Get unlimited access to the best of Medium for less than $1/week. Become a member

TDS Archive · Follow publication

Building MobileNet from Scratch Using TensorFlow

Creating the MobileNet architecture from scratch in TensorFlow

6 min read · Sep 1, 2021

Arjun Sarkar Follow

Listen Share More

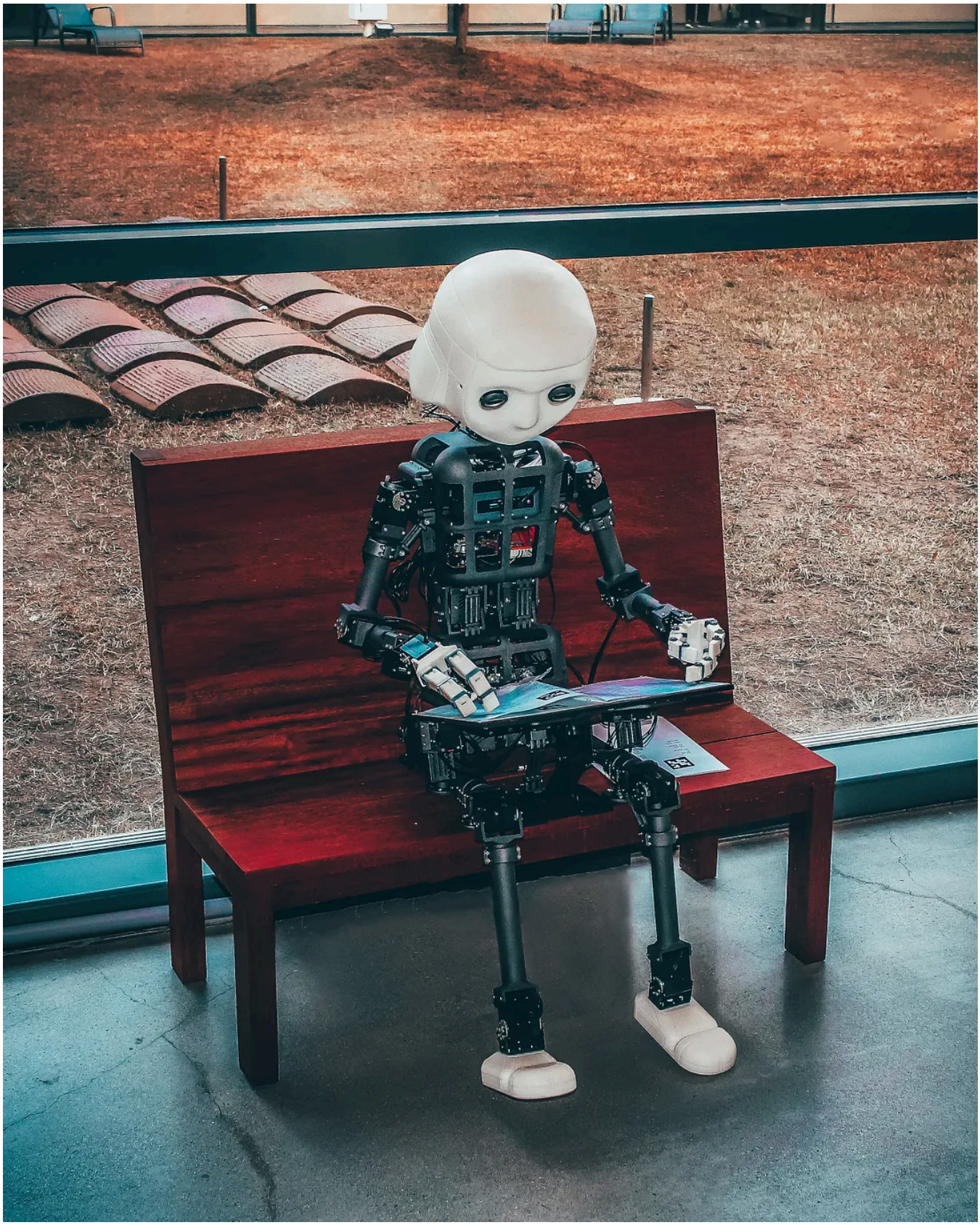


Figure 1. (Source: Photo by Andrea De Santis on Unsplash)

Previously I have discussed the architecture of MobileNet and its most important layer "***Depthwise Separable Convolutions***" in the story — Understanding Depthwise Separable Convolutions and the efficiency of MobileNets.

Next, we will see how to implement this architecture from scratch using TensorFlow.

Implementation:

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 2. MobileNet architecture (Source: Image from the original paper)

Figure 2 shows the MobileNet architecture that we will implement in code. The network starts with Vonv, BatchNorm, ReLU block, and follows multiple MobileNet

blocks from thereon. It finally ends with an Average Pooling and a Fully connected layer, with a Softmax activation.

We see the architecture has the pattern — Conv dw/s1, followed by Conv/s1, and so on. Here dw is the depthwise layer with the number of strides, followed by the Conv layer with the number of strides. These two lines are the MobileNet block.

The 'Filter Shape' column gives the details about the kernel size and the number of filters to be used. The last number of the column gives the number of filters. We see the filter number gradually increase from 32 to 64, 64 to 128, 128 to 256, and so on.

The last column shows how the size of the image changes as we go deeper into the network. The input size is chosen as 224*224 pixels, with 3 channels and the output layer classifies 1000 classes.

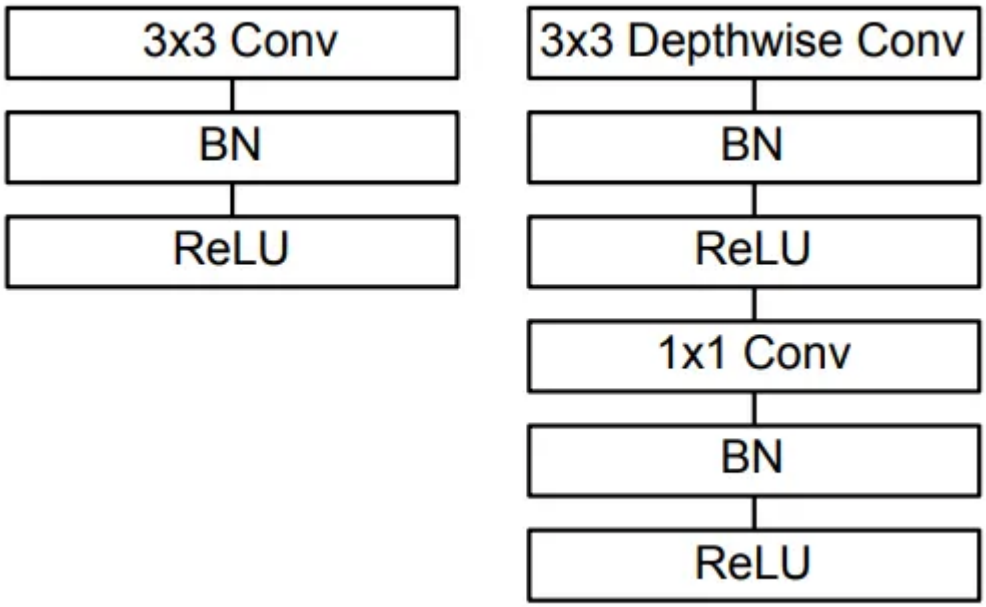


Figure 3. Difference between blocks of normal CNN architectures — left, vs MobileNet architecture — right (Source: Image from the original paper)

Few things to keep in mind while constructing the network:

1. All layers are followed by a Batch Normalization and a ReLU nonlinearity.
2. Unlike normal CNN models which have a Conv2D layer, MobileNet's have Depthwise Conv layers, as seen in Figure 3. To understand this layer better please refer to — Depthwise Convolutional Blocks.

Workflow:

1. Import all the necessary layers from the TensorFlow library
2. Writing a helper function for the MobileNet block
3. Building the stem of the model
4. Use the helper function to build the main part of the model

Importing the layers

```
import tensorflow as tf

#import all necessary layers

from tensorflow.keras.layers import Input, DepthwiseConv2D
from tensorflow.keras.layers import Conv2D, BatchNormalization
from tensorflow.keras.layers import ReLU, AvgPool2D, Flatten, Dense

from tensorflow.keras import Model
```

Keras has a DepthwiseConv layer already built-in, so we do not need to create it from scratch.

MobileNet block

Type / Stride	Filter Shape
Conv dw/s**1**	3x3x32 dw
Conv/s1	1x1x32x**64**

Figure 4. Representation of a MobileNet block (Source: image from the original paper)

For creating the function for the MobileNet block, we need the following steps:

1. Input to the function:

a. A tensor (x)

b. the number of filters for the convolutional layer (filters)

c. the strides for the Depthwise convolutional layer (strides)

2. Run (Figure 3 — right side image):

a. applying a 3x3 Dethwise convolutional layer with strides followed by a Batch Normalization layer and a ReLU activation

b. Applying a 1x1 Convolutional layer with filters followed by a batch normalization layer and a ReLU activation

3. Return the tensor (output)

These 3 steps are implemented in the code blocks below.

```
# MobileNet block

def mobilnet_block (x, filters, strides):

    x = DepthwiseConv2D(kernel_size = 3, strides = strides, padding
= 'same')(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)

    x = Conv2D(filters = filters, kernel_size = 1, strides = 1)(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)

    return x
```

Building the stem of the Model

As seen in Figure 2, the first layer is Conv/s2 with filter shape 3x3x3x32.

Type / Stride	Filter Shape
Conv/s2	3x3x3x32

Figure 5. The stem of the model (Source: image from the original paper)

```
#stem of the model

input = Input(shape = (224,224,3))

x = Conv2D(filters = 32, kernel_size = 3, strides = 2, padding =
'same')(input)
x = BatchNormalization()(x)
x = ReLU()(x)
```

The main part of the Model

Conv dw / s1	$3 \times 3 \times 32$ dw
Conv / s1	$1 \times 1 \times 32 \times 64$
Conv dw / s2	$3 \times 3 \times 64$ dw
Conv / s1	$1 \times 1 \times 64 \times 128$
Conv dw / s1	$3 \times 3 \times 128$ dw
Conv / s1	$1 \times 1 \times 128 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw
Conv / s1	$1 \times 1 \times 128 \times 256$
Conv dw / s1	$3 \times 3 \times 256$ dw
Conv / s1	$1 \times 1 \times 256 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw
Conv / s1	$1 \times 1 \times 256 \times 512$
$5\times$ Conv dw / s1 Conv / s1	$3 \times 3 \times 512$ dw $1 \times 1 \times 512 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw
Conv / s1	$1 \times 1 \times 512 \times 1024$
Conv dw / s2	$3 \times 3 \times 1024$ dw
Conv / s1	$1 \times 1 \times 1024 \times 1024$
Avg Pool / s1	Pool 7×7
FC / s1	1024×1000
Softmax / s1	Classifier

Figure 6. The main part of the model (Source: image from the original paper)

```
# main part of the model

x = mobilnet_block(x, filters = 64, strides = 1)
x = mobilnet_block(x, filters = 128, strides = 2)
x = mobilnet_block(x, filters = 128, strides = 1)
x = mobilnet_block(x, filters = 256, strides = 2)
x = mobilnet_block(x, filters = 256, strides = 1)
x = mobilnet_block(x, filters = 512, strides = 2)

for _ in range (5):
     x = mobilnet_block(x, filters = 512, strides = 1)

x = mobilnet_block(x, filters = 1024, strides = 2)
x = mobilnet_block(x, filters = 1024, strides = 1)
```

```
x = AvgPool2D (pool_size = 7, strides = 1,
data_format='channels_first')(x)
output = Dense (units = 1000, activation = 'softmax')(x)

model = Model(inputs=input, outputs=output)
model.summary()
```

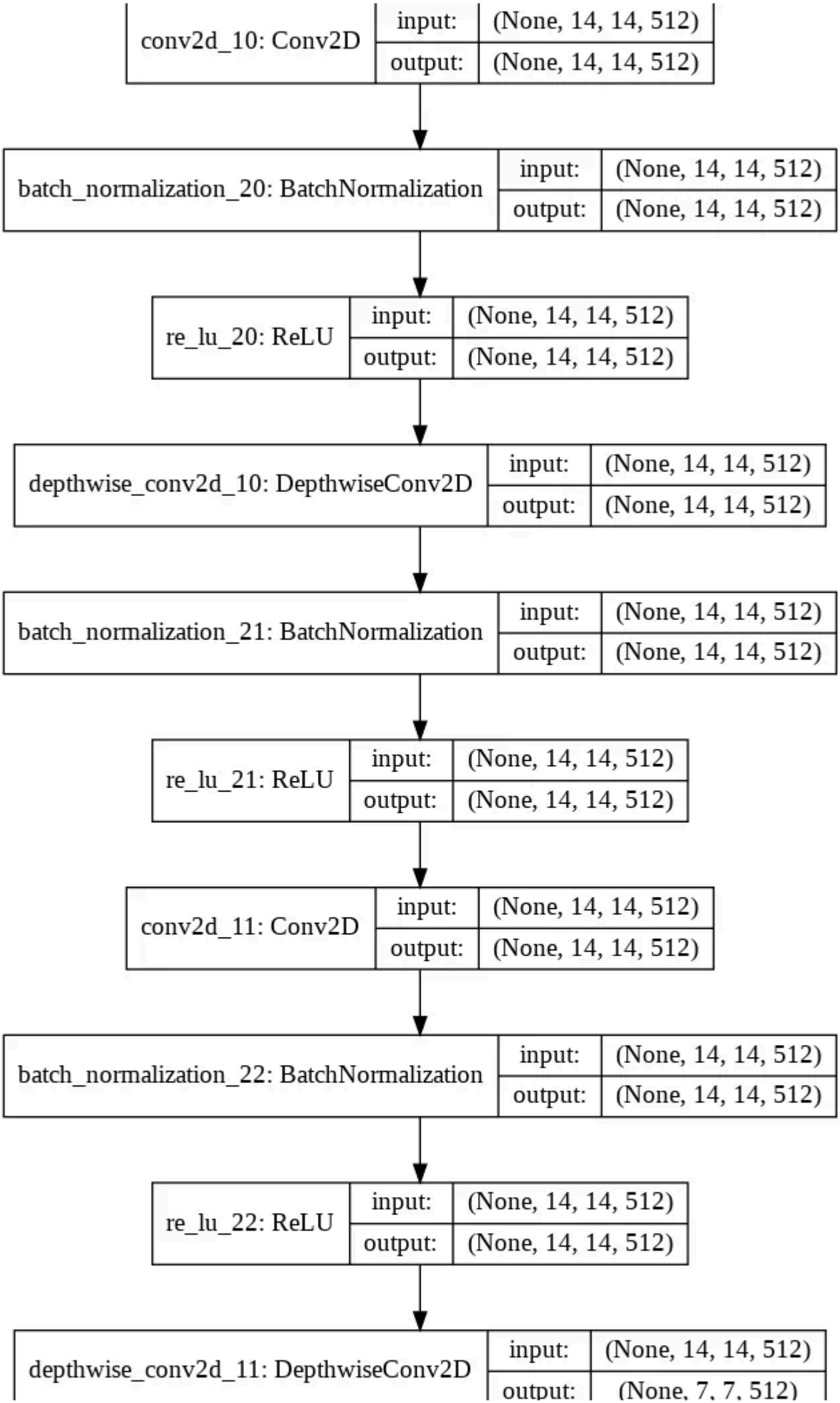
```
re_lu_19 (ReLU)              (None, 14, 14, 512)       0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 14, 14, 512)       262656
_________________________________________________________________
batch_normalization_20 (Batc (None, 14, 14, 512)       2048
_________________________________________________________________
re_lu_20 (ReLU)              (None, 14, 14, 512)       0
_________________________________________________________________
depthwise_conv2d_10 (Depthwi (None, 14, 14, 512)       5120
_________________________________________________________________
batch_normalization_21 (Batc (None, 14, 14, 512)       2048
_________________________________________________________________
re_lu_21 (ReLU)              (None, 14, 14, 512)       0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 14, 14, 512)       262656
_________________________________________________________________
batch_normalization_22 (Batc (None, 14, 14, 512)       2048
_________________________________________________________________
re_lu_22 (ReLU)              (None, 14, 14, 512)       0
_________________________________________________________________
depthwise_conv2d_11 (Depthwi (None, 7, 7, 512)         5120
_________________________________________________________________
batch_normalization_23 (Batc (None, 7, 7, 512)         2048
_________________________________________________________________
re_lu_23 (ReLU)              (None, 7, 7, 512)         0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 7, 7, 1024)        525312
_________________________________________________________________
batch_normalization_24 (Batc (None, 7, 7, 1024)        4096
_________________________________________________________________
re_lu_24 (ReLU)              (None, 7, 7, 1024)        0
_________________________________________________________________
depthwise_conv2d_12 (Depthwi (None, 7, 7, 1024)        10240
_________________________________________________________________
batch_normalization_25 (Batc (None, 7, 7, 1024)        4096
_________________________________________________________________
re_lu_25 (ReLU)              (None, 7, 7, 1024)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 7, 7, 1024)        1049600
_________________________________________________________________
batch_normalization_26 (Batc (None, 7, 7, 1024)        4096
_________________________________________________________________
re_lu_26 (ReLU)              (None, 7, 7, 1024)        0
_________________________________________________________________
average_pooling2d (AveragePo (None, 7, 1, 1018)        0
_________________________________________________________________
dense (Dense)                (None, 7, 1, 1000)        1019000
=================================================================
Total params: 4,258,808
Trainable params: 4,236,920
Non-trainable params: 21,888
_________________________________________________________________
```

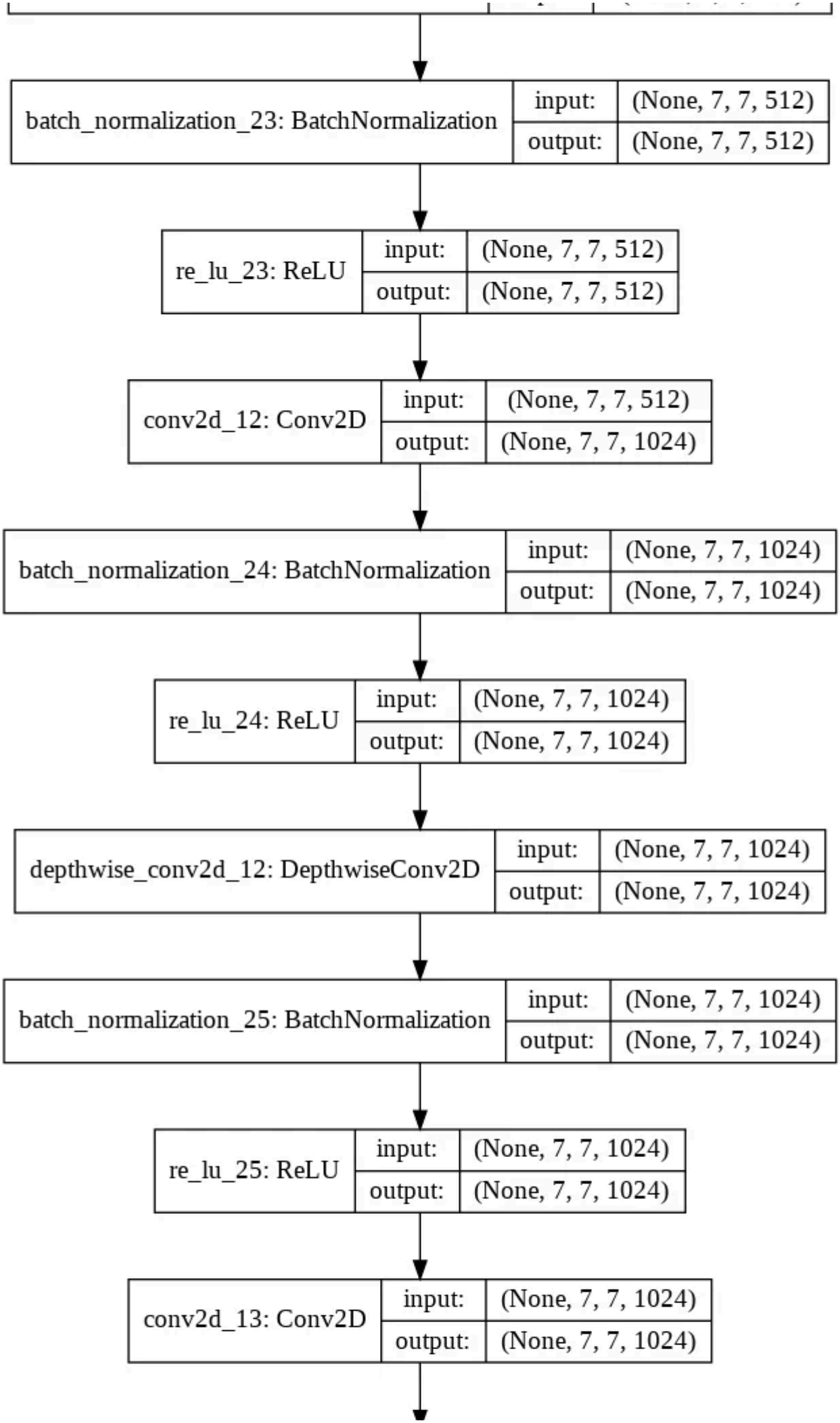
Figure 7. A snippet of the model summary

Plotting the Model

```
#plot the model

tf.keras.utils.plot_model(model, to_file='model.png',
show_shapes=True, show_dtype=False,show_layer_names=True,
rankdir='TB', expand_nested=False, dpi=96)
```


conv2d_10: Conv2D
input: (None, 14, 14, 512)
output: (None, 14, 14, 512)
batch_normalization_20: BatchNormalization
input: (None, 14, 14, 512)
output: (None, 14, 14, 512)
re_lu_20: ReLU
input: (None, 14, 14, 512)
output: (None, 14, 14, 512)
depthwise_conv2d_10: DepthwiseConv2D
input: (None, 14, 14, 512)
output: (None, 14, 14, 512)
batch_normalization_21: BatchNormalization
input: (None, 14, 14, 512)
output: (None, 14, 14, 512)
re_lu_21: ReLU
input: (None, 14, 14, 512)
output: (None, 14, 14, 512)
conv2d_11: Conv2D
input: (None, 14, 14, 512)
output: (None, 14, 14, 512)
batch_normalization_22: BatchNormalization
input: (None, 14, 14, 512)
output: (None, 14, 14, 512)
re_lu_22: ReLU
input: (None, 14, 14, 512)
output: (None, 14, 14, 512)
depthwise_conv2d_11: DepthwiseConv2D
input: (None, 14, 14, 512)
output: (None, 7, 7, 512)


batch_normalization_23: BatchNormalization
input: (None, 7, 7, 512)
output: (None, 7, 7, 512)
re_lu_23: ReLU
input: (None, 7, 7, 512)
output: (None, 7, 7, 512)
conv2d_12: Conv2D
input: (None, 7, 7, 512)
output: (None, 7, 7, 1024)
batch_normalization_24: BatchNormalization
input: (None, 7, 7, 1024)
output: (None, 7, 7, 1024)
re_lu_24: ReLU
input: (None, 7, 7, 1024)
output: (None, 7, 7, 1024)
depthwise_conv2d_12: DepthwiseConv2D
input: (None, 7, 7, 1024)
output: (None, 7, 7, 1024)
batch_normalization_25: BatchNormalization
input: (None, 7, 7, 1024)
output: (None, 7, 7, 1024)
re_lu_25: ReLU
input: (None, 7, 7, 1024)
output: (None, 7, 7, 1024)
conv2d_13: Conv2D
input: (None, 7, 7, 1024)
output: (None, 7, 7, 1024)

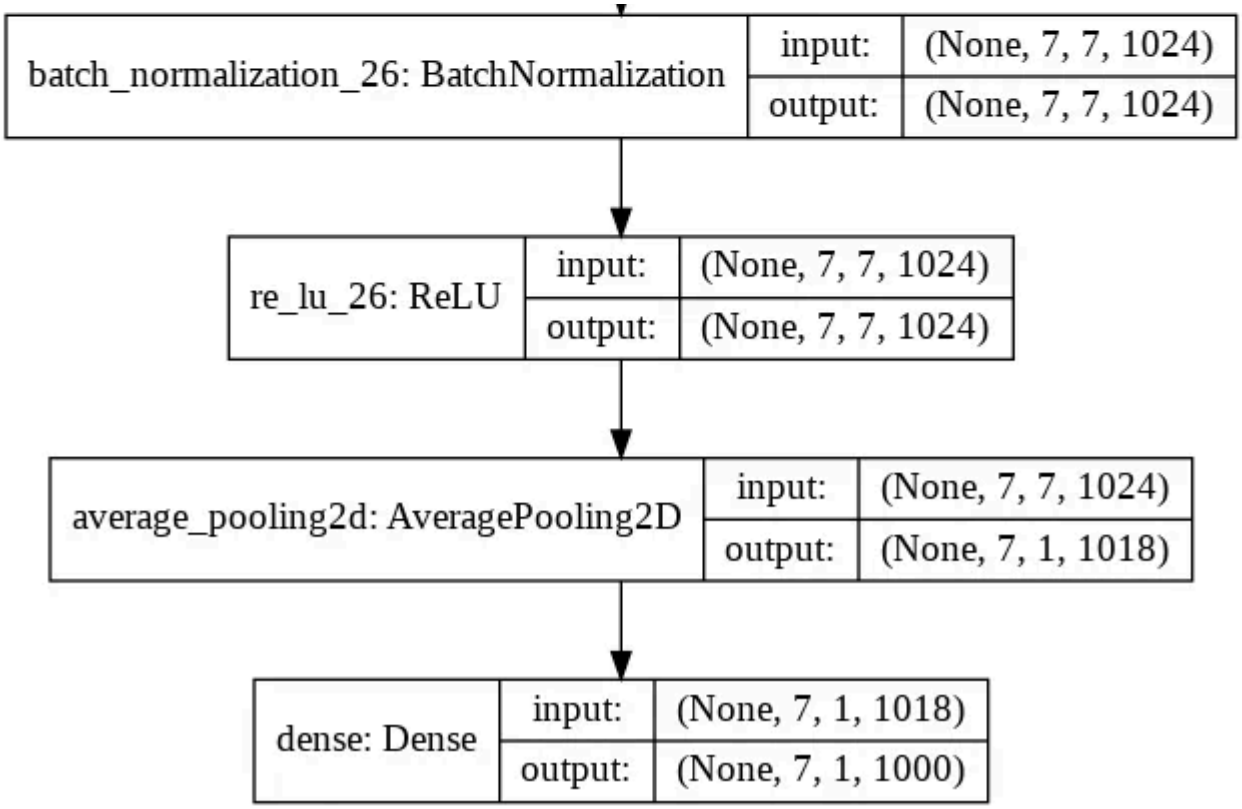


Figure 8: A snippet of the model plot

The entire MobileNet Model implementation using TensorFlow:

```
import tensorflow as tf

#import all necessary layers

from tensorflow.keras.layers import Input, DepthwiseConv2D
from tensorflow.keras.layers import Conv2D, BatchNormalization
from tensorflow.keras.layers import ReLU, AvgPool2D, Flatten, Dense

from tensorflow.keras import Model

# MobileNet block

def mobilnet_block (x, filters, strides):

    x = DepthwiseConv2D(kernel_size = 3, strides = strides, padding
= 'same')(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)

    x = Conv2D(filters = filters, kernel_size = 1, strides = 1)(x)
    x = BatchNormalization()(x)
    x = ReLU()(x)

    return x

#stem of the model

input = Input(shape = (224,224,3))
```

```
x = Conv2D(filters = 32, kernel_size = 3, strides = 2, padding =
'same')(input)
x = BatchNormalization()(x)
x = ReLU()(x)

# main part of the model

x = mobilnet_block(x, filters = 64, strides = 1)
x = mobilnet_block(x, filters = 128, strides = 2)
x = mobilnet_block(x, filters = 128, strides = 1)
x = mobilnet_block(x, filters = 256, strides = 2)
x = mobilnet_block(x, filters = 256, strides = 1)
x = mobilnet_block(x, filters = 512, strides = 2)

for _ in range (5):
     x = mobilnet_block(x, filters = 512, strides = 1)

x = mobilnet_block(x, filters = 1024, strides = 2)
x = mobilnet_block(x, filters = 1024, strides = 1)

x = AvgPool2D (pool_size = 7, strides = 1,
data_format='channels_first')(x)
output = Dense (units = 1000, activation = 'softmax')(x)

model = Model(inputs=input, outputs=output)
model.summary()

#plot the model

tf.keras.utils.plot_model(model, to_file='model.png',
show_shapes=True, show_dtype=False,show_layer_names=True,
rankdir='TB', expand_nested=False, dpi=96)
```

Conclusion

MobileNet is one of the smallest Deep Neural networks that are fast and efficient and can be run on devices without high-end GPUs. Implementation of these networks is very simple when using a framework such as Keras (on TensorFlow).

Related Articles

For learning about how to implement other famous CNN architectures using TensorFlow, kindly visit the links below -

1. Xception
2. ResNet
3. VGG
4. DenseNet

References:

Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *ArXiv, abs/1704.04861*.

Towards Data Science | Deep Learning | Machine Learning | TensorFlow | Computer Vision

Data Science

Follow

Published in TDS Archive

822K followers · Last published Feb 4, 2025

An archive of data science, data analytics, data engineering, machine learning, and artificial intelligence writing from the former Towards Data Science Medium publication.

Follow

Written by Arjun Sarkar

Open in app ↗

Medium Search

Responses (2)

Akshat Kumar

What are your thoughts?

Curio, Swifty & Swaggy
Apr 8, 2022

Hello Arjun, nice article. I would recommend two changes:

In Mobilenet, they used a globalaveragepool2d layer instead of an average pooling layer and the activation function they used was relu6 to enable the network learn sparse features earlier.

Reply

Abdullah Jirjees
Oct 29, 2021

Hi and thank you for this nice article,

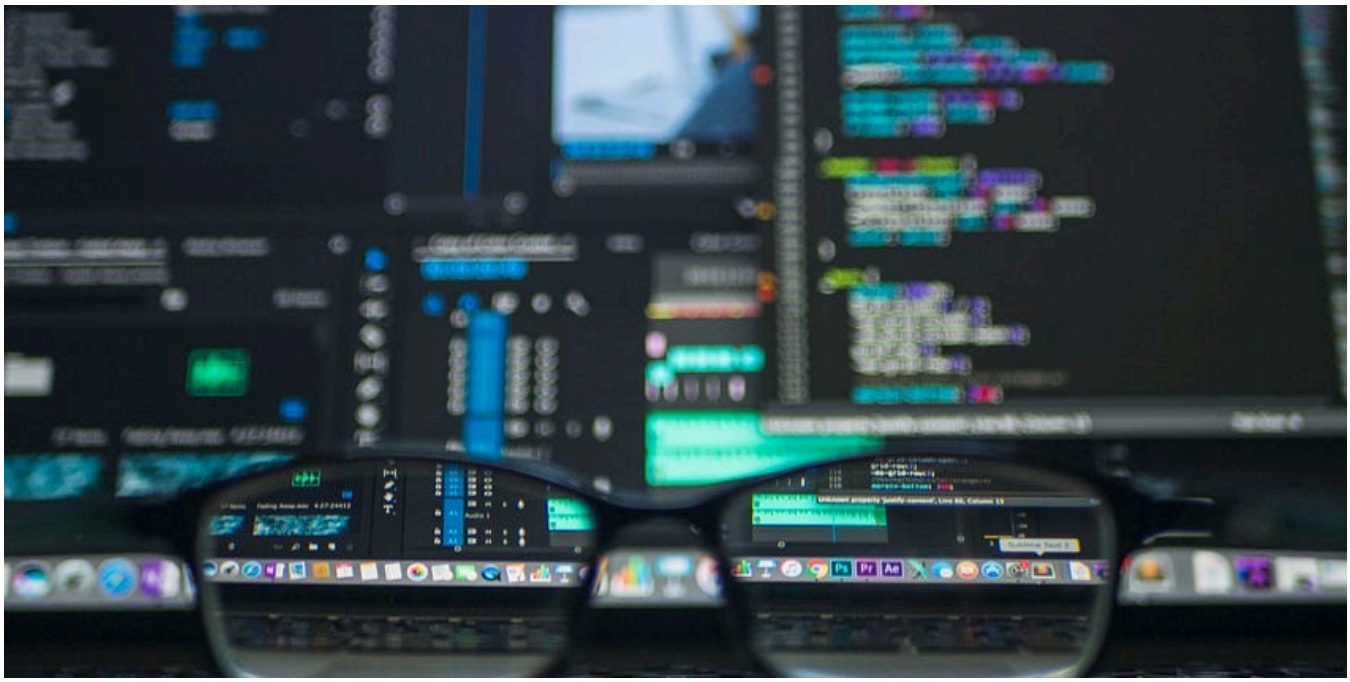
Can I change def mobilnet_block (number_of_classes)

I have to classes to classify how to add that to the archicet?

Thank you

Reply

More from Arjun Sarkar and TDS Archive

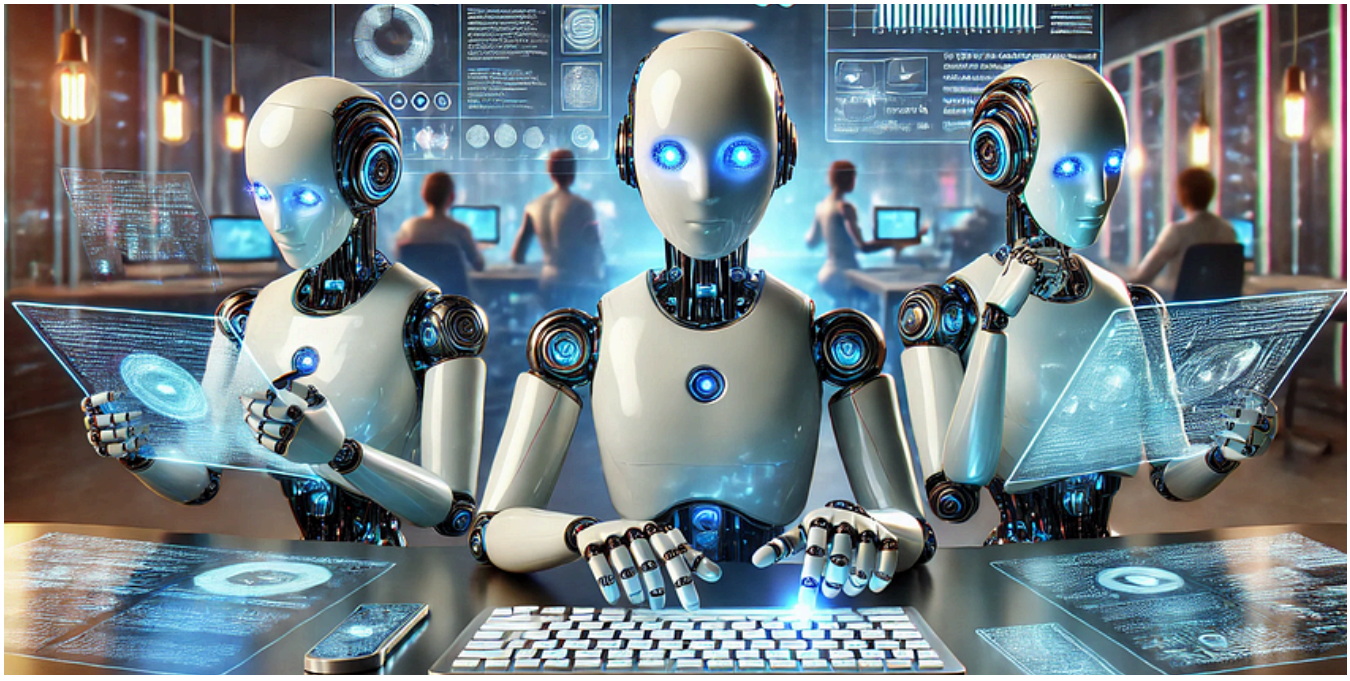


In TDS Archive by Arjun Sarkar

Build your own Transformer from scratch using Pytorch

Building a Transformer model step by step in Pytorch

Apr 26, 2023 786 15



In TDS Archive by Luís Roque

Agentic AI: Building Autonomous Systems from Scratch

A Step-by-Step Guide to Creating Multi-Agent Frameworks in the Age of Generative AI

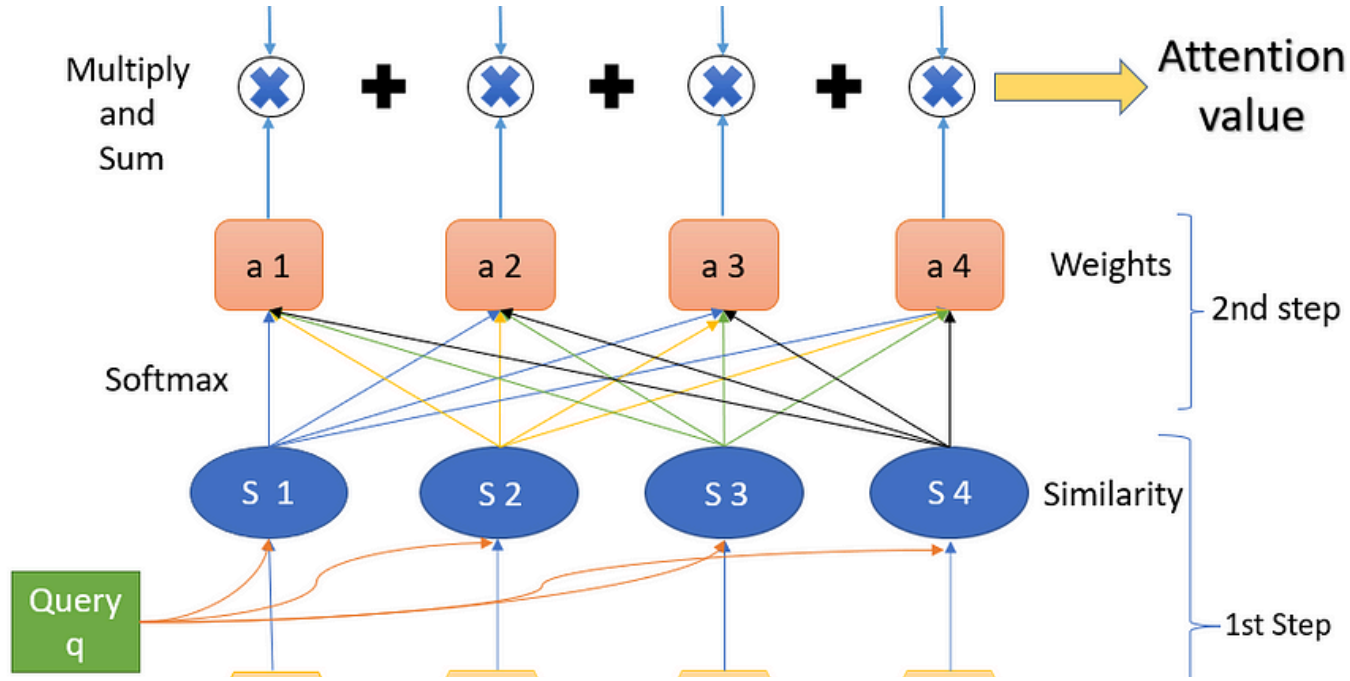
Dec 13, 2024 1.1K 24

In TDS Archive by Thuwarakesh Murallie

How to Build a Knowledge Graph in Minutes (And Make It Enterprise-Ready)

I tried and failed creating one—but it was when LLMs were not a thing!

Jan 13 1.1K 8



In TDS Archive by Arjun Sarkar

All you need to know about 'Attention' and 'Transformers'—In-depth Understanding—Part 1

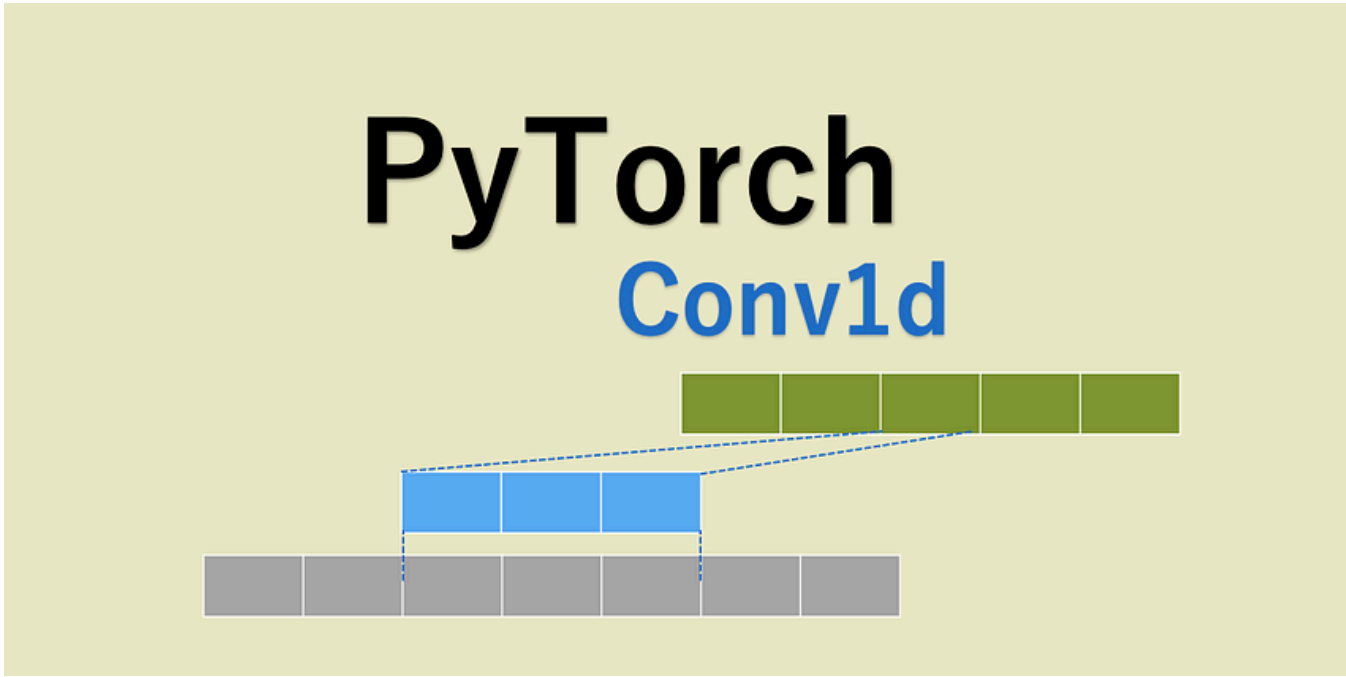
Attention, Self-Attention, Multi-head Attention, and Transformers

Feb 15, 2022 2K 34

See all from Arjun Sarkar

See all from TDS Archive

Recommended from Medium



Zhixiang Zhu

Understanding PyTorch's Conv1d Through Code: A Step-by-Step Guide

When I first encountered PyTorch's Conv1d as a beginner, I found myself puzzled by its parameters and overall mechanics.

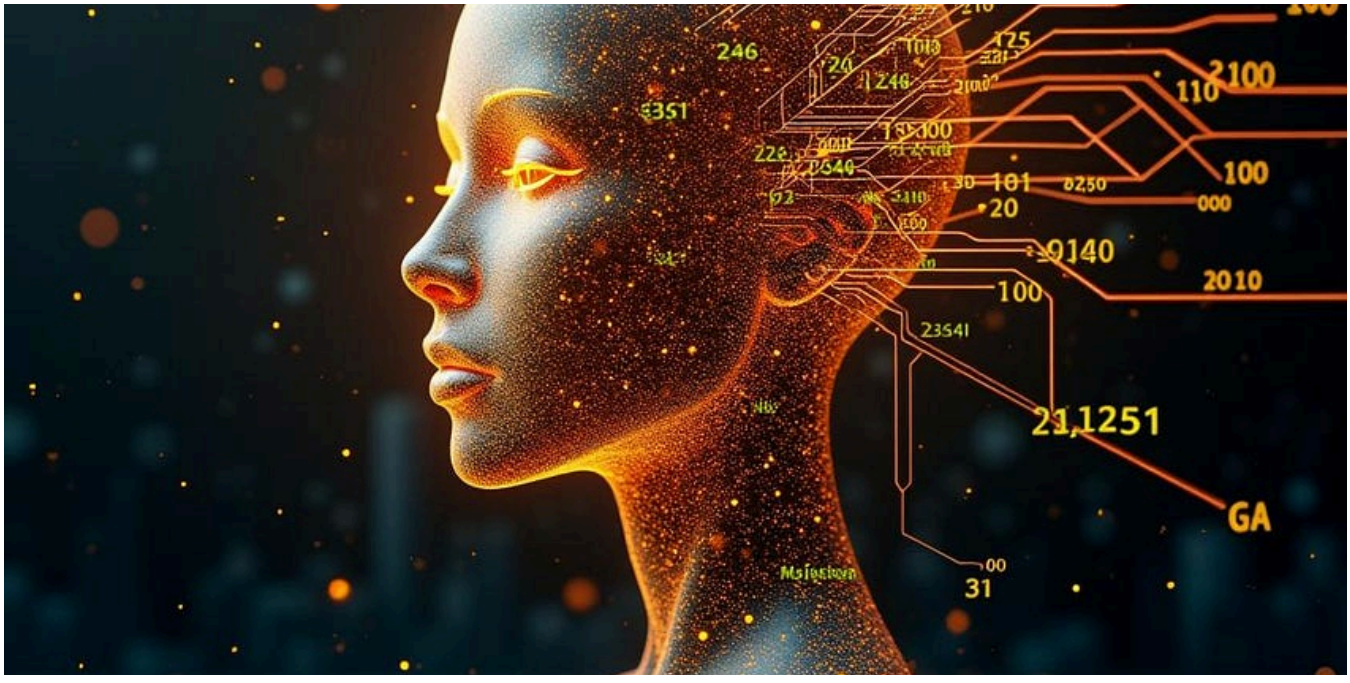
Jan 16 1

Edwin Vivek

End-to-End MLOps project with Open Source tools

Part 4: Feedback loop—Continuous Monitoring & Retraining

Feb 3



In The Deep Hub by Palash Mishra

Getting Started with PyTorch: A Beginner-Friendly Guide

If you've ever wondered how to build and train deep learning models, PyTorch is one of the most beginner-friendly and powerful frameworks...

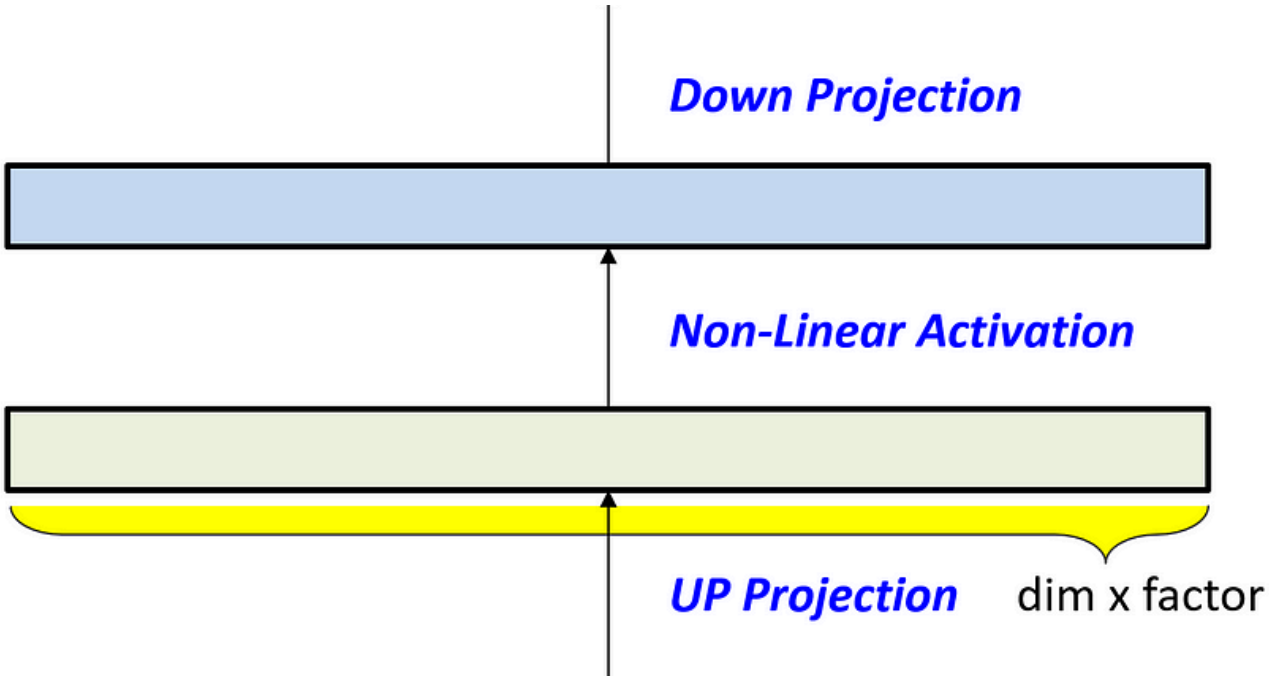
Dec 4, 2024 70

Piyush Kashyap

Transfer Learning in PyTorch: Fine-Tuning Pretrained Models for Custom Datasets

In recent years, deep learning has revolutionized the way we approach complex tasks such as image classification, object detection, and...

Dec 4, 2024

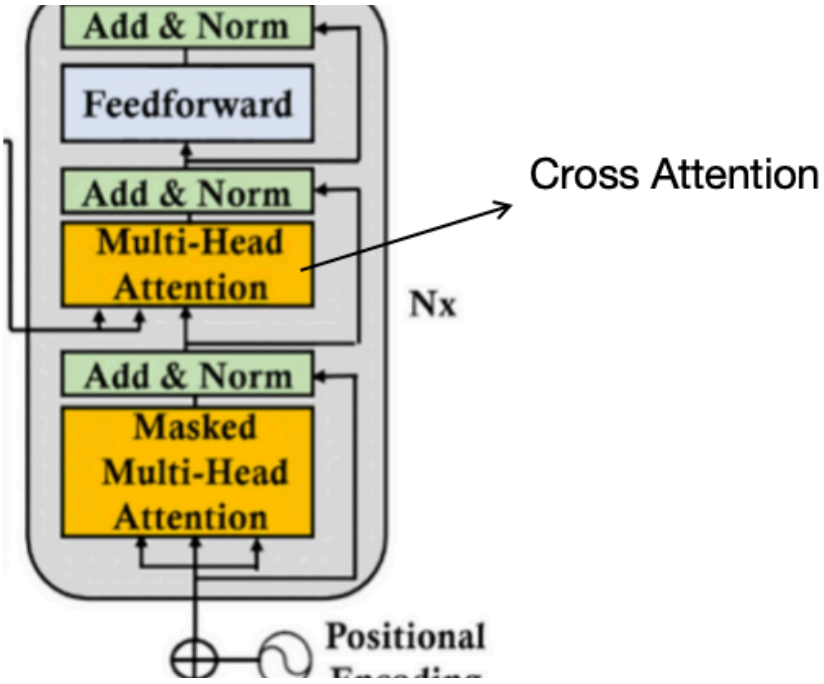


Hugman Sangkeun Jung

Mastering LLama—FeedForward (1/2): Up & Down Projection and Its Effects

Understanding Up & Down Projection in LLaMA's FeedForward Architecture: Enhanced Model Expressiveness and Information Processing

Nov 21, 2024 2



Yashwanth S

Deep Learning Series 21: Understanding Cross Attention in Transformer Models

When it comes to modern natural language processing (NLP) and sequence-to-sequence tasks, attention mechanisms have revolutionized how...

Dec 24, 2024

See more recommendations