Sapienza Training Camp 2020

Building an Image Search Engine

3 - 5 September, 2020

Deep Neural Networks (DNN)

```
model = tf.keras.Sequential([
      tf.keras.layers.Flatten(input_shape=([28, 28, 1]),
      tf.keras.layers.Dense(200, activation='relu'),
      tf.keras.layers.Dense(60, activation='relu'),
      tf.keras.layers.Dense(10, activation='softmax')
  ])

# this configures the training of the model.
# Keras calls it "compiling" the model.
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# train the model
model.fit(training_dataset, ...)
```

Convolutional Neural Networks (CNN)

```
model = tf.keras.Sequential([
    tf.keras.layers.Reshape(input_shape=(28*28,), target_shape=(28, 28, 1)),
    tf.keras.layers.Conv2D(kernel_size=3, filters=12, activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=6, filters=24, strides=2, activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=6, filters=32, strides=2, activation='relu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(10, activation='softmax')
])

# this configures the training of the model.
# Keras calls it "compiling" the model.
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# train the model
model.fit(training_dataset, ...)
```

Convolutional Neural Networks (CNN)

```
model = tf.keras.Sequential([
    tf.keras.layers.Reshape(input_shape=(28*28,), target_shape=(28, 28, 1)),
    tf.keras.layers.Conv2D(kernel_size=3, filters=12, activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=6, filters=24, strides=2, activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=6, filters=32, strides=2, activation='relu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(10, activation='softmax')
])

# this configures the training of the model.
# Keras calls it "compiling" the model.
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# train the model
model.fit(training_dataset, ...)
```

strides?

Downsampling with Stride

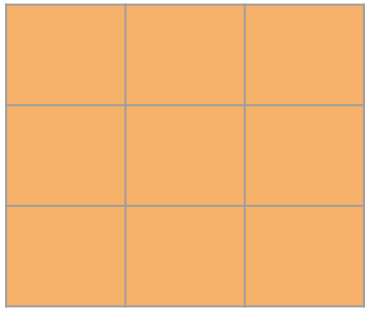
It's often advantageous to downsample the image while convolving. A **stride** dictates how much the input filter shifts between each dot product.

Example: Stride of 2

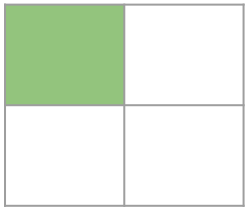
Input Image

0	0	0	0	0
0				0
0				0
0				0
0	0	0	0	0

3x3 Spatial Filter



Output Image

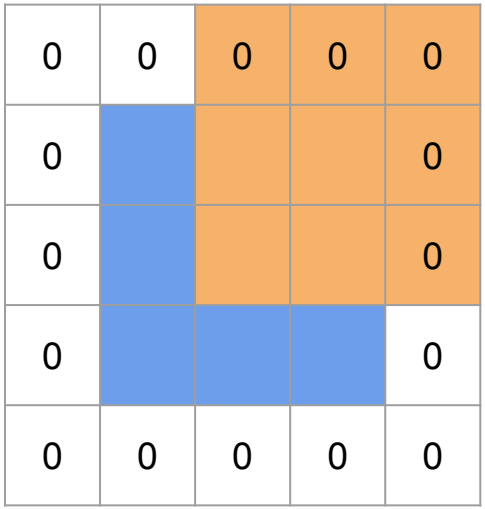


Downsampling with Stride

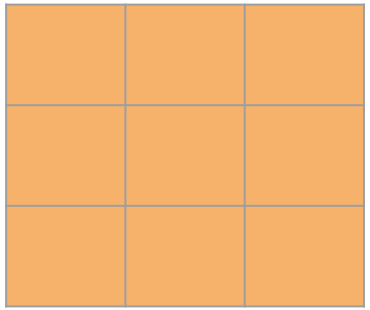
It's often advantageous to downsample the image while convolving. A **stride** dictates how much the input filter shifts between each dot product.

Example: Stride of 2

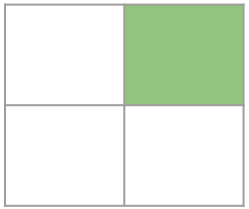
Input Image



3x3 Spatial Filter



Output Image

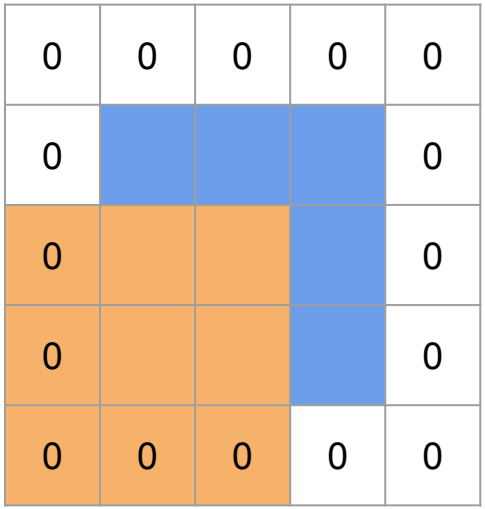


Downsampling with Stride

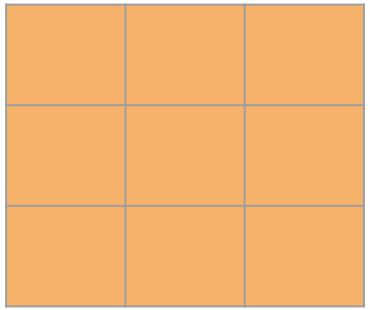
It's often advantageous to downsample the image while convolving. A **stride** dictates how much the input filter shifts between each dot product.

Example: Stride of 2

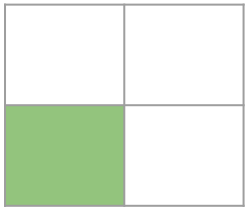
Input Image



3x3 Spatial Filter



Output Image

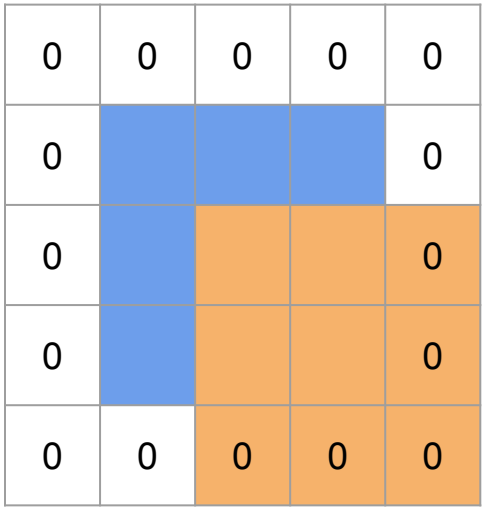


Downsampling with Stride

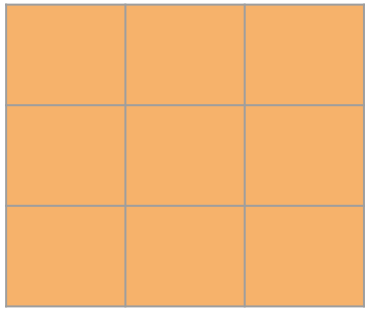
It's often advantageous to downsample the image while convolving. A **stride** dictates how much the input filter shifts between each dot product.

Example: Stride of 2

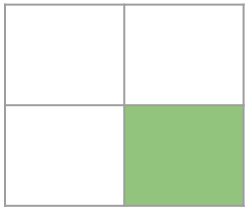
Input Image



3x3 Spatial Filter



Output Image



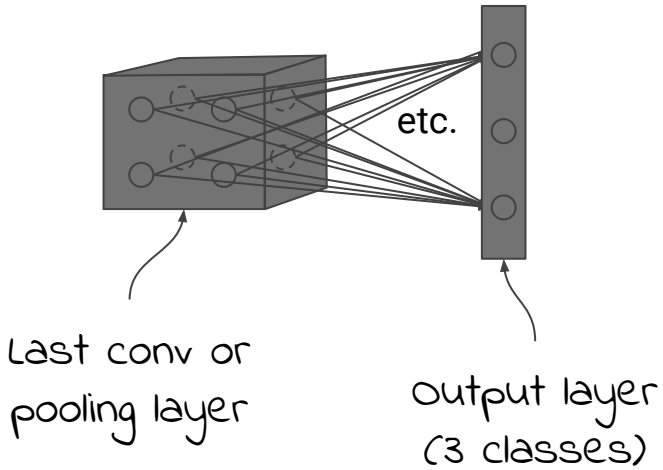
Last layer before softmax

```
model = tf.keras.Sequential([
    tf.keras.layers.Reshape(input_shape=(28*28,), target_shape=(28, 28, 1)),
    tf.keras.layers.Conv2D(kernel_size=3, filters=12, activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=6, filters=24, strides=2, activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=6, filters=32, strides=2, activation='relu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(10, activation='softmax')
])

# this configures the training of the model.
# Keras calls it "compiling" the model.
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# train the model
model.fit(training_dataset, ...)
```

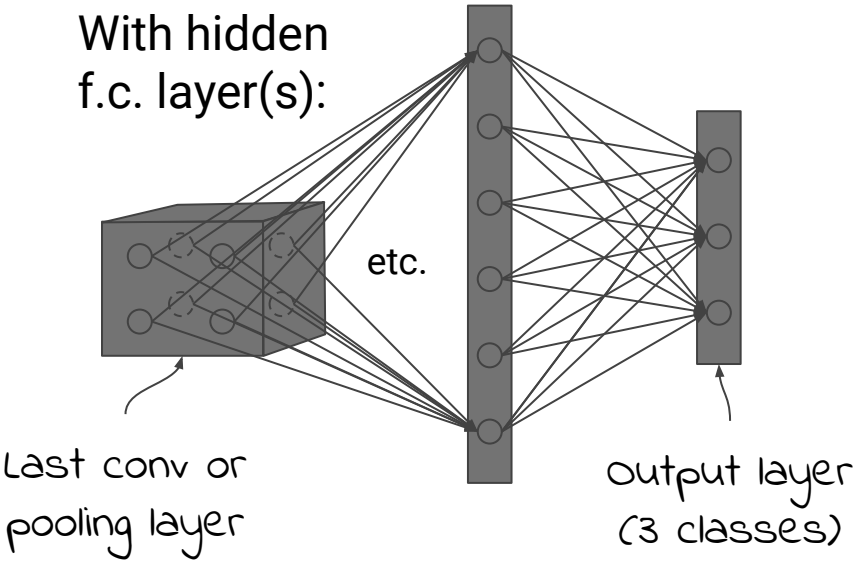
Last layer before softmax

Most classification ConvNets terminate with global pooling + **fully-connected layers**.

With output layer only:



With hidden f.c. layer(s):



Convolutional Neural Networks (CNN)

Simple CNN model used in the tutorial:

```
model = tf.keras.Sequential([
    tf.keras.layers.Reshape(input_shape=(28*28,), target_shape=(28, 28, 1)),
    tf.keras.layers.Conv2D(kernel_size=3, filters=12, activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=6, filters=24, strides=2, activation='relu'),
    tf.keras.layers.Conv2D(kernel_size=6, filters=32, strides=2, activation='relu'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(10, activation='softmax')
])

# this configures the training of the model.
# Keras calls it "compiling" the model.
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# train the model
model.fit(training_dataset, ...)
```

Convolutional Neural Networks (CNN)

Typical use of pre-trained CNN in practice:

```
image_model = tf.keras.applications.InceptionV3(include_top=False,
                                                 weights='imagenet')
new_input = image_model.input
hidden_layer = image_model.layers[-1].output

image_features_extract_model = tf.keras.Model(new_input, hidden_layer)
```

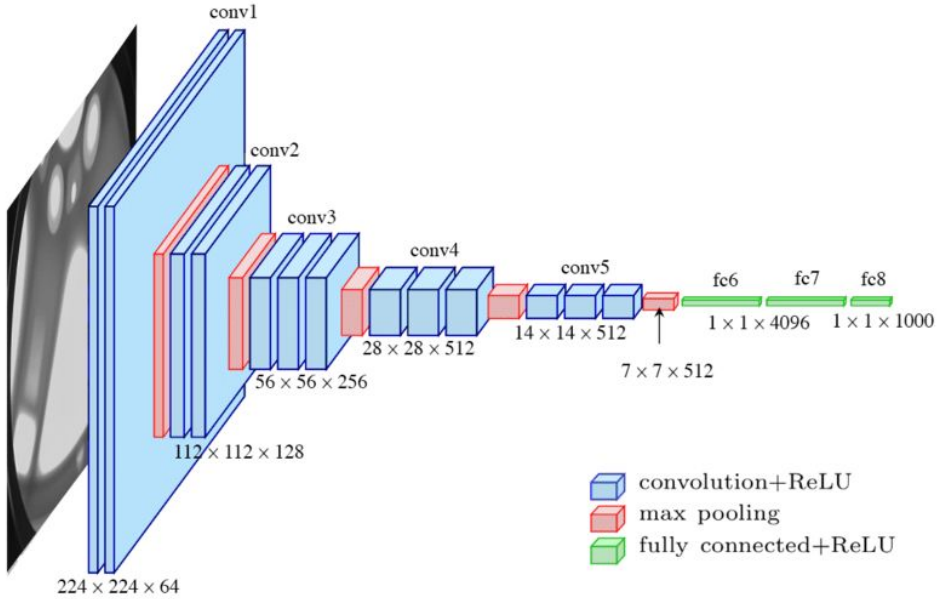
Convolutional Neural Networks (CNN)

Typical use of pre-trained CNN in practice:

```
image_model = tf.keras.applications.InceptionV3(include_top=False,
                                                 weights='imagenet')

new_input = image_model.input
hidden_layer = image_model.layers[-1].output

image_features_extract_model = tf.keras.Model(new_input, hidden_layer)
```

Inception?

VGG [Simonyan&Zisserman, ICLR'15]



Bigger VGG ?

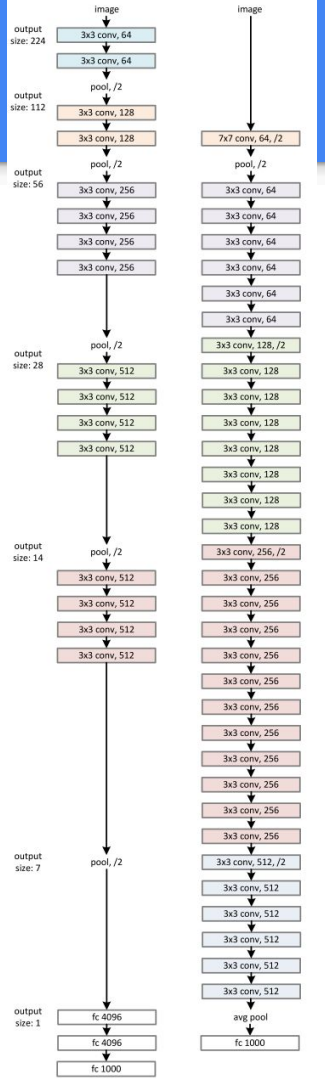
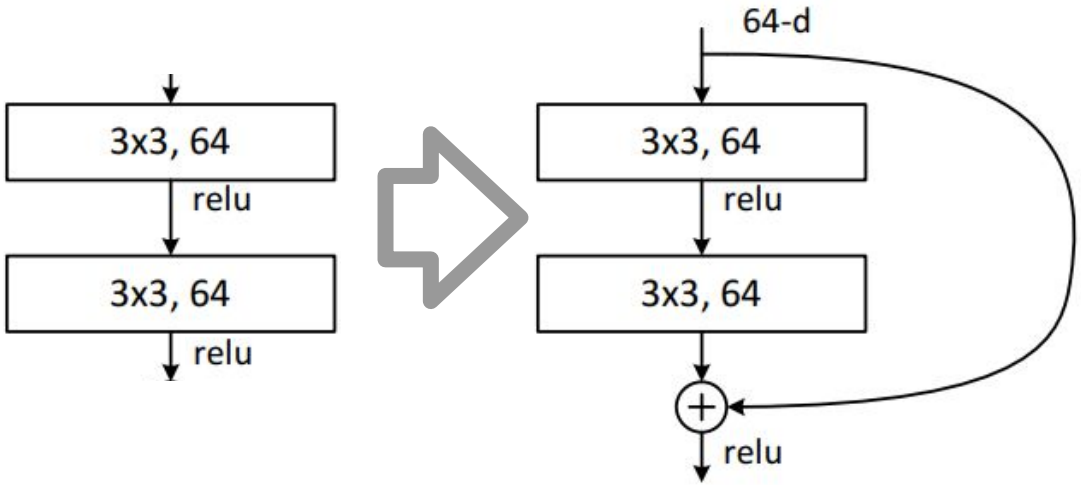
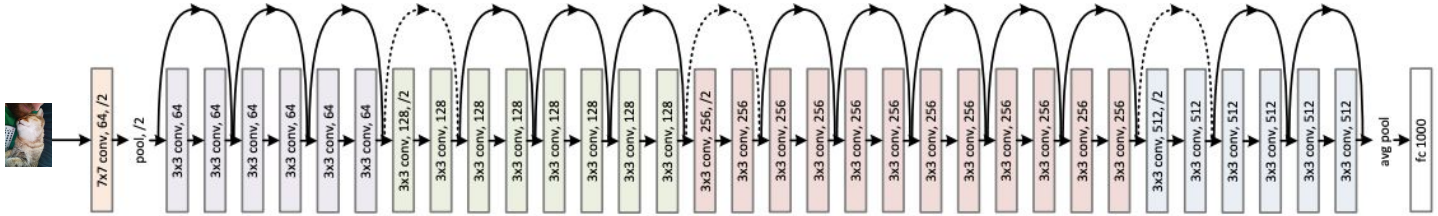


image
output size: 224
3x3 conv, 64
3x3 conv, 64
pool, /2
output size: 112
3x3 conv, 128
3x3 conv, 128
pool, /2
output size: 56
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
output size: 28
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 14
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 7
fc 4096
fc 4096
fc 1000
output size: 1
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000

One new trick introduced in 2015



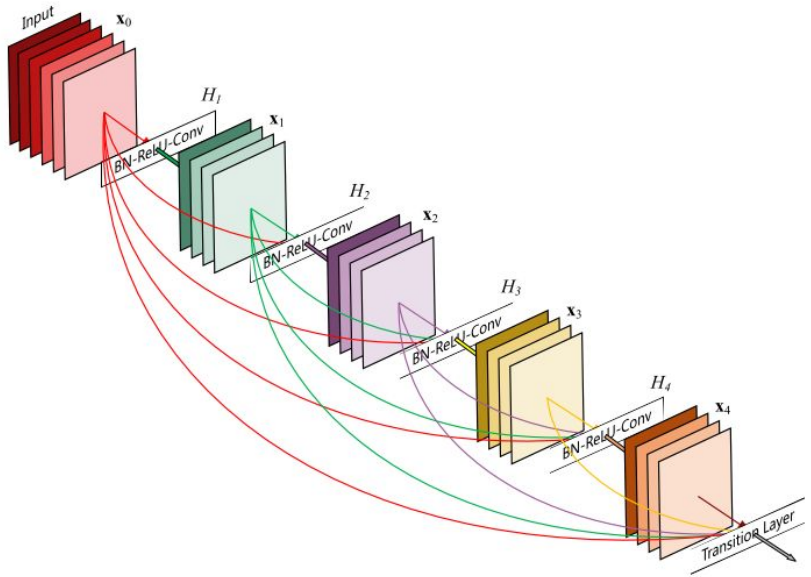
ResNet

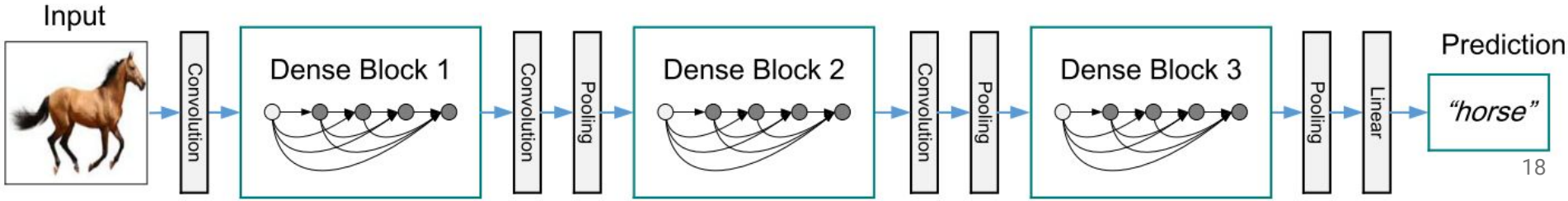


- Residual connections allow to train 20~1k layers
- Resnet design is elegant, regular, and simplistic
- Family of restnets:
  ResNet V1: 50, 101, 152.
  ResNet V2: 50, 101, 152, 200, 1001(!).

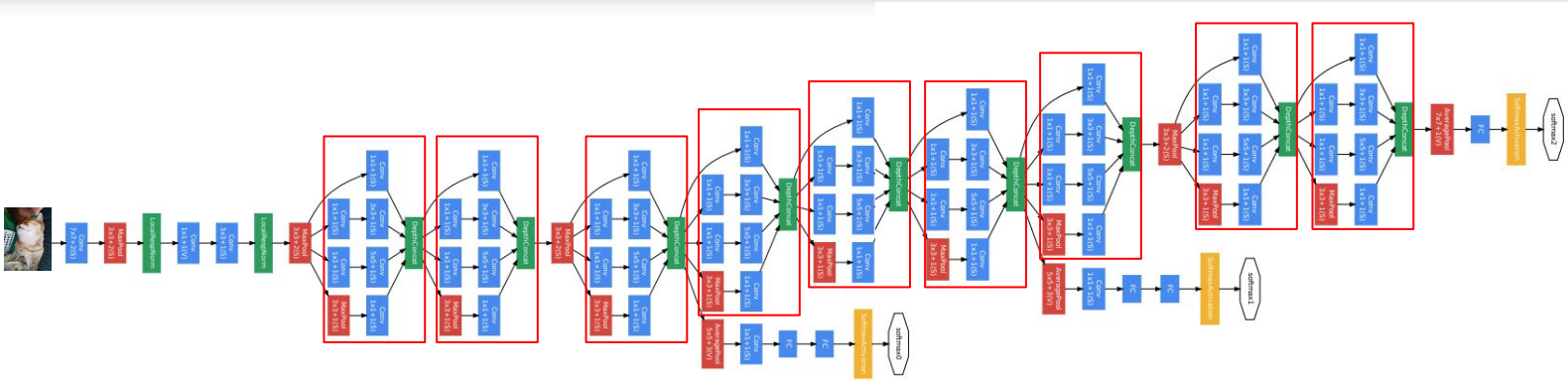
[He et al., 2015/2016]

Post-Resnet: everything goes!





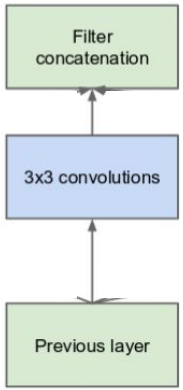
Inception V1, V2, V3, V4 [Szegedy et al., ’15/’16]



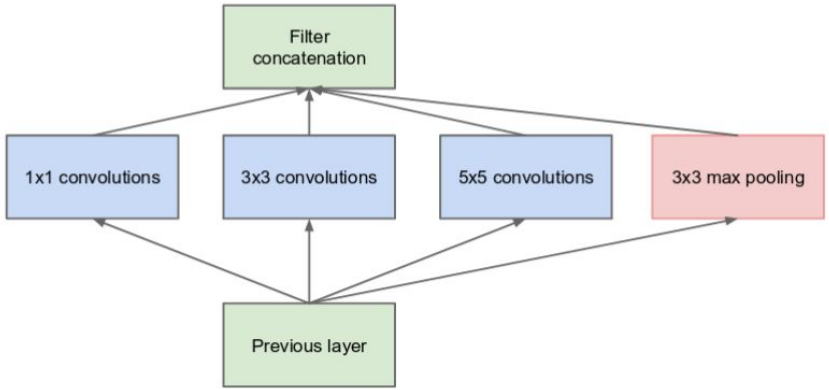
- Convolutions replaced by “Inception cells”.
- Try multiple sizes of filters; let network learn tweaks!
- Use 1x1 convolutions to reduce dimension (~ embed).

```
image_model = tf.keras.applications.InceptionV3(include_top=False,
                                                 weights='imagenet')

new_input = image_model.input
hidden_layer = image_model.layers[-1].output

image_features_extract_model = tf.keras.Model(new_input, hidden_layer)
```

Inception V1, V2, V3, V4 [Szegedy et al., ’15/’16]

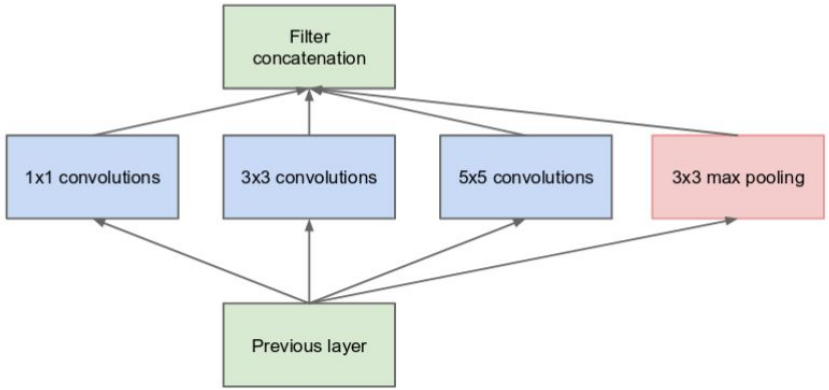


Inception V1, V2, V3, V4 [Szegedy et al., ’15/’16]

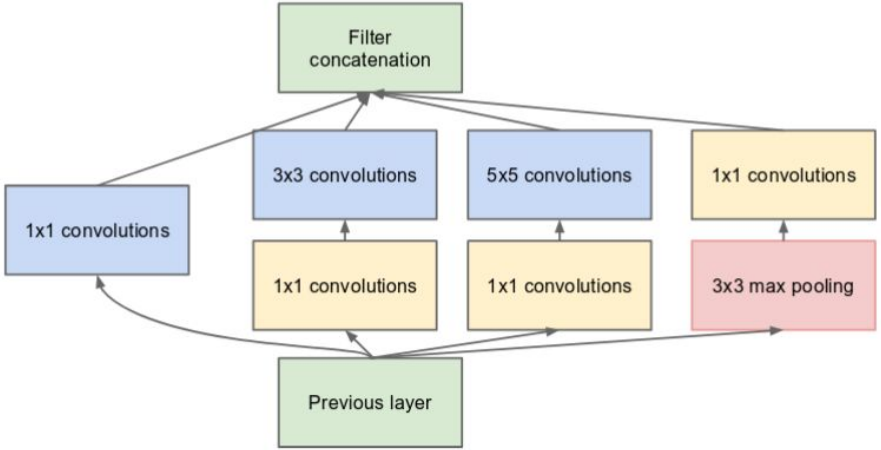


(a) Inception module, naïve version

Inception V1, V2, V3, V4 [Szegedy et al., ’15/’16]



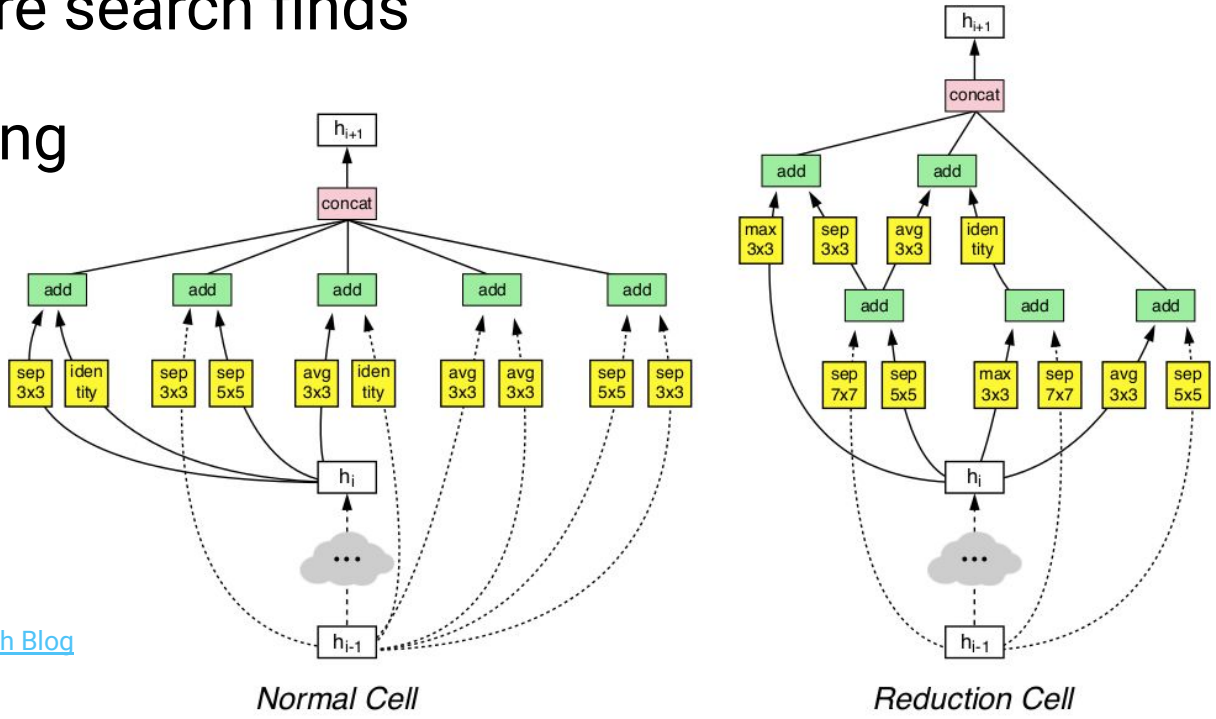
(a) Inception module, naïve version



(b) Inception module with dimension reductions

NASNet [Zoph et al., '17]

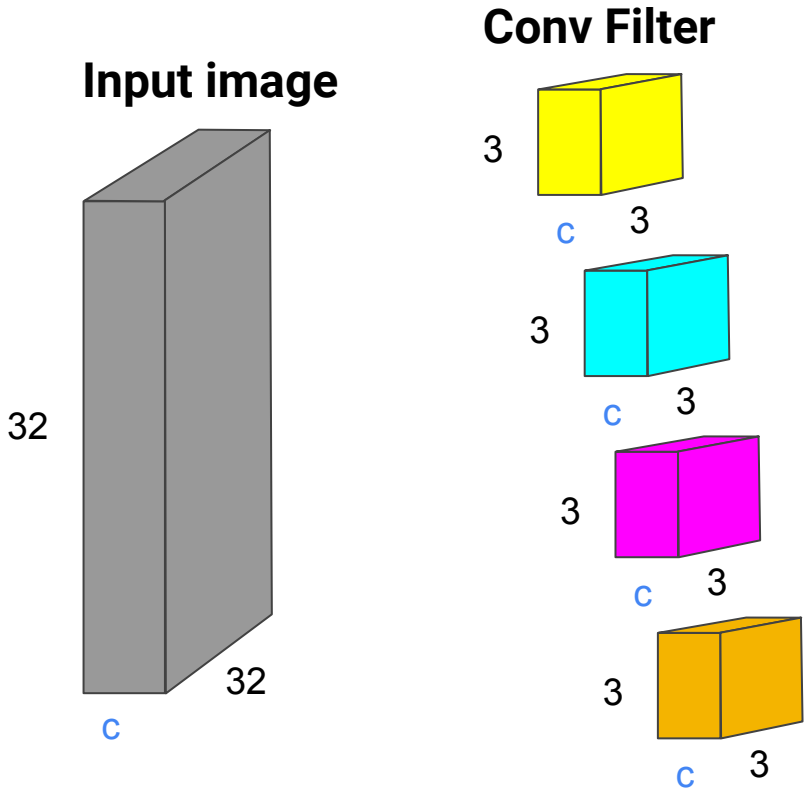
Neural architecture search finds the best-performing cell architecture.



Source: Google Research Blog

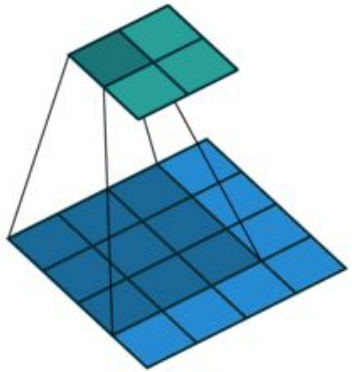
Research at Google

Reminder: **Convolution Example**



Convolve (i.e. slide) this 3D filter (or “kernel”) across the image, taking linear combinations at all locations.

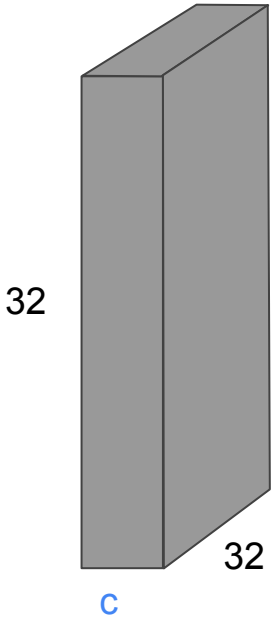
Note that the depth of the filter must match the depth of the input layer.



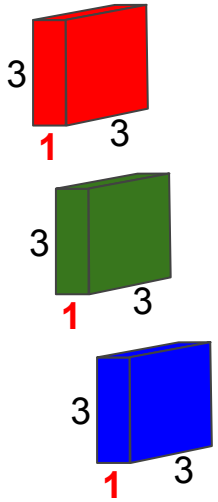
Depthwise separable convolution

...pioneered by **Xception** [Chollet, '17]

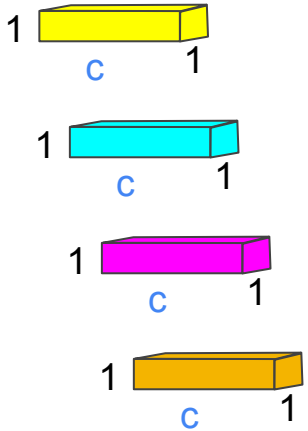
Input features



Depthwise Filters
applied per channel

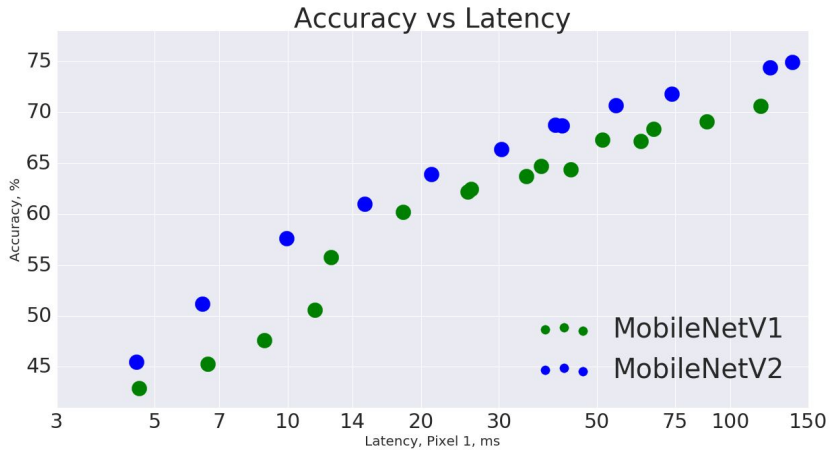


Pointwise Filters
applied per point
(1x1 conv)



MobileNet V1, V2 [Howard&al., Sandler&al., 2018]

- Think: “Inception squeezed down for mobile devices”.
- Uses depthwise separable convolutions.
- Tunable tradeoff of accuracy vs feature depth and image size



Source: Google Research Blog

Take a quiz!