**Development of C programs for**

**Convolutional Neural Network Accelerators**

1. **Introduction**

In modern society, deep learning is rapidly developing based on massive data generated by various media. Especially, CNN is popular and has achieved many results. However, existing CNN requires a lot of computing resources and is time-consuming. Many people are trying to optimize CNN through accelerators. As part of this effort, the purpose of this project is to create an accelerator that maximizes computational efficiency while minimizing computational latency and memory access by applying differentiated hardware optimizations.

This report contains the implementation of a generator for c code and optimized code based on layer information and a verifier to check the generated code. There is a template library for generator applied to various models, implemented generator and verifier and verified whether the results of the software version, Python (keras) are same.

We are working on other models such as MobileNet and GoogleNet etc. And by optimizing hardware, it will reduce the amount of computing energy and time consuming in CNN.

1. **Git repository**

repository link : <https://github.com/Hamidu68/ML-acceleration>

1. **Development environment**

* python 2.7
* tensorflow
* keras(2.2)
* numpy

1. **Various Models**

**implemented models** : vgg16, vgg19, resnet50, inceptionv3, mobilenet, mobilenetv2,

densenet121, densenet169, densenet201, inceptionresnetv2

cf. Dropout layer is not contained because this is for testing.

1. **Vgg16 (implemented)**

Layers : Conv2D, MaxPooling2D, Flatten, Dense

1. **Vgg19 (implemented)**

Layers : Conv2D, MaxPooling2D, Flatten, Dense

1. **ResNet50 (implemented)**

Layers : Conv2D, MaxPooling2D, Add, BatchNormalization, Activation, Flatten,

ZeroPadding2D, AveragePooling2D, Dense,

1. **InceptionV3 (implemented)**

Layers : Conv2D, MaxPooling2D, Add, BatchNormalization, Activation, Flatten,

ZeroPadding2D,AveragePooling2D, Dense, GlobalAveragePooling2D, Concatenate

1. **AlexNet** (need csv file -> keras doesn’t support this model )
2. **LeNet5** (implemented but need csv file to test-> keras doesn’t support this model )
3. **GoogleNet** (need csv file -> keras doesn’t support this model )
4. **MobileNet (implemented)**

Layers : Conv2D, BatchNormalization, Activation, Reshape, DepthwiseConv2D, ReLU,

ZeroPadding2D, GlobalAveragePooling2D

1. **MobileNetV2 (implemented)**

Layers: Conv2D, BatchNormalization, DepthwiseConv2D, ReLU, Add,

GlobalAveragePooling2D, Dense

1. **SqueezeNet(**implemented but need csv file to test-> keras doesn’t support this model**)**
2. **DenseNet(densenet121, densenet169, densenet201 all implemented)**

Layers : Zeropadding2D, Conv2D, BatchNormalization, Activation, MaxPooling2D,

Concatenate, AveragePooling2D, GlobalAveragePooling2D, Dense

1. **NasNet(2 layers are not implemented yet)**
2. **Xception (1 layer is not implemented yet)**
3. **InceptionResNetV2 (implemented)**

Layers : Conv2D, BatchNormalization, Activation, MaxPooling2D, AveragePooling2D,

Concatenate, Lambda, GlobalAveragePooling2D, Dense

1. **Usage**

#### **A. Download the repository**

**git clone "git@github.com:Hamidu68/ML-acceleration.git"**

#### **B. Set condition**

in ML-acceleration/config.json file, change the value of various keys.

**{**

**"network": "vgg19",**

**"model\_info\_file": "vgg19\_test.csv",**

**"c\_code\_generate": "True",**

**"keras\_generate": "True",**

**"sw\_verify": "True",**

**"hw\_code\_generate": "False",**

**"vivado\_generate": "False",**

**"data\_type": "int",**

**"random\_range": "5",**

**"skip\_batch\_layer": "False"**

**}**

* network : name of the model(network) ex. vgg19, resnet50
* model\_info\_file : name of the test file which contains layer information of the model (ex. vgg19\_test.csv)
* c\_code\_generate : generate software code(c code of the model, {model\_name}.cpp file) or not (True/False)
* keras\_generate : build model from keras or not (True/False)
* sw\_verify : verify the output value between c code and keras or not (True/False)
* hw\_code\_generate : generate hardware code(optimized code, {model\_name}.cpp file & {model\_name}\_test.cpp file or not(True/False)
* vivado\_generate : compare the value between software code and hardware code or not (True/False)
* data\_type : data type (int, unsinged int, float, ap\_uint<16>)
* random\_range : number of range that will be used to generate input.bin, weight.bin value (ex. 5 means input.bin, weight.bin files consist of value between 1 to 5)
* skip\_batch\_layer : skip batch layer or not (True/False)

#### **C. Run run.sh**

in ML-acceleration folder, use the command below to run run.sh script file.

**./run.sh**

in run.sh file,

**python main.py**

script file 'run.sh' will run main.py file.

1. **Code implementation**
2. **Generate Software Code**

For generating software code(.cpp) in various models, I followed the steps below.

1. **Generate .csv Input file**
2. Import the model(network) that you want to generate in ML-acceleration/src/extract\_configs.py file

For example, if we want to generate resnet50 file,

from keras.applications.resnet50 import ResNet50

write these at the top of extract\_configs.py.

#### Build model and call extract\_configs function

#### For example, if we want to generate resnet50\_test.csv file in model\_info folder,

put absolute path or relative path including file name in {file\_path} without file

extension.

(ex. {file\_path} : '../model\_info/resnet50\_test')

After that, build model and call extract\_configs function like the below.

model = ResNet50(include\_top=True, weights='imagenet', input\_tensor=None, input\_shape=(224,224,3), pooling=None, classes=1000)

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*resnet50\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

extract\_configs(model, {file\_path})

#### Run extract\_configs.py file by using the command below.

python extract\_configs.py

1. **Generate C code for each layer**

Write a code in function below depends on layer type.

**SWGenerators class** in ML-acceleration/sw\_generator.py :

- function ‘gen\_sw\_def\_layer’ to write software code of each layer

- function ‘gen\_sw\_static\_variables’ to declare static variables of each layer (input/

weight/ bias)

- function ‘gen\_sw\_output\_variables’ to declare an output variable of each layer

- function ‘gen\_sw\_call\_layer’ to call function in each layer

- function ‘gen\_print\_result’ to write code that prints the output values

- function ‘gen\_initialization’ to write code that initializes variables (input/ weight/

bias)

**\_\_init\_\_.py** in Layers folder :

from .Data import Data

from .Layers import Layers

from .InputLayer import InputLayer

from .Conv2D import Conv2D

from .Layers import Conv2D\_HW

from .Layers import Conv2D\_DAC2017

from .Layers import MaxPooling2D

from .Layers import BatchNormalization

from .Layers import Activation

from .Layers import AveragePooling2D

from .Layers import ZeroPadding2D

from .Layers import Flatten

from .Layers import Dense

from .Layers import Add

import each layer like above ‘from .Layers import [Layer\_name]’

**Layers.py** in Layers folder :

Define new class in each layer(that inherits Layers class)

**class Layer\_name(Layers)**

In each class, declare constructor to get shape, set output, set weight and bias (if it

is needed) and add a software code.

1. **Add condition statement (to check bias, padding, stride…)**

For example,

**if** self.config['scale'] == 'False':

batch\_normal = open("cpp\_generator/resnet50/Template/Function/BatchNormalizati

on\_no\_scale.txt")

**else**:

batch\_normal = open("cpp\_generator/resnet50/Template/Function/BatchNormalizati

on.txt")

**if** self.config['activation'] == 'relu': *# Activation = relu*

func =dense\_softmax.format(Name=self.config["name"], Input\_channel=input\_shape[1],

Output\_channel=output\_shape[1], comment\_begin=comment1, comment\_end=comment1, comment=comment)

self.function['code'] += func + "\n"

**else**: *# Activation = softmax*

func = dense\_relu.format(Name=self.config["name"], Input\_channel=input\_shape[1],

Output\_channel=output\_shape[1], comment\_begin=comment1, comment\_end=comment1,

comment=comment)

self.function['code'] += func + "\n"

**if** self.config['padding'] == 'valid':

func =conv2d\_valid.format(Name=self.config["name"], Input\_channel=input\_shape[3],

Input\_width=input\_shape[1], Stride\_width=stride\_shape[0], Stride\_height=stride\_shape[1],

Input\_height=input\_shape[2], Output\_channel=output\_shape[3], Filter\_width=filter\_shape[0],

Filter\_height=filter\_shape[1], Output\_width=output\_shape[1],Output\_height=output\_shape[2],

comment\_begin=comment1,

comment\_end=comment1, comment=comment)

self.function['code'] = func + "\n"

**else**:

func = conv2d\_same.format(Name=self.config["name"], Input\_channel=input\_shape[3],

Input\_width=input\_shape[1], Stride\_width=stride\_shape[0], Stride\_height=stride\_shape[1],

Input\_height=input\_shape[2], Output\_channel=output\_shape[3], Filter\_width=filter\_shape[0],

Filter\_height=filter\_shape[1], Output\_width=output\_shape[1],Output\_height=output\_shape[2],

comment\_begin=comment1, comment\_end=comment1, comment=comment)

self.function['code'] = func + "\n"

add condition statement if it is needed to check bias, padding, stride, scale etc. (like the

picture above)

1. **Generate C code**

Create script file ‘run.sh’ to generate C code

By running run.sh file, generate C code. For more detail, refer to **Ⅴ. Usage**)

1. **Create Keras\_Verifier.py**

Add command-line arguments

# sys.argv[1] : test file (./test.csv)

# sys.argv[2] : weight file (.bin)

# sys.argv[3] : input file (.bin)

# sys.argv[4] : data type (int, unsigned int, float, ap\_uint<16>)

Define function of each layer like below,

**def add\_Conv2D(input\_tensor=None, info=None, fid=None, dtype=int)**

def add\_[layer\_type] : contain ‘read information of layer’, ‘read weights from file’, ‘get output tensor

Define print function and separate it into two parts depends on output dimension.

1. **Compare the outputs**

Use ‘vimdiff’ to compare the outputs(read README.md)

1. **Check maximum error range**

Caculate maximum error range by writing code below.

**for** i **in** range(len(c)):

k\_num = float(k[i])

c\_num = float(c[i])

**if** c\_num > k\_num:

**if** k\_num == 0.0:

error = c\_num - k\_num

**else**:

error = (c\_num - k\_num) /k\_num

**else**:

**if** k\_num == 0.0:

error = k\_num - c\_num

**else**:

error = (k\_num - c\_num) / k\_num

**if** maximum < error:

maximum = error

c\_max = c\_num

k\_max = k\_num

**print**("maximum error : " + str(maximum) + " when c has an element of " + str(c\_max) +

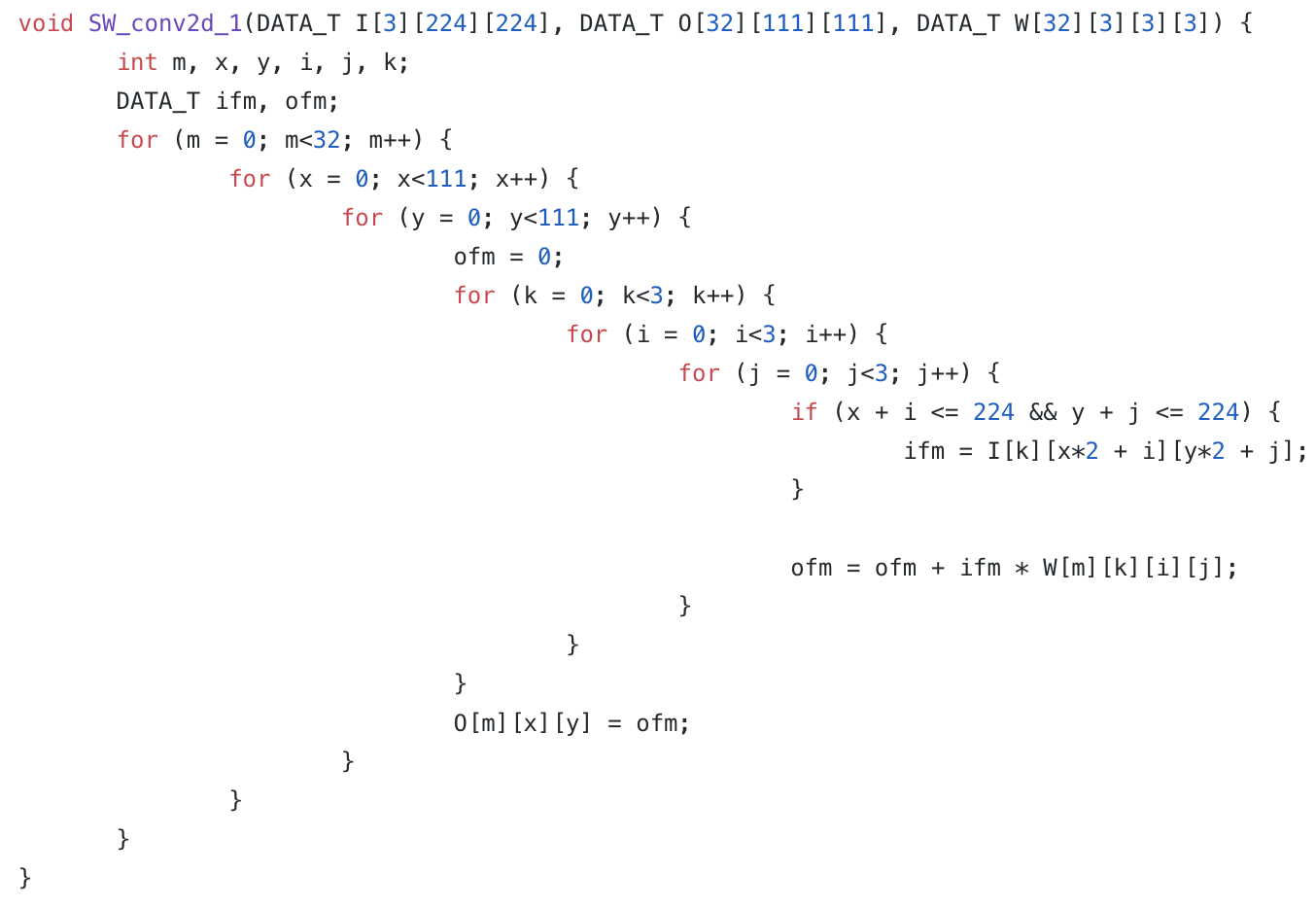
" and keras has an element of " + str(k\_max))

1. **Generate Hardware Code**
2. **Code**

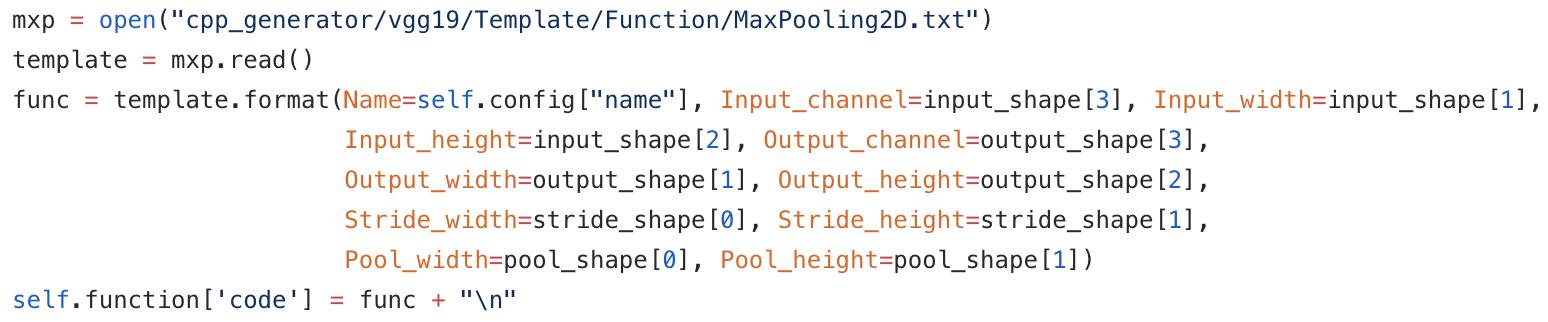
I attached the code of generator and generated code of each layer.

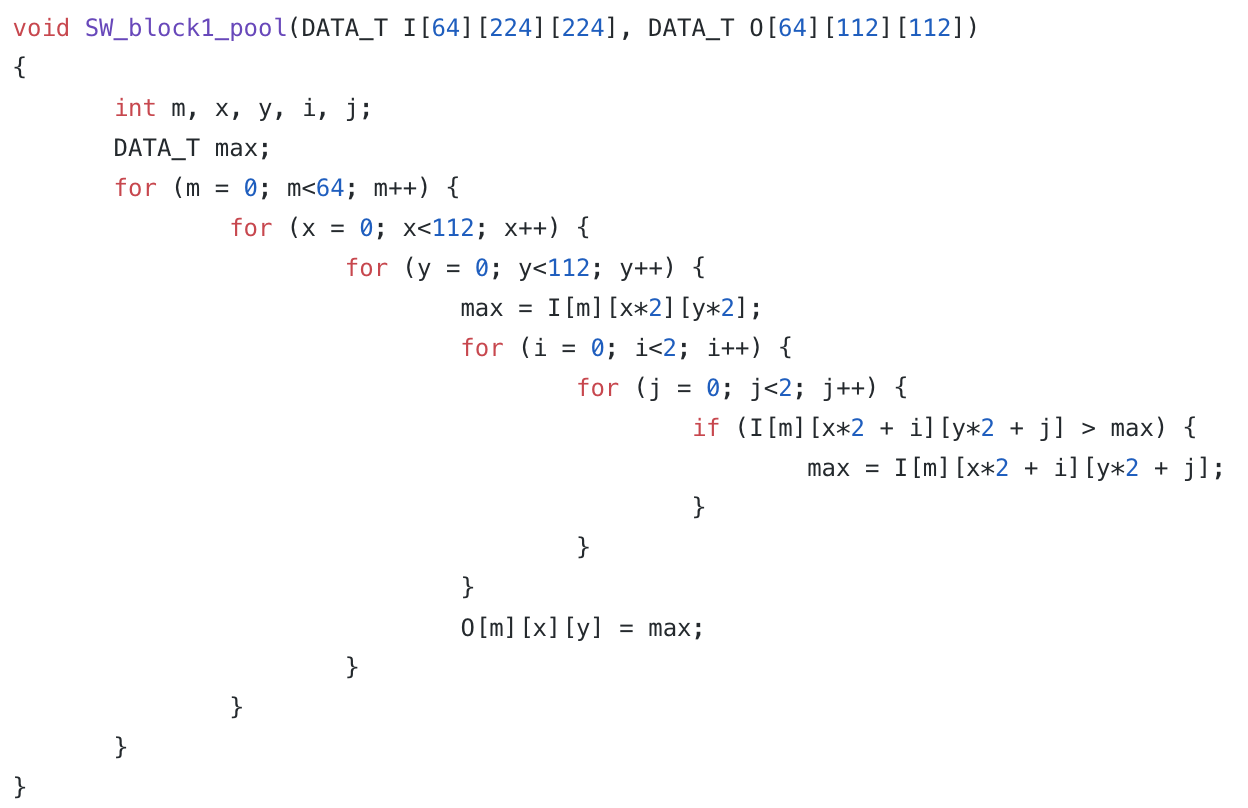
Conv2D



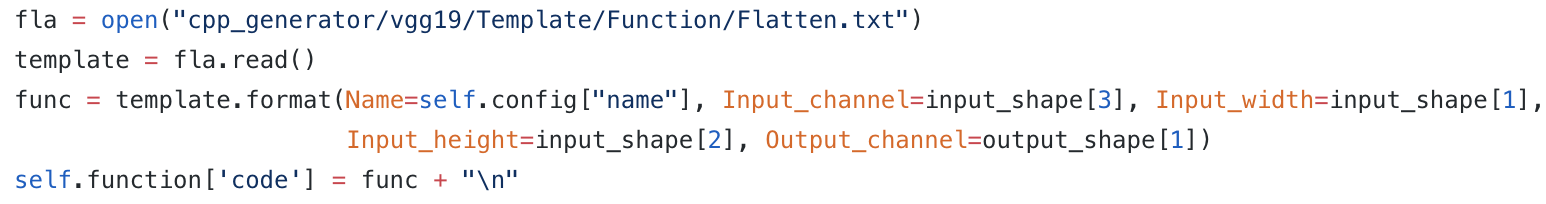


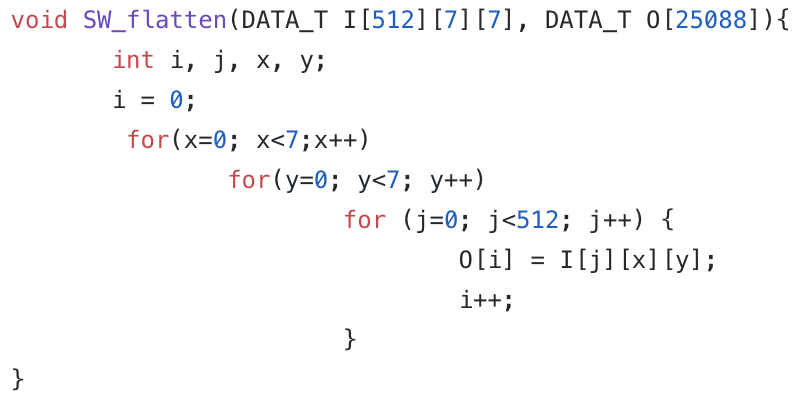
**MaxPooling2D**





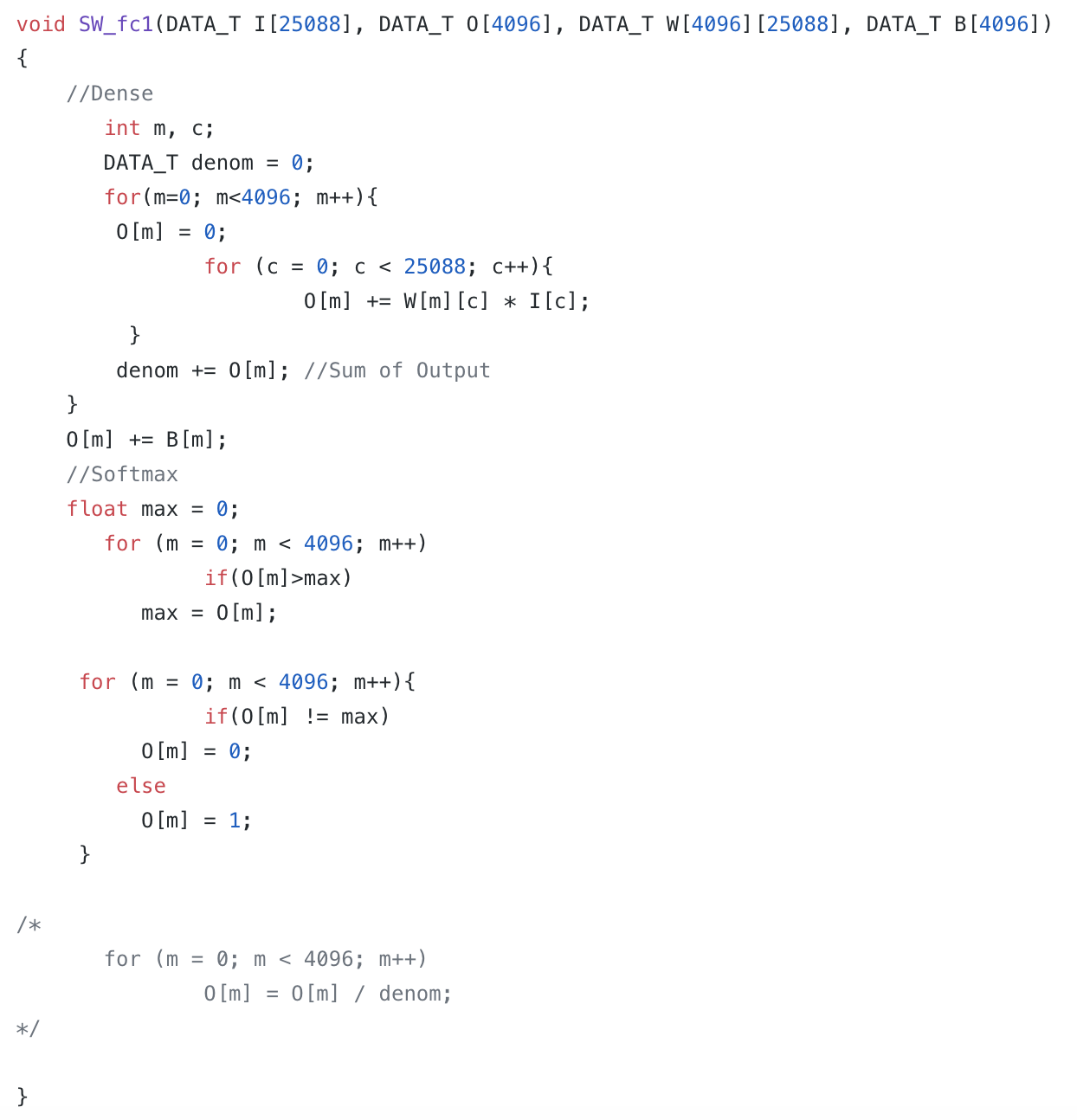
**Flatten**



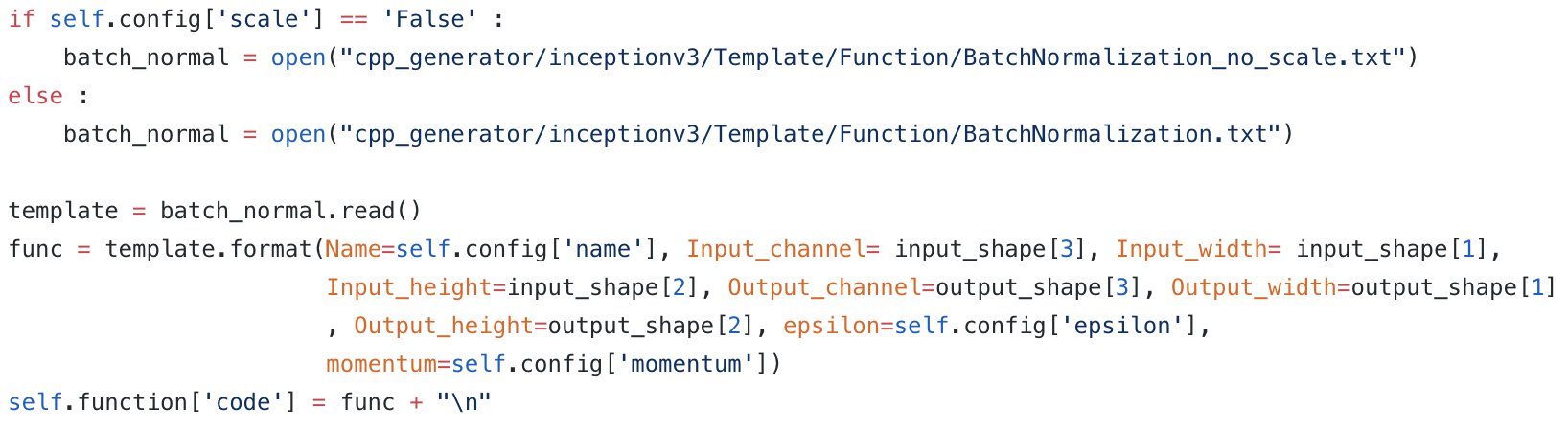


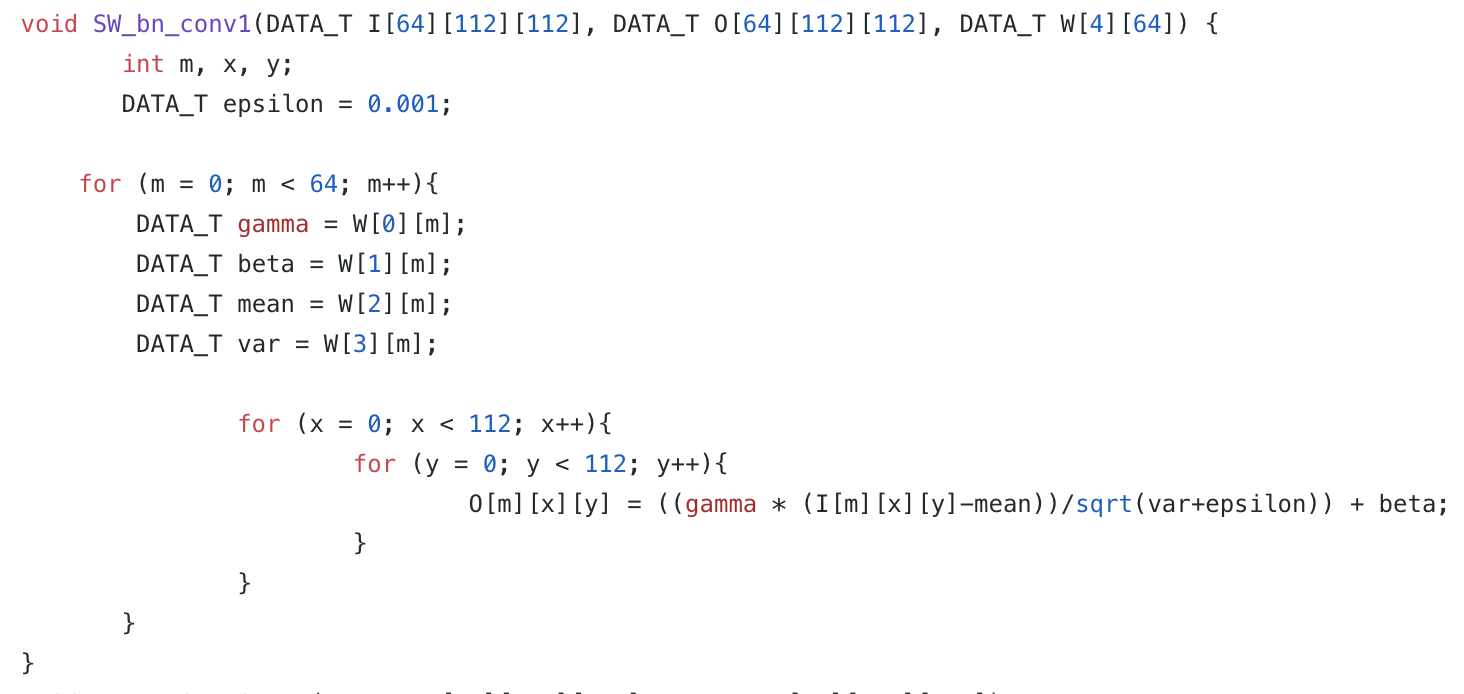
**Dense**



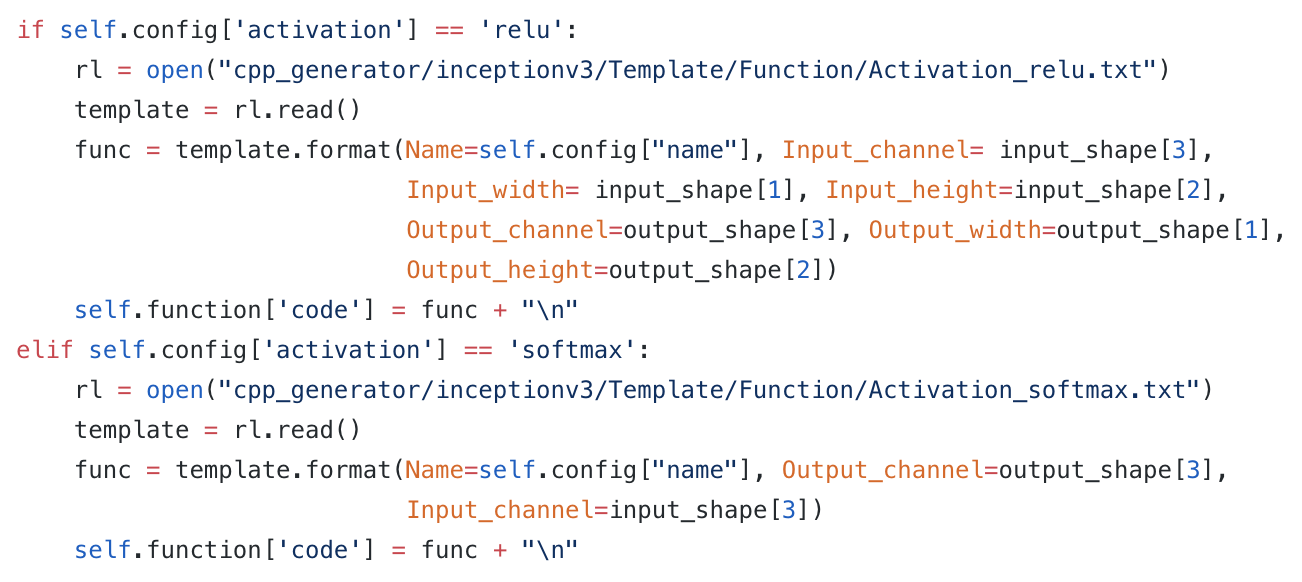


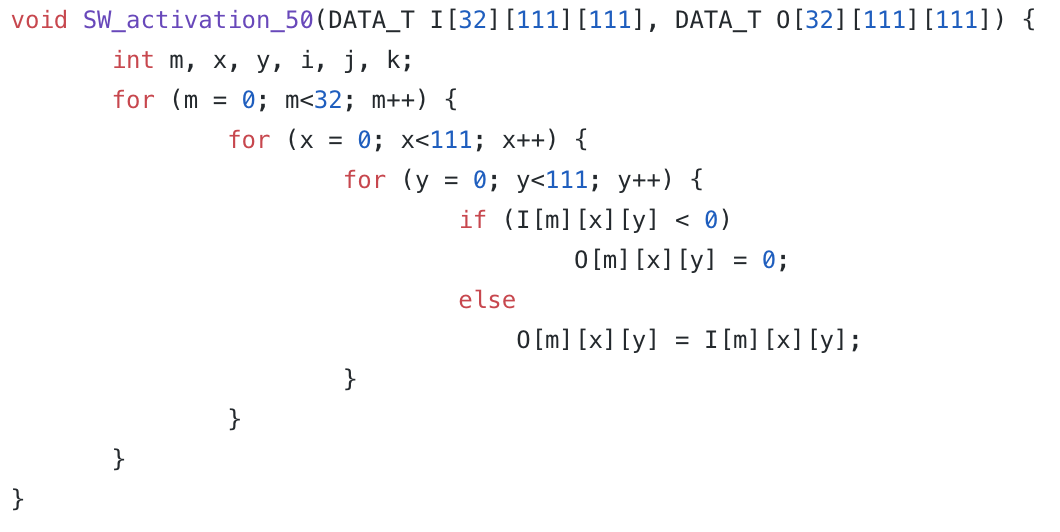
**BatchNormalization**



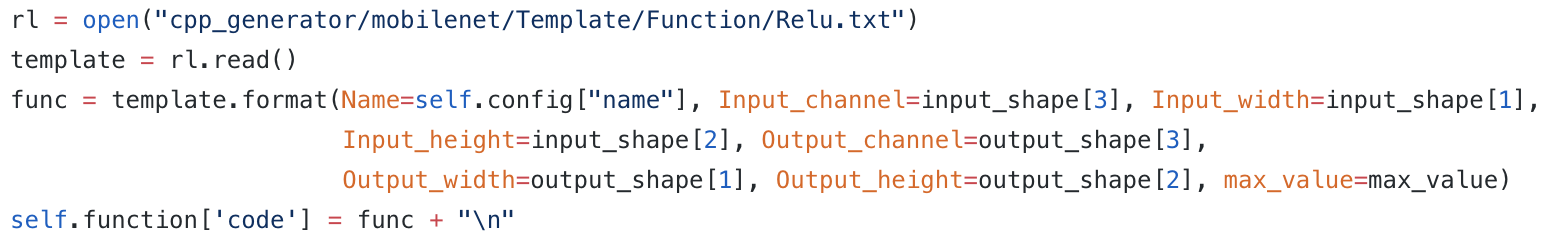


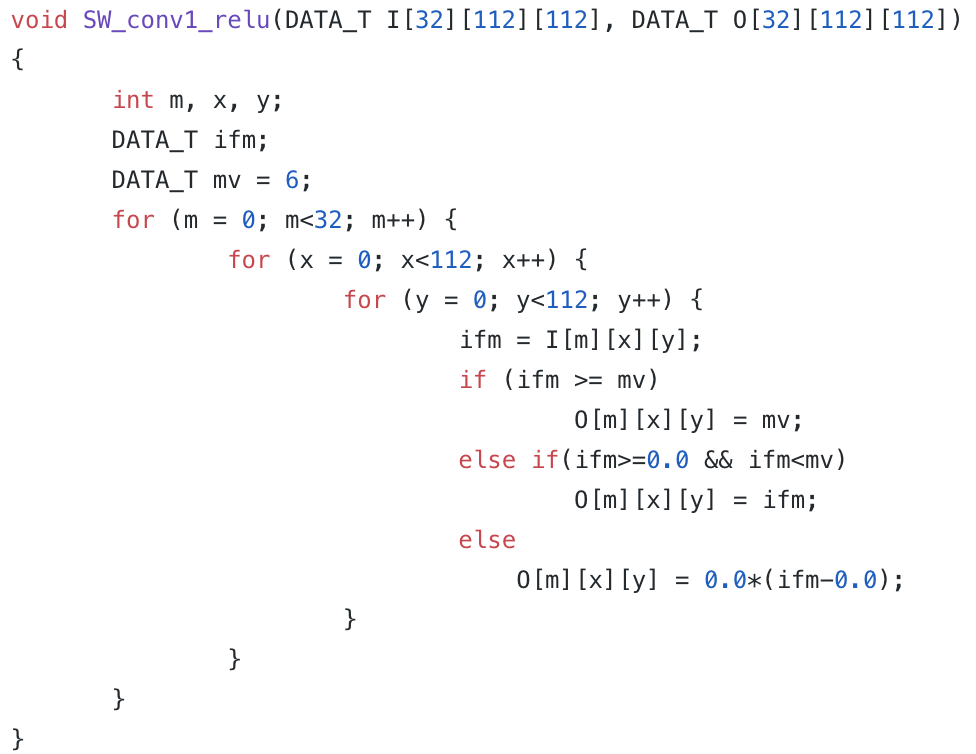
**Activation**



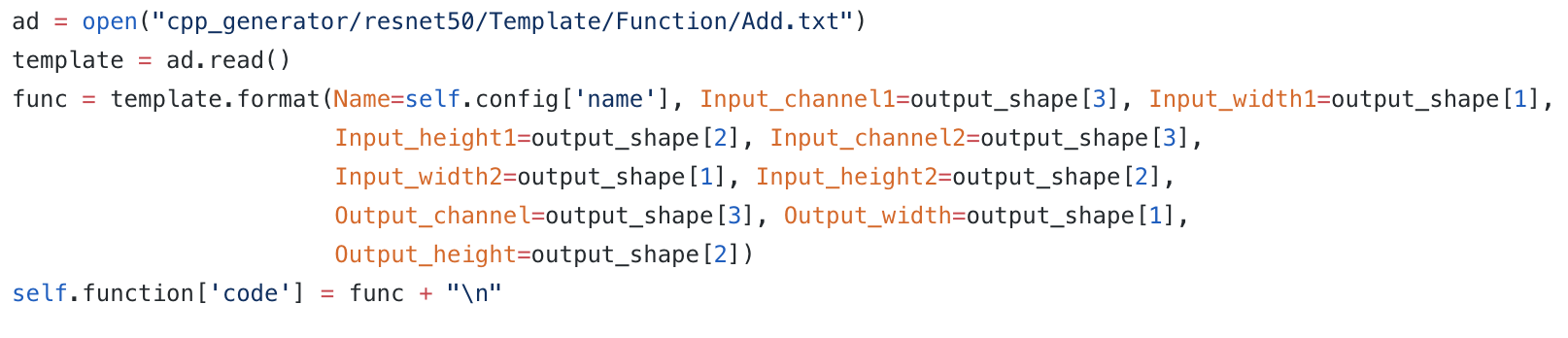


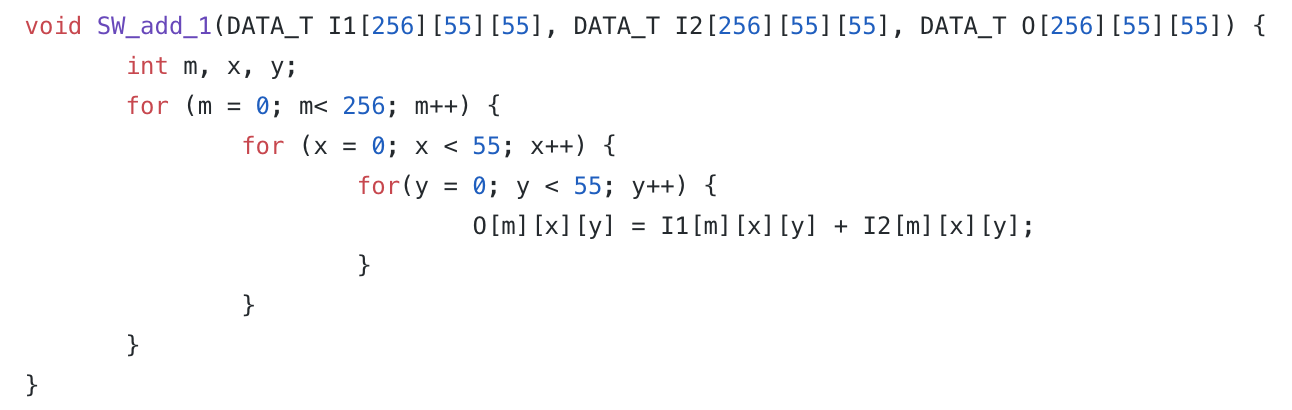
**ReLU**



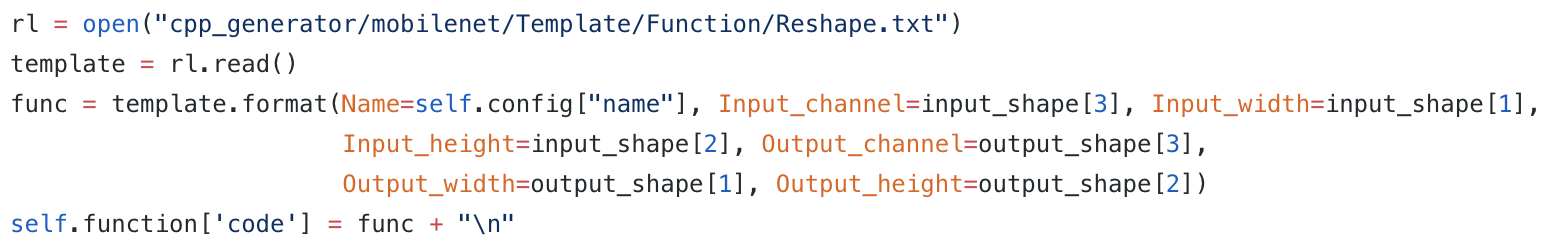


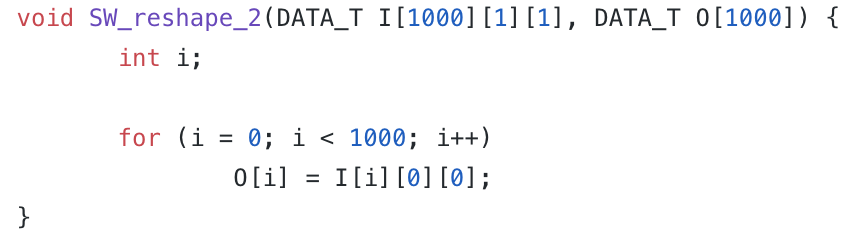
**Add**



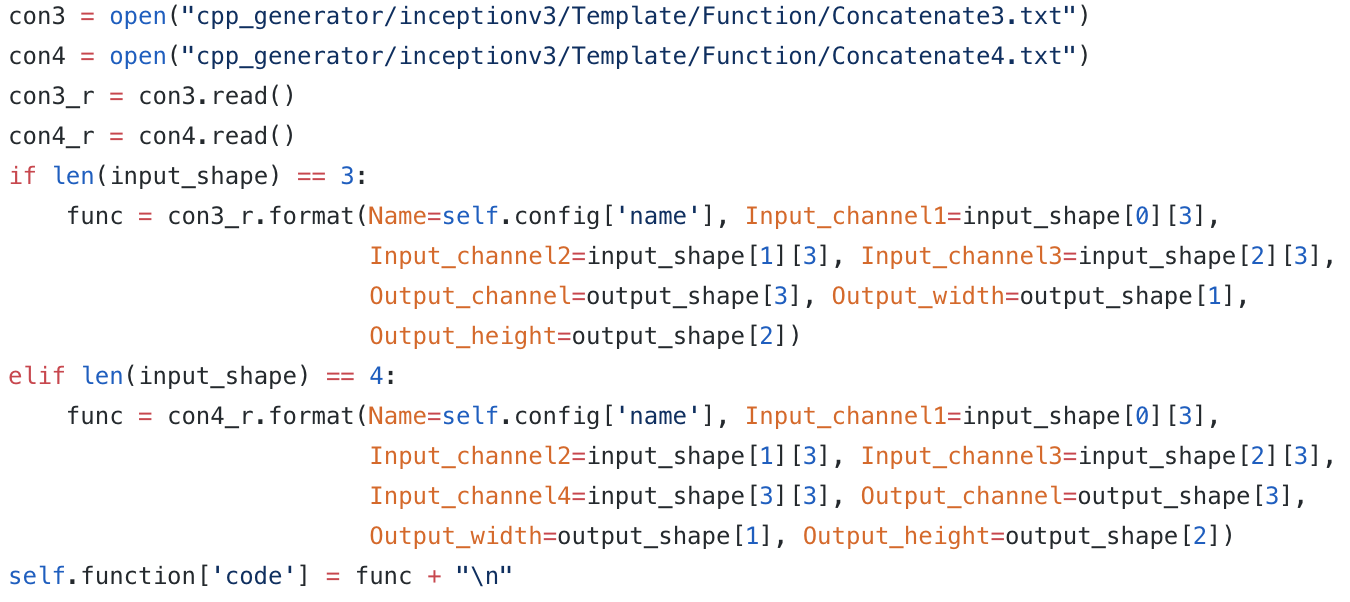


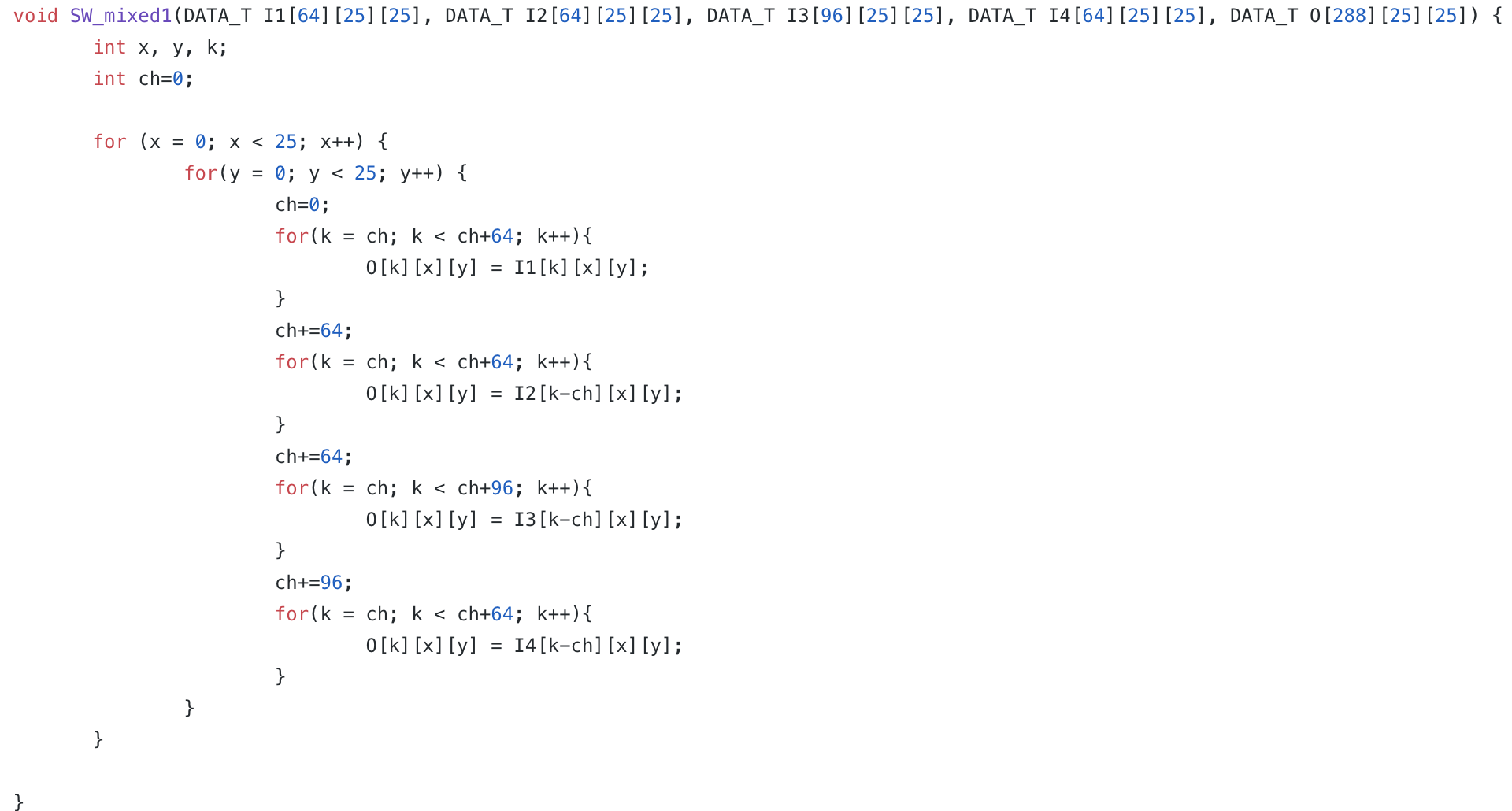
**Reshape**



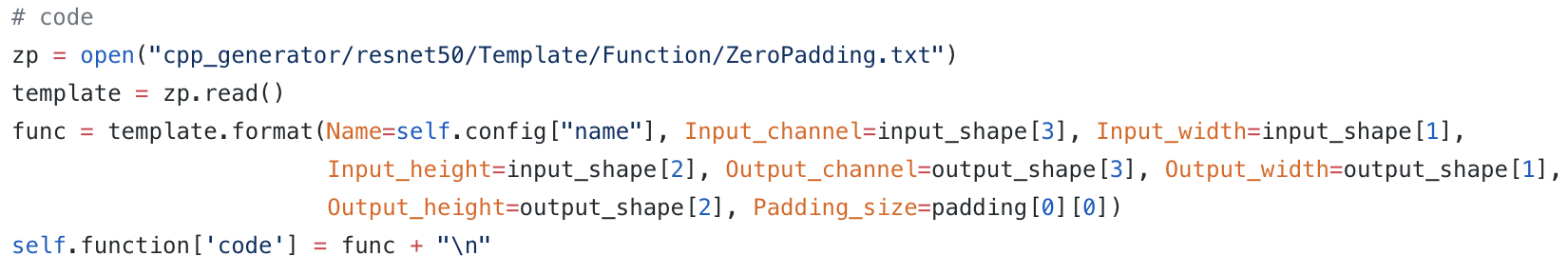


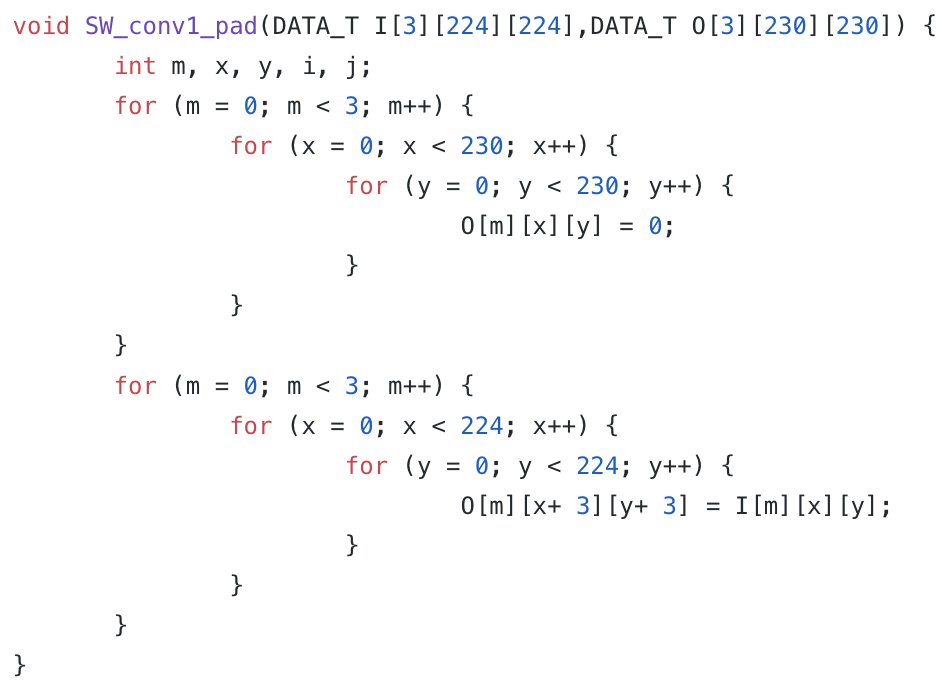
**Concatenate**



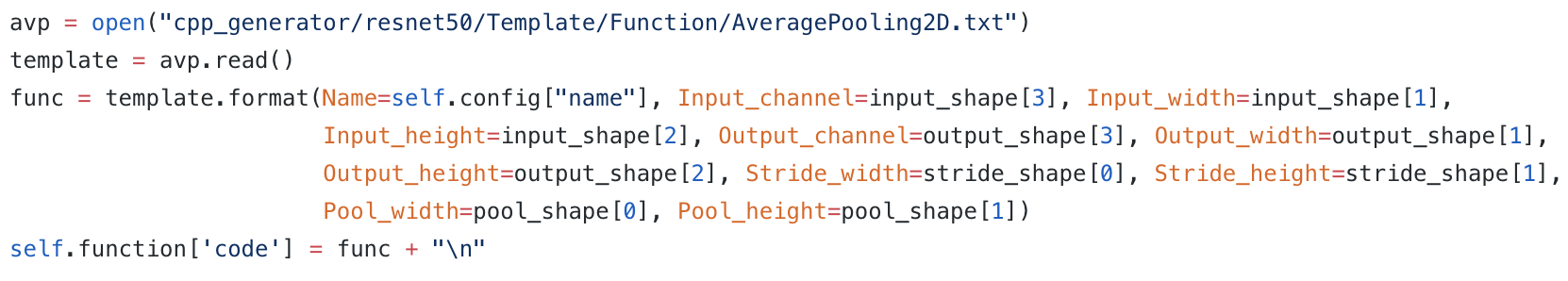


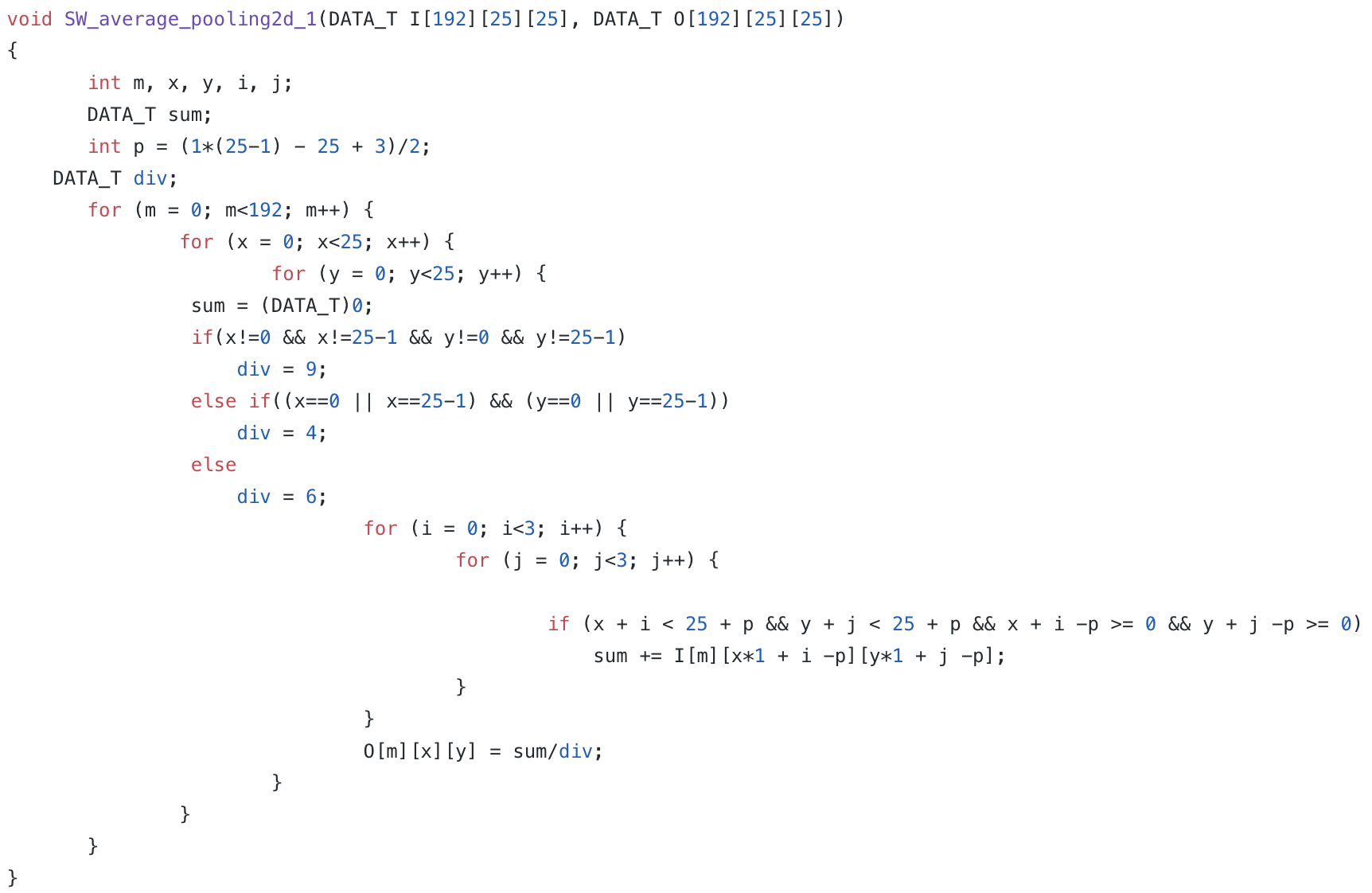
**ZeroPadding2D**



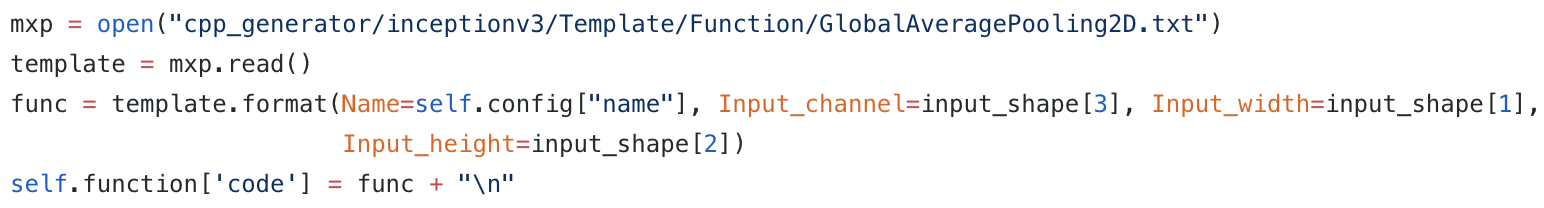


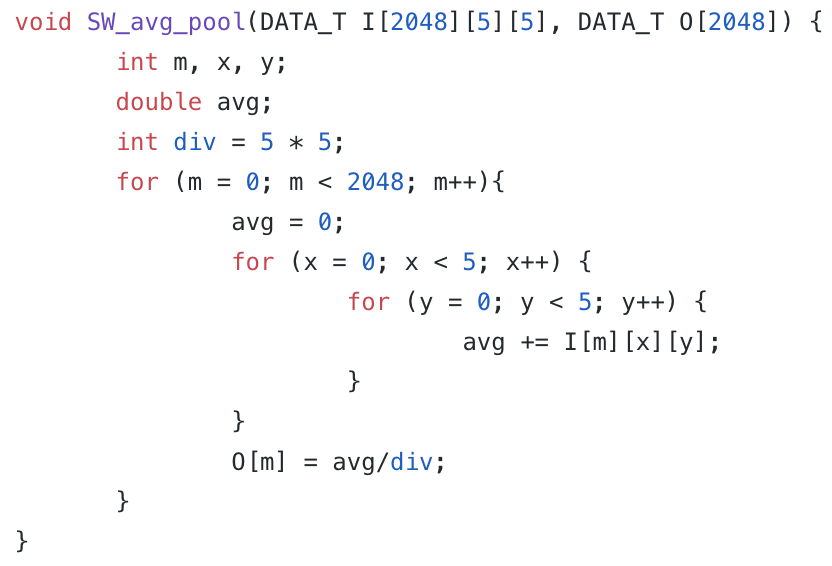
**AveragePooling2D**





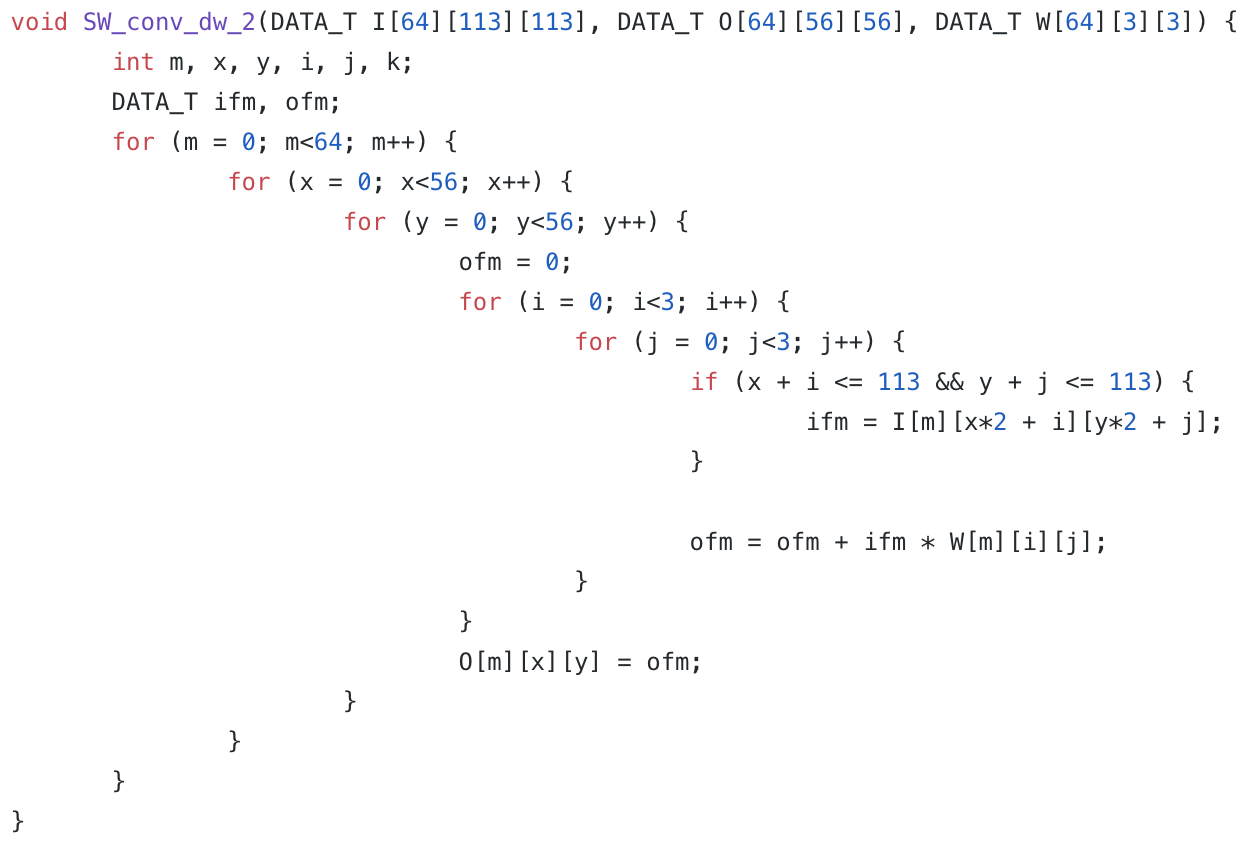
**GlobalAveragePooling2D**



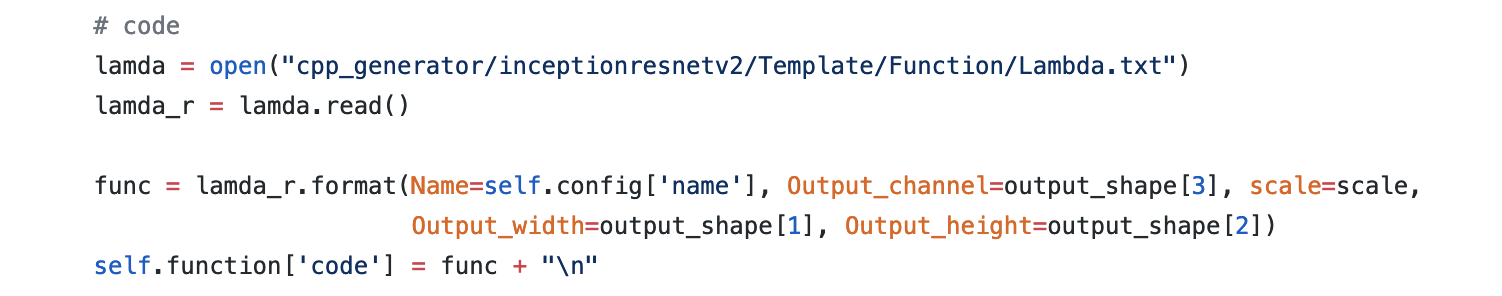


**DepthwiseConv2D**





**Lambda**

****