term Memory Networks ›

24 Responses to *CNN Long Short-Term Memory Networks*

Erick August 21, 2017 at 6:07 pm #

REPLY

Would this architecture, with some adaptations, also be suitable to do speech recognition, speaker separation, language detection and other natural language processing tasks?

Jason Brownlee August 22, 2017 at 6:37 am #

Perhaps.

I have seen it most used for document classification / sentiment analysis in NLP.

Dan Lim August 25, 2017 at 2:45 pm #

cnn + lstm architecture used for speech recognition – https://arxiv.org/pdf/1610.03022.pdf

Jason Brownlee August 25, 2017 at 3:58 pm #

REPLY

Thanks for sharing Dan.

birol August 22, 2017 at 6:27 pm #

REPLY

what is difference with ConvLSTM2D layer ?
https://github.com/fchollet/keras/blob/master/examples/conv_lstm.py

Jason Brownlee August 23, 2017 at 6:45 am #

REPLY

As far as I know, that layer is not yet supported. I have tried to stay away from it until all the bugs are worked out of it.

Dan Lim August 25, 2017 at 2:59 pm #

REPLY

ConvLSTM is variant of LSTM which use convolution to replace inner procut within LSTM u
while CNN LSTM is just stack of layer; CNN followed by LSTM.

Jason Brownlee August 25, 2017 at 3:58 pm #

Have you used it on a project Dan?

Dan Lim August 30, 2017 at 2:09 pm #

Not yet, I'm just waiting next tensorflow release since it seems that convlstm would
I've used cnn + lstm on simple speech recognition experiments and it gives better results than stack of lstm. It really works!

Jason Brownlee August 30, 2017 at 4:18 pm #

Thanks Dan.

I hope to try some examples myself for the blog soon.

Miles August 25, 2017 at 7:13 am #

REPLY

Hi, Jason.
Do you think the CNNLSTM can solve the regression problem, whose inputs are some time series data and some properties/exogenous data (spatial), not image data? If yes, how to deal with the properties/exogenous data (2D) in CNN. Thank you.

Tahir August 25, 2017 at 2:24 pm #

REPLY

I m having the same question

Jason Brownlee August 25, 2017 at 3:56 pm #

Perhaps, I have not tried using CNN LSTMs for time series.

Perhaps each series could be processed by a 1D-CNN.

It might not make sense given that the LSTM is already interpreting the long term relationships in the

It might be interesting if the CNN can pick out structure that is new/different from the LSTM. Perhaps
of the series and use another model to integrate and interpret the results.

Shamane Siriwardana September 28, 2017 at 1:27 pm #

REPLY

Hi do you have a github implementation ?

Get Your Start in Machine Learning

Jason Brownlee September 28, 2017 at 4:45 pm #

REPLY

I have a full code example in my book on LSTMs.

Elisa October 6, 2017 at 1:10 pm #

REPLY

Hi Jason,
Thank you for the great work and posts.

I'm starting my studies with deep learning, python and keras.
I would like knowing how to implement the CNN with ELM (extreme learning machine) architecture in Py
github implementation?

Jason Brownlee October 7, 2017 at 5:46 am #

Sorry, I do not.

gana October 12, 2017 at 4:16 pm #

Thank you for your great examples…

May i ask you full code of the CNN LSTM you explained above?
Because,..i am having errors related to dimensions of CNN and LSTM.

I have followed your previous examples and trying to build VGG-16Net stacked with LSTM.

My database is just 10 different human motion (10 classes) such as walking and running etc…

My code is as below:

```
# dimensions of our images.
img_width, img_height = 224, 224

train_data_dir = 'db/train'
validation_data_dir = 'db/test'
nb_train_samples = 400
nb_validation_samples = 200
num_timesteps = 10 # length of sequence
num_class = 10
epochs = 10
batch_size = 8

lstm_input_len = 224 * 224
input_shape=(224,224,3)
num_chan = 3

# VGG16 as CNN
cnn = Sequential()
cnn.add(ZeroPadding2D((1,1),input_shape=input_shape))
cnn.add(Conv2D(64, 3, 3, activation='relu'))
cnn.add(ZeroPadding2D((1,1)))
cnn.add(Conv2D(64, 3, 3, activation='relu'))
cnn.add(MaxPooling2D((2,2), strides=(2,2),dim_ordering="th"))

cnn.add(ZeroPadding2D((1,1)))
cnn.add(Conv2D(128, 3, 3, activation='relu'))
cnn.add(ZeroPadding2D((1,1)))
cnn.add(Conv2D(128, 3, 3, activation='relu'))
cnn.add(MaxPooling2D((2,2), strides=(2,2),dim_ordering="th"))

cnn.add(ZeroPadding2D((1,1)))
cnn.add(Conv2D(256, 3, 3, activation='relu'))
cnn.add(ZeroPadding2D((1,1)))
cnn.add(Conv2D(256, 3, 3, activation='relu'))
cnn.add(ZeroPadding2D((1,1)))
```

```
cnn.add(Conv2D(256, 3, 3, activation='relu'))
cnn.add(MaxPooling2D((2,2), strides=(2,2),dim_ordering="th"))

cnn.add(ZeroPadding2D((1,1)))
cnn.add(Conv2D(512, 3, 3, activation='relu'))
cnn.add(ZeroPadding2D((1,1)))
cnn.add(Conv2D(512, 3, 3, activation='relu'))
cnn.add(ZeroPadding2D((1,1)))
cnn.add(Conv2D(512, 3, 3, activation='relu'))
cnn.add(MaxPooling2D((2,2), strides=(2,2),dim_ordering="th"))

cnn.add(ZeroPadding2D((1,1)))
cnn.add(Conv2D(512, 3, 3, activation='relu'))
cnn.add(ZeroPadding2D((1,1)))
cnn.add(Conv2D(512, 3, 3, activation='relu'))
cnn.add(ZeroPadding2D((1,1)))
cnn.add(Conv2D(512, 3, 3, activation='relu'))
cnn.add(MaxPooling2D((2,2), strides=(2,2),dim_ordering="th"))

cnn.add(Flatten())
cnn.add(Dense(4096, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(4096, activation='relu'))

#LSTM
model = Sequential()
model.add(TimeDistributed(cnn, input_shape=(num_timesteps, 224, 224,num_chan)))
model.add(LSTM(num_timesteps))
model.add(Dropout(.2)) #added
model.add(Dense(num_class, activation='softmax'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(rescale=1. / 255)
```

```
# this is the augmentation configuration we will use for testing:
# only rescaling
test_datagen = ImageDataGenerator(rescale=1. / 255)

train_generator = train_datagen.flow_from_directory(
train_data_dir,
target_size=(224, 224),
batch_size=batch_size,
class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
validation_data_dir,
target_size=(224, 224),
batch_size=batch_size,
class_mode='binary')

model.fit_generator(
train_generator,
steps_per_epoch=nb_train_samples // batch_size,
epochs=epochs,
validation_data=validation_generator,
validation_steps=nb_validation_samples // batch_size)
```


gana October 12, 2017 at 4:18 pm #

I forgot to put error which is :

ValueError: Error when checking input: expected time_distributed_9_input to have 5 dimensions, but got array with shape (8, 224, 224, 3)

Jason Brownlee October 13, 2017 at 5:43 am #

REPLY

Sorry, I cannot debug your code. I list some places to get help with code here:
https://machinelearningmastery.com/get-help-with-keras/

Long October 15, 2017 at 12:33 pm #

REPLY

Hi Jason,

Assuming there are a data set with time series data (e.g temperature, rainfall) and geographic data(e.g. elevation, slope) for many grid positions, I need to use the data set to predict(regression) future weathers.

I think of a method with LSTM (for time series data) + auxiliary (geographic data) to be a solution. But the results of forecast is not very good. Do you have other better methods? Or do you have a related lessons?

Thank you very much.

Jason Brownlee October 16, 2017 at 5:41 am #

Perhaps a deep MLP with a window over lag obs.

Long October 16, 2017 at 8:30 pm #

Hi Jason,

Could you please explain it in detail? Thank you very much.

Jason Brownlee October 17, 2017 at 5:43 am #

REPLY

This function will help you reshape your time series data to be a supervised learning problem:
https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-pytho

You can then use a neural network model.

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Welcome to Machine Learning Mastery

Hi, I'm Dr. Jason Brownlee.
My goal is to make practitioners like YOU awesome at applied machine learning.

Read More

Get Your Start in Machine Learning

Your First Machine Learning Project in Python Step-By-Step
JUNE 10, 2016

Develop Your First Neural Network in Python With Keras Step-By-Step
MAY 24, 2016

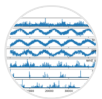
Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras
JULY 26, 2016

How to Setup a Python Environment for Machine Learning and Deep Learning with Anaconda
MARCH 13, 2017

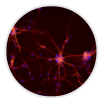
Time Series Forecasting with the Long Short-Term Memory Network in Python
APRIL 7, 2017

Multi-Class Classification Tutorial with the Keras Deep Learning Library
JUNE 2, 2016

Regression Tutorial with the Keras Deep Learning Library in Python
JUNE 9, 2016



Multivariate Time Series Forecasting with LSTMs in Keras
AUGUST 14, 2017



How to Implement the Backpropagation Algorithm From Scratch In Python
NOVEMBER 7, 2016

Get Your Start in Machine Learning

Get Your Start in Machine Learning