Convolutional Neural Network (CNN)
Notable Architectures II

M. R. Hasan

Deep Learning
Notable Architectures

- We will study the following architectures that are winners or runner-ups in the ImageNet challenge.
  - AlexNet (2012 winner)
  - VGGNet (2014 runner-up)
  - GoogLeNet (Inception-v1) (2014 winner)
  - ResNet (2015 winner)
  - SENet (2017 winner)

A more comprehensive analysis on a larger set of architectures is given:


• First we will describe what problem motivated the invention of the ResNet architecture.
• The modern CNN architectures that we have studied so far share a common principle.
• Deeper networks improve classification performance.
• We have seen that deep networks are created by stacking computational layers (e.g., VGG block, Inception block).
• It facilitates learning more expressive features.
• Thus, the key takeaway message is that network depth is the main factor towards creating powerful architectures.

• However, the effort to build deeper networks soon hit a roadblock.

• It was found that it was challenging to optimally train deeper networks.
With the network depth increasing, training accuracy gets saturated and then degrades rapidly.
• Researchers argued that the problem was caused **not due to overfitting**.

• Adding more layers increased **training error**!

• Also the vanishing gradient problem was handled using Batch Normalization and good initializers.

• Then, what caused **training degradation**?

The problem was identified as **training optimization problem** that was due to the **increased depth** of the network.
• The 34-layer plain network has **higher training error** throughout the whole training procedure.

• This is surprising because the **solution space of the 18-layer plain network is a subspace** of that of the 34-layer one.

The solution space of the n-layer is **subspace** of the (n+2) layer network
This illustration indicates the training performance degradation problem is a **consequence of the network depth**.
• Kaiming He and his colleagues at Microsoft research hypothesized that this problem is an optimization problem.
• I.e., deeper models are harder to optimize.
• Signals were not propagating properly after adding new layers.
He et al. proposed a simple yet *ingenious solution* to ensure that signals propagate through the network (forward & backward) without any loss.
As we add a new layer, train it into an identity function $f(x) = x$.

Signals will propagate without any loss through the identity channel.
• How do we make the network to learn an **identity function**?
• Well, we could simply **add the input of a layer to the output of the layer.**
• In other words, we could add a **short-cut** from input to output.
• I.e., add a **skip connection**.
• As the new model gets a better solution to fit the training dataset, the added layer **might make it easier** to reduce training errors.
• Let’s say that the goal of the network is to learn a target function $f(x)$.

• The left figure below directly learns the mapping $f(x)$ (the part within the dotted-line box).

• However, on the right, the part within the dotted-line box needs to learn the residual mapping $f(x) - x$.

Now our goal to learn an identity mapping $f(x) = x$ is easier to achieve.
• For example, the learning process needs to push the weights and biases of the **upper weight layer** (e.g., fully-connected layer and Conv layer) within the dotted-line box to **zero**.

• Thus, by adding a skip connection we could train an identity mapping.

• This micro-network to learn identity map is known as the **residual block**.

With residual blocks, inputs can **forward propagate faster** through the residual connections across layers.
A useful aspect of the residual learning is that if we add many skip connections, the network can **start making progress** even if several layers have not started learning yet.

Due to the skip connections, the signal can **easily make its way across** the whole network.

![Diagram showing residual learning](image)

- **X** = Layer blocking backpropagation
- **=** = Layer not learning

Residual units

• Very Deep Models using Residual Learning
• Kaiming He et al. used residual blocks to **avoid the training degradation problem**.
• They designed deeper networks by stacking a set of residual blocks.
• It resulted into a **new architecture** named ResNet.
• Kaiming He et al.’s ResNet won the ILSVRC 2015 challenge by achieving **3.57% top-5 error rate.**

The winning variant used an extremely deep ResNet composed of **152 layers** (other variants had 34, 50, and 101 layers).
- Residual Block Architecture
- We use a special notation for the presentation.
- For example, the notation “$f_k, 3 \times 3 + 1 (S)$” indicates that the layer uses $f_k$ number of $3 \times 3$ filters with stride 1, and SAME padding.

The size of the filters used in the Conv layers in a residual block is always $3 \times 3$, which is inspired by VGG-design.

The residual block has two $3 \times 3$ convolutional layers with the same number of filters $f_k$. 
• The first Conv layer uses **either a stride of 1 or 2** with SAME padding.
• The Conv layers use **stride of 1** everywhere except for the **starting Conv layer in a stack** of residual blocks with the same $f_k$ where a stride of 2 is used.

Each convolutional layer is followed by a **batch normalization** layer and a **ReLU** activation function.

Then, the two convolution operations are skipped and add the input directly **before the final ReLU** activation function.
• Special Note on the Skip Connection
  • When the residual blocks are **stacked** sequentially in a ResNet module (shaded green):
    • The number of feature maps \( f_k \) is **doubled** every few residual blocks.
    • As \( f_k \) is doubled, their height and width are **halved** (using a Conv layer with **stride 2**).

When this happens, the inputs cannot be added directly to the outputs of the residual block because they **don’t have the same shape**.
• To solve this problem, the inputs are passed through a $1 \times 1$ Conv layer with **stride 2** and the right number of output feature maps.

• The stride 2 ensures that the input map height and width are halved, thus it **matches the size of the input** through the two Conv layers.
Thus, every time we change the number of channels and **halve the size** of the input, we need to introduce an additional $1 \times 1$ Conv layer on the skip channel to transform the input into the desired shape for the addition operation.

Original input through the skip channel **doesn’t have the same shape** as it was not halved, thus cannot be added.

The height and width are halved due to stride 2.

Input through the skip channel is halved.
• ResNet Model Architecture
• The ResNet architecture is surprisingly simple, which starts and ends exactly like GoogLeNet (except without a dropout layer), and in between is just a very deep stack of simple residual blocks.

The difference is the batch normalization (BN) layer added after each Conv layer in ResNet.
• **ResNet-18 Model Architecture**
• It has **4 residual modules** (contains stack of residual blocks).
• Each module contains a **stack of 2 residual blocks**.
• Thus a total of **8 residual blocks** (16 Conv layers, excluding the 1 x 1 Conv layer) + input Conv layer + output FC layer.
• A total of 18 layers with learnable parameters (excluding batch normalization parameters).

---

**ResNet-18**
Conv (64 filters of size 7 x 7, stride = 2, same padding) + BN + Activation
Max pool (pool size 3 x 3, stride = 2, same padding)

Residual Module: 2 Residual blocks each with 64 filters
Residual Module: 2 Residual blocks each with 128 filters
Residual Module: 2 Residual blocks each with 256 filters
Residual Module: 2 Residual blocks each with 512 filters

Global Average Pool
Flatten
Dense (neurons=Number of classes, softmax activation)

---

TensorFlow Implementation:
• **Deeper** ResNet

• For ImageNet classification **deeper ResNet architectures** were used with layer numbers 34, 50, 101, 152.

• The ResNet building blocks are shown in brackets with the number of blocks stacked.

<table>
<thead>
<tr>
<th>layer name</th>
<th>output size</th>
<th>18-layer</th>
<th>34-layer</th>
<th>50-layer</th>
<th>101-layer</th>
<th>152-layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1</td>
<td>112×112</td>
<td></td>
<td></td>
<td>7×7, 64, stride 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv2_x</td>
<td>56×56</td>
<td>[3×3, 64] ×2</td>
<td>[3×3, 64] ×3</td>
<td>[1×1, 64]</td>
<td>[1×1, 64]</td>
<td>[1×1, 64]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[3×3, 64]</td>
<td>[3×3, 64]</td>
<td>[1×1, 256]</td>
<td>[1×1, 256]</td>
<td>[1×1, 256]</td>
</tr>
<tr>
<td>conv3_x</td>
<td>28×28</td>
<td>[3×3, 128] ×2</td>
<td>[3×3, 128] ×4</td>
<td>[1×1, 128]</td>
<td>[1×1, 128]</td>
<td>[1×1, 128]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[3×3, 128]</td>
<td>[3×3, 128]</td>
<td>[1×1, 512]</td>
<td>[1×1, 512]</td>
<td>[1×1, 512]</td>
</tr>
<tr>
<td>conv4_x</td>
<td>14×14</td>
<td>[3×3, 256] ×2</td>
<td>[3×3, 256] ×6</td>
<td>[1×1, 256]</td>
<td>[1×1, 256]</td>
<td>[1×1, 256]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[3×3, 256]</td>
<td>[3×3, 256]</td>
<td>[1×1, 1024]</td>
<td>[1×1, 1024]</td>
<td>[1×1, 1024]</td>
</tr>
<tr>
<td>conv5_x</td>
<td>7×7</td>
<td>[3×3, 512] ×2</td>
<td>[3×3, 512] ×3</td>
<td>[1×1, 512]</td>
<td>[1×1, 512]</td>
<td>[1×1, 512]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[3×3, 512]</td>
<td>[3×3, 512]</td>
<td>[1×1, 2048]</td>
<td>[1×1, 2048]</td>
<td>[1×1, 2048]</td>
</tr>
<tr>
<td>1×1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>average pool, 1000-d fc, softmax</td>
<td></td>
</tr>
<tr>
<td>FLOPs</td>
<td>1.8×10⁹</td>
<td>3.6×10⁹</td>
<td>3.8×10⁹</td>
<td>7.6×10⁹</td>
<td>11.3×10⁹</td>
<td></td>
</tr>
</tbody>
</table>
• **Flexibility** of the ResNet Architecture
• We can create different ResNet models by configuring a ResNet module (stack of Residual blocks):
  - Different numbers of *channels* and
  - Different numbers of *Residual blocks*

<table>
<thead>
<tr>
<th>layer name</th>
<th>output size</th>
<th>18-layer</th>
<th>34-layer</th>
<th>50-layer</th>
<th>101-layer</th>
<th>152-layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1</td>
<td>112×112</td>
<td>7×7, 64, stride 2</td>
<td>3×3 max pool, stride 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv2_x</td>
<td>56×56</td>
<td>3×3, 64</td>
<td>3×3, 64</td>
<td>1×1, 64</td>
<td>1×1, 64</td>
<td>1×1, 64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>×2</td>
<td>×3</td>
<td>×3</td>
<td>×3</td>
<td>×3</td>
</tr>
<tr>
<td>conv3_x</td>
<td>28×28</td>
<td>3×3, 128</td>
<td>3×3, 128</td>
<td>1×1, 128</td>
<td>1×1, 128</td>
<td>1×1, 128</td>
</tr>
<tr>
<td></td>
<td></td>
<td>×2</td>
<td>×4</td>
<td>×4</td>
<td>×4</td>
<td>×8</td>
</tr>
<tr>
<td>conv4_x</td>
<td>14×14</td>
<td>3×3, 256</td>
<td>3×3, 256</td>
<td>1×1, 256</td>
<td>1×1, 256</td>
<td>1×1, 256</td>
</tr>
<tr>
<td></td>
<td></td>
<td>×2</td>
<td>×6</td>
<td>×6</td>
<td>×23</td>
<td>×36</td>
</tr>
<tr>
<td>conv5_x</td>
<td>7×7</td>
<td>3×3, 512</td>
<td>3×3, 512</td>
<td>1×1, 512</td>
<td>1×1, 512</td>
<td>1×1, 512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>×2</td>
<td>×3</td>
<td>×3</td>
<td>×3</td>
<td>×3</td>
</tr>
<tr>
<td>1×1</td>
<td></td>
<td>1×1, 2048</td>
<td>1×1, 2048</td>
<td>1×1, 2048</td>
<td>1×1, 2048</td>
<td>1×1, 2048</td>
</tr>
<tr>
<td>FLOPs</td>
<td>1.8×10^9</td>
<td>3.6×10^9</td>
<td>3.8×10^9</td>
<td>7.6×10^9</td>
<td>11.3×10^9</td>
<td></td>
</tr>
</tbody>
</table>
• **Flexibility** of the ResNet Architecture
• Although the main architecture of ResNet is similar to that of GoogLeNet, ResNet’s structure is **simpler and easier to modify**.
• All these factors have resulted in the **rapid and widespread use** of ResNet.

<table>
<thead>
<tr>
<th>layer name</th>
<th>output size</th>
<th>18-layer</th>
<th>34-layer</th>
<th>50-layer</th>
<th>101-layer</th>
<th>152-layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1</td>
<td>112×112</td>
<td></td>
<td></td>
<td>7×7, 64, stride 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv2.x</td>
<td>56×56</td>
<td>[3×3, 64]</td>
<td>[3×3, 64]</td>
<td>[1×1, 64]</td>
<td>[1×1, 64]</td>
<td>[1×1, 64]</td>
</tr>
<tr>
<td></td>
<td>3×3, 64</td>
<td>×2</td>
<td>3×3, 64</td>
<td>3×3, 64</td>
<td>3×3, 64</td>
<td>3×3, 64</td>
</tr>
<tr>
<td></td>
<td>3×3, 64</td>
<td></td>
<td>1×1, 256</td>
<td>1×1, 256</td>
<td>1×1, 256</td>
<td>1×1, 256</td>
</tr>
<tr>
<td>conv3.x</td>
<td>28×28</td>
<td>[3×3, 128]</td>
<td>[3×3, 128]</td>
<td>[1×1, 128]</td>
<td>[1×1, 128]</td>
<td>[1×1, 128]</td>
</tr>
<tr>
<td></td>
<td>3×3, 128</td>
<td>×2</td>
<td>3×3, 128</td>
<td>3×3, 128</td>
<td>3×3, 128</td>
<td>3×3, 128</td>
</tr>
<tr>
<td></td>
<td>3×3, 128</td>
<td></td>
<td>1×1, 512</td>
<td>1×1, 512</td>
<td>1×1, 512</td>
<td>1×1, 512</td>
</tr>
<tr>
<td>conv4.x</td>
<td>14×14</td>
<td>[3×3, 256]</td>
<td>[3×3, 256]</td>
<td>[1×1, 256]</td>
<td>[1×1, 256]</td>
<td>[1×1, 256]</td>
</tr>
<tr>
<td></td>
<td>3×3, 256</td>
<td>×2</td>
<td>3×3, 256</td>
<td>3×3, 256</td>
<td>3×3, 256</td>
<td>3×3, 256</td>
</tr>
<tr>
<td></td>
<td>3×3, 256</td>
<td></td>
<td>1×1, 1024</td>
<td>1×1, 1024</td>
<td>1×1, 1024</td>
<td>1×1, 1024</td>
</tr>
<tr>
<td>conv5.x</td>
<td>7×7</td>
<td>[3×3, 512]</td>
<td>[3×3, 512]</td>
<td>[1×1, 512]</td>
<td>[1×1, 512]</td>
<td>[1×1, 512]</td>
</tr>
<tr>
<td></td>
<td>3×3, 512</td>
<td>×2</td>
<td>3×3, 512</td>
<td>3×3, 512</td>
<td>3×3, 512</td>
<td>3×3, 512</td>
</tr>
<tr>
<td></td>
<td>3×3, 512</td>
<td></td>
<td>1×1, 2048</td>
<td>1×1, 2048</td>
<td>1×1, 2048</td>
<td>1×1, 2048</td>
</tr>
<tr>
<td>1×1</td>
<td>average pool, 1000-d fc, softmax</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLOPs</td>
<td>1.8×10^9</td>
<td>3.6×10^9</td>
<td>3.8×10^9</td>
<td>7.6×10^9</td>
<td>11.3×10^9</td>
<td></td>
</tr>
</tbody>
</table>
• Residual Block for Deeper ResNet Models
• Deeper ResNets use a **slightly different design** for the residual blocks.
• They use “**bottleneck**” layers, which is 1 x 1 Conv layer, to improve the computational efficiency.

To understand the design of the residual block for deeper ResNets, consider the following figure.
• We use an example of input of height and size 28 x 28 with 256 channels.

• **1\textsuperscript{st} Conv layer**: it’s a bottleneck 1 x 1 Conv layer. It reduces the dimension of the input filter channels to 64.

• **2\textsuperscript{nd} Conv layer**: it’s a scanning 3 x 3 Conv layer that works on the **reduced** 64 filters.

**3\textsuperscript{rd} Conv layer**: it’s a 1 x 1 Conv layer to **increase the number of channels** back to 256, such that the number of input and output channels remain the same.

- ResNet Training: He et al. Recommendations (for ImageNet)
  - Perform batch normalization after every Conv layer
  - He/Glorot weight initialization
  - SGD + Momentum (0.9)
  - Learning rate schedule: initial rate 0.1, then divided by 10 when validation error plateaus
  - Mini-batch size 256
  - Weight decay of 1e-5
  - No dropout

- Variants of ResNet
- Key variants of the ResNet architecture are:
  - Pre-activation ResNet
  - Wide ResNet
  - ResNeXt
The main motivation behind the pre-activation ResNet is to ensure the smooth flow of signals such that deeper models can be trained easily and effectively.

First the maps are activated by passing it through the batch normalization and ReLU activation layer, then the activated maps are convolved (figure b below).

- Residual block based **deeper** models suffer from **two issues**.
  - Increased number of parameters and computations.
  - **Diminishing feature reuse**.
- The identity mappings in the residual blocks may become a weakness in deep networks.

As gradient flows through the network there is **nothing to force it to go through residual block weights** and it can **avoid learning** anything during training.

How to we build effective ResNet that are **shallow**?

- The Wide ResNet builds **shallow network by widening** the ResNet blocks.
- It was argued that **residuals are the important factor** to improve performance
- **Depth is not the key!**

It was a novel observation!

- ResNet vs Wide ResNet: Depth vs Width
- Following table shows the performance comparison between original ResNet and Wide Resnet.
- We see that with a widening factor of 2.0 the resulting 50-layer Wide Resnet “WRN-50-2-bottleneck” outperforms ResNet-152.
- The wide ResNet has 3 times less layers, and is significantly faster.

<table>
<thead>
<tr>
<th>Model</th>
<th>top-1 err, %</th>
<th>top-5 err, %</th>
<th>#params</th>
<th>time/batch 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-50</td>
<td>24.01</td>
<td>7.02</td>
<td>25.6M</td>
<td>49</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>22.44</td>
<td>6.21</td>
<td>44.5M</td>
<td>82</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>22.16</td>
<td>6.16</td>
<td>60.2M</td>
<td>115</td>
</tr>
<tr>
<td><strong>WRN-50-2-bottleneck</strong></td>
<td>21.9</td>
<td>6.03</td>
<td>68.9M</td>
<td>93</td>
</tr>
<tr>
<td>pre-ResNet-200</td>
<td>21.66</td>
<td>5.79</td>
<td>64.7M</td>
<td>154</td>
</tr>
</tbody>
</table>
ResNeXt (2016)

- Pre-activation ResNet
- Wide ResNet
- ResNeXt

• ResNeXt stands for **Aggregated Residual Transformations for Deep Neural Networks**.

• Designed by Kaiming He’s group.

• The main idea is to combine ResNet with split-transform-merge strategy of **Inception**.
ResNeXt (2016)

- The ResNeXt induces the **boosting effect**.
- It does so by combining a set of simpler classifier into a powerful one.
- The ResNeXt blocks use cardinality=32 (# parallel paths) and bottleneck width=4d (filters), denoted as ResNeXt-50 (32 x 4d).
ResNeXt (2016)

• The ResNeXt models outperforms the vanilla ResNet models without increasing model complexity.
Notable Architectures

• We will study the following architectures that are winners or runner-ups in the ImageNet challenge.
  - AlexNet (2012 winner)
  - VGGNet (2014 runner-up)
  - GoogLeNet (Inception-v1) (2014 winner)
  - ResNet (2015 winner)
  - SENet (2017 winner)

A more comprehensive analysis on a larger set of architectures is given:

SENet (2017)

• The Squeeze-and-Excitation Network (SENet) was the winning architecture in the ILSVRC 2017 challenge, which lowered the top-5 error rate down to 2.25%!
SENet (2017)

- Unlike the previous design approaches, SENet focused on a **different aspect of network design**: How to improve feature maps (channels) in relation to other feature maps (channels)?
- It developed a technique to improve the quality of feature maps by performing “**feature recalibration**”.
- A new architectural unit was proposed.
- It was named as the SE block.
- It learns to **adaptively reweight feature maps**.
Say that we want to recognize cat faces from images.

A reasonable assumption about a cat’s face is that on an image representing a cat, its ears, nose, eyes and whiskers will **appear together**.

These are the features of a cat’s face.

A Conv layer in a neural network will learn **different feature detectors** to identify these features.
SENet (2017)

- The feature maps (output channels) of a Conv layer learn each of these features *individually*.
- For example, one feature map for ears, one for nose, one for eyes, etc.

All these feature maps should have *similar activation* if the features appear in the input image (i.e., if the input image represents a cat’s face).
SENet (2017)

• What if the feature maps identifying ear, eyes and nose have strong activation, but the map detecting the **whiskers have only weak activation**?

• May be the Conv layer was confused about the whiskers.

Then, how do we help the Conv layer to **increase the activation** of the map that detects whiskers?
SENet (2017)

• We could design a micro-network that will use the output feature maps of each Conv block and will learn which maps need to be “boosted”.

• It will do so by analyzing the inter-relationship between the feature maps.
SENet (2017)

• For the cat image example, this micro-network will increase the activation of the whisker-detector map by finding the fact that this map is strongly related to the strongly activated ears-eyes-nose maps.

• In other words, this micro-network will perform feature map recalibration.

This is the idea behind the SE block.
SENet (2017)

• An SE block analyzes the output of the unit it is attached to.
• It focuses **exclusively on the depth dimension** (it does not look for any spatial pattern).
• It learns which features are usually **most active together**.
• It **learns the global information** to selectively emphasize informative features and suppress less useful ones.
- A global average pooling layer
- A hidden FC layer using the ReLU activation function, and
- A FC output layer using the sigmoid activation function
• First the feature maps of size $H \times W \times C$ with $C$ channels computed by a Conv layer are passed through the global average pooling layer.

• It computes the **mean activation** for each feature map.

For example, if its input contains $C$ feature maps, it will **output $C$ numbers** representing the overall level of response for each filter.
• The next layer is where the **squeeze operation happens**.
• It produces a map descriptor (channel descriptor) by aggregating feature maps across their spatial dimensions \((H \times W)\).
• This layer has **significantly fewer than \(C\) neurons**, \(r\) times fewer than the number of feature maps.

Typically \(r = 16\). Thus, the \(C\) numbers get **compressed into a small vector** (e.g., \(C/r\) dimensions).
• This is a **low-dimensional vector representation** (i.e., an embedding) of the distribution of channel-wise feature responses.

• This **bottleneck step** forces the SE block to learn a general representation of the **feature combinations**.

This aggregation is followed by an **excitation operation** in the output layer.
• The output FC layer takes the embedding and **outputs a recalibration vector**.
• It contains **one number per feature map** between 0 and 1.
• These numbers are used to recalibrate feature maps.
• The feature maps are then **multiplied by the recalibration vector**.

• As a result, **irrelevant features** (with a low recalibration score) get scaled down while relevant features (with a recalibration score close to 1) are left alone.
• SE Block based Convnet
• The SE block can be integrated into standard architectures by inserting it after the nonlinearity following each convolution.
• Below we show illustrations of how the SE block can be used to extend existing architectures such as Inception networks and ResNets.
• These are called SE-Inception and SE-ResNet, respectively.
• SE Block based Convnets: Complexity
  • By comparing ResNet-50 with SE-ResNet-50, it was shown that there was only 0.26% relative increase in GFLOPS (1 gigaflops = 1 billion FLOPS).
  • But with this slight additional computational burden, the SE-ResNet-50 approached the accuracy of deeper ResNet-101 architecture.
  • The number of parameters in the SE-ResNet-50 model increased by 10%, which is mainly due to the two FC layers in each SE block.
Comparison

- We compare the performance of the notable architectures by analyzing **computational complexity** (number of operations) and **model complexity** (number of parameters).
- This comparison is drawn from the following paper. [https://arxiv.org/abs/1810.00736](https://arxiv.org/abs/1810.00736)
• The computational complexity is measured using floating-point operations (FLOPS) required for a single forward pass.
• The model complexity is measured by counting the total amount of learnable parameters.
• The size of each ball corresponds to the model complexity.
Accuracy Rate vs. Computational Complexity vs. Model Complexity

- NASNet-A-Large: highest accuracy, but has the highest computational complexity.
Accuracy Rate vs. Computational Complexity vs. Model Complexity

- **SE-ResNeXt-50(32 x 4d):** most effective with low complexity.
- The lowest computational complexity (i.e. lower than 5 GFLOPs).
- A low level of model complexity, with approximately 2.76 M-params.
Accuracy Rate vs. Computational Complexity vs. Model Complexity

• The second-best low-complexity and high performance model is Xception.
• It has slightly lower accuracy than the SE-ResNeXt-50(32 x 4d), but it requires less parameters.
Accuracy Rate vs. Computational Complexity vs. Model Complexity

- One of the **most efficient** model is GoogLeNet.
- The closest is ResNet-18.
Accuracy-Rate vs. Learning Power

- One issue with deep networks is that they are **inefficient in the use of their full learning power** (measured as the number of parameters with respect to the degrees of freedom).
• Top-1 accuracy vs. top-1 accuracy density (top-1 accuracy divided by the number of parameters).
• The higher is this value and the higher is the efficiency.

Among the highest top-1 accuracy (i.e., higher than 80%) models, following two models use their parameters more efficiently:
Inception-v4 and SE-ResNeXt-101 (32 x 4d).