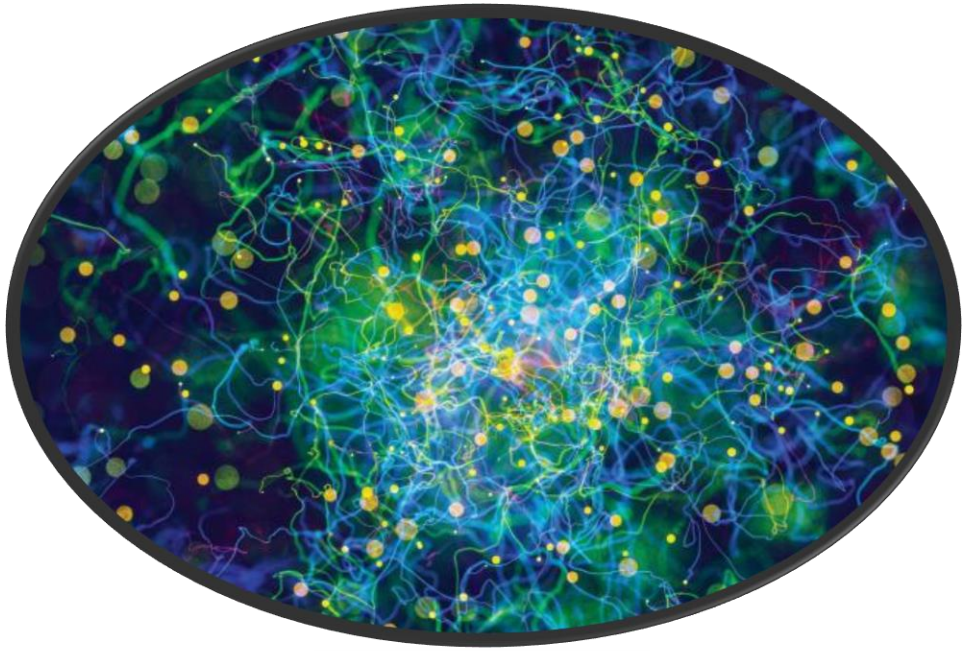
# Deep Learning



*Día 2*

**EXPOSITOR: Ing. Giorgio Morales Luna**

**Junio 2018**

## 3.1.6. Creación de modelos en Keras

Se realiza a través de la clase “Model” o “Sequential” de Keras

Métodos:

- Compile
- Fit
- Evaluate
- Predict
- Train_on_batch / Test_on_batch / Predict_on_batch
- Fit_generator / Evaluate_generator / Predict_ generator
- Get_layer

Model

```
from keras.models import Model
from keras.layers import Input, Dense

a = Input(shape=(32,))
b = Dense(32)(a)
model = Model(inputs=a, outputs=b)
```

Sequential

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

## 3.1.6. Creación de modelos en Keras

### Compile

Antes de entrenar un modelo, se necesita configurar el proceso de aprendizaje. Recibe tres principales argumentos:

- Optimizador: Un string con el método de optimización a usar. Ejm: SGD, Adam, etc. (https://keras.io/optimizers/)
- Función de costo: Es el objetivo que se tratará de minimizar durante el entrenamiento. Ejm: Binary crossentropy, categorical crossentropy, etc. https://keras.io/losses/
- Métricas: Se debe especificar la lista de métricas que se calcularán para evaluar la performancedel entrenamiento. Ejm: Accuracy, MSE, IoU, Precision, loss, etc.

```
# For a multi-class classification problem
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# For a binary classification problem
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

## 3.1.6. Creación de modelos en Keras

### Fit

Inicia el entrenamiento de un modelo (https://keras.io/models/sequential/#fit). Recibe los siguientes parámetros:

- X : Numpy array que contiene la data de entrenamiento
- Y : Numpy array del target (label).
- Batch_size: Número de muestras usadas para actualizar el gradiente. Default: 32
- Epochs: Número de épocas usadas para entrenar el modelo.
- Valdiation_Split: Float entre 0 y 1 que determinan la fracción de X a ser usada como datos de validación.
- Shuffle: Si es True, reordenará aleatoriamente los datos de entrenamiento al comenzar una época.

```
# Train the model, iterating on the data in batches of 32 samples
model.fit(data, labels, epochs=10, batch_size=32)
```

## 3.1.6. Creación de modelos en Keras

### Evaluate

Calcula el valor del costo y las métricas elegidas del modelo en modo de test.

```
model.fit(x_train, y_train,
          epochs=20,
          batch_size=128)
score = model.evaluate(x_test, y_test, batch_size=128)
```

### Predict

Devuelve las predicciones para las muestras de entrada dadas.

```
predict(x, batch_size=None, verbose=0, steps=None, callbacks=None)
```

## 3.1.6. Creación de modelos en Keras

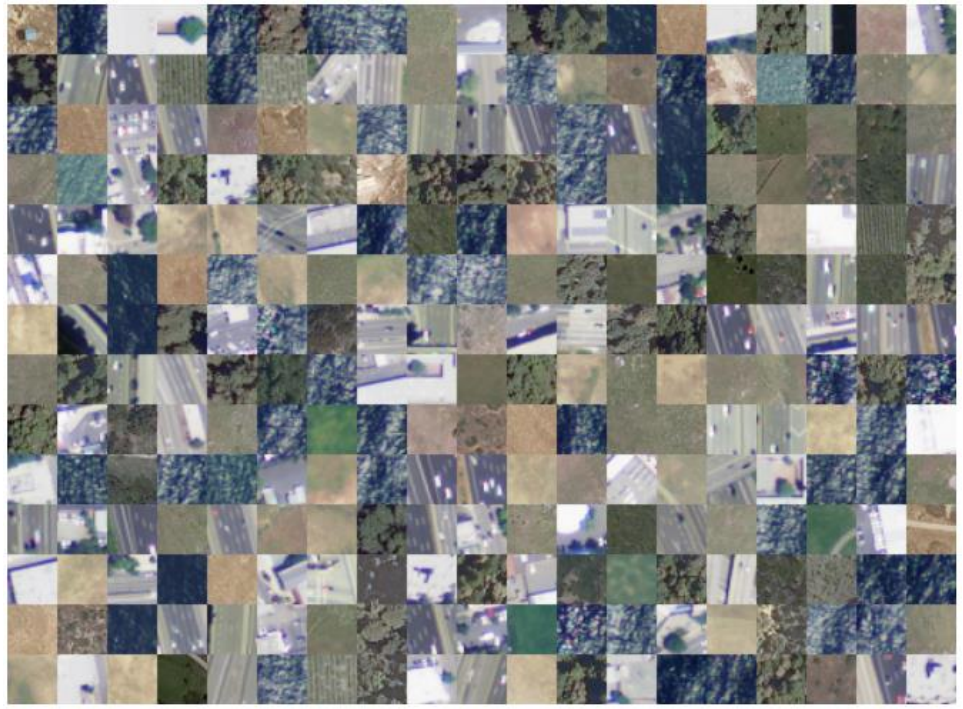
**Día_2/CreateModel.py**

## 3.1.7. CNN Example (DeepSat)

Sat-6: Dataset de imágenes aéreas RGB-NIR del Estado de California.
https://csc.lsu.edu/~saikat/deepsat/
https://drive.google.com/uc?id=0B0Fef71_vt3PUkZ4YVZ5WWNvZWs&export=download

Características:

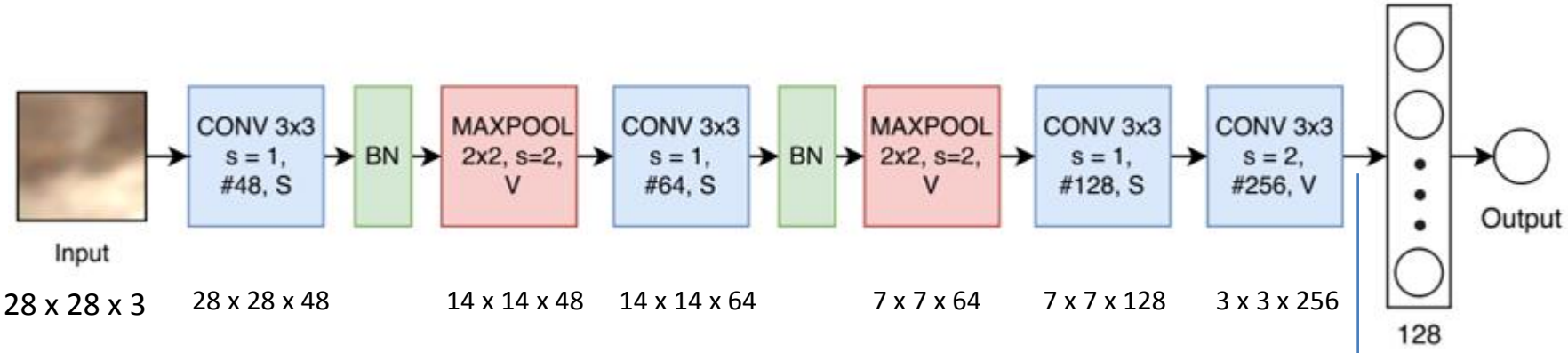
- Tamaño de patch: 28 x 28 x 4.
- Muestras de entrenamiento: 324,000.
- Muestras de test: 81,000.
- Clases: Barren land, trees, grassland, roads, buildings and water bodies.



Sample images from the SAT-4 and SAT-6 datasets

## 3.1.7. CNN Example (DeepSat)

**Día_2/EntrenamientoDeepSat1.py**



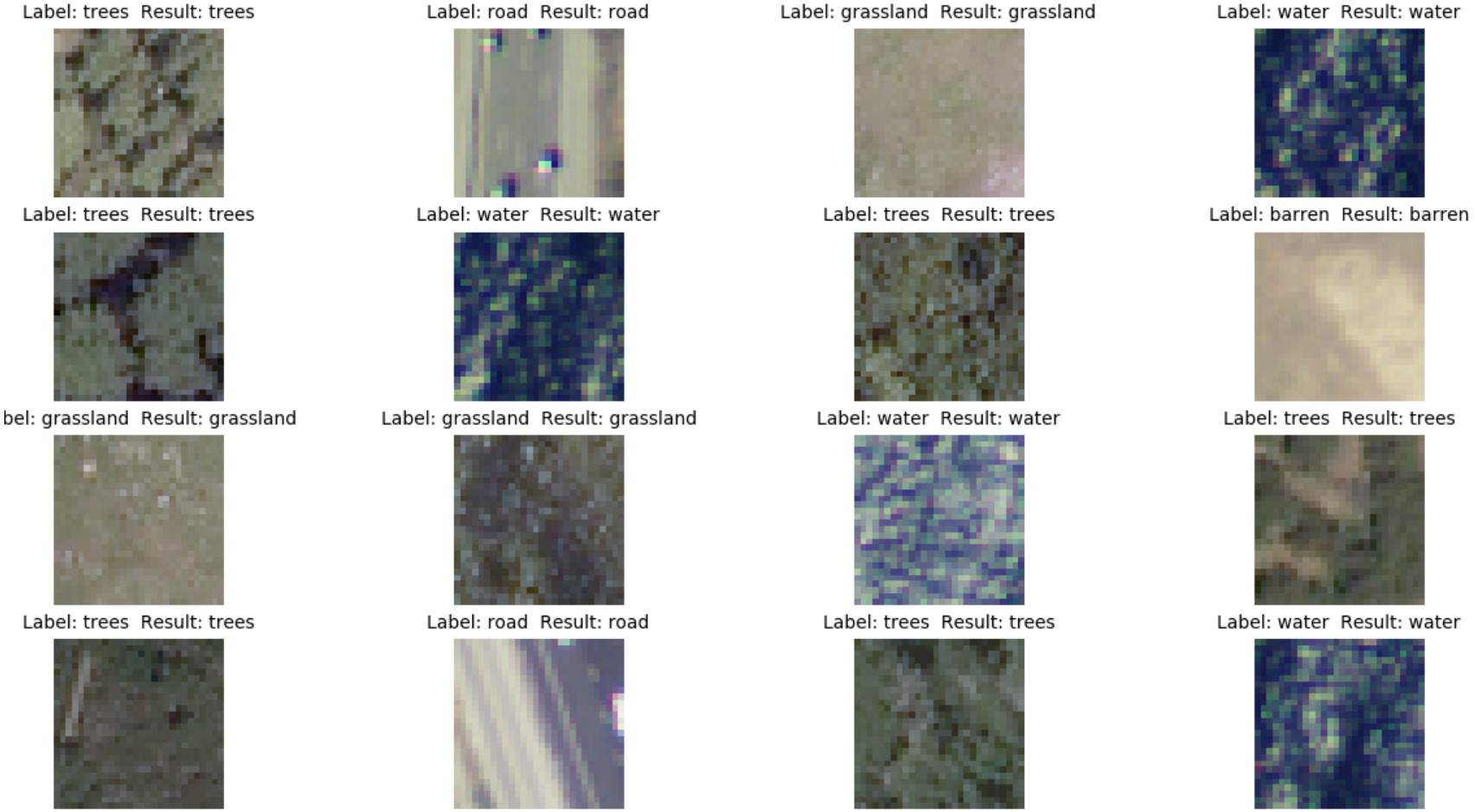

- **CONV 3 x 3**: Convolución con filtros de 3 x 3.
- **BN**: Batch Normalization.
- **MAXPOOL 2 x2**: Maxpooling con filstros de 2 x 2.
- **s**: Stride. Ejm: s = 1 -> s = (1,1)
- **S**: padding 'same'. V: padding 'valid'.
- Todas las activaciones son **'relu'**, expecto la de la salida: **'softmax'**

Para transformar la salida 2D a 1D se usa la función Flatten antes de la Fully Connected:

```
# output layer
x = Flatten()(x)
x = Dense(128, activation=
```

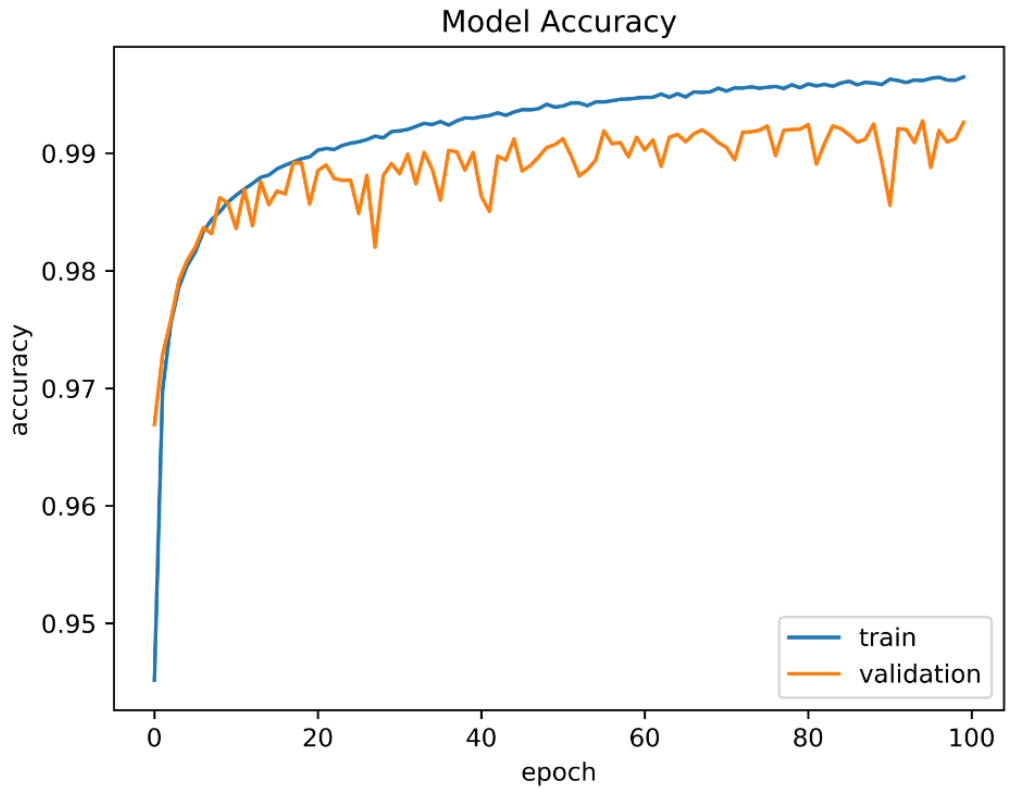
## 3.1.7. CNN Example (DeepSat)

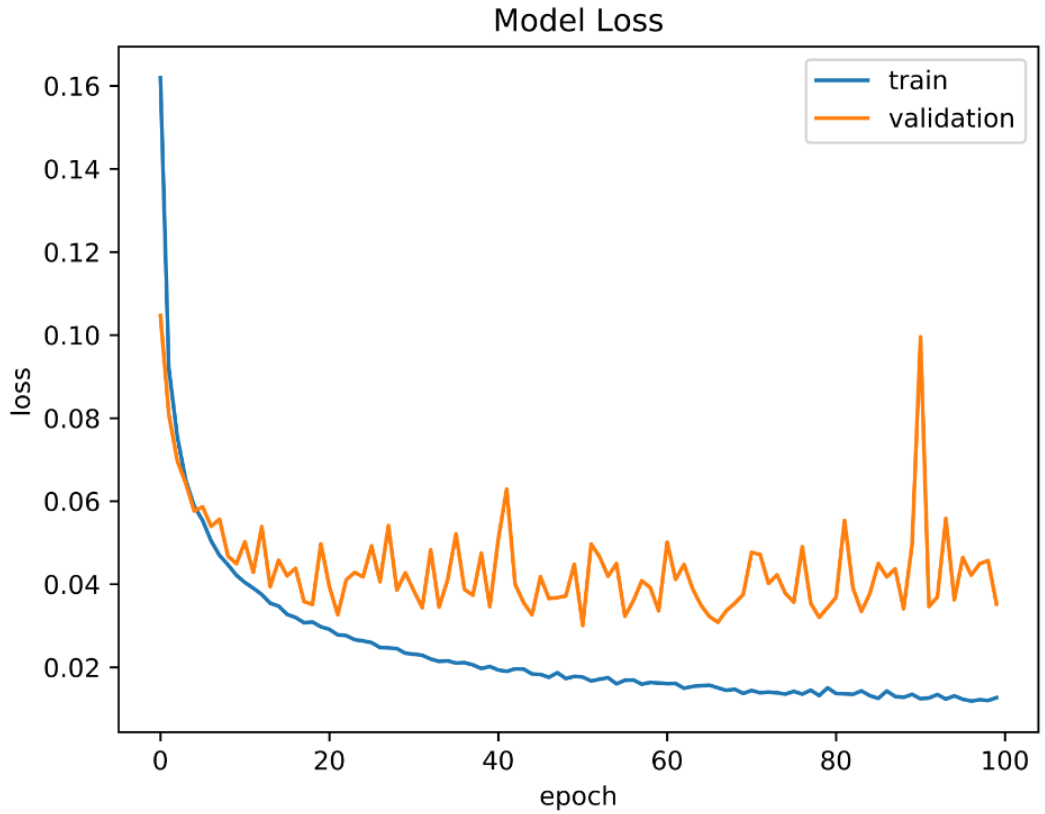
**Resultado**



## 3.1.7. CNN Example (DeepSat)

### Resultado



# 3.2. Classic Networks

¿Por qué estudiarlas?

- Para entender intuitivamente la estructura y combinación de bloques de las redes neuronales convolucionales más efectivas.
- Las CNN suelen ser multipropósito. Una red que detecta gatos y perros puede ser adaptada para detectar carros y bicicletas.

| Model | Size | Top-1 Accuracy | Top-5 Accuracy | Parameters | Depth |
|---|---|---|---|---|---|
| Xception | 88 MB | 0.790 | 0.945 | 22,910,480 | 126 |
| VGG16 | 528 MB | 0.713 | 0.901 | 138,357,544 | 23 |
| VGG19 | 549 MB | 0.713 | 0.900 | 143,667,240 | 26 |
| ResNet50 | 98 MB | 0.749 | 0.921 | 25,636,712 | - |
| ResNet101 | 171 MB | 0.764 | 0.928 | 44,707,176 | - |
| ResNet152 | 232 MB | 0.766 | 0.931 | 60,419,944 | - |
| ResNet50V2 | 98 MB | 0.760 | 0.930 | 25,613,800 | - |
| ResNet101V2 | 171 MB | 0.772 | 0.938 | 44,675,560 | - |
| ResNet152V2 | 232 MB | 0.780 | 0.942 | 60,380,648 | - |
| ResNeXt50 | 96 MB | 0.777 | 0.938 | 25,097,128 | - |
| ResNeXt101 | 170 MB | 0.787 | 0.943 | 44,315,560 | - |
| InceptionV3 | 92 MB | 0.779 | 0.937 | 23,851,784 | 159 |
| InceptionResNetV2 | 215 MB | 0.803 | 0.953 | 55,873,736 | 572 |
| MobileNet | 16 MB | 0.704 | 0.895 | 4,253,864 | 88 |
| MobileNetV2 | 14 MB | 0.713 | 0.901 | 3,538,984 | 88 |
| DenseNet121 | 33 MB | 0.750 | 0.923 | 8,062,504 | 121 |
| DenseNet169 | 57 MB | 0.762 | 0.932 | 14,307,880 | 169 |
| DenseNet201 | 80 MB | 0.773 | 0.936 | 20,242,984 | 201 |
| NASNetMobile | 23 MB | 0.744 | 0.919 | 5,326,716 | - |
| NASNetLarge | 343 MB | 0.825 | 0.960 | 88,949,818 | - |

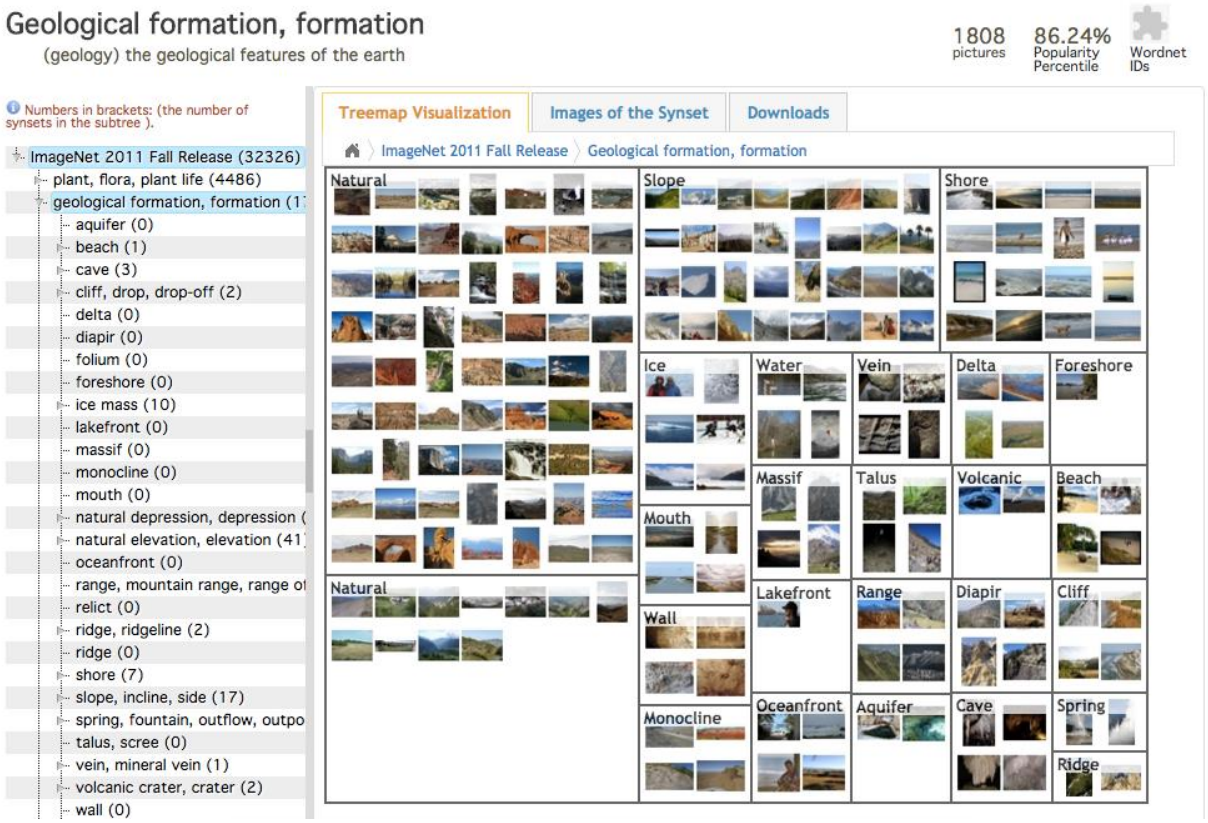
The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

https://keras.io/applications/

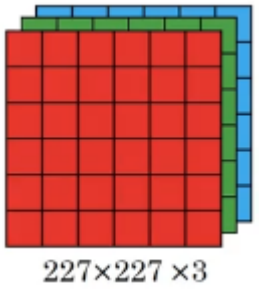
# 3.2. Classic Networks

## Imagenet Large Scale Visual Recognition Challenge (ILSVRC)

- Imagenet es el dataset más extenso, conformado por más de 14 millones de imágenes manualmente anotadas y más de 20,000 categorías.
- Ha sido diseñado para ser usado en la investigación de reconocimiento visual de objetos.
- Debido a la gran variedad de escenarios, se suelen usar modelos pre-entrenados en Imagenet para obtener resultados más robustos.

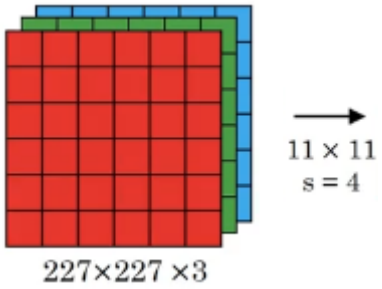


## 3.2.1. AlexNet



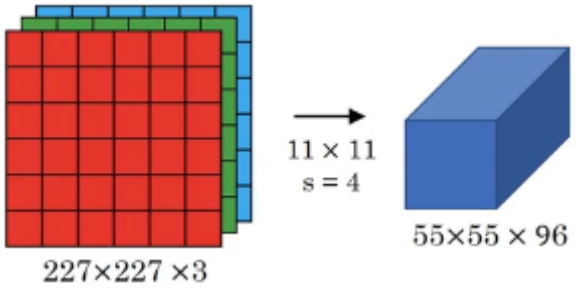

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

## 3.2.1. AlexNet



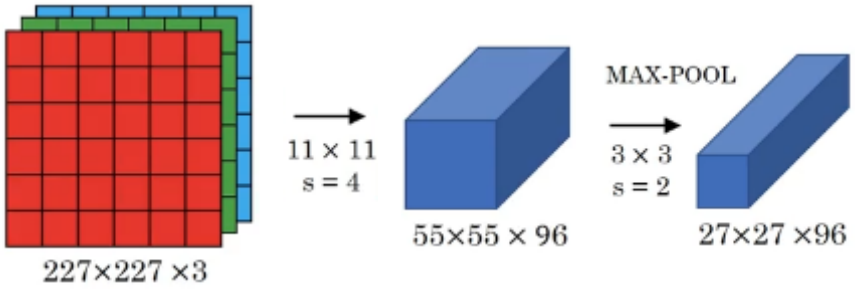

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

## 3.2.1. AlexNet



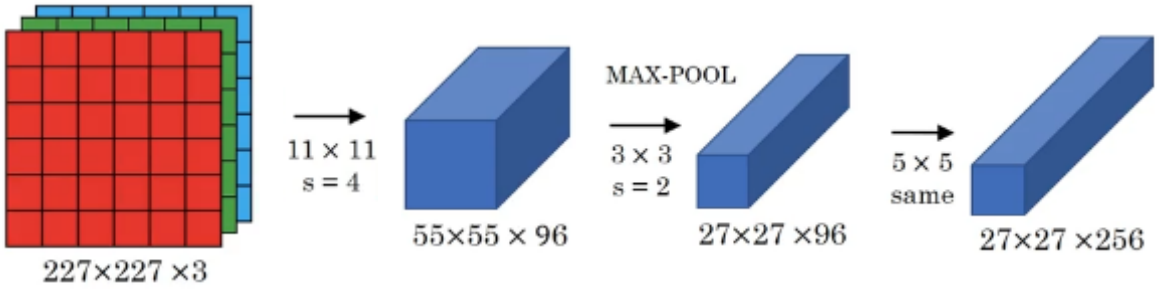

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

## 3.2.1. AlexNet



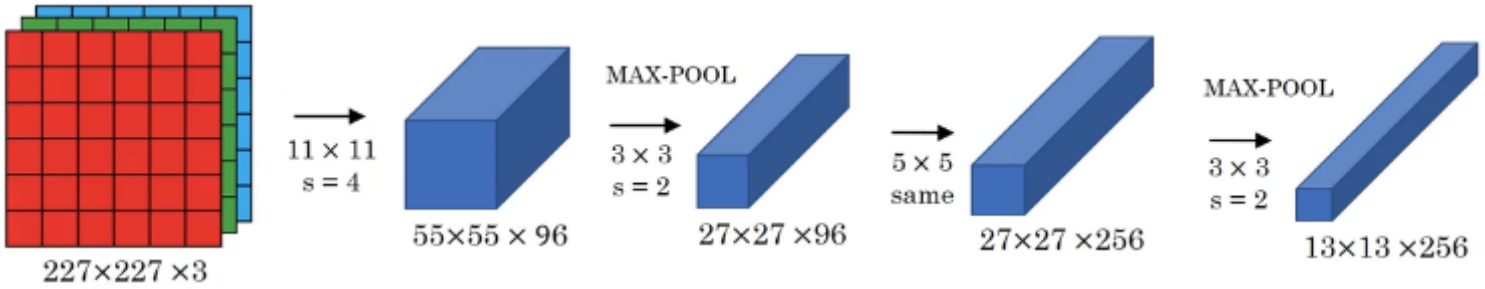

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

## 3.2.1. AlexNet



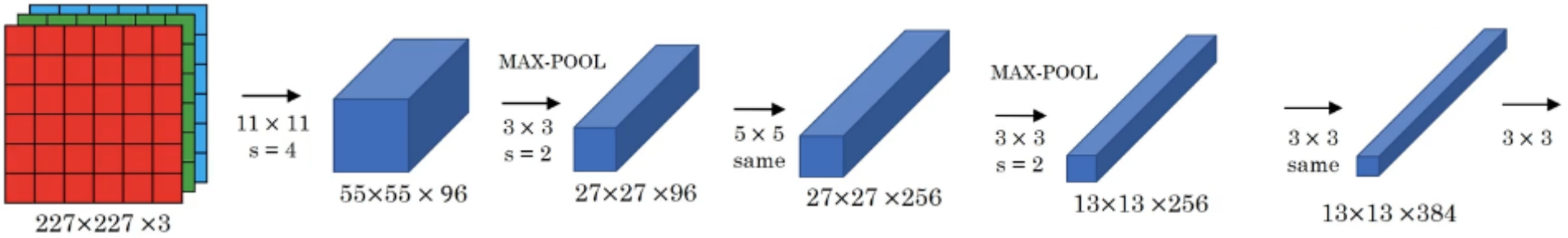

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

## 3.2.1. AlexNet



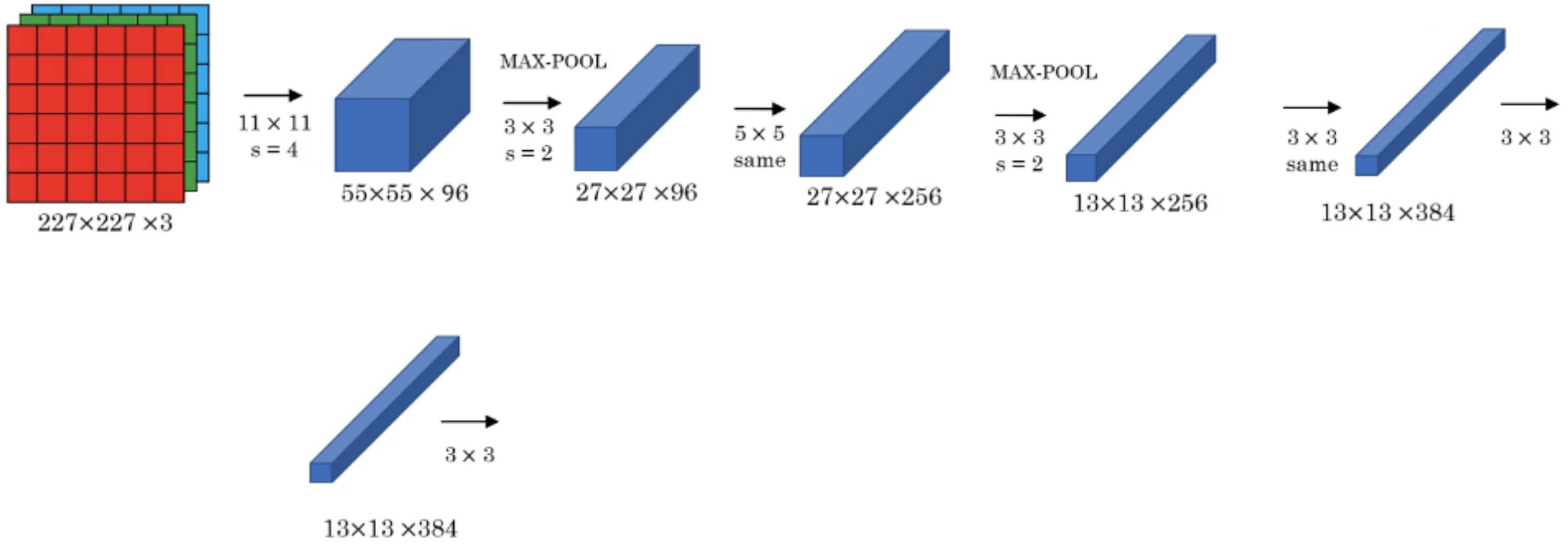

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

## 3.2.1. AlexNet



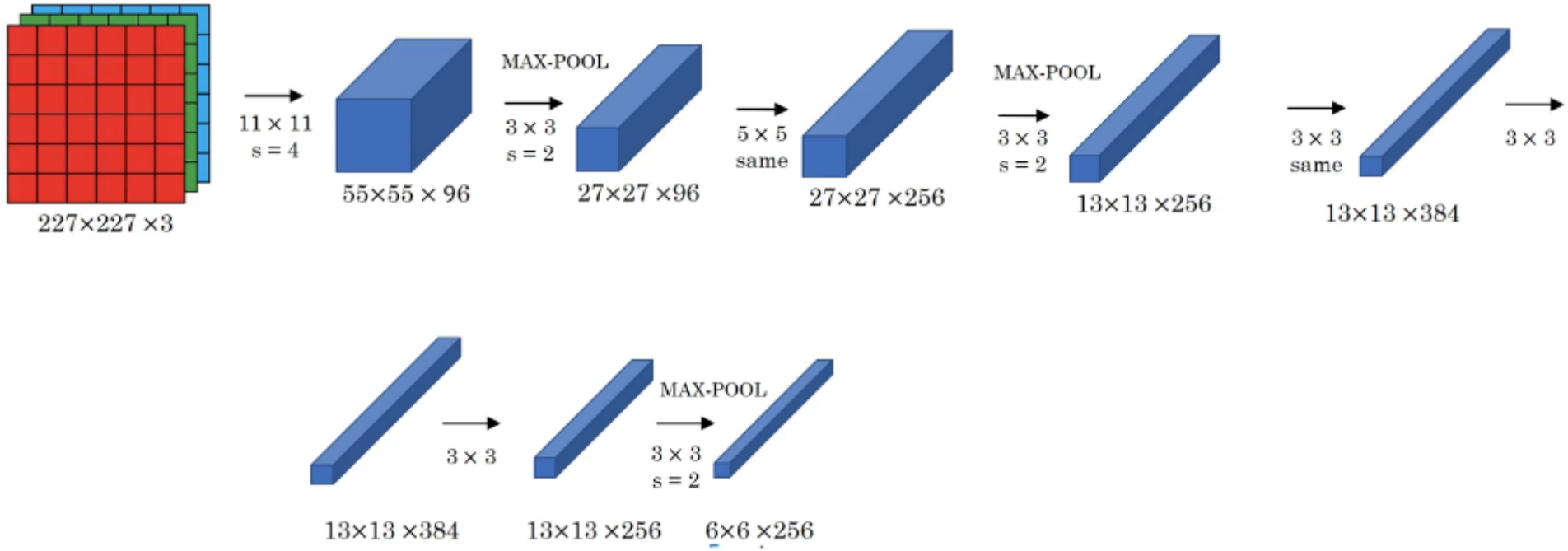

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

## 3.2.1. AlexNet



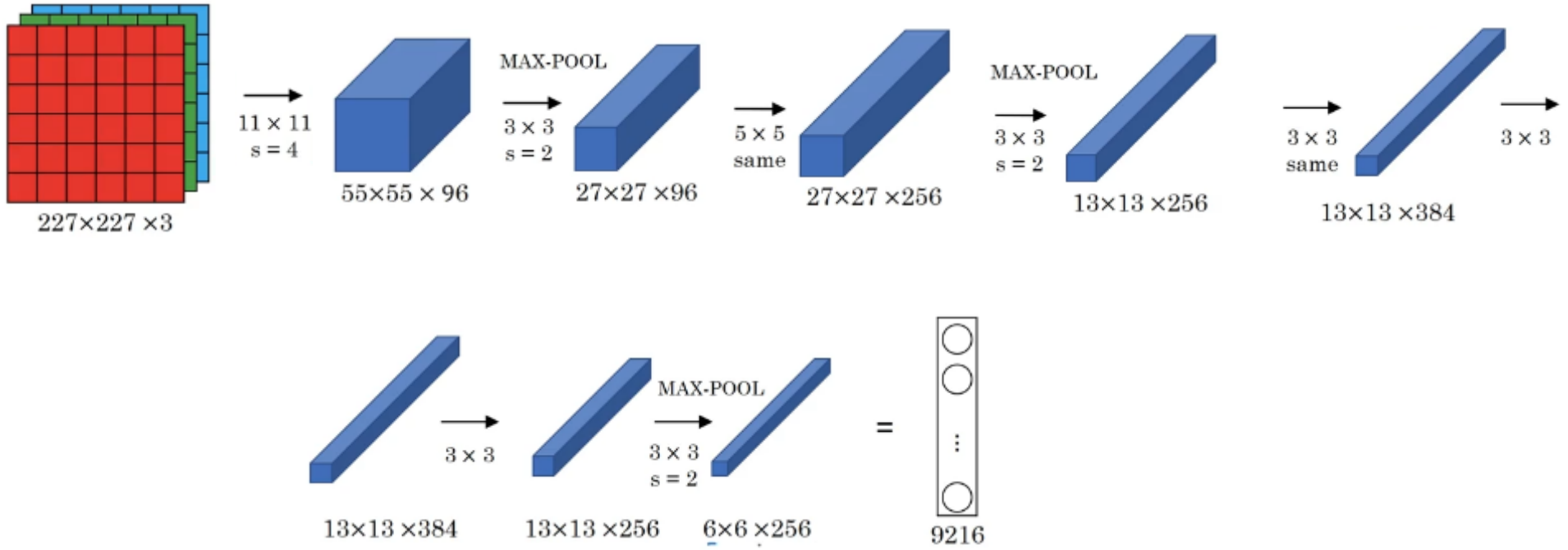

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

## 3.2.1. AlexNet



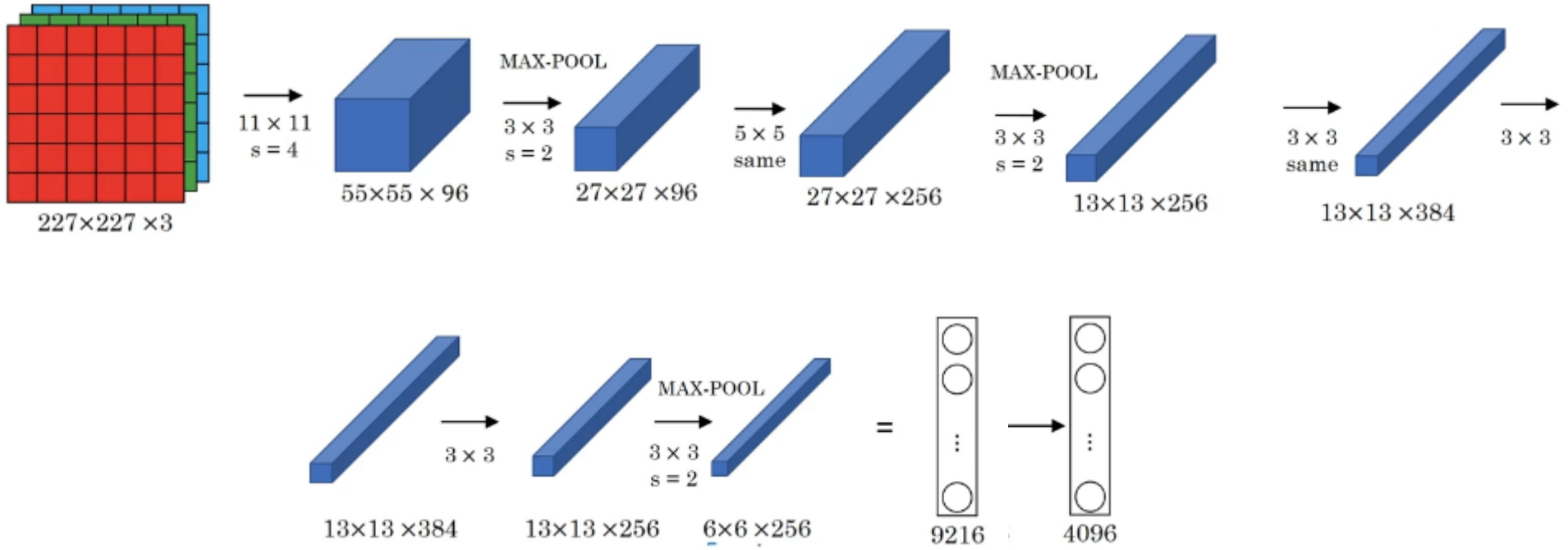

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

## 3.2.1. AlexNet



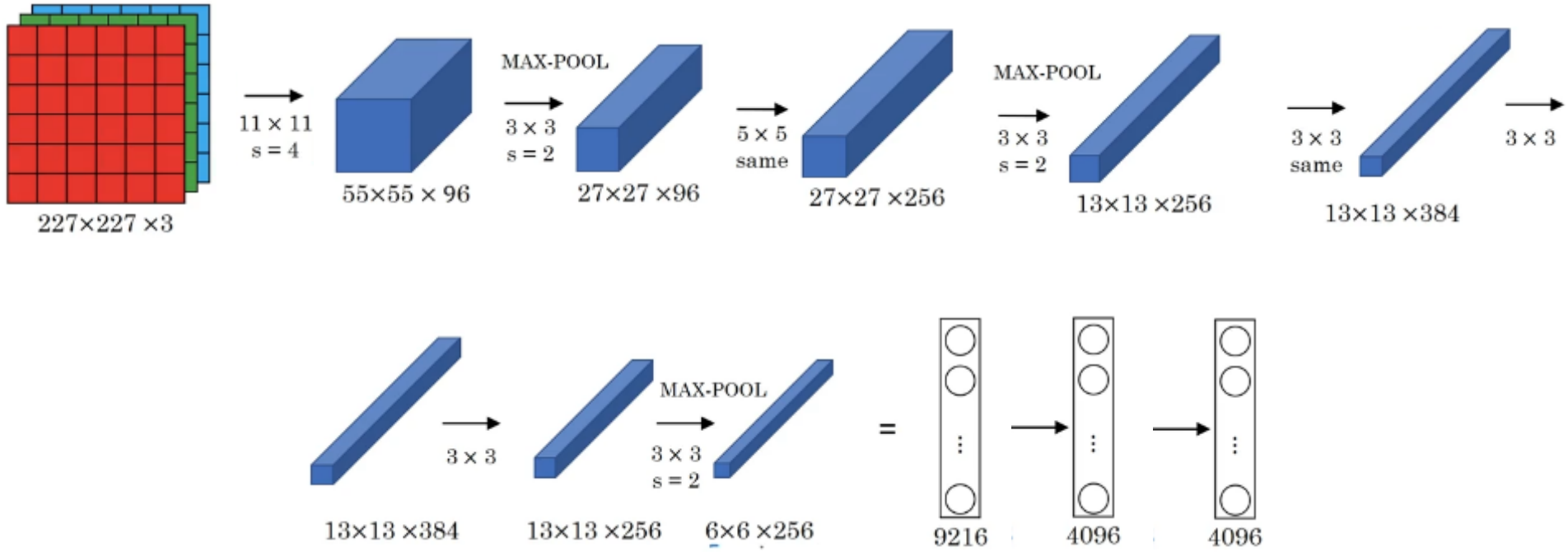

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

## 3.2.1. AlexNet



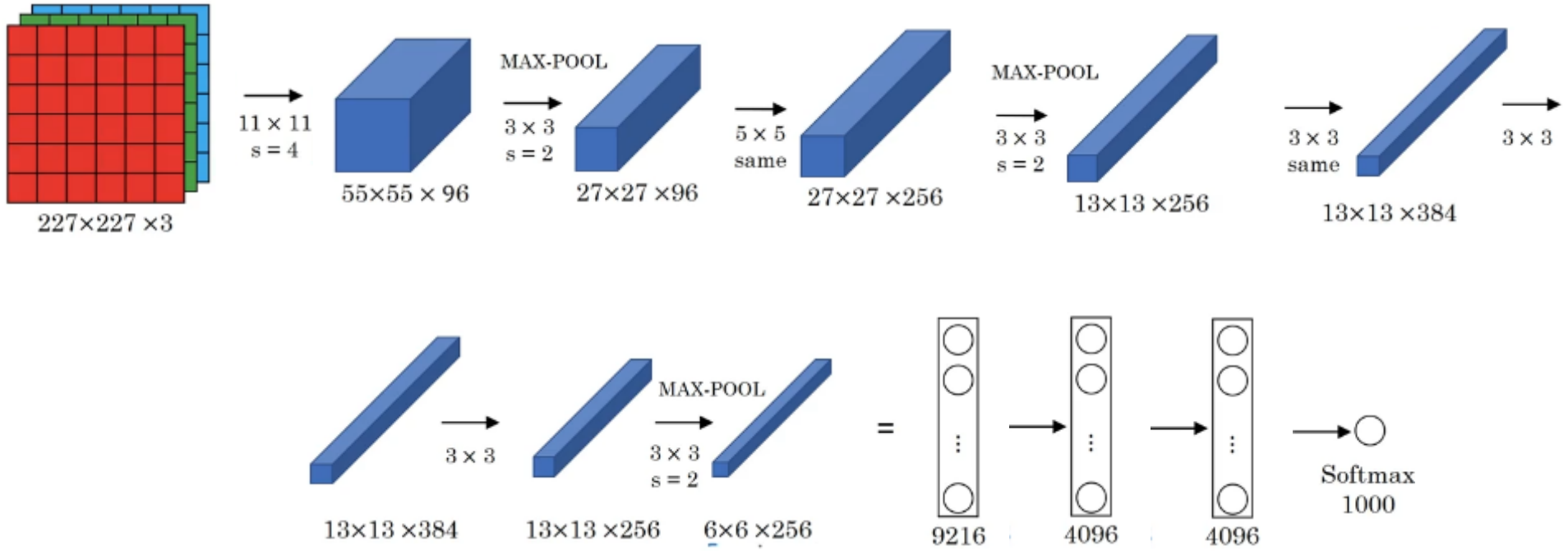

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

## 3.2.1. AlexNet



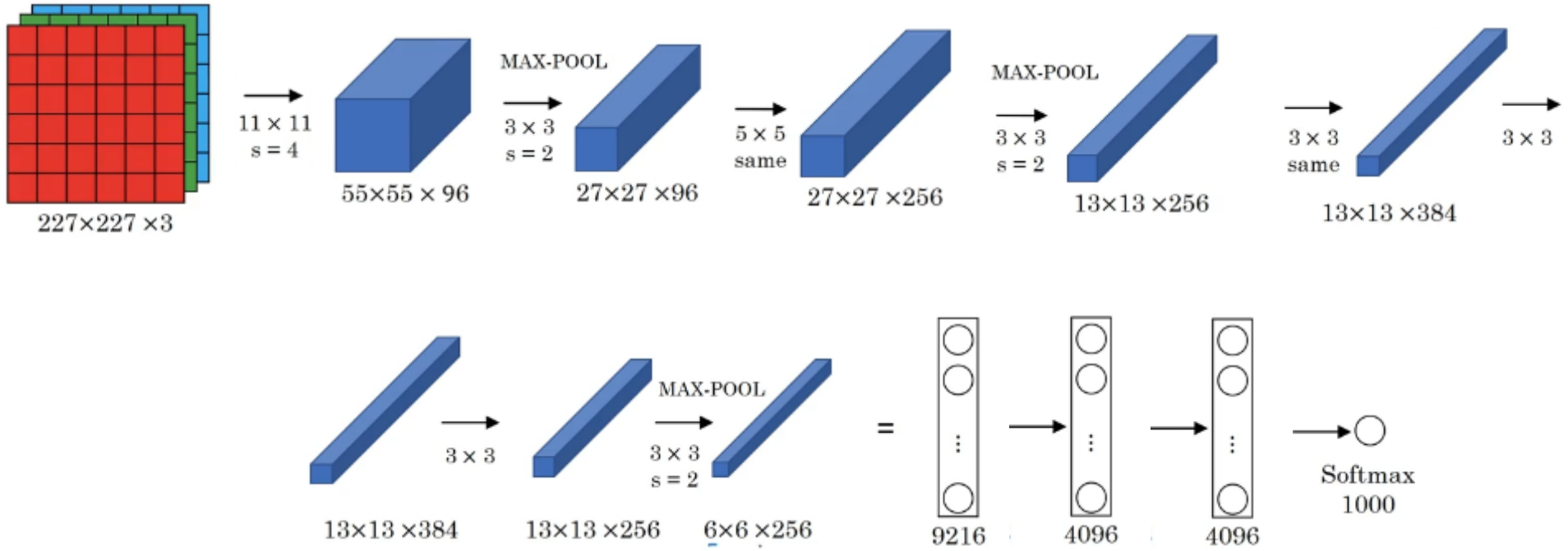

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

## 3.2.1. AlexNet




https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

## 3.2.1. AlexNet




https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

~60 Millones de parámetros

## 3.2.2. VGG - 16

Sólo usa CONV con filtros de 3x3, s=1, same

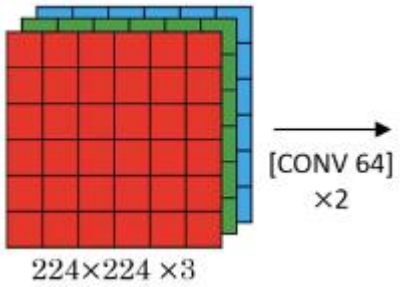
MAX-POOL = 2x2, s=2

https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

Sólo usa CONV con filtros de 3x3, s=1, same

MAX-POOL = 2x2, s=2

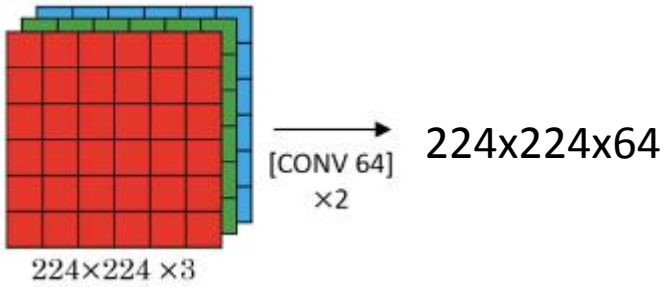



https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

Sólo usa CONV con filtros de 3x3, s=1, same

MAX-POOL = 2x2, s=2

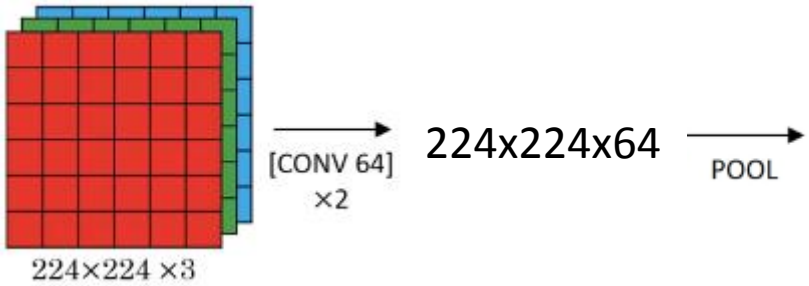



https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

Sólo usa CONV con filtros de 3x3, s=1, same

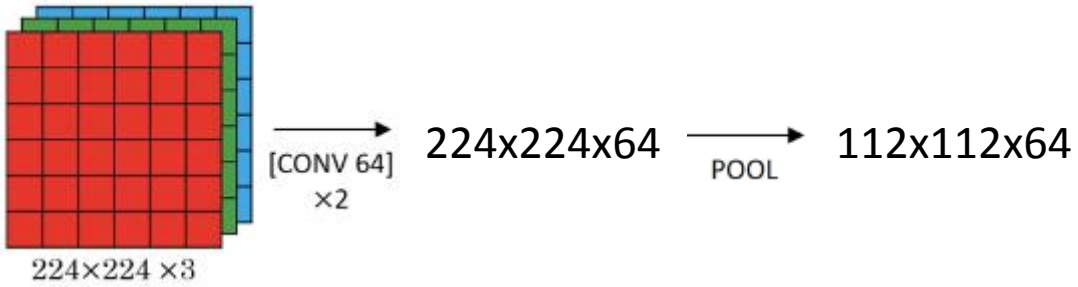
MAX-POOL = 2x2, s=2




https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

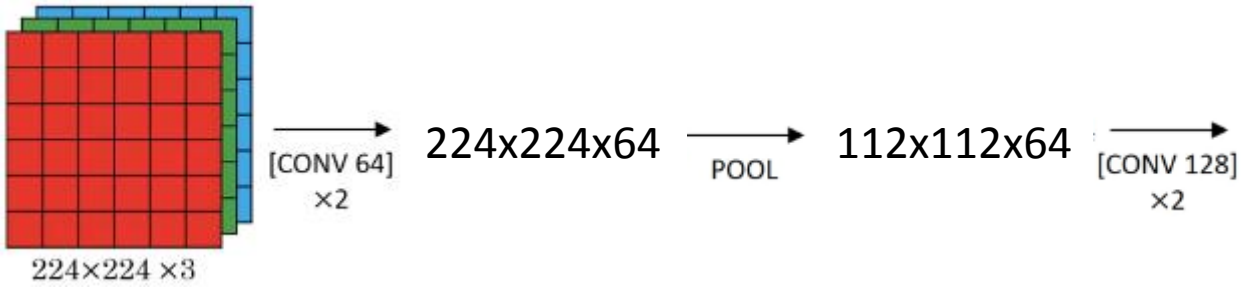
Sólo usa CONV con filtros de 3x3, s=1, same MAX-POOL = 2x2, s=2




https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

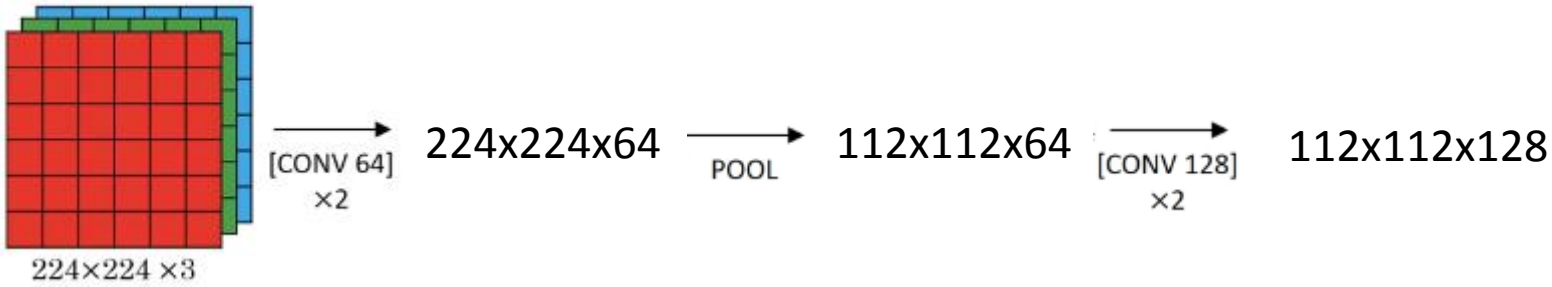
Sólo usa CONV con filtros de 3x3, s=1, same MAX-POOL = 2x2, s=2




https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

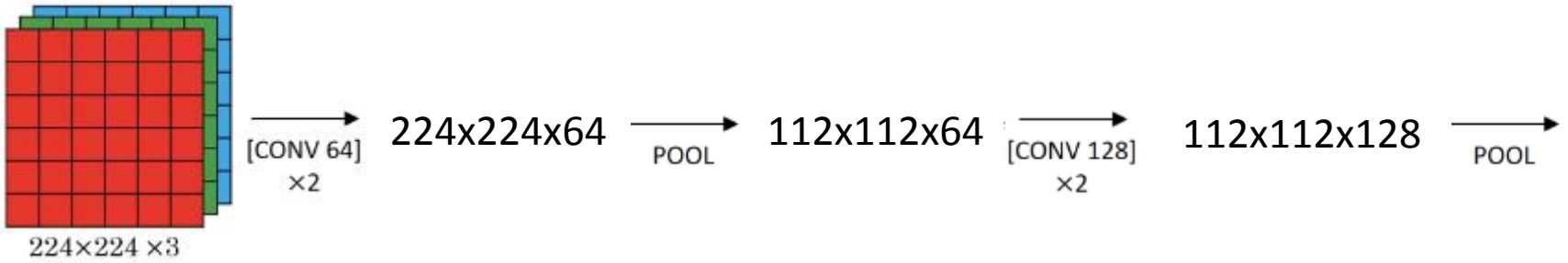
Sólo usa CONV con filtros de 3x3, s=1, same MAX-POOL = 2x2, s=2




https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

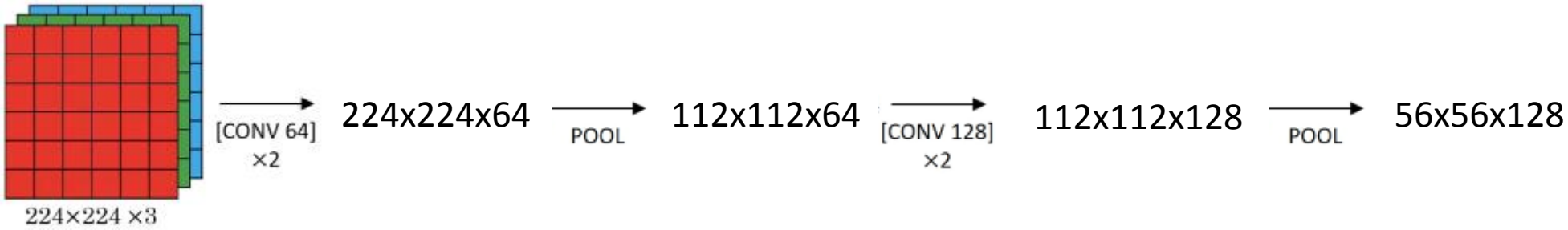
Sólo usa CONV con filtros de 3x3, s=1, same MAX-POOL = 2x2, s=2




https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

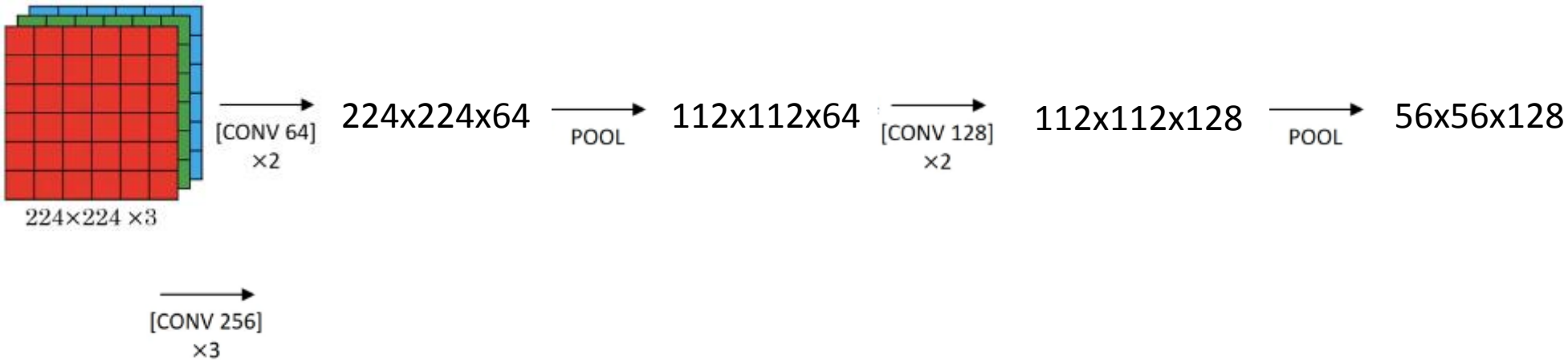
Sólo usa CONV con filtros de 3x3, s=1, same MAX-POOL = 2x2, s=2




https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

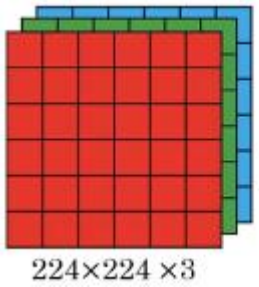
Sólo usa CONV con filtros de 3x3, s=1, same MAX-POOL = 2x2, s=2




https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

Sólo usa CONV con filtros de 3x3, s=1, same MAX-POOL = 2x2, s=2



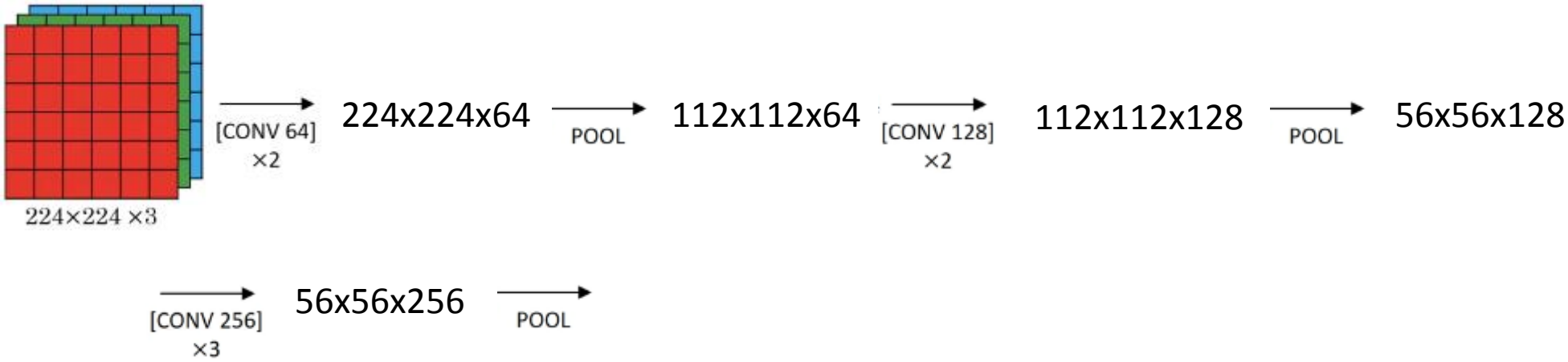

→ [CONV 64] ×2 → 224x224x64 → POOL → 112x112x64 → [CONV 128] ×2 → 112x112x128 → POOL → 56x56x128

→ [CONV 256] ×3 → 56x56x256

https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

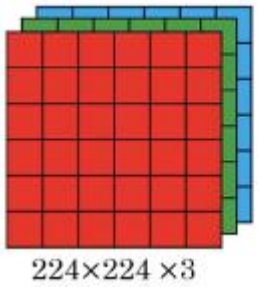
Sólo usa CONV con filtros de 3x3, s=1, same MAX-POOL = 2x2, s=2




https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

Sólo usa CONV con filtros de 3x3, s=1, same MAX-POOL = 2x2, s=2



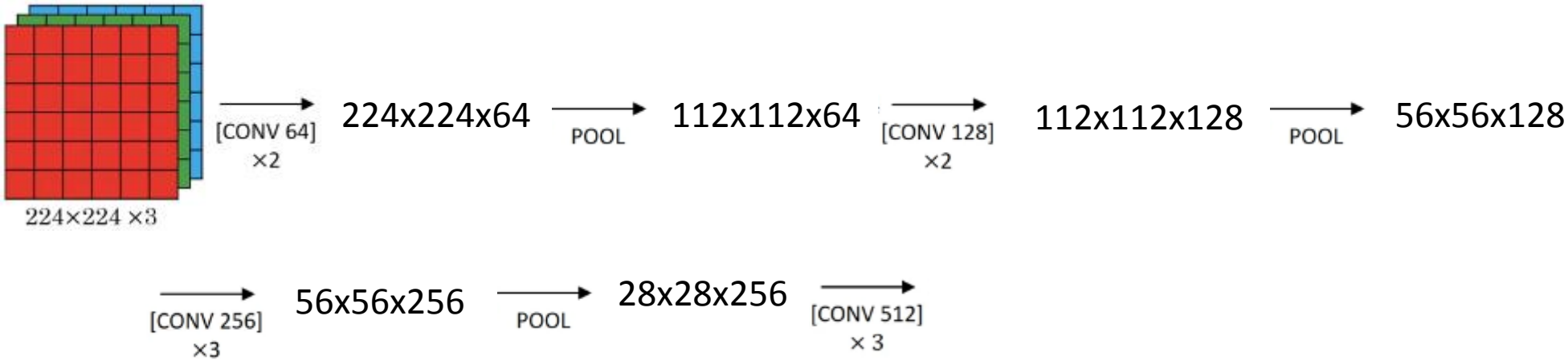

→ [CONV 64] ×2 → 224x224x64 → POOL → 112x112x64 → [CONV 128] ×2 → 112x112x128 → POOL → 56x56x128

→ [CONV 256] ×3 → 56x56x256 → POOL → 28x28x256

https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

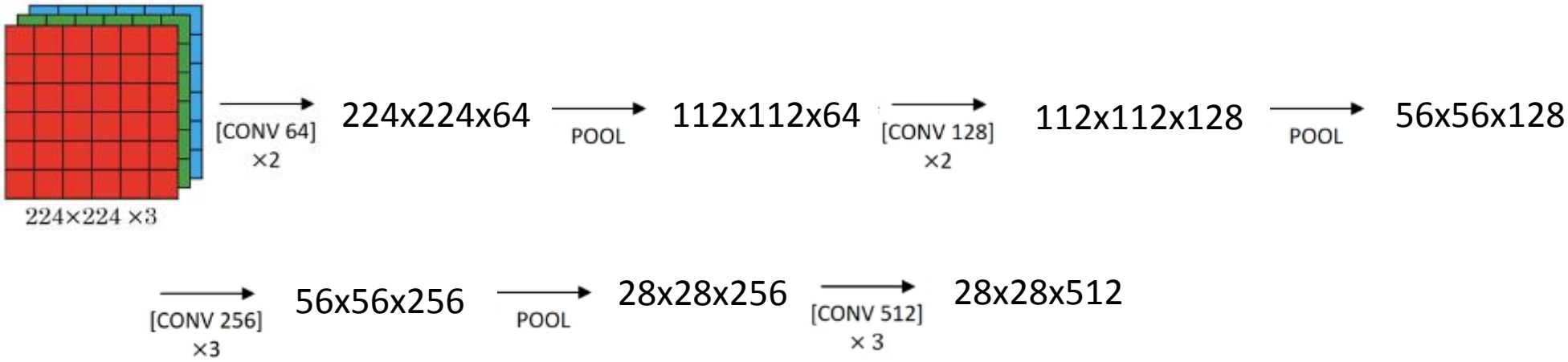
Sólo usa CONV con filtros de 3x3, s=1, same MAX-POOL = 2x2, s=2




https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

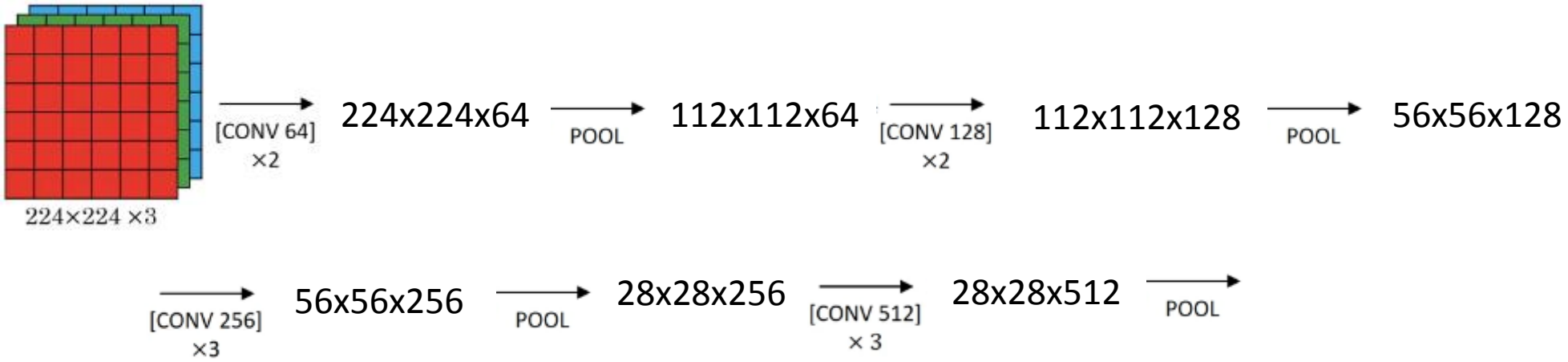
Sólo usa CONV con filtros de 3x3, s=1, same MAX-POOL = 2x2, s=2




https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

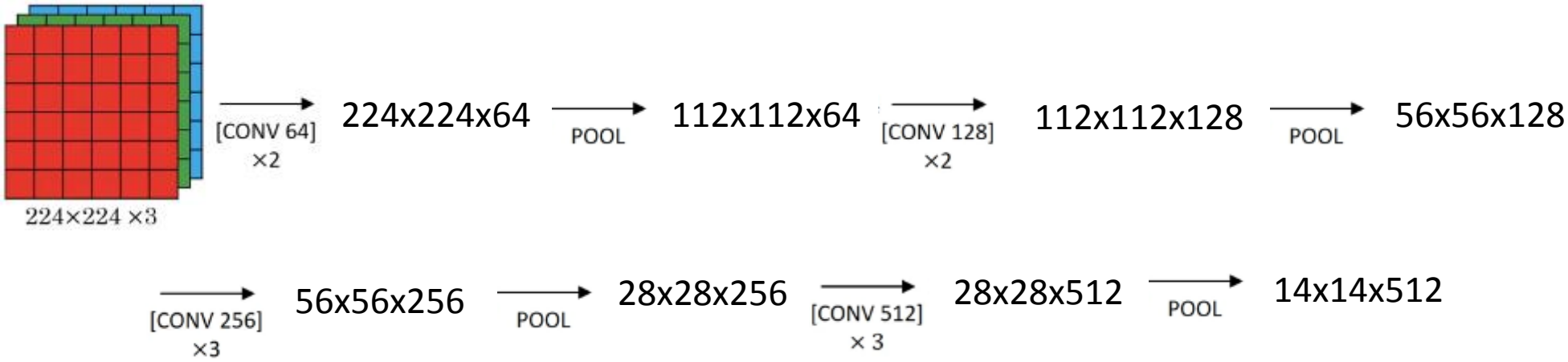
Sólo usa CONV con filtros de 3x3, s=1, same MAX-POOL = 2x2, s=2




https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

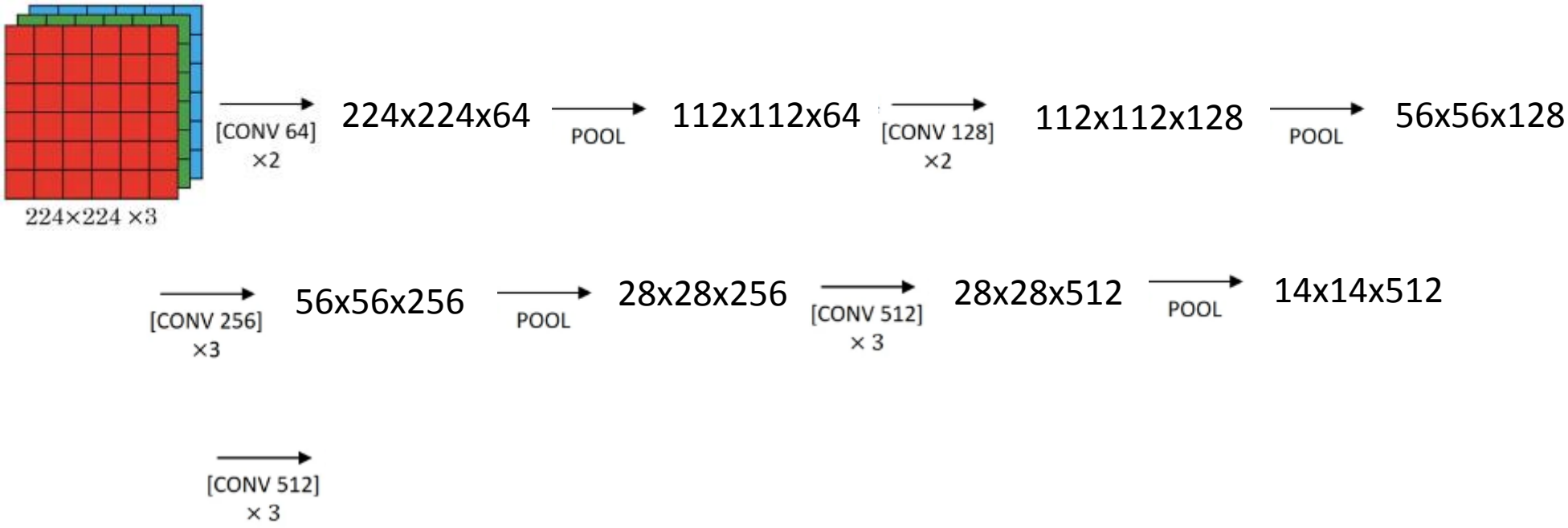
Sólo usa CONV con filtros de 3x3, s=1, same MAX-POOL = 2x2, s=2




https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

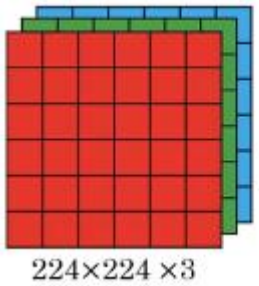
Sólo usa CONV con filtros de 3x3, s=1, same MAX-POOL = 2x2, s=2




https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

Sólo usa CONV con filtros de 3x3, s=1, same MAX-POOL = 2x2, s=2




→ [CONV 64] ×2 → 224x224x64 → POOL → 112x112x64 → [CONV 128] ×2 → 112x112x128 → POOL → 56x56x128

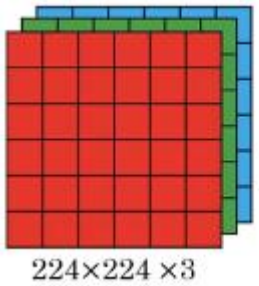
→ [CONV 256] ×3 → 56x56x256 → POOL → 28x28x256 → [CONV 512] × 3 → 28x28x512 → POOL → 14x14x512

→ [CONV 512] × 3 → 14x14x512

https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

Sólo usa CONV con filtros de 3x3, s=1, same MAX-POOL = 2x2, s=2




[CONV 64] ×2 → 224x224x64 → POOL → 112x112x64 → [CONV 128] ×2 → 112x112x128 → POOL → 56x56x128

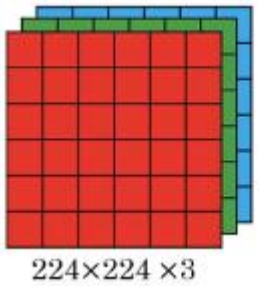
[CONV 256] ×3 → 56x56x256 → POOL → 28x28x256 → [CONV 512] × 3 → 28x28x512 → POOL → 14x14x512

[CONV 512] × 3 → 14x14x512 → POOL

https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

Sólo usa CONV con filtros de 3x3, s=1, same MAX-POOL = 2x2, s=2




→ [CONV 64] ×2 → 224x224x64 → POOL → 112x112x64 → [CONV 128] ×2 → 112x112x128 → POOL → 56x56x128

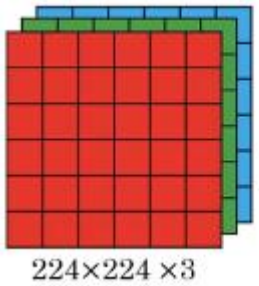
→ [CONV 256] ×3 → 56x56x256 → POOL → 28x28x256 → [CONV 512] × 3 → 28x28x512 → POOL → 14x14x512

→ [CONV 512] × 3 → 14x14x512 → POOL → 7x7x512

https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

Sólo usa CONV con filtros de 3x3, s=1, same MAX-POOL = 2x2, s=2




[CONV 64] ×2 → 224x224x64 → POOL → 112x112x64 → [CONV 128] ×2 → 112x112x128 → POOL → 56x56x128

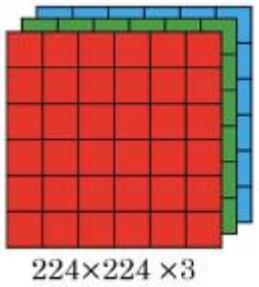
→ [CONV 256] ×3 → 56x56x256 → POOL → 28x28x256 → [CONV 512] × 3 → 28x28x512 → POOL → 14x14x512

→ [CONV 512] × 3 → 14x14x512 → POOL → 7x7x512 → FC 4096 → FC 4096

https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

Sólo usa CONV con filtros de 3x3, s=1, same MAX-POOL = 2x2, s=2




[CONV 64] ×2 → 224x224x64 → POOL → 112x112x64 → [CONV 128] ×2 → 112x112x128 → POOL → 56x56x128

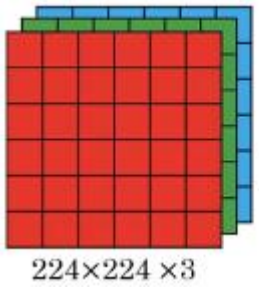
→ [CONV 256] ×3 → 56x56x256 → POOL → 28x28x256 → [CONV 512] × 3 → 28x28x512 → POOL → 14x14x512

→ [CONV 512] × 3 → 14x14x512 → POOL → 7x7x512 → FC 4096 → FC 4096 → Softmax 1000

https://arxiv.org/pdf/1409.1556.pdf

## 3.2.2. VGG - 16

Sólo usa CONV con filtros de 3x3, s=1, same MAX-POOL = 2x2, s=2




[CONV 64] ×2 → 224x224x64 → POOL → 112x112x64 → [CONV 128] ×2 → 112x112x128 → POOL → 56x56x128

→ [CONV 256] ×3 → 56x56x256 → POOL → 28x28x256 → [CONV 512] × 3 → 28x28x512 → POOL → 14x14x512

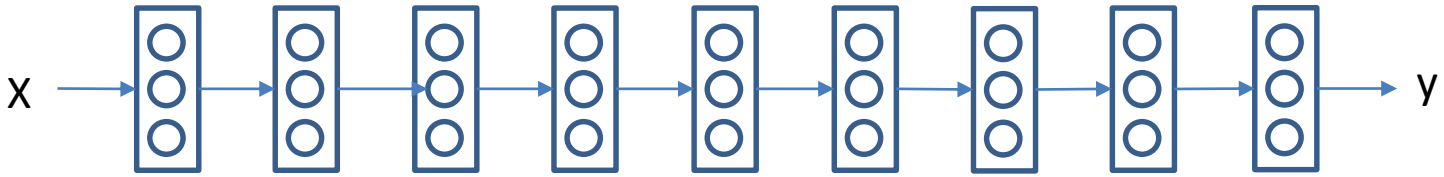
→ [CONV 512] × 3 → 14x14x512 → POOL → 7x7x512 → FC 4096 → FC 4096 → Softmax 1000

https://arxiv.org/pdf/1409.1556.pdf

~138 Millones de parámetros

## 3.2.3. ResNets

Problema: Vanishing/Exploding Gradients



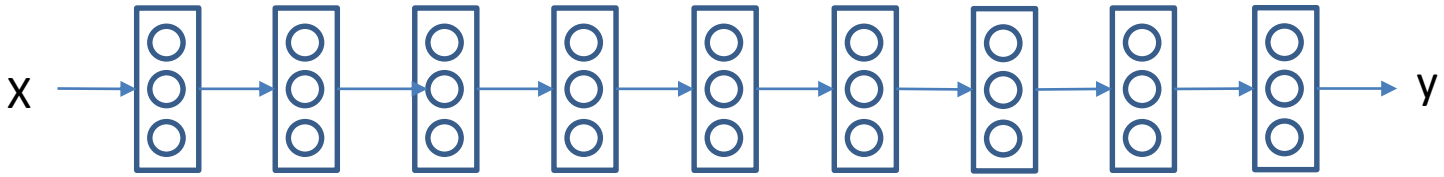

$$y = W_1 W_2 W_3 W_4 W_5 \dots W_L x$$

Si $W > 1$: Exploding

Si $W < 1$: Vanishing

## 3.2.3. ResNets

Problema: Vanishing/Exploding Gradients



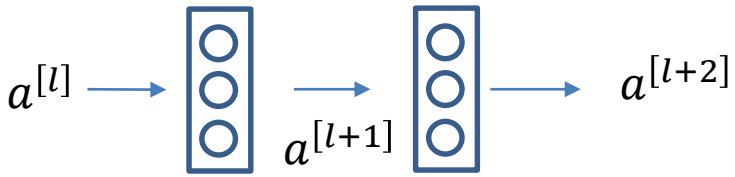

$$y = W_1 W_2 W_3 W_4 W_5 \dots W_L x$$

Si $W > 1$: Exploding $y = 1.5^L x$

Si $W < 1$: Vanishing $y = 0.5^L x$

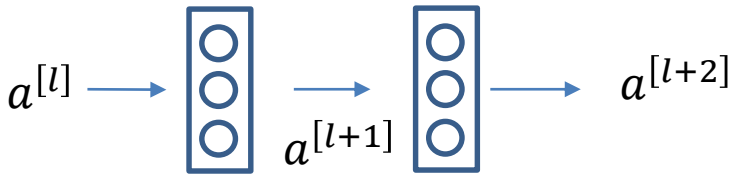
## 3.2.3. ResNets

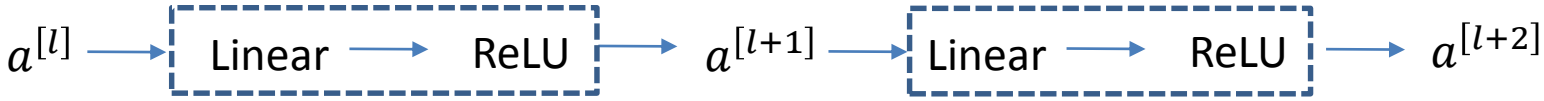
### Residual Block



## 3.2.3. ResNets

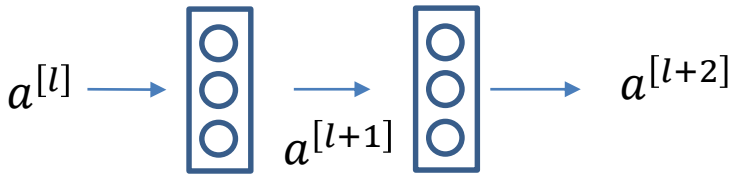
### Residual Block

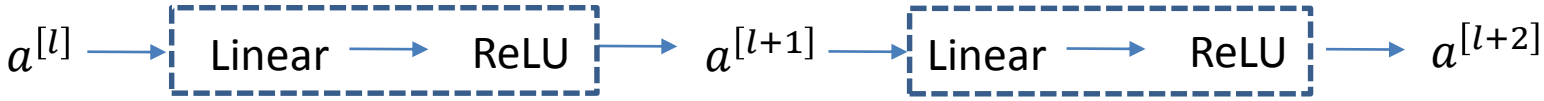


## 3.2.3. ResNets

### Residual Block

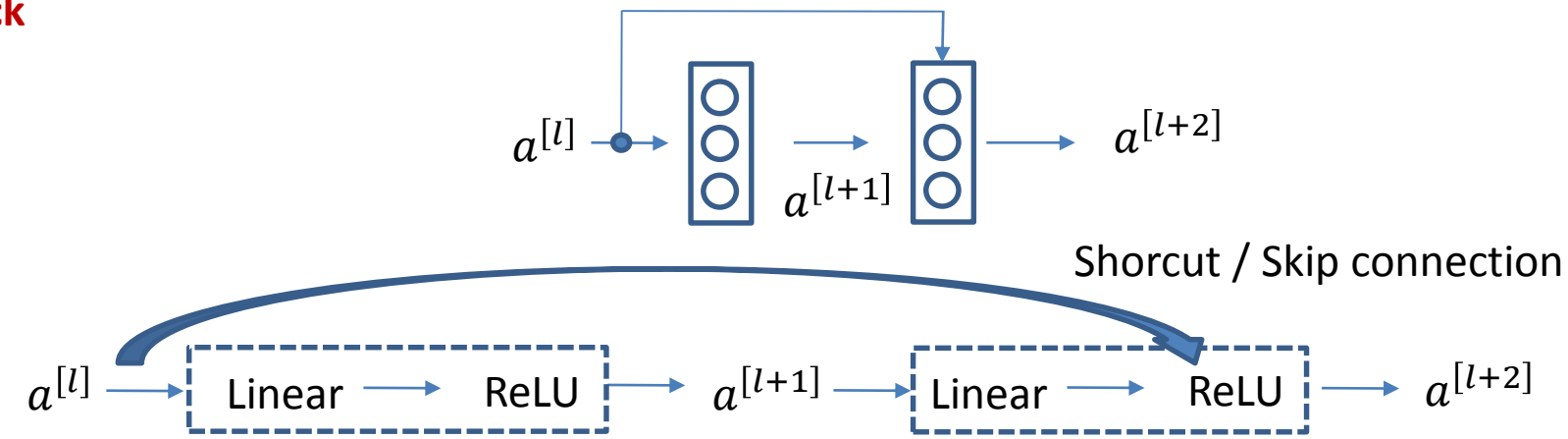



$$z^{[l+1]} = W^{[l+1]}a^{[l]} + b^{[l+1]} \quad a^{[l+1]} = g(z^{[l+1]}) \qquad z^{[l+2]} = W^{[l+2]}a^{[l+1]} + b^{[l+2]} \quad a^{[l+2]} = g(z^{[l+2]})$$

## 3.2.3. ResNets

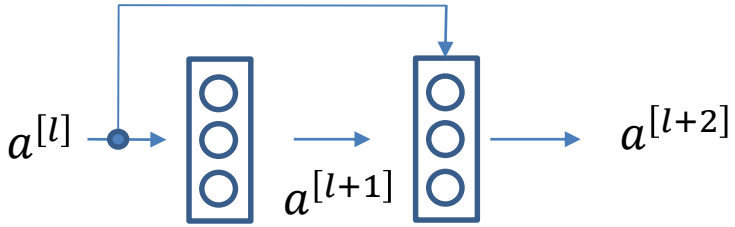
### Residual Block

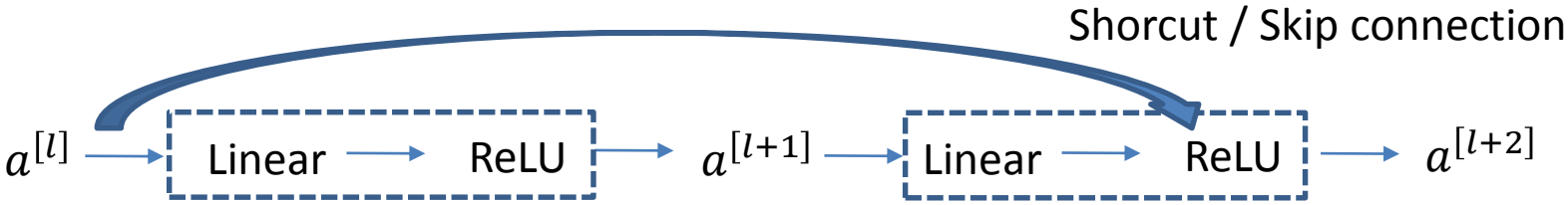



$$z^{[l+1]} = W^{[l+1]}a^{[l]} + b^{[l+1]} \quad a^{[l+1]} = g(z^{[l+1]}) \qquad z^{[l+2]} = W^{[l+2]}a^{[l+1]} + b^{[l+2]} \quad a^{[l+2]} = g(z^{[l+2]})$$

## 3.2.3. ResNets

### Residual Block



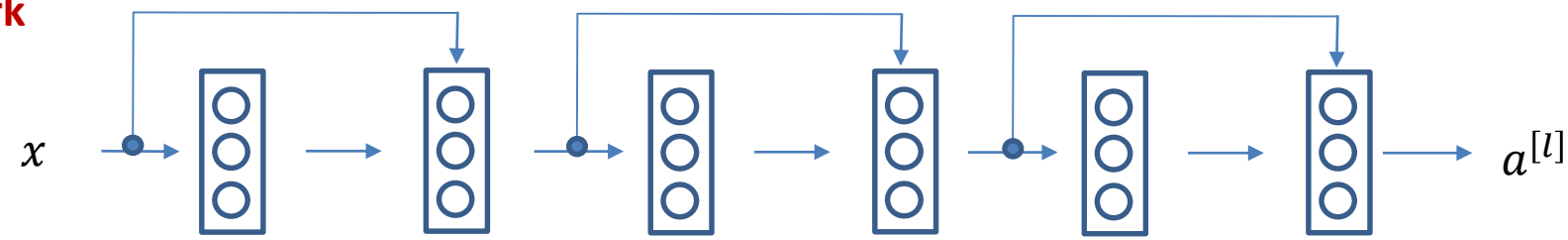

$$z^{[l+1]} = W^{[l+1]}a^{[l]} + b^{[l+1]} \quad a^{[l+1]} = g(z^{[l+1]})$$

$$z^{[l+2]} = W^{[l+2]}a^{[l+1]} + b^{[l+2]} \quad a^{[l+2]} = g($$  $$)$$

$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

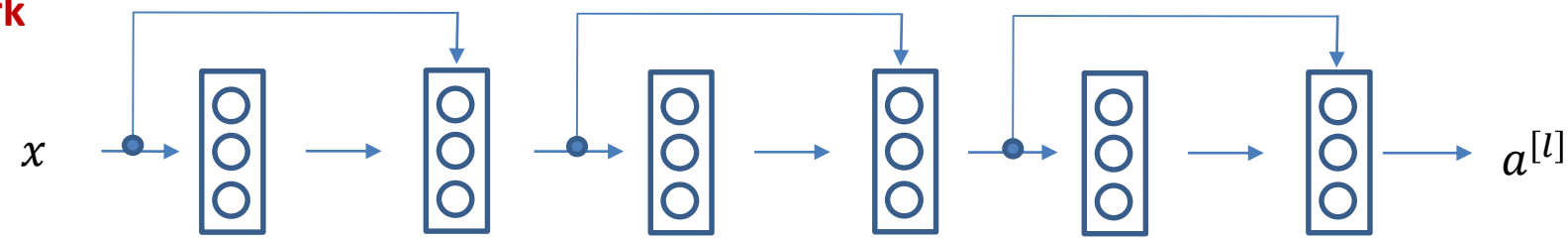
## 3.2.3. ResNets

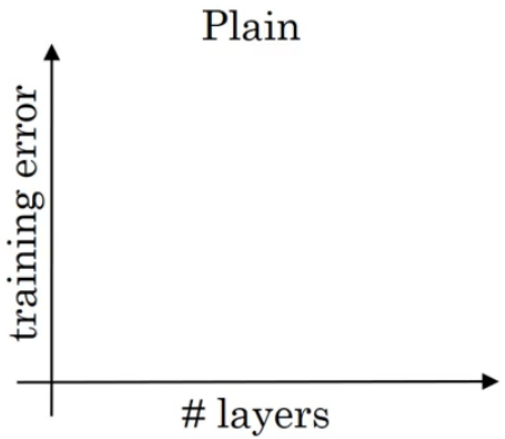
**Residual Network**



## 3.2.3. ResNets

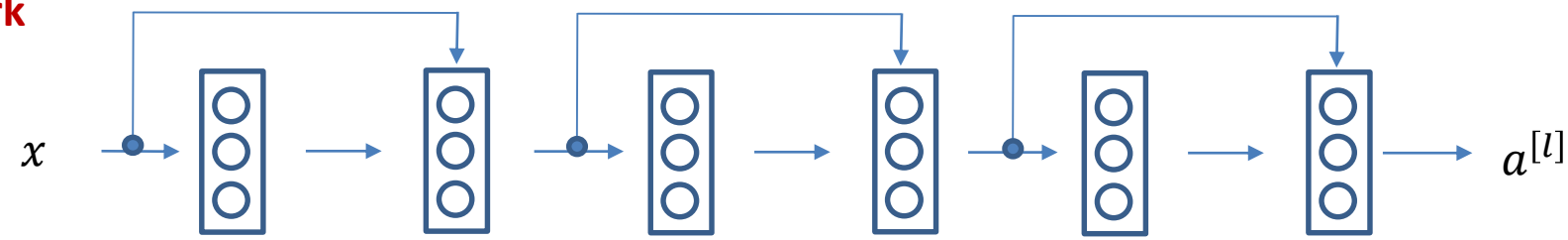
**Residual Network**

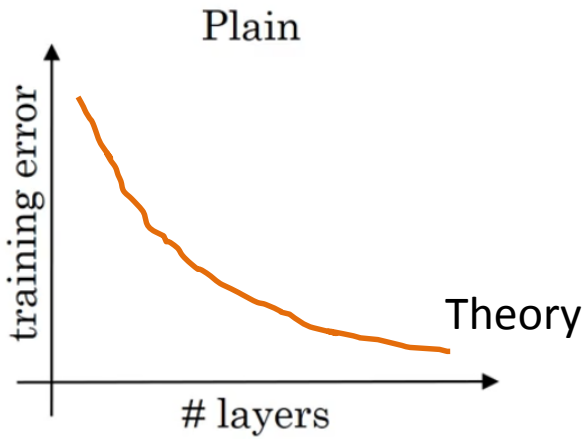


## 3.2.3. ResNets

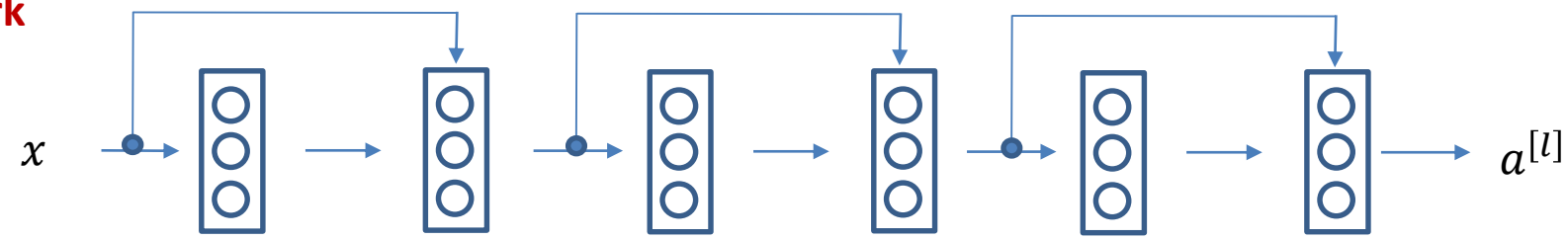
**Residual Network**

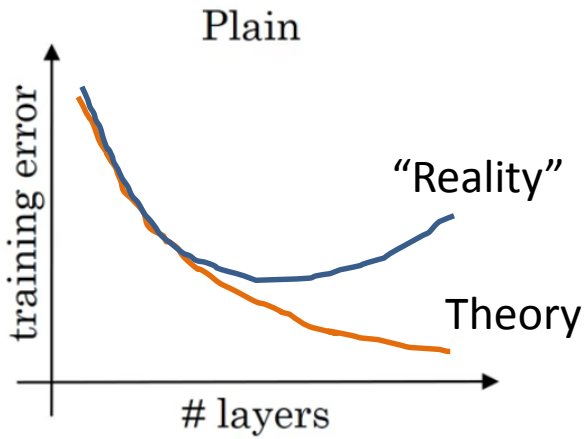


## 3.2.3. ResNets

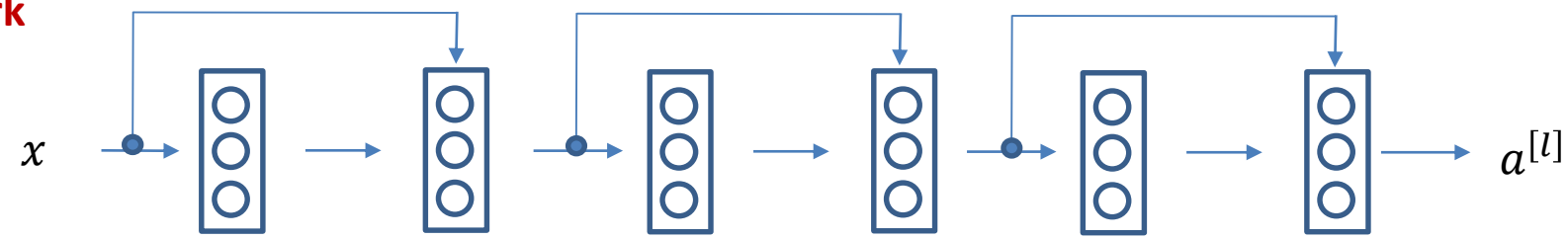
### Residual Network

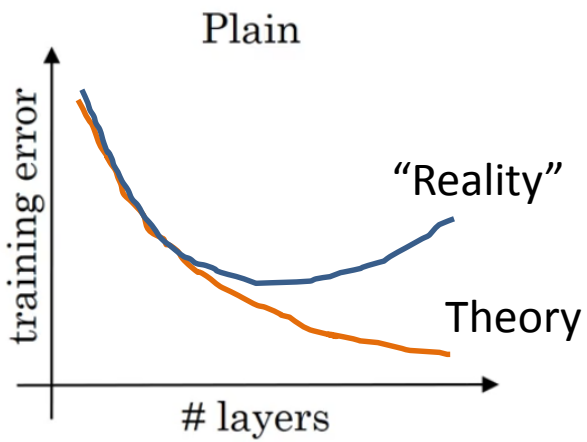


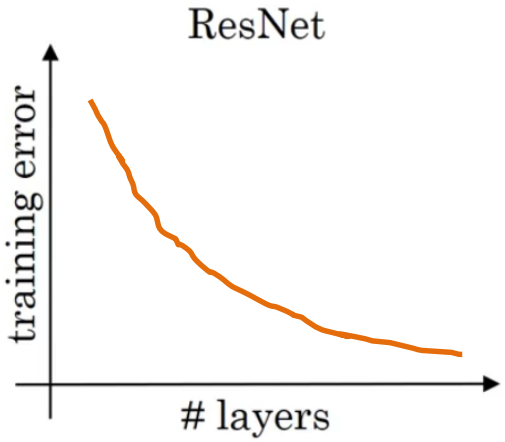



## 3.2.3. ResNets

### Residual Network

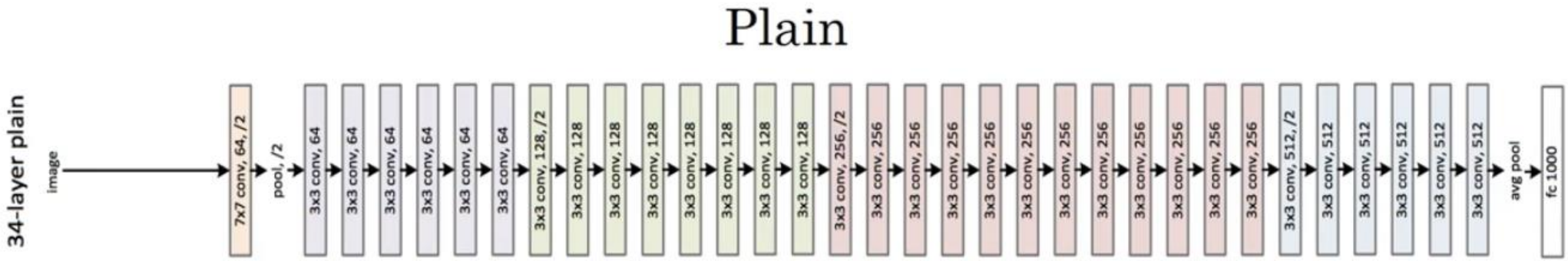


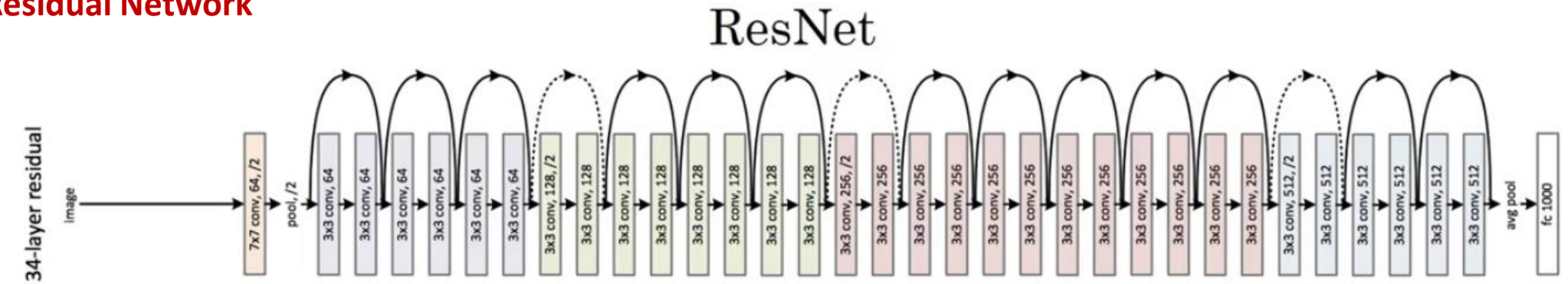



## 3.2.3. ResNets

### Plain Network




### Residual Network



## 3.2.4. Network in Network (1x1 convolution)

| | | | | | |
|---|---|---|---|---|---|
| 2 | 3 | 7 | 4 | 6 | 2 |
| 6 | 6 | 9 | 8 | 7 | 4 |
| 3 | 4 | 8 | 3 | 8 | 9 |
| 7 | 8 | 3 | 6 | 6 | 3 |
| 4 | 2 | 1 | 8 | 3 | 4 |
| 3 | 2 | 4 | 1 | 9 | 8 |

6 x 6 x 1

* 2 =

## 3.2.4. Network in Network (1x1 convolution)

| | | | | | |
|---|---|---|---|---|---|
| 2 | 3 | 7 | 4 | 6 | 2 |
| 6 | 6 | 9 | 8 | 7 | 4 |
| 3 | 4 | 8 | 3 | 8 | 9 |
| 7 | 8 | 3 | 6 | 6 | 3 |
| 4 | 2 | 1 | 8 | 3 | 4 |
| 3 | 2 | 4 | 1 | 9 | 8 |

6 x 6 x 1

* 2 =

| | | | | | |
|---|---|---|---|---|---|
| 4 | 6 | 14 | … | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

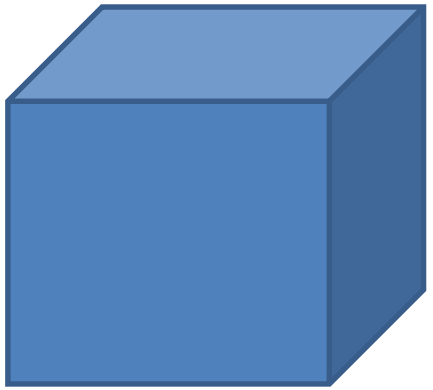
## 3.2.4. Network in Network (1x1 convolution)

| | | | | | |
|---|---|---|---|---|---|
| 2 | 3 | 7 | 4 | 6 | 2 |
| 6 | 6 | 9 | 8 | 7 | 4 |
| 3 | 4 | 8 | 3 | 8 | 9 |
| 7 | 8 | 3 | 6 | 6 | 3 |
| 4 | 2 | 1 | 8 | 3 | 4 |
| 3 | 2 | 4 | 1 | 9 | 8 |

6 x 6 x 1

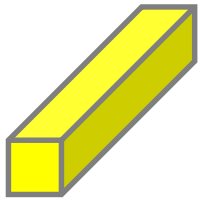
* 2 =

| | | | | | |
|---|---|---|---|---|---|
| 4 | 6 | 14 | ... | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |



6 x 6 x 32

*



1 x 1 x 32

=

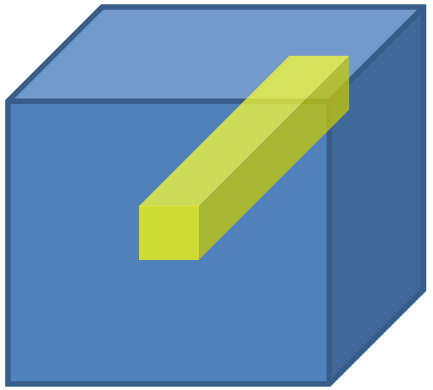
## 3.2.4. Network in Network (1x1 convolution)

| | | | | | |
|---|---|---|---|---|---|
| 2 | 3 | 7 | 4 | 6 | 2 |
| 6 | 6 | 9 | 8 | 7 | 4 |
| 3 | 4 | 8 | 3 | 8 | 9 |
| 7 | 8 | 3 | 6 | 6 | 3 |
| 4 | 2 | 1 | 8 | 3 | 4 |
| 3 | 2 | 4 | 1 | 9 | 8 |

6 x 6 x 1

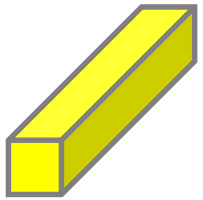
* 2 =

| | | | | | |
|---|---|---|---|---|---|
| 4 | 6 | 14 | ... | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |



6 x 6 x 32

*



1 x 1 x 32

=

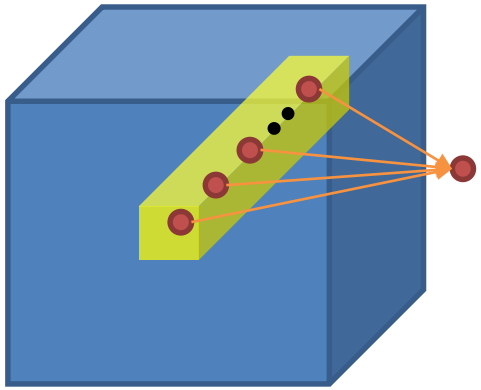
## 3.2.4. Network in Network (1x1 convolution)

| | | | | | |
|---|---|---|---|---|---|
| 2 | 3 | 7 | 4 | 6 | 2 |
| 6 | 6 | 9 | 8 | 7 | 4 |
| 3 | 4 | 8 | 3 | 8 | 9 |
| 7 | 8 | 3 | 6 | 6 | 3 |
| 4 | 2 | 1 | 8 | 3 | 4 |
| 3 | 2 | 4 | 1 | 9 | 8 |

6 x 6 x 1

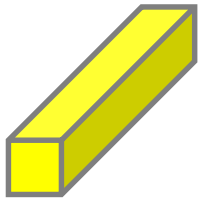
* 2 =

| | | | | | |
|---|---|---|---|---|---|
| 4 | 6 | 14 | ... | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |



6 x 6 x 32

*



1 x 1 x 32

=

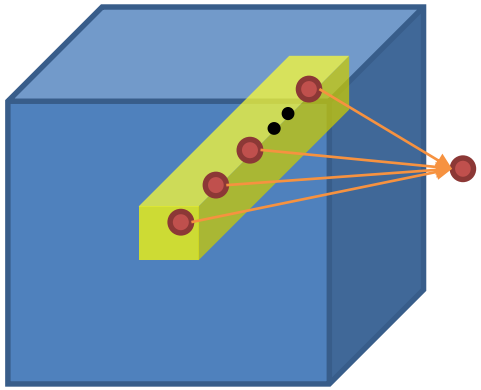
## 3.2.4. Network in Network (1x1 convolution)

| | | | | | |
|---|---|---|---|---|---|
| 2 | 3 | 7 | 4 | 6 | 2 |
| 6 | 6 | 9 | 8 | 7 | 4 |
| 3 | 4 | 8 | 3 | 8 | 9 |
| 7 | 8 | 3 | 6 | 6 | 3 |
| 4 | 2 | 1 | 8 | 3 | 4 |
| 3 | 2 | 4 | 1 | 9 | 8 |

6 x 6 x 1

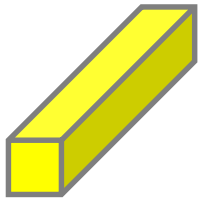
* 2 =

| | | | | | |
|---|---|---|---|---|---|
| 4 | 6 | 14 | ... | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |



6 x 6 x 32

*



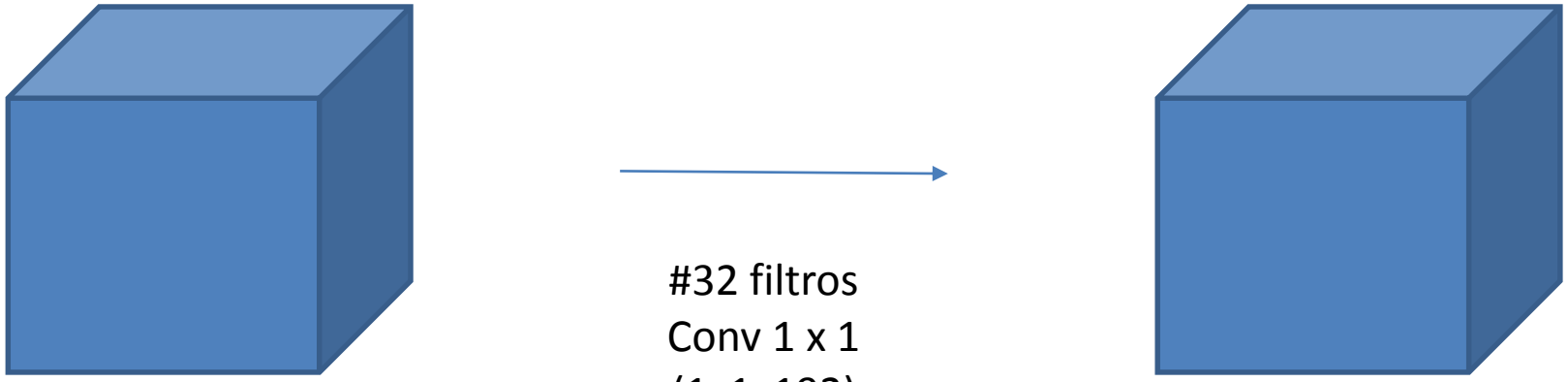
1 x 1 x 32

=

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

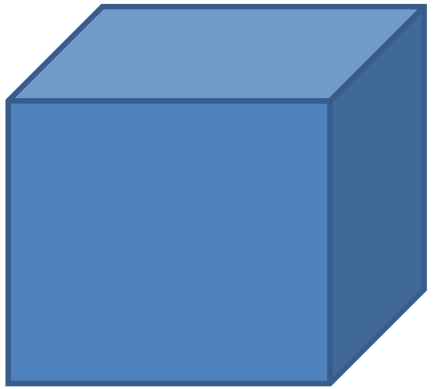
6 x 6 x #filtros

## 3.2.4. Network in Network (1x1 convolution)



## 3.2.5. Inception Network

### Inception block

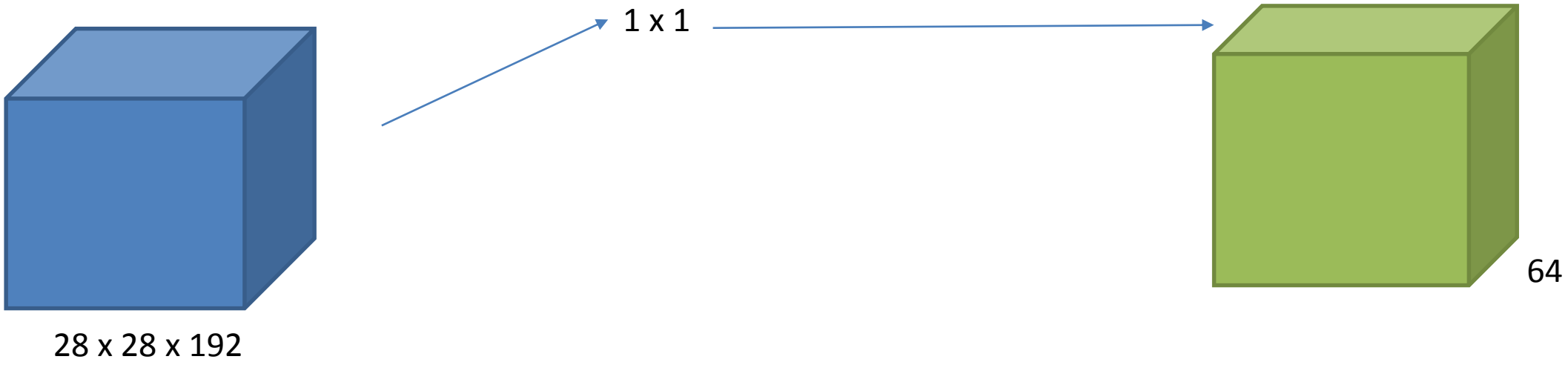


28 x 28 x 192

**Going Deeper with Convolutions**
https://arxiv.org/abs/1409.4842

## 3.2.5. Inception Network

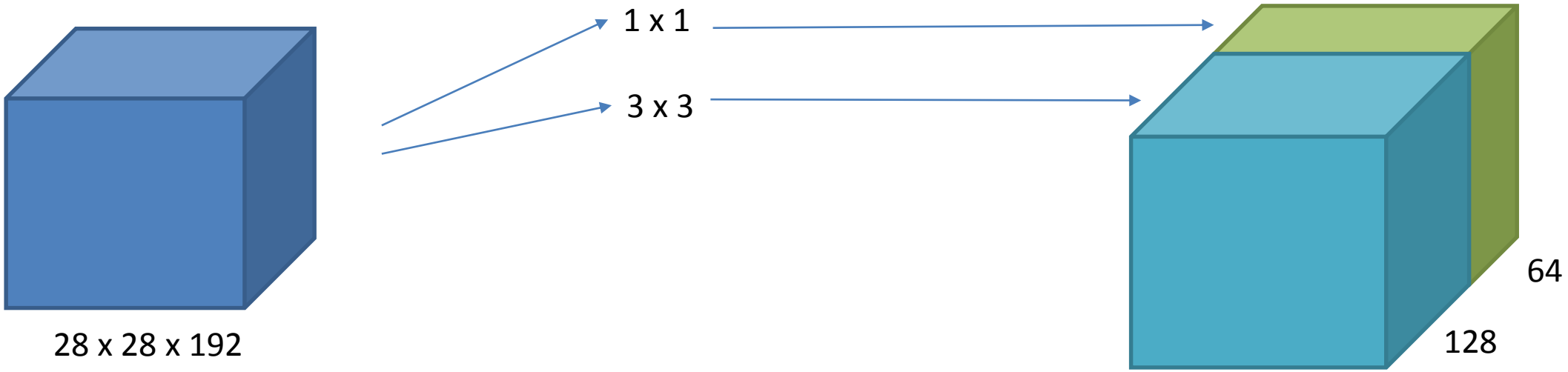
### Inception block




**Going Deeper with Convolutions**
https://arxiv.org/abs/1409.4842

## 3.2.5. Inception Network

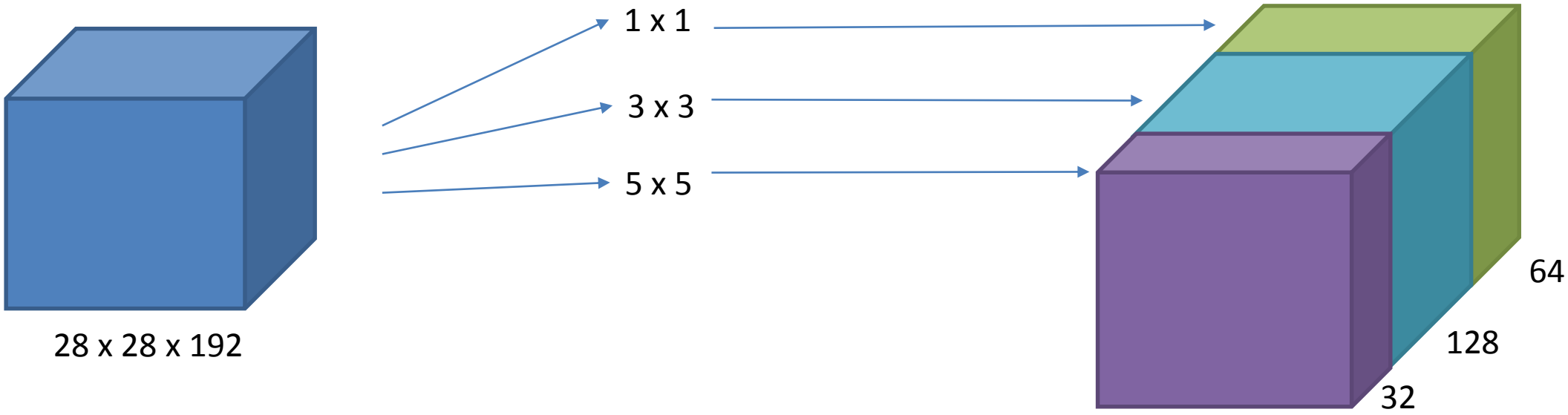
### Inception block




**Going Deeper with Convolutions**
https://arxiv.org/abs/1409.4842

## 3.2.5. Inception Network

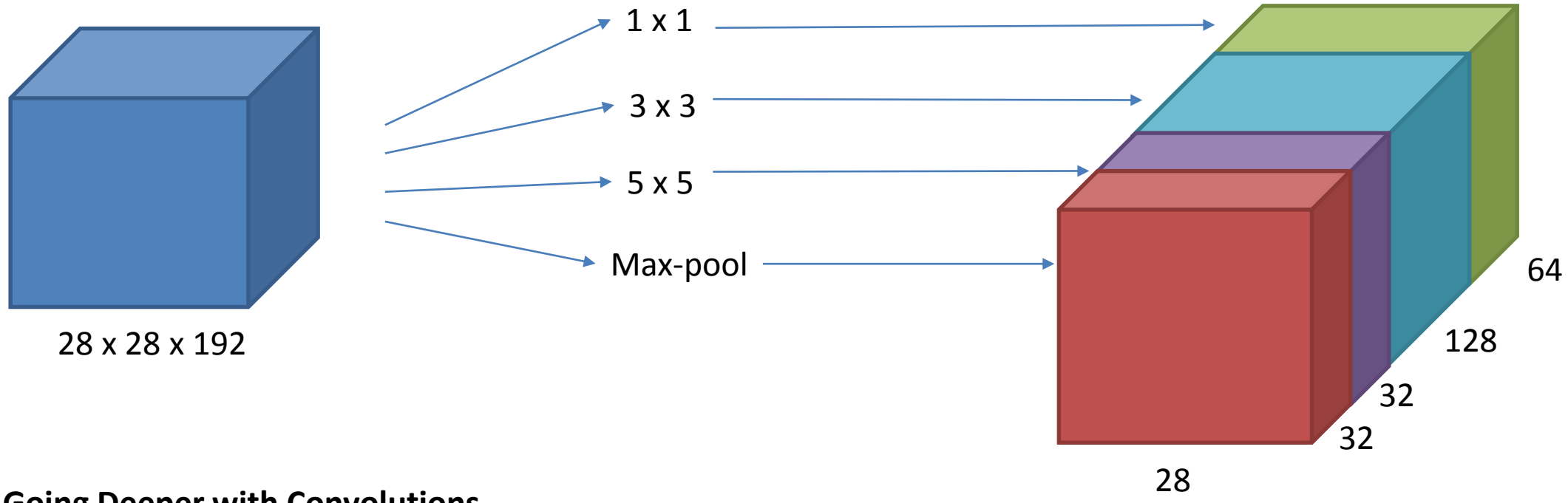
### Inception block




**Going Deeper with Convolutions**
https://arxiv.org/abs/1409.4842

## 3.2.5. Inception Network

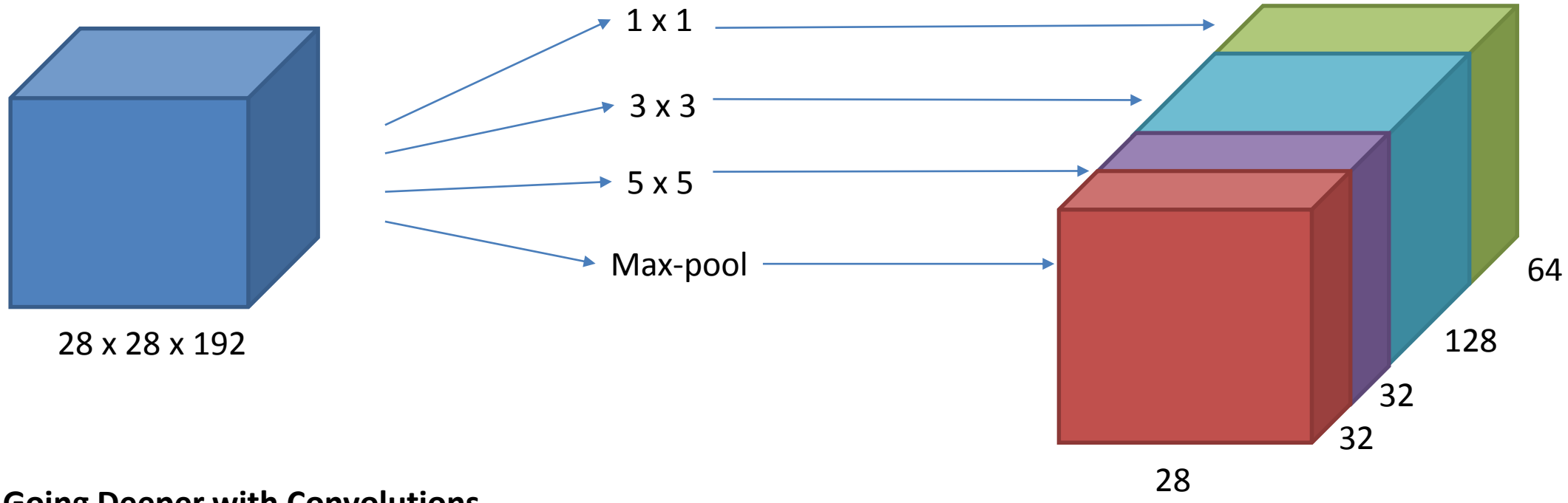
### Inception block




**Going Deeper with Convolutions**
https://arxiv.org/abs/1409.4842

## 3.2.5. Inception Network

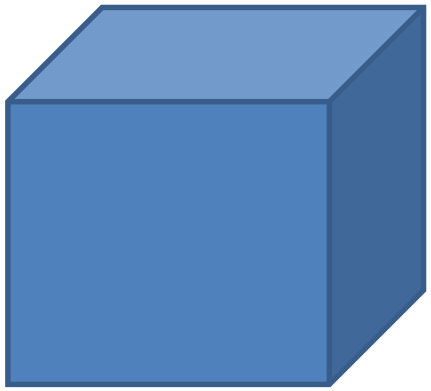
### Inception block




**Going Deeper with Convolutions**
https://arxiv.org/abs/1409.4842

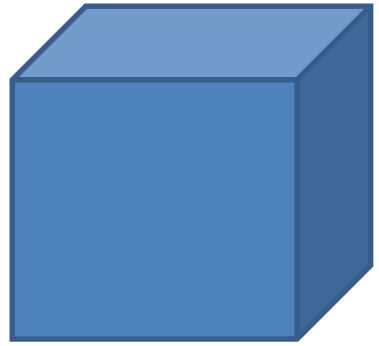
## 3.2.5. Inception Network

### Computational cost



28 x 28 x 192

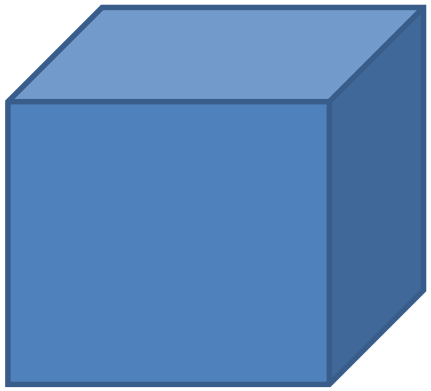
#32 filtros
Conv 5 x 5
Same



28 x 28 x 32

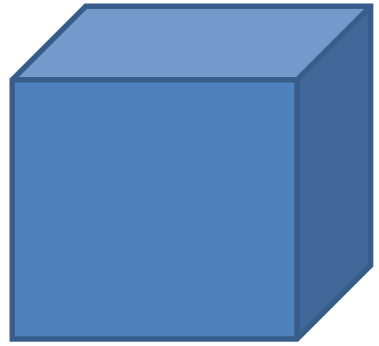
## 3.2.5. Inception Network

### Computational cost



28 x 28 x 192

#32 filtros
Conv 5 x 5
Same

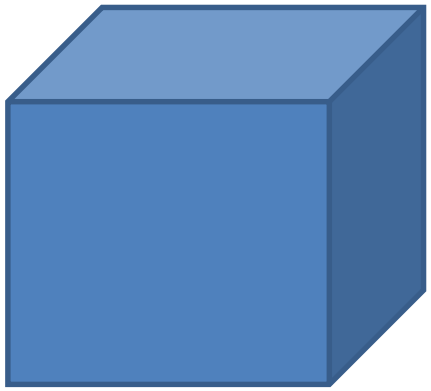


28 x 28 x 32

#Operaciones por filtro = 5 x 5 x 192

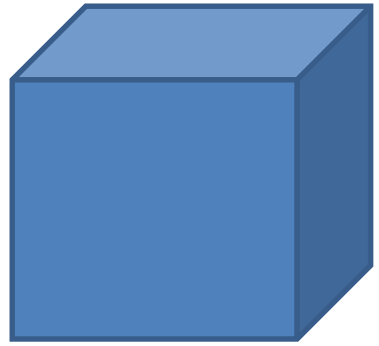
## 3.2.5. Inception Network

### Computational cost



28 x 28 x 192

#32 filtros
Conv 5 x 5
Same



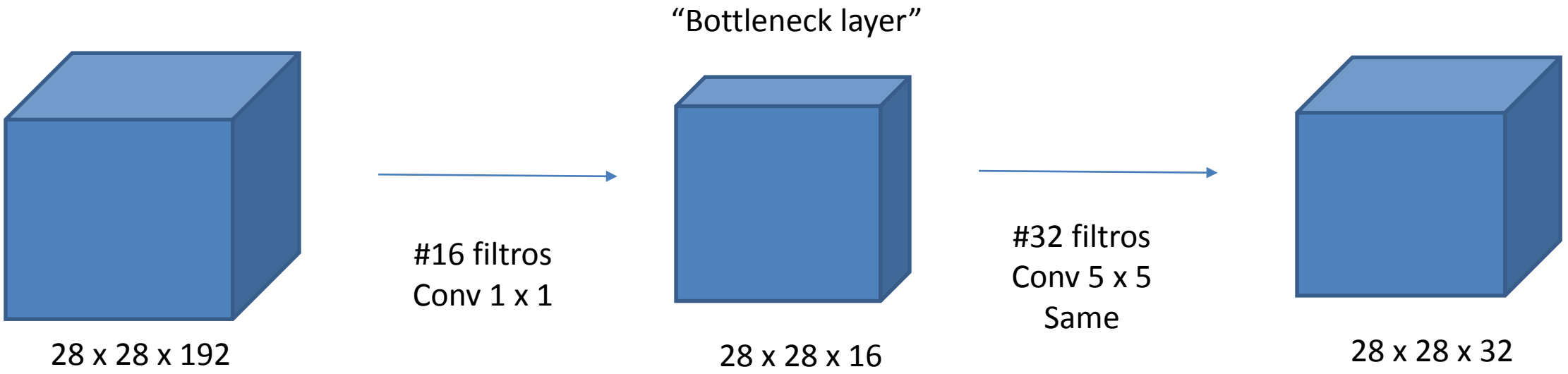
28 x 28 x 32

#Operaciones por filtro = 5 x 5 x 192

#Operaciones en total = (28 x 28 x 32) x (5 x 5 x 192) ~ 120 M

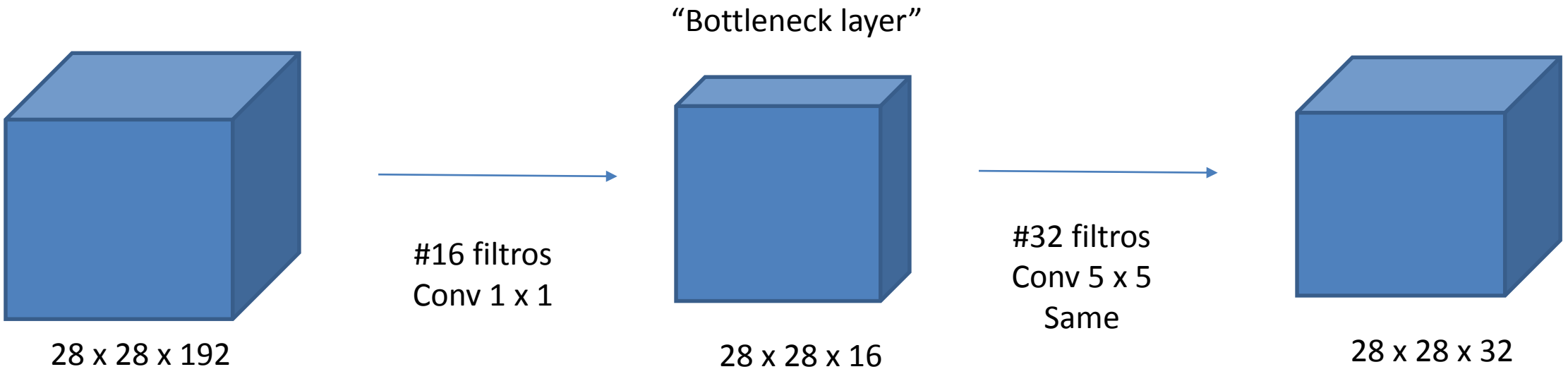
## 3.2.5. Inception Network

### Computational cost



## 3.2.5. Inception Network

### Computational cost

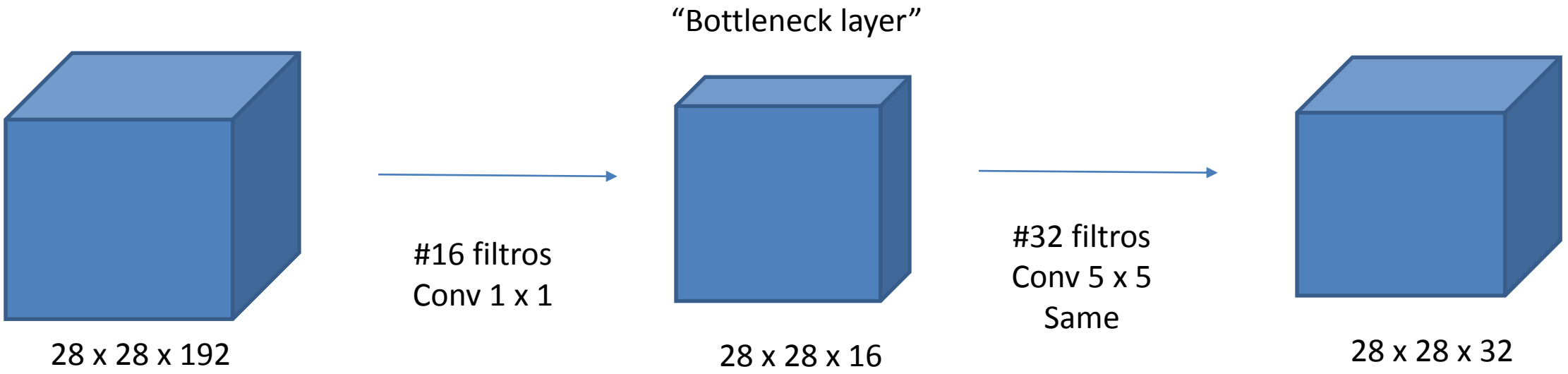



#Operaciones por filtro = 1 x 1 x 192

#Operaciones en total = (28 x 28 x 16) x (1 x 1 x 192) ~ 2.4 M

## 3.2.5. Inception Network

### Computational cost




#Operaciones por filtro = 1 x 1 x 192

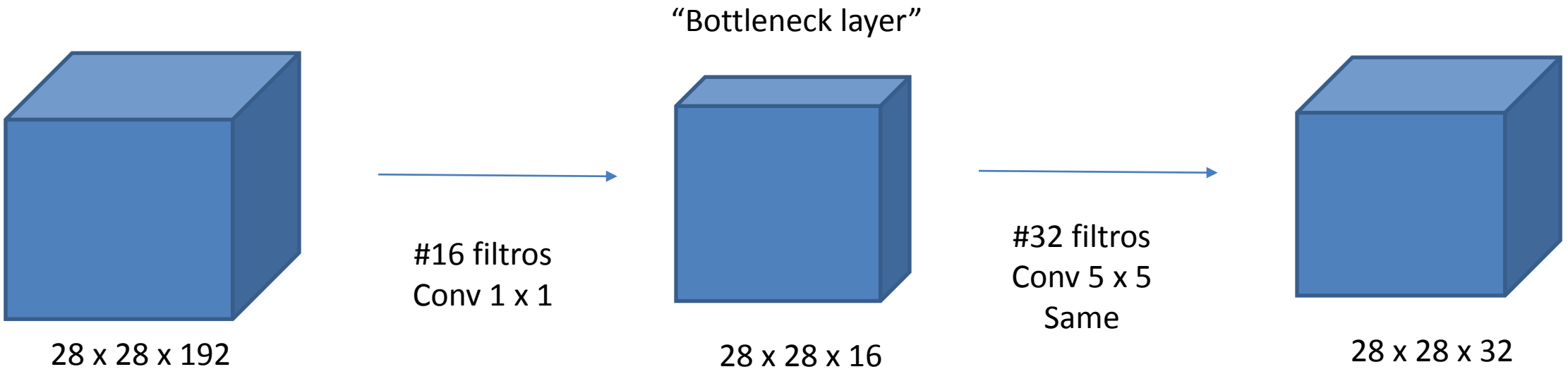
#Operaciones en total = (28 x 28 x 16) x (1 x 1 x 192) ~ 2.4 M

#Operaciones por filtro = 5 x 5 x 16

#Operaciones en total = (28 x 28 x 32) x (5 x 5 x 16) ~ 10 M

## 3.2.5. Inception Network

### Computational cost




#Operaciones por filtro = 1 x 1 x 192

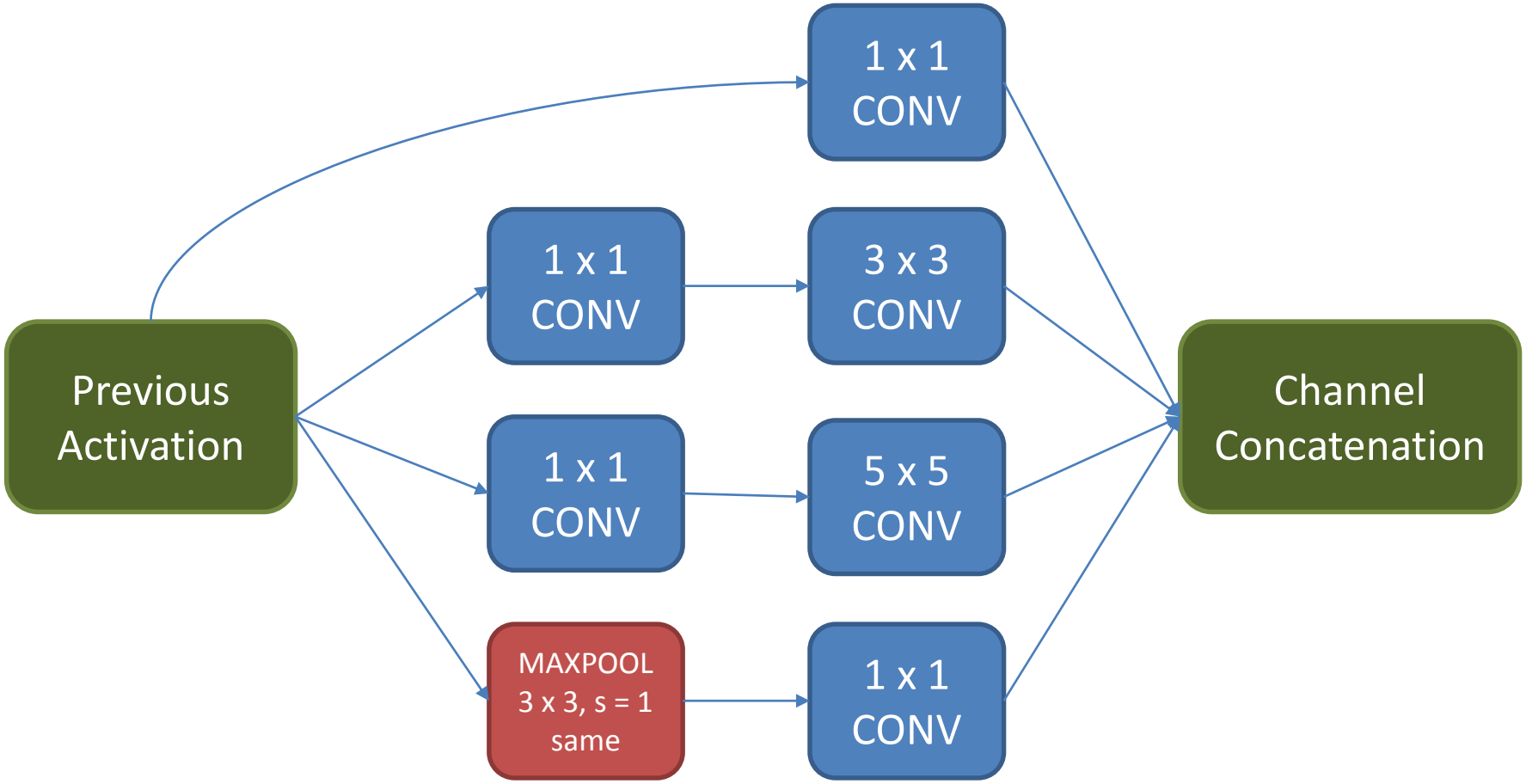
#Operaciones en total = (28 x 28 x 16) x (1 x 1 x 192) ~ 2.4 M

#Operaciones por filtro = 5 x 5 x 16

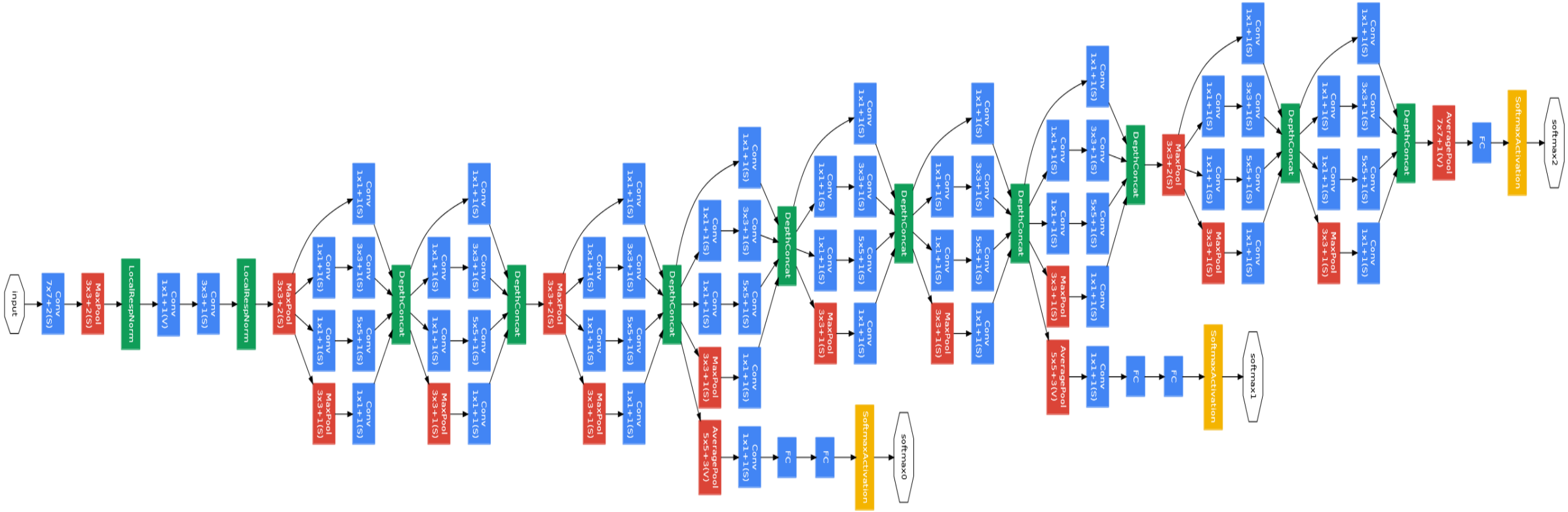
#Operaciones en total = (28 x 28 x 32) x (5 x 5 x 16) ~ 10 M

~ 12.4 M << 120 M

## 3.2.5. Inception Network



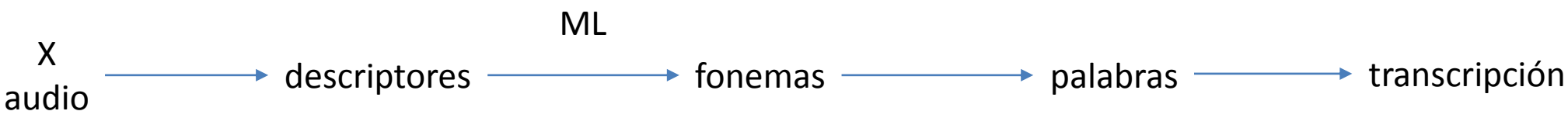
## 3.2.5. Inception Network



## 3.2.5. Inception Network

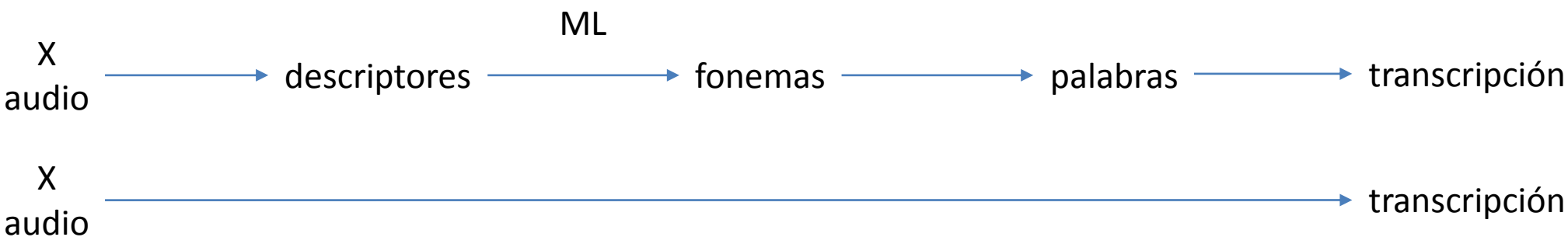
## (End-to-end Learning)

Reconocimiento de voz



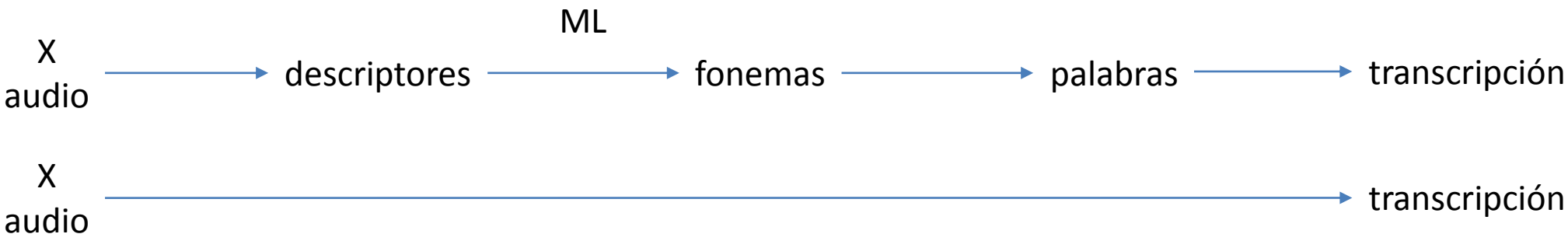
## (End-to-end Learning)

Reconocimiento de voz

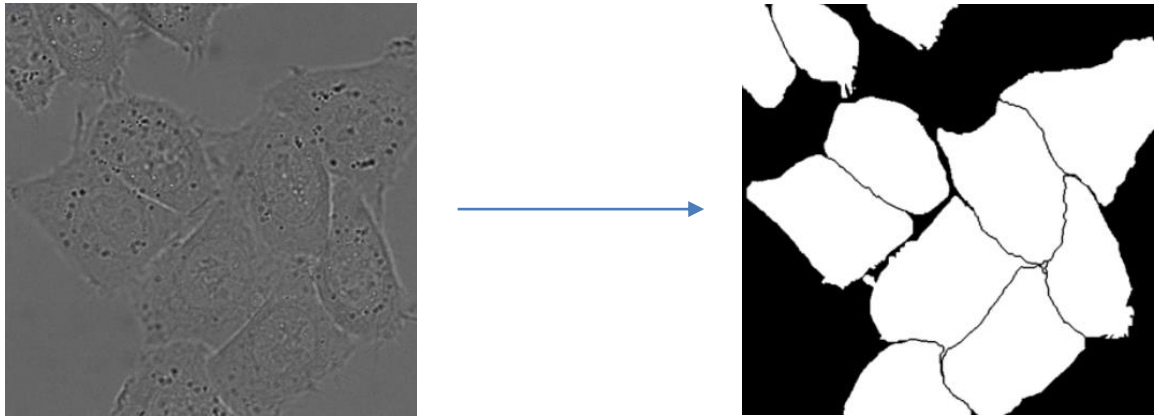


## (End-to-end Learning)

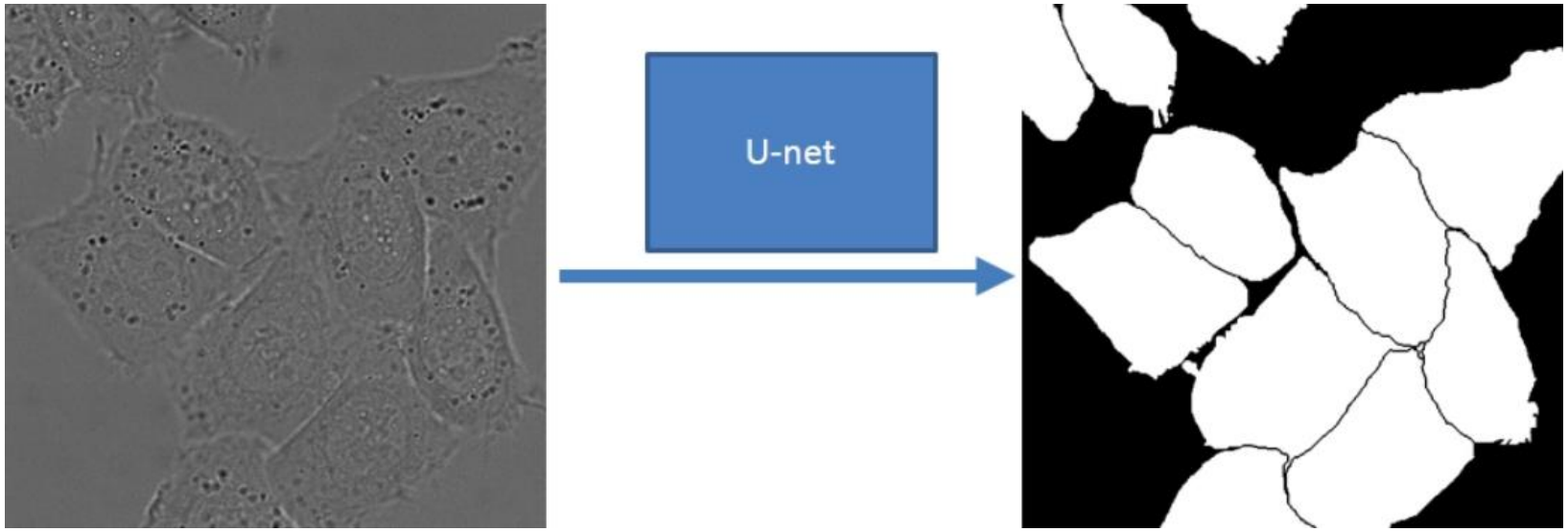
Reconocimiento de voz




Segmentación de imágenes médicas



## 3.2.6. U-NET

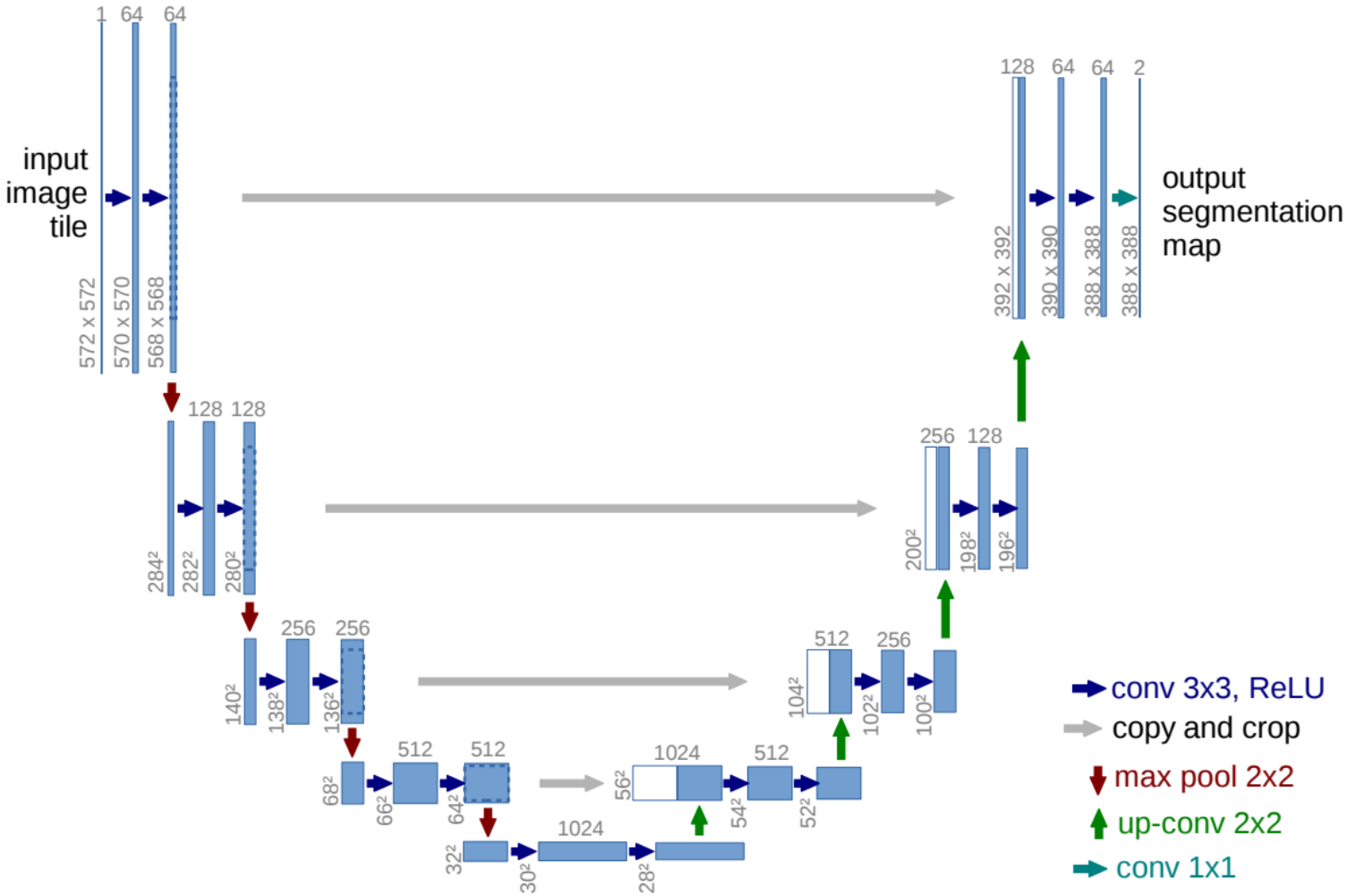



Input

Segmentation

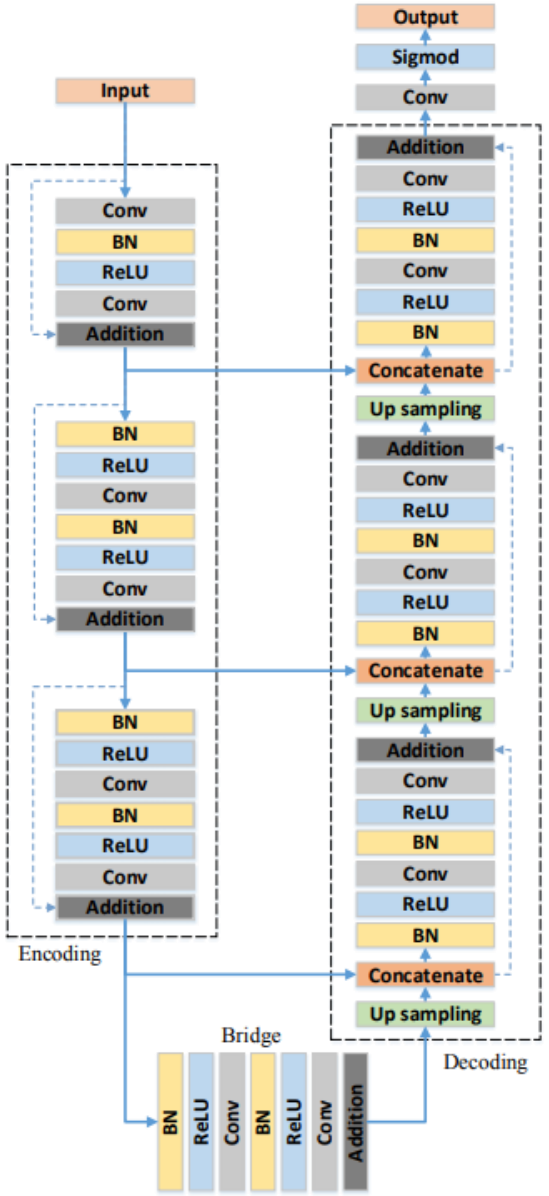
**U-Net: Convolutional Networks for Biomedical Image Segmentation**
https://arxiv.org/abs/1505.04597

## 3.2.6. U-NET


input image tile
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

## 3.2.6. U-NET

### Deep Residual U-NET



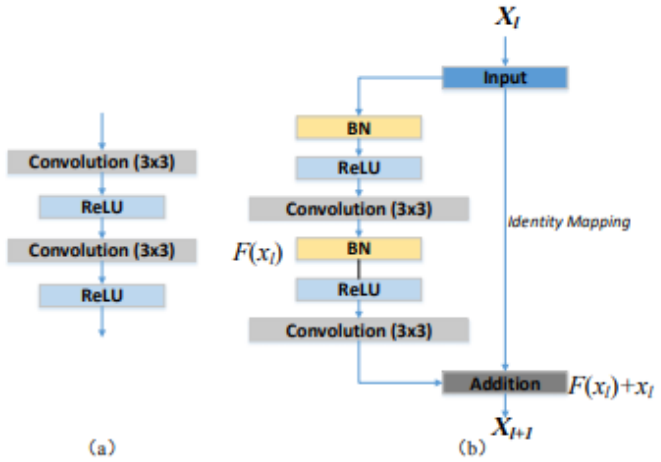




Fig. 1. Building blocks of neural networks. (a) Plain neural unit used in U-Net and (b) residual unit with identity mapping used in the proposed ResUnet.

**Road Extraction by Deep Residual U-Net**
https://arxiv.org/abs/1711.10684

## 3.2.6. U-NET

### Deep Residual U-NET


Original


Ground truth


U-NET

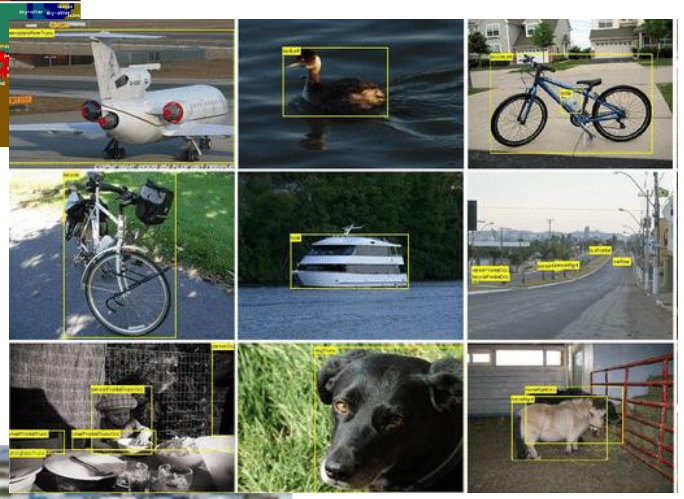

ResU-NET

## 3.2.7. Transfer Learning

### Motivación

- Disponibilidad de grandes bases de datos. Ejm: CIFAR-10, Pascal, Imagenet, MS COCO.
- Entrenar una red en estos modelos puede tomar meses de diseño entrenamiento, además de herramientas computacionales (GPUs).
- Es posible reutilizar el “conocimiento” aprendido por dichos modelos (open source).

MS-COCO



Pascal

Imagenet

## 3.2.7. Transfer Learning

### Ejemplo: Mask-RCNN


class_names = ['BG', 'person', 'bicycle', 'car', 'motorcycle', 'airplane','bus', 'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball','kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone', microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush']

**Mask R-CNN for Object Detection and Segmentation** https://github.com/matterport/Mask_RCNN

## 3.2.7. Transfer Learning

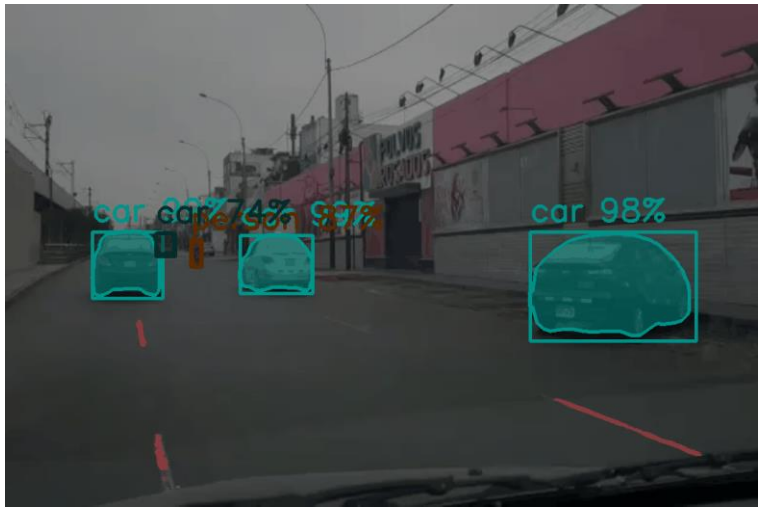
### Ejemplo: Mask-RCNN


class_names = ['BG', 'person', 'bicycle', 'car', 'motorcycle', 'airplane','bus', 'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball','kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone', microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush']

**Mask R-CNN for Object Detection and Segmentation** https://github.com/matterport/Mask_RCNN

## 3.2.7. Transfer Learning

### Ejemplo: Mask-RCNN

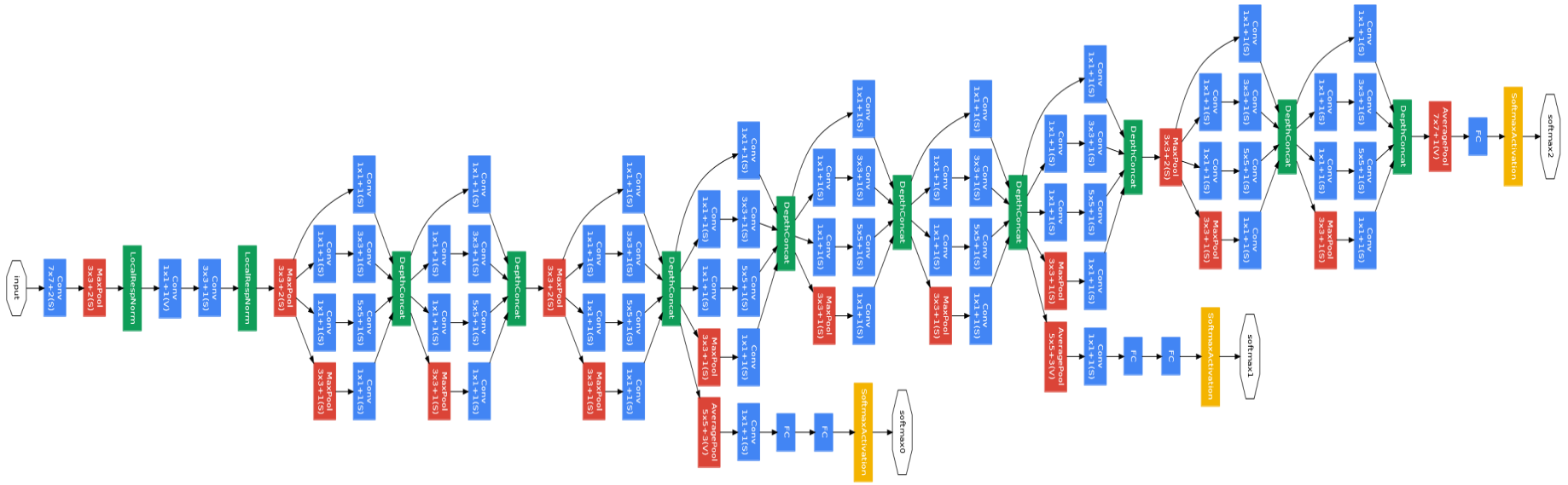




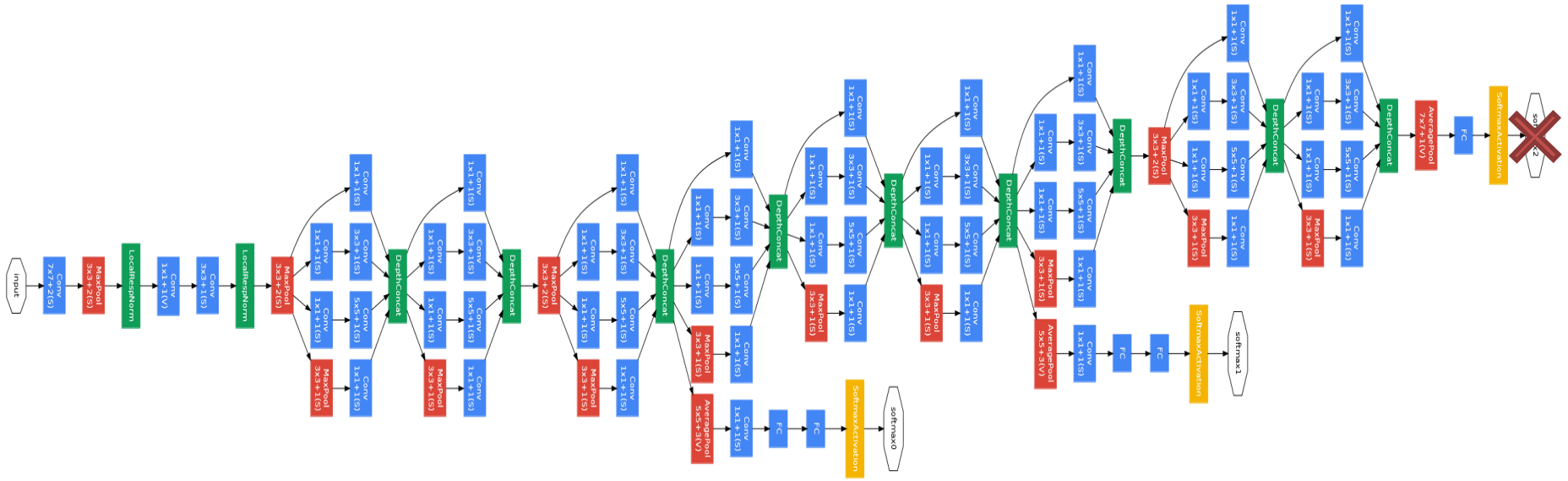

**Mask R-CNN for Object Detection and Segmentation**
https://github.com/matterport/Mask_RCNN

**LaneSegmentationNetwork**
https://github.com/POSTECH-IMLAB/LaneSegmentationNetwork

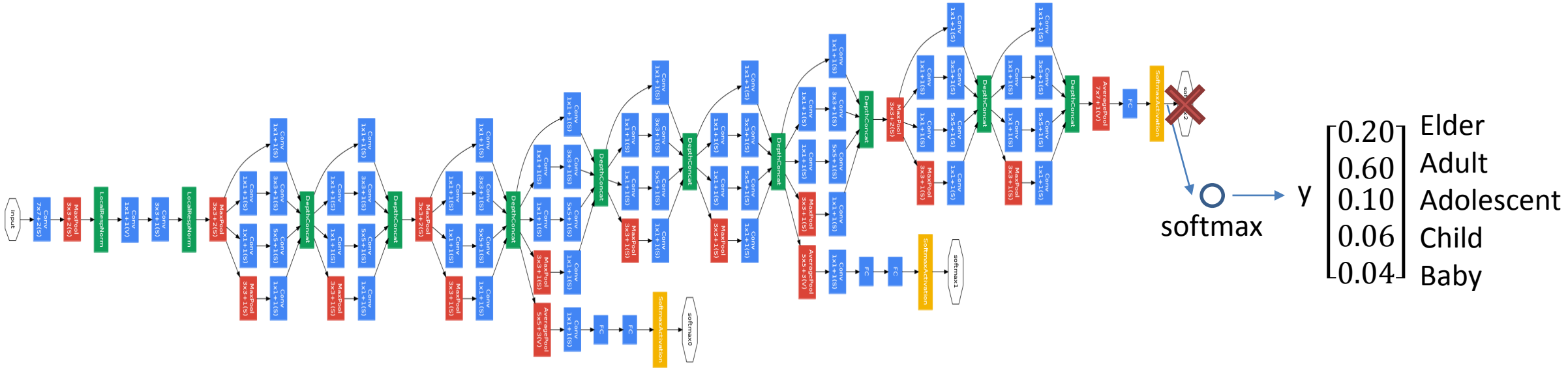
## 3.2.7. Transfer Learning



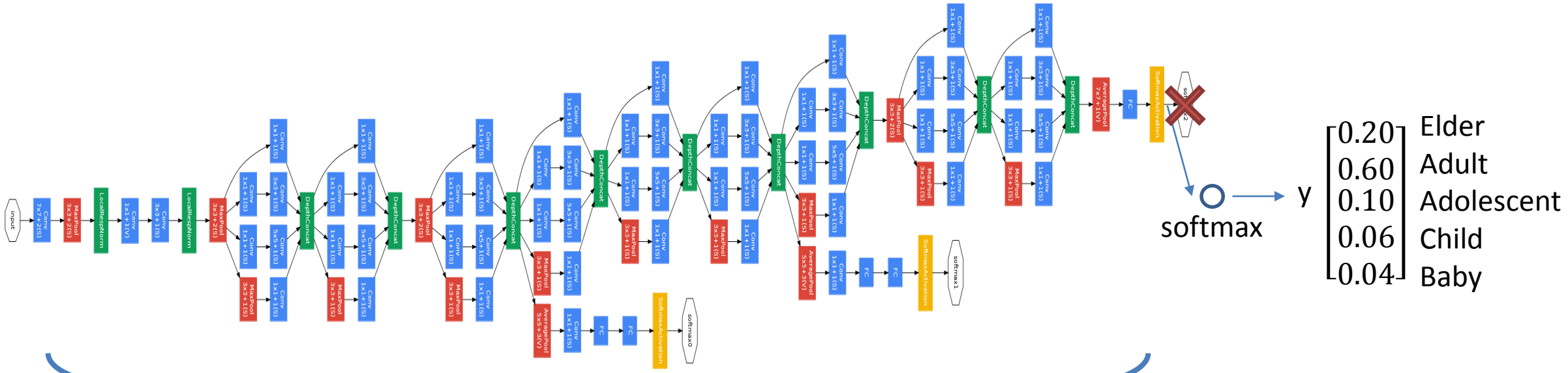
## 3.2.7. Transfer Learning



## 3.2.7. Transfer Learning

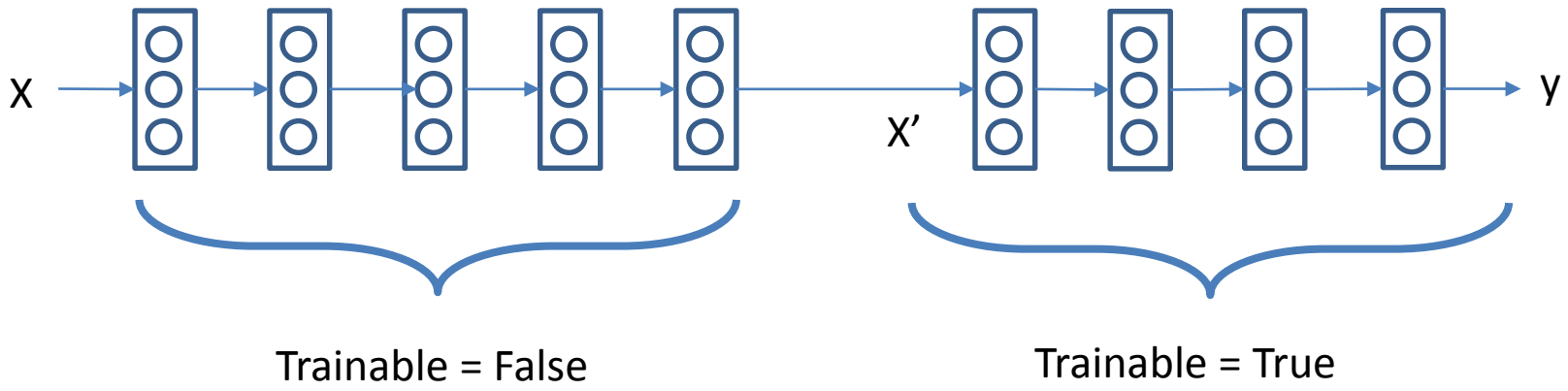


## 3.2.7. Transfer Learning

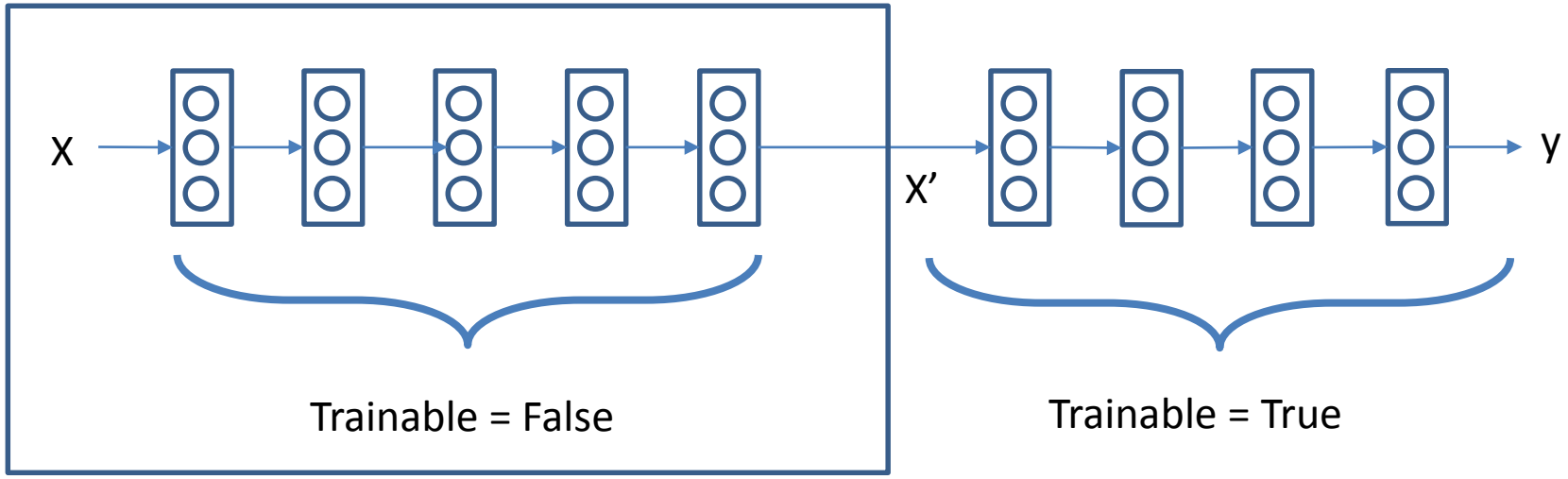



"Freeze"
Trainable = False

## 3.2.7. Transfer Learning

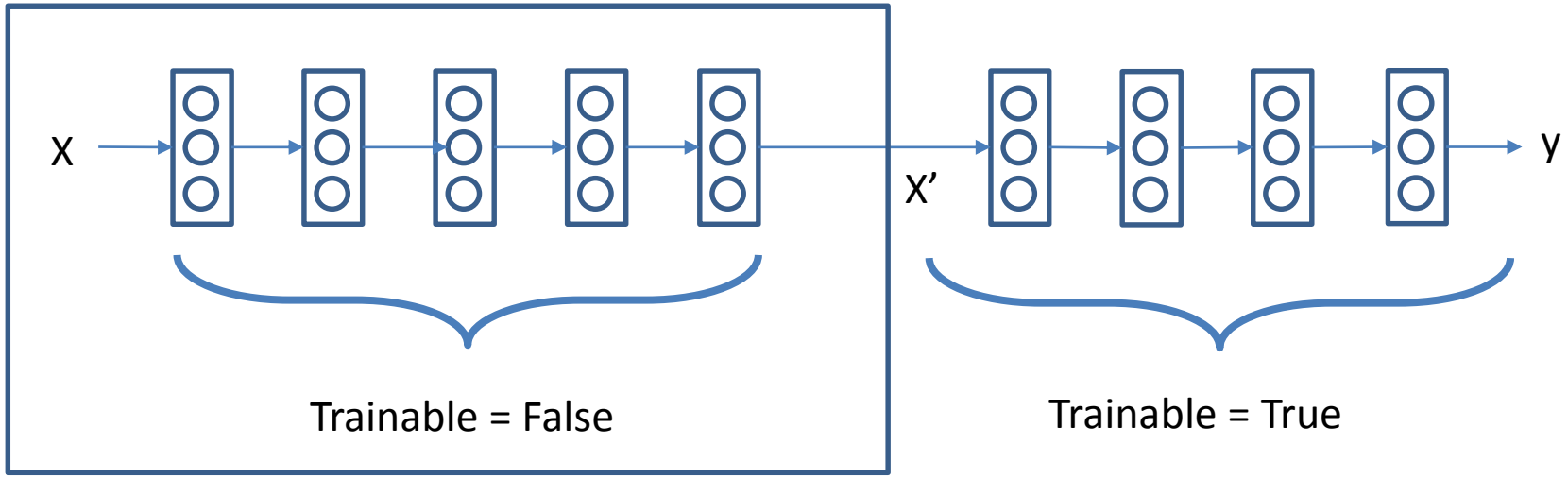


## 3.2.7. Transfer Learning




Guardar en disco

## 3.2.7. Transfer Learning




Guardar en disco