Machine Learning

Convolutional Neural Networks and Autoencoders

Dr Guillermo Hamity

University of Edinburgh

October 18, 2021

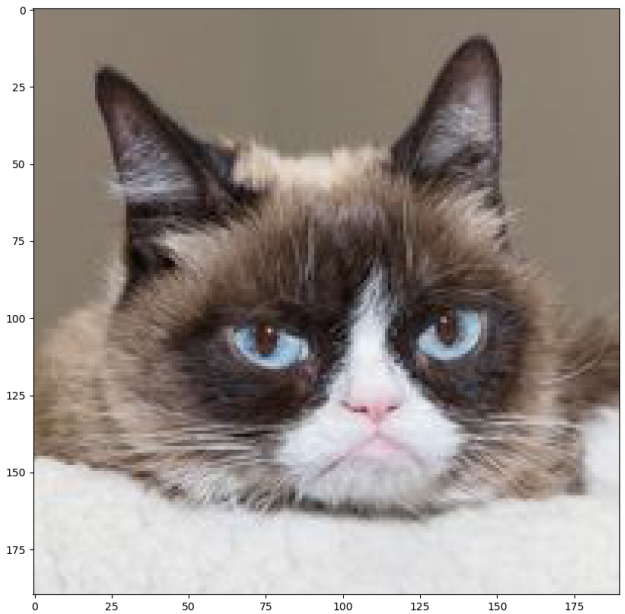
European Research Council

Established by the European Commission

Checkpoint1

- Well done on checkpoint 1, many people doing very well!
- Still waiting on a few more marks but should get feedback before checkpoint on Wednesday.

Thanks for being here at Monday at 9AM

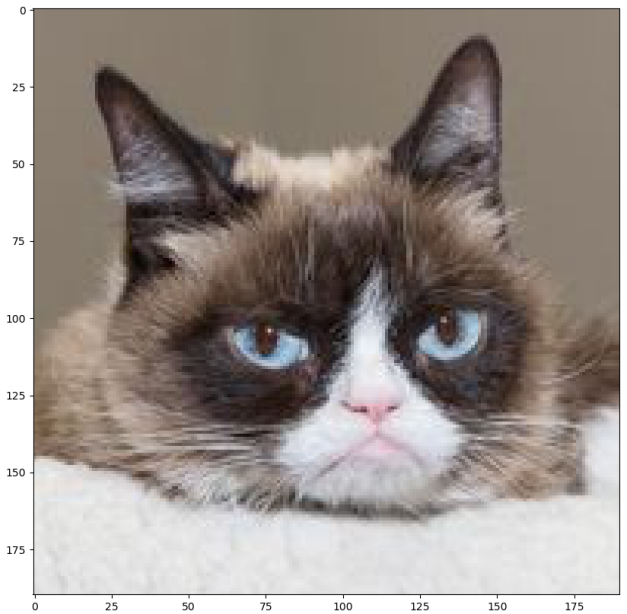


Today's lecture

- We are going to tackle the problem of computer vision
  - Computers deriving meaningful information from images

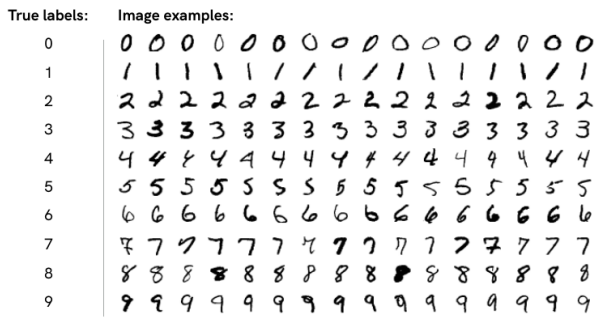
How

1. Discuss image representations and abstractions
2. Introduce and discuss **Convolutional Neural Networks**
3. Revisit image abstractions with **Auto Encoders**

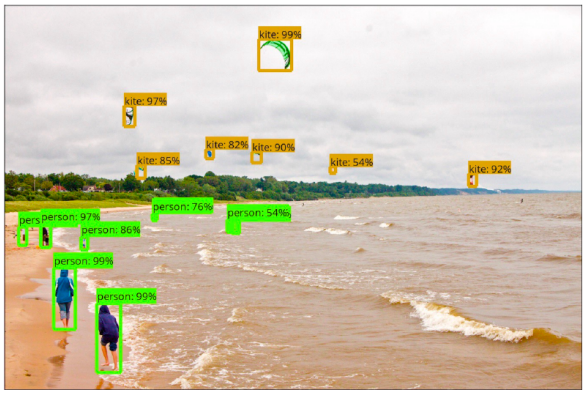


Computer Vision has many applications
e.g. Segmentation, Localisation, Classification, Generation

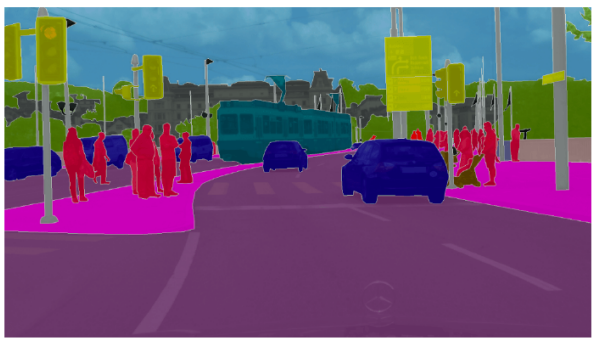
Classification



Localisation



Segmentation



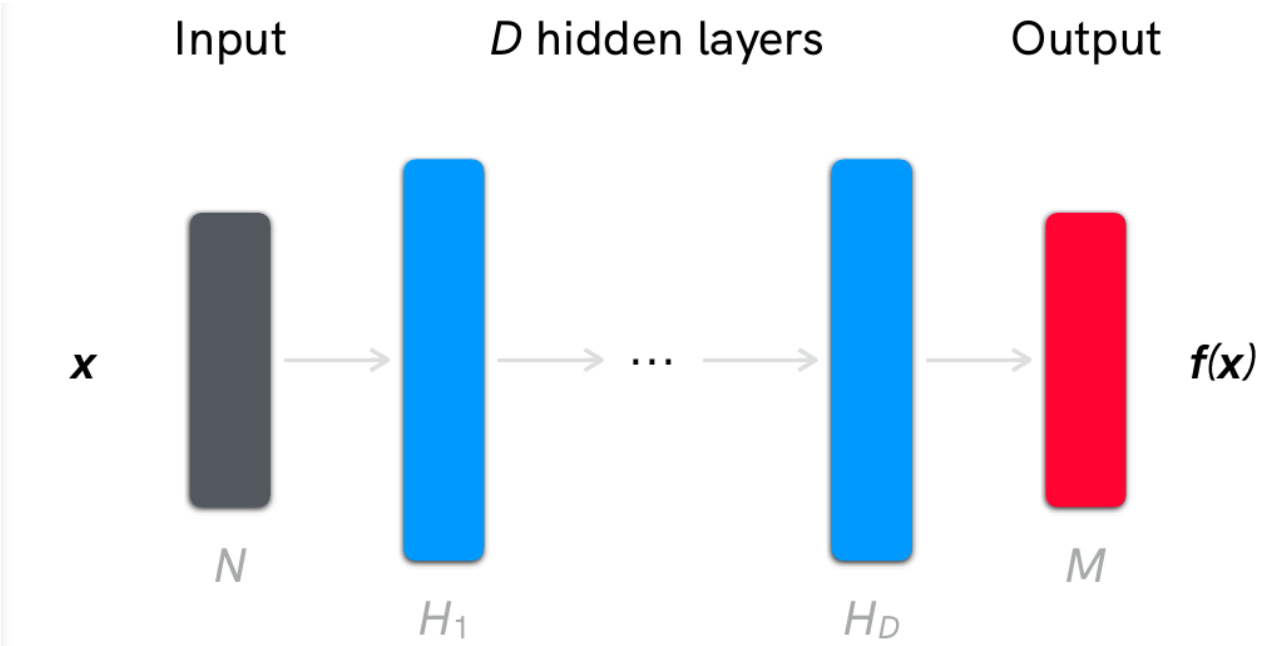
Generation

(left picture is fake)

Classification

Focus on classification as our introduction into computer vision

- How would we go about applying machine learning (ML) to image classification?
  - Want N-class classification: `human`, `cat`, `dog`, `car`, etc.
- We could try Deep Neural Network from **Lecture3**



- At first glance this seems reasonable, but we will run into problems quickly

Image data { R G B}

- Understanding image data
  - This image is `190X190` pixels (resolution), with 3 colour channels:
    - Red Green Blue
  - Represent as tensor $\mathbf{T = (190,190,3)}$
  - Elements are unsigned integers (8bits) with values $0 \leq T_{xyz} \leq 255$

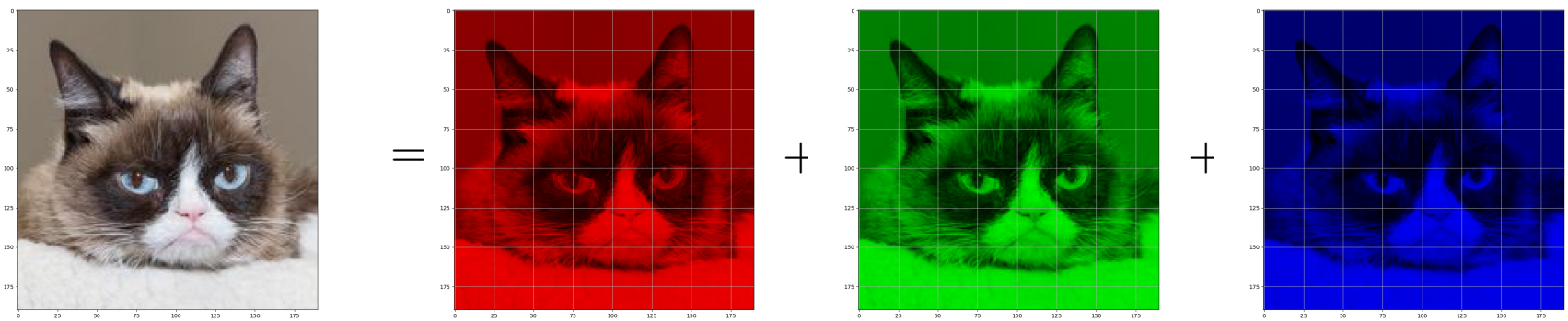
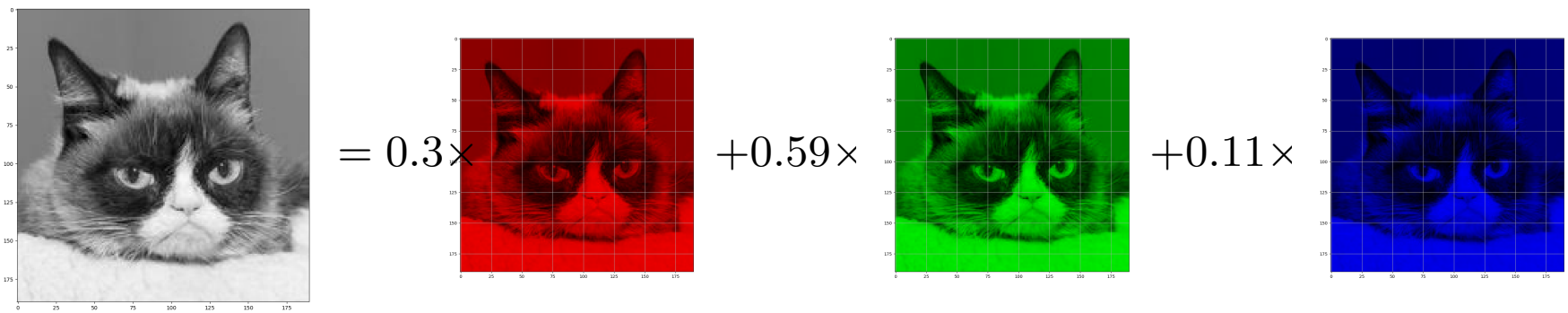


Image data (Grayscale)

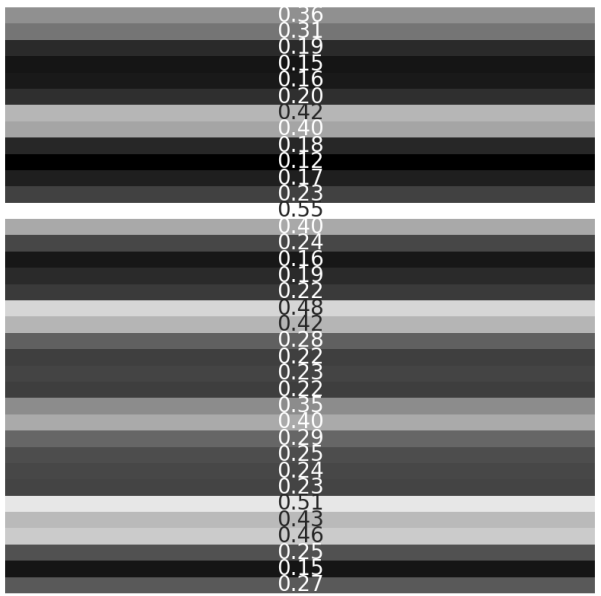
- Can also convert to grayscale/black-and-white:
- Then, represent as tensor $\mathbf{T = (190,190,1)}$
- Elements are still integers with values $0 \leq T_{xy0} \leq 255$



Vectorised Image data

- To train our DNN we should scale pixels to `[0,1]`: `pixs/255`
- Next we unravel the image to 1D feature vector needed as input to NN
  - e.g. **(6,6,1)** -> **(36,1,1)**

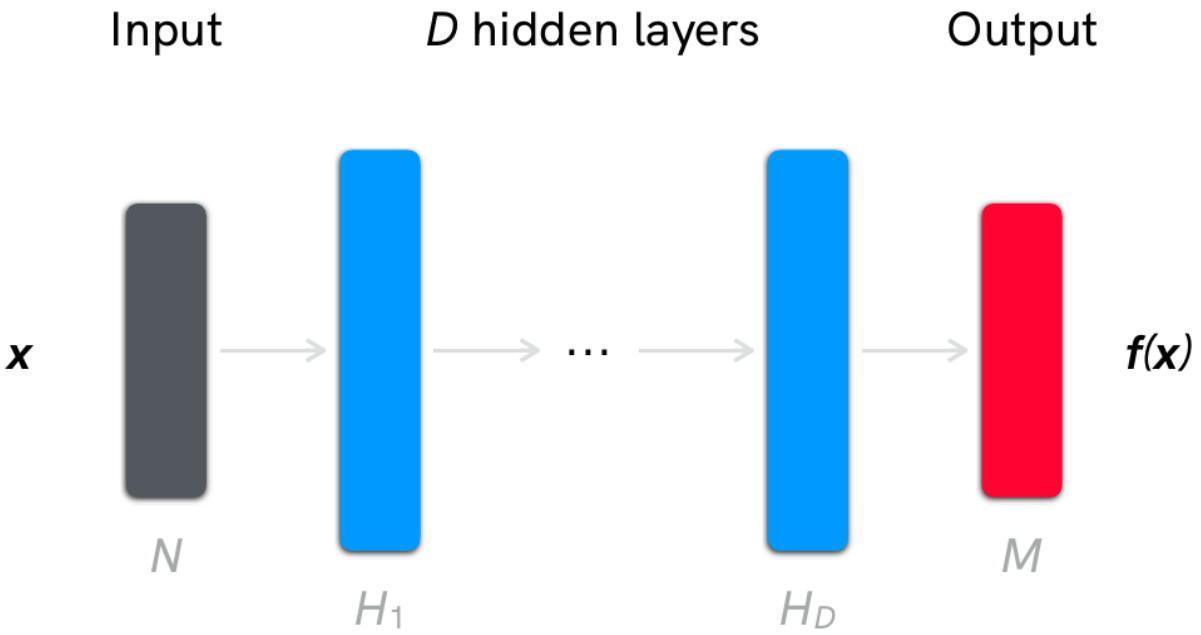
0.36	0.31	0.19	0.15	0.16	0.20
0.42	0.40	0.18	0.12	0.17	0.23
0.55	0.40	0.24	0.16	0.19	0.22
0.48	0.42	0.28	0.22	0.23	0.22
0.35	0.40	0.29	0.25	0.24	0.23
0.51	0.43	0.46	0.25	0.15	0.27



DNN Problem 1

Flattened image can be input into DNN

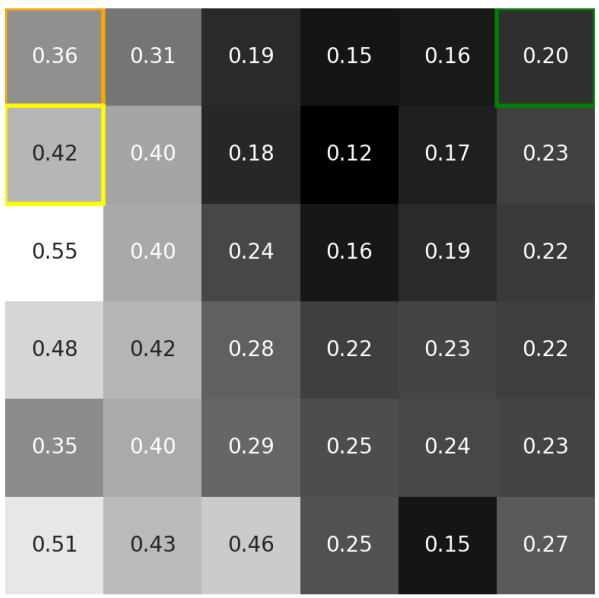
- For our cat `190x190x1` (or `3`) we have `36100` (`108300`) vector
- For relatively low res image, we get ($O(100'000)$) input features
- **That is a lot!**

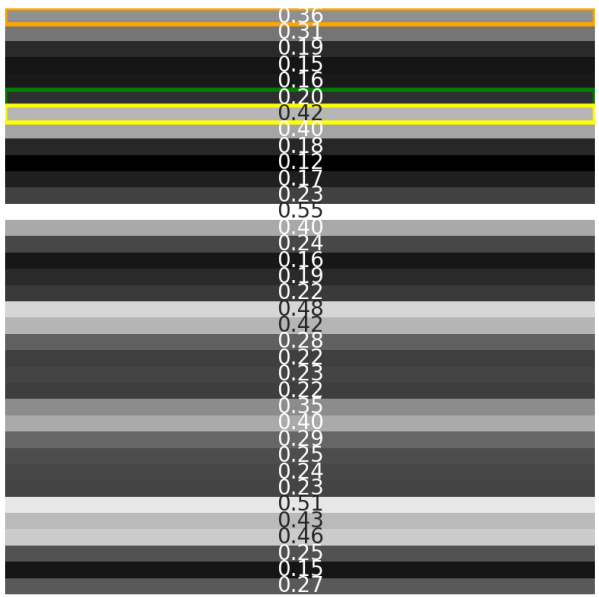


Problem with dimensionality: If first hidden layer has, say, 100 nodes, then first layer alone has $O(100'000) \times 100 \approx 10$ million free parameters

DNN Problem 2

- Another problem with flattening: image information with adjacent pixels containing information about local structures
- By flattening to vector we unravel the local structure, which will need to be relearned.



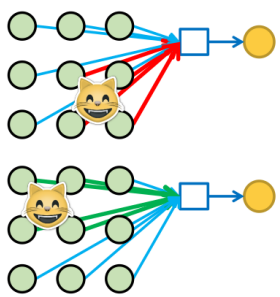


Problem with loss of local correlations

DNN Problem 3

- DNNs are not designed for shifts in feature positions
  - Good with fixed inputs (`x1 = weight`, `x2 = height`), and we don't change to (`x2 = weight`, `x1 = height`) for different event

- If shifting image by 1 pixel in any direction, each `pixel feature` changes
- What if cats in the test set appear in different places?



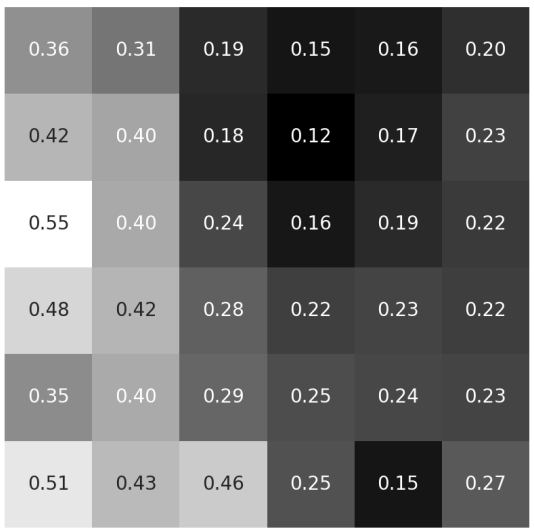
On this training image red weights w_{ij} will change a little bit to better detect a cat

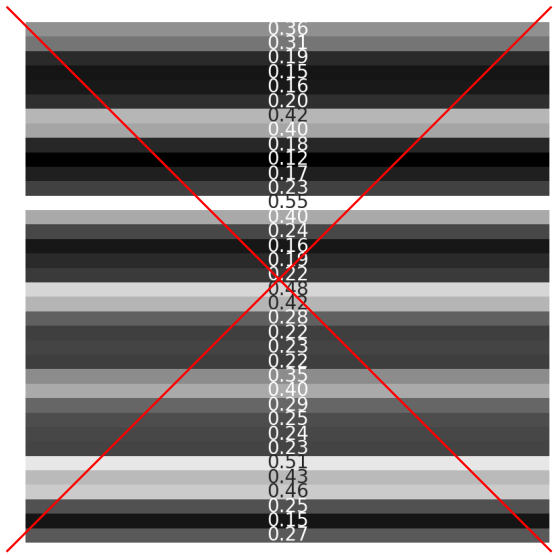
On this training image green *weights* w_{ij} will change...

Problem that DNN is not invariant under translation

Convolutions

- To preserve local structure we will use the image directly



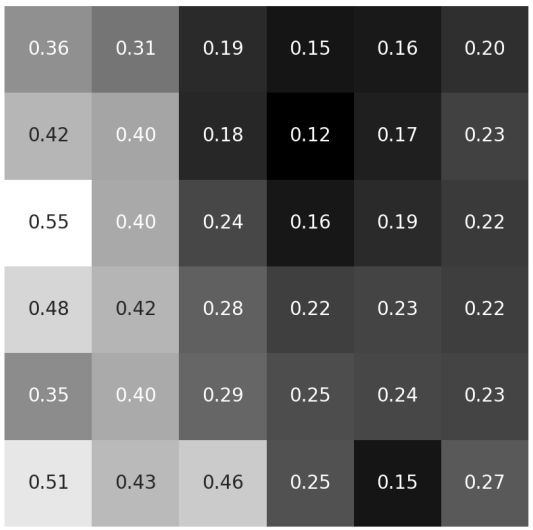


- We will use **convolution** to reduce dimensionality while preserving local correlations
- Convolutions are transformations of original image to new image through a **filter**

Example of convolution with filter kernel

- Common to do image processing using **kernels** (filter matrix)
  - Transformations which offer different representations of the image
- Filtering an image by sliding window **dot product** of `3x3 matrices`

Image



Kernel

$$\cdot \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

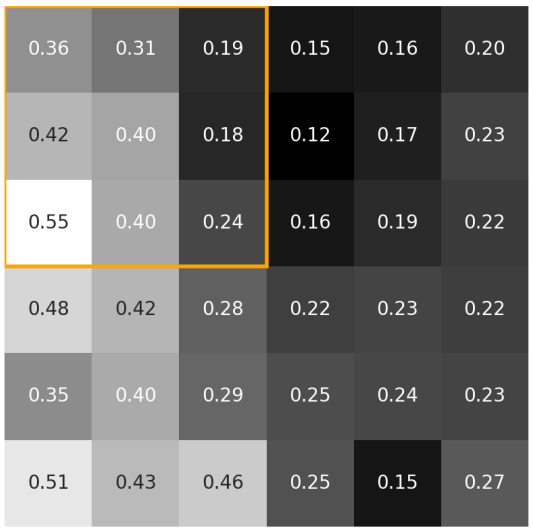
Output Image

$$= \begin{pmatrix} & \\ & \end{pmatrix}$$

Example of convolution with filter kernel

- Common to do image processing using **kernels**
  - Transformations which offer different representations of the image
- Filtering an image by sliding window **dot product** of `3x3 matrices`

Image **Kernel** **Output Image**



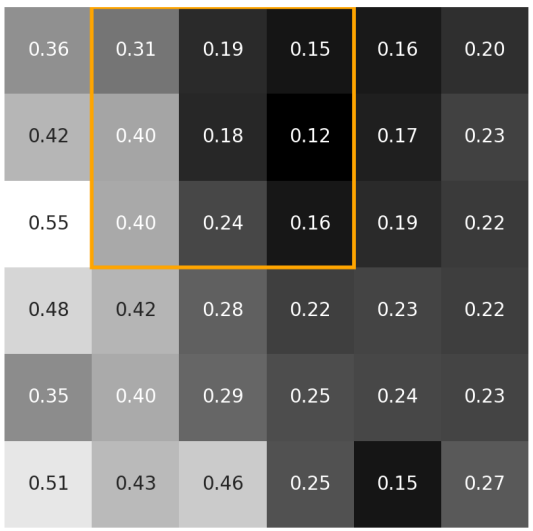
$$\cdot \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} = \begin{pmatrix} 0.55 & & \\ & & \\ & & \end{pmatrix}$$

```
-0.36 -0.31 -0.19 -0.42+ 0.4*8 -0.18 -0.55 -0.4 -0.24 = 0.55
```

Example of convolution with filter kernel

- Common to do image processing using **kernels**
  - Transformations which offer different representations of the image
- Filtering an image by sliding window **dot product** of `3x3 matrices`

Image



Kernel

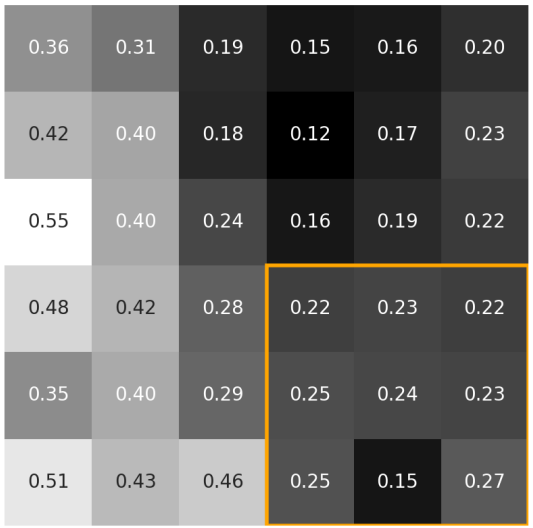
$$\cdot \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

Output Image

$$= \begin{pmatrix} 0.55 & -0.53 & \\ & & \\ & & \end{pmatrix}$$

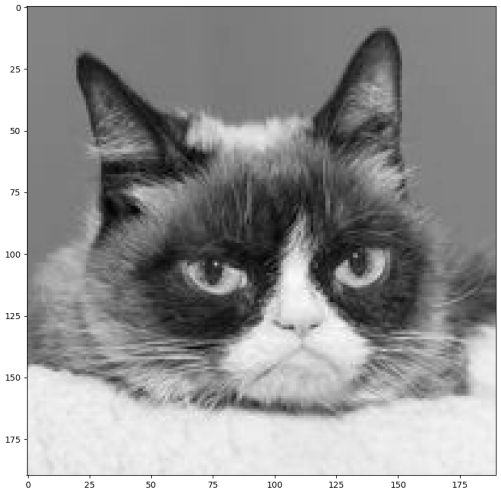
New convolution image

- Common to do image processing using **kernels**
  - Transformations which offer different representations of the image
- Filtering an image by sliding window **dot product** of `3x3 matrices`

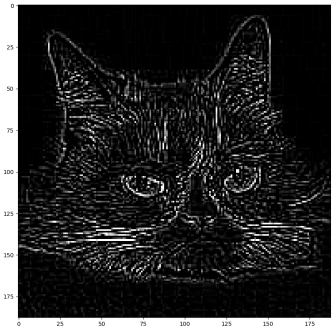


$$\cdot \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} = \begin{pmatrix} 0.55 & -0.53 & -0.48 & -0.07 \\ 0.23 & -0.26 & -0.35 & -0.05 \\ 0.37 & -0.14 & -0.12 & 0.11 \\ -0.02 & -0.39 & -0.12 & 0.1 \end{pmatrix}$$

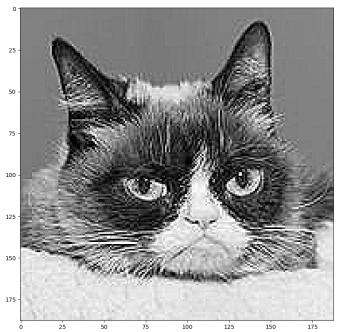
- Keep sliding the kernel till it transforms whole image
- Can you spot what it is doing?
- Note: without `padding` we reduce dimensions: `6x6` to `4x4`



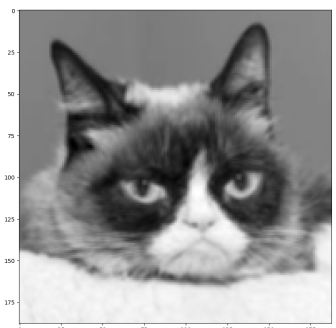
$$\cdot \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$



$$\cdot \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$



$$\cdot \frac{1}{9} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

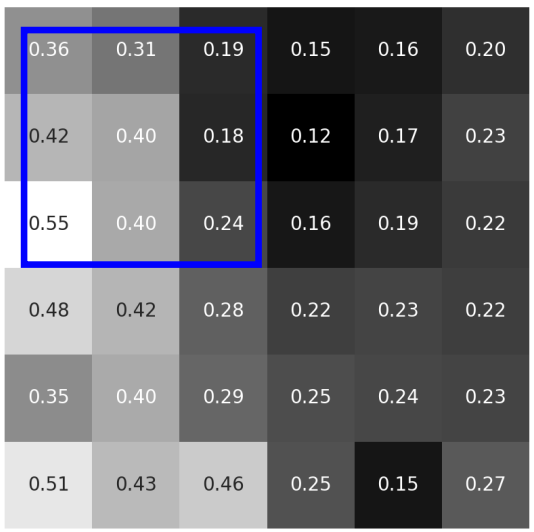


1. **Edge detection**
   Sums up to 0 (dark color) when the patch is a solid fill
2. **Sharpening**
   Adds intensity at edges
3. **Blurring**
   Averages together localised pixels

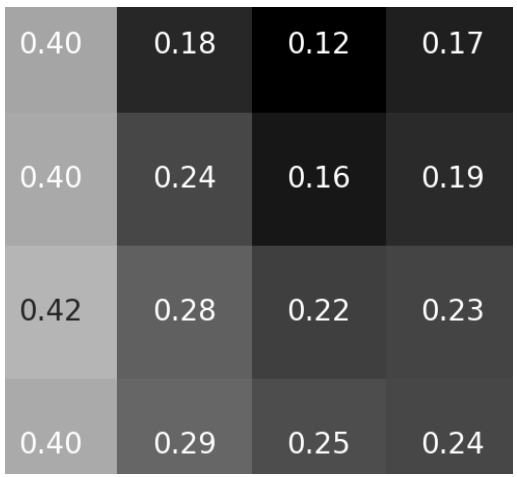
Padding

- Consider `3x3` identity kernel.
- Output image is cropped since central value `1` never sees boundary points of image

Image **Identity kernel** **Output is cropped**

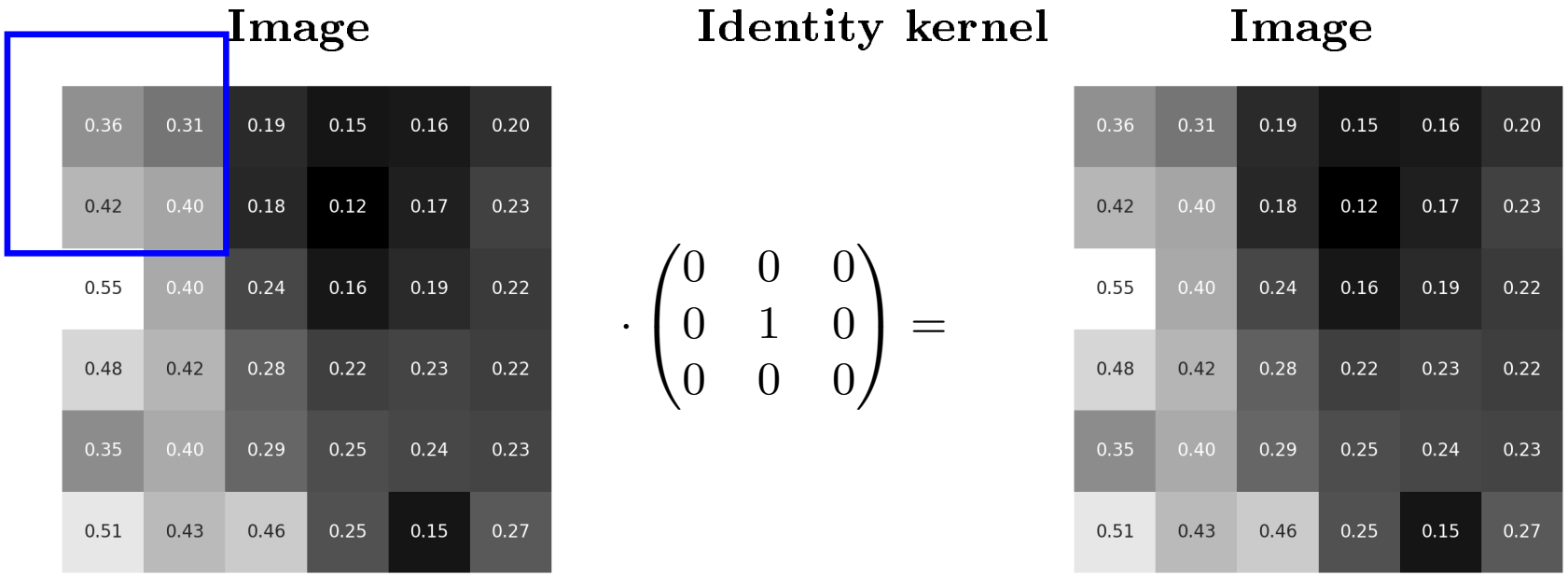


$$\cdot \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} =$$



Padding

- Padding adds rows and columns of zeros
- Kernel will now see boundary points of image



Convolutions in image NNs

- Convolutions offer transformations of images to different representations
- You could consider some to be very useful for image classification
  - e.g. **edge detection** kernel
- How do we decide on which transformations are useful?

Convolutional NN

- Allow a Neural Network to find the best weights **(kernels)** based on minimising Loss
- The NN will extract relevant image representations to tackle the problem

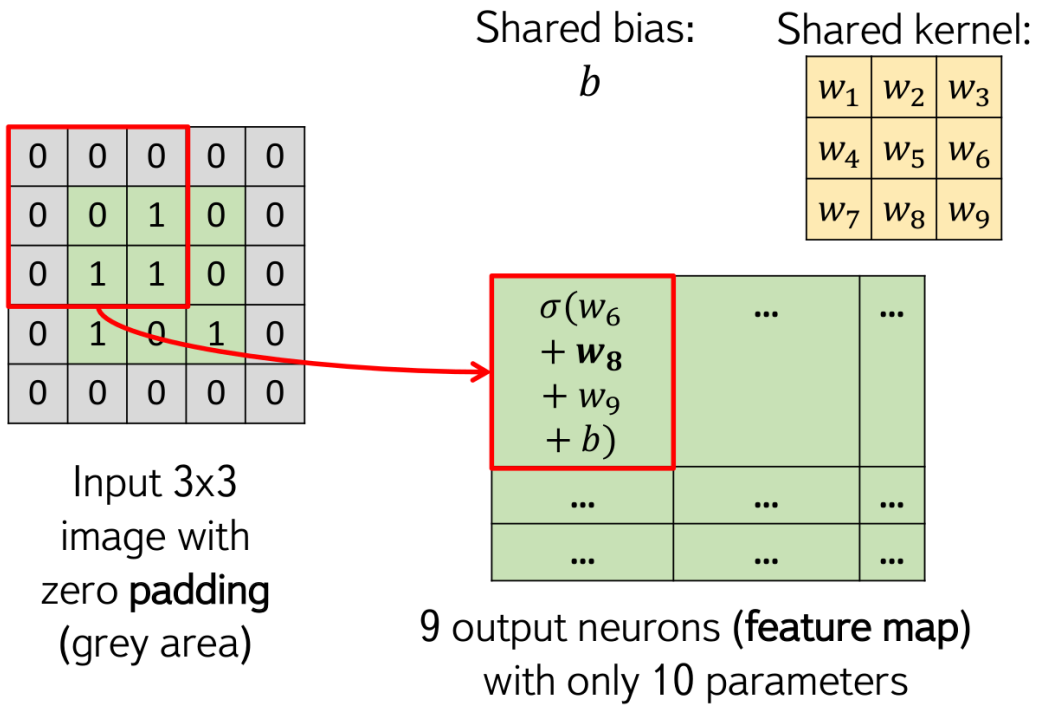
Here our kernels are the trainable weights:

$$\begin{pmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{pmatrix}$$

Convolution layer in neural network

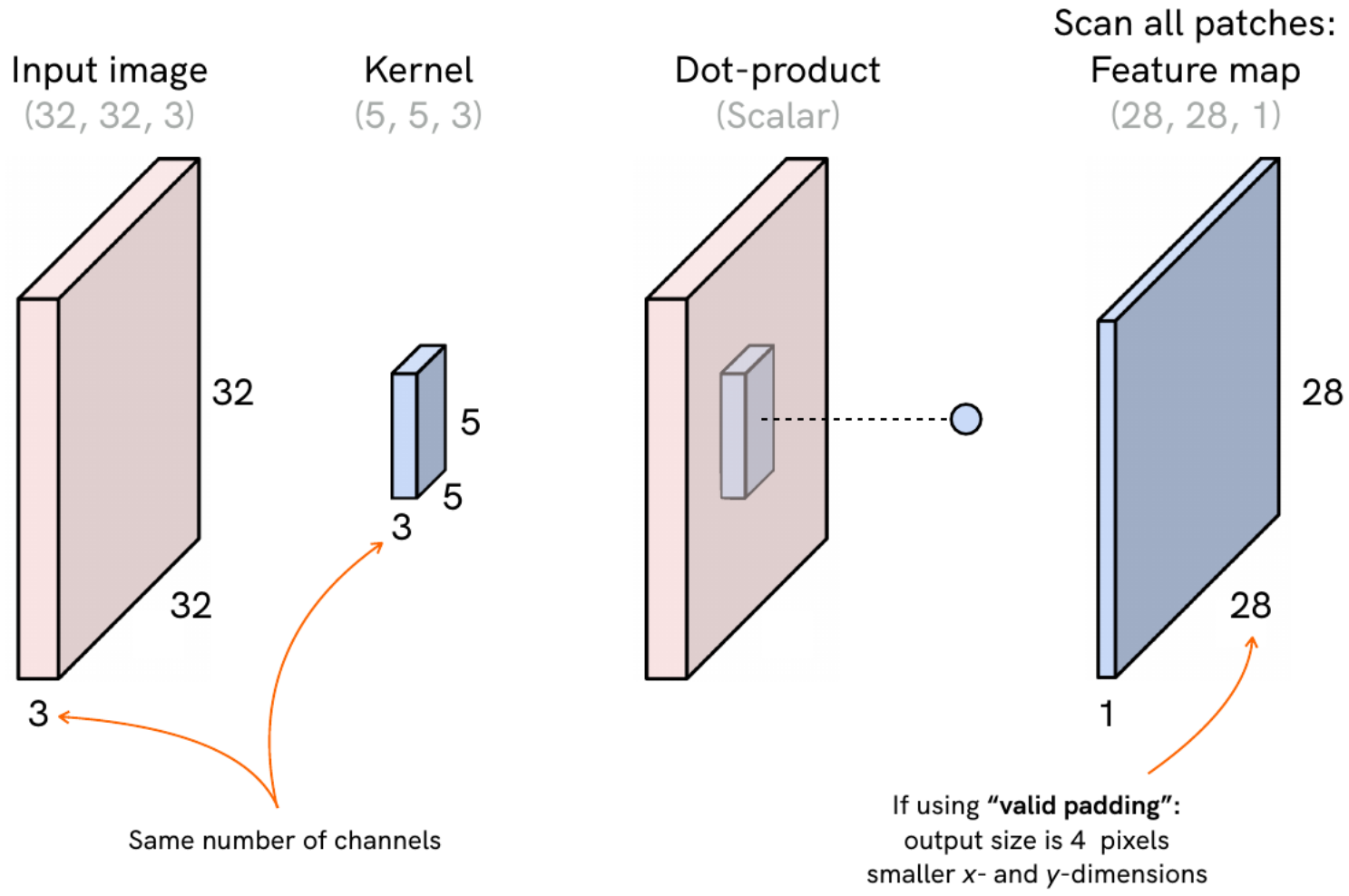
- For `3x3` kernel we have total `10` free parameters
  - `9` kernel weights `w1..w9`
  - `1` bias
- Can add activation function σ to introduce non-linearity
  - acts per convolution dot product

Even for large images, **only 10 trainable parameters**



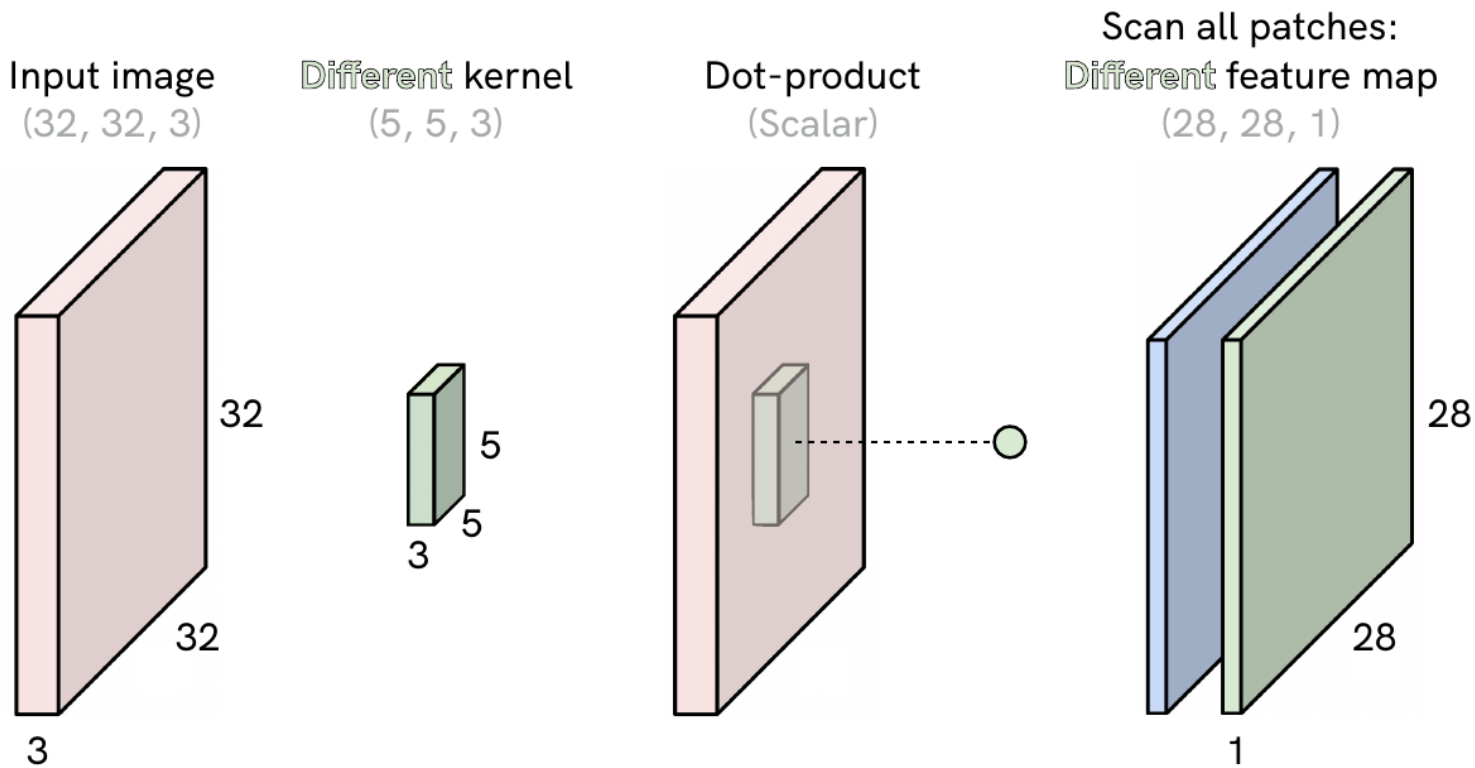
Convolution layer in neural network (1 kernel)

(32,32, **3**) channel image, (5,5, **3**) channel kernel -> (28,28, **1**)



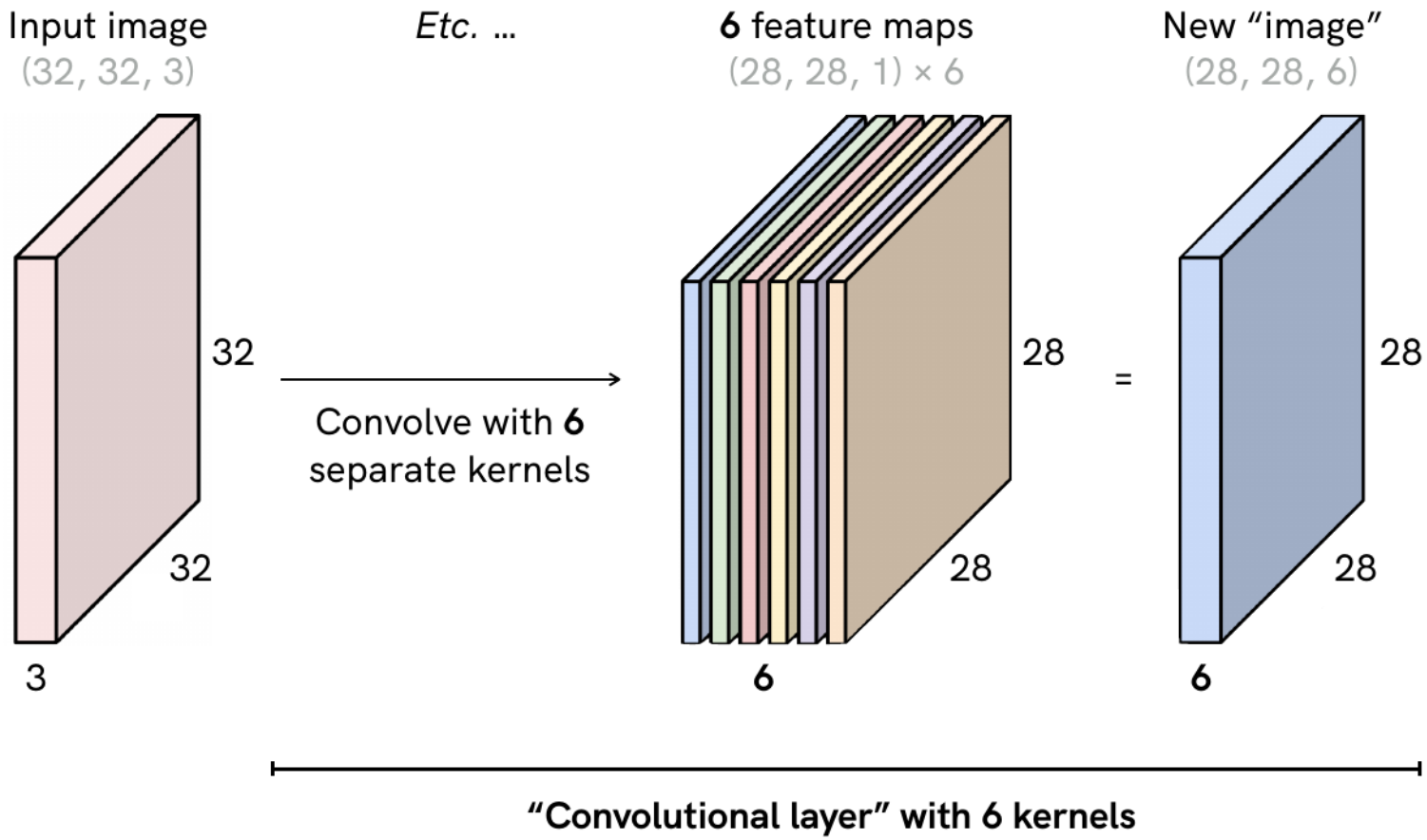
Convolution layer in neural network (2 kernels)

We want to build up different representations of image (several weights)



Convolution layer in neural network (multiple kernels)

and train multiple kernels (**6** kernels) -> image w. 6 channels



Recap: Convolution vs fully connected layer

- In convolutional layer, the same kernel is used for every output neuron, this way we **share parameters** of the network and train a better model;
- 300x300 input, 300x300 output,
  - 5x5 kernel – 26 parameters in convolutional layer
  - 8.1×10^9: parameters in fully connected layer
- Convolutional layer can be viewed as a special case of a fully connected layer, when **all non-local weights** of each neuron **equal 0** and kernel parameters are shared between neurons.

Convolutional layer works better than fully connected layer for images: **fewer parameters and acts the same for every patch of input.**
This layer will be used as a building block for larger neural networks!

Convolution layer in Keras

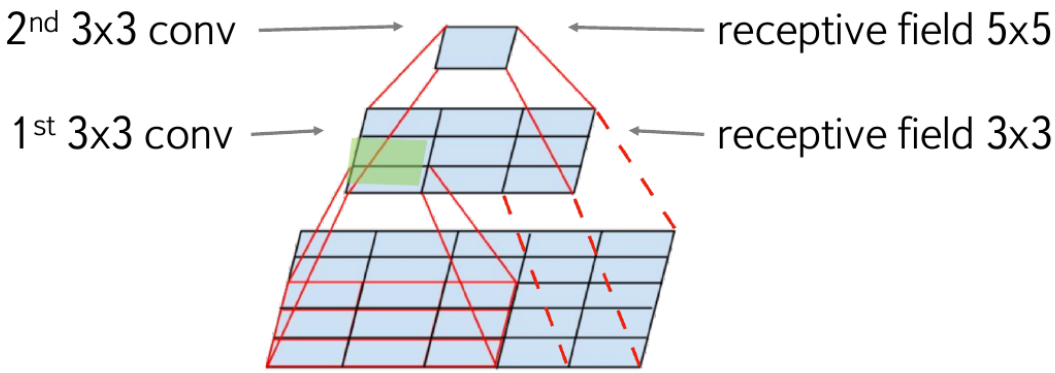
In Keras we can define convolution layers

```
tf.keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding="valid",
    activation=None,
    use_bias=True,
)
```

- `filters`: the number of output filters in the convolution
- `kernel_size`: tuple of 2 integers, specifying the height and width of the 2D convolution window.
  - `3x3` is good choice
- `strides`: Steps size for shifting the kernel
- padding: either
  - `valid` means no padding
  - `same` results in padding with zeros so output has same dimensions as input
- `activation`: Activation function to use.

One convolution layer is not enough!

- Let's say neurons of the 1st convolutional layer look at the patches of the image of size 3x3.
- What if an object of interest is bigger than that?
- We need a 2nd convolutional layer on top of the 1st!

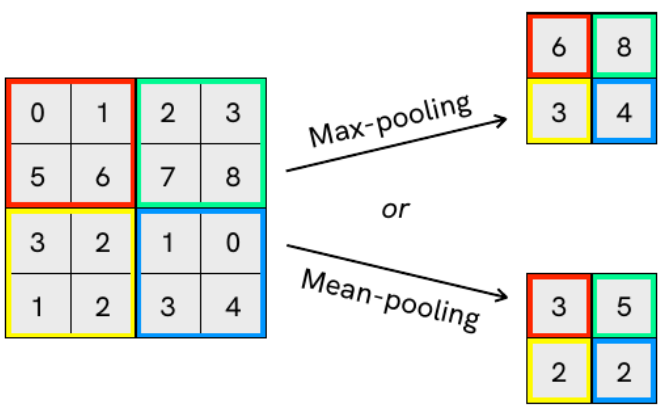


It looks like we need to stack a lot of convolutional layers!
Identify objects as big as the input image 300x300 need 150 3x3 convolutional layers!

Max Pooling

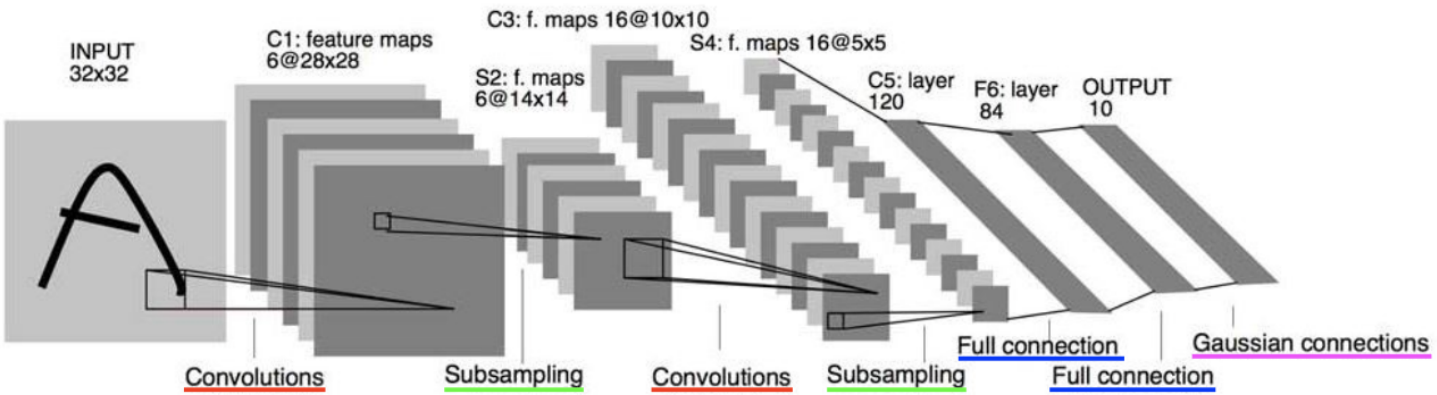
Solution: We can grow perceptive field quicker by **pooling**

- We want to learn relevant, large-scale features by gradually condensing information going from input to output
- We can aggregate activations by **pooling**, typically with size 2×2 (stride 2):
  - **Max-pooling**: Max. of values within window
  - **Mean-pooling**: Average of values within window



Convolutional Neural Network (Putting it all together)

- Sequence convolutional layers with pooling layers to construct feature abstraction
- For each convolutional layer you choose the number of feature maps
- **Readout**: flatten last convolutional layer, connect to output **dense/fully** connected layers



- Typical architecture:
  `[(Conv + ReLU) x N + Pool] x M + (FC+ReLU) x K +softmax`

Convolutional neural networks: Architecture

Example (My CNN)

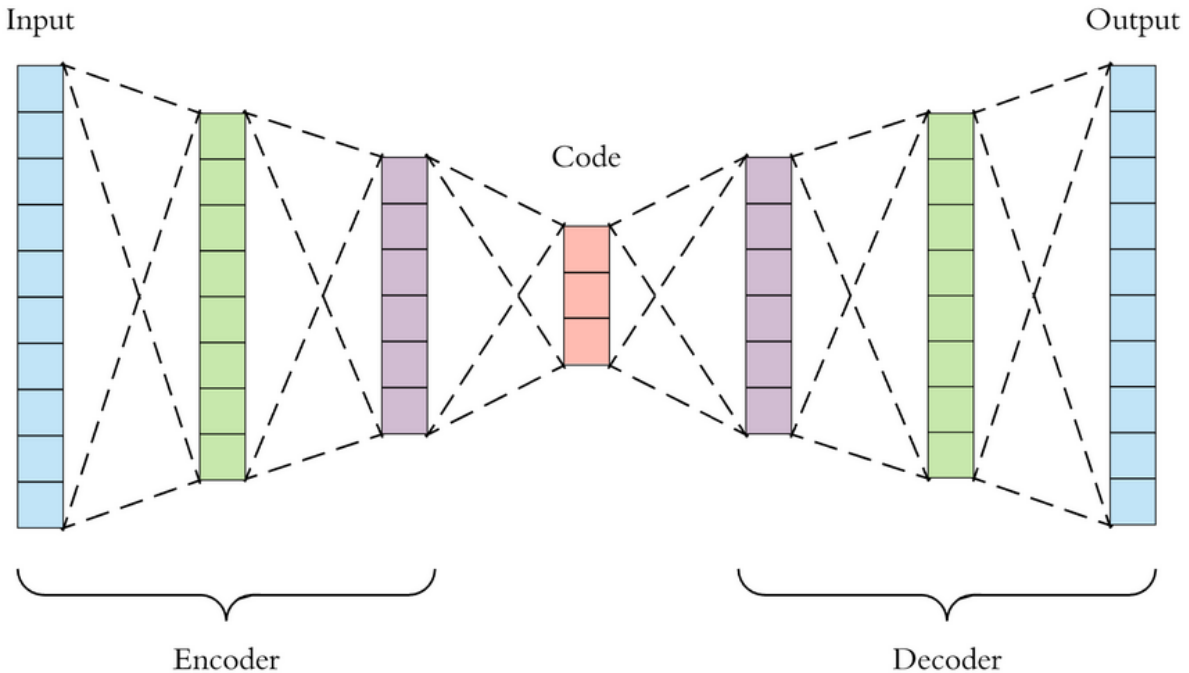
```
visible = Input(shape=(64,64,1))
conv1 = Conv2D(32, kernel_size=5, activation='relu')(visible)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
conv2 = Conv2D(16, kernel_size=5, activation='relu')(pool1)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
flat = Flatten()(pool2)
hidden1 = Dense(10, activation='relu')(flat)
output = Dense(1, activation='sigmoid')(hidden1)
model = Model(inputs=visible, outputs=output)
```

Autoencoders

- CNNs are learning through abstraction, finding low dimensionality representation or larger complex images
  - Learn in supervised way to solve problem (like classification)
- Autoencoders **focus on this dimensionality reduction** and image representation is abstract space
- Form of unsupervised learning (**no target labels**)

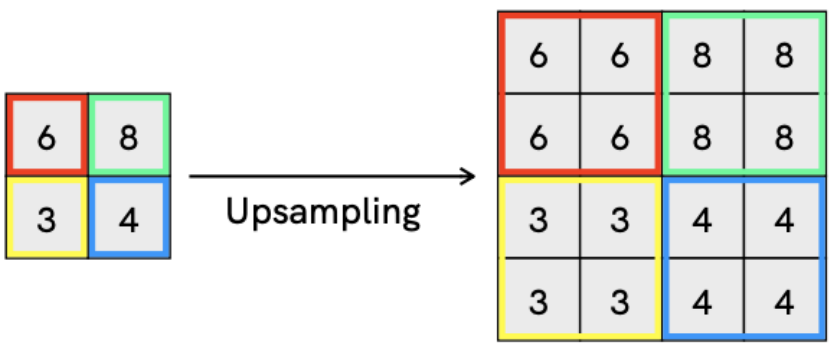
Autoencoder architecture

- A NN architecture is used to impose a *bottleneck* in the network that imposes a compressed representation of the original input
- **Encoding**: Reduce dimensionality of input (e.g. image)
- **Decoding**: Reconstruct the original image from low dimensional **laten space**
- Loss function: minimise difference between **input and output**. For images its per pixel `mse`.



Upsampling

- In a Autoencoder model, the **encoder** down-samples images (through pooling) while the **decoder** needs to up-sample to recover the original image dimension.
- **Upsampling** "reverses" pooling operation



- By first pooling and then upsampling we can compress and then decompress images inside autoencoders.

Encoder in Keras

- The encoding part of the autoencoder contains the convolutional and max-pooling layers to decode the image.
- The max-pooling layer decreases the sizes of the image by using a pooling function.

```
enc_conv1 = Conv2D(12, (3, 3), activation='relu', padding='same')(input_img)
enc_pool1 = MaxPooling2D((2, 2), padding='same')(enc_conv1)
enc_conv2 = Conv2D(8, (4, 4), activation='relu', padding='same')(enc_pool1)
enc_ouput = MaxPooling2D((4, 4), padding='same')(enc_conv2)
```

Decoder in Keras

- The decoding part of the autoencoder contains convolutional and upsampling layers.
- The up-sampling layer helps to reconstruct the sizes of the image. It is the opposite of the pooling function.
- The last convolutional layer holds sigmoid activation.
- Then, we'll combine both layers into the final autoencoder model

```
dec_conv2 = Conv2D(8, (4, 4), activation='relu', padding='same')(enc_ouput)
dec_upsample2 = UpSampling2D((4, 4))(dec_conv2)
dec_conv3 = Conv2D(12, (3, 3), activation='relu')(dec_upsample2)
dec_upsample3 = UpSampling2D((2, 2))(dec_conv3)
dec_output = Conv2D(1, (3, 3), activation='sigmoid',
    padding='same')(dec_upsample3)
```

Autoencoder Applications

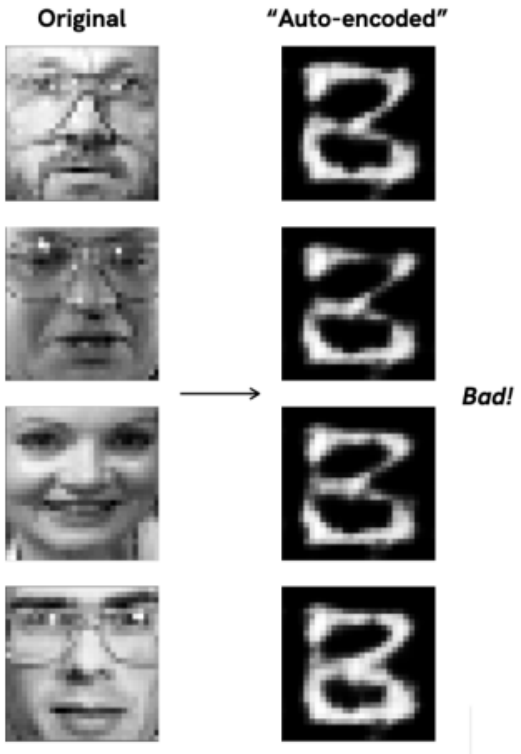
Good image decoders understand feature space of input domain

Anomaly detection

- AE trained on e.g., `numbers` good at decoding numbers
- bad at decoding e.g. `faces`

Noise reduction

- Resolution reduction and upsampling builds in noise reduction



Conclusions

- Introduced computer vision
- Description of Convolutional neural networks
- Description of Autoencoders models

Thank you for your attention and enjoy the checkpoint!

Using Keras Callbacks

Some extra tips to make life a lot easier, though not assessed in this course.

Early stopping

- Stop training when `val_loss` stops improving

```
from tf.keras.callbacks import EarlyStopping,ModelCheckpoint,ReduceLROnPlateau
callbacks=[]

early_stopping = EarlyStopping(monitor="val_loss", min_delta=0.0001, verbose=1)
callbacks.append(early_stopping)
```

Save best model

- NN saved is one with minimum `val_loss`

```
model_checkpoint = ModelCheckpoint(args.model, monitor="val_loss",
    save_best_only=True, verbose=1)
callbacks.append(model_checkpoint)
```

Using Keras Callbacks

Reduce Learning Rate

- Reduce RNN learning rate when training does not improve after certain epochs

```
reduce_lr = ReduceLROnPlateau(patience=4, verbose=1, min_lr=1e-4)
callbacks.append(reduce_lr)
```

Apply callbacks as parameter to fit

```
history_cae = cae.fit(...,callbacks=callbacks)
```

Using Keras Callbacks (Tensorboard)

Another useful feature is tensorboard, for NN and training visualisation.

First from terminal start tensorboard

```
tensorboard --logdir=/tmp/autoencoder
```

Adding TB callback monitors

```
history_cae = cae.fit(x=X_train, y=X_train, epochs=10, validation_split=0.2,
    batch_size=32,callbacks=[TensorBoard(log_dir='/tmp/autoencoder')])
```

Open `http://localhost:6006/` to view your models/fits