• *Summary of the models:*

-First Model

Architecture Of the network	*Train accuracy*	*Validation accuracy*	*Test accuracy (kaggle)*	*Epochs*	*Count of the images*	*optimiz*
Resnet50	99	94	82	3	10661	Sgd

-We make some data augmentation so we increase it from 1680 to 10661

-We make 333 batches to the image, batch size is 32

-Split 10661 images to train and validation by 267 batch to train and 66 batch to test.

-Resize the data to (244,244)

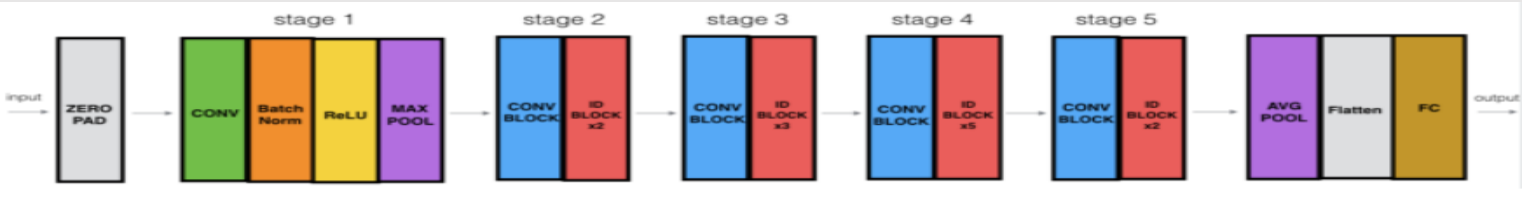
*-**resnet50:** we have identity block and that work is the standard block used in ResNets and corresponds*

To the case where the input activation has the same dimension as the output activation.

And we have **Convolutional Block We can use this type of block when the input and output dimension don't match up. The difference with the identity block is that there is a CONV2D layer in the**

Shortcut path.

-Architecture



-model

```
X_input = Input(input_shape)

    X = ZeroPadding2D((3, 3))(X_input)

    X = Conv2D(64, (7, 7), strides=(2, 2), name='conv1',
 kernel_initializer=glorot_uniform(seed=0))(X)
    X = BatchNormalization(axis=3, name='bn_conv1')(X)
    X = Activation('relu')(X)
    X = MaxPooling2D((3, 3), strides=(2, 2))(X)
  X = convolutional_block(X, f=3, filters=[64, 64, 256], stage=2, block='a', s=1)
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='b')
    X = identity_block(X, 3, [64, 64, 256], stage=2, block='c')


    X = convolutional_block(X, f=3, filters=[128, 128, 512], stage=3, block='a', s=2)
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='b')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='c')
    X = identity_block(X, 3, [128, 128, 512], stage=3, block='d')

    X = convolutional_block(X, f=3, filters=[256, 256, 1024], stage=4, block='a', s=2)
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='b')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='c')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='d')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='e')
    X = identity_block(X, 3, [256, 256, 1024], stage=4, block='f')

    X = X = convolutional_block(X, f=3, filters=[512, 512, 2048], stage=5, block='a', s=2)
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='b')
    X = identity_block(X, 3, [512, 512, 2048], stage=5, block='c')

    X = AveragePooling2D(pool_size=(2, 2), padding='same')(X)

    model = Model(inputs=X_input, outputs=X, name='ResNet50')
```

train logs

```
Epoch 1/3
267/267 [==============================] - ETA: 0s - loss: 0.4608 - accuracy: 0.8409
Epoch 1: saving model to /content/gdrive/MyDrive/LOGS_CNN/best_model.h5
267/267 [==============================] - 149s 538ms/step - loss: 0.4608 - accuracy: 0.8409 - val_loss: 0.4097 - val_accuracy: 0.8698
Epoch 2/3
267/267 [==============================] - ETA: 0s - loss: 0.0567 - accuracy: 0.9814
Epoch 2: saving model to /content/gdrive/MyDrive/LOGS_CNN/best_model.h5
267/267 [==============================] - 148s 550ms/step - loss: 0.0567 - accuracy: 0.9814 - val_loss: 0.2270 - val_accuracy: 0.9309
Epoch 3/3
267/267 [==============================] - ETA: 0s - loss: 0.0166 - accuracy: 0.9956
Epoch 3: saving model to /content/gdrive/MyDrive/LOGS_CNN/best_model.h5
267/267 [==============================] - 147s 546ms/step - loss: 0.0166 - accuracy: 0.9956 - val_loss: 0.1697 - val_accuracy: 0.9489
```

-second Model

Architecture Of the network	Train accuracy	Validation accuracy	Test accuracy (kaggle)	Epochs	Count of the images	optimizer
Inception	97	90	80	22	2712	Adam

-We make some data augmentation so we increase it from 1680 to 10661

-We make 85 batches to the image, batch size is 32

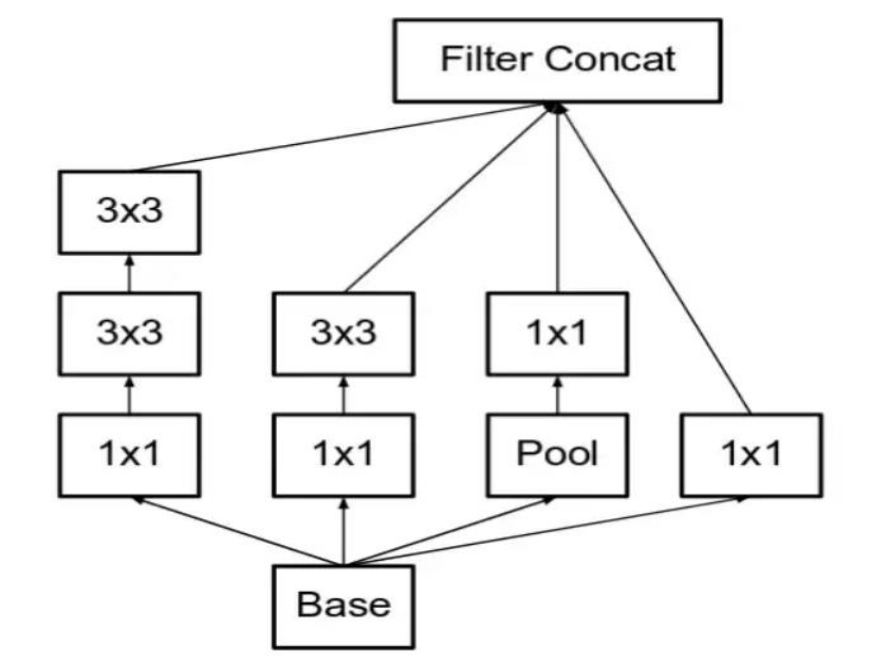
-Split 2712 images to train and validation by 68 batch to train and 17 batch to test.

-scaled the data between [0,1]

-Resize the data to (244,244)

*-**inception**: Factorize 5x5 convolution to two 3x3 convolution operations to improve computational speed. Although this may seem counterintuitive, a 5x5 convolution is 2.78 times more expensive than a 3x3 convolution. So stacking two 3x3 convolutions infact leads to a boost in performance. This is illustrated in the below image.*

-Architecture



-model

```
def inception_module(x, base_channels=16):
  a = Conv2D(base_channels*2, 1, 1, activation='relu')(x)

  b_1 = Conv2D(base_channels*2, 1, 1, activation='relu')(x)
  b_2 = Conv2D(base_channels*4, 3, 1, padding='same', activation='relu')(b_1)

  c_1 = Conv2D(base_channels, 1, 1, activation='relu')(x)
  c_2 = Conv2D(base_channels, 5, 1, padding='same', activation='relu')(c_1)
```

```
  d_1 = MaxPooling2D(3, 1, padding='same')(x)
  d_2 = Conv2D(base_channels, 1, 1, activation='relu')(d_1)

  return Concatenate(axis=-1)([a, b_2, c_2, d_2])

inp = Input((256,256, 3))

maps_1 = inception_module(inp)
max_1 = MaxPooling2D()(maps_1)

maps_2 = inception_module(max_1, base_channels=32)
max_2 = MaxPooling2D()(maps_2)

maps_3 = inception_module(max_2, base_channels=16)
max_3 = MaxPooling2D()(maps_3)

maps_4 = inception_module(max_3, base_channels=32)
max_4 = MaxPooling2D()(maps_4)

maps_5 = inception_module(max_4, base_channels=64)
max_5 = MaxPooling2D()(maps_5)

flat=Flatten()(max_5)
output = Dense(45, activation='relu',)(flat)
output = Dense(6, activation='softmax')(output)

model = Model(inputs=inp, outputs=output)
```

-train logs

```
Epoch 18/30
68/68 [==============================] - 41s 598ms/step - loss: 0.0403 - accuracy: 0.9839 - val_loss: 0.5761 - val_accuracy: 0.8862
Epoch 19/30
68/68 [==============================] - 41s 597ms/step - loss: 0.0472 - accuracy: 0.9835 - val_loss: 0.6144 - val_accuracy: 0.8675
Epoch 20/30
68/68 [==============================] - 41s 597ms/step - loss: 0.0450 - accuracy: 0.9862 - val_loss: 0.7344 - val_accuracy: 0.8806
Epoch 21/30
68/68 [==============================] - 41s 601ms/step - loss: 0.0972 - accuracy: 0.9655 - val_loss: 0.4025 - val_accuracy: 0.9049
Epoch 22/30
68/68 [==============================] - 43s 619ms/step - loss: 0.0831 - accuracy: 0.9701 - val_loss: 0.5089 - val_accuracy: 0.9011
```