Chair Classification

by Squeeze-and-Excitation Networks (SE-Net)

- Data Preprocess
- SE-Net Introduction
- Model Design
- Model Training
- Test Result
- Conclusion

Data Preprocess

- Rescale (Covert input size to min image size)
- Grayscale (avoid overfitting of colors)

```
#check max and min of Length, max and min of width
l_max=0;w_max=0;l_min=0;w_min=0
for i in os.listdir('./data'):
    for j in os.listdir('./data/'+i):
        path='./data/'+i+'/'+j
        img = cv2.imread(path,cv2.IMREAD_GRAYSCALE)
        l_min=img.shape[0];w_min=img.shape[0]
        if(img.shape[0]>l_max):
            l_max=img.shape[0]
        if(img.shape[0]<l_min):
            l_min=img.shape[0]
        if(img.shape[1]>w_max):
            w_max=img.shape[1]
        if(img.shape[1]<w_min):
            w_min=img.shape[1]
print(l_max,l_min,w_max,w_min)
```

- Check the min size

```
4167 473 4167 473
```

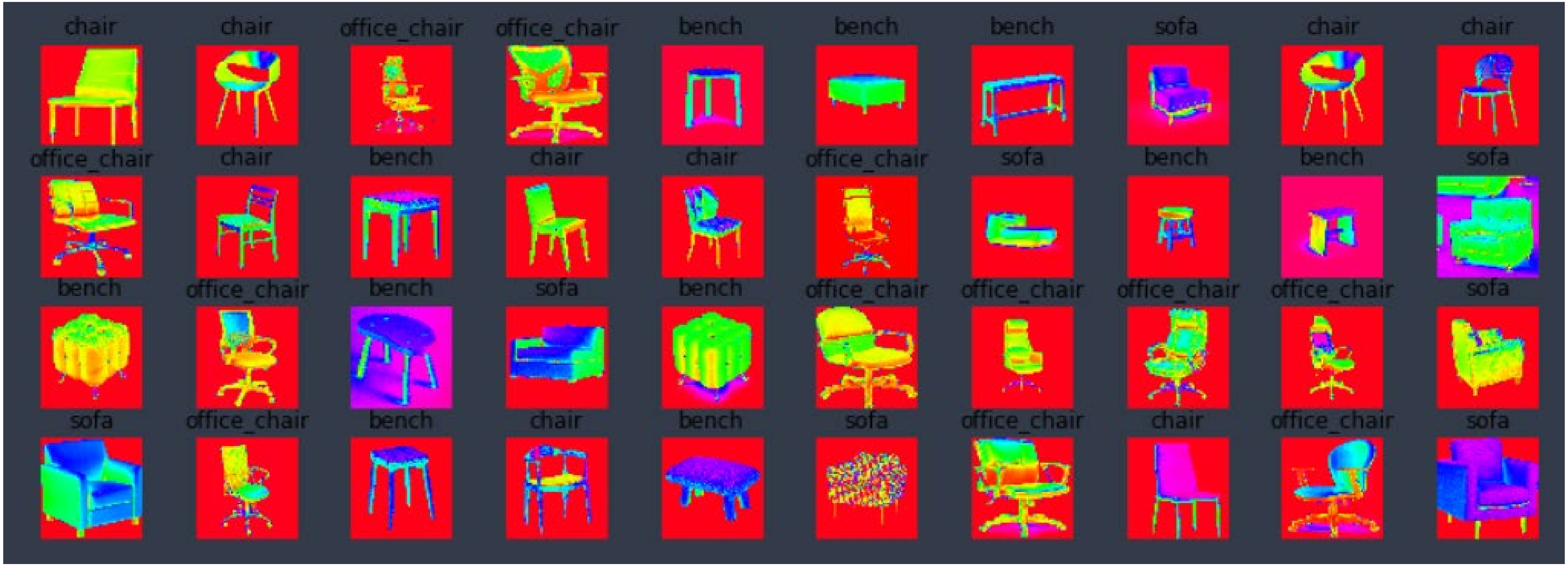
```
X=[]
for i in os.listdir('./data'):
    for j in os.listdir('./data/'+i):
        path='./data/'+i+'/'+j
        img = cv2.imread(path,cv2.IMREAD_GRAYSCALE)
        #print(img.shape)
        img = skimage.transform.resize(img, (l_min, w_min))
        img = np.asarray(img)
        X.append(img)
```

- Grayscale
- Rescale Image Size

Data Preprocess

- Rescale & Grayscale Result

```
x.shape

(1158, 473, 473, 1)
```



Data Preprocess

- Covert image type to float32 from float64 for computing resource reduction

```
train_images.astype('float32');test_images.astype('float32')
```

- Normalization

```
train_images, test_images = train_images / 255.0, test_images / 255.0
```

- Categorical of label

```
In [229]: test_labels[0]

array([3])
```

```
test_labels = to_categorical(np.array(test_labels[:, 0]))
test_labels[0]

array([0., 0., 0., 1.], dtype=float32)
```

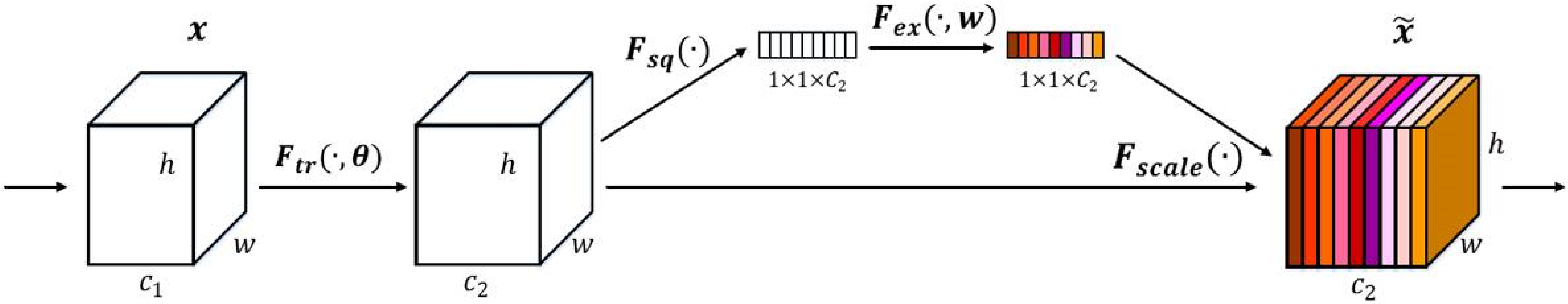
Function	Label	Categorical
bench	0	[1, 0, 0, 0]
chair	1	[0, 1, 0, 0]
office_chair	2	[0, 0, 1, 0]
sofa	3	[0, 0, 0, 1]

SE-Net Introduction

Jie Hu[1], Li Shen[2], Gang Sun[1]

[1] Momenta [2] University of Oxford

SENet: *Squeeze-and-Excitation Networks, Jie Hu, Li Shen, Gang Sun*



Squeeze	Excitation	Scale
• Shrinking feature maps $\in \mathbb{R}^{w\times h\times c_2}$ through spatial dimensions $(w \times h)$ • Global distribution of channel-wise responses	• Learning $W \in \mathbb{R}^{c_2\times c_2}$ to explicitly model channel-association • Gating mechanism to produce channel-wise weights	• Reweighting the feature maps $\in \mathbb{R}^{w\times h\times c_2}$

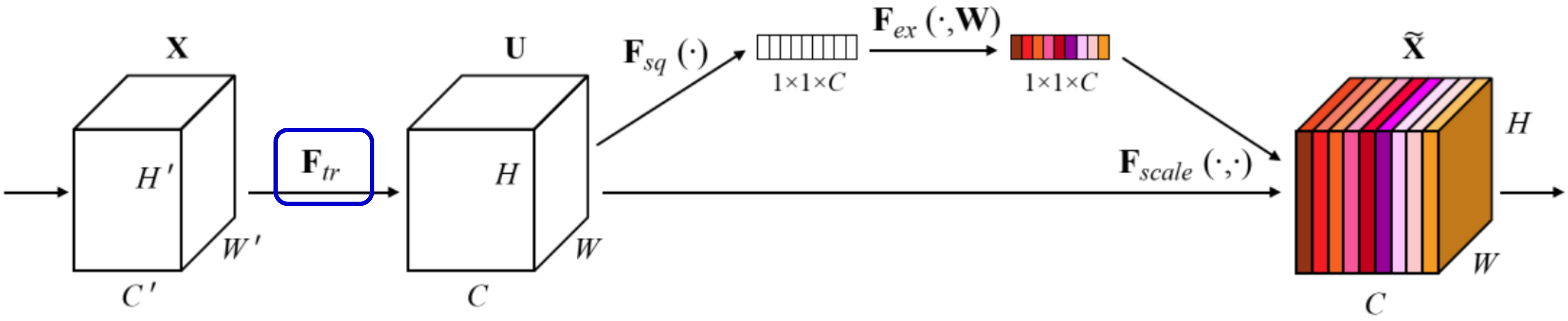
SENet (Squeeze-and-Excitation Networks)

Step-1: Feature transformation

- ✓ Ftr is the convolutional operator for transformation of X to U.
- ✓ This Ftr can be the residual block or Inception block
- ✓ where V=[v1, v2, …, vc] is the learnt set of filter kernels.

$$\mathbf{F}_{tr} : \mathbf{X} \rightarrow \mathbf{U}, \mathbf{X} \in \mathbb{R}^{H' \times W' \times C'}, \mathbf{U} \in \mathbb{R}^{H \times W \times C}$$

$$\mathbf{u}_c = \mathbf{v}_c * \mathbf{X} = \sum_{s=1}^{C'} \mathbf{v}_c^s * \mathbf{x}^s.$$

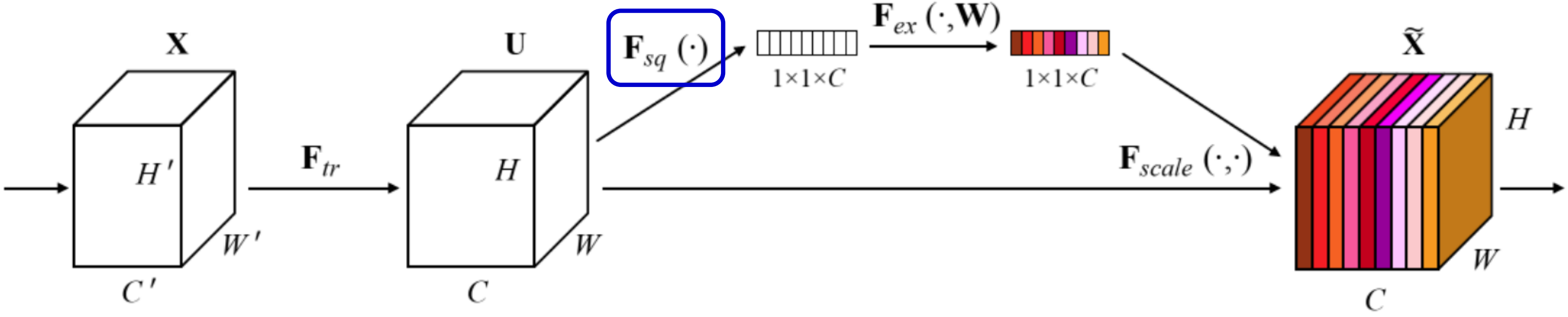


SENet (Squeeze-and-Excitation Networks)

Step-2: Squeeze (Global Average Pooling)

- ✓ The transformation output U can be interpreted as a collection of the local descriptors whose statistics are expressive for the whole image.
- ✓ It is proposed to squeeze global spatial information into a channel descriptor.
- ✓ This is achieved by using global average pooling to generate channel-wise statistics.

$$z_c = \mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i, j).$$

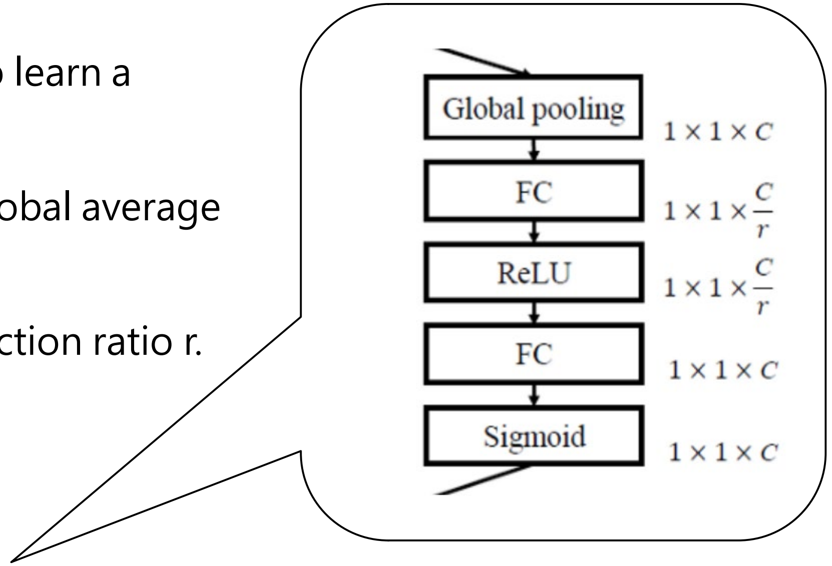


SENet (Squeeze-and-Excitation Networks)

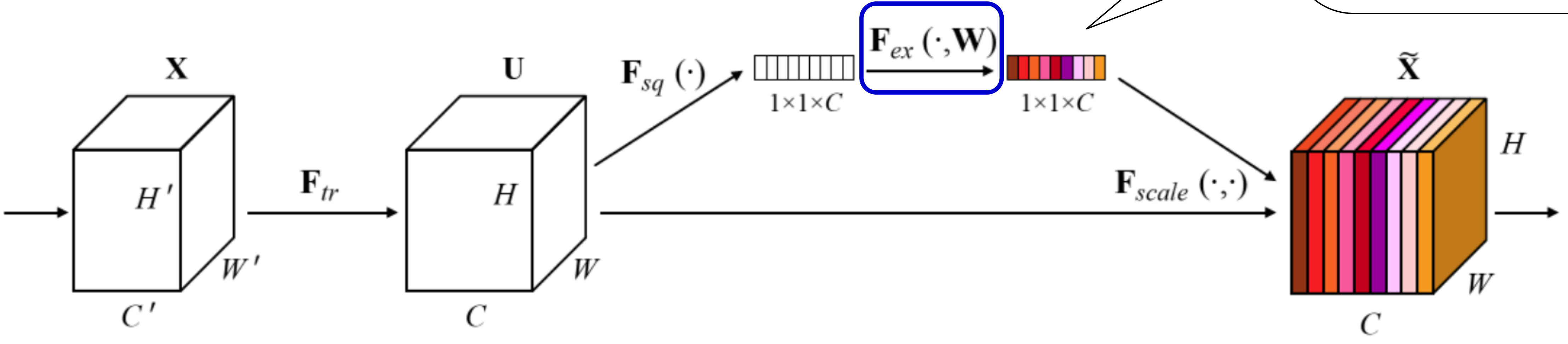
Step-3: Excitation (Adaptive Recalibration)

$$\frac{2}{r}\sum_{s=1}^{S} N_s \cdot {C_s}^2$$

- ✓ where δ is the ReLU function.
- ✓ A simple gating mechanism using sigmoid activation σ is used.
- ✓ An excitation operation is proposed to fully capture channel-wise dependencies, and to learn a nonlinear and non-mutually-exclusive relationship between channels.
- ✓ As we can see that there are W1 and W2, and the input z is a channel descriptor after global average pooling, there are two fully connected (FC) layers.
- ✓ The bottleneck with two FC layers are formed with dimensionality reduction using reduction ratio r.



$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \sigma(\mathbf{W}_2\delta(\mathbf{W}_1\mathbf{z})),$$



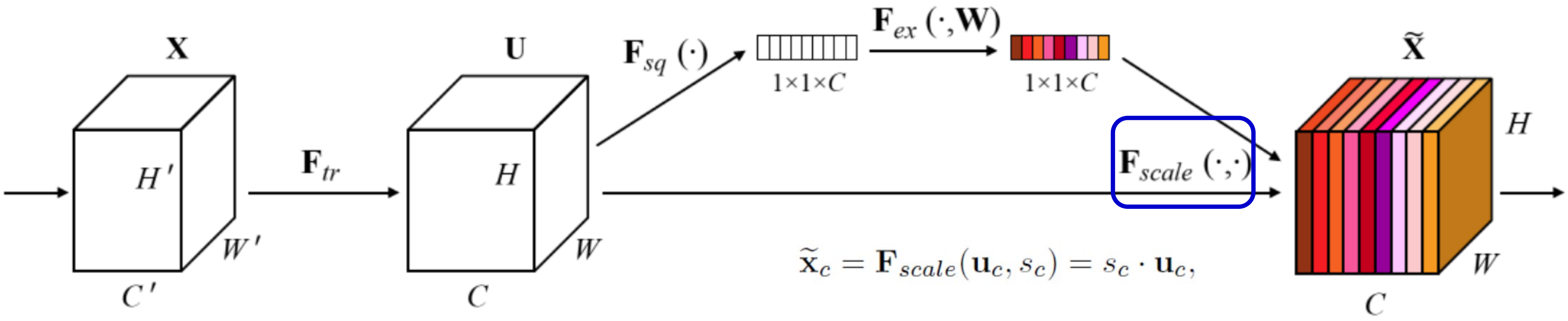
SENet (Squeeze-and-Excitation Networks)

Step-3: Excitation (Adaptive Recalibration)

- ✓ In particular, we found that setting r=16 achieved a good tradeoff between accuracy and complexity and consequently, we used this value for all experiments.
- ✓ The number of additional parameters introduced depends on r as above where S refers to the number of stages (where each stage refers to the collection of blocks operating on feature maps of a common spatial dimension), Cs denotes the dimension of the output channels and Ns denotes the repeated block number for stage s.
- ✓ The final output of the block is obtained by rescaling the transformation output U with the activations, as shown above.
- ✓ Fscare refers to channel-wise multiplication between the feature map and the scalar Sc

Table 5: Single-crop error rates (%) on the ImageNet validation set and corresponding model sizes for the SE-ResNet-50 architecture at different reduction ratios r. Here original refers to ResNet-50.

Ratio r	**top-1 err.**	**top-5 err.**	**model size (MB)**
4	23.21	6.63	137
8	23.19	6.64	117
16	23.29	6.62	108
32	23.40	6.77	103
original	24.80	7.48	98



$$\widetilde{\mathbf{x}}_c = \mathbf{F}_{scale}(\mathbf{u}_c, s_c) = s_c \cdot \mathbf{u}_c,$$

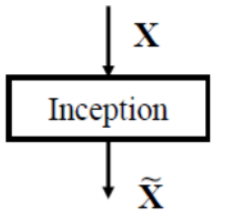
SENet (Squeeze-and-Excitation Networks)

- **SE-INCEPTION AND SE-RESNET**

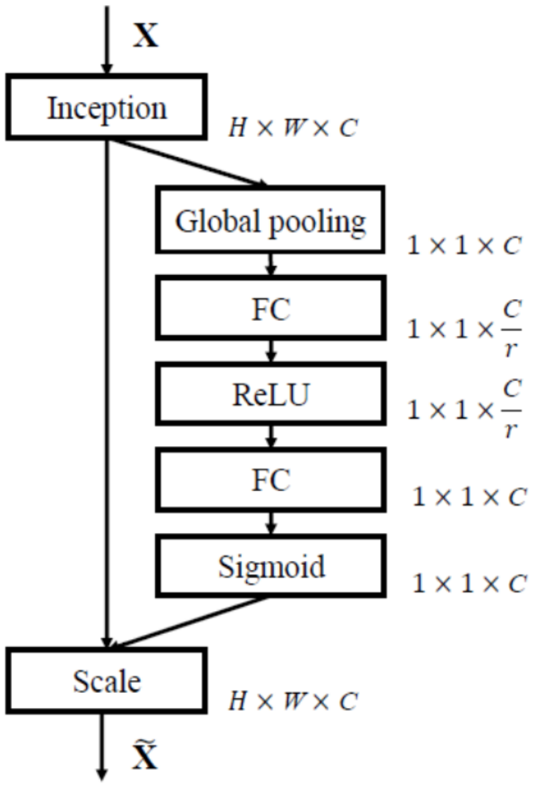
✓ SE block can be added to both Inception and ResNet block easily as SE-Inception and SE-ResNet.

✓ Particularly in SE-ResNet, squeeze and excitation both act before summation with the identity branch.

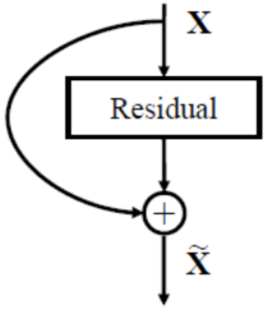
✓ More variants that integrate with ResNeXt, Inception-ResNet, MobileNetV1 and ShuffleNet V1 can be constructed by following the similar schemes.



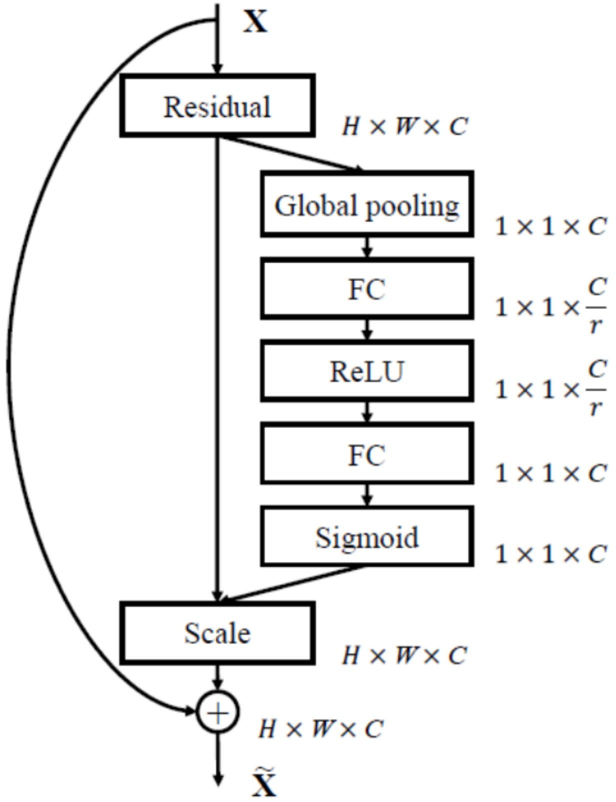
Inception Module



SE-Inception Module



ResNet Module



SE-ResNet Module

SENet (Squeeze-and-Excitation Networks)

- Single-Crop Error Rates on ImageNet Validation Set

	original		re-implementation			SENet		
	top-1 err.	top-5 err.	top-1err.	top-5 err.	GFLOPs	top-1 err.	top-5 err.	GFLOPs
ResNet-50 [10]	24.7	7.8	24.80	7.48	3.86	$23.29_{(1.51)}$	$6.62_{(0.86)}$	3.87
ResNet-101 [10]	23.6	7.1	23.17	6.52	7.58	$22.38_{(0.79)}$	$6.07_{(0.45)}$	7.60
ResNet-152 [10]	23.0	6.7	22.42	6.34	11.30	$21.57_{(0.85)}$	$5.73_{(0.61)}$	11.32
ResNeXt-50 [47]	22.2	-	22.11	5.90	4.24	$21.10_{(1.01)}$	$5.49_{(0.41)}$	4.25
ResNeXt-101 [47]	21.2	5.6	21.18	5.57	7.99	$20.70_{(0.48)}$	$5.01_{(0.56)}$	8.00
VGG-16 [39]	-	-	27.02	8.81	15.47	$25.22_{(1.80)}$	$7.70_{(1.11)}$	15.48
BN-Inception [16]	25.2	7.82	25.38	7.89	2.03	$24.23_{(1.15)}$	$7.14_{(0.75)}$	2.04
Inception-ResNet-v2 [42]	$19.9^{\dagger}$	$4.9^{\dagger}$	20.37	5.21	11.75	$19.80_{(0.57)}$	$4.79_{(0.42)}$	11.76

	original		re-implementation				SENet			
	top-1 err.	top-5 err.	top-1 err.	top-5 err.	MFLOPs	Million Parameters	top-1 err.	top-5 err.	MFLOPs	Million Parameters
MobileNet [13]	29.4	-	29.1	10.1	569	4.2	$25.3_{(3.8)}$	$7.9_{(2.2)}$	572	4.7
ShuffleNet [52]	34.1	-	33.9	13.6	140	1.8	$31.7_{(2.2)}$	$11.7_{(1.9)}$	142	2.4

SENet (Squeeze-and-Excitation Networks)

- ILSVRC 2017 Classification Competition

	224 × 224		320 × 320 / 299 × 299	
	top-1 err.	top-5 err.	top-1 err.	top-5 err.
ResNet-152 [10]	23.0	6.7	21.3	5.5
ResNet-200 [11]	21.7	5.8	20.1	4.8
Inception-v3 [44]	-	-	21.2	5.6
Inception-v4 [42]	-	-	20.0	5.0
Inception-ResNet-v2 [42]	-	-	19.9	4.9
ResNeXt-101 (64 × 4d) [47]	20.4	5.3	19.1	4.4
DenseNet-264 [14]	22.15	6.12	-	-
Attention-92 [46]	-	-	19.5	4.8
Very Deep PolyNet [51] †	-	-	18.71	4.25
PyramidNet-200 [8]	20.1	5.4	19.2	4.7
DPN-131 [5]	19.93	5.12	18.55	4.16
SENet-154	**18.68**	**4.47**	**17.28**	**3.79**
NASNet-A (6@4032) [55] †	-	-	17.3‡	3.8‡
SENet-154 (post-challenge)	-	-	**16.88**‡	**3.58**‡

- Scene Classification

	top-1 err.	top-5 err.
Places-365-CNN [37]	41.07	11.48
ResNet-152 (ours)	41.15	11.61
SE-ResNet-152	**40.37**	**11.01**

- Object Detection on COCO

	AP@IoU=0.5	AP
ResNet-50	45.2	25.1
SE-ResNet-50	46.8	26.4
ResNet-101	48.4	27.2
SE-ResNet-101	49.2	27.9

Model Design

- Squeeze

$$z_c = \mathbf{F}_{sq}(\mathbf{u}_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i, j).$$

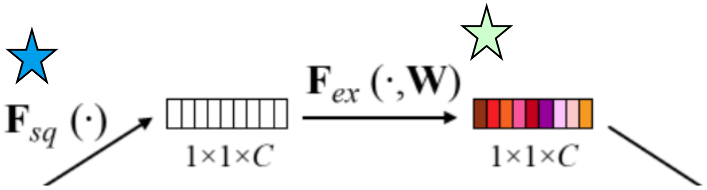
```
def squeeze(init):
    se = GlobalAveragePooling2D()(init)
    return(se)
```

- Excitation

$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \sigma(\mathbf{W}_2 \delta(\mathbf{W}_1 \mathbf{z})),$$

```
def excitation(init, ratio=16):
    filters = init._keras_shape[-1]
    se = Dense(filters // ratio, activation='relu', use_bias=False)(se)
    se = Dense(filters, activation='sigmoid', use_bias=False)(se)
    return(se)
```

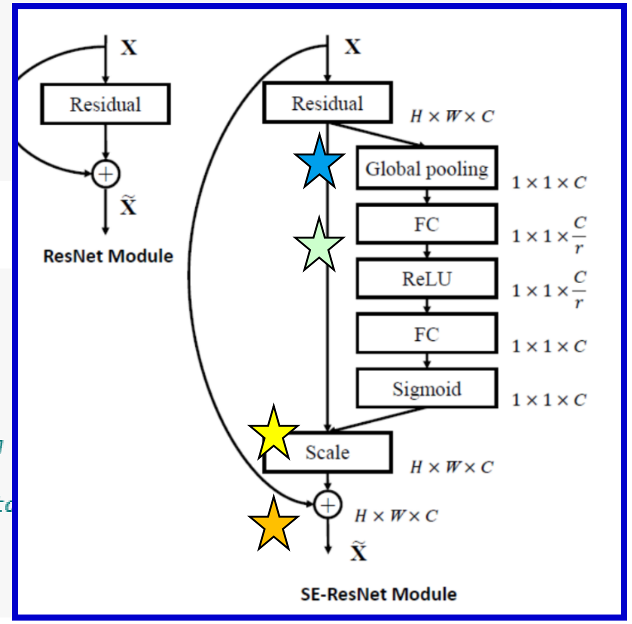
- Squeeze & Excitation Block




```
def squeeze_excite_block(input, ratio=16):
    init = input
    filters = init._keras_shape[-1]
    se_shape = (1, 1, filters)

    se = GlobalAveragePooling2D()(init) # Squeeze: Global Information Embedding
    se = Reshape(se_shape)(se)
    se = Dense(filters // ratio, activation='relu', use_bias=False)(se) # Excita
    se = Dense(filters, activation='sigmoid', use_bias=False)(se)
    x = multiply([init, se])
    return x
```

- ResNet Block

```
def resnet_block(input, filters, k=1, strides=(1, 1)):
    init = input
    channel_axis = -1

    x = BatchNormalization(axis=channel_axis)(input)
    x = Activation('relu')(x)

    if strides != (1, 1) or init._keras_shape[channel_axis] != filters * k:
        init = Conv2D(filters * k, (1, 1), padding='same', kernel_initializer='he_normal',
                      use_bias=False, strides=strides)(x)

    x = Conv2D(filters * k, (3, 3), padding='same', kernel_initializer='he_normal',
               use_bias=False, strides=strides)(x)
    x = BatchNormalization(axis=channel_axis)(x)
    x = Activation('relu')(x)

    x = Conv2D(filters * k, (3, 3), padding='same', kernel_initializer='he_normal',
               use_bias=False)(x)

    # squeeze and excite block
    x = squeeze_excite_block(x)

    m = add([x, init])
    return m
```



Model Design

- **SE-ResNet**

Total params: 9,777,,472

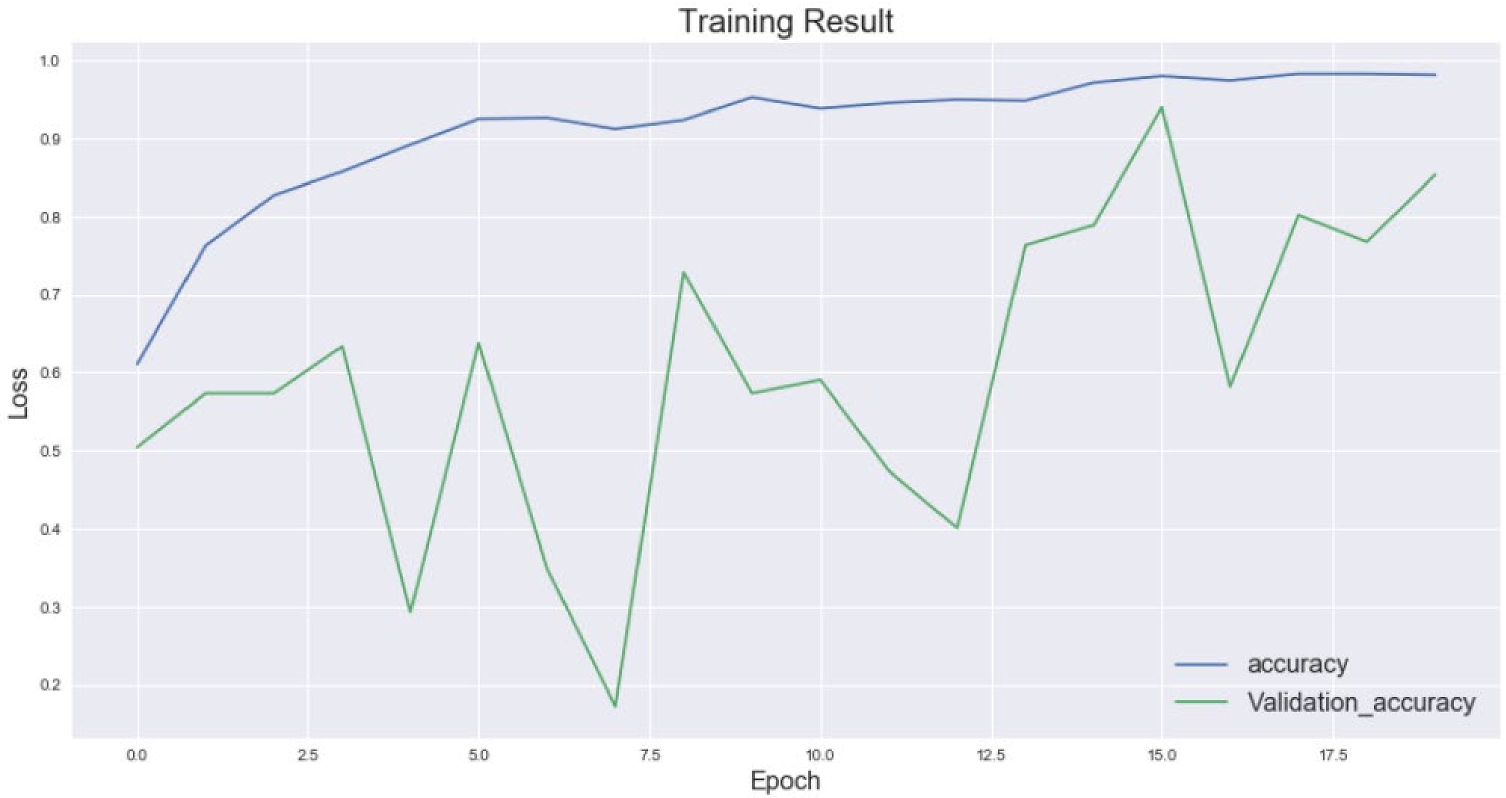
```python
def create_se_resnet(classes, img_input, include_top, initial_conv_filters, filters,
                     depth, width, weight_decay, pooling):

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

    x = Conv2D(initial_conv_filters, (7, 7), padding='same', use_bias=False, strides=(2, 2),
               kernel_initializer='he_normal', kernel_regularizer=l2(weight_decay))(img_input)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)

    for i in range(N[0]):
            x = resnet_block(x, filters[0], width)

    for k in range(1, len(N)):
            x = resnet_block(x, filters[k], width, strides=(2, 2))

    for i in range(N[k] - 1):
            x = resnet_block(x, filters[k], width)

    x = BatchNormalization(axis=channel_axis)(x)
    x = Activation('relu')(x)


    x = GlobalAveragePooling2D()(x)
    x = Dense(classes, use_bias=False, kernel_regularizer=l2(weight_decay),
                activation='softmax')(x)

    return x
```

```
model created
________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================================
input_4 (InputLayer)            (None, 473, 473, 1)  0
________________________________________________________________________________________________
conv2d_49 (Conv2D)              (None, 237, 237, 64) 3136        input_4[0][0]
________________________________________________________________________________________________
max_pooling2d_4 (MaxPooling2D)  (None, 119, 119, 64) 0           conv2d_49[0][0]
________________________________________________________________________________________________
batch_normalization_40 (BatchNo (None, 119, 119, 64) 256         max_pooling2d_4[0][0]
________________________________________________________________________________________________
activation_40 (Activation)      (None, 119, 119, 64) 0           batch_normalization_40[0][0]
________________________________________________________________________________________________
conv2d_50 (Conv2D)              (None, 119, 119, 64) 36864       activation_40[0][0]
________________________________________________________________________________________________
batch_normalization_41 (BatchNo (None, 119, 119, 64) 256         conv2d_50[0][0]
________________________________________________________________________________________________
activation_41 (Activation)      (None, 119, 119, 64) 0           batch_normalization_41[0][0]
________________________________________________________________________________________________
conv2d_51 (Conv2D)              (None, 119, 119, 64) 36864       activation_41[0][0]
________________________________________________________________________________________________
```

```
________________________________________________________________________________________________
dense_50 (Dense)                (None, 1, 1, 32)     16384       reshape_24[0][0]
________________________________________________________________________________________________
dense_51 (Dense)                (None, 1, 1, 512)    16384       dense_50[0][0]
________________________________________________________________________________________________
multiply_24 (Multiply)          (None, 15, 15, 512)  0           conv2d_64[0][0]
                                                                 dense_51[0][0]
________________________________________________________________________________________________
add_24 (Add)                    (None, 15, 15, 512)  0           multiply_24[0][0]
                                                                 add_23[0][0]
________________________________________________________________________________________________
batch_normalization_52 (BatchNo (None, 15, 15, 512)  2048        add_24[0][0]
________________________________________________________________________________________________
activation_52 (Activation)      (None, 15, 15, 512)  0           batch_normalization_52[0][0]
________________________________________________________________________________________________
global_average_pooling2d_28 (Gl (None, 512)          0           activation_52[0][0]
________________________________________________________________________________________________
dense_52 (Dense)                (None, 4)            2048        global_average_pooling2d_28[0][0]
================================================================================================
Total params: 9,777,472
Trainable params: 9,771,200
Non-trainable params: 6,272
________________________________________________________________________________________________
```

Model Training

- Epochs = 20, Train: 75%, Validation: 25%, Learning Rate: 0.001, optimizer: Adam (beta_1=0.9, beta_2=0.999, epsilon=1e-08)



- Training time: 6hr

```
Epoch 20/20
694/694 [==============================] - 1012s 1s/step - loss: 0.0666 - acc: 0.9813 - val_loss: 0.4822 - val_acc: 0.8534
```

NVIDIA
控制面板
版本 446.14
GeForce MX150

Test Result

- Training: 80% (926 cnt), Test: 20% (232 cnt)

```
In [228]:
from sklearn.model_selection import train_test_split
train_images, test_images, train_labels, test_labels = train_test_split(X, y, test_size = 0.2, random_state
print(train_images.shape, train_labels.shape, test_images.shape, test_labels.shape)
```

```
(926, 473, 473, 1) (926, 1) (232, 473, 473, 1) (232, 1)
```

- Accuracy: 85.34%

```
score = model.evaluate(test_images, test_labels, verbose=0)
print('accuracy: ',score[1])
print('loss: ',score[0])
```

```
accuracy:  0.853448275862069
loss:  0.46818723144202395
```

Test Result

- Confusion Matrix:

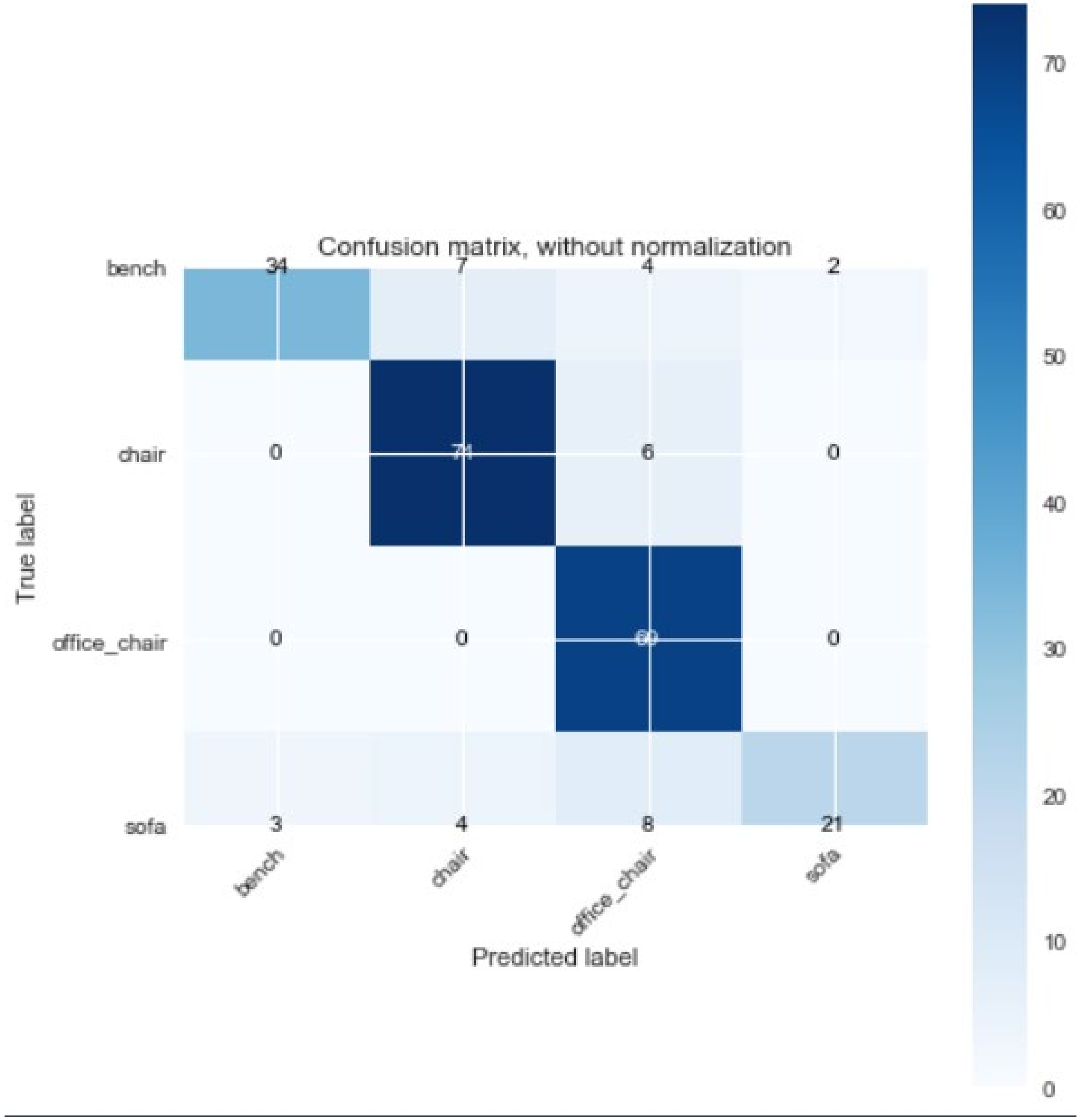
  Worse prediction pair (True/Predict):

  - Bench/Chair (7 cnt, 15%), Sofa/Office_Chair (8 cnt, 22%)

```
In [223]: class_names=['bench', 'chair', 'office_chair', 'sofa']
```

```
In [210]: cm = confusion_matrix(y_true, predict)
          cm

array([[34,  7,  4,  2],
       [ 0, 74,  6,  0],
       [ 0,  0, 69,  0],
       [ 3,  4,  8, 21]], dtype=int64)
```

```
In [212]: cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
          cm

array([[0.72, 0.15, 0.09, 0.04],
       [0.  , 0.93, 0.07, 0.  ],
       [0.  , 0.  , 1.  , 0.  ],
       [0.08, 0.11, 0.22, 0.58]])
```

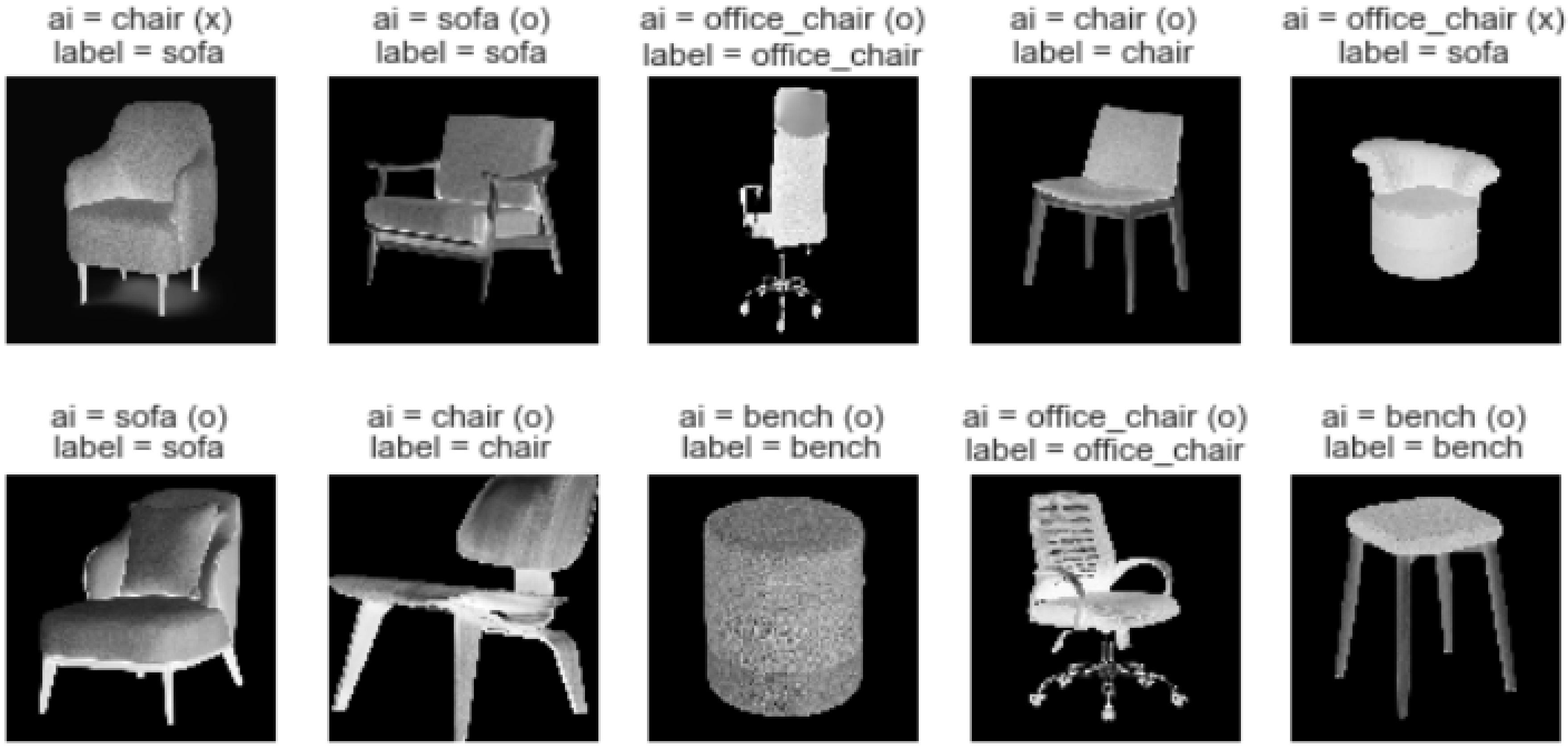


Test Result

- Predict Error Case:

Test Result

- 10 Predict Case:



Conclusion

- The function of chair is predictable (acc: 85%)
- Training time is 6hr, advanced device (e.g. Nvidia A100, V100...) is necessary.
- If we have advanced device, ablation study can been performed.

  (Advanced Mode, Cosine Learning Rate, Optimizers, et al.)