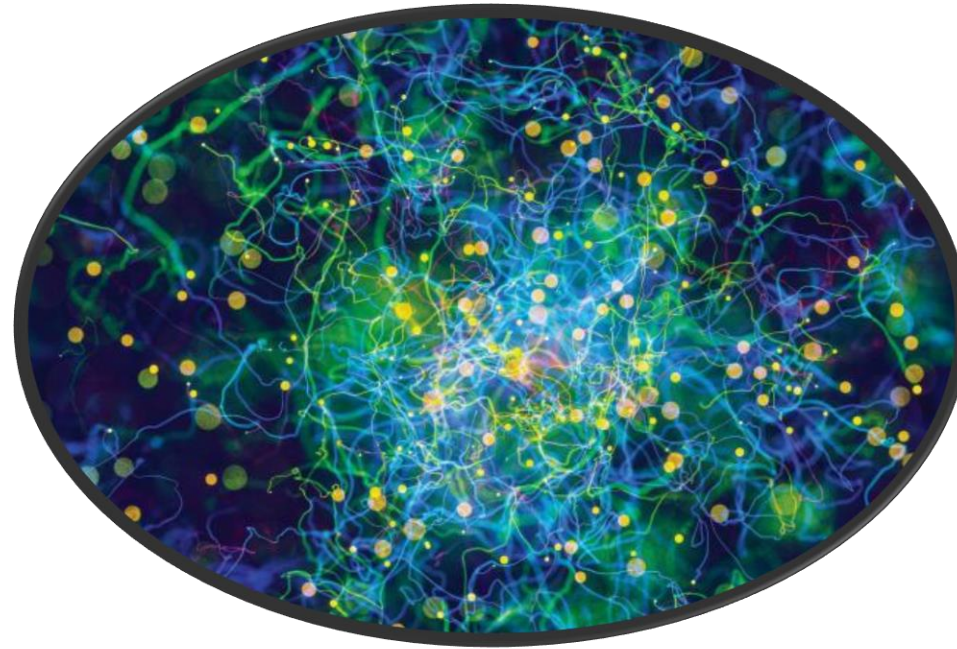


# Deep Learning

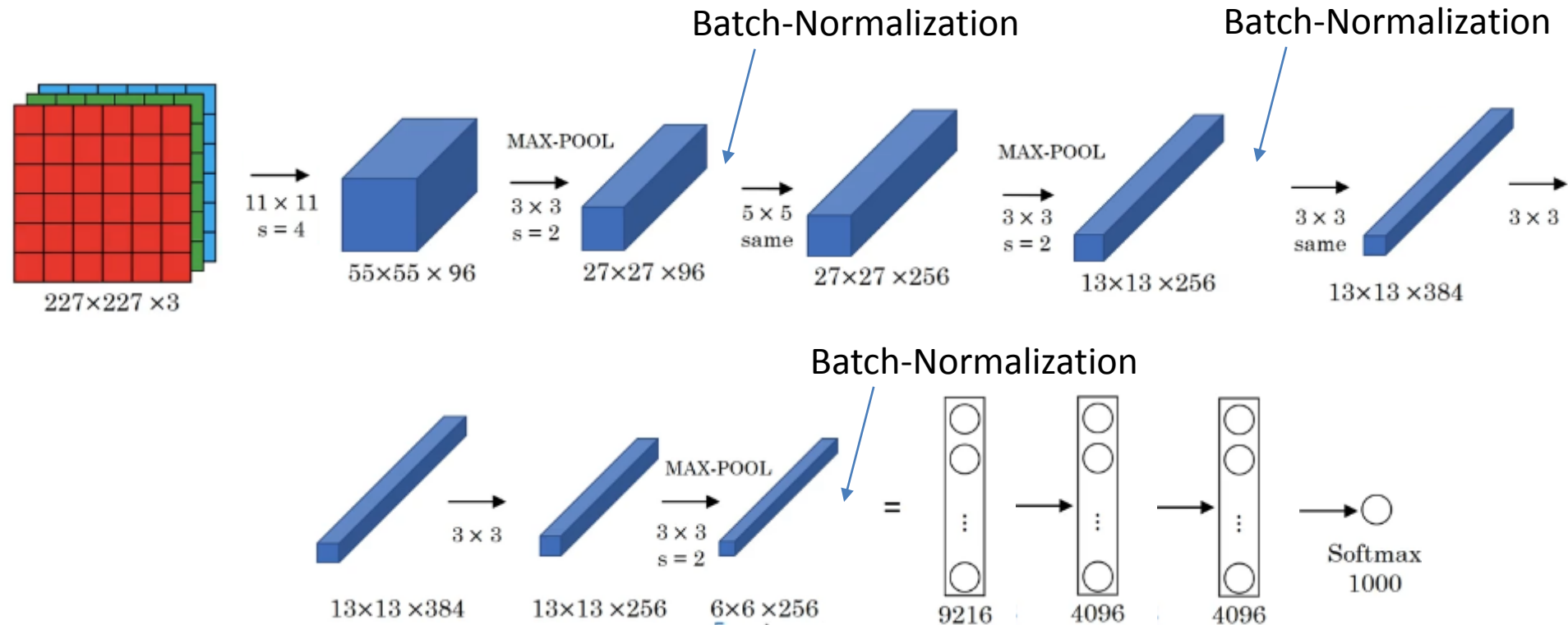


***Día 3***

**EXPOSITOR: Ing. Giorgio Morales Luna**

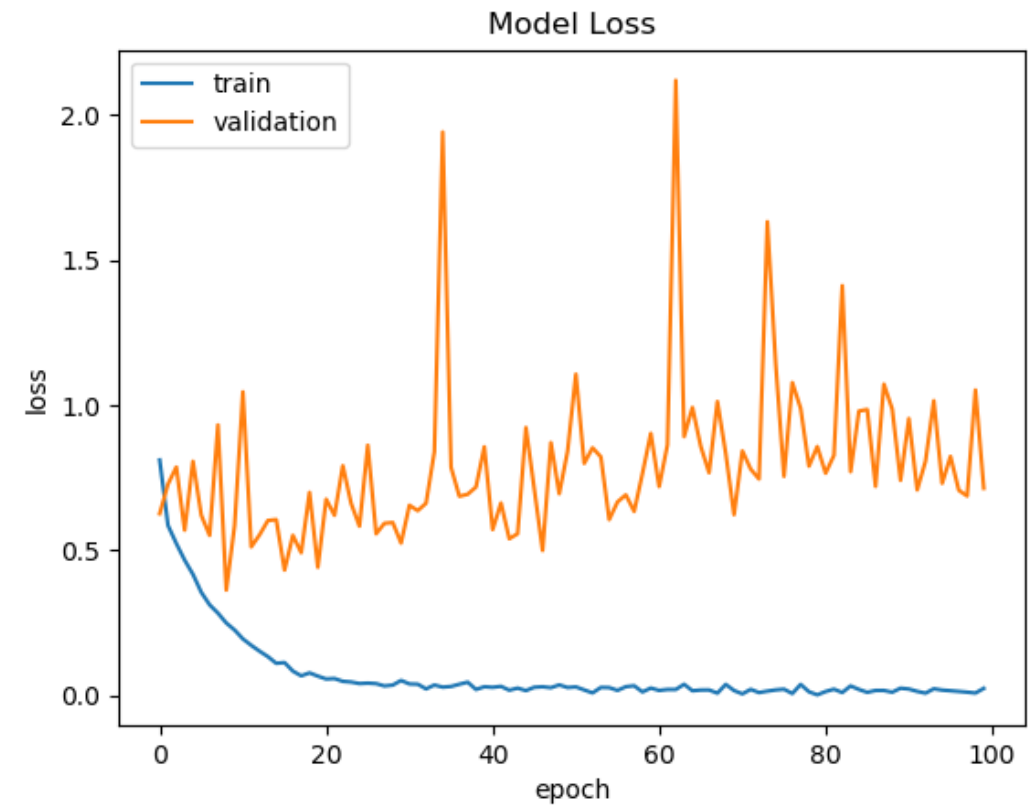
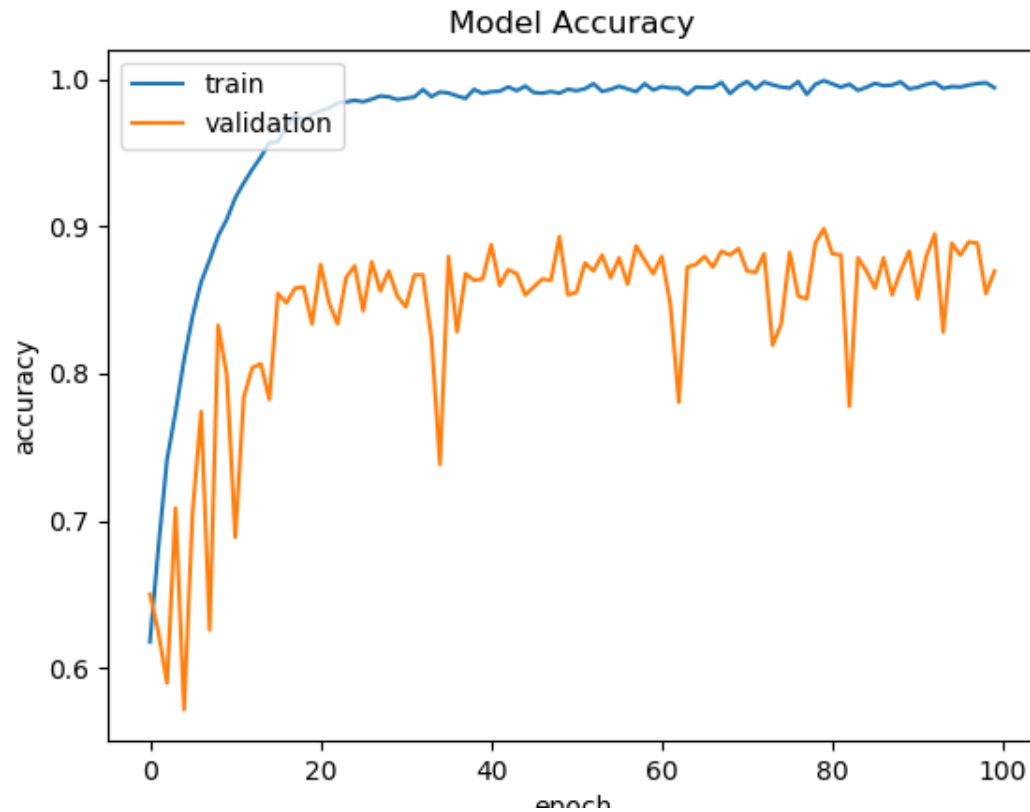
**Junio 2018**

### 3.2.8. AlexNet from scratch



### 3.2.8. AlexNet from scratch

#### Resultado



### 3.2.9. VGG16 with Keras

#### **keras.applications.vgg16.VGG16**

Carga el modelo VGG16 para Keras:

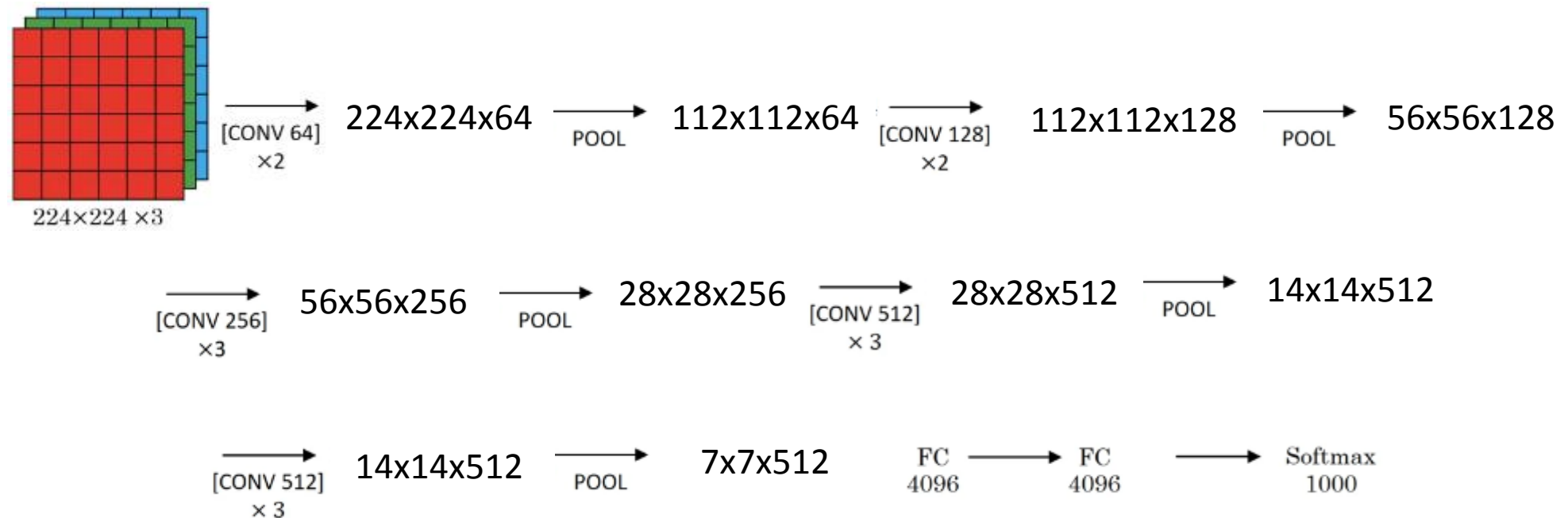
- **Include\_top**: Si es True, incluye las 3 capas fully-connected al final de la red.
- **Weights**: Si es 'None', inicializa los pesos con valores aleatorios. Si es 'imagenet', carga los pesos de una red VGG16 entrenada en Imagenet.
- **input\_shape**: (224,224,3) por defecto

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/>

### 3.2.9. VGG16 with Keras



### 3.2.9. VGG16 with Keras

#### Entrenamiento VGG16 sin Transfer Learning

```
block4_conv2 (Conv2D)          (None, 28, 28, 512)      2359808
block4_conv3 (Conv2D)          (None, 28, 28, 512)      2359808
block4_pool (MaxPooling2D)     (None, 14, 14, 512)      0
block5_conv1 (Conv2D)          (None, 14, 14, 512)      2359808
block5_conv2 (Conv2D)          (None, 14, 14, 512)      2359808
block5_conv3 (Conv2D)          (None, 14, 14, 512)      2359808
block5_pool (MaxPooling2D)     (None, 7, 7, 512)        0
flatten_1 (Flatten)            (None, 1568)              0
fc1 (Dense)                    (None, 1000)              1001000
fc2 (Dense)                    (None, 10)                 110
Prediction (Dense)             (None, 1)                  0
Total params: 40,668,481
Trainable params: 40,668,481
Non-trainable params: 0
Comienza entrenamiento.....
```

```
gmorales@super01:~/Curso_DeepLearning/DÃ-a3
mean that there could be performance gains if more memory were available.
2019-05-21 12:16:15.320923: W tensorflow/core/common_runtime/bfc_allocator.cc:211] Allocator (GPU_0_bfc) ran out of memory trying to allocate 1048576 bytes. Please consider reducing the size of your tensors to fit into the available memory.
8888/8888 [=====] - 138s 16ms/step - loss: 0.6933 - acc: 0.4994 - val_loss: 0.6930 - val_acc: 0.5153e
Epoch 2/50
8888/8888 [=====] - 133s 15ms/step - loss: 0.6931 - acc: 0.5041 - val_loss: 0.6930 - val_acc: 0.5153
Epoch 3/50
1472/8888 [==>.....] - ETA: 1:41 - loss: 0.6931 - acc: 0.5088
```



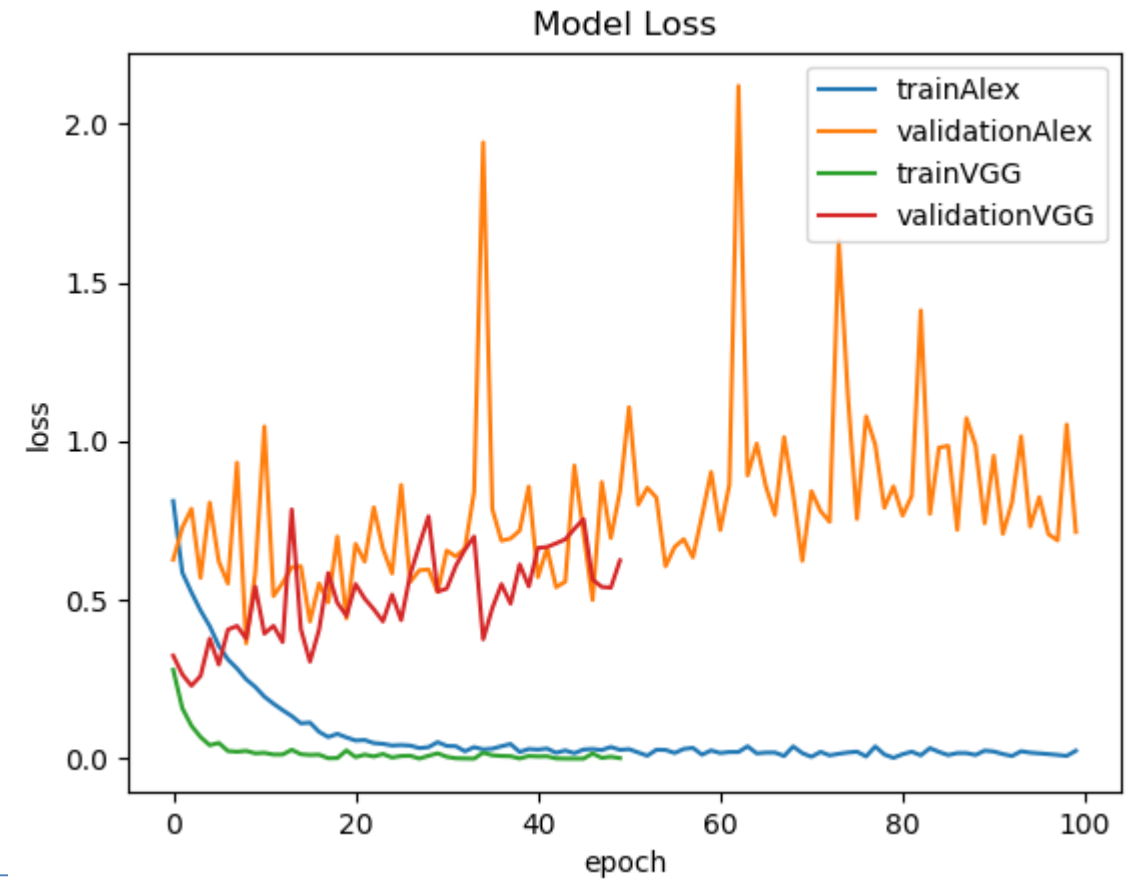
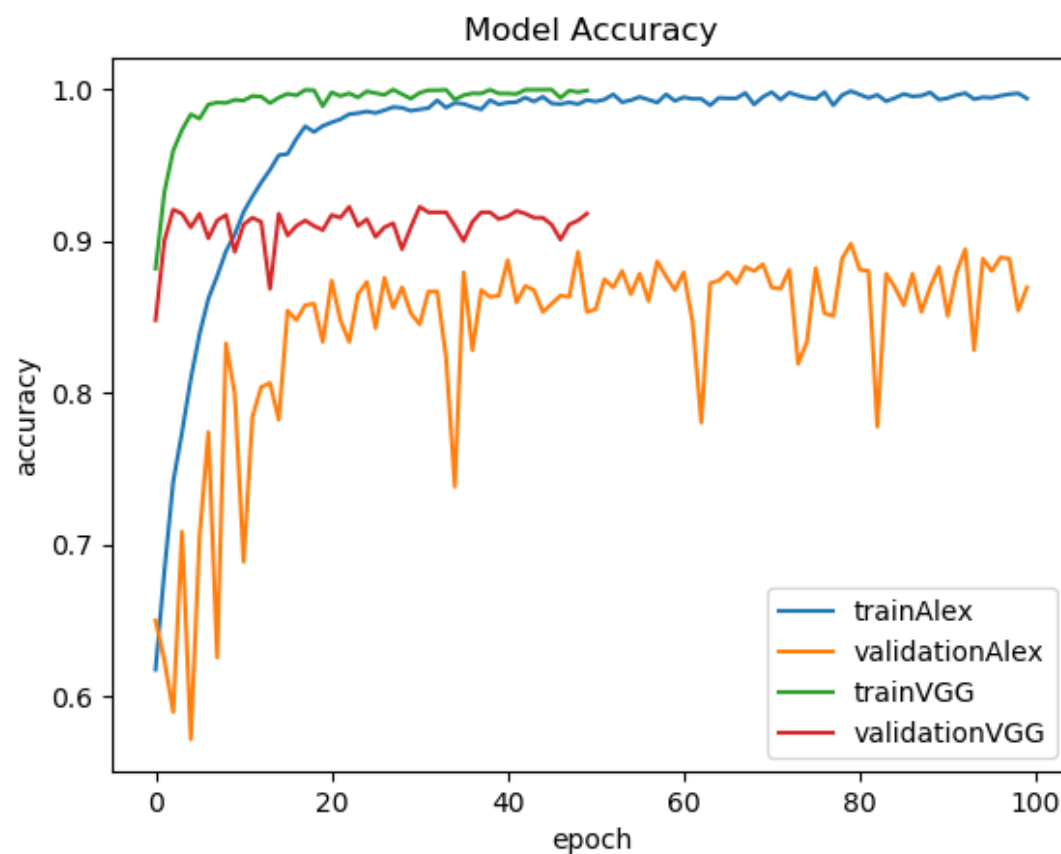
### 3.2.9. VGG16 with Keras

#### Entrenamiento VGG16 con Transfer Learning

```
gmorales@nodo01:~/Dia3
flatten_1 (Flatten)          (None, 25088)          0
fc1 (Dense)                  (None, 1024)           25691136
fc2 (Dense)                  (None, 256)            262400
Prediction (Dense)          (None, 1)              257
=====
Total params: 40,668,481
Trainable params: 25,953,793
Non-trainable params: 14,714,688
=====
Comienza entrenamiento.....
WARNING:tensorflow:From /home/gmorales/giorgio/lib64/python3.6/site-packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 8888 samples, validate on 1112 samples
Epoch 1/50
2019-05-21 11:38:06.972436: I tensorflow/stream_executor/dso_loader.cc:152] successfully opened CUDA library libcublas.so.10.0 locally
8888/8888 [=====] - 115s 13ms/step - loss: 0.2802 - acc: 0.8820 - val_loss: 0.3248 - val_acc: 0.8480
Epoch 2/50
8888/8888 [=====] - 110s 12ms/step - loss: 0.1590 - acc: 0.9323 - val_loss: 0.2656 - val_acc: 0.9002
Epoch 3/50
8888/8888 [=====] - 110s 12ms/step - loss: 0.1038 - acc: 0.9596 - val_loss: 0.2299 - val_acc: 0.9209
Epoch 4/50
5584/8888 [=====>.....] - ETA: 36s - loss: 0.0709 - acc: 0.9726
```

### 3.2.9. VGG16 with Keras

#### Resultado





### 3.2.9. VGG16 with Keras

Resultado





### 3.2.10. Data Augmentation

### 3.2.10. Data Augmentation

#### Mirroring



### 3.2.10. Data Augmentation

#### Mirroring



### 3.2.10. Data Augmentation

#### Mirroring



#### Random Cropping



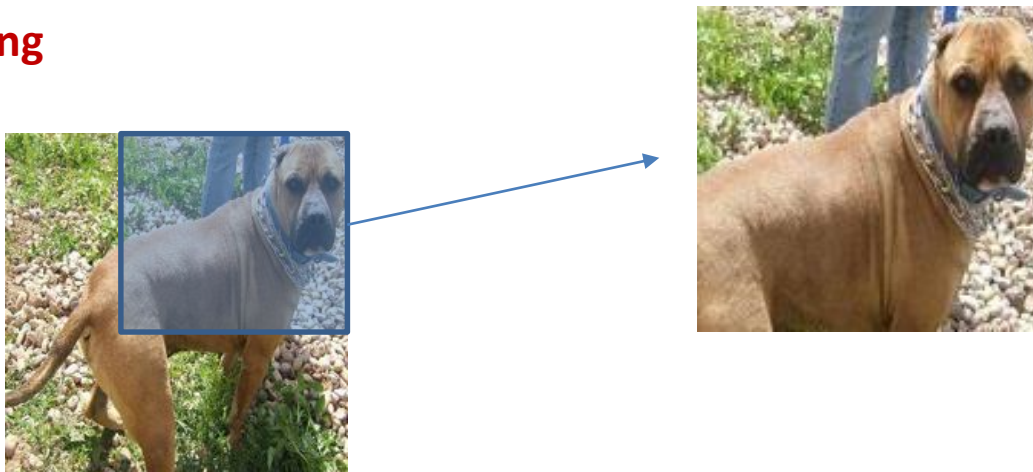


### 3.2.10. Data Augmentation

#### Mirroring



#### Random Cropping



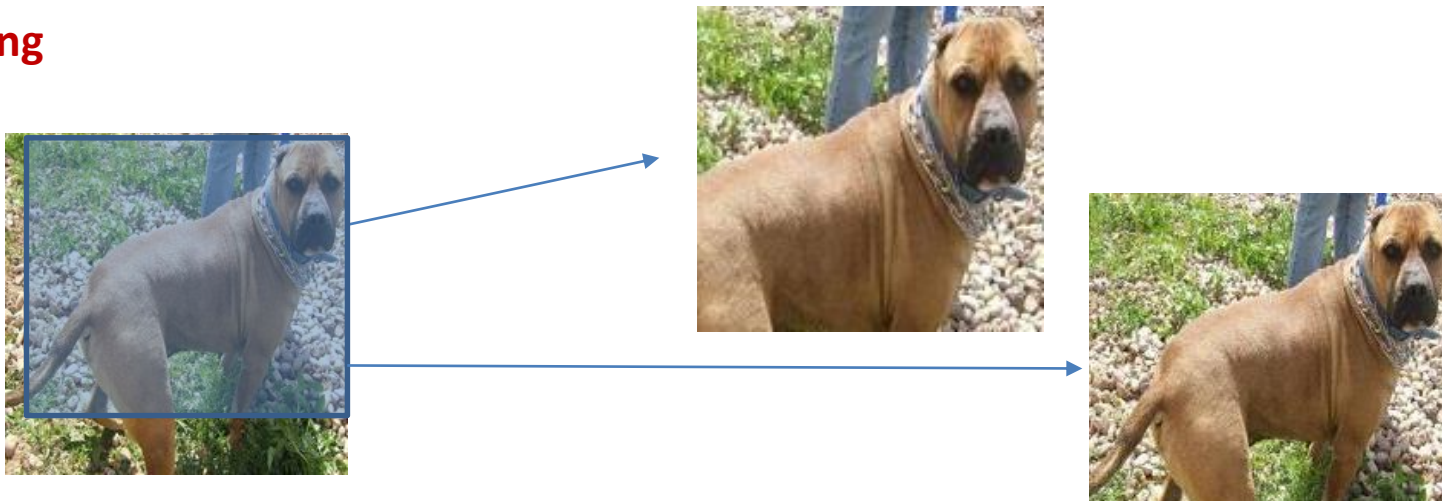


### 3.2.10. Data Augmentation

#### Mirroring



#### Random Cropping

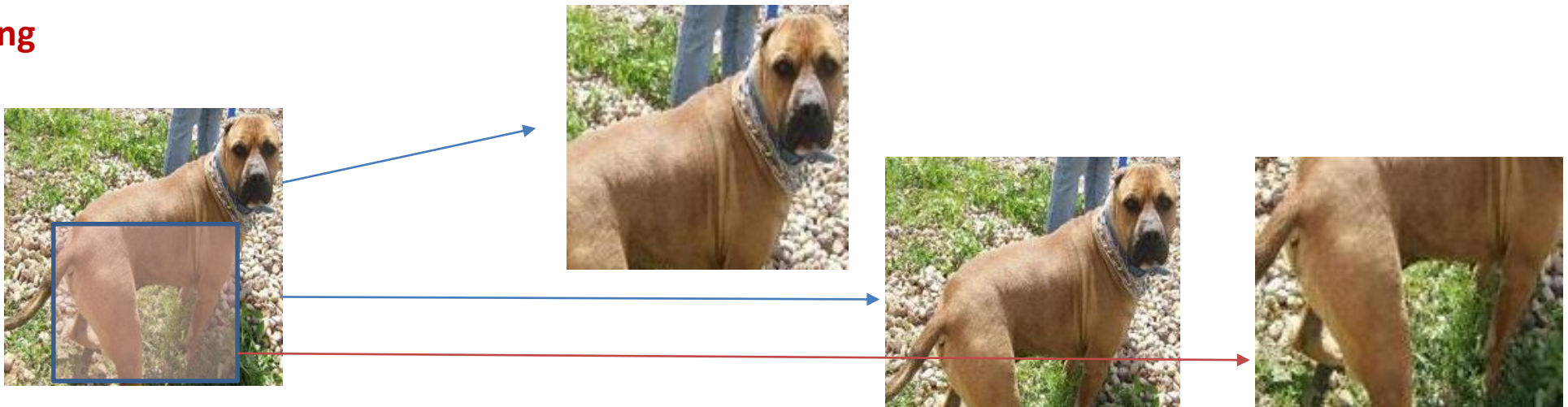


### 3.2.10. Data Augmentation

#### Mirroring



#### Random Cropping





### 3.2.10. Data Augmentation

#### Rotación

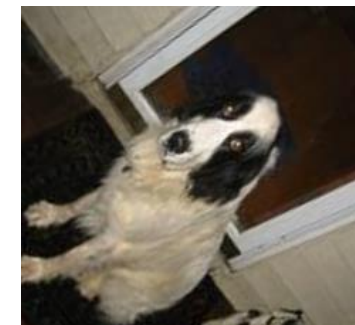


#### Rotación + Mirroring



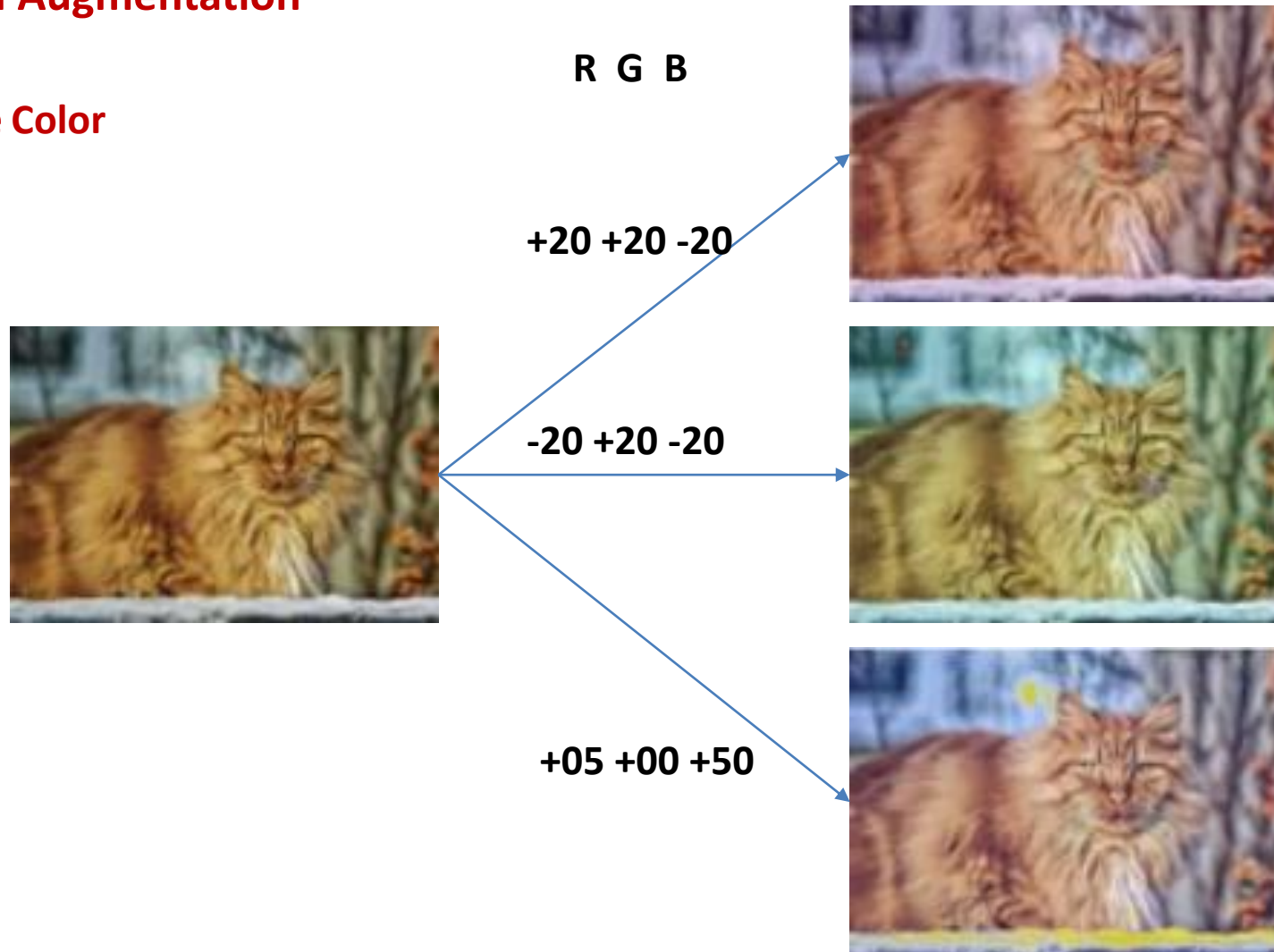
### 3.2.10. Data Augmentation

Rotación + Cropping



### 3.2.10. Data Augmentation

#### Distorsión de Color





## 3.2.10. Data Augmentation

### Casos Reales





### 3.2.10. Data Augmentation

#### Casos Reales



### 3.2.10. Data Augmentation

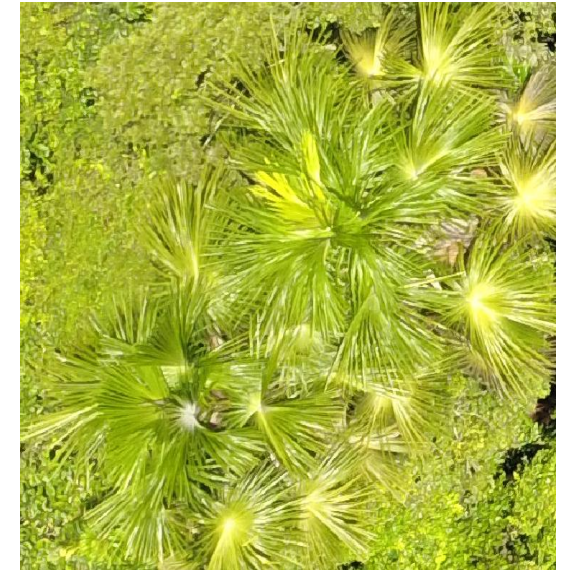
#### Casos Reales





### 3.2.10. Data Augmentation

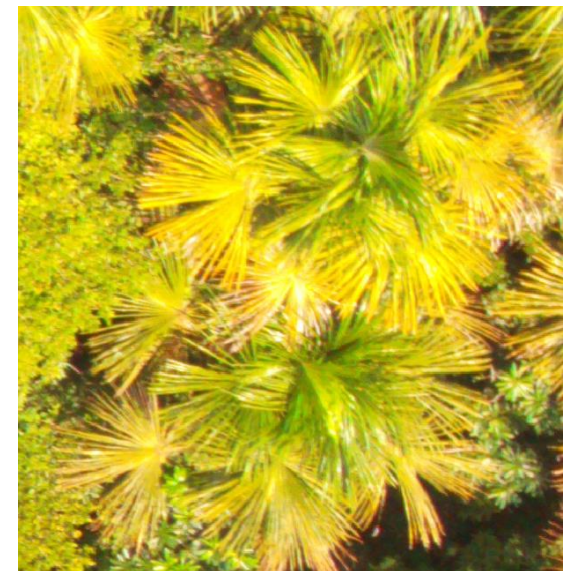
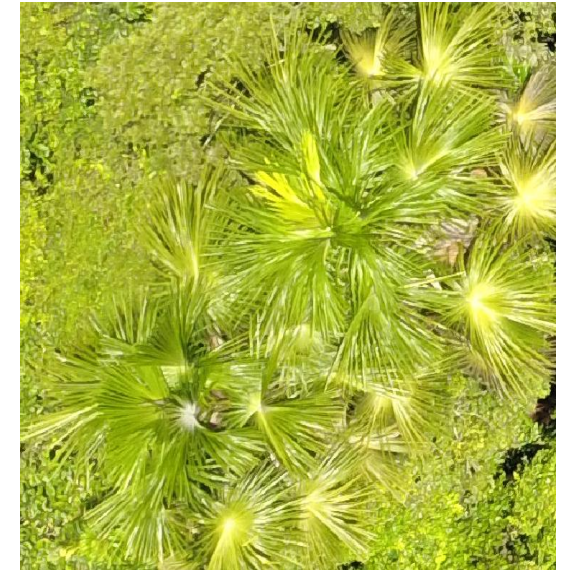
#### Casos Reales





### 3.2.10. Data Augmentation

#### Casos Reales





### 3.2.10. Data Augmentation

¿En qué momento se aplica Data Augmentation?

- En el momento en el que se lee cada batch de entrenamiento.

### 3.2.10. Data Augmentation

¿En qué momento se aplica Data Augmentation?

- En el momento en el que se lee cada batch de entrenamiento.

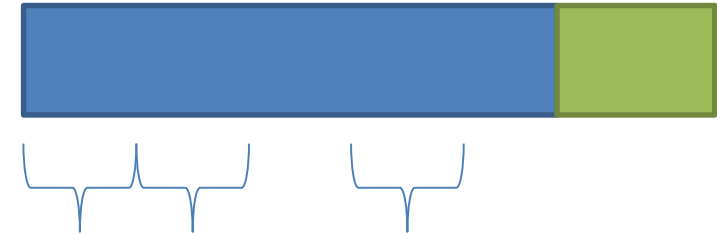




### 3.2.10. Data Augmentation

¿En qué momento se aplica Data Augmentation?

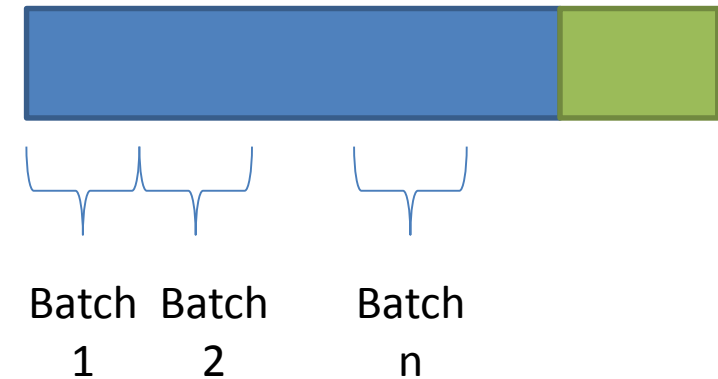
- En el momento en el que se lee cada batch de entrenamiento.



### 3.2.10. Data Augmentation

¿En qué momento se aplica Data Augmentation?

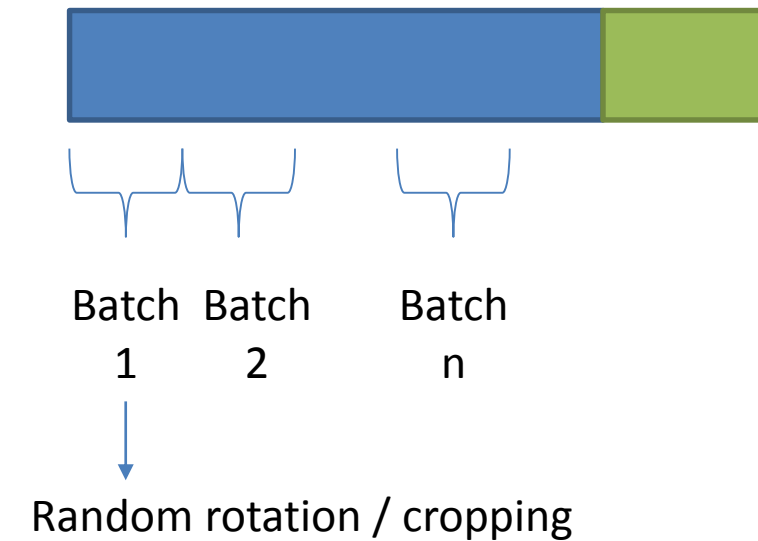
- En el momento en el que se lee cada batch de entrenamiento.



### 3.2.10. Data Augmentation

¿En qué momento se aplica Data Augmentation?

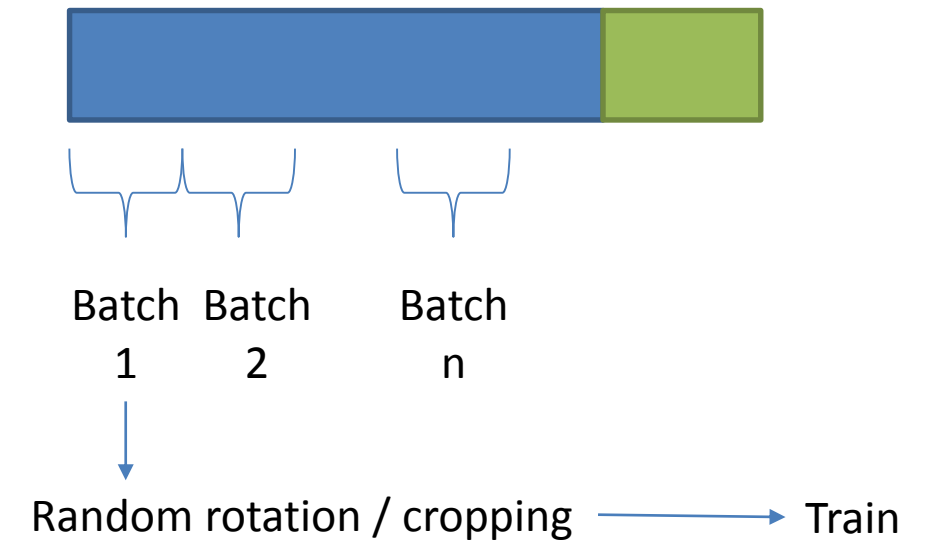
- En el momento en el que se lee cada batch de entrenamiento.



### 3.2.10. Data Augmentation

¿En qué momento se aplica Data Augmentation?

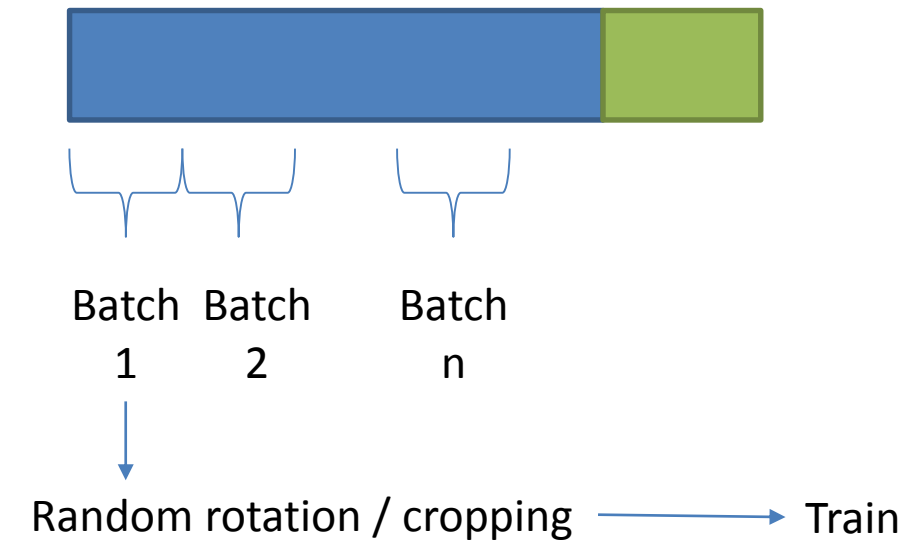
- En el momento en el que se lee cada batch de entrenamiento.



### 3.2.10. Data Augmentation

¿En qué momento se aplica Data Augmentation?

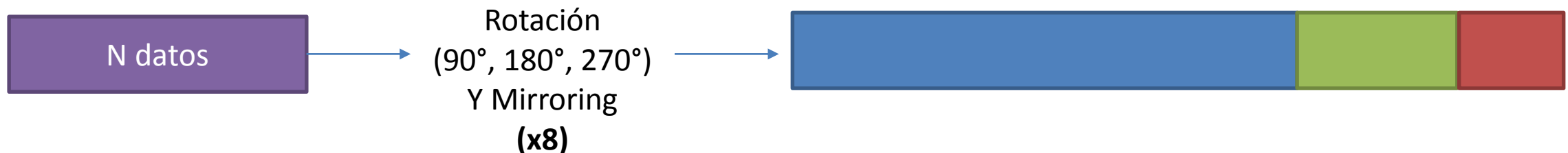
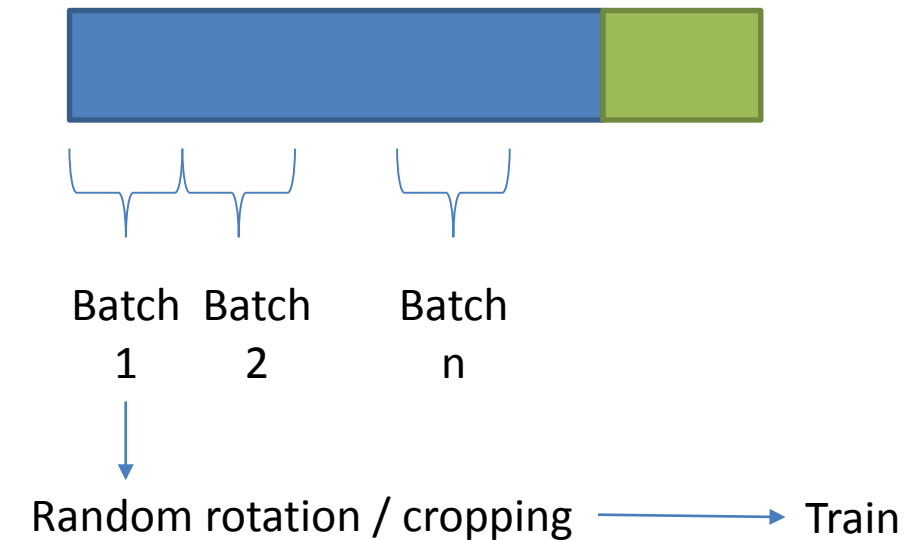
- En el momento en el que se lee cada batch de entrenamiento.
- Antes de crear los sets de entrenamiento, validación y test



### 3.2.10. Data Augmentation

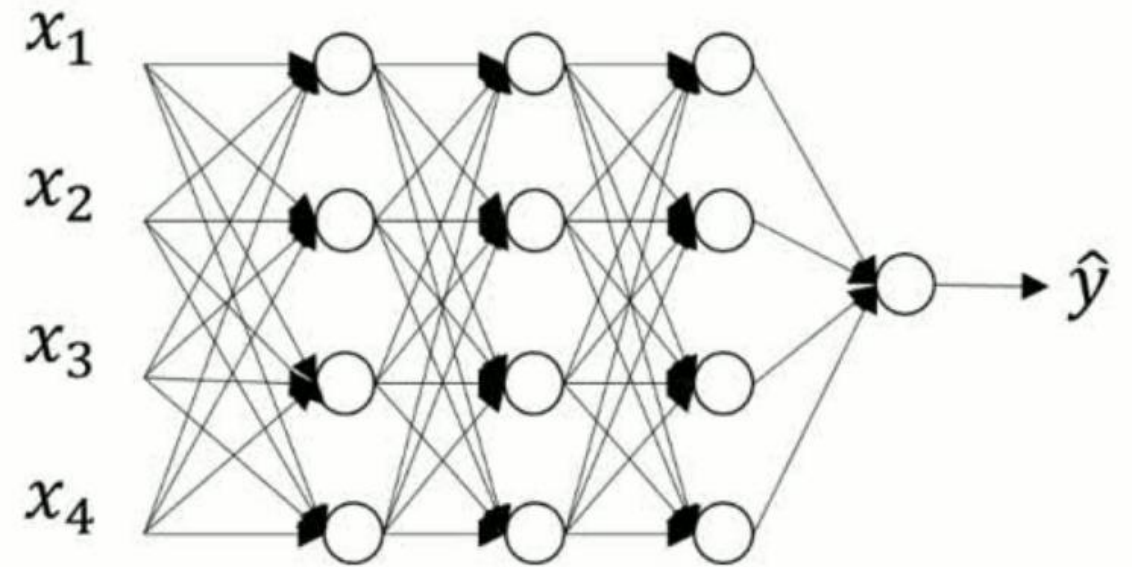
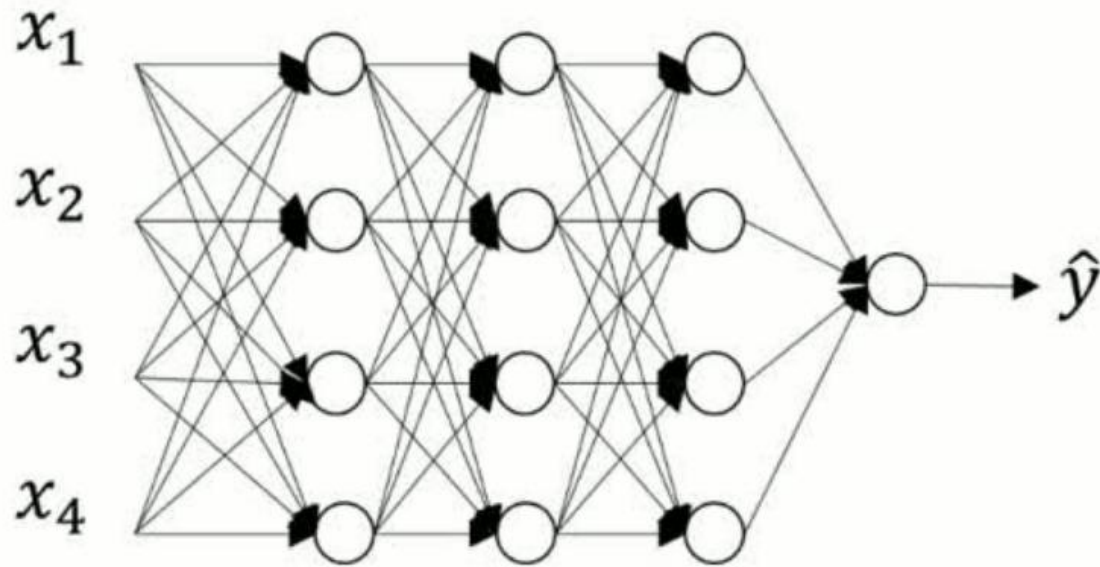
¿En qué momento se aplica Data Augmentation?

- En el momento en el que se lee cada batch de entrenamiento.
- Antes de crear los sets de entrenamiento, validación y test

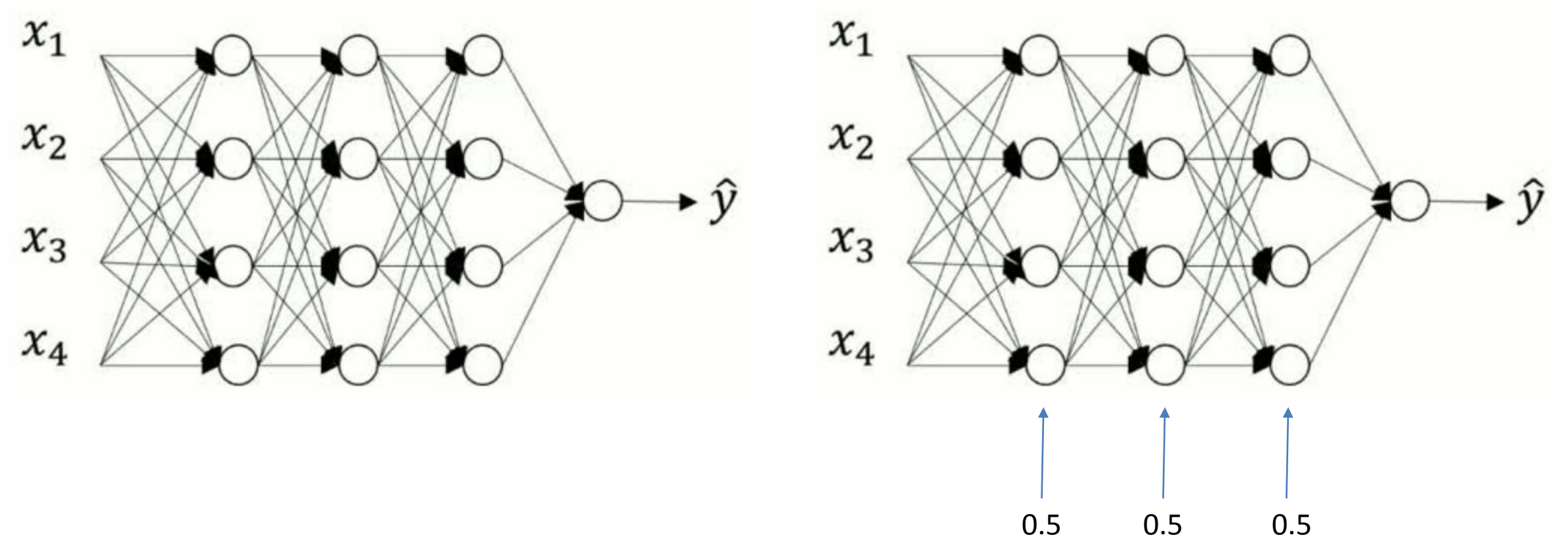




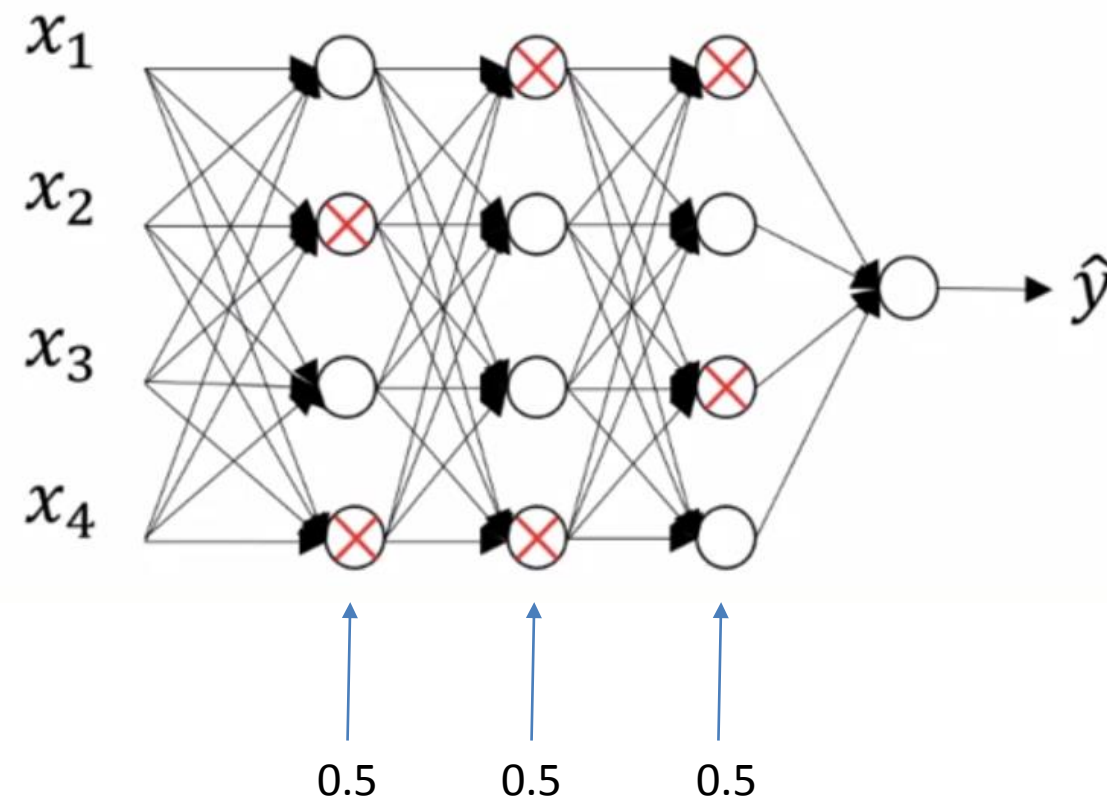
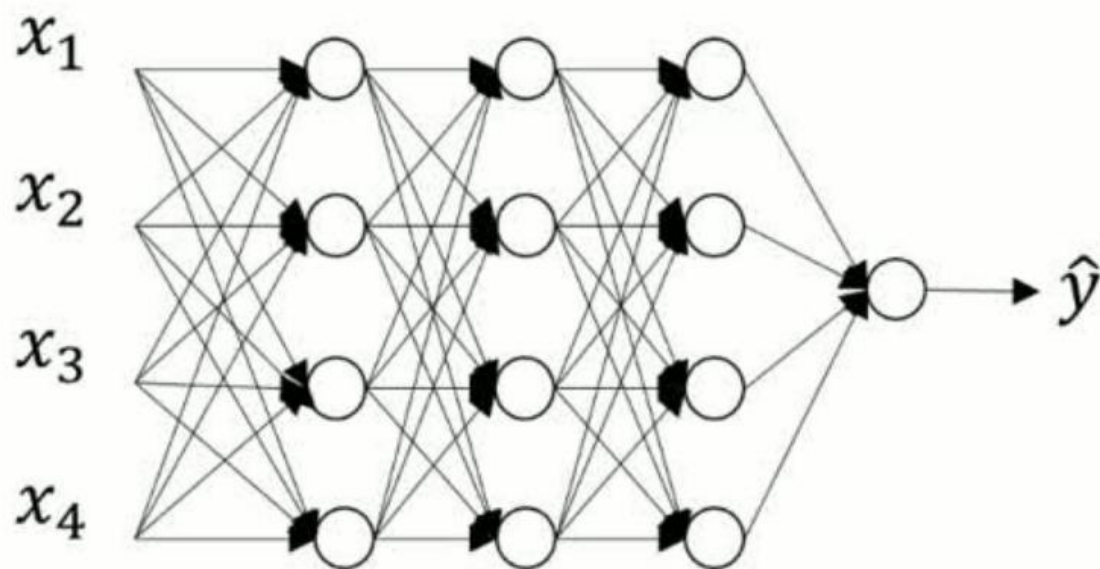
### 3.2.11. Dropout



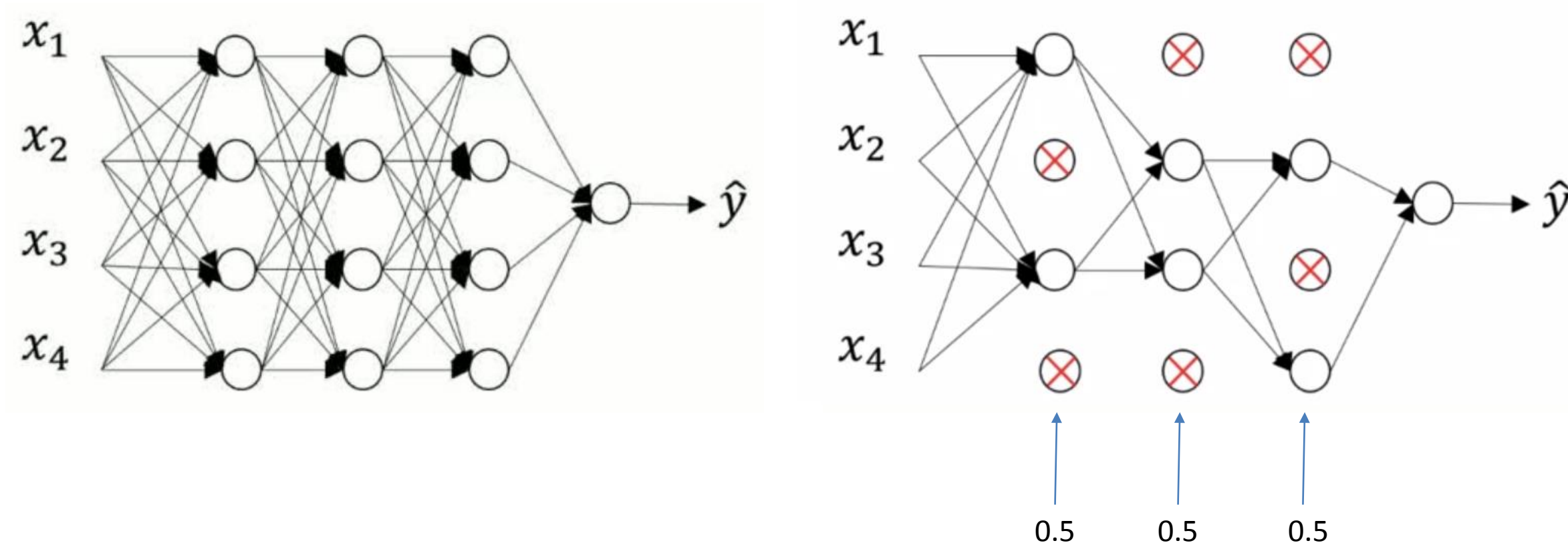
### 3.2.11. Dropout



### 3.2.11. Dropout



### 3.2.11. Dropout



### 3.2.11. Dropout

Dropout (Srivastava et al., 2014) may be the first instance of a human curated artisanal regularization technique that entered wide scale use in machine learning. Dropout, simply described, is the concept that if you can learn how to do a task repeatedly whilst drunk, you should be able to do the task even better when sober. This insight has resulted in numerous state of the art results and a nascent field dedicated

<http://smerity.com/arxiv.org/abs/1804.404/1804.404.pdf?fbclid=IwAR313y1aTjBWThO8A71EDaII0t6tImZVyn6BhzegkXHyyGU9dUY1jPaTlnU>



### 3.2.11. Dropout

#### Implementación: Inverted Dropout

Aplicamos Dropout a la tercera capa de una red neuronal con probabilidad de 0.8 de mantener cada neurona:  
 $l = 3, keep_{prob} = 0.8$





### 3.2.11. Dropout

#### Implementación: Inverted Dropout

Aplicamos Dropout a la tercera capa de una red neuronal con probabilidad de 0.8 de mantener cada neurona:  
 $l = 3, keep_{prob} = 0.8$

$$d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep_{prob}$$

### 3.2.11. Dropout

#### Implementación: Inverted Dropout

Aplicamos Dropout a la tercera capa de una red neuronal con probabilidad de 0.8 de mantener cada neurona:  
 $l = 3, keep_{prob} = 0.8$

$d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep\_prob$

Crea matriz dropout →      ← Distribución uniforme      ← Mismas dimensiones que a3      ← Binariza resultados

### 3.2.11. Dropout

#### Implementación: Inverted Dropout

Aplicamos Dropout a la tercera capa de una red neuronal con probabilidad de 0.8 de mantener cada neurona:  
 $l = 3, keep_{prob} = 0.8$

$d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep\_prob$

Crea matriz dropout →      Distribución uniforme      Mismas dimensiones que a3      Binariza resultados

$a3 = np.multiply(a3, d3)$  #Multiplicación punto a punto

### 3.2.11. Dropout

#### Implementación: Inverted Dropout

Aplicamos Dropout a la tercera capa de una red neuronal con probabilidad de 0.8 de mantener cada neurona:  
 $l = 3, keep_{prob} = 0.8$

$d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep\_prob$

Crea matriz dropout →  $d3$   
Distribución uniforme →  $np.random.rand$   
Mismas dimensiones que  $a3$  →  $a3.shape[0], a3.shape[1]$   
Binariza resultados →  $< keep\_prob$

$a3 = np.multiply(a3, d3)$  #Multiplicación punto a punto

$a3 = a3 / keep_{prob}$

### 3.2.11. Dropout

#### Dropout en Keras

```
from keras.layers import Input, Dense, BatchNormalization, Flatten, Conv2D, MaxPooling2D, Dropout
from keras.models import Model
```

```
def alex_net(input_shape=(227, 227, 3), classes=2):

    # Define the input as a tensor with shape input_shape
    x_input = Input(input_shape)
    # Stage 1
    x = Conv2D(96, (11, 11), strides=(4, 4), padding='valid', activation='relu', name='conv_1')(x_input)
    x = MaxPooling2D(pool_size=(3, 3), strides=(2, 2))(x)
    x = BatchNormalization(axis=3, name='bn_conv1')(x)
    .
    .
    .
    # Stage 4
    x = Flatten(name="flatten")(x)
    x = Dense(4096, activation='relu', name='FC_1')(x)
    x = Dropout(0.5)(x)
    x = Dense(4096, activation='relu', name='FC_2')(x)
    x = Dropout(0.5)(x)
    if classes == 2:
        x = Dense(1, activation='sigmoid', name='FC_3')(x)
    else:
        x = Dense(classes, activation='softmax', name='FC_3')(x)
    # Create model
    model = Model(input=x_input, output=x)

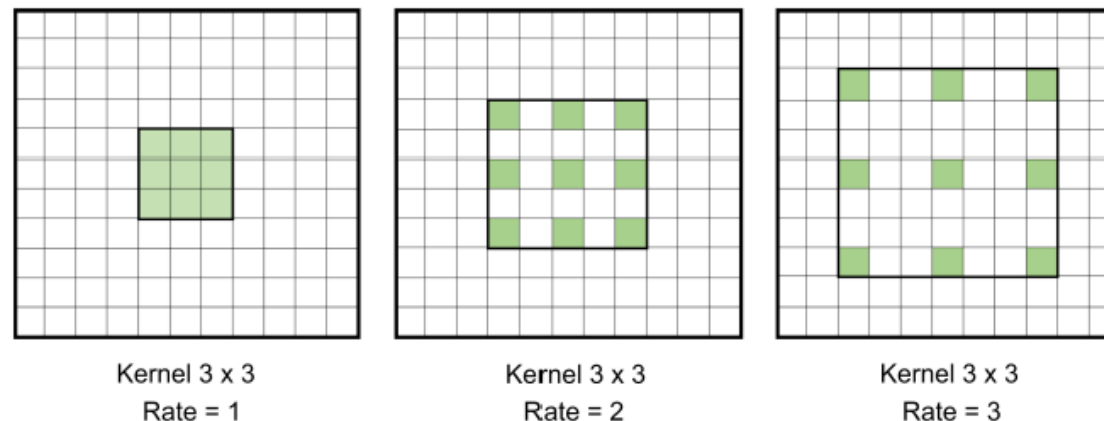
    return model
```



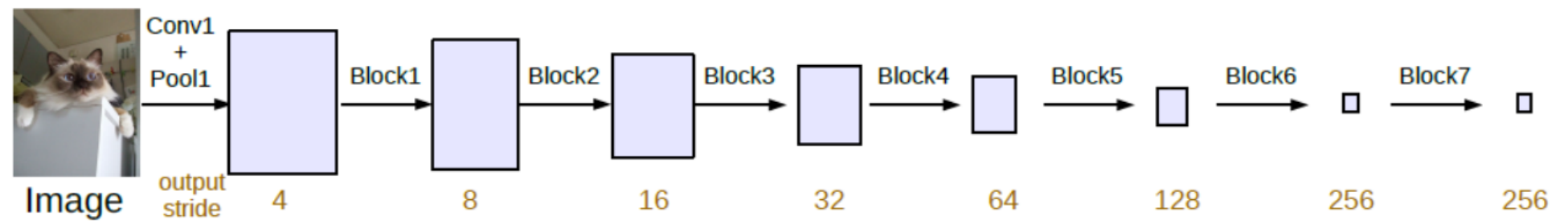
## 3.3. Special Convolutions

### 3.3.1. Atrous Convolution

Conocida también como “Convolución Dilatada”. Es básicamente una convolución con filtros dilatados. Permite incrementar el campo de visión de los filtros sin necesidad de incrementar el número de parámetros o reducir el tamaño original del canal (pooling). Esto permite agregar varios filtros con distintos niveles de dilatación para un análisis multi-escala.

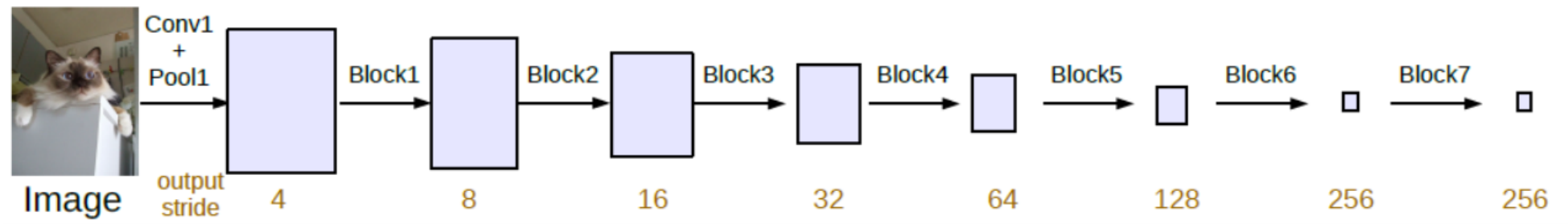


### 3.3.1. Atrous Convolution

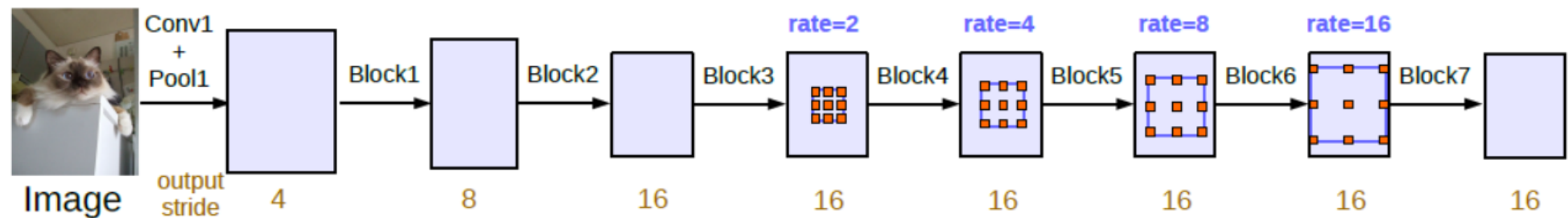


(a) Going deeper without atrous convolution.

### 3.3.1. Atrous Convolution

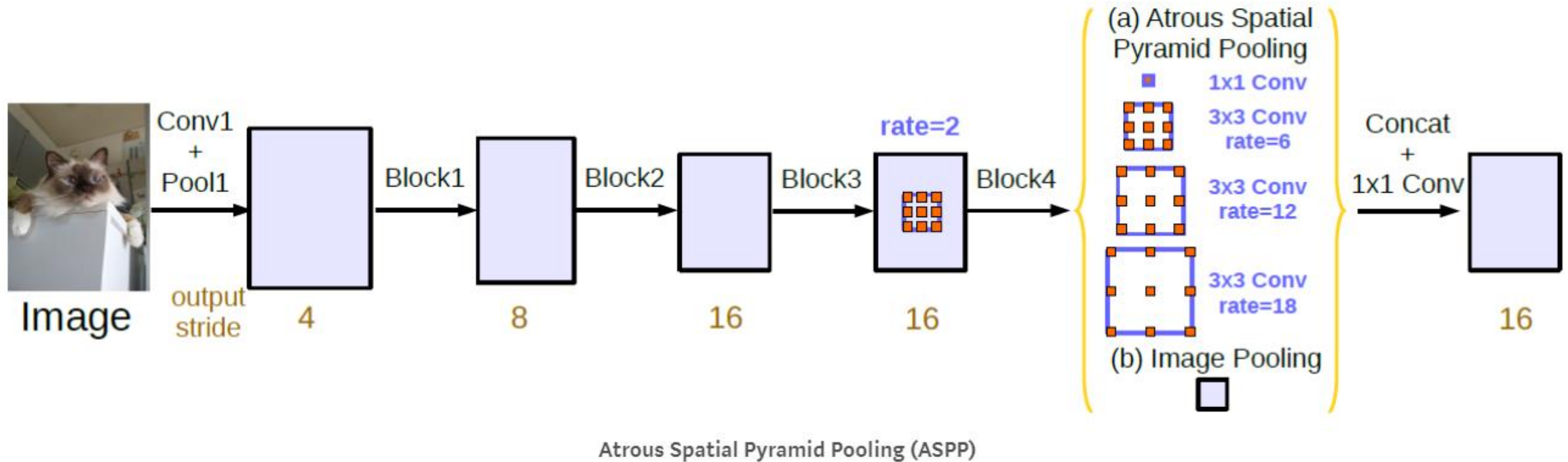


(a) Going deeper without atrous convolution.



(b) Going deeper with atrous convolution. Atrous convolution with  $rate > 1$  is applied after block3 when  $output\_stride = 16$ .

### 3.3.1. Atrous Convolution





### 3.3.1. Atrous Convolution

#### Atrous Convolution in Keras

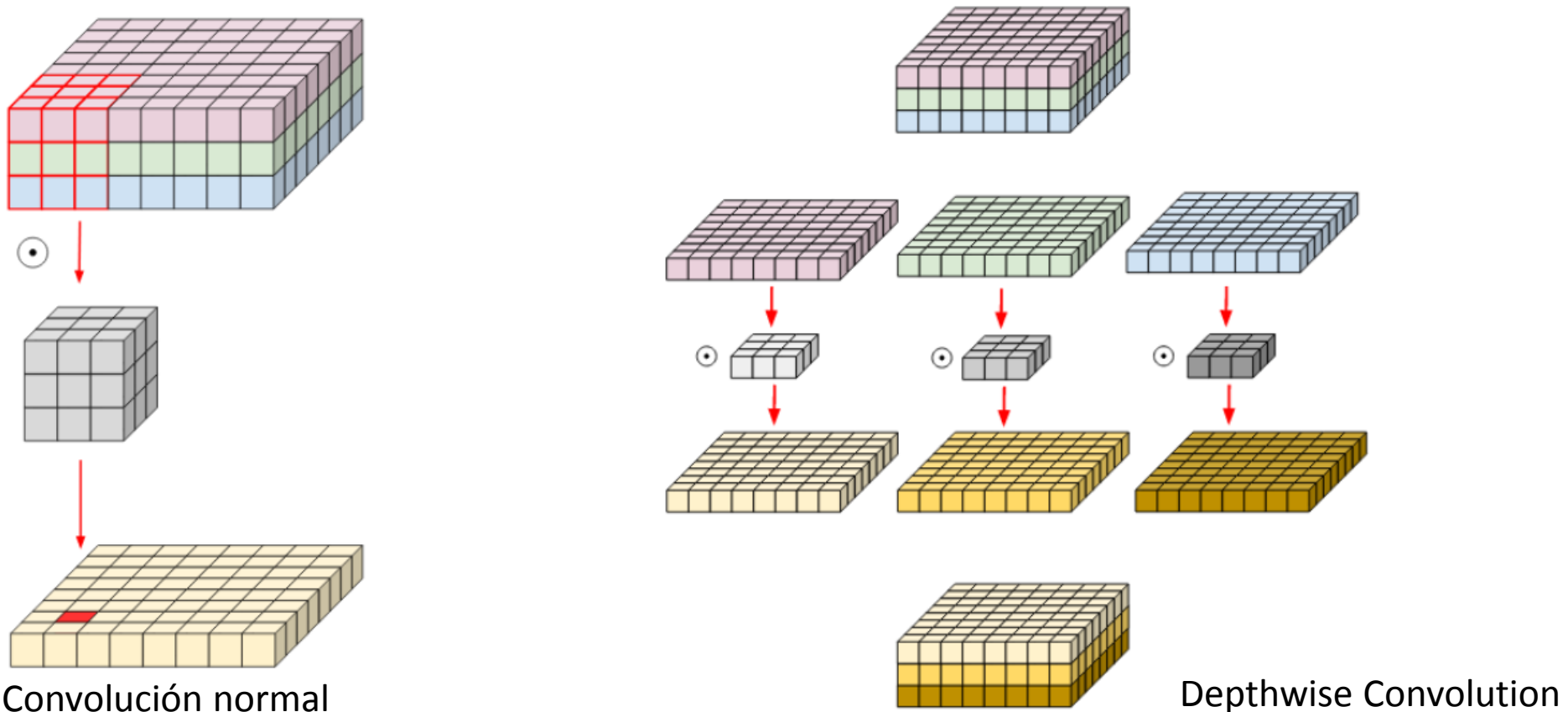
##### Conv2D

```
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,  
dilation_rate=(1, 1),)
```



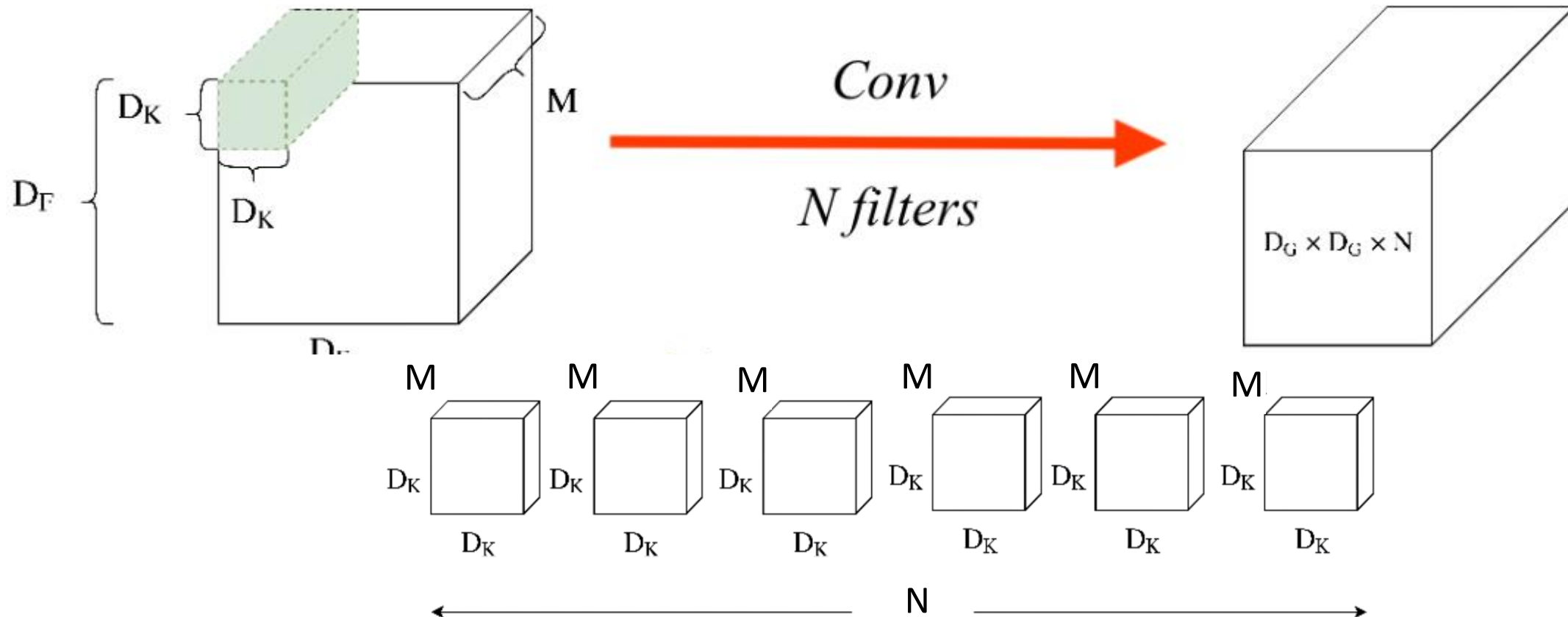
### 3.3.2. Depthwise Convolution

Se aplica un filtro 2D a cada canal (“depth level”) del tensor de entrada.

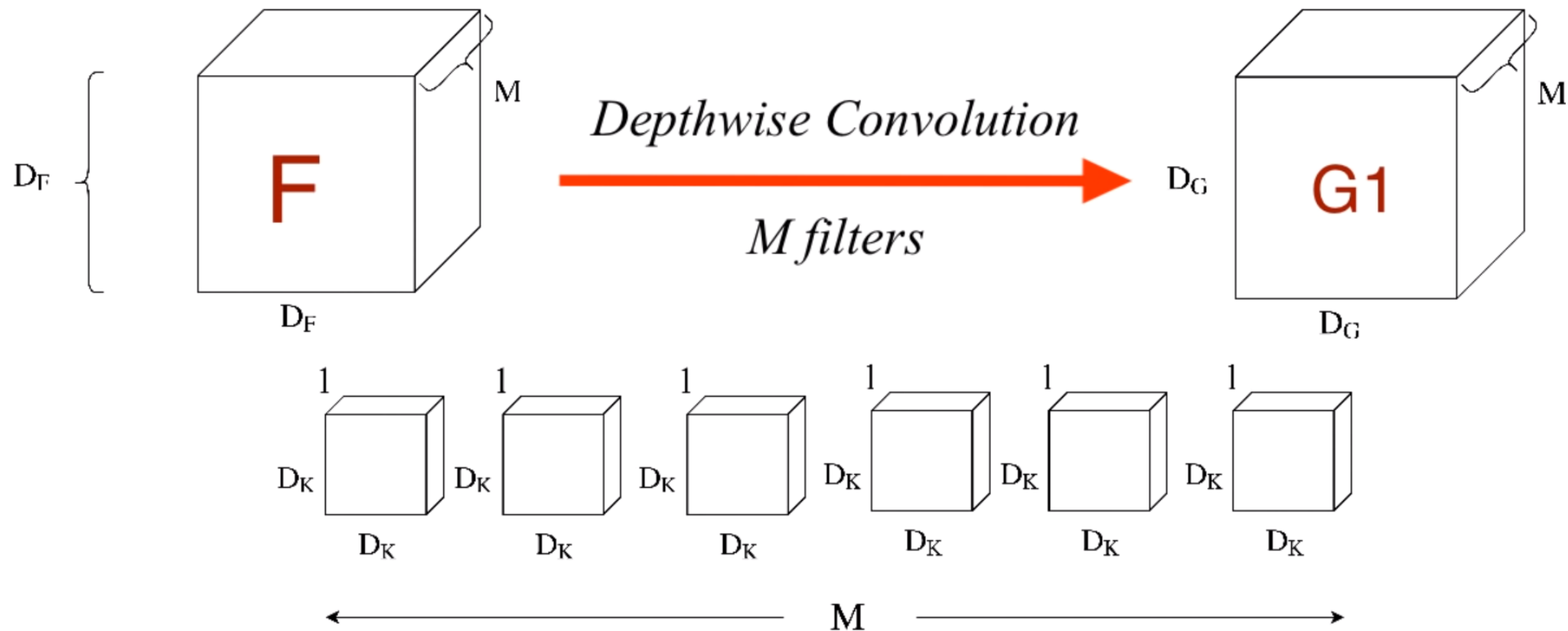


### 3.3.2. Depthwise Convolution

Convolución normal

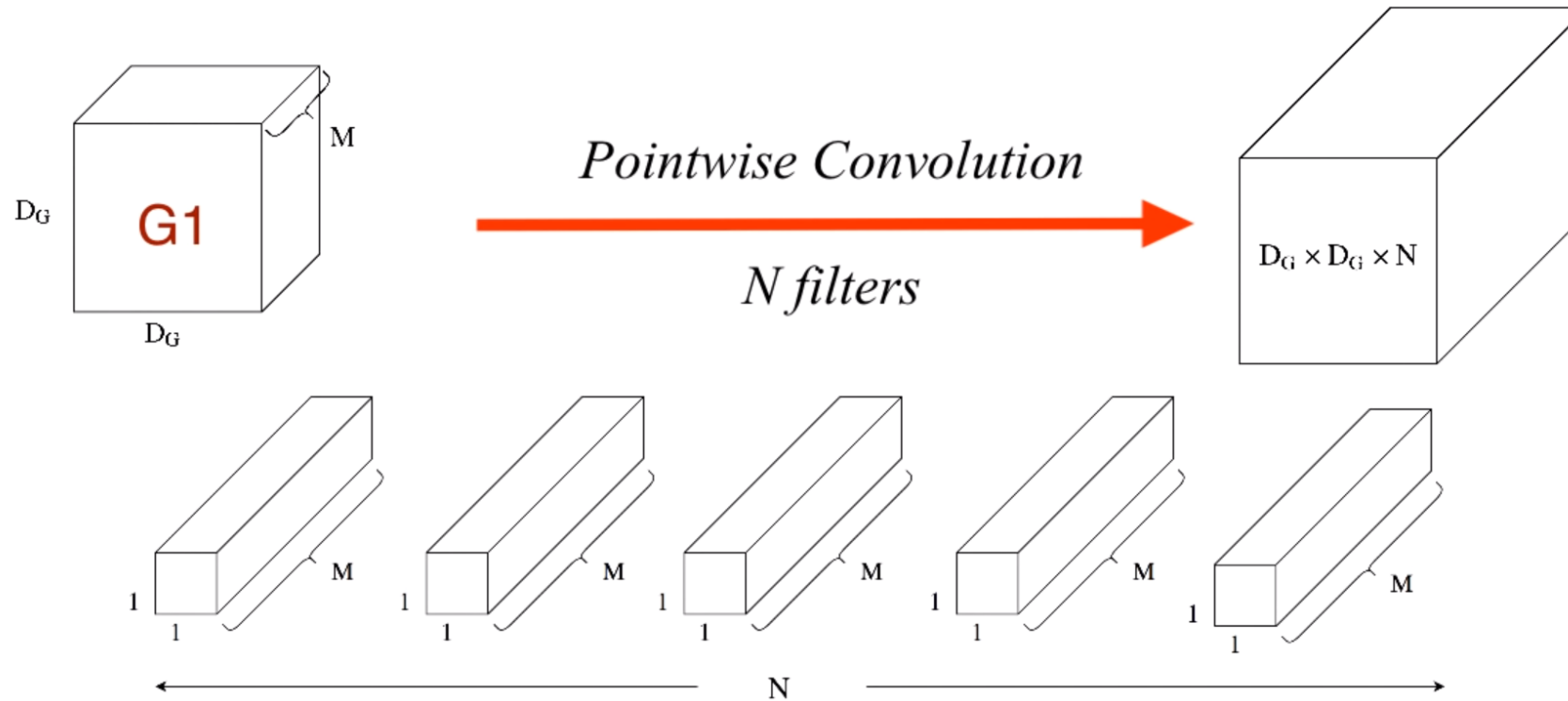


### 3.3.2. Depthwise Convolution



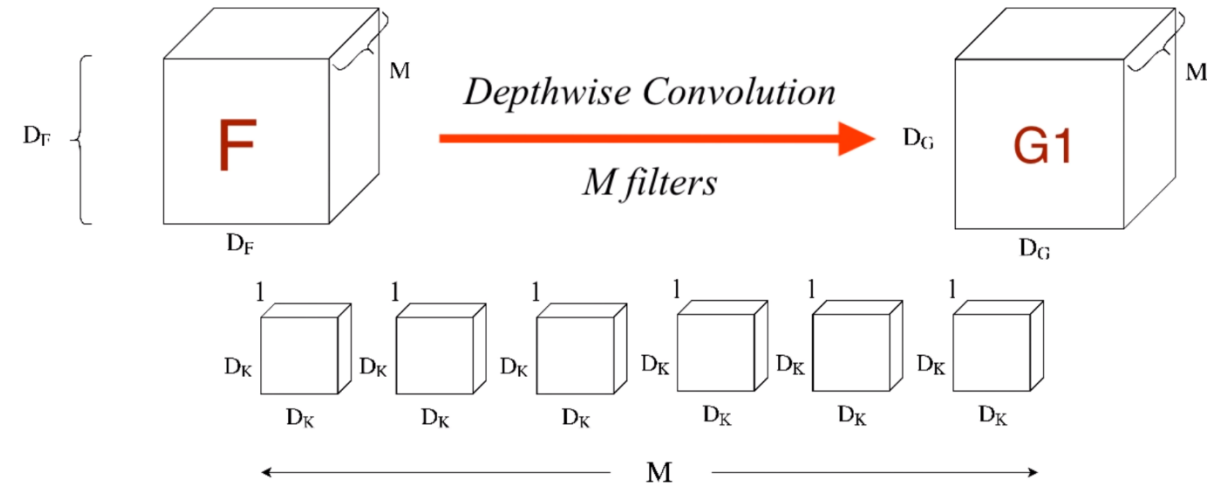
### 3.3.3. Separable Convolution

Consiste en una Depthwise Convolution seguida de una Poitwise Convolution (N filtros de 1x1):



### 3.3.3. Separable Convolution

Paso 1: Depthwise. Cálculo del costo computacional

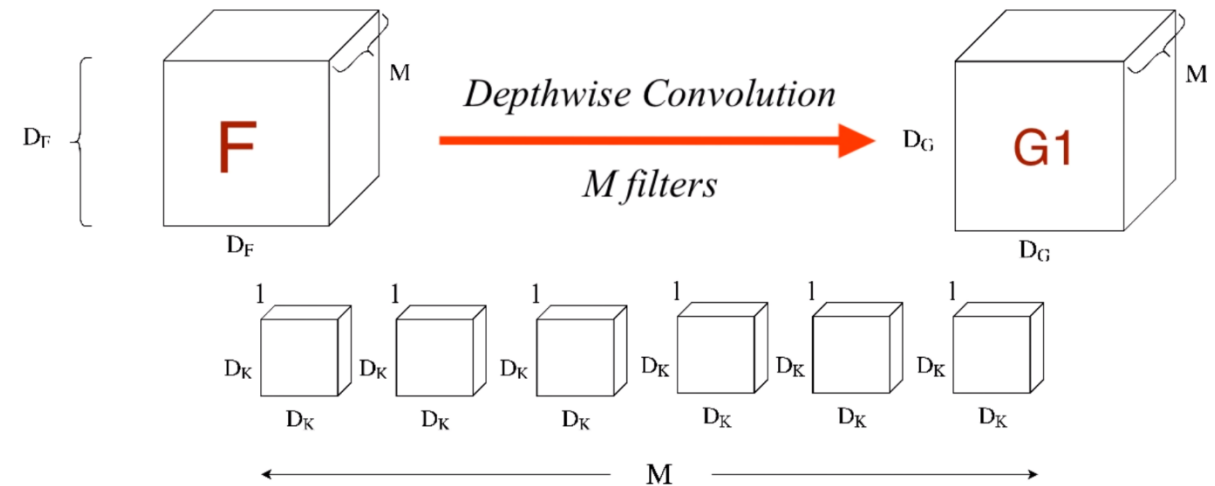




### 3.3.3. Separable Convolution

Paso 1: Depthwise. Cálculo del costo computacional

$$\#Op. por filtro = D_k \times D_k \times 1$$

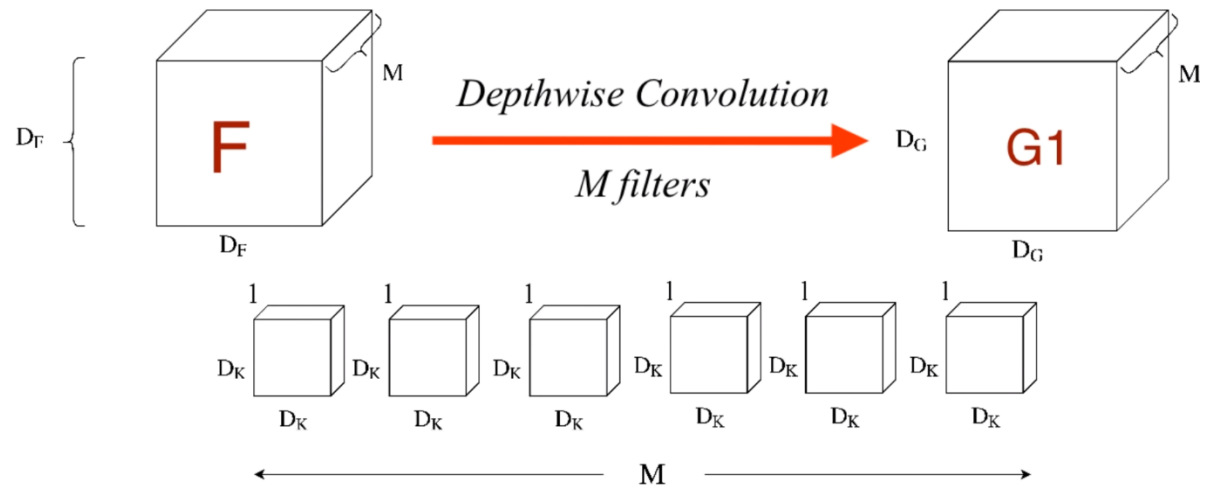


### 3.3.3. Separable Convolution

Paso 1: Depthwise. Cálculo del costo computacional

$$\#Op. por filtro = Dk \times Dk \times 1$$

$$\#Op. por canal = D_G^2 \times Dk^2$$



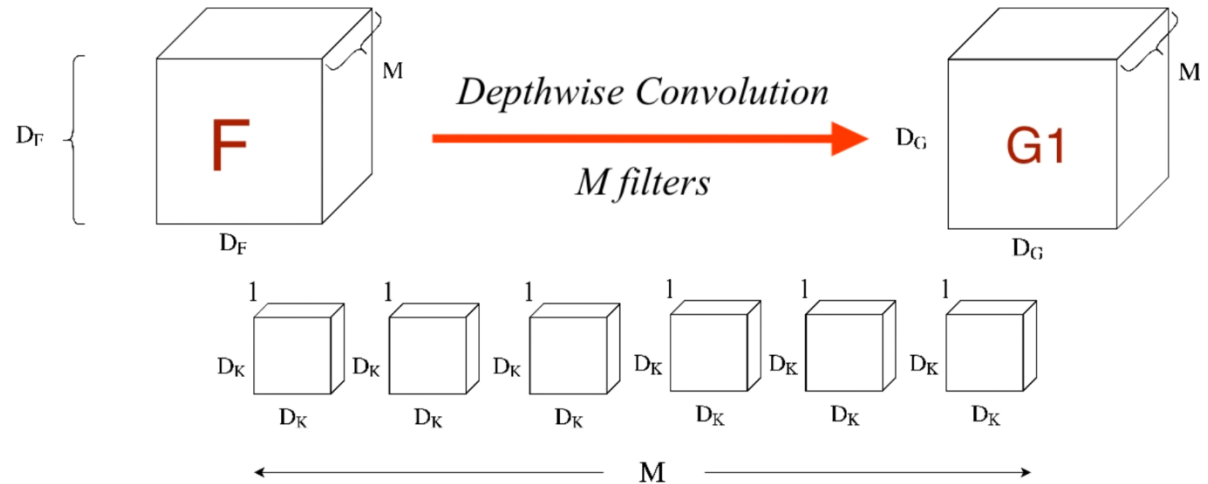
### 3.3.3. Separable Convolution

Paso 1: Depthwise. Cálculo del costo computacional

$$\#Op. por filtro = Dk \times Dk \times 1$$

$$\#Op. por canal = D_G^2 \times Dk^2$$

$$\#Op. total1 = M \times D_G^2 \times Dk^2$$



### 3.3.3. Separable Convolution

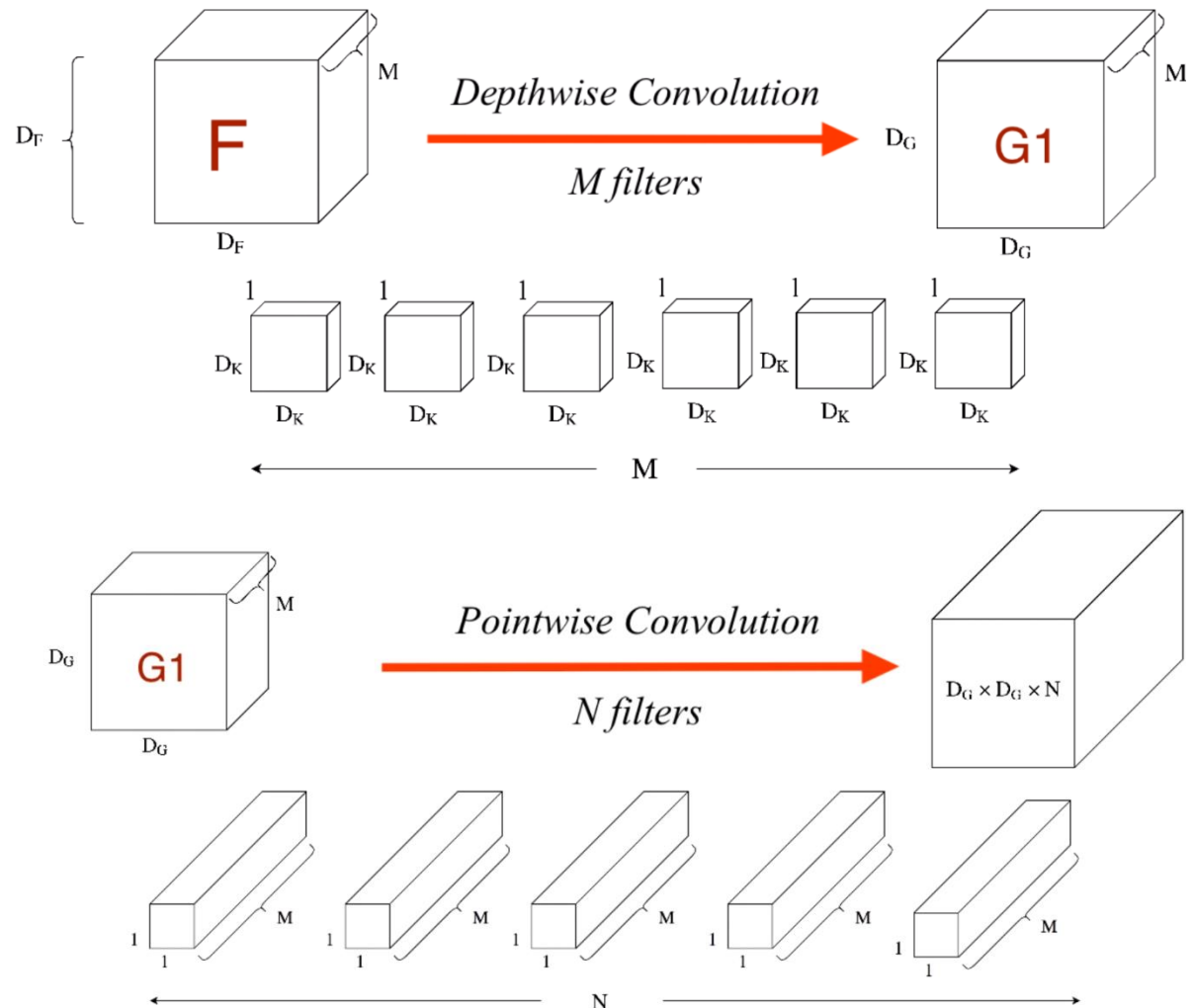
Paso 1: Depthwise. Cálculo del costo computacional

$$\#Op. por filtro = D_k \times D_k \times 1$$

$$\#Op. por canal = D_G^2 \times D_k^2$$

$$\#Op. total1 = M \times D_G^2 \times D_k^2$$

Paso 2: Pointwise.



### 3.3.3. Separable Convolution

Paso 1: Depthwise. Cálculo del costo computacional

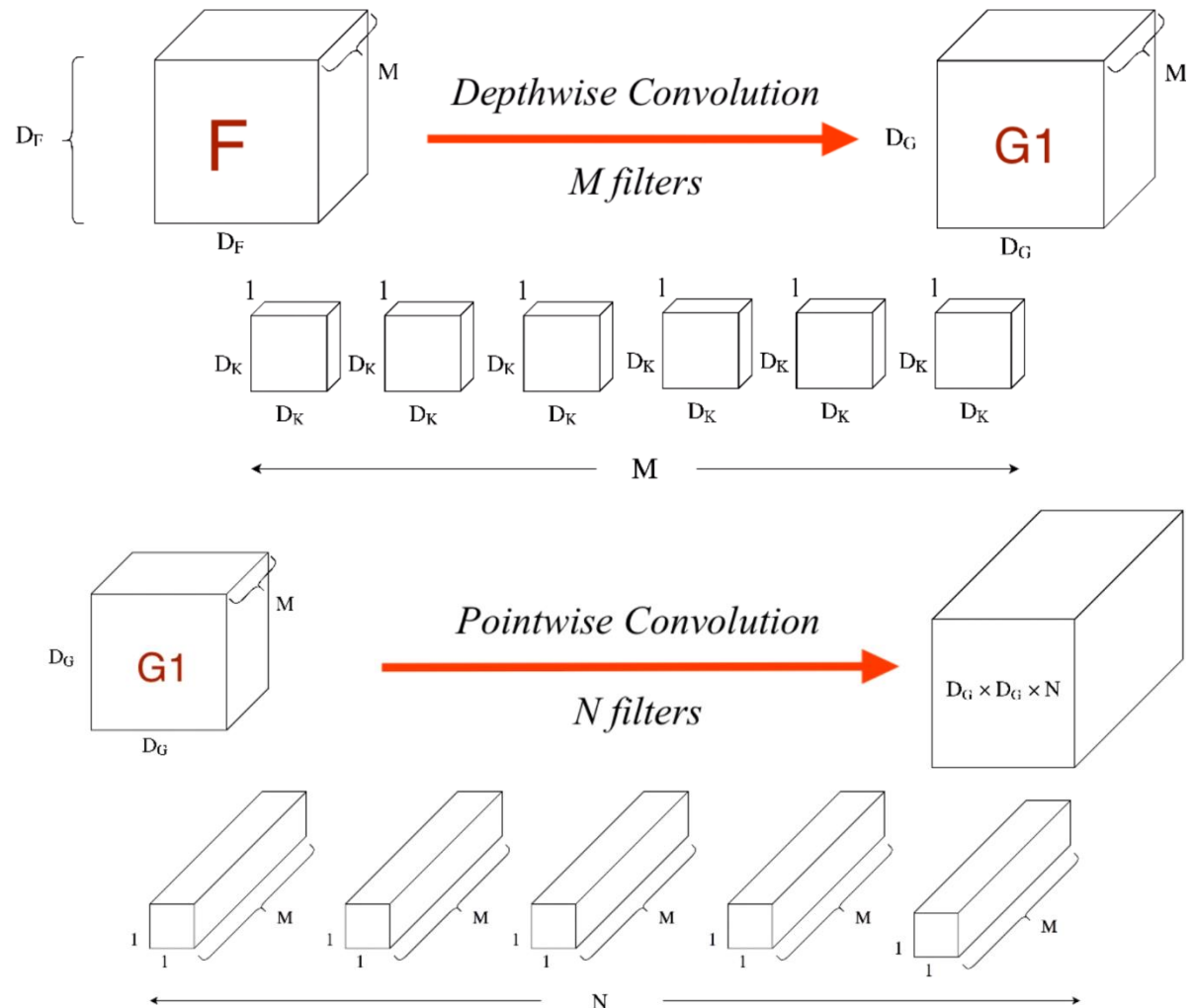
$$\#Op. por filtro = D_k \times D_k \times 1$$

$$\#Op. por canal = D_G^2 \times D_k^2$$

$$\#Op. total1 = M \times D_G^2 \times D_k^2$$

Paso 2: Pointwise.

$$\#Op. por filtro = 1 \times 1 \times M$$





### 3.3.3. Separable Convolution

Paso 1: Depthwise. Cálculo del costo computacional

$$\#Op. por filtro = D_k \times D_k \times 1$$

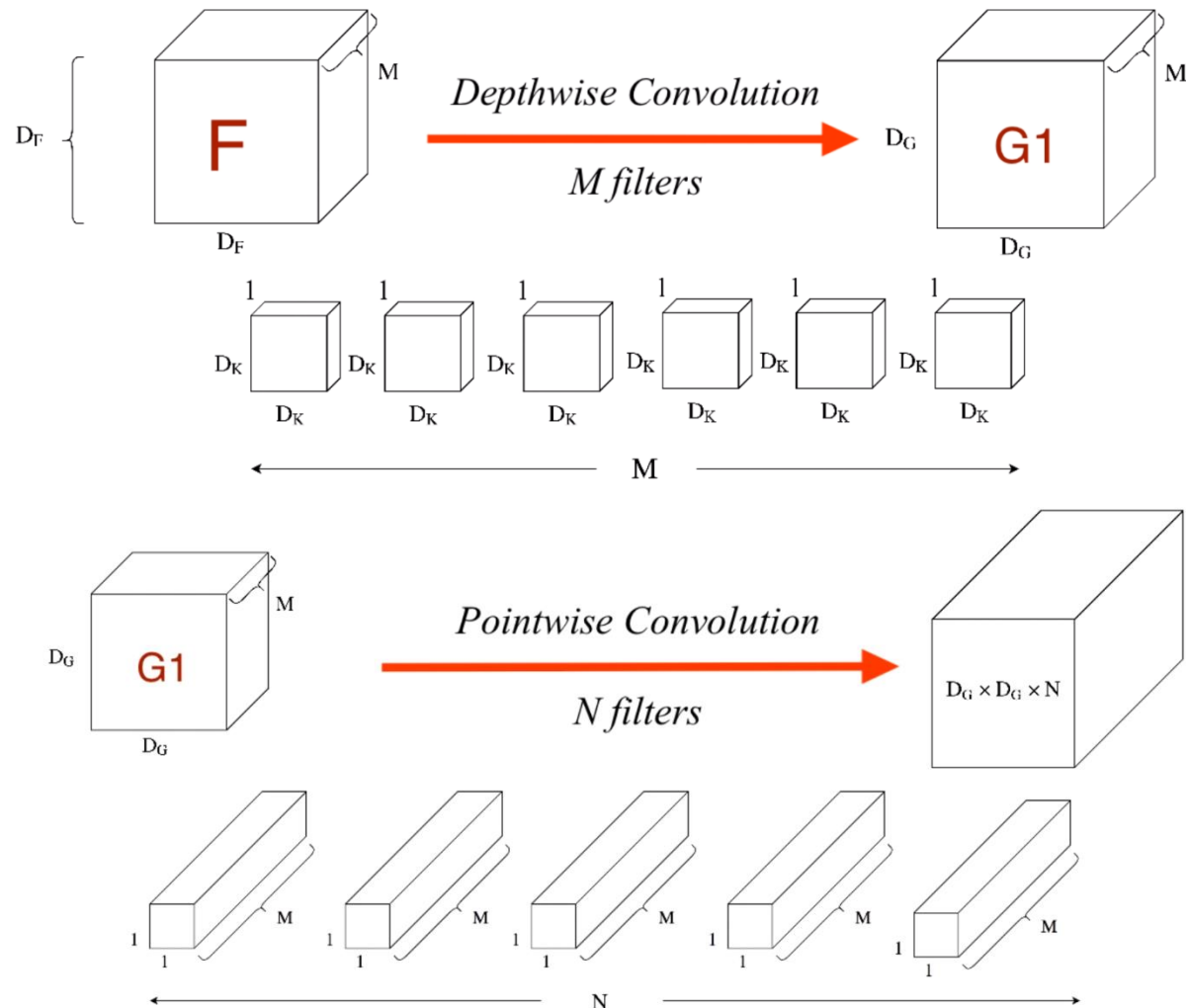
$$\#Op. por canal = D_G^2 \times D_k^2$$

$$\#Op. total1 = M \times D_G^2 \times D_k^2$$

Paso 2: Pointwise.

$$\#Op. por filtro = 1 \times 1 \times M$$

$$\#Op. por canal = D_G^2 \times M$$



### 3.3.3. Separable Convolution

Paso 1: Depthwise. Cálculo del costo computacional

$$\#Op. por filtro = D_k \times D_k \times 1$$

$$\#Op. por canal = D_G^2 \times D_k^2$$

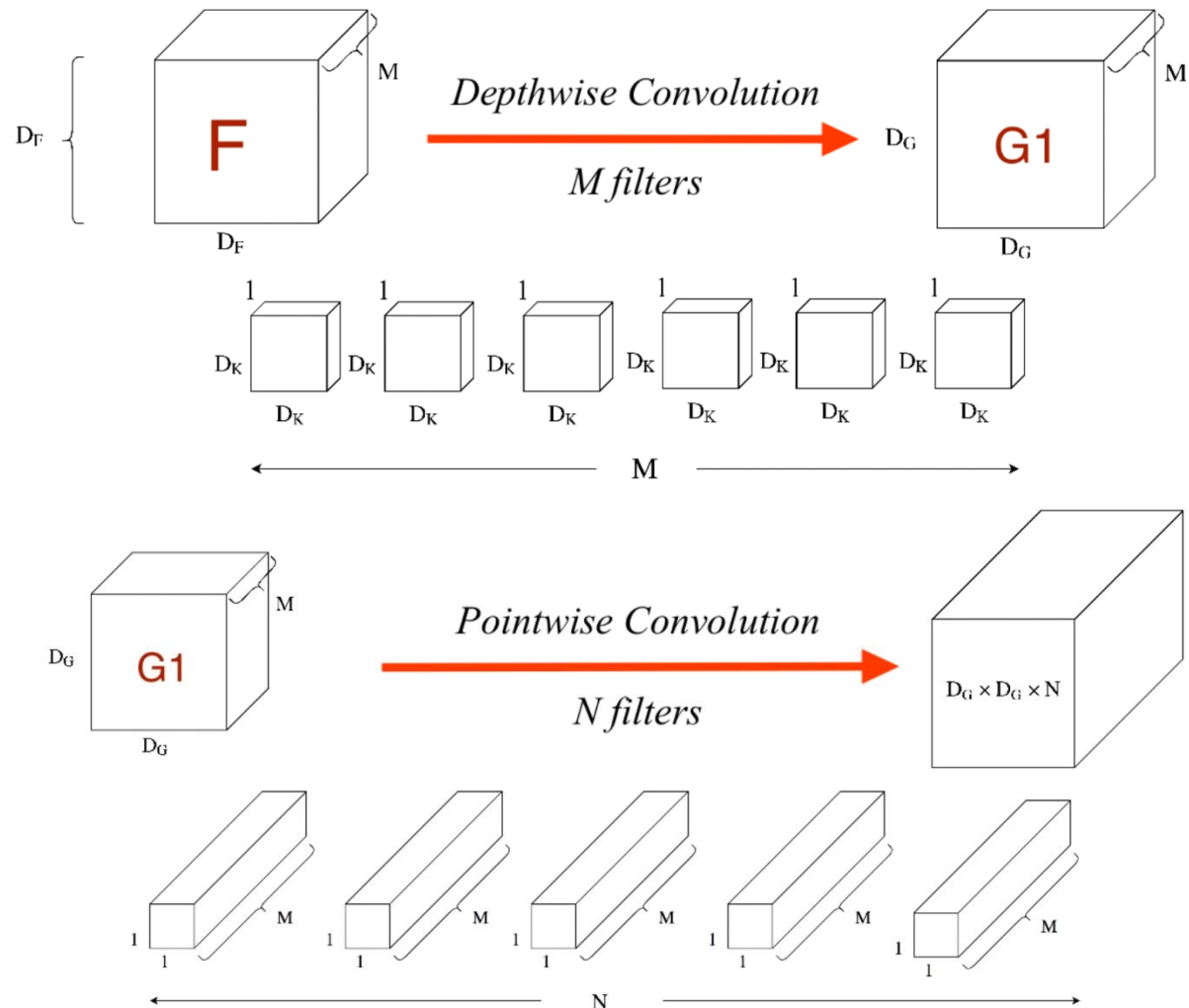
$$\#Op. total1 = M \times D_G^2 \times D_k^2$$

Paso 2: Pointwise.

$$\#Op. por filtro = 1 \times 1 \times M$$

$$\#Op. por canal = D_G^2 \times M$$

$$\#Op. total2 = N \times D_G^2 \times M$$



### 3.3.3. Separable Convolution

Paso 1: Depthwise. Cálculo del costo computacional

$$\#Op. por filtro = Dk \times Dk \times 1$$

$$\#Op. por canal = D_G^2 \times Dk^2$$

$$\#Op. total1 = M \times D_G^2 \times Dk^2$$

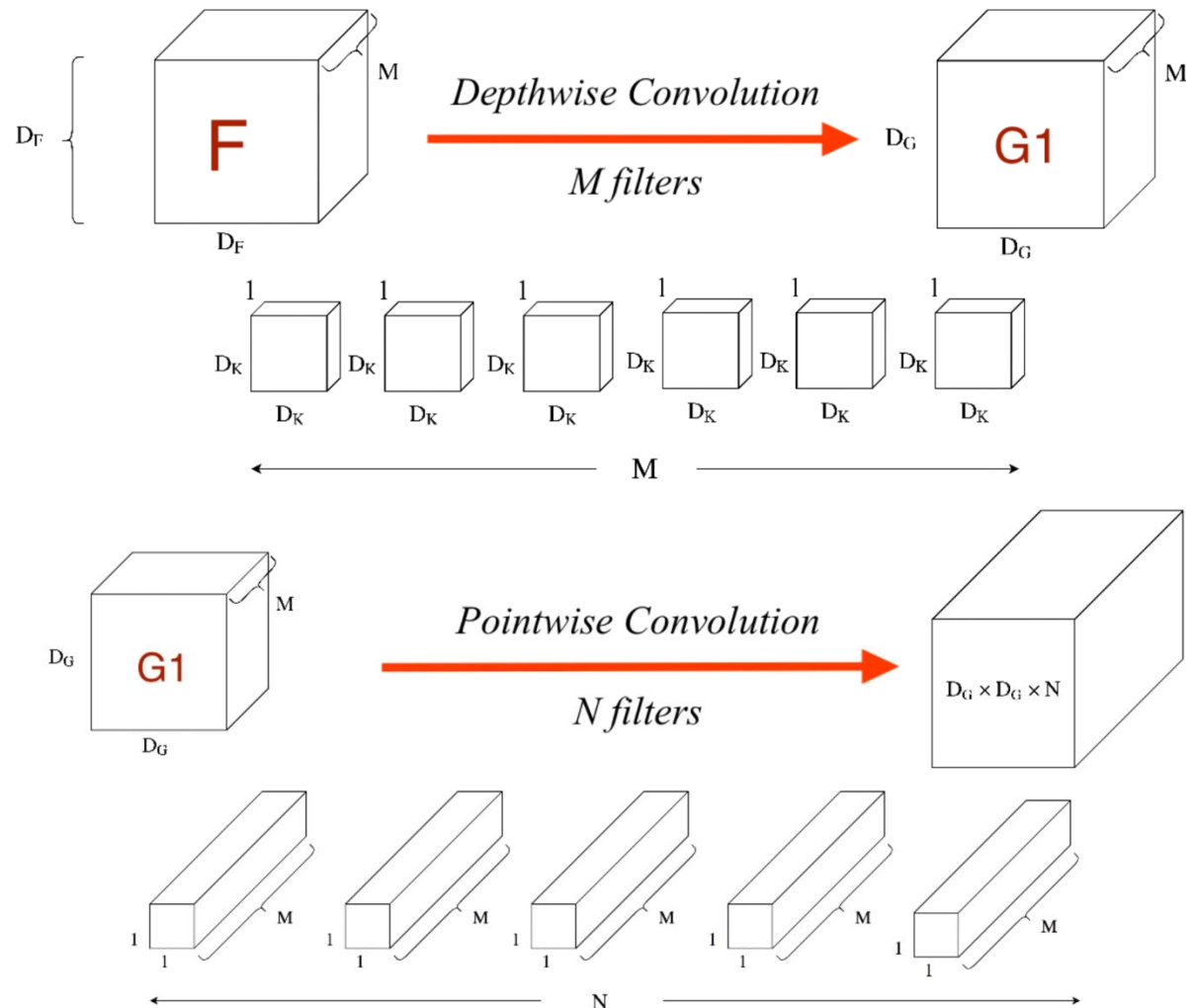
Paso 2: Pointwise.

$$\#Op. por filtro = 1 \times 1 \times M$$

$$\#Op. por canal = D_G^2 \times M$$

$$\#Op. total2 = N \times D_G^2 \times M$$

$$Total = D_G^2 \times M \times (Dk^2 + N)$$



### 3.3.3. Separable Convolution

Cálculo del costo computacional de una convolución normal (ver Día 2, Sección 3.2.5)



### 3.3.3. Separable Convolution

Cálculo del costo computacional de una convolución normal (ver Día 2, Sección 3.2.5)



#Operaciones por filtro =  $D_k \times D_k \times M$

### 3.3.3. Separable Convolution

Cálculo del costo computacional de una convolución normal (ver Día 2, Sección 3.2.5)



#Operaciones por filtro =  $D_k \times D_k \times M$

#Operaciones en total =  $(D_G \times D_G \times N) \times (D_k \times D_k \times M) = N \times M \times D_G^2 \times D_k^2$



### 3.3.3. Separable Convolution

Comparación del cálculo del costo computacional de una convolución normal y una separable convolution

$$\frac{\#Op. Depthwise}{\#Op. conv. normal} = \frac{D_G^2 \times M \times (Dk^2 + N)}{N \times M \times D_G^2 \times Dk^2}$$

### 3.3.3. Separable Convolution

Comparación del cálculo del costo computacional de una convolución normal y una separable convolution

$$\frac{\#Op. Depthwise}{\#Op. conv. normal} = \frac{D_G^2 \times \cancel{M} \times (Dk^2 + N)}{N \times \cancel{M} \times \cancel{D_G^2} \times Dk^2}$$

### 3.3.3. Separable Convolution

Comparación del cálculo del costo computacional de una convolución normal y una separable convolution

$$\frac{\#Op. Depthwise}{\#Op. conv. normal} = \frac{\cancel{D_G^2} \times \cancel{M} \times (Dk^2 + N)}{N \times \cancel{M} \times \cancel{D_G^2} \times Dk^2}$$

$$\frac{\#Op. Depthwise}{\#Op. conv. normal} = \frac{Dk^2 + N}{Dk^2 \times N} = \frac{1}{N} + \frac{1}{Dk^2}$$

### 3.3.3. Separable Convolution

Comparación del cálculo del costo computacional de una convolución normal y una separable convolution

$$\frac{\#Op. Depthwise}{\#Op. conv. normal} = \frac{\cancel{D_G^2} \times \cancel{M} \times (Dk^2 + N)}{N \times \cancel{M} \times \cancel{D_G^2} \times Dk^2}$$

$$\frac{\#Op. Depthwise}{\#Op. conv. normal} = \frac{Dk^2 + N}{Dk^2 \times N} = \frac{1}{N} + \frac{1}{Dk^2}$$

$$N = 1024 \qquad Dk = 3$$

### 3.3.3. Separable Convolution

Comparación del cálculo del costo computacional de una convolución normal y una separable convolution

$$\frac{\#Op. Depthwise}{\#Op. conv. normal} = \frac{\cancel{D_G^2} \times \cancel{M} \times (Dk^2 + N)}{N \times \cancel{M} \times \cancel{D_G^2} \times Dk^2}$$

$$\frac{\#Op. Depthwise}{\#Op. conv. normal} = \frac{Dk^2 + N}{Dk^2 \times N} = \frac{1}{N} + \frac{1}{Dk^2}$$

$$N = 1024 \quad Dk = 3$$

$$\frac{\#Op. Depthwise}{\#Op. conv. normal} = \frac{1}{1024} + \frac{1}{3k^2} = 0.112$$



### 3.3.3. Atrous Separable Convolution (ASC)

Depthwise con filtros dilatados + Pointwise Convolution

**SeparableConv2D**

```
keras.layers.SeparableConv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,  
dilation_rate=(1, 1), depth_multiplier=1, activation=None, use_bias=True)
```

<https://keras.io/layers/convolutional/>



## 3.4. Training with Generators

### ImageGenerator in Keras

- **featurewise\_center**: Boolean. Normaliza a 0 el promedio de todo el dataset.
- **samplewise\_center**: Boolean. Setea a 0 el promedio de cada muestra.
- **featurewise\_std\_normalization**: Boolean. Divide los input por la std de todo el dataset.
- **samplewise\_std\_normalization**: Boolean. Divide cada input por su std.
- **rotation\_range**: Int. Rango de grados para rotaciones aleatorias.
- **width\_shift\_range / height\_shift\_range** : Float, Rangos (fracciones del ancho o alto totals) en los que aleatoriamente se moverán las imágenes horizontalmente o verticalmente.
- **shear\_range**: Float. Rango de ángulo de corte en grados.
- **zoom\_range**: Float. Rango de zoom.
- **horizontal\_flip / vertical\_flip**: Boolean. Voltea horizontal y verticalmente aleatoriamente.

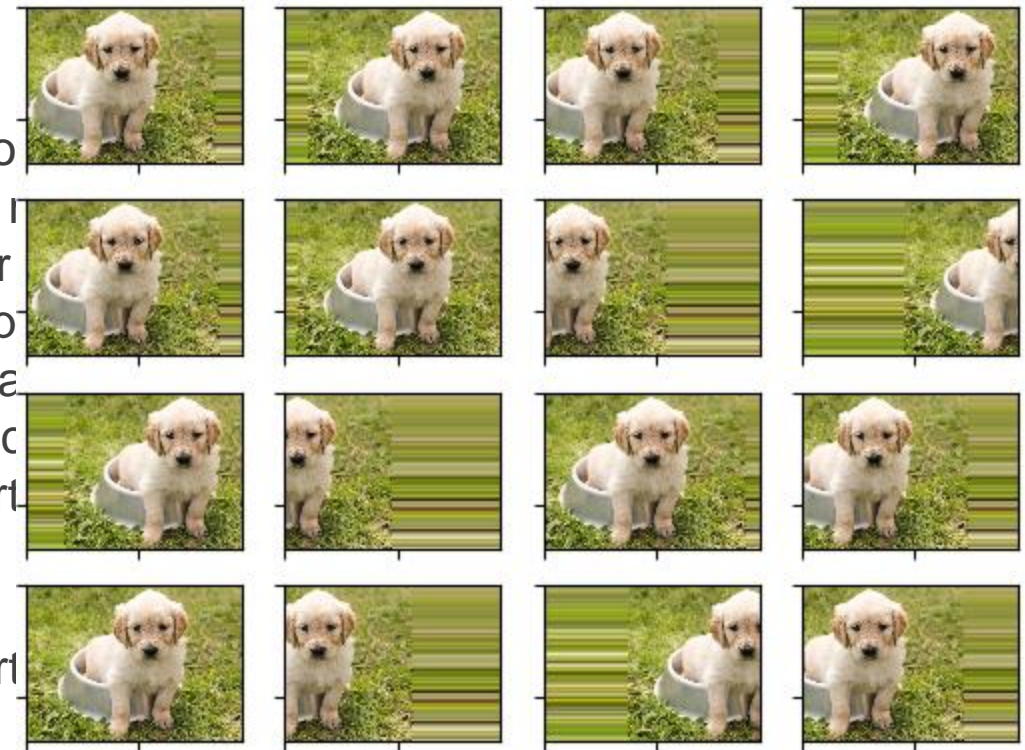
<https://keras.io/preprocessing/image/>

<https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html>

## 3.4. Training with Generators

### ImageGenerator in Keras

- **featurewise\_center**: Boolean. Normaliza a 0 el promedio de todos los canales de cada imagen.
- **samplewise\_center**: Boolean. Setea a 0 el promedio de cada una de las imágenes.
- **featurewise\_std\_normalization**: Boolean. Divide los input por el promedio de cada canal.
- **samplewise\_std\_normalization**: Boolean. Divide cada input por el promedio de cada imagen.
- **rotation\_range**: Int. Rango de grados para rotaciones aleatorias.
- **width\_shift\_range** / **height\_shift\_range** : Float, Rangos (fracción de la imagen) por la que aleatoriamente se moverán las imágenes horizontalmente o verticalmente.
- **shear\_range**: Float. Rango de ángulo de corte en grados.
- **zoom\_range**: Float. Rango de zoom.
- **horizontal\_flip** / **vertical\_flip**: Boolean. Voltea horizontal y verticalmente.



<https://keras.io/preprocessing/image/>

<https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html>

## 3.4. Training with Generators

### ImageGenerator in Keras

- **featurewise\_center**: Boolean. Normaliza a 0 el promedio de todo el dataset.
- **samplewise\_center**: Boolean. Setea a 0 el promedio de cada muestra.
- **featurewise\_std\_normalization**: Boolean. Divide los input por la std de todo el dataset.
- **samplewise\_std\_normalization**: Boolean. Divide cada input por su std.
- **rotation\_range**: Int. Rango de grados para rotaciones aleatorias.
- **width\_shift\_range / height\_shift\_range** : Float, Rangos (fracciones del ancho o alto totals) en los que aleatoriamente se moverán las imágenes horizontalmente o verticalmente.
- **shear\_range**: Float. Rango de ángulo de corte en grados.
- **zoom\_range**: Float. Rango de zoom.
- **horizontal\_flip / vertical\_flip**: Boolean. Voltea horizontal y verticalmente aleatoriamente.

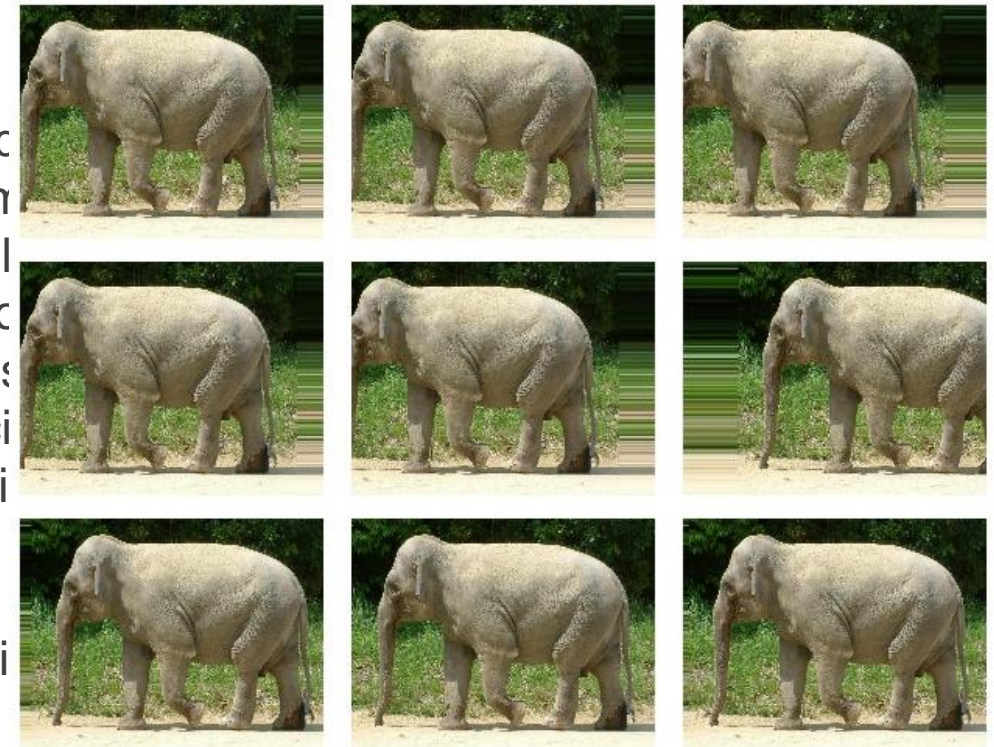
<https://keras.io/preprocessing/image/>

<https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html>

## 3.4. Training with Generators

### ImageGenerator in Keras

- **featurewise\_center**: Boolean. Normaliza a 0 el promedio de todos los features.
- **samplewise\_center**: Boolean. Setea a 0 el promedio de cada muestra.
- **featurewise\_std\_normalization**: Boolean. Divide los input por la desviación estándar de todos los features.
- **samplewise\_std\_normalization**: Boolean. Divide cada input por la desviación estándar de la muestra.
- **rotation\_range**: Int. Rango de grados para rotaciones aleatorias.
- **width\_shift\_range** / **height\_shift\_range** : Float, Rangos (fracciones de la imagen) para mover las imágenes horizontalmente o verticalmente.
- **shear\_range**: Float. Rango de ángulo de corte en grados.
- **zoom\_range**: Float. Rango de zoom.
- **horizontal\_flip** / **vertical\_flip**: Boolean. Voltea horizontal y verticalmente.



<https://keras.io/preprocessing/image/>

<https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html>



## 3.4. Training with Generators

### ImageGenerator in Keras

- **featurewise\_center**: Boolean. Normaliza a 0 el promedio de todo el dataset.
- **samplewise\_center**: Boolean. Setea a 0 el promedio de cada muestra.
- **featurewise\_std\_normalization**: Boolean. Divide los input por la std de todo el dataset.
- **samplewise\_std\_normalization**: Boolean. Divide cada input por su std.
- **rotation\_range**: Int. Rango de grados para rotaciones aleatorias.
- **width\_shift\_range / height\_shift\_range** : Float, Rangos (fracciones del ancho o alto totals) en los que aleatoriamente se moverán las imágenes horizontalmente o verticalmente.
- **shear\_range**: Float. Rango de ángulo de corte en grados.
- **zoom\_range**: Float. Rango de zoom.
- **horizontal\_flip / vertical\_flip**: Boolean. Voltea horizontal y verticalmente aleatoriamente.

<https://keras.io/preprocessing/image/>

<https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html>

## 3.4. Training with Generators

### Flow

Toma la data “x” y sus target “y” y genera batches de data aumentada

```
datagen.flow(x_train, y_train,  
batch_size=32)
```

<https://keras.io/preprocessing/image/>

### Flow from directory

Toma como entrada la carpeta donde están los datos separados en subcarpetas (1 por clase) y genera los batches con data augmentation.

```
train_generator = datagen.flow_from_directory(  
    train_data_dir,  
    target_size=(img_width, img_height),  
    batch_size=batch_size,  
    class_mode='binary',  
    subset='training')
```

```
data/  
  train/  
    dogs/  
      dog001.jpg  
      dog002.jpg  
      ...  
    cats/  
      cat001.jpg  
      cat002.jpg  
      ...  
  validation/  
    dogs/  
      dog001.jpg  
      dog002.jpg  
      ...  
    cats/  
      cat001.jpg  
      cat002.jpg
```





## 3.4. Training with Generators

**ImageGenerator in Keras: CatsvsDogAugmentation.py**