Food Recognition Based on Deep Learning

Jingyi Zhao (N19315072), Zichun Du (N12864205)
Department of Electrical & Computer Engineering
New York University

***Abstract*—After learning several ways to analysis pictures, we try to apply them on basic topics liking food recognize.**

I. Introduction

In this project, we propose a deep learning based food image recognition algorithm to recognize different food types and accuracy.

II. Proposed Approach

A. Dataset

In this project, we used the dataset of ETHZ-FOOD-101. https://www.vision.ee.ethz.ch/datasets_extra/food-101/. In this dataset, there are 101 kinds of food, 1000 images of each kind of food. But since the dataset is too big to train, we only choose some of them.

B. Depth Wise Operation

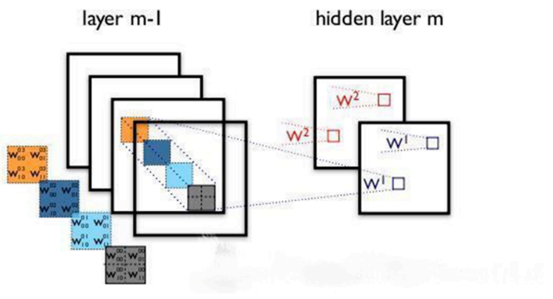
Depthwise Separable Convolutions are a key building block for many efficient neural network architectures [27, 28, 20] and we use them in the present work as well. The basic idea is to replace a full convolutional operator with a factorized version that splits convolution into two separate layers. The first layer is called a depthwise convolution, it performs lightweight filtering by applying a single convolutional filter per input channel. The second layer is a 1 × 1 convolution, called a pointwise convolution, which is responsible for building new features through computing linear combinations of the input channels. Standard convolution takes an $h_i \times w_i \times d_i$ input tensor L_i, and applies convolutional kernel $K \in \mathcal{R}^{k \times k \times d_i \times d_j}$ to produce an $h_i \times w_i \times d_j$ output tensor L_j. Standard convolutional layers have the computational cost of

$$h_i \cdot w_i \cdot d_i \cdot d_j \cdot k \cdot k.$$

Depthwise separable convolutions are a drop-in replacement for standard convolutional layers. Empirically they work almost as well as regular convolutions but only cost:

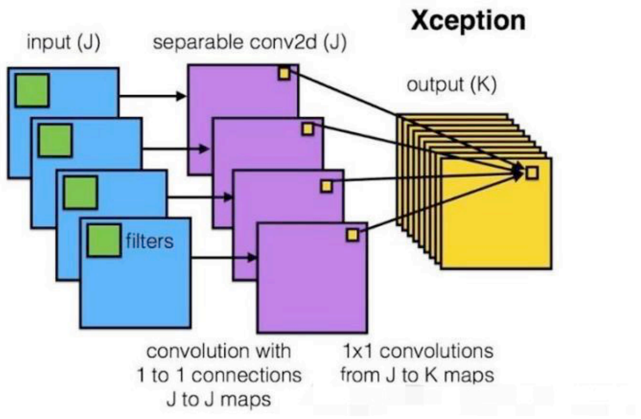
$$h_i \cdot w_i \cdot d_i (k^2 + d_j)$$

which is the sum of the depthwise and 1 × 1 pointwise convolutions. Effectively depthwise separable convolution reduces computation compared to traditional layers by almost a factor of k 21. MobileNetV2 uses k = 3 (3 × 3 depthwise separable convolutions) so the computational cost is 8 to 9 times smaller than that of standard convolutions at only a small reduction in accuracy.



(a)

The standard convolution process can be seen in the figure above. When a 2 *2 convolution core is convoluted, all channels in the corresponding image area are considered simultaneously. The problems are: why do we have to consider both the image area and the channel? Why can't we separate the passageway from the space area?


Xception
input (J)
separable conv2d (J)
output (K)
filters
convolution with 1 to 1 connections J to J maps
1x1 convolutions from J to K maps

(b)

The Xception network was invented on the basis of the above problems. Firstly, we do the convolution operation for each channel, and there are as many filters as there are channels. After the new channel feature maps are obtained, the standard 1*1 cross-channel convolution operation is performed on these new channel feature maps. This operation is called "Depth Wise convolution" or "DW".

This operation is quite effective. The performance of Inception V3 has been exceeded in ImageNet 1000 classification tasks, and a large number of parameters have been reduced. Let's calculate it. Assuming that the number of input channels is 3, the number of output channels is 256. Two ways are required:

1. Directly connect a convolution core of 3 x 3 x 256 with parameters of 3 x 3 x 3 x 3 x 256 = 6,912.

2. DW operation is completed in two steps. The parameters are 3 *3 *3 + 3 *1 *256 = 795, and the parameters are reduced to one ninth.

Therefore, a Depthwise operation reduces a lot of parameters compared with the standard convolution operation. At the same time, the paper points out that this model has a better classification effect.

C. *Linear Bottlenecks*

In MobileNet V1, in addition to introducing depthwise separable convolution instead of traditional convolution, an experiment was conducted to reduce the model channel by using width multiplier parameter, which is equivalent to "slimming" the model. So that the feature information can be more concentrated in the reduced channel. But if a non-linear activation layer, such as ReLU, is added at this time, there will be a loss of information. So, the linear bottleneck is used to reduce the loss.

Input is a two-dimensional circular data. After linear transformation multiplied by Tx, dimension becomes n, then ReLU, then T'x changes back, and the second to the last is the obtained graph. It can be seen that n = 2 and N = 3, data has a relatively large loss, the greater n, the less loss. What does this mean? If the input dimension is 2 and the conv + ReLU comes out dimension 100, is it really necessary? Can I change it linearly to 2? This information is not lost much, so we plug in a bottlenecks later to capture this real dimension. This acquisition is a linear transformation, so no ReLU is added to the front of the bottlenecks layer. This kind of non-linear activation may cause a decrease in accuracy.

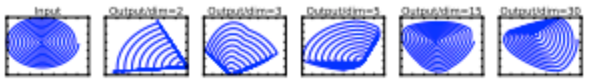


Figure 1: Examples of ReLU transformations of low-dimensional manifolds embedded in higher-dimensional spaces. In these examples the initial spiral is embedded into an n-dimensional space using random matrix T followed by ReLU, and then projected back to the 2D space using T^{-1}. In examples above $n = 2, 3$ result in information loss where certain points of the manifold collapse into each other, while for $n = 15$ to 30 the transformation is highly non-convex.

D. *Inverted residuals*

Figure 2 shows the differences from traditional convolution to depthwise separable convolution to inverted residual block.

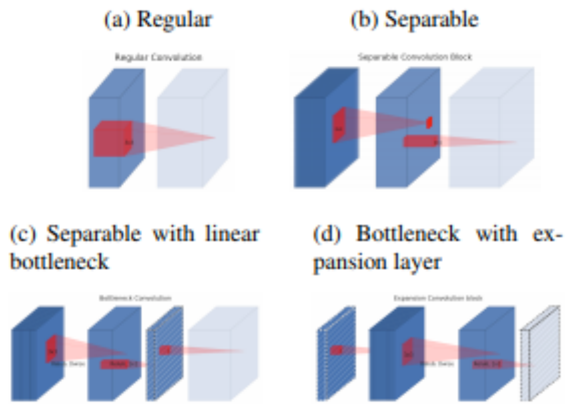

(a) Regular
Regular Convolution
(b) Separable
Separable Convolution Block
(c) Separable with linear bottleneck
Bottleneck Convolution
(d) Bottleneck with expansion layer
Expansion Convolution block

Figure 2: Evolution of separable convolution blocks. The diagonally hatched texture indicates layers that do not contain non-linearities. The last (lightly colored) layer indicates the beginning of the next block. Note: 2d and 2c are equivalent blocks when stacked. Best viewed in color.

For ResNet Block, first, through ReLU with 1x1 descending channel, then through ReLU with 3x3 space convolution, and then through ReLU recovery channel with 1x1 convolution, and add with input. The reason for the 1x1 convolution descent channel is to reduce the computational load, otherwise the 3x3 space convolution calculation in the middle is too large. So Residual block is hourglass, wide on both sides and narrow in the middle.

For Inverted Residual Block, through 1x1 convolution, the number of channels is first increased, then the 3x3 space convolution of Depthwise, and then the dimension is reduced by 1x1 convolution. The number of channels at both ends is very small, so the computational complexity of 1x1 convolution ascending channel or descending channel is not large, while the number of intermediate channels is large, but the computational complexity of Depthwise convolution is not large.

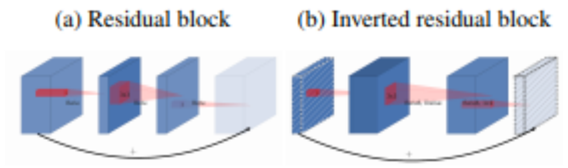


Figure 3: The difference between residual block [8, 30] and inverted residual. Diagonally hatched layers do not use non-linearities. We use thickness of each block to indicate its relative number of channels. Note how classical residuals connects the layers with high number of channels, whereas the inverted residuals connect the bottlenecks. Best viewed in color.

III. Conclusion

In this project, we propose a deep learning based food image recognition algorithm to recognize different food types and improve the accuracy. The convolutional block has a unique property that allows to separate the network expressiveness (encoded by expansion layers) from its capacity (encoded by bottleneck inputs). In the previous work, Liu, C. et al. has done the bounding box which did image segmentation first and then do the classification. This would effectively increase the testing accuracy since we can eliminate other non-related objects and keep the main part of the food. This requires manual cropping or automatic segmentation which makes the model even more complicated and will give a much better result. Extracting features in the last layer is a very common way to visualize the main feature component in each class and will give us very straightforward sense of why the CNN can understand each food.

Appendix

```
import keras
from keras.models import Model
from keras.layers import Input, Conv2D,
GlobalAveragePooling2D, Dropout
from keras.layers import Activation, BatchNormalization, add,
Reshape
#from keras.applications.mobilenet import relu6,
DepthwiseConv2D
from keras.utils.vis_utils import plot_model

from keras import backend as K


def _conv_block(inputs, filters, kernel, strides):
    """Convolution Block
    This function defines a 2D convolution operation with BN
and relu6.

    # Arguments
        inputs: Tensor, input tensor of conv layer.
        filters: Integer, the dimensionality of the output space.
        kernel: An integer or tuple/list of 2 integers, specifying
the
            width and height of the 2D convolution window.
        strides: An integer or tuple/list of 2 integers,
            specifying the strides of the convolution along the
width and height.
            Can be a single integer to specify the same value for
            all spatial dimensions.

    # Returns
        Output tensor.
    """

    channel_axis = 1 if K.image_data_format() ==
'channels_first' else -1

    x = Conv2D(filters, kernel, padding='same',
strides=strides)(inputs)
    x = BatchNormalization(axis=channel_axis)(x)
    return keras.layers.ReLU(6.)(x)


def _bottleneck(inputs, filters, kernel, t, s, r=False):
    """Bottleneck
    This function defines a basic bottleneck structure.

    # Arguments
        inputs: Tensor, input tensor of conv layer.
        filters: Integer, the dimensionality of the output space.
        kernel: An integer or tuple/list of 2 integers, specifying
the
            width and height of the 2D convolution window.
        t: Integer, expansion factor.
            t is always applied to the input size.
        s: An integer or tuple/list of 2 integers,specifying the
strides
            of the convolution along the width and height.Can be a
single
            integer to specify the same value for all spatial
dimensions.
        r: Boolean, Whether to use the residuals.

    # Returns
        Output tensor.
    """

    channel_axis = 1 if K.image_data_format() ==
'channels_first' else -1
    tchannel = K.int_shape(inputs)[channel_axis] * t

    x = _conv_block(inputs, tchannel, (1, 1), (1, 1))

    x = keras.layers.DepthwiseConv2D(kernel, strides=(s, s),
depth_multiplier=1, padding='same')(x)
    x = BatchNormalization(axis=channel_axis)(x)
    #x = Activation(relu6)(x)
    x = keras.layers.ReLU(6.)(x)

    x = Conv2D(filters, (1, 1), strides=(1, 1), padding='same')(x)
    x = BatchNormalization(axis=channel_axis)(x)

    if r:
        x = add([x, inputs])
    return x


def _inverted_residual_block(inputs, filters, kernel, t, strides,
n):
    """Inverted Residual Block
    This function defines a sequence of 1 or more identical
layers.

    # Arguments
```

```
        inputs: Tensor, input tensor of conv layer.
        filters: Integer, the dimensionality of the output space.
        kernel: An integer or tuple/list of 2 integers, specifying
the
            width and height of the 2D convolution window.
        t: Integer, expansion factor.
            t is always applied to the input size.
        s: An integer or tuple/list of 2 integers,specifying the
strides
            of the convolution along the width and height.Can be a
single
            integer to specify the same value for all spatial
dimensions.
        n: Integer, layer repeat times.
    # Returns
        Output tensor.
    """

    x = _bottleneck(inputs, filters, kernel, t, strides)

    for i in range(1, n):
        x = _bottleneck(x, filters, kernel, t, 1, True)

    return x


def MobileNetv2(input_shape, k):
    """MobileNetv2
    This function defines a MobileNetv2 architectures.

    # Arguments
        input_shape: An integer or tuple/list of 3 integers, shape
            of input tensor.
        k: Integer, number of classes.
    # Returns
        MobileNetv2 model.
    """

    inputs = Input(shape=input_shape)
    x = _conv_block(inputs, 32, (3, 3), strides=(2, 2))

    x = _inverted_residual_block(x, 16, (3, 3), t=1, strides=1,
n=1)
    x = _inverted_residual_block(x, 24, (3, 3), t=6, strides=2,
n=2)
    x = _inverted_residual_block(x, 32, (3, 3), t=6, strides=2,
n=3)
    x = _inverted_residual_block(x, 64, (3, 3), t=6, strides=2,
n=4)
    x = _inverted_residual_block(x, 96, (3, 3), t=6, strides=1,
n=3)
    x = _inverted_residual_block(x, 160, (3, 3), t=6, strides=2,
n=3)
    x = _inverted_residual_block(x, 320, (3, 3), t=6, strides=1,
n=1)

    x = _conv_block(x, 1280, (1, 1), strides=(1, 1))
    x = GlobalAveragePooling2D()(x)
    x = Reshape((1, 1, 1280))(x)
    x = Dropout(0.3, name='Dropout')(x)
    x = Conv2D(k, (1, 1), padding='same')(x)

    x = Activation('softmax', name='softmax')(x)
    output = Reshape((k,))(x)

    model = Model(inputs, output)
    plot_model(model, to_file='images/MobileNetv2.png',
show_shapes=True)

    return model


if __name__ == '__main__':
    MobileNetv2((224, 224, 3), 100)
```

IV. References

Lukas Bossard, M. G. (n.d.). *Food-101 – Mining Discriminative Components with Random Forests*. Retrieved from https://www.vision.ee.ethz.ch/datasets_extra/food-101/

Mark Sandler, A. H.-C. (2018, 1 13). MobileNetV2: Inverted Residuals and Linear Bottlenecks. *The IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4510-4520.