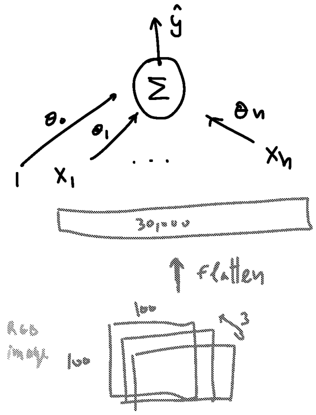
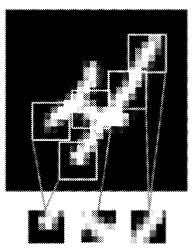
Flattening an image

- Problems:
  - Numerous coefficients for each unit
  - Loss of spatial context
  - Loss of shift invariance (same object at different locations)



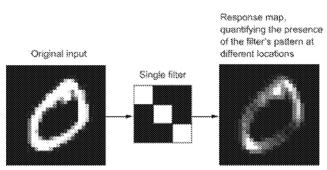
Convolution layers

- Solution:
  - Convolve image with small filters (e.g. 3x3 or 5x5)
  - Share weights of filter between locations (shift invariance)
  - Sharing weights effectively increases data



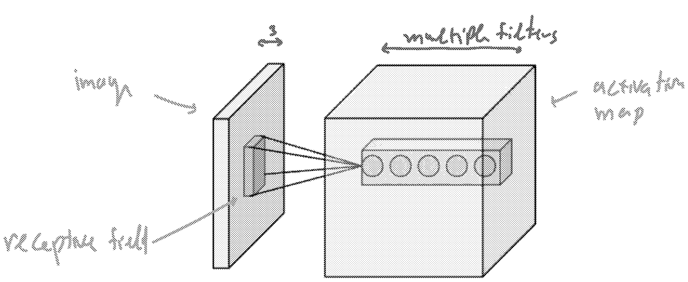
3x3 conv filter → $\overbrace{(3\times 3+1)}^{\text{filter size}}$ coefficients

FC layer → $(\underbrace{100\times 100}_{\text{image size}}+1)$ coefficients

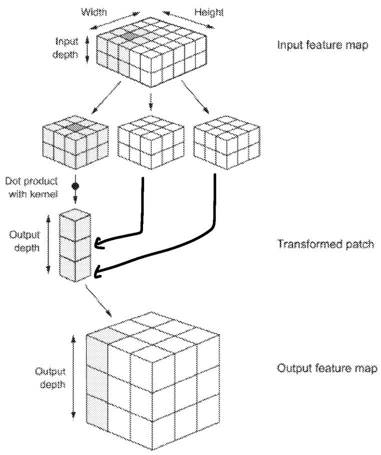


Convolution layers

- Extract image patches (windows)
- Vectorize image window and filter (dot product + bias)
- Filter extends full depth of image
- Multiple convolution filters per location (e.g. oriented edges)
- Use stride to move filter and so activation map maybe smaller



Convolution layers



5×5×2 image

3×(3×3×2) convolution filters

convolution output at one location

Assembled outputs (no padding) 3×3×3

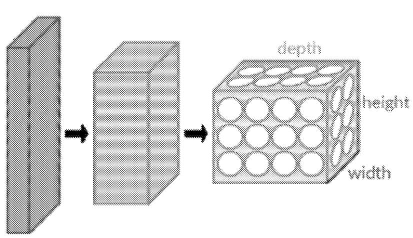
Convolution layers

(batch size, img height, img width, img channels) → convolution layer with 10 filters → (batch size, height, width, channels)

e.g. (128, 28, 28, 1) → $(128, \underbrace{28, 28}_{\text{assume zero padding}}, 10)$

Convolution layers

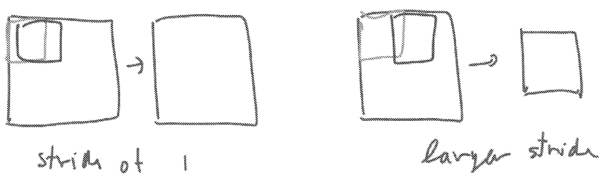
- **Multiple layers: spatial dimensions decrease and depth increases to compensate for reduced coefficients (keep the same number of coefficients)**
- **Nonlinear activation and sampling between layers**
- **Final FC layers perform classification after feature extraction**



multiple scale analysis with higher complexity

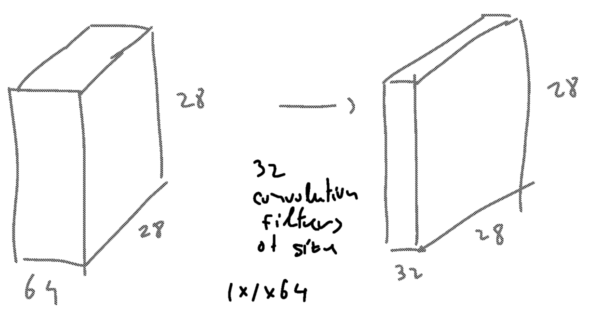
Convolution layers

- **Layer dimensions:**
  - **Zero padding is needed to prevent shrinkage, especially with deep networks.**
  - **The width and height of the output of a layer depend on stride. With a stride of 1 there is large overlap of receptive fields and large output dimension.**



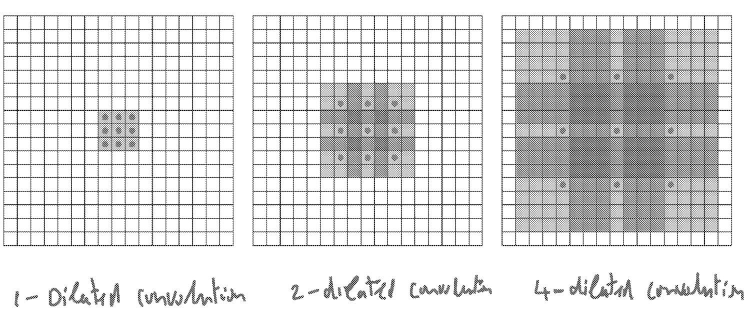
Convolution layers

- **1×1 convolution is used to reduce dimensions (i.e. lower the number of channels)**



Dilated convolution

- **A.k.a. Atrous convolution**
- **Increase receptive field without increasing the number of parameters**
- **Take a weighted sum at the red dot locations**



Dialated convolution

* Instead of ordinary convolution (1D):

$$(I * k)(t) = \sum_{\tau} I(t-\tau)\,k(\tau)$$

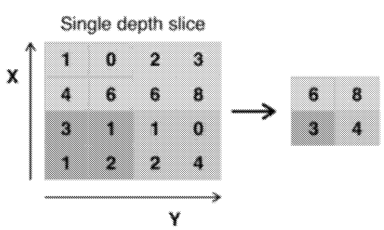
Use:

$$(I *_{\ell} k)(t) = \sum_{\tau} I(t-\ell\tau)\,k(\tau)$$

take steps of size ℓ in image when performing the convolution

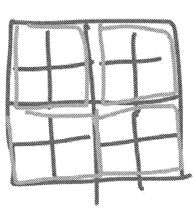
Pooling

- Pooling = down sampling spatial dimensions (depth unchanged)
- Max pooling: partition non overlapping regions and choose max in each region
- Using 2 x 2 regions reduces the layer dimensions by 75%
- Alternatives:
  - Average pooling
  - L2 norm pooling
  - ROI pooling (output size is fixed and input size is variable)



Pooling

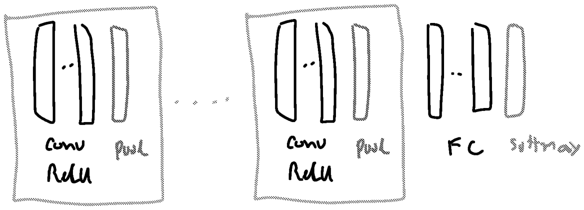
- Pooling is done by scanning with a filter with stride
- Stride is normally selected without filter overlap (downsampling). For example a 2 x 2 filter with a stride of 2
- Convolution with a stride can also be used to downsample but this will average and so we normally downsample with pooling instead of convolution
- Pooling retains more information (e.g. indicates if a feature is there or not)



Pooling

- Pulling is a layer without parameters and has no learning
- The depth dimension is normally not pooled
- Pulling supports multiple scale analysis (a 3x3 window in a pooled layer covers a larger area in the layer before it)
- Pulling helps in reducing the number of network coefficients
- The amount of pooling is a design choice (hyperparameters)

Example network:



Keras MNIST example

```
from keras import layers
from keras import models
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.summary()
```

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.summary()
```

Convolution parameters

```
Conv2D(output_depth, (window_height, window_width))
```

strides = 1, padding = 'valid' ('valid': no padding, 'same': yes padding)
dilation_rate = 1

Keras MNIST example

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
maxpooling2d_1 (MaxPooling2D) (*)	(None, 13, 13, 32)	0
conv2d_2 (Conv2D) (**)	(None, 11, 11, 64)	18496
maxpooling2d_2 (MaxPooling2D) (*)	(None, 5, 5, 64)	0
conv2d_3 (Conv2D) (**)	(None, 3, 3, 64)	36928
flatten_1 (Flatten)	(None, 576)	0
dense_1 (Dense)	(None, 64)	36928
dense_2 (Dense)	(None, 10)	650

Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0

32 3×3×1 filters (32×(3×3×1+1))

64 3×3×32 filters (64×(3×3×32+1))

64 3×3×64 filters (64×(3×3×64+1))

3×3×64

64 units with 576 inputs (64×(576+1))

10 units with 64 inputs (10×(64+1))

(*) Downsample
(**) Reduction by 2 due to no zero padding

Keras MNIST example

* Train on MNIST:

```
from keras.datasets import mnist
from keras.utils import to_categorical

# load data
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# compile model, fit, and evaluate
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)
test_loss, test_acc = model.evaluate(test_images, test_labels)

test_acc               # 0.99080000000000001
```

Convnets improved the 97.8% accuracy of a fully connected network to 99.1%

Keras cats/dogs example

- **Cats and dogs classification (Kaggle challenge)**
- **We use a subset of 2000 cats + 2000 dogs**
- **Larger image size (150 x 150) compared with MNIST (28 x 28) and hence there's a need for a deeper network**

```
from keras import layers
from keras import models
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.summary()
```

Keras cats/dogs example

```
Layer (type)                     Output Shape          Param #
================================================================
conv2d_1 (Conv2D)                (None, 148, 148, 32)  896
________________________________________________________________
maxpooling2d_1 (MaxPooling2D)    (None, 74, 74, 32)    0
________________________________________________________________
conv2d_2 (Conv2D)                (None, 72, 72, 64)    18496
________________________________________________________________
maxpooling2d_2 (MaxPooling2D)    (None, 36, 36, 64)    0
________________________________________________________________
conv2d_3 (Conv2D)                (None, 34, 34, 128)   73856
________________________________________________________________
maxpooling2d_3 (MaxPooling2D)    (None, 17, 17, 128)   0
________________________________________________________________
conv2d_4 (Conv2D)                (None, 15, 15, 128)   147584
________________________________________________________________
maxpooling2d_4 (MaxPooling2D)    (None, 7, 7, 128)     0
________________________________________________________________
flatten_1 (Flatten)              (None, 6272)          0
________________________________________________________________
dense_1 (Dense)                  (None, 512)           3211776
________________________________________________________________

dense_2 (Dense)                  (None, 1)             513
================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

Keras cats/dogs example

* Compile network:

```
from keras import optimizers
model.compile(
    loss='binary_crossentropy',
    optimizer=optimizers.RMSprop(lr=1e-4),
    metrics=['acc'])
```

* Pre-process data:
  - load and rescale to [0., 1] using a keras data generator

Keras cats/dogs example

* pre-process data using ImageDataGenerator :

```
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    train_dir,              # image folder
    target_size=(150, 150),     # resize images
    batch_size=20,
    class_mode='binary')
validation_generator = test_datagen.flow_from_directory(
    validation_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary')
```

train — cats, dogs
validation — cats, dogs
test — cats, dogs

- Generate 20×150×150×3 tensors for training and validation and corresponding binary labels

Keras cats/dogs example

* Fit model:
  use 'fit_generator' instead of 'fit'
  - The # of draws of size 'batch_size' to complete a complete epoch is:

$$\text{steps\_per\_epoch} = \frac{\#\text{ examples}}{\text{batch\_size}} = \frac{2000}{20} = 100$$

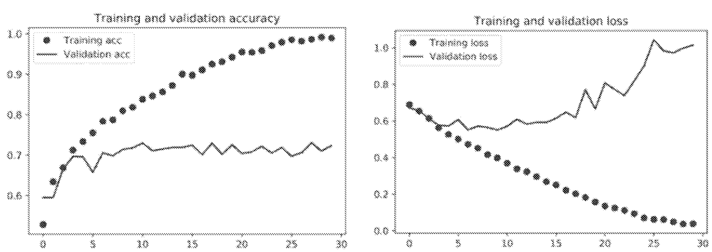
```
history = model.fit_generator(
    train_generator,
    steps_per_epoch=100,
    epochs=30,
    validation_data=validation_generator,
    validation_steps=50)
model.save('cats_and_dogs_small_1.h5')
```

Keras cats/dogs example

* plot results:

```
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

# Plot accuracy
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

# Plot loss
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

Keras cats/dogs example



- overfit after 5 iterations due to small dataset
67% validation accuracy

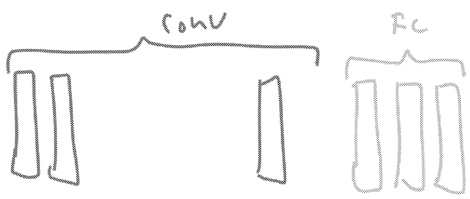
Keras cats/dogs example

* To further prevent overfitting, add dropout:

```
# define a new convnet that includes dropout
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

Keras cats/dogs example

- **To address the problem with limited data:**
  1. **Data augmentation: add examples with perturbations (e.g., rotation, flip, contrast change)**
  2. **Transfer learning: use a pre-trained convolution base**
- **Freeze loaded weights of pre-trained blocks after loading them**

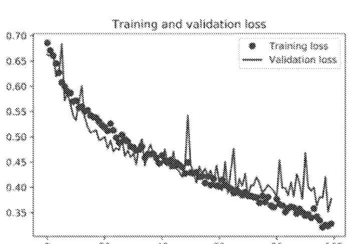


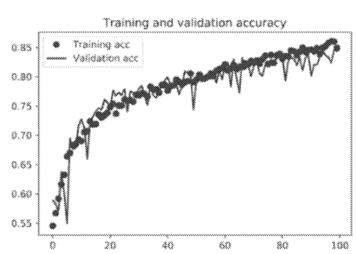
Data augmentation

```
train_datagen = ImageDataGenerator(                    # different from before
rescale=1./255,
rotation_range=40,
width_shift_range=0.2,
height_shift_range=0.2,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True,)
test_datagen = ImageDataGenerator(rescale=1./255)      # as before

train_generator = train_datagen.flow_from_directory(   # as before
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary')
validation_generator = test_datagen.flow_from_directory( # as before
    validation_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary')

history = model.fit_generator(                         # as before
    train_generator,
    steps_per_epoch=100,
    epochs=100,
    validation_data=validation_generator,
    validation_steps=50)

model.save('cats_and_dogs_small_2.h5')                 # as before
```

no augmentation during testing or validation

Data augmentation



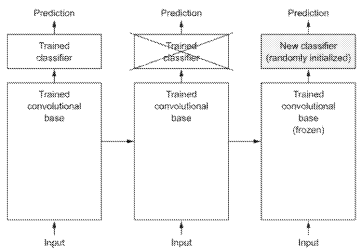


* does not overfit. 82% validation accuracy

* Additional improvement is possible with hyperparameter tuning.

Transfer learning

- Use a pre-trained convnet trained on a large data set (e.g., ImageNet object classification)
- ImageNet: 1.4 million labeled images, 1000 classes (animals, objects)
- Use the convolution layers of VGG16 to extract features for the cat/dog problem (representation learning)
- Whether to use higher convolution layers depends on how similar the data sets are



Transfer learning

* Pre-trained models available in keras:

Models for image classification with weights trained on ImageNet:

- Xception
- VGG16
- VGG19
- ResNet, ResNetV2, ResNeXt
- InceptionV3
- InceptionResNetV2
- MobileNet
- MobileNetV2
- DenseNet
- NASNet

Transfer learning

* Load VGG16

```
from keras.applications import VGG16
conv_base = VGG16(
    weights='imagenet',          # weights checkpoint from which to initialize model
    include_top=False,           # do not include the fully connected layers
                                 # (responsible for classifying 1000 classes)
    input_shape=(150, 150, 3))  # optional
conv_base.summary()
```

Transfer learning

```
>>> conv_base.summary()
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 150, 150, 3)	0
block1_conv1 (Convolution2D)	(None, 150, 150, 64)	1792
block1_conv2 (Convolution2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Convolution2D)	(None, 75, 75, 128)	73856
block2_conv2 (Convolution2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Convolution2D)	(None, 37, 37, 256)	295168
block3_conv2 (Convolution2D)	(None, 37, 37, 256)	590080
block3_conv3 (Convolution2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Convolution2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Convolution2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Convolution2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Convolution2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Convolution2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Convolution2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0

Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

* 14M parameters

* The input feature map is $150 \times 150 \times 3$

* The final feature map is $4 \times 4 \times 512$

Transfer learning

- **Add pre-trained layers to the network:**
  - **Add conv-base (the loaded model) as a layer**
  - **Freeze weights of pre-trained network**
  - **Train end-to-end**
- **Larger and slower network**

```
from keras import models
from keras import layers
model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.summary()
```

Transfer learning

Layer (type)	Output Shape	Param #
vgg16 (Model)	(None, 4, 4, 512)	14714688
flatten_1 (Flatten)	(None, 8192)	0
dense_1 (Dense)	(None, 256)	2097408
dense_2 (Dense)	(None, 1)	257

Total params: 16,812,353
Trainable params: 16,812,353
Non-trainable params: 0

Transfer learning

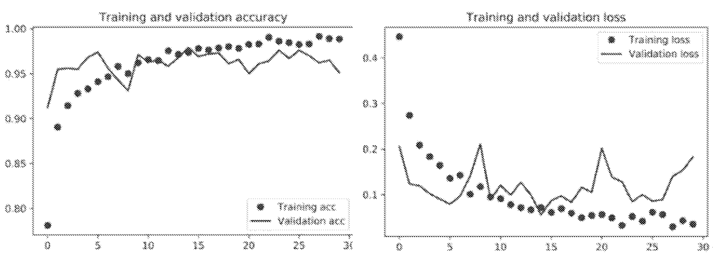
- **Freeze the loaded model so as to not to destroy the weights by gradients from untrained fully connected layers on top**

```
conv_base.trainable = False
```

```
print('Number of weight tensors ' 'before freezing the conv base:',
      len(model.trainable_weights))
#   Number of weight tensors before freezing the conv base: 30

conv_base.trainable = False ####
print('Number of weight tensors ' 'after freezing the conv base:',
      len(model.trainable_weights))
#   Number of weight tensors after freezing the conv base: 4
```

- **There are two weight tensors per layer: weight matrix, bias vector**

Transfer learning



- **Validation accuracy of 96% with a very small data set**
- **Less overfitting (due to data augmentation)**

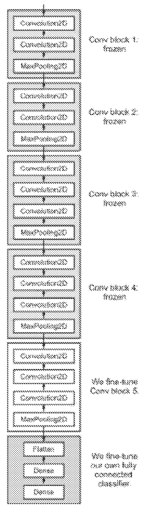
Transfer learning

- **Fine tuning:**
  - **After training the fully connected layers, unfreeze some top layers in the conv-base and retrain to allow the model to fit the data**
- **Steps:**
  1. **Add custom network on top of the trained layers**
  2. **Freeze the trained layers**
  3. **Train the custom network**
  4. **Unfreeze the top layers in the base network**
  5. **Jointly train the custom network and unfrozen layers**

Transfer learning

```
conv_base.summary()
```

```
Layer (type)                     Output Shape          Param #
================================================================
input_1 (InputLayer)             (None, 150, 150, 3)   0
________________________________________________________________
block1_conv1 (Convolution2D)     (None, 150, 150, 64)  1792
________________________________________________________________
block1_conv2 (Convolution2D)     (None, 150, 150, 64)  36928
________________________________________________________________
block1_pool (MaxPooling2D)       (None, 75, 75, 64)    0
________________________________________________________________
block2_conv1 (Convolution2D)     (None, 75, 75, 128)   73856
________________________________________________________________
block2_conv2 (Convolution2D)     (None, 75, 75, 128)   147584
________________________________________________________________
block2_pool (MaxPooling2D)       (None, 37, 37, 128)   0
________________________________________________________________
block3_conv1 (Convolution2D)     (None, 37, 37, 256)   295168
________________________________________________________________
block3_conv2 (Convolution2D)     (None, 37, 37, 256)   590080
________________________________________________________________
block3_conv3 (Convolution2D)     (None, 37, 37, 256)   590080
________________________________________________________________
block3_pool (MaxPooling2D)       (None, 18, 18, 256)   0
________________________________________________________________
block4_conv1 (Convolution2D)     (None, 18, 18, 512)   1180160
________________________________________________________________
block4_conv2 (Convolution2D)     (None, 18, 18, 512)   2359808
________________________________________________________________
block4_conv3 (Convolution2D)     (None, 18, 18, 512)   2359808
________________________________________________________________
block4_pool (MaxPooling2D)       (None, 9, 9, 512)     0

________________________________________________________________
block5_conv1 (Convolution2D)     (None, 9, 9, 512)     2359808
________________________________________________________________
block5_conv2 (Convolution2D)     (None, 9, 9, 512)     2359808
________________________________________________________________
block5_conv3 (Convolution2D)     (None, 9, 9, 512)     2359808
________________________________________________________________
block5_pool (MaxPooling2D)       (None, 4, 4, 512)     0
================================================================
Total params: 14714688
```

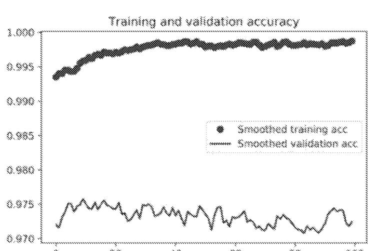


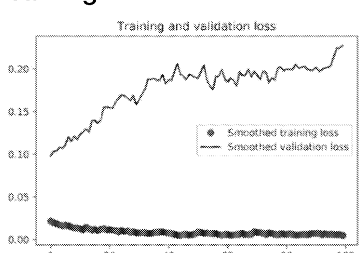
Transfer learning

- **Freezing all layers up to block number 5:**
  - **Step through all the layers**
  - **Set "trainable" starting with block5_conv1**

```
conv_base.trainable = True
set_trainable = False
for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

Transfer learning





* smoothed curves: 97% accuracy (1% improvement)
* Final test accuracy:

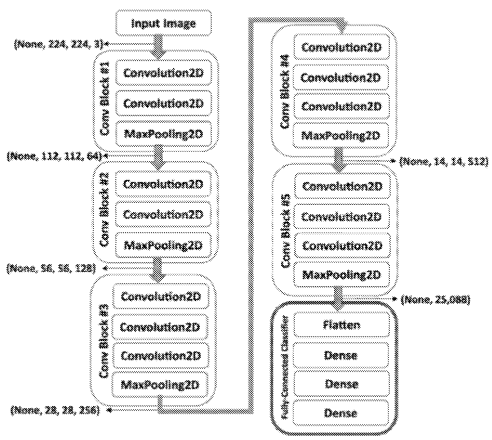
```
test_loss, test_acc = model.evaluate_generator(
    test_generator,
    steps=50)

print('test acc:', test_acc) #  97%
```

Gradual training

- Given a deep network:
  - Train of shallow network
  - Freeze the trained layers
  - Add layers
  - Retrain
  - Unfreeze and retrain all layers
- Advantage: constrain the search space and so converge better
- Disadvantage: because of limited solution space possibly larger error
- For example this procedure was used in VGG16 (but is less common now)

CNN architectures

x VGG16 (2014) : 16 layers



CNN architectures

x VGG 19

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

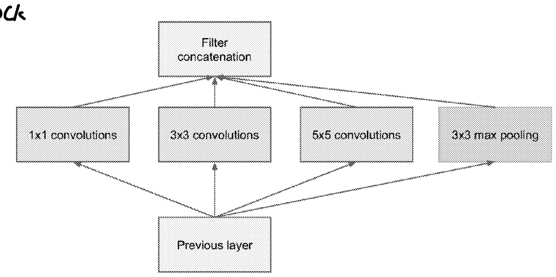
Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

CNN architectures

* GoogLeNet / Inception (2014): 22 layers + fewer parameters (5M)

* Inception block



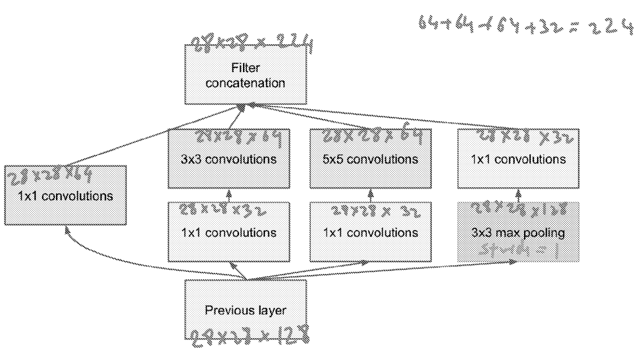
(a) Inception module, naïve version

* multiple receptive fields concatenated ⇒ dimension increase

CNN architectures

- **To reduce the number of parameters use a modified inception block that reduces parameters by using a 1x1 convolution to reduce the number of channels**

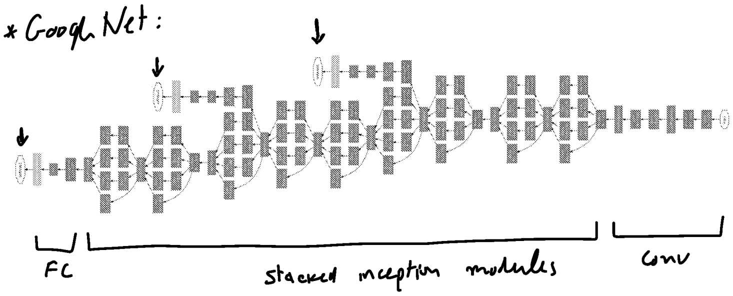
64+64+64+32=224



(b) Inception module with dimensionality reduction

CNN architectures

* GoogLeNet:



- **A single fully connected layer results in less parameters**
- **Convergence may be difficult and so use auxiliary classifier outputs to help with vanishing gradients (not needed when using batch normalization)**

CNN architectures

* using residual blocks:

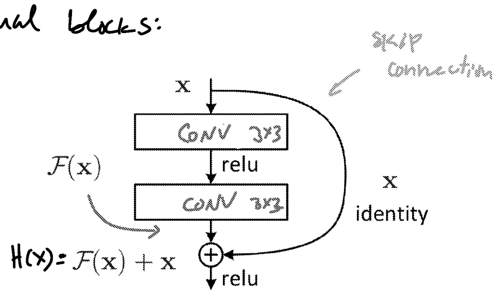


Figure 2. Residual learning: a building block.

- **It is easier to learn the F(x) residual compared with H(x) because we need to learn deviation from identity instead of a function**
- **Skip connections help with vanishing gradients**

CNN architectures

VGG 19: 19.6 B FLOPs
Plain 34: 3.6 B FLOPs
ResNet 34: 3.6 B FLOPs

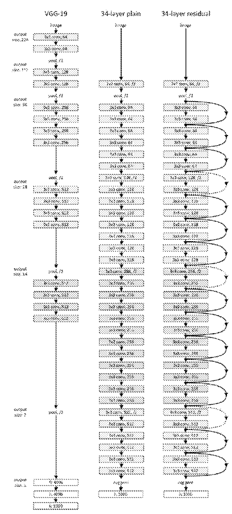


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

MobileNets

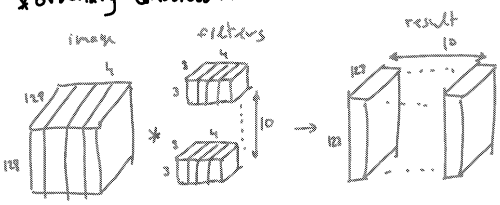
- **MobileNets:**
  - **For mobile and embedded devices**
  - **Trade-off between latency and accuracy**
  - **Use depth-wise separable convolutions**

Separable convolution. Separate convolution into:
  - **Depth-wise (channel) convolution followed by**
  - **Point-wise 1x1 convolution**

Simplification parameters:
- **Width multiplier (fewer channels)**
- **Resolution multiplier (lower resolution)**

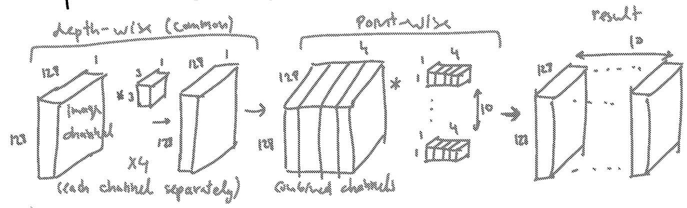
MobileNets

* ordinary convolution:



$128 \times 128 \times 3 \times 3 \times 4 \times 10$
$= 5{,}898{,}240$ ops

* separable convolution:



$4 \times 128 \times 128 \times 3 \times 3 =$
$589{,}824$ ops

+

$128 \times 128 \times 4 \times 10 =$
$655{,}360$ ops

$= 1{,}245{,}184$ ops

MobileNets

* ordinary convolution:

$$\overset{\text{input}}{D_f \times D_f \times M} \rightarrow \overset{\text{filters}}{(N \times D_k \times D_k \times M)} \rightarrow \overset{\text{output}}{D_f \times D_f \times N}$$

$$(D_k \times D_k \times M \times N) \times (D_f \times D_f) \text{ ops}$$

* separable convolution:

$$\overset{\text{input}}{D_f \times D_f \times M} \rightarrow \overset{\text{depth-wise}}{(M \times D_k \times D_k \times 1)} \rightarrow \overset{\text{Depth-wise output}}{D_f \times D_f \times M}$$

$$\rightarrow \overset{\text{point-wise}}{(N \times 1 \times 1 \times M)} \rightarrow \overset{\text{output}}{D_f \times D_f \times N}$$

$$(D_k \times D_k \times 1 \times M) \times (D_f \times D_f) + (1 \times 1 \times M \times N) \times (D_f \times D_f) \text{ ops.}$$

MobileNets

* Reduction in computation:

$$\frac{\text{separable depth-wise conv ops}}{\text{ordinary conv ops}} =$$

$$= \frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F}$$

$$= \frac{1}{N} + \frac{1}{D_K^2}$$

MobileNets

* Implementation with Batch Normalization.

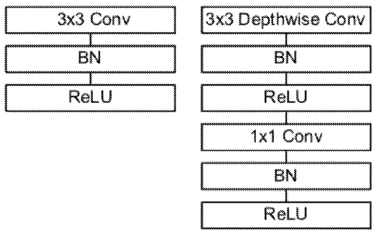


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

MobileNets

* width multiplier:
  - Reduce # of input and output channels to each layer by a factor of α.

    input channels: $M \rightarrow \alpha M$

    output channels: $N \rightarrow \alpha N$

    $\alpha \in [0,1]$ e.g. 0.5

  - New computational cost:

$$\underbrace{D_K \times D_K \times \alpha M \times D_F \times D_F}_{\text{reduction by factor } \alpha} + \underbrace{\alpha M \times \alpha N \times D_f \times D_f}_{\text{reduction by factor } \alpha^2}$$

MobileNets

* Resolution multiplier:
  - reduce input resolution by a factor $\rho \in [0,1]$

    input resolution: $D_f \times D_f \rightarrow \rho D_f \times \rho D_f$

    output resolution: $D_f \times D_f \rightarrow \rho D_f \times \rho D_f$

  - New computational cost:

$$\underbrace{D_K \times D_K \times \alpha M \times \rho D_F \times \rho D_F + \alpha M \times \alpha N \times \rho D_f \times \rho D_f}_{\text{reduction by factor } \rho^2}$$

MobileNets

* Experimental evaluation

Table 3. Resource usage for modifications to standard convolution. Note that each row is a cumulative effect adding on top of the previous row. This example is for an internal MobileNet layer with $D_K = 3$, $M = 512$, $N = 512$, $D_F = 14$.

Layer/Modification	Million Mult-Adds	Million Parameters
Convolution	462	2.36
Depthwise Separable Conv	52.3	0.27
$\alpha = 0.75$	29.6	0.15
$\rho = 0.714$	15.1	0.15

# of operations

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Depth-wise separable conv.

accuracy

MobileNets

* Experimental evaluation (contd.)

Table 6. MobileNet Width Multiplier

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Table 7. MobileNet Resolution

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2