Convolutional Neural Networks

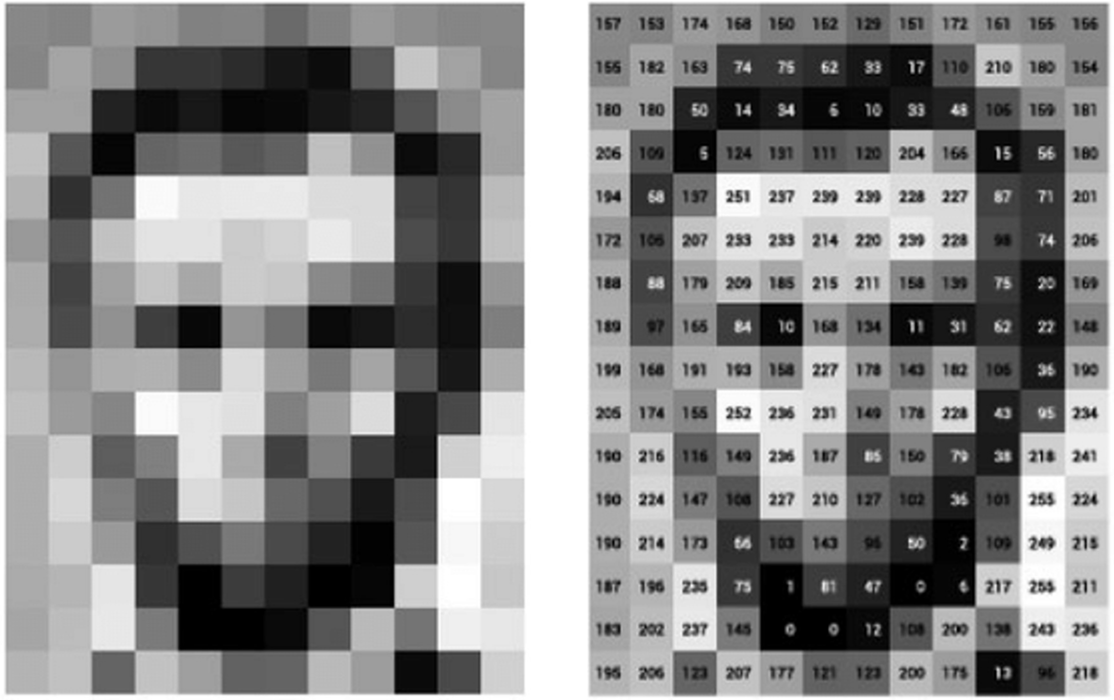
AI+X INSTITUTE FOR ARTIFICIAL INTELLIGENCE

Image Data

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Figure 1. Matrix of pixel values to illustrate how computers see an image.

Image Data



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Figure 2. Matrix of pixel values to illustrate how computers see an image.

Multidimensional Data

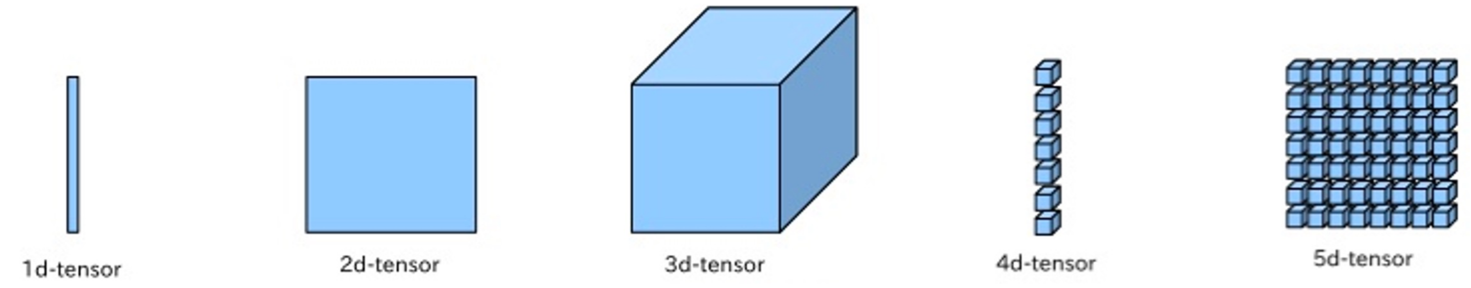


Figure 3. Input data with different dimensionalities. Examples: (1d-tensor) array of measurements; (2d-tensor) grayscale image; (3d-tensor) color image; (4d-tensor) color video; (5d-tensor) collection of videos.

What is a Convolution?

- An operation on two functions (f and g) that produces a third function ($f * g$)

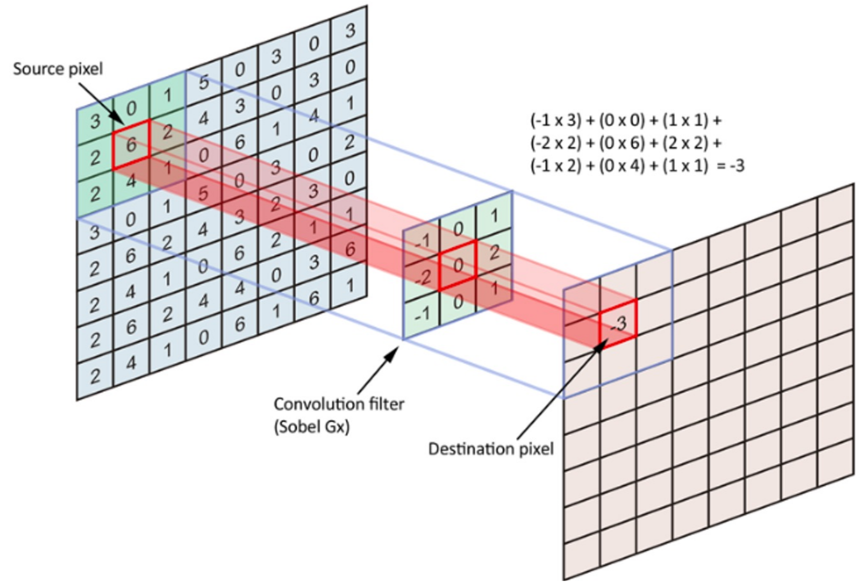


Figure 4. Illustration of a convolution operation.

Convolution Example

- Sobel operator
- 3x3 filters
  - Vertical edges
  - Horizontal edges
- Activation maps

X – Direction Kernel

-1	0	1
-2	0	2
-1	0	1

Y – Direction Kernel

-1	-2	-1
0	0	0
1	2	1

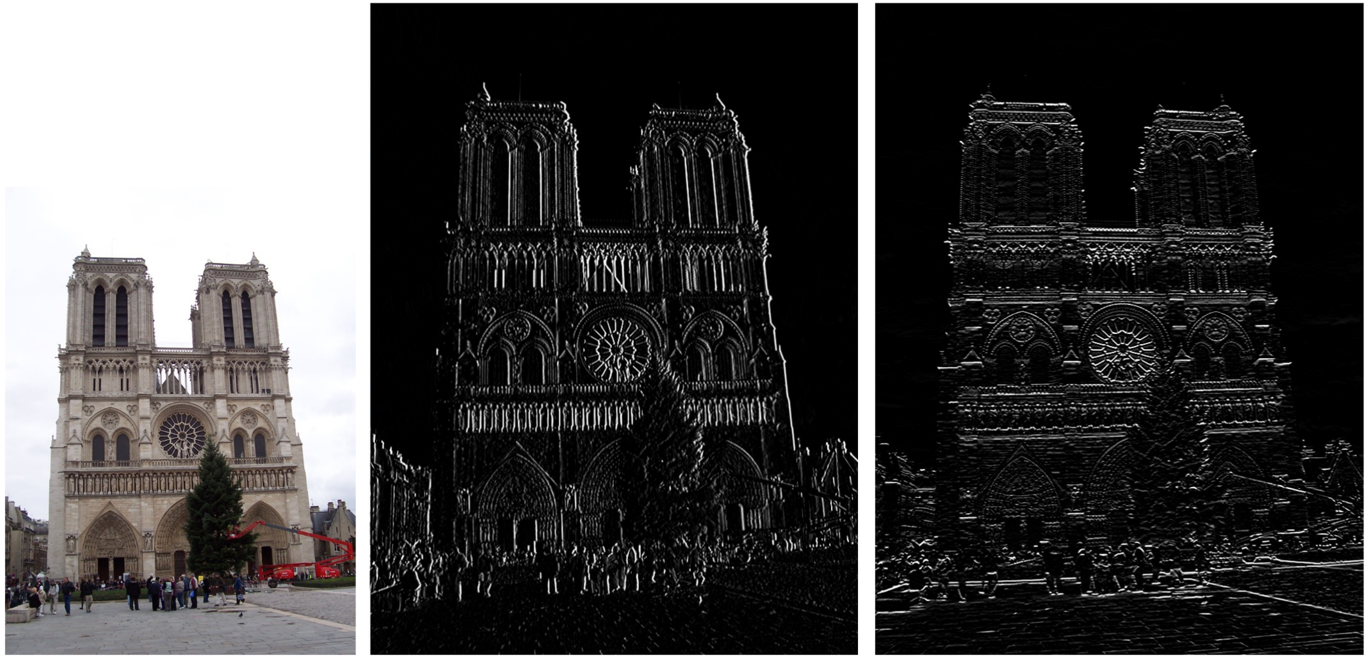


Figure 5. Example of the convolution result for Sobel's horizontal and vertical operators.

Convolution Example

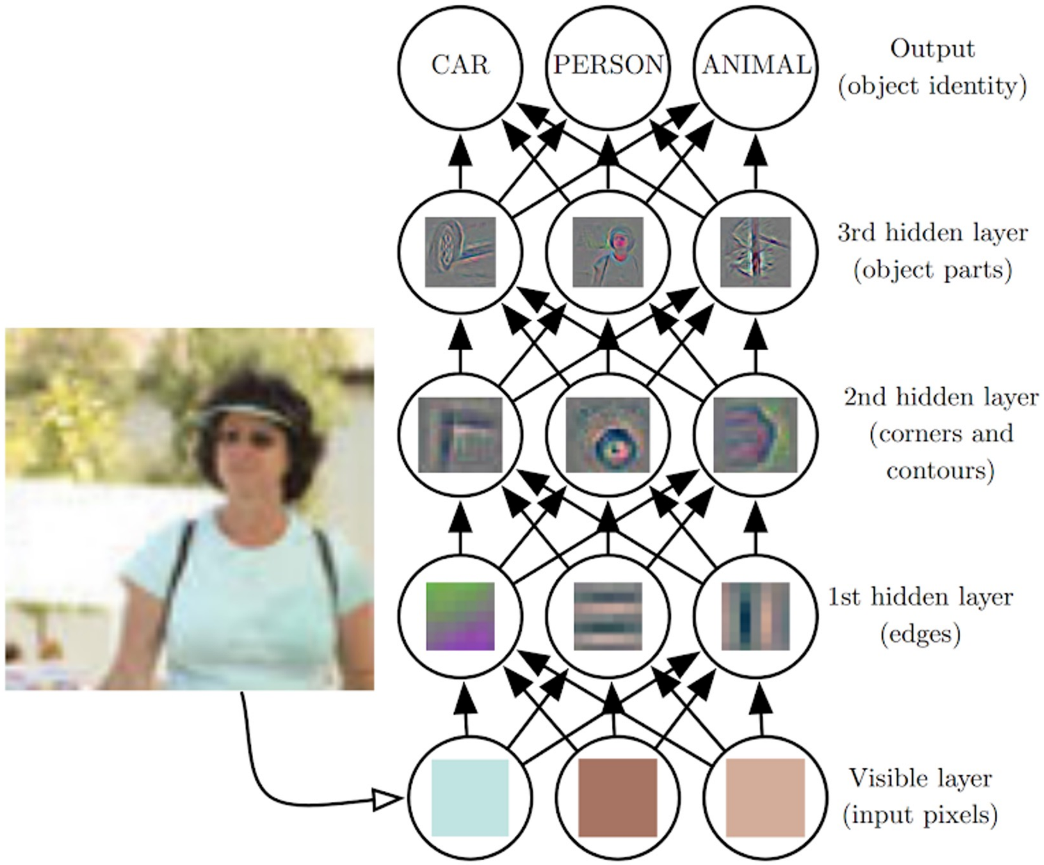


Figure 6. Illustration of a deep learning model.

Convolutions in Keras

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D

```
tf.keras.layers.Conv2D(
        filters,
        kernel_size,
        strides=(1, 1),
        padding='valid',
        data_format=None,
        dilation_rate=(1, 1),
        groups=1,
        activation=None,
        use_bias=True,
        kernel_initializer='glorot_uniform',
        bias_initializer='zeros',
        kernel_regularizer=None,
        bias_regularizer=None,
        activity_regularizer=None,
        kernel_constraint=None,
        bias_constraint=None,
        **kwargs
)
```

Args	
`filters`	Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
`kernel_size`	An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
`strides`	An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
`padding`	one of `"valid"` or `"same"` (case-insensitive). `"valid"` means no padding. `"same"` results in padding with zeros evenly to the left/right or up/down of the input. When `padding="same"` and `strides=1`, the output has the same size as the input.

Input shape

4+D tensor with shape: `batch_shape + (channels, rows, cols)` if `data_format='channels_first'` or 4+D tensor with shape: `batch_shape + (rows, cols, channels)` if `data_format='channels_last'`.

Output shape

4+D tensor with shape: `batch_shape + (filters, new_rows, new_cols)` if `data_format='channels_first'` or 4+D tensor with shape: `batch_shape + (new_rows, new_cols, filters)` if `data_format='channels_last'`. `rows` and `cols` values might have changed due to padding.

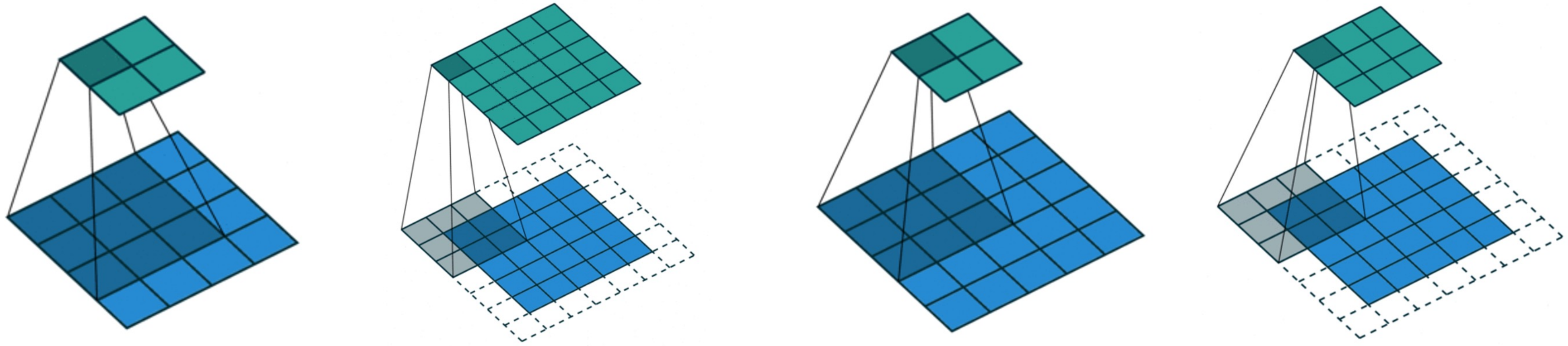
Returns

A tensor of rank 4+ representing `activation(conv2d(inputs, kernel) + bias)`.

Raises	
`ValueError`	if `padding` is `"causal"`.
`ValueError`	when both `strides > 1` and `dilation_rate > 1`.

Figure 7. Convolutions in Keras.

How Convolutions Work?



kernel_size: (3,3)
strides: (1,1)
padding: 'valid'

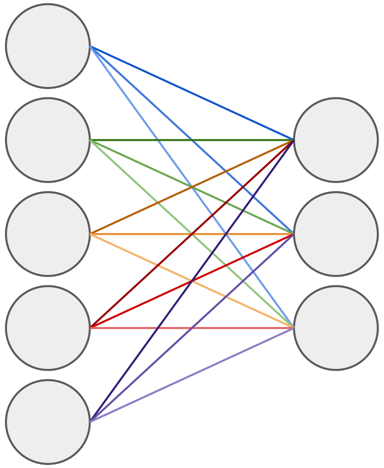
kernel_size: (3,3)
strides: (1,1)
padding: 'same'

kernel_size: (3,3)
strides: (2,2)
padding: 'valid'

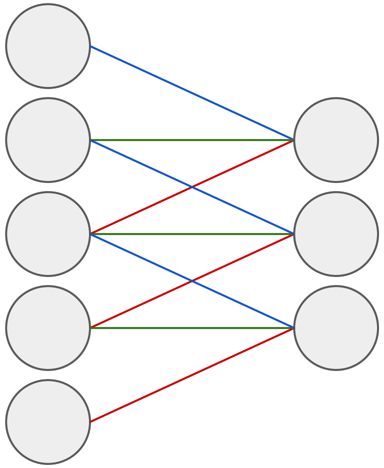
kernel_size: (3,3)
strides: (2,2)
padding: 'same'

Figure 8. Convolution animations. N.B.: Blue maps are inputs, and green maps are outputs.

Dense vs. Convolution



Every output node depends on all input nodes.
|weights| = |input| x |output|

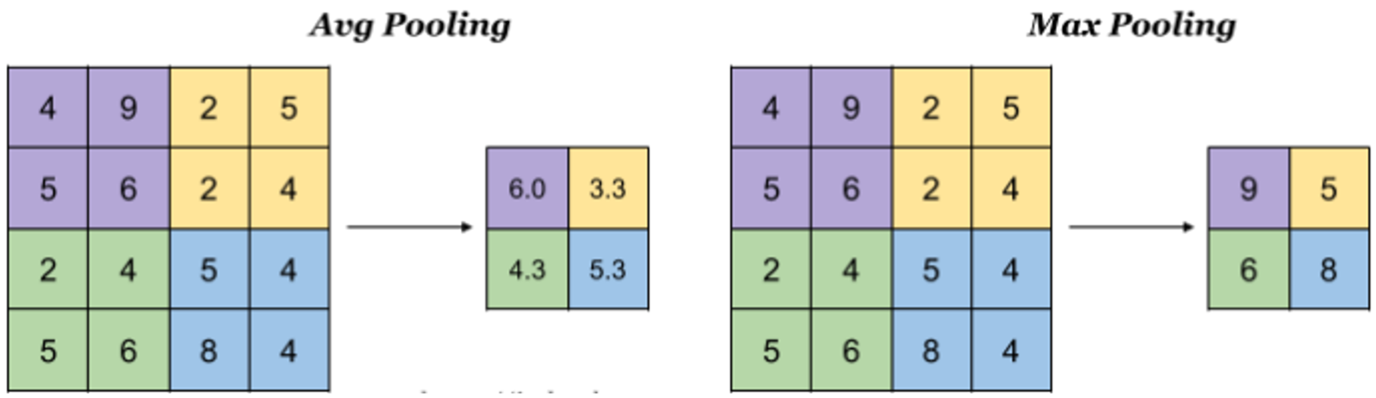


Every output node depends on a small neighborhood.
|weights| = |neighborhood|

Figure 9. Dense vs. convolution.

Pooling

- An aggregation operation that 1) reduces dimensionality, 2) increases the receptive field, and 3) helps on translation invariance.



pool_size: (2,2) / strides: (2,2) / padding: 'valid'

Figure 10. Sample types of pooling are max pooling and average pooling.

Pooling in Keras

https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D

```
tf.keras.layers.MaxPool2D(
        pool_size=(2, 2),
        strides=None,
        padding='valid',
        data_format=None,
        **kwargs
)
```

Args	
`pool_size`	integer or tuple of 2 integers, window size over which to take the maximum. `(2, 2)` will take the max value over a 2x2 pooling window. If only one integer is specified, the same window length will be used for both dimensions.
`strides`	Integer, tuple of 2 integers, or None. Strides values. Specifies how far the pooling window moves for each pooling step. If None, it will default to `pool_size`.
`padding`	One of `"valid"` or `"same"` (case-insensitive). `"valid"` means no padding. `"same"` results in padding evenly to the left/right or up/down of the input such that output has the same height/width dimension as the input.
`data_format`	A string, one of `channels_last` (default) or `channels_first`. The ordering of the dimensions in the inputs. `channels_last` corresponds to inputs with shape `(batch, height, width, channels)` while `channels_first` corresponds to inputs with shape `(batch, channels, height, width)`. It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels_last".

Input shape

- If `data_format='channels_last'`: 4D tensor with shape `(batch_size, rows, cols, channels)`.
- If `data_format='channels_first'`: 4D tensor with shape `(batch_size, channels, rows, cols)`.

Output shape

- If `data_format='channels_last'`: 4D tensor with shape `(batch_size, pooled_rows, pooled_cols, channels)`.
- If `data_format='channels_first'`: 4D tensor with shape `(batch_size, channels, pooled_rows, pooled_cols)`.

Returns

A tensor of rank 4 representing the maximum pooled values. See above for output shape.

Figure 11. Pooling in Keras.

MNIST Dataset

- 10 classes
- 28x28 grayscale images
  - 784 pixels per image

Figure 12. First images from each MNIST class.

MNIST Classification with MLP

```
num_classes = 10

model = tf.keras.models.Sequential()
model.add(tf.keras.Input(shape=(784,)))
model.add(tf.keras.layers.Dense(8, activation='relu'))
model.add(tf.keras.layers.Dense(num_classes, activation='softmax'))

optimizer = tf.keras.optimizers.SGD(lr=0.001)
model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizer,
metrics=['accuracy'])
model.fit(X_train, y_train, batch_size=32, validation_data=(X_val, y_val), epochs=512)
```

CNN Architecture

- LeNet-5

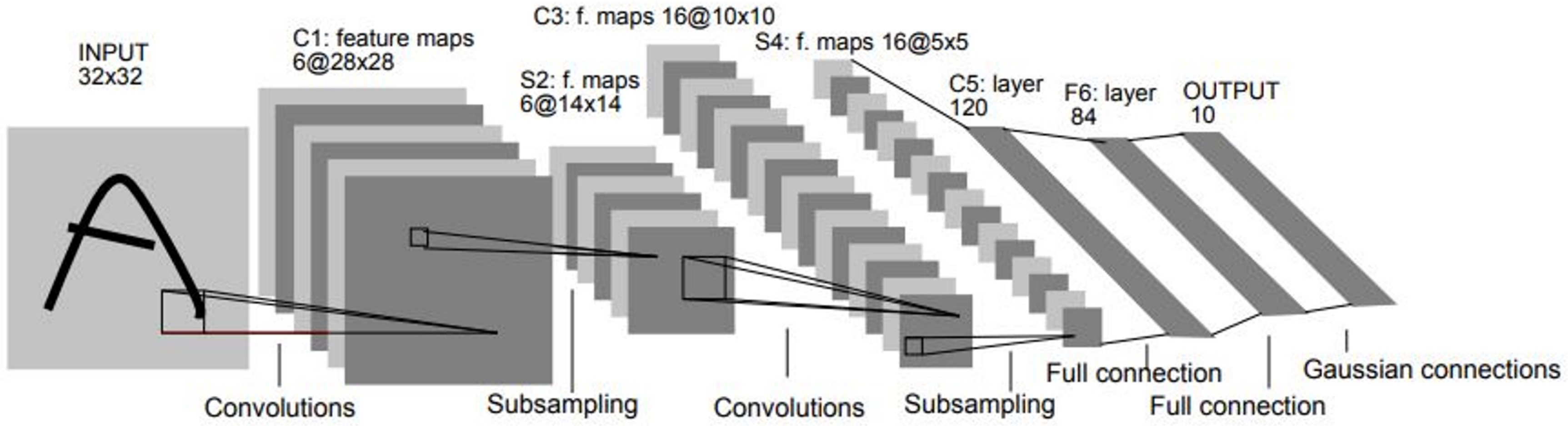


Figure 13. CNN architecture.

MNIST classification with CNN

```
model = tf.keras.models.Sequential()
model.add(tf.keras.Input(shape=(28, 28, 1)))
model.add(tf.keras.layers.Conv2D(filters=6, kernel_size=(5, 5), activation='relu'))
model.add(tf.keras.layers.MaxPool2D())
model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=(5, 5), activation='relu'))
model.add(tf.keras.layers.MaxPool2D())
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(units=120, activation='relu'))
model.add(tf.keras.layers.Dense(units=84, activation='relu'))
model.add(tf.keras.layers.Dense(units=10, activation = 'softmax'))

optimizer = tf.keras.optimizers.SGD(lr=0.001)
model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizer,
metrics=['accuracy'])
model.fit(X_train, y_train, batch_size=32, validation_data=(X_val, y_val), epochs=512)
```

Knowledge Check 1

Which of the options below is an advantage of CNNs over MLPs?

A. Convolutional layers learn feature representations of the entire image.

B. CNNs consider the context information in the small neighborhood, which helps to achieve a better prediction in data like images.

C. CNNs do not require activation functions to operate on nonlinear data.

D. All the above.

You have reached the end of the lecture.

Image/Figure References

Figure 1. Matrix of pixel values to illustrate how computers see an image. Source: Smits, T. & Wevers, M. (2018). The visual digital turn: Using neural networks to study historical images. Digital Scholarship in the Humanities.
Figure 2. Matrix of pixel values to illustrate how computers see an image. Source: Smits, T. & Wevers, M. (2018). The visual digital turn: Using neural networks to study historical images. Digital Scholarship in the Humanities.
Figure 3. Input data with different dimensionalities. Examples: (1d-tensor) array of measurements; (2d-tensor) grayscale image; (3d-tensor) color image; (4d-tensor) color video; (5d-tensor) collection of videos.
Figure 4. Illustration of a convolution operation.
Figure 5. Example of the convolution result for Sobel's horizontal and vertical operators. Source: https://www.cc.gatech.edu/classes/AY2016/cs4476_fall/results/proj2/html/jwang660/index.html
Figure 6. Illustration of a deep learning model. Source: Goodfellow, Bengio and Courville, Deep Learning, MIT Press, 2016.
Figure 7. Convolutions in Keras. Source: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D
Figure 8. Convolution animations. N.B.: Blue maps are inputs, and cyan maps are outputs. Source: https://github.com/vdumoulin/conv_arithmetic
Figure 9. Dense vs. convolution.
Figure 10. Sample types of pooling are max pooling and average pooling. Source: https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/
Figure 11. Pooling in Keras. Source: https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D
Figure 12. First images from each MNIST class.
Figure 13. CNN architecture. Source: LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE.