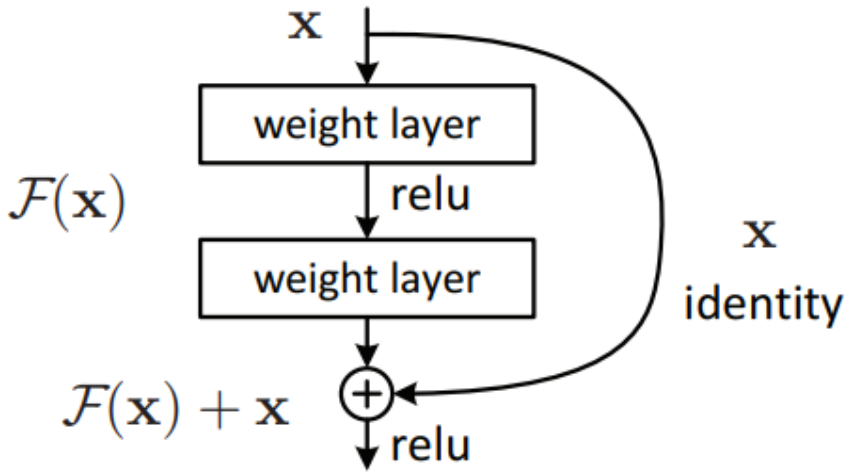
Residual Networks (ResNet):

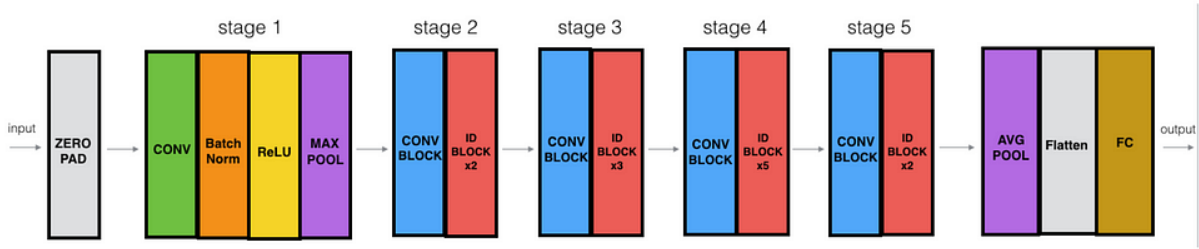
- After the success of AlexNet in the ImageNet 2012 competition, subsequent deeper architectures were developed to further reduce error rates.
- However, increasing the depth of neural networks led to the vanishing/exploding gradient problem, causing gradients to become too small or too large during training.
- The consequence of this problem was increased training and test error rates with deeper architectures.
- ResNet, proposed in 2015 by researchers at Microsoft Research, introduced a novel architecture called Residual Network to address the vanishing/exploding gradient issue.
- ResNet utilized the concept of Residual Blocks, which included skip connections. These connections allowed activations of a layer to skip certain layers in between, forming a residual block.
- The fundamental idea was to learn the residual mapping rather than the direct mapping. This was expressed as:
- Let H(x) be the initial mapping.
- Allow the network to fit F(x):=H(x)−x, resulting in H(x):=F(x)+x.
- The advantage of skip connections is that if a layer negatively impacts the performance, it can be bypassed during training, acting as a form of regularization.



Types of ResNet Blocks:

1. Identity Block:
   - Standard block where input and output dimensions are the same.
   - Skip connection "skips over" 2 layers.
   - Commonly used in ResNets.
2. Convolutional Block:
   - Used when input and output dimensions don't match.
   - Includes a CONV2D layer in the shortcut path.

ResNet-50:



Zero-padding is applied to the input, adding a pad of (3,3).

Stage 1:

The first layer is a 2D Convolutional layer with 64 filters of shape (7,7) and a stride of (2,2), named "conv1".

Batch Normalization is applied to the channels axis.

MaxPooling layer uses a (3,3) window and a (2,2) stride.

Stage 2:

Convolutional Block:

Three sets of filters of size 64x64x256.

Filter size: f=3, Stride: s=1.

Block name: "a".

Identity Blocks:

Three sets of filters of size 64x64x256 for each block.

Filter size: f=3.

Blocks named "b" and "c".

Stage 3:

Convolutional Block:

Three sets of filters of size 128x128x512.

Filter size: f=3, Stride: s=2.

Block name: "a".

Identity Blocks:

Three sets of filters of size 128x128x512 for each block.

Filter size: f=3.

Blocks named "b", "c", and "d".

Stage 4:

Convolutional Block:

Three sets of filters of size 256x256x1024.

Filter size: f=3, Stride: s=2.

Block name: "a".

Identity Blocks:

Five sets of filters of size 256x256x1024 for each block.

Filter size: f=3.

Blocks named "b", "c", "d", "e", and "f".

Stage 5:

Convolutional Block:

Three sets of filters of size 512x512x2048.

Filter size: f=3, Stride: s=2.

Block name: "a".

Identity Blocks:

Two sets of filters of size 512x512x2048 for each block.

Filter size: f=3.

Blocks named "b" and "c".

2D Average Pooling:

Applies average pooling with a window of shape (2,2).

Name: "avg_pool".

Flatten Layer:

No hyperparameters or name specified.

Fully Connected (Dense) Layer:

Reduces the input to the number of classes using a softmax activation.

Name: 'fc' + str(classes).

<u>Implementation of ResNet50 Using Keras:</u>

```
import keras
from keras.layers import Dense, Conv2D, BatchNormalization,
Activation
from keras.layers import AveragePooling2D, Input, Flatten
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint,
LearningRateScheduler
from keras.callbacks import ReduceLROnPlateau
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import MaxPooling2D
from keras.layers import Add
from keras.regularizers import l2
from keras import backend as K
from keras.models import Model
from keras.datasets import cifar10
import numpy as np
```

```
import os

def identity_block(x, filters, kernel_size=3, stride=1):
    # Identity block for ResNet
    y = Conv2D(filters, kernel_size=kernel_size, strides=stride,
padding='same')(x)
    y = BatchNormalization()(y)
    y = Activation('relu')(y)

    y = Conv2D(filters, kernel_size=kernel_size, padding='same')(y)
    y = BatchNormalization()(y)

    if stride != 1 or x.shape[-1] != filters:
        x = Conv2D(filters, kernel_size=1, strides=stride,
padding='same')(x)
        x = BatchNormalization()(x)

    y = Add()([x, y])
    y = Activation('relu')(y)

    return y

def conv_block(x, filters, kernel_size=3, stride=2):
    # Convolutional block for ResNet
    y = Conv2D(filters, kernel_size=kernel_size, strides=stride,
padding='same')(x)
    y = BatchNormalization()(y)
    y = Activation('relu')(y)

    y = Conv2D(filters, kernel_size=kernel_size, padding='same')(y)
    y = BatchNormalization()(y)

    x = Conv2D(filters, kernel_size=1, strides=stride,
padding='same')(x)
    x = BatchNormalization()(x)
```

```
    y = Add()([x, y])
    y = Activation('relu')(y)

    return y

def build_resnet50(input_shape=(224, 224, 3), num_classes=1000):
    # Build ResNet50 model
    input_tensor = Input(shape=input_shape)

    x = Conv2D(64, kernel_size=7, strides=2,
padding='same')(input_tensor)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = MaxPooling2D(pool_size=3, strides=2, padding='same')(x)

    x = conv_block(x, filters=64)
    x = identity_block(x, filters=64)
    x = identity_block(x, filters=64)

    x = conv_block(x, filters=128)
    x = identity_block(x, filters=128)
    x = identity_block(x, filters=128)
    x = identity_block(x, filters=128)

    x = conv_block(x, filters=256)
    x = identity_block(x, filters=256)
    x = identity_block(x, filters=256)
    x = identity_block(x, filters=256)
    x = identity_block(x, filters=256)

    x = conv_block(x, filters=512)
    x = identity_block(x, filters=512)
    x = identity_block(x, filters=512)
```

```
    x = AveragePooling2D(pool_size=4)(x)
    x = Flatten()(x)
    output = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=input_tensor, outputs=output,
name='resnet50')

    return model

# Instantiate ResNet50 model
resnet50_model = build_resnet50()

# Print model summary
resnet50_model.summary()
```