Artificial intelligence

Deep Learning: CNNs and application to image classification

- Convolutional Neural Networks (CNN)
- pre-trained models and transfer Learning
- **Kaggle competition: Cats vs. Dogs classification**

Carl McBride Ellis
carl.mcbride@u-tad.com

Convolutional Neural Networks (CNN)

A convolutional neural network (CNN) (aka ConvNet)
is a feedforward (sequential) neural network + filters or 'kernels'

Digital image processing

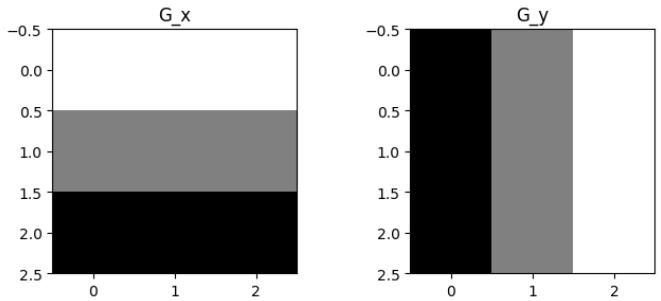
Prewitt edge detector (1970) is a 3×3 'kernel' used to estimate the first derivative

$$G_x = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

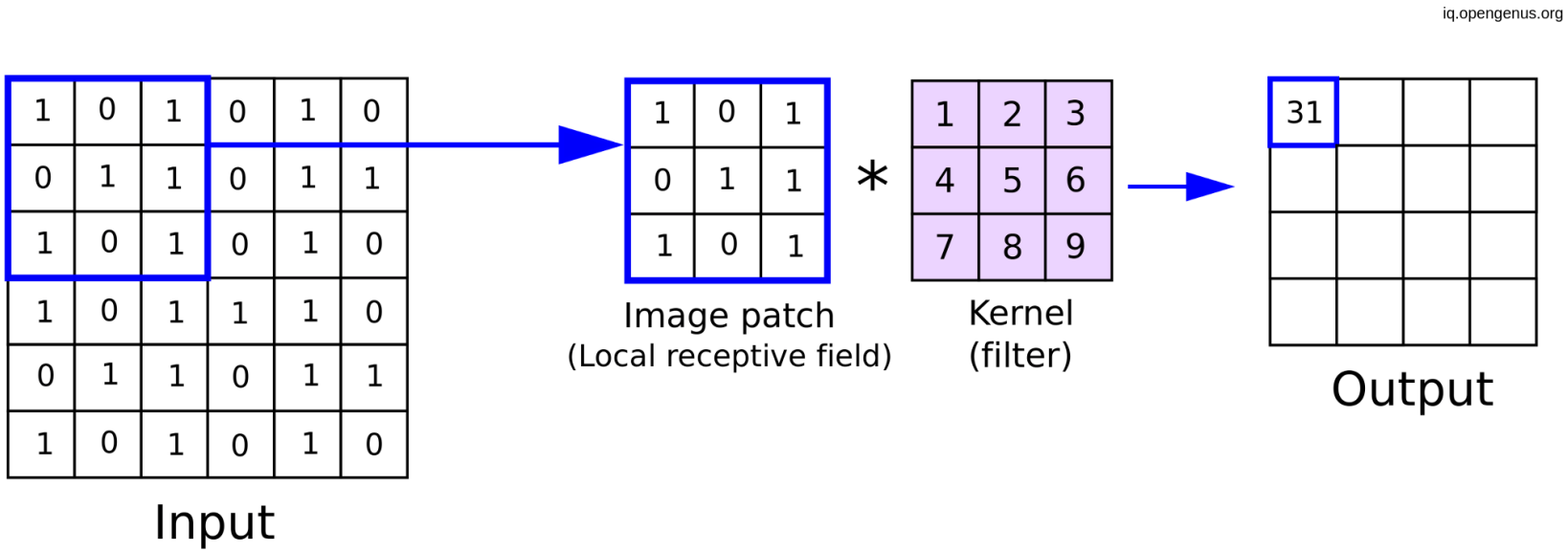
and

$$G_y = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

and as greyscale plots



The convolution operation (written as $*$ or $\otimes$)



Results for Prewitt:

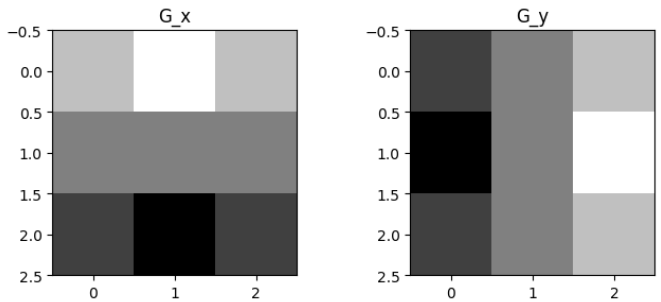
Another edge detector is the Sobel 3×3 'kernel'

$$G_x = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

and

$$G_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

and as greyscale plots



Practical session

Apply the Prewitt G_y kernel by hand to the following image:

0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1

(Notice that we shall loose the 'outer ring' of the original image pixels in the new image.

To avoid this we could apply 'padding' to the original image before convolution.)

Let us try the Prewitt filter ourselves on a couple of images:

notebook_CNN_Prewitt.ipynb

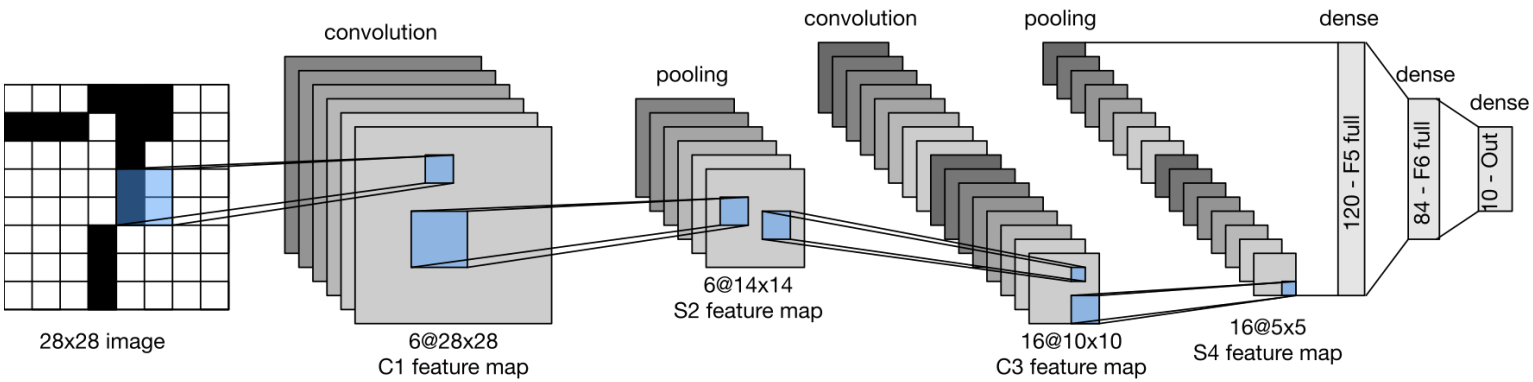
CNN approach:

rather than use specific filters, we now learn the best filters for our problem

Optical character recognition (OCR)

Recognizing digits and letters.

LeNet-1 (1989) LeNet-5 (1998) by Yann LeCun *et. al* for handwritten character recognition



```
model = models.Sequential()
model.add(layers.Conv2D(6, (5, 5), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(16, (5, 5), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())    # go from 2D to 1D
model.add(layers.Dense(120, activation='relu'))
model.add(layers.Dense(84, activation='relu'))
model.add(layers.Dense(n_classes, activation='softmax'))

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

(Source: LeNet-5 - A complete guide)

Image classification

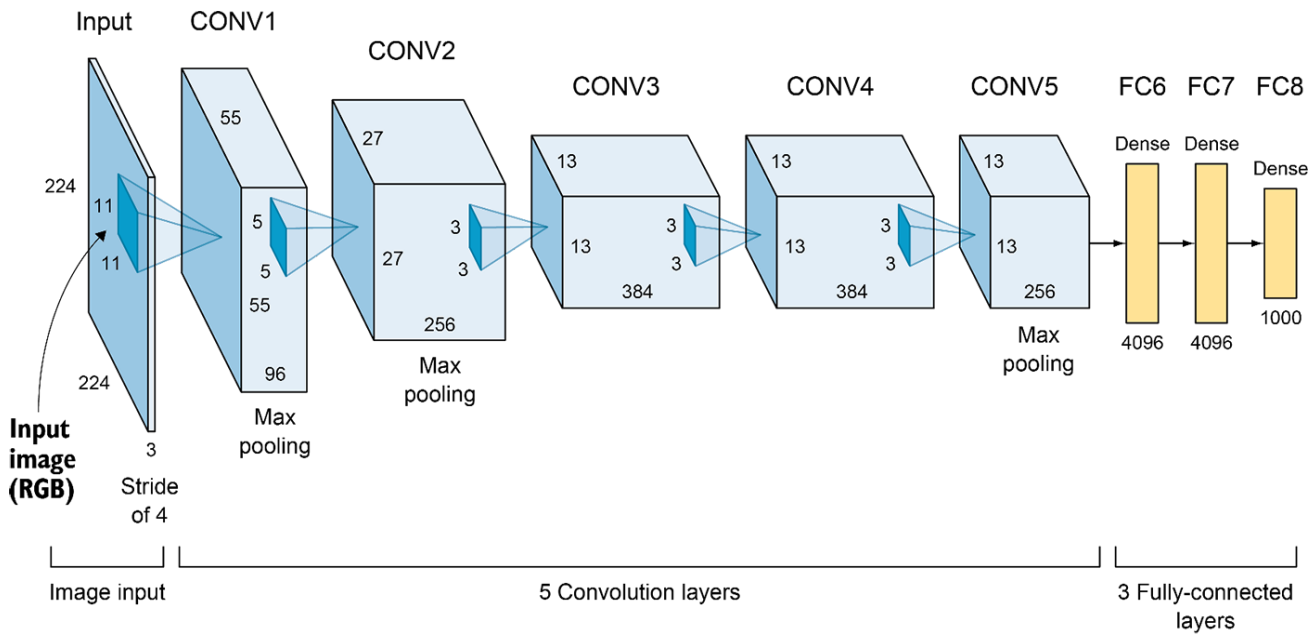
The ImageNet competition (2010-2017)

The ImageNet dataset has over 1.2 million images with 1,000 classes/categories (paper).

In 2012 a CNN called AlexNet scored a huge 10% better than rivals on the ImageNet competition

This work by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton ignited the current AI revolution.

The AlexNet architecture:



AlexNet is a sequential model.

AlexNet as a Keras model:

```
model = keras.models.Sequential([
  keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='relu', input_shape=(128,128,
  keras.layers.BatchNormalization(),
  keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
  keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='relu', padding="same"),
  keras.layers.BatchNormalization(),
  keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
  keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same"),
  keras.layers.BatchNormalization(),
  keras.layers.Conv2D(filters=384, kernel_size=(1,1), strides=(1,1), activation='relu', padding="same"),
  keras.layers.BatchNormalization(),
  keras.layers.Conv2D(filters=256, kernel_size=(1,1), strides=(1,1), activation='relu', padding="same"),
  keras.layers.BatchNormalization(),
  keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
  keras.layers.Flatten(),
  keras.layers.Dense(4096, activation='relu'),
  keras.layers.Dropout(0.5),
  keras.layers.Dense(4096, activation='relu'),
  keras.layers.Dropout(0.5),
  keras.layers.Dense(n_classes, activation='softmax')
])
```

8 October 2024: Geoffrey Hinton wins Nobel Prize

Scientific Background to the Nobel Prize in Physics 2024

"FOR FOUNDATIONAL DISCOVERIES AND INVENTIONS
THAT ENABLE MACHINE LEARNING
WITH ARTIFICIAL NEURAL NETWORKS"

The Nobel Committee for Physics

As well as the layers we saw before for ANN

- `Flatten` - a reshaping layer
- `Dense` - a fully connected layer

we now have these two new layers

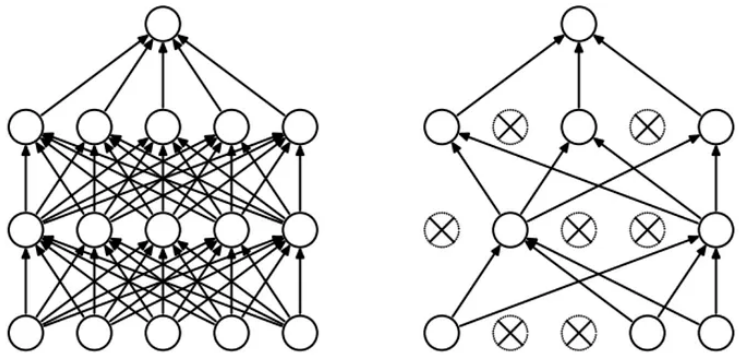
- `BatchNormalization` - a normalization layer
- `Dropout` - a regularization layer

A batch normalization layer can speed up training by re-scaling each (mini)-batch individually

```
layers.BatchNormalization(),
```

(Paper: *"Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift"* (2015))

Dropout layers:



(a) Standard Neural Net (b) After applying dropout.

we can randomly drop out some fraction of a layer's input units. Note that the weights of the remaining neurons are multiplied by the `rate` so as to compensate for the now missing neurons.

```
# apply 30% dropout to the next layer
layers.Dropout(rate=0.3),
```

(Paper: "Dropout: A Simple Way to Prevent Neural Networks from Overfitting" (2014))

and we also now have two new 2-dimensional layers

- `Conv2D` - a convolution layer
- `MaxPool2D` - a pooling layer

The convolutional layer (Conv2D) is what makes a CNN a CNN

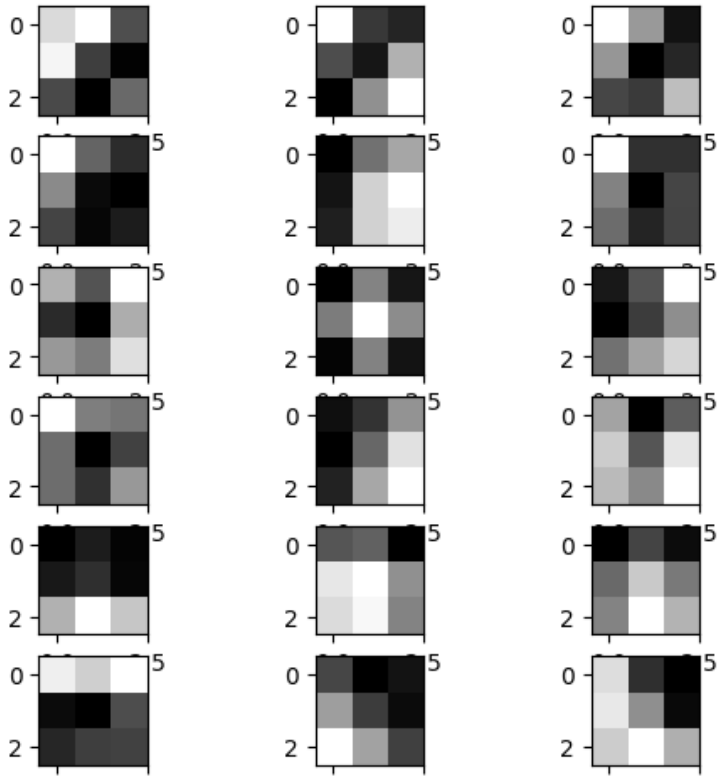
(This is the first Conv2D layer of AlexNet)

```
layers.Conv2D(filters=96,
              kernel_size=(3,3),
              padding="valid",
              strides=(1, 1),
              dilation_rate=(1, 1),),
```

options:

- filters - the number of kernels that the layer will learn
- kernel size - the shape of the filters
- padding
- strides (see animation)
- dilation (see animation)

example of some 3×3 filters learned by a CNN

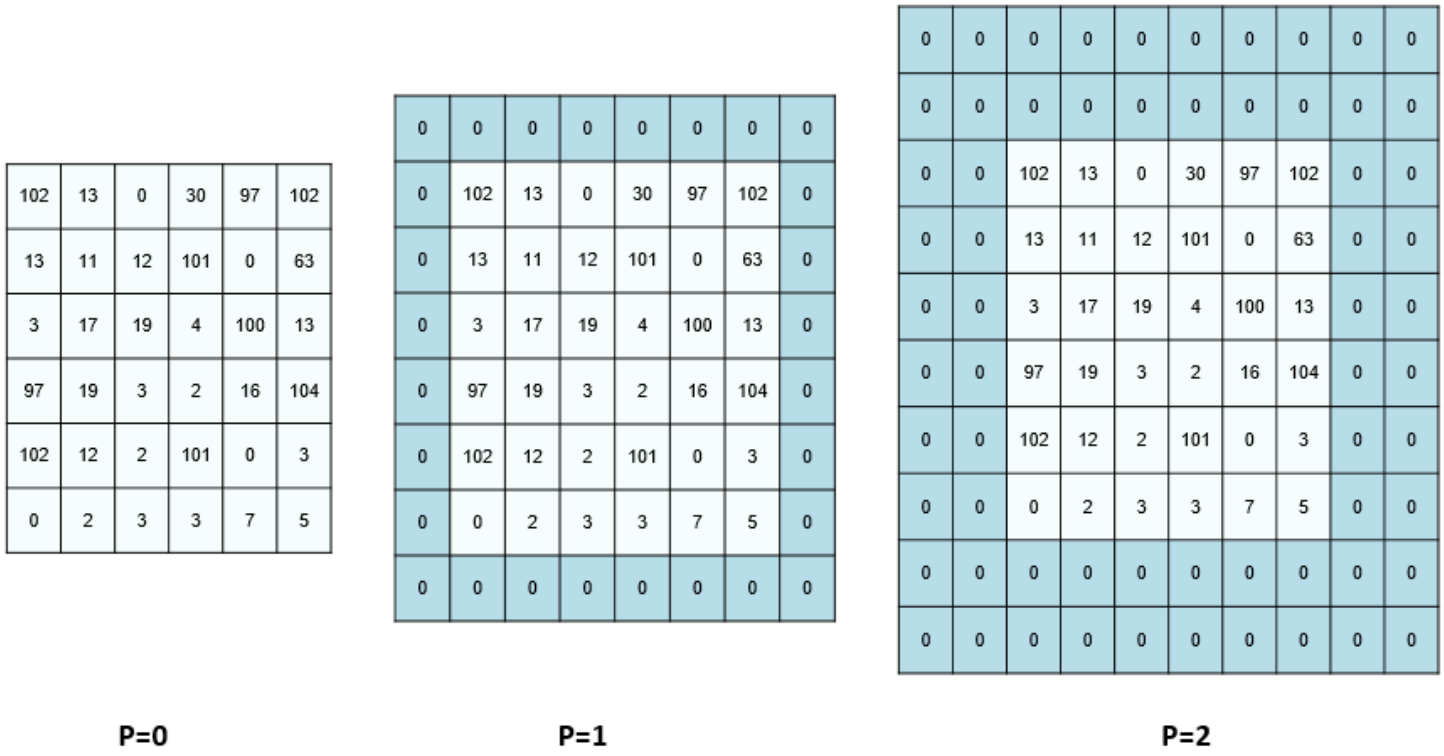


See the 64 filters of the first layer of the VGG-16 model:
notebook_CNN_VGG16_filters.ipynb

we can also vizualize the results of a `Conv2D` operation

`notebook_CNN_VGG16_feature_maps.ipynb`

Padding examples



102	13	0	30	97	102
13	11	12	101	0	63
3	17	19	4	100	13
97	19	3	2	16	104
102	12	2	101	0	3
0	2	3	3	7	5

P=0

0	0	0	0	0	0	0	0
0	102	13	0	30	97	102	0
0	13	11	12	101	0	63	0
0	3	17	19	4	100	13	0
0	97	19	3	2	16	104	0
0	102	12	2	101	0	3	0
0	0	2	3	3	7	5	0
0	0	0	0	0	0	0	0

P=1

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	102	13	0	30	97	102	0	0
0	0	13	11	12	101	0	63	0	0
0	0	3	17	19	4	100	13	0	0
0	0	97	19	3	2	16	104	0	0
0	0	102	12	2	101	0	3	0	0
0	0	0	2	3	3	7	5	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

P=2

for example

```
model.add(ZeroPadding2D((1,1)))
```

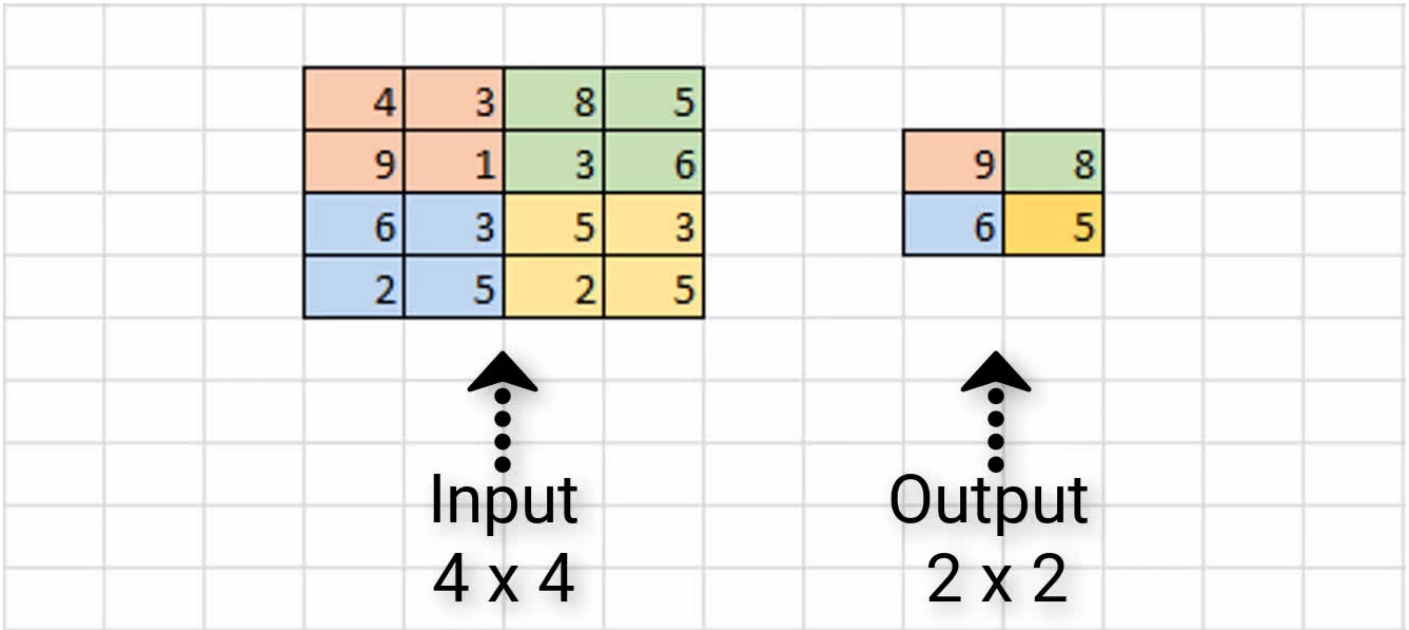
Condense (downsample) using a maximum pooling layer

```
layers.MaxPool2D(pool_size=(2, 2))
```

MaxPool is usually added after a convolution layer.

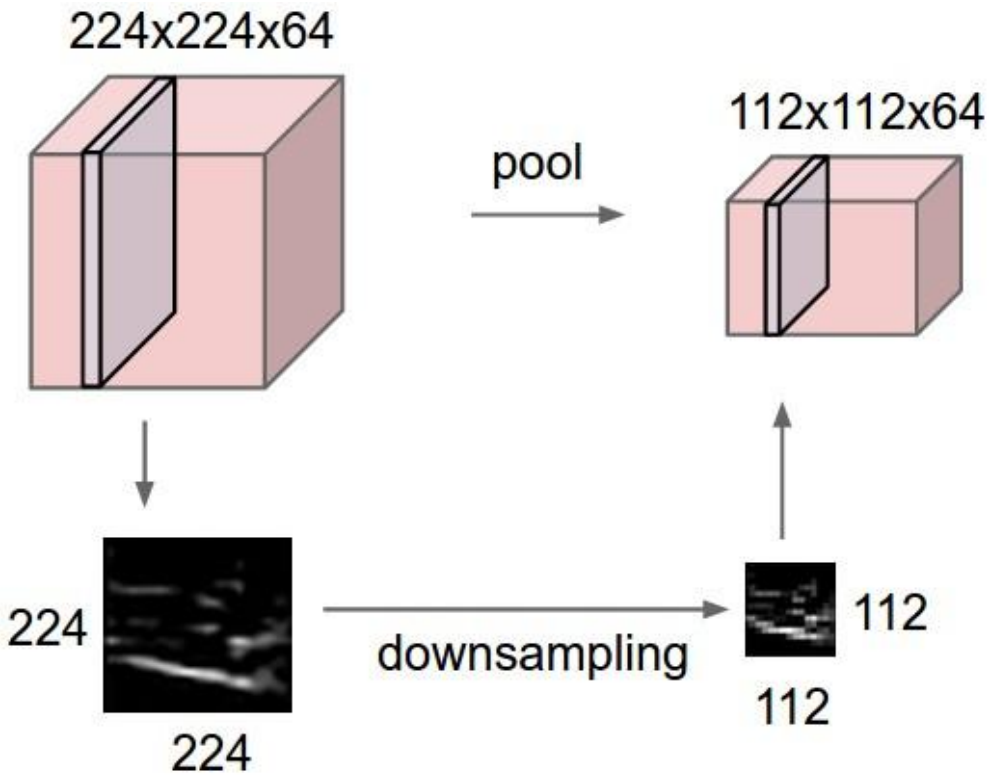
MaxPool basically shrinks an image (array) using the maximum value. The size of the sliding window is defined by `pool_size`, and the `stride` defaults to the pool size.

```
MaxPool2D(pool_size=(2, 2))
```



also there is average pooling:

```
layers.AveragePooling2D(pool_size=(2, 2)
```


224x224x64
pool
112x112x64
224
224
downsampling
112
112

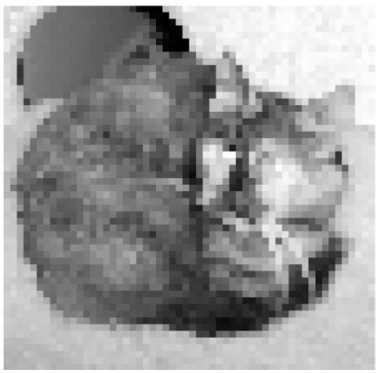
max pooling

(446, 450)

(223, 225)

(111, 112)

(55, 56)



average pooling

(446, 450)

(223, 225)

(111, 112)

(55, 56)

¿Why do we do this?

- smaller images mean less computation
- reduce noise, thus less potential for ‘overfitting’

Practical session

Adapt your MNIST ANN model

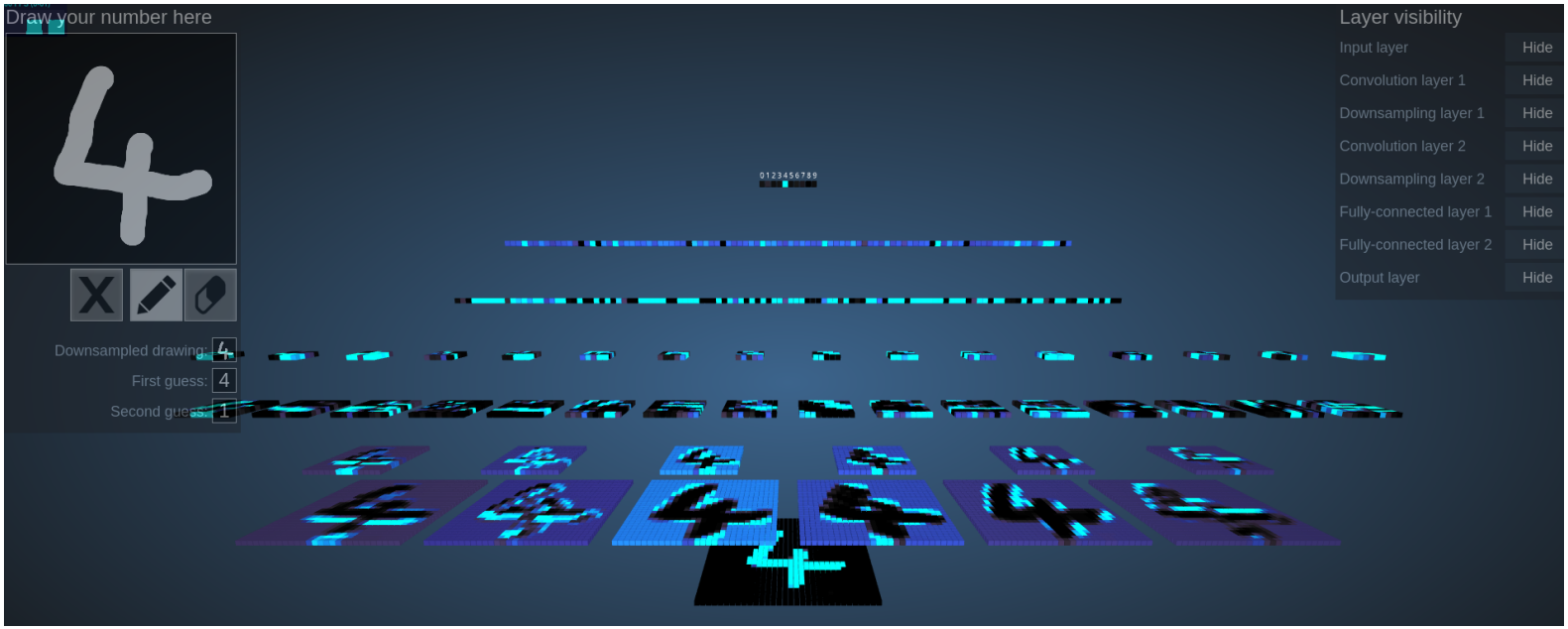
`notebook_ANN_Keras_MNIST.ipynb`

to now be a CNN model, based on the Keras "Simple MNIST convnet" page.

To do: Compare the learning curves and the accuracy (better still, the MCC) score between the ANN and the CNN models.

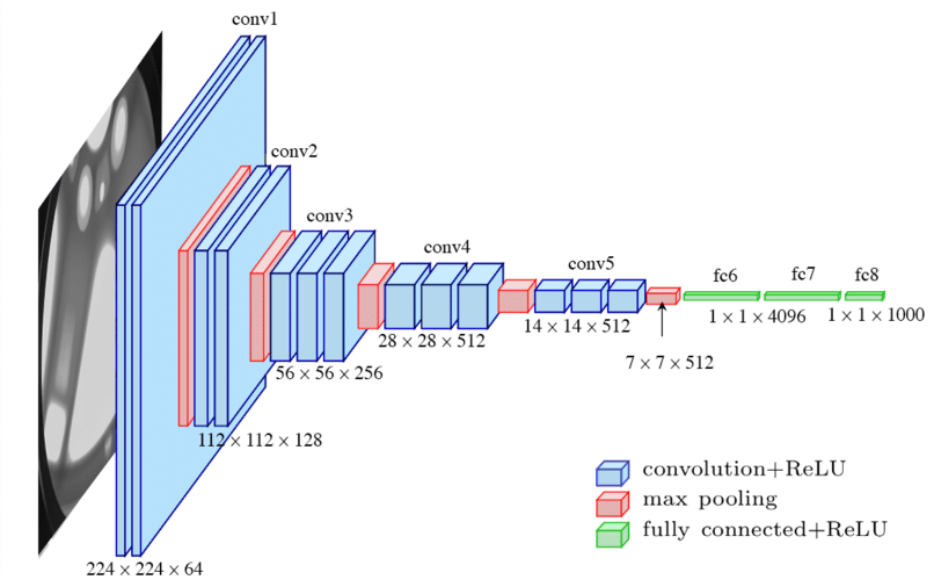
Notice that we had no `flatten` layer
our CNN was working with 2D data

Interactive: 2D convolutional network visualization



This is now the CNN version (for more information see the paper).

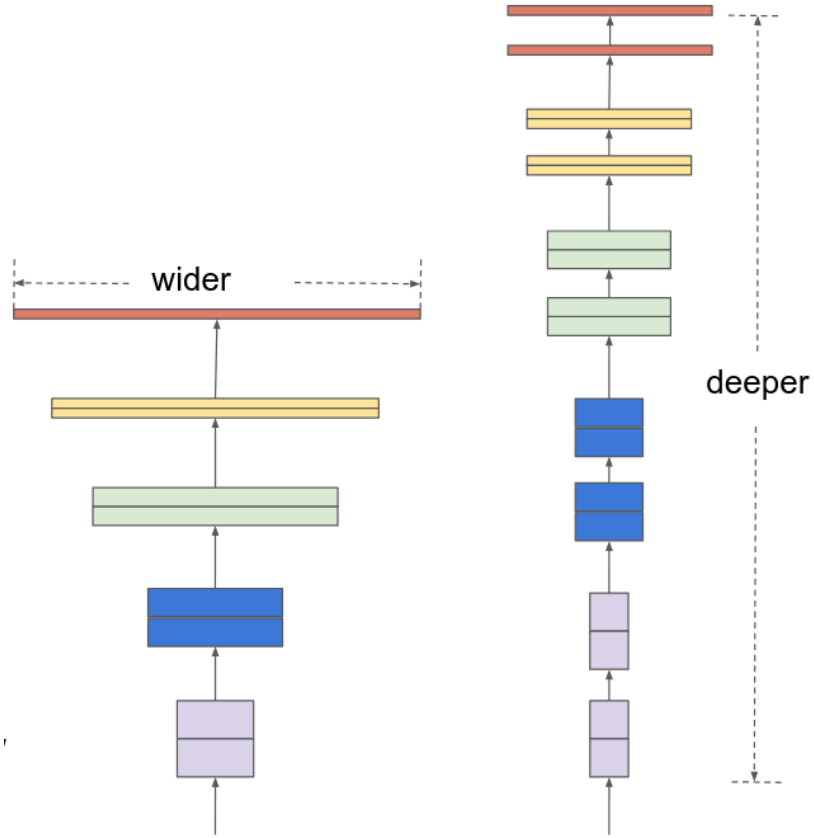
The VGG (Visual Geometry Group) VGG-16 architecture from 2014:



(VGG-16 model source example)

(Paper: *"Very Deep Convolutional Networks for Large-Scale Image Recognition"*)

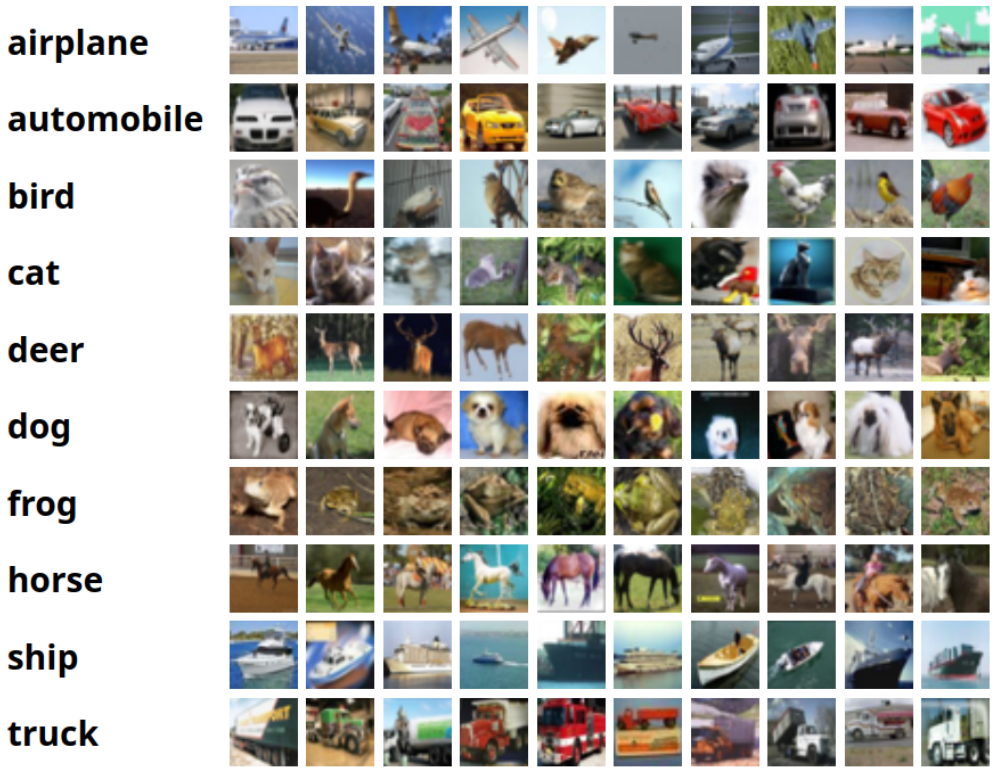
Neural networks: wide vs deep?



(Paper: *"EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks"*)

Colour images: CIFAR-10

The CIFAR-10 dataset (2009)



Getting the CIFAR-10 data using Keras

```
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
```

the CIFAR images now have red, green, and blue blue channels (stored in row-major order)

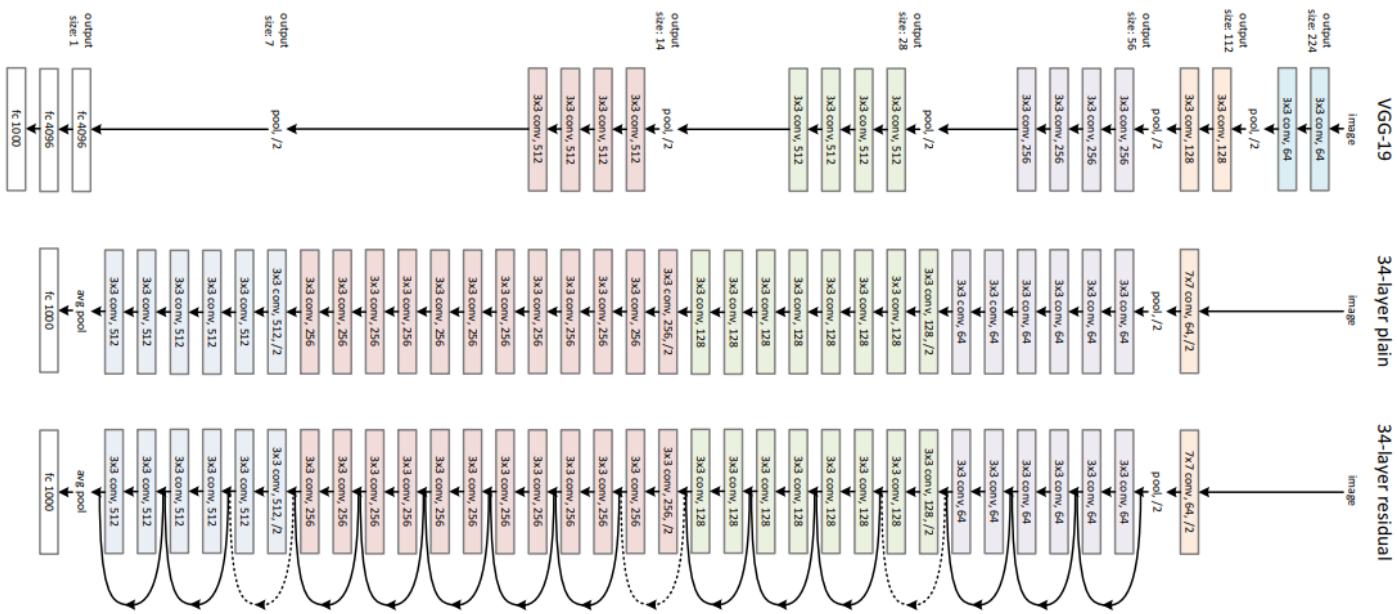
```
image_size = (32, 32)
input_shape = image_size + (3,)
```

Practical session

Let us take a look at CIFAR-10

notebook_CNN_Keras_CIFAR10.ipynb

The Residual Network (ResNet) architecture:



won the 2015 edition of the ImageNet Challenge. ResNet is no longer sequential, having 'residual blocks'.

(Paper: *"Deep Residual Learning for Image Recognition"*)

"The difference in ResNetV1 and ResNetV2 rests in the structure of their individual building blocks. In ResNetV2, the batch normalization and ReLU activation precede the convolution layers, as opposed to ResNetV1 where the batch normalization and ReLU activation are applied after the convolution layers."

2017 was the last year of the 2D ImageNet challenge

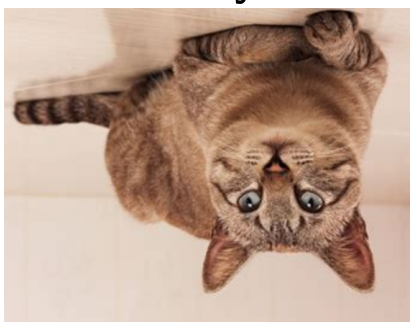
2D object detection and image classification is now a ‘solved problem’; computers essentially outperform humans.

Data augmentation

The more data we train on, the better the model.
In classification we can make use of symmetry to perform label-invariant transformations, for example we can randomly flip some of our images:

```
keras.layers.RandomFlip(mode="horizontal_and_vertical",)
```


...but only within reason

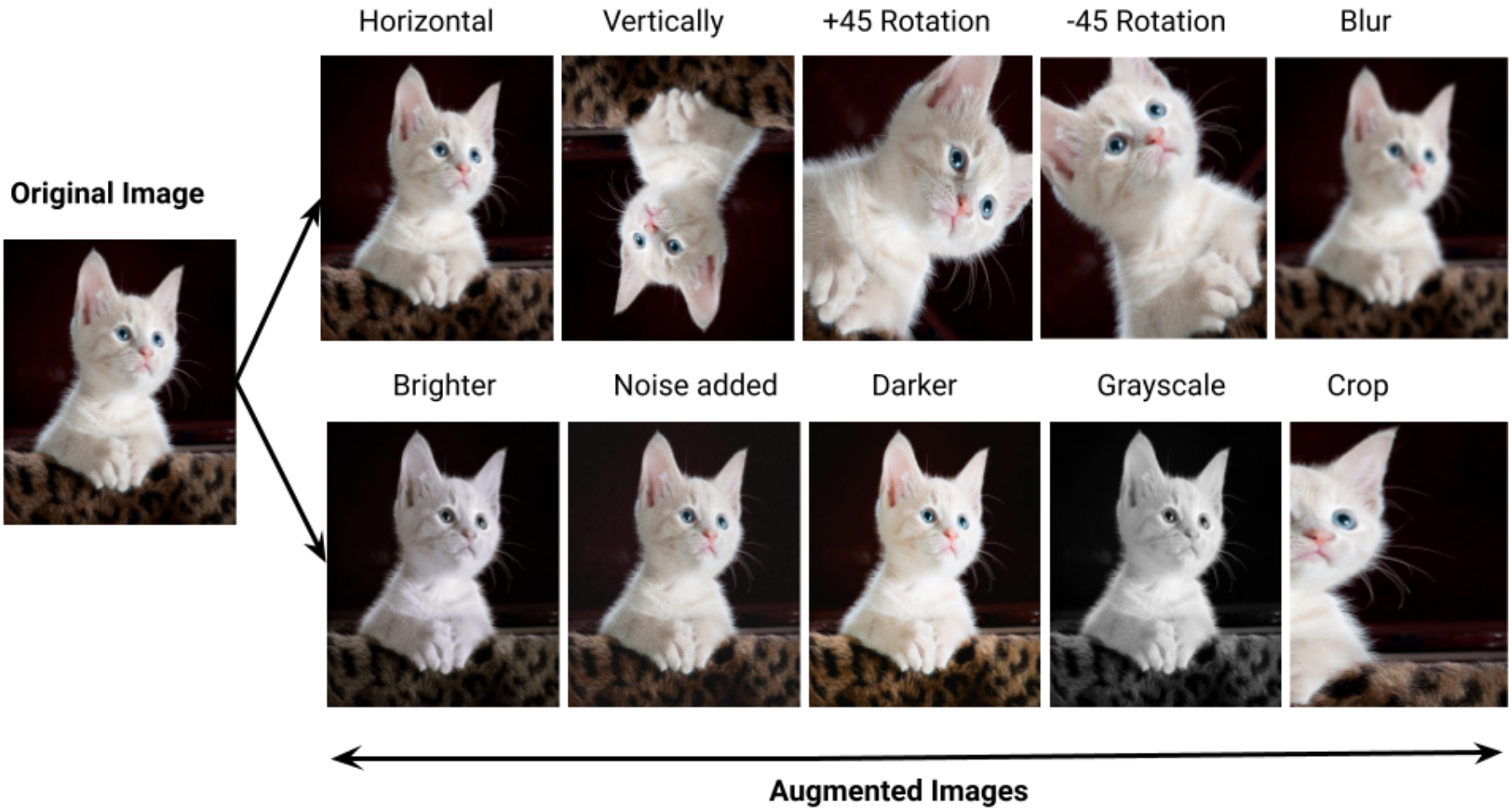


Keras provides a number of such transformations
that can be applied to (mini-)batches as a layer in ones sequence:

Image augmentation layers

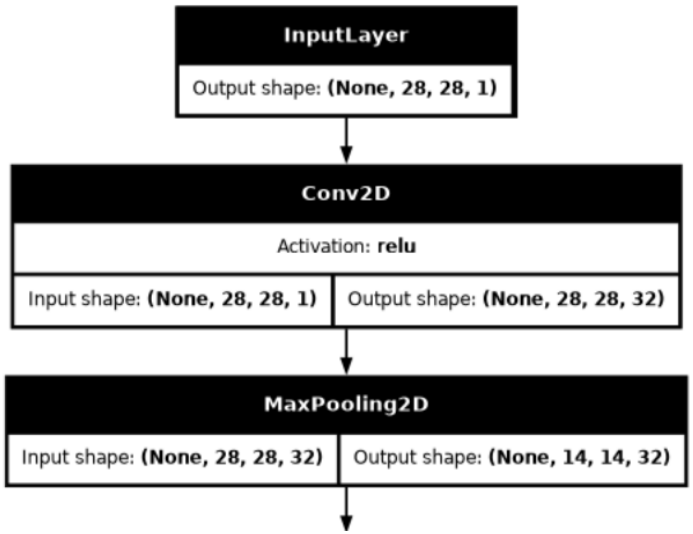
- RandomCrop layer
- RandomFlip layer
- RandomTranslation layer
- RandomRotation layer
- RandomZoom layer
- RandomContrast layer
- RandomBrightness layer

See also the albumentations package


Original Image
Horizontal
Vertically
+45 Rotation
-45 Rotation
Blur
Brighter
Noise added
Darker
Grayscale
Crop
Augmented Images

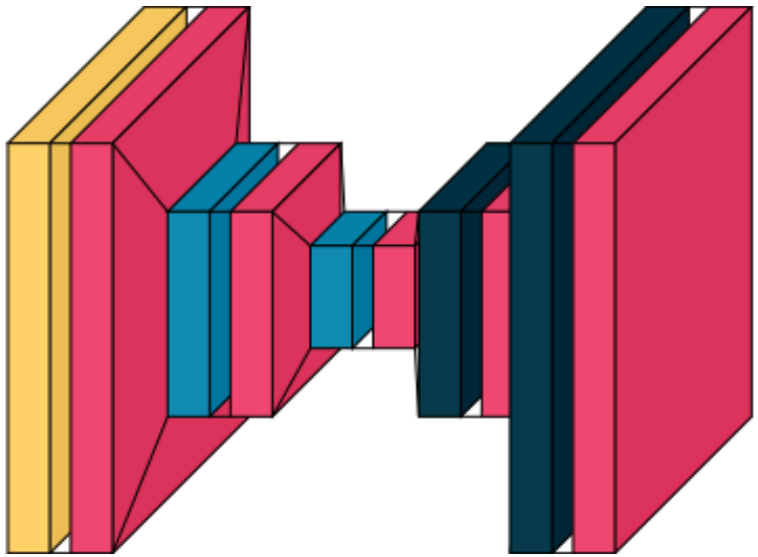
We shall now see two ways of visualizing our neural network architecture:

```
keras.utils.plot_model(model,
           show_shapes=True,
           show_layer_activations=True,)
```



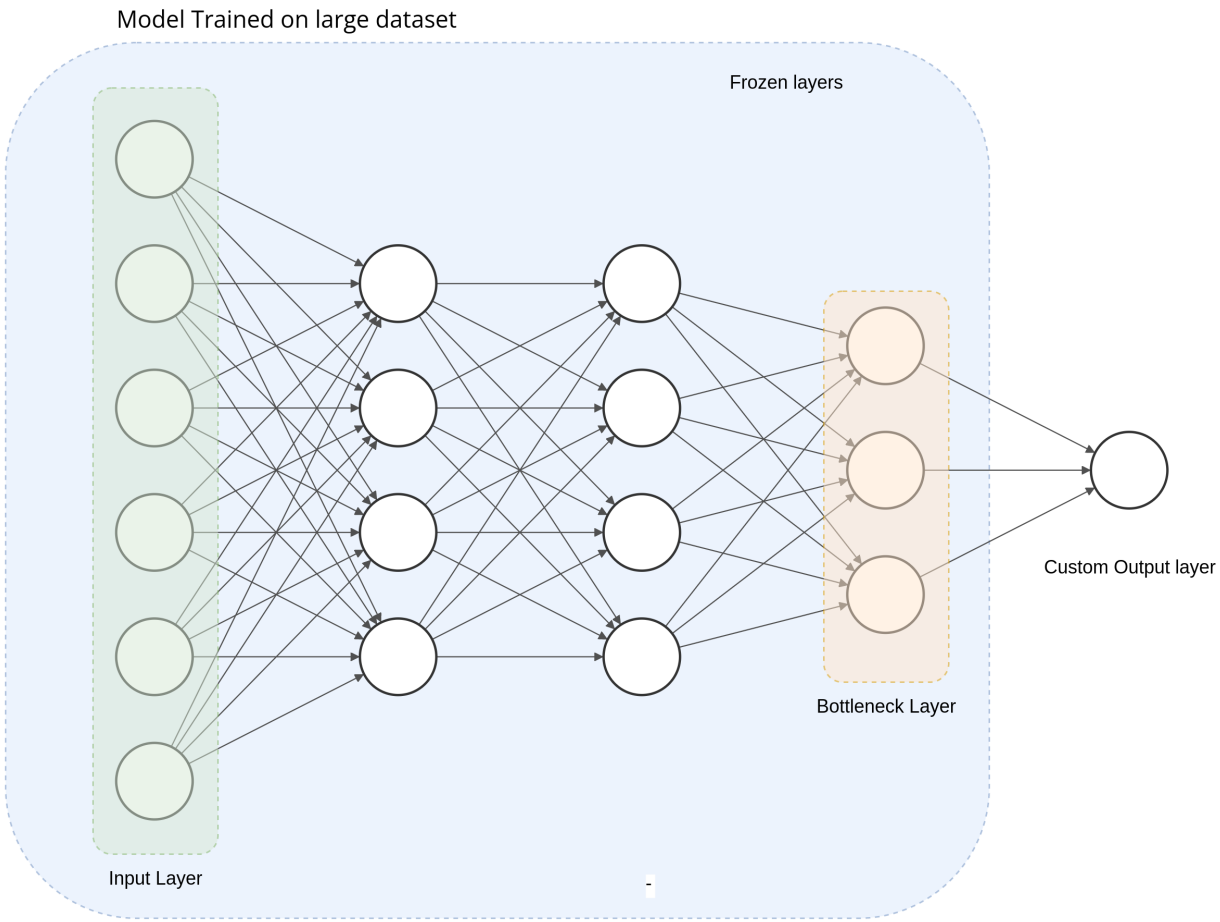
and the visualkeras package

```
!pip install -q visualkeras
import visualkeras
visualkeras.layered_view(model)
```



Transfer learning

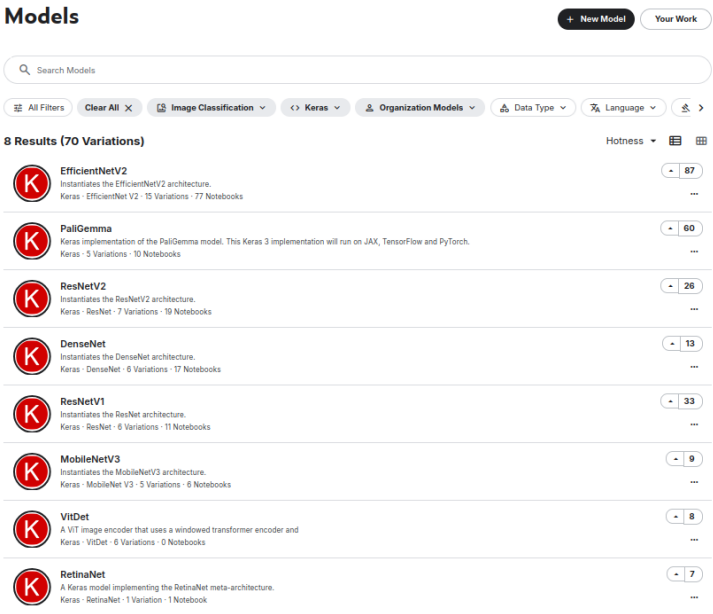
"Transfer learning consists of taking features learned on one problem, and leveraging them on a new, similar problem. For instance, features from a model that has learned to identify racoons may be useful to kick-start a model meant to identify tanukis.
Transfer learning is usually done for tasks where your dataset has too little data to train a full-scale model from scratch."



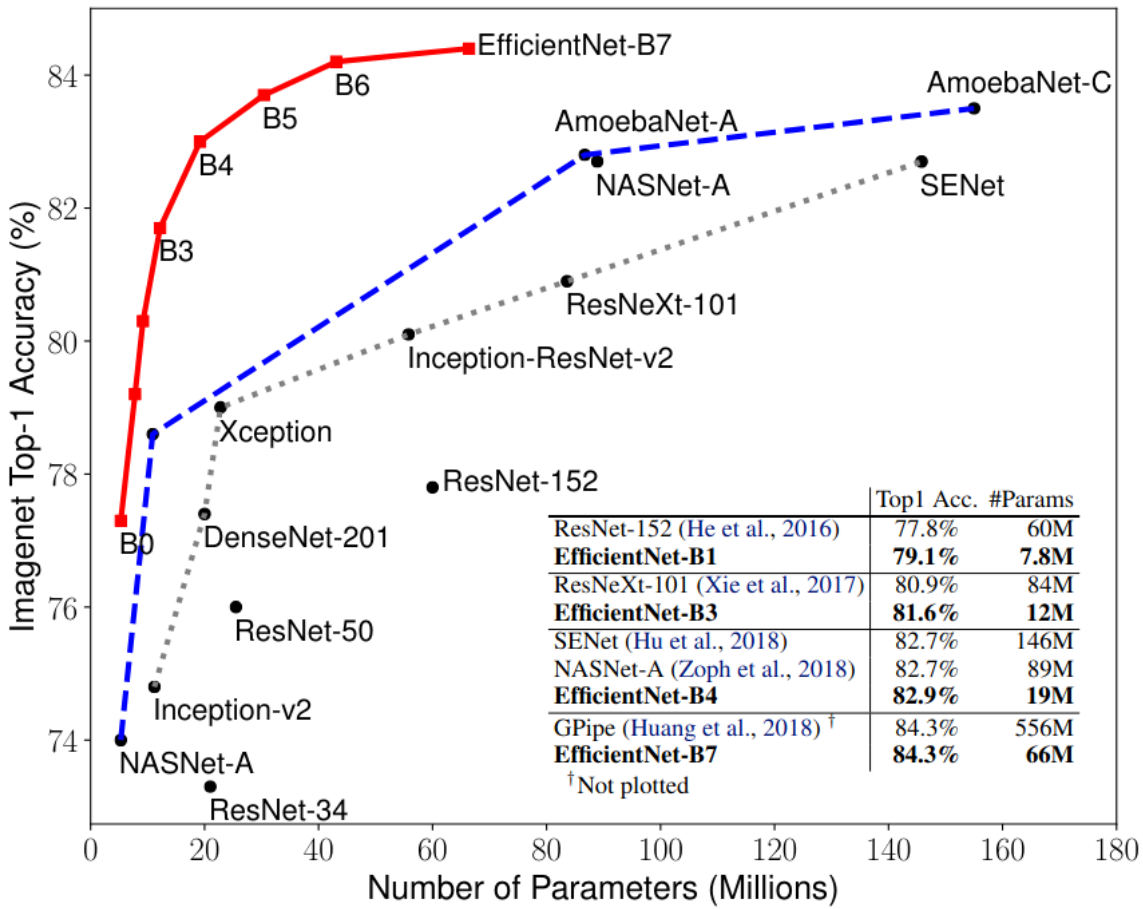
Freezing the layers of a pre-trainind model avoids destroying any of the information it contains during future training rounds.

Pre-trained models

Keras provides many models along with pre-trained weights



Pre-trained Keras image classification models on Kaggle, of particular note are ResNetV2 and EfficientNetV2.



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

Keras also provides a VGG-16 model pre-trained on the `imagenet` data:

```
from keras.applications.vgg16 import VGG16

base_model = VGG16(
    include_top=False,
    weights="imagenet",
    input_tensor=None,
    input_shape=(32, 32, 3),
    pooling=None,
    classes=10,
    classifier_activation="softmax",
    name="vgg16",
)
```

(source code)

then

```
for layer in base_model.layers:
    layer.trainable = False

model = Sequential()
model.add(base_model) # for example use a VGG16 base model here
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(n_classes, activation='softmax'))
```

Keras has a ResNet for `imagenet` too.

```
model = keras_cv.models.ImageClassifier.from_preset(
    "resnet50_imagenet",
    n_classes=2,)
```

For binary classification set `num_classes=2`

For example, to instantiate a 'base' model (here Xception) having pre-trained weights and biases:

```
base_model = keras.applications.Xception(
    weights='imagenet',
    input_shape=(150, 150, 3),
    include_top=False)

base_model.trainable = False

base_model.summary()
```

Fine-tuning

Fine-tuning consists of unfreezing the entire model you obtained above (or part of it), and re-training it on the new data with a very low learning rate (for example 0.00001). This can potentially achieve meaningful improvements, by incrementally adapting the pretrained features to the new data.

(Very) highly recommended reading

Keras manual: Transfer learning & fine-tuning

Kaggle competition

Binary image classification competition:
Is the photo of a cat or of a dog?