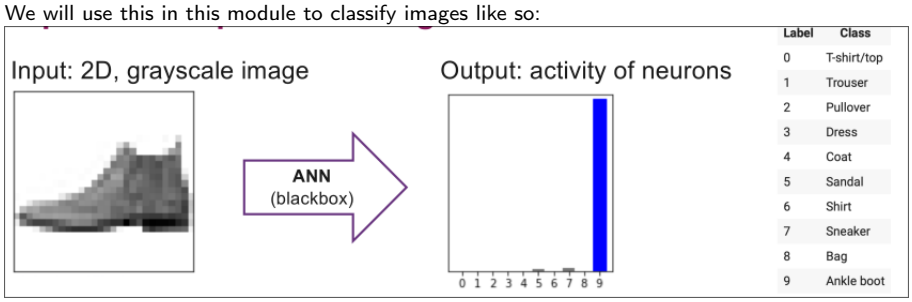
0 Contents

1 CNN Convolutional Neural Networks

1.1 Keras

A python library that wraps tensorflow for classification.
We will use this in this module to classify images like so:



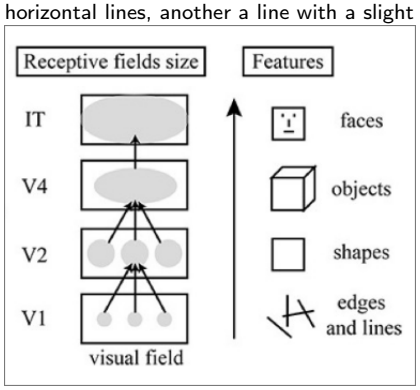
1.2 Flattening

When we convert an image into a long vector, we lose information in the human sense, or rather make it hidden.
todo, explain what is hidden and why

2 Convolution

2.1 Firing of neurons

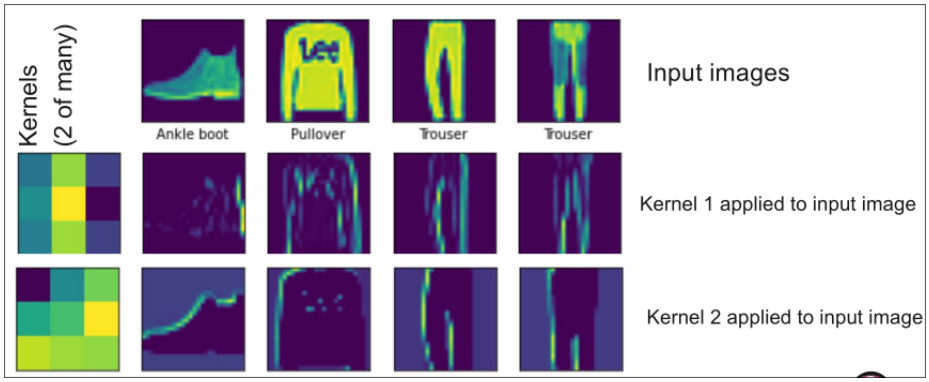
Neurons are clearly connected to something very specific, this would then also be reflected in the artificial neural network. In other words, neuron 1 handles horizontal lines, another a line with a slight angle and so-on.



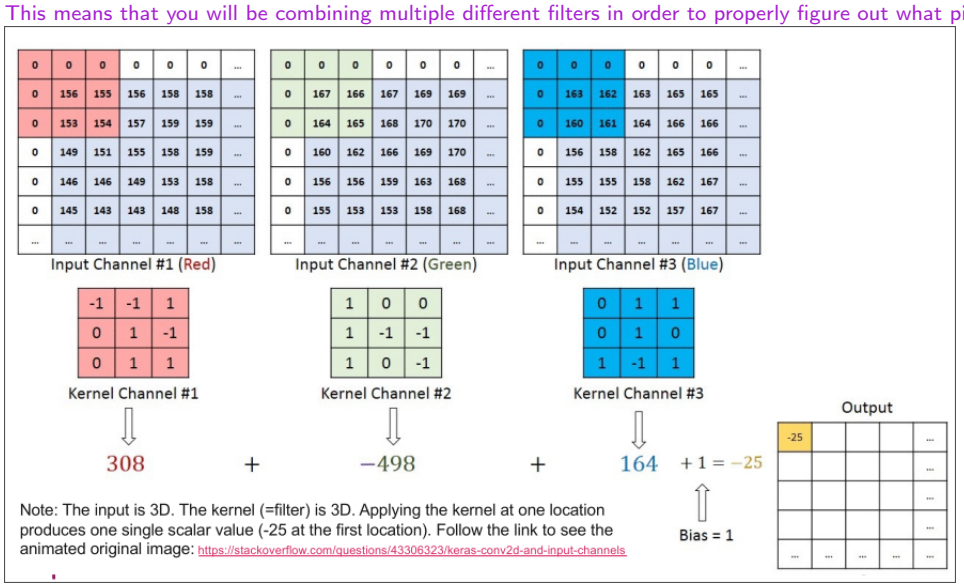
2.2 mathematical model of a feature detector

- Two Inputs
  - a picture
    Note that rgb would give you 3 channels red, green and blue
  - A filter(kernel)
    - an m by n matrix in the simplest case (1 channel, grayscale).
    - an m by n x 3 "stack of matrices" in the case of a 3 channel input (e.g. an RGB image)
    - an m by n x d "stack of matrices". The depth of the kernel must equal the number of input channels.
- One Output
  A feature map (where is the thing that we wanted to search / determine by)
  One convolution produces one feature map. Even if the input and the filter have multiple channels, the output of the convolution has one channel.
- The Operation: Convolution
  We convolve the input image with the convolutional kernel

2.3 Example of convolution



As you can see here, a filter will be used to detect something specific, like a pattern.
This means that you will be combining multiple different filters in order to properly figure out what picture the underlying image is composed of.

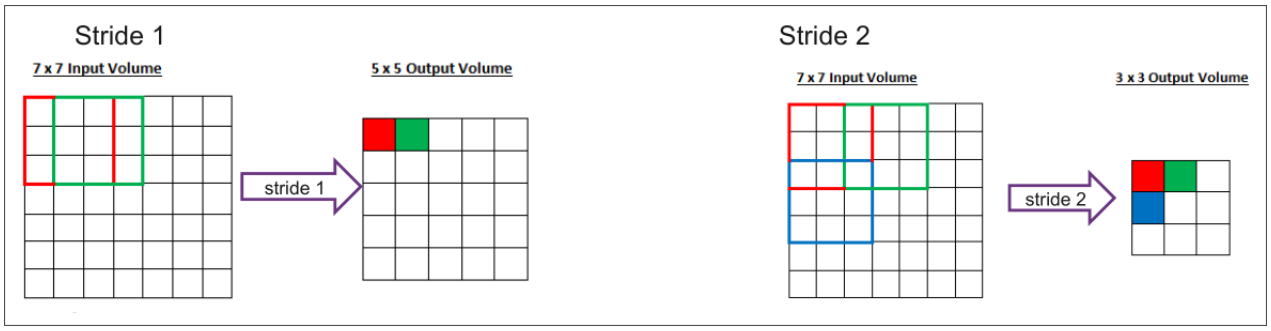


Explanation for the first calculation: 0 * -1 + 0 * -1 + 0 * 1 + 0 * 0 + 156 * 1 + 155 * -1 + 0 * 0 + 153 * 1 + 154 * 1 = 308
multiply each number in the top red square with the number in the same position in the bottom square. We then proceed to do this for all channels (complexity of input, 3 for rgb), which *will then be combined to 1 single output value*.
This output value will then also be combined with a *bias*.
The entire reason we do this, is so that we can have an easier time calculating the images with a pc.

2.4 Reasons for convolution

- Features can be detected independent of location -> filters will always find what they are supposed/created to find
- This calculation is done in parallel, which is very fast for gpus when doing matrix calculations! -> Hence the use of tensorflow with cuda!
- Shared weights mean using the same *kernel values*, this reduces the use of a singular value for each neuron. -> more processing etc

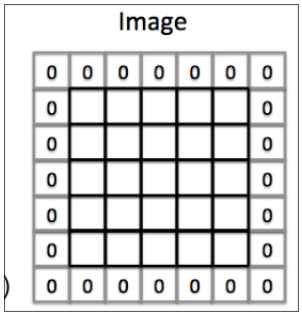
2.5 Stride



Stride is simply the offset by which we move towards the right and the bottom when we move to the next calculation.
The default value here is 1, which means Stride 1.

2.6 Padding

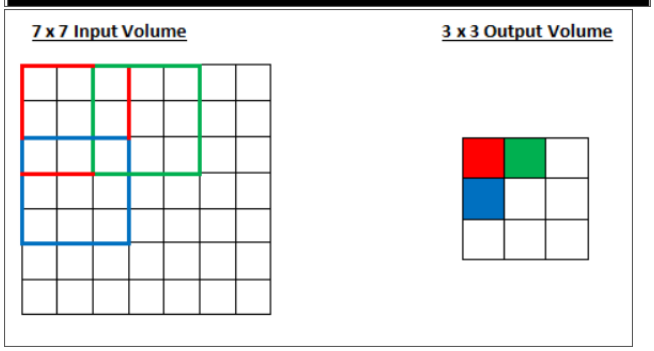
Padding is simply the edge of the matrix that has been predefined.
A same padding is made entirely of 0's!
Note, with padding, we can effectively go over the bounds of the matrix!



2.7 Max Pooling

MaxPooling2D class

```
tf.keras.layers.MaxPooling2D(
    pool_size=(2, 2), strides=None, padding="valid", data_format=None, **kwargs
)
```



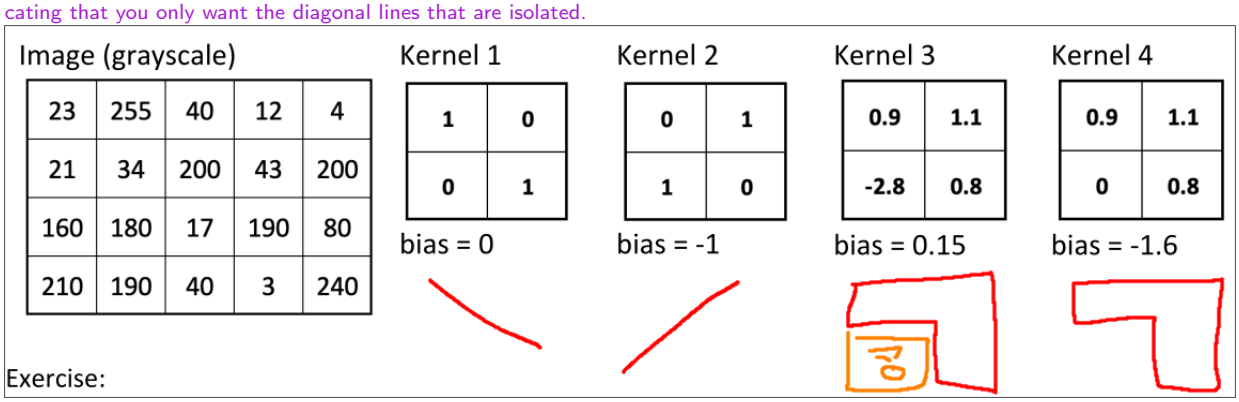
Max pooling simplyfies the image by taking the maximum value in each sector.
For example in the first red square, the highest value will be placed in the right red square.
Similar to before, we then iterate with the stride as the length to iterate and do this again and again until we are done.

2.8 Example with Keras

```
model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

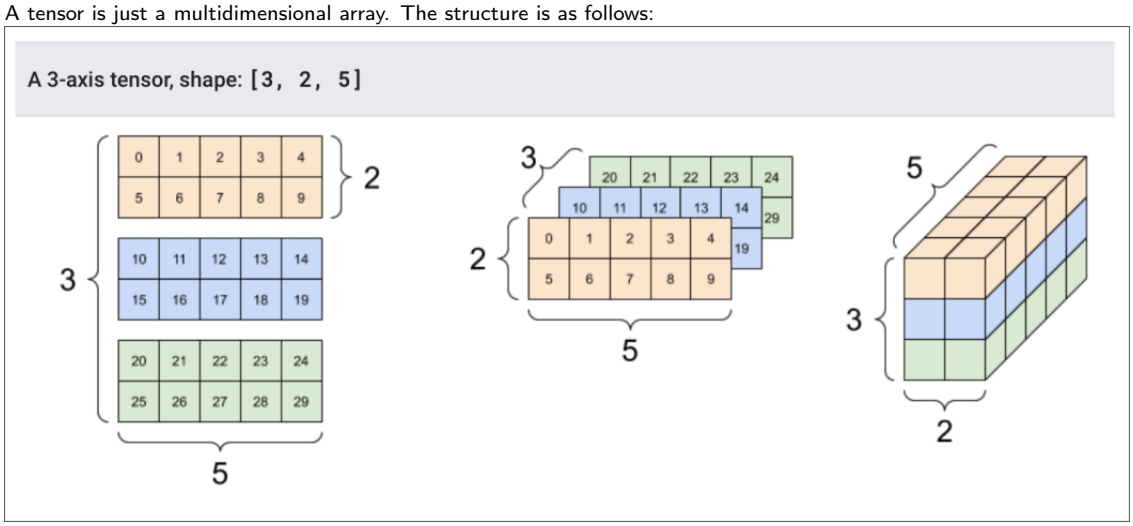
2.9 Why?

The entire filter is essentially just a shape, packed into a matrix.
If you for example have a filter of a 2x2 matrix and you only fill the bottom left and top right square with one and the others with 0, then you will detect whether or not the shape in question has a diagonal line inside of it.
Note that with the others being 0, you will *ignore* the rest of the shape around it. This means you will detect ALL diagonal lines, *even if they are within another shape.*.
Should you want to only detect diagonal lines that have no other color around it, then you need to make the other values in the filter matrix as negative, indicating that you only want the diagonal lines that are isolated.



2.10 Tensor

A tensor is just a multidimensional array. The structure is as follows:



In other words: *amount of sub-arrays, width of sub-array, length of sub-array*
Tensors are immutable, meaning you can only create new ones, you can't update them!
Terms:

- Rank
  The amount of parameters, in the example we had rank3, but what if the sub-arrays had a height as well? -> rank 4!
- Axis or Dimension
  a particular dimension of a tensor, remember 3 dimensions = rank3
  A rank-4 tensor, shape: [3, 2, 4, 5]
- Shape
  The length (number of elements) of each of the axes of a tensor.
- Size
  The total number of items in the tensor, the product of the shape vector's elements.
- Indexing
  simply the indexing of the array like in python.

```
rank_1_tensor = tf.constant([0, 1, 1, 2, 3, 5, 8, 13, 21, 34])
print(rank_1_tensor.numpy())
print("First:", rank_1_tensor[0].numpy())
print("Second:", rank_1_tensor[1].numpy())
print("Last:", rank_1_tensor[-1].numpy())
# First: 0
# Second: 1
# Last: 34
```

- Reshaping
  reshapes a vector by a given list. -> shape is a list!
  Remember that the order and amount of axis need to match! Otherwise you get trash!

```
# Shape returns a 'TensorShape' object that shows the size along each axis
x = tf.constant([[1], [2], [3]])
print(x.shape)
# You can reshape a tensor to a new shape.
# Note that you're passing in a list
reshaped = tf.reshape(x, [1, 3])
print(x.shape)
print(reshaped.shape)
# (3, 1)
# (1, 3)
```

- Broadcasting
  When tensors aren't the same size, you can essentially default extend them, in this case the last value will be taken.

```
x = tf.constant([1, 2, 3])
y = tf.constant(2)
z = tf.constant([2, 2, 2])
# All of these are the same computation
print(tf.multiply(x, 2))
print(x * y)
print(x * z)
# tf.Tensor([2 4 6], shape=(3,), dtype=int32)
# tf.Tensor([2 4 6], shape=(3,), dtype=int32)
# tf.Tensor([2 4 6], shape=(3,), dtype=int32)
```

code tutorial

2.11 An example for image classification

```
# import necessary modules
import tensorflow as tf

from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

# show pictures
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()

# create convolutional base
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 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(64, (3, 3), activation='relu'))

# Add dense layers on top
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))

# Compile and train the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# Evaluate the model
```

```
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

# print the accuracy at the end
print(test_acc)
```

full tutorial