
Densenet-Tensorflow

Tensorflow implementation of Densenet using **Cifar10, MNIST** * The code that implements *this paper* is **Densenet.py** * There is a *slight difference*, I used **AdamOptimizer**

If you want to see the **original author's code** or **other implementations**, please refer to this link

Requirements

- Tensorflow 1.x
- Python 3.x
- tflearn (If you are easy to use **global average pooling**, you should install **tflearn**)

```
1 However, I implemented it using tf.layers, so don't worry
```

Issue

- I used **tf.contrib.layers.batch_norm**

```
1 def Batch_Normalization(x, training, scope):
2     with arg_scope([batch_norm],
3                     scope=scope,
4                     updates_collections=None,
5                     decay=0.9,
6                     center=True,
7                     scale=True,
8                     zero_debias_moving_mean=True) :
9         return tf.cond(training,
10                          lambda : batch_norm(inputs=x, is_training=
11                                              training, reuse=None),
12                          lambda : batch_norm(inputs=x, is_training=
13                                              training, reuse=True))
```

- If not enough GPU memory, Please edit the code

```
1 with tf.Session() as sess : NO
2 with tf.Session(config=tf.ConfigProto(allow_soft_placement=True)) as
   sess : OK
```

- <https://github.com/taki0112/Densenet-Tensorflow/issues/10>

Idea

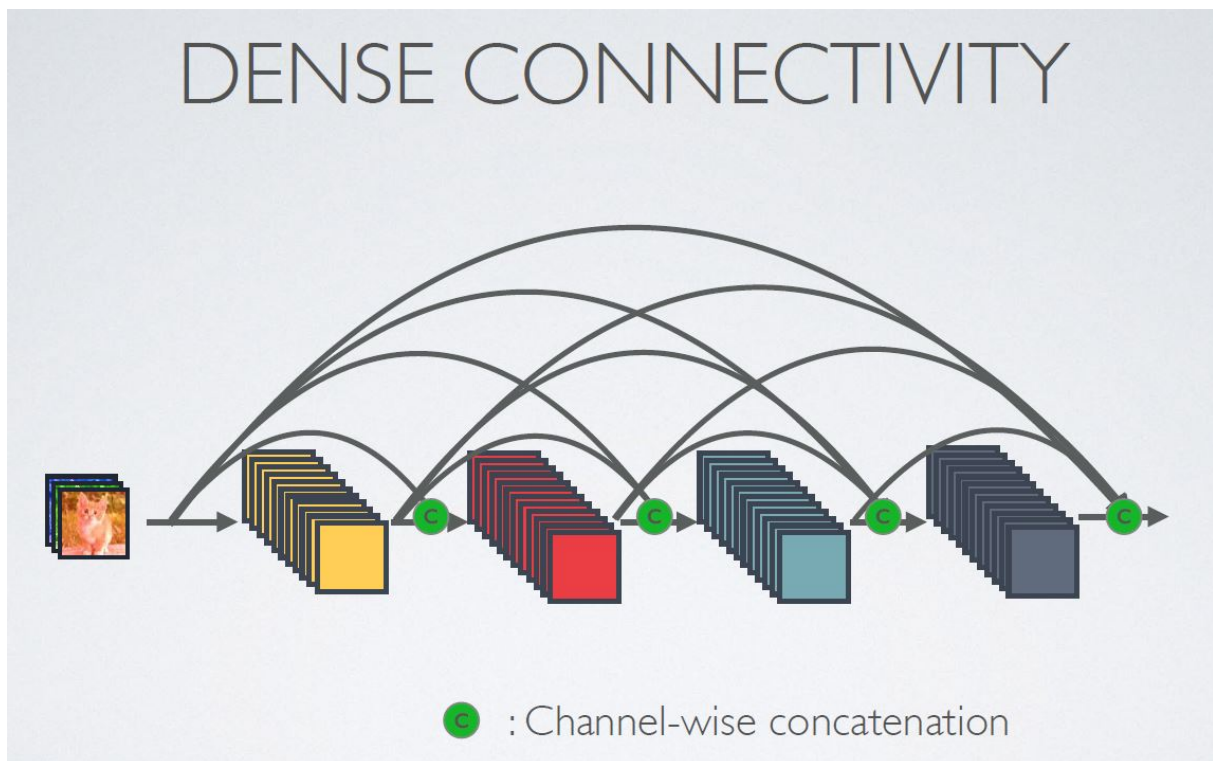
What is the “Global Average Pooling” ?

```
1  def Global_Average_Pooling(x, stride=1) :  
2      width = np.shape(x)[1]  
3      height = np.shape(x)[2]  
4      pool_size = [width, height]  
5      return tf.layers.average_pooling2d(inputs=x, pool_size=  
6          pool_size, strides=stride)  
# The stride value does not matter
```

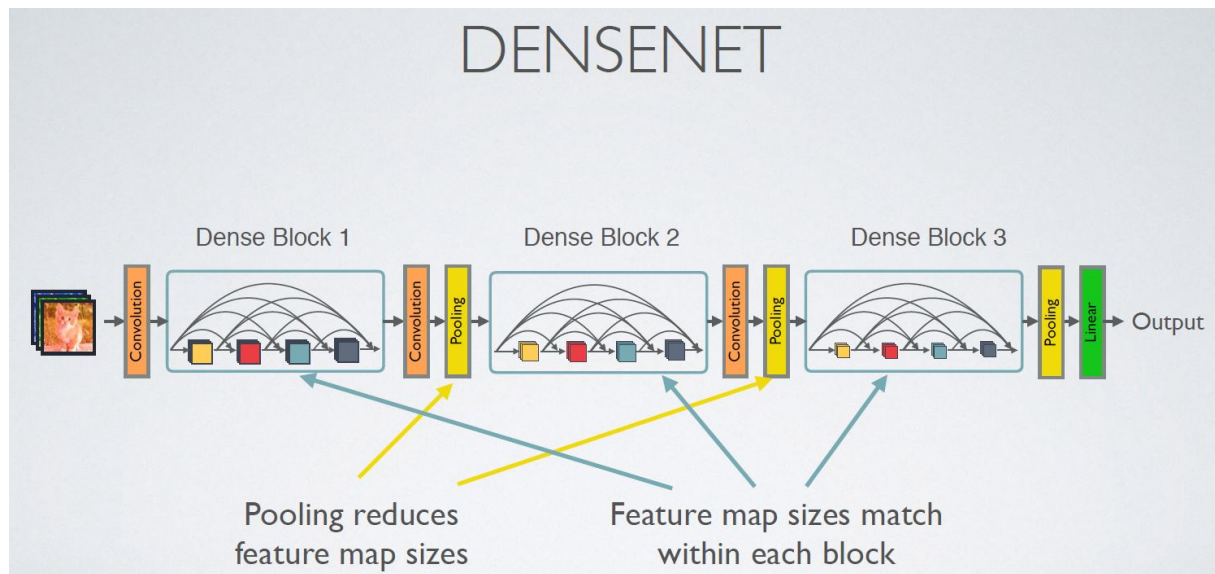
- If you use tflearn, please refer to this link

```
1  def Global_Average_Pooling(x):  
2      return tflearn.layers.conv.global_avg_pool(x, name='  
    Global_avg_pooling')
```

What is the “Dense Connectivity” ?

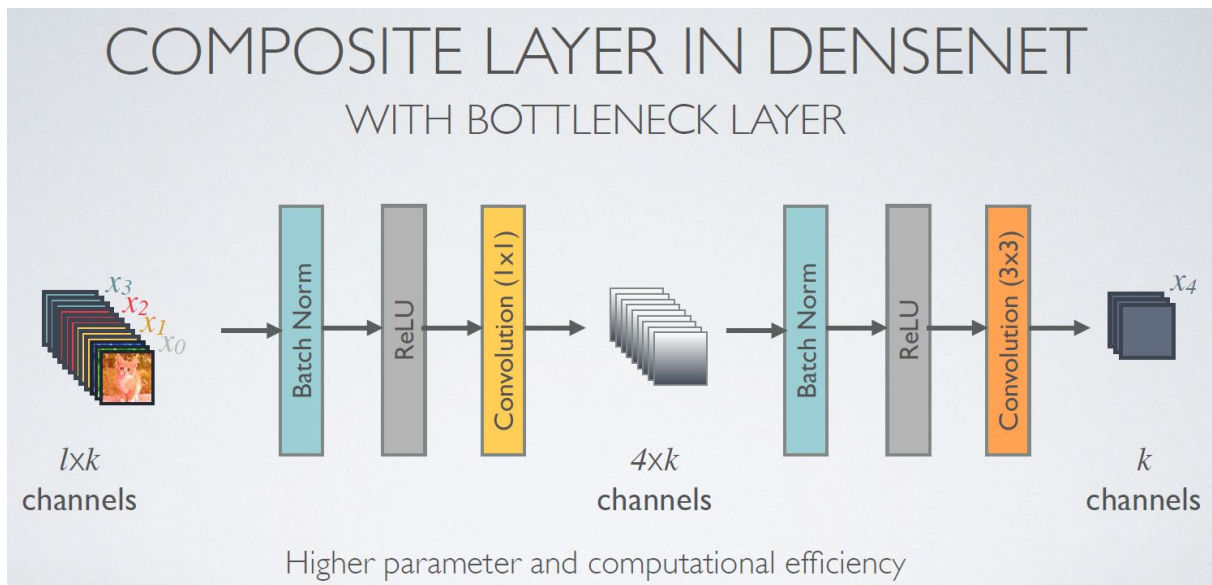


What is the “Densenet Architecture” ?



```
1  def Dense_net(self, input_x):
2      x = conv_layer(input_x, filter=2 * self.filters, kernel=[7,7],
3                      stride=2, layer_name='conv0')
4      x = Max_Pooling(x, pool_size=[3,3], stride=2)
5
6      x = self.dense_block(input_x=x, nb_layers=6, layer_name='
7                          dense_1')
8      x = self.transition_layer(x, scope='trans_1')
9
10     x = self.dense_block(input_x=x, nb_layers=12, layer_name='
11                         dense_2')
12     x = self.transition_layer(x, scope='trans_2')
13
14     x = self.dense_block(input_x=x, nb_layers=48, layer_name='
15                         dense_3')
16     x = self.transition_layer(x, scope='trans_3')
17
18     x = self.dense_block(input_x=x, nb_layers=32, layer_name='
19                         dense_final')
20
21     x = Batch_Normalization(x, training=self.training, scope='
22                             linear_batch')
23     x = Relu(x)
24     x = Global_Average_Pooling(x)
25     x = Linear(x)
26
27     return x
```

What is the “Dense Block” ?



```
1  def dense_block(self, input_x, nb_layers, layer_name):
2      with tf.name_scope(layer_name):
3          layers_concat = list()
4          layers_concat.append(input_x)
5
6          x = self.bottleneck_layer(input_x, scope=layer_name + '_bottleN_' + str(0))
7
8          layers_concat.append(x)
9
10         for i in range(nb_layers - 1):
11             x = Concatenation(layers_concat)
12             x = self.bottleneck_layer(x, scope=layer_name + '_bottleN_' + str(i + 1))
13             layers_concat.append(x)
14
15         x = Concatenation(layers_concat)
16
17         return x
```

What is the “Bottleneck Layer” ?

```
1  def bottleneck_layer(self, x, scope):
2      with tf.name_scope(scope):
3          x = Batch_Normalization(x, training=self.training, scope=scope + '_batch1')
4          x = Relu(x)
```

```

5         x = conv_layer(x, filter=4 * self.filters, kernel=[1,1],
6                         layer_name=scope+'_conv1')
7         x = Drop_out(x, rate=dropout_rate, training=self.training)
8         x = Batch_Normalization(x, training=self.training, scope=
9                                 scope+'_batch2')
10        x = Relu(x)
11        x = conv_layer(x, filter=self.filters, kernel=[3,3],
12                        layer_name=scope+'_conv2')
13        x = Drop_out(x, rate=dropout_rate, training=self.training)
14
15        return x

```

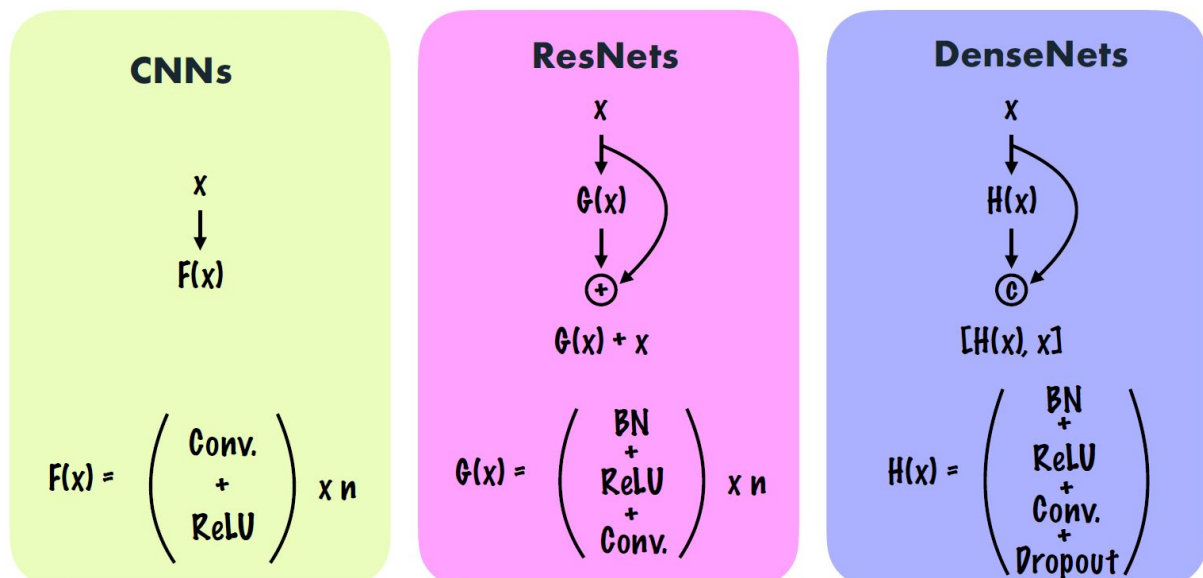
What is the “Transition Layer” ?

```

1    def transition_layer(self, x, scope):
2        with tf.name_scope(scope):
3            x = Batch_Normalization(x, training=self.training, scope=
4                                    scope+'_batch1')
5            x = Relu(x)
6            x = conv_layer(x, filter=self.filters, kernel=[1,1],
7                            layer_name=scope+'_conv1')
8            x = Drop_out(x, rate=dropout_rate, training=self.training)
9            x = Average_pooling(x, pool_size=[2,2], stride=2)
10
11        return x

```

Compare Structure (CNN, ResNet, DenseNet)

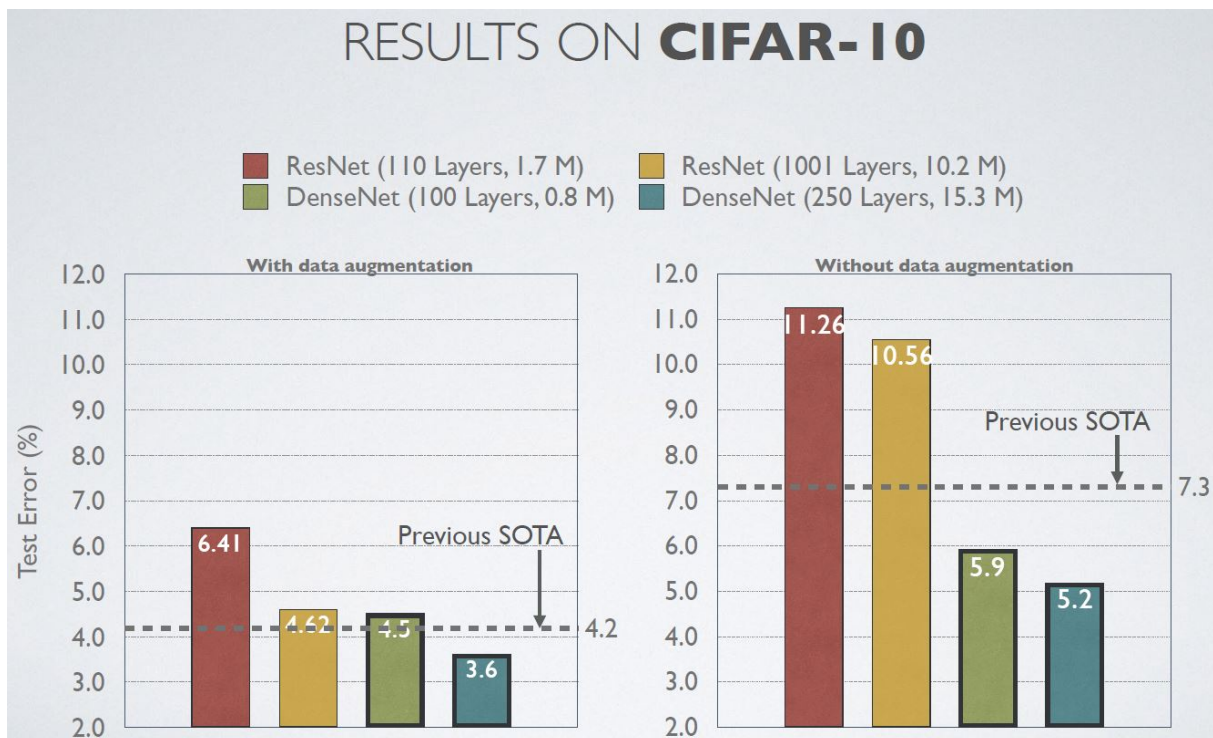


Results

- (**MNIST**) The highest test accuracy is **99.2%** (This result does **not use dropout**)
- The number of dense block layers is fixed to **4**

```
1     for i in range(self.nb_blocks) :
2         # original : 6 -> 12 -> 48
3
4         x = self.dense_block(input_x=x, nb_layers=4, layer_name='dense_'
5                               +str(i))
6         x = self.transition_layer(x, scope='trans_'+str(i))
```

CIFAR-10



CIFAR-100

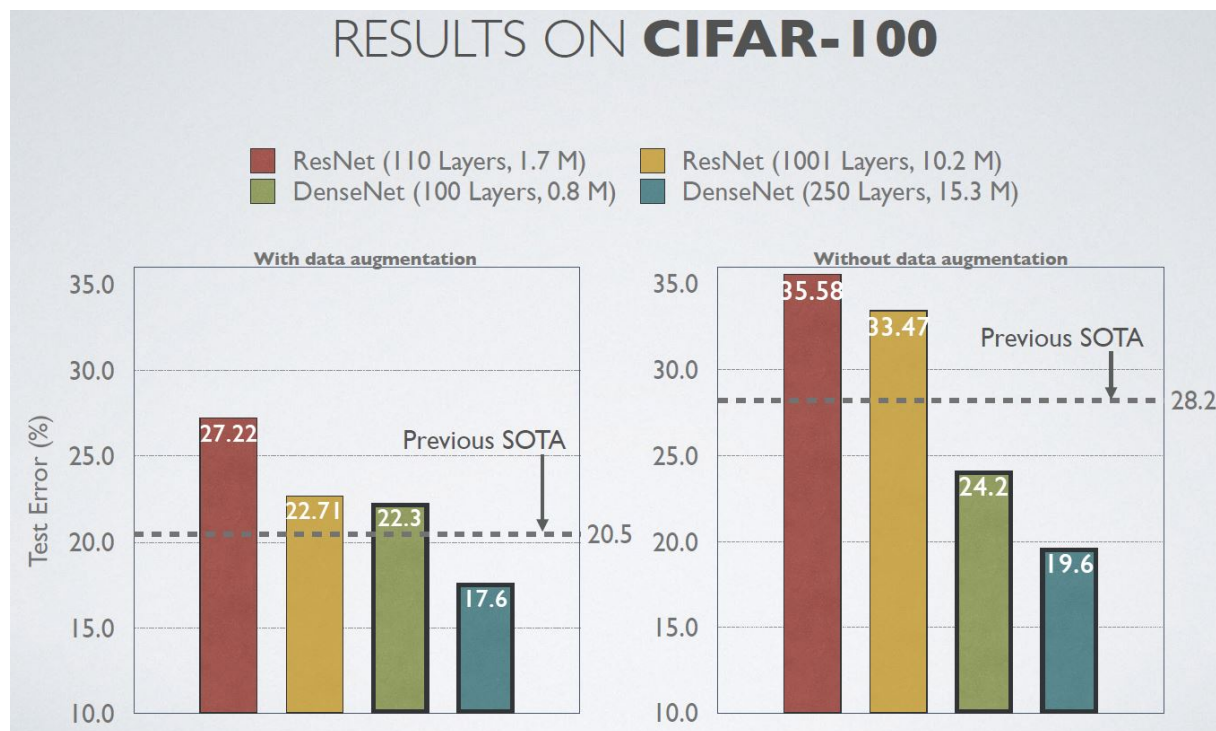
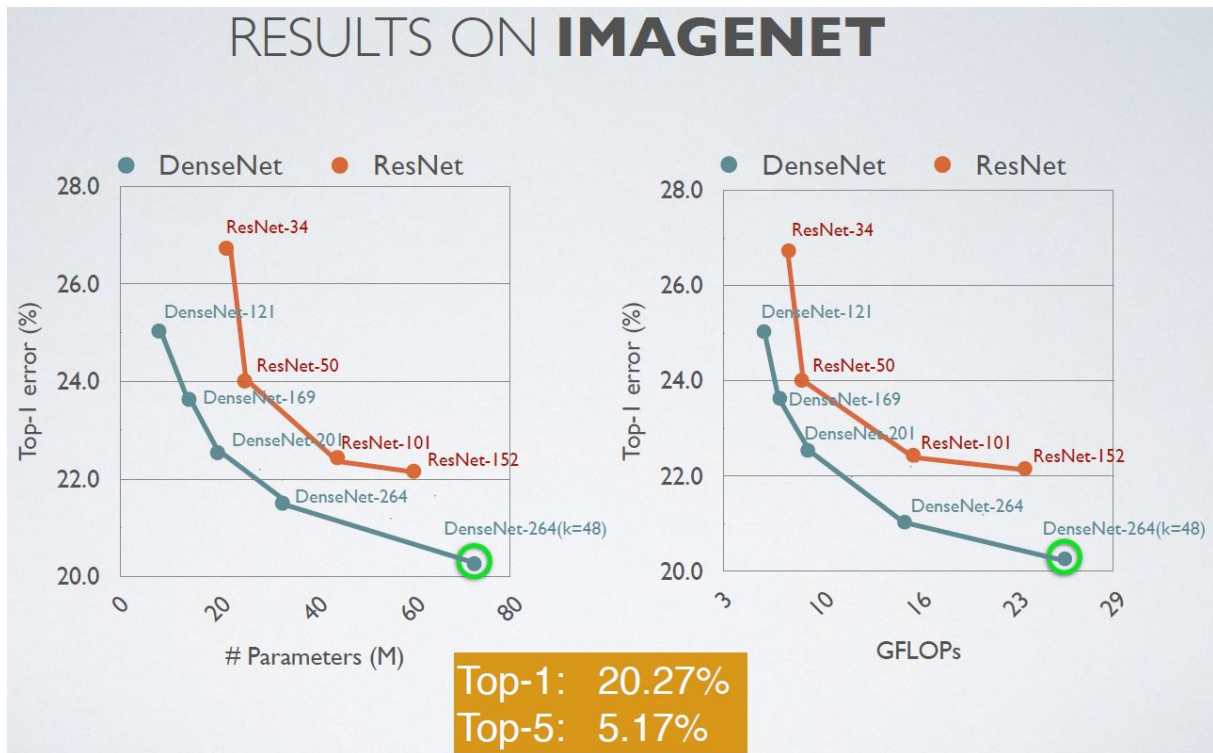


Image Net



Related works

- ResNeXt-Tensorflow
- SENet-Tensorflow
- ResNet-Tensorflow

References

- Korean
- English
- Classification Datasets Results

Author

Junho Kim