# Deep Residual Learning for Image Recognition

Leonardo Hügens

Faculdade de Ciências, Universidade do Porto

November 11, 2023

# Paper

This presentation is based on the paper `Deep Residual Learning for Image Recognition` by `He, Zhang, Ren & Sun`, from `Microsoft Research`.

**Deep Residual Learning for Image Recognition**

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

# Outline

# Introduction

- Deeper neural networks are more difficult to train.

- The paper presents a *residual* learning framework to ease the training of networks that are substantially deeper than those used previously.

# Introduction

- Deeper neural networks are more difficult to train.

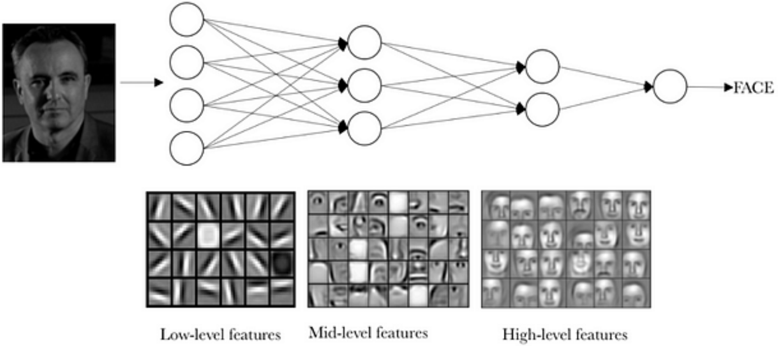
- The paper presents a *residual* learning framework to ease the training of networks that are substantially deeper than those used previously.

- The layers are reformulated as learning residual functions with reference to the layers inputs, instead of learning unreferenced functions.

- These networks are:
    - Easier to optimize
    - Can gain accuracy from considerably increased depth.

# Introduction

In a deep neural network (DNN) the "levels" of features can be enriched by the number of stacked layers.



# Introduction

The leading results on the challenging ImageNet dataset all exploit "very deep" models, with depth up to 30.

# Problem

**Question**: Is learning better networks as easy as stacking more layers?

# Problem

**Question**: Is learning better networks as easy as stacking more layers?

**Problem**: Vanishing/Exploding gradients.

**Solution**: Normalized initialization (e.g. for `ReLU` activation functions, the weights of the layer are `He` initialized, drawn from a Gaussian distribution with mean $\mu = 0$ and variance $\sigma = \frac{2}{\#inputs}$), and intermediate normalization layers.

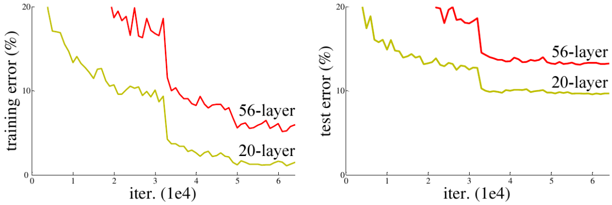
# Problem

- With an increase of the depth of a network, a *degradation* problem arises, were accuracy gets saturated, and then degrades rapidly.

# Problem

- With an increase of the depth of a network, a *degradation* problem arises, were accuracy gets saturated, and then degrades rapidly.

- This is not due to *overfitting*, and adding more layers leads to *higher training error*.



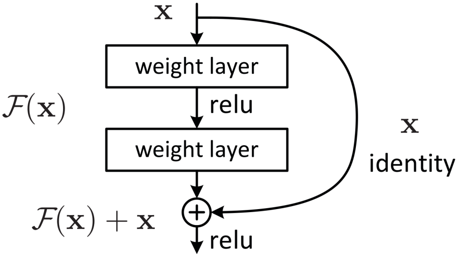
# Solution

**Deep residual learning framework** - Instead of directly learning a desired underlying mapping $\mathcal{H}(\mathbf{x})$, a *residual* mapping $\mathcal{F}(\mathbf{x})$is learned instead, were $\mathcal{F}(\mathbf{x}) \coloneqq \mathcal{H}(\mathbf{x}) - \mathbf{x}$.

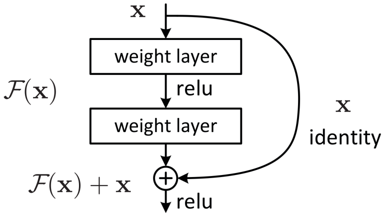
# Solution

**Deep residual learning framework** - Instead of directly learning a desired underlying mapping $\mathcal{H}(\mathbf{x})$, a *residual* mapping $\mathcal{F}(\mathbf{x})$is learned instead, were $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$.

`Shortcut` connections:

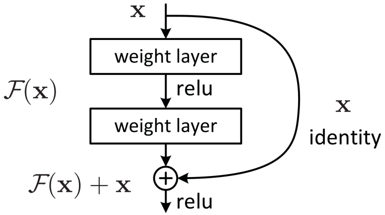


# Dimension detail




In order to sum two tensors, $\mathcal{F}(\mathbf{x}) + \mathbf{x}$ , they must have the same shape.

# Dimension detail




In order to sum two tensors, $\mathcal{F}(\mathbf{x}) + \mathbf{x}$ , they must have the same shape.

If this is not the case (e.g. when changing the input/output dimensions) there are two options.

# Dimension detail

To keep in mind: For most architectures, the dimension increases.
In order to match dimensions:

- Option (A): The same identity shortcut, and add extra zero entries.

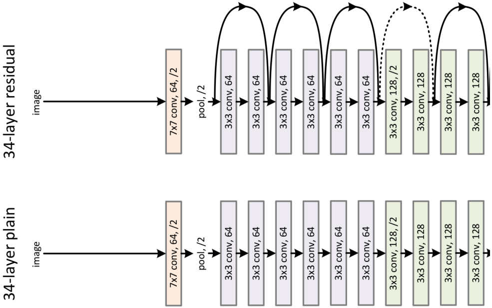
# Dimension detail

To keep in mind: For most architectures, the dimension increases.
In order to match dimensions:

- Option (A): The same identity shortcut, and add extra zero entries.
- Option (B): The projection shortcut, with parameters $W_i$:
$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}$

# Dimension detail

**ResNet-50** - a residual network architecture for image classification, specifically, for ImageNet.




Notice the 2 when the shape changes.

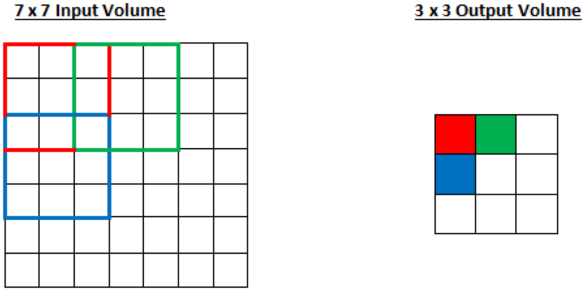
# Projection Shortcut

**ResNet-50** Tensorflow implementation.

```
x = residual_block_v1(x, filters, stride=2, name=name + "_block1")
for i in range(2, blocks + 1):
|   x = residual_block_v1(x, filters, conv_shortcut=False, name=name + "_block" + str(i))
return x
```

First the dimension is matched, by using a **stride**=2, then we can use regular identity shortcuts.

# Stride



# Projection Shortcut

**ResNet-50** Tensorflow implementation.

```
# here, strid=2
if conv_shortcut:
|   shortcut = layers.Conv2D(4 * filters, 1, strides=stride, name=name + "_0_conv")(x)
|   shortcut = layers.BatchNormalization(axis=bn_axis, epsilon=1.001e-5, name=name + "_0_bn")(shortcut)
else:
|   shortcut = x

x = layers.Conv2D(filters, 1, strides=stride, name=name + "_1_conv")(x)
x = layers.BatchNormalization(axis=bn_axis, epsilon=1.001e-5, name=name + "_1_bn")(x)
x = layers.Activation("relu", name=name + "_1_relu")(x)

x = layers.Conv2D(filters, kernel_size, padding="SAME", name=name + "_2_conv")(x)
x = layers.BatchNormalization(axis=bn_axis, epsilon=1.001e-5, name=name + "_2_bn")(x)
x = layers.Activation("relu", name=name + "_2_relu")(x)

x = layers.Conv2D(4 * filters, 1, name=name + "_3_conv")(x)
x = layers.BatchNormalization(axis=bn_axis, epsilon=1.001e-5, name=name + "_3_bn")(x)

x = layers.Add(name=name + "_add")([shortcut, x])
x = layers.Activation("relu", name=name + "_out")(x)
```