Statistical Methods in Particle Physics

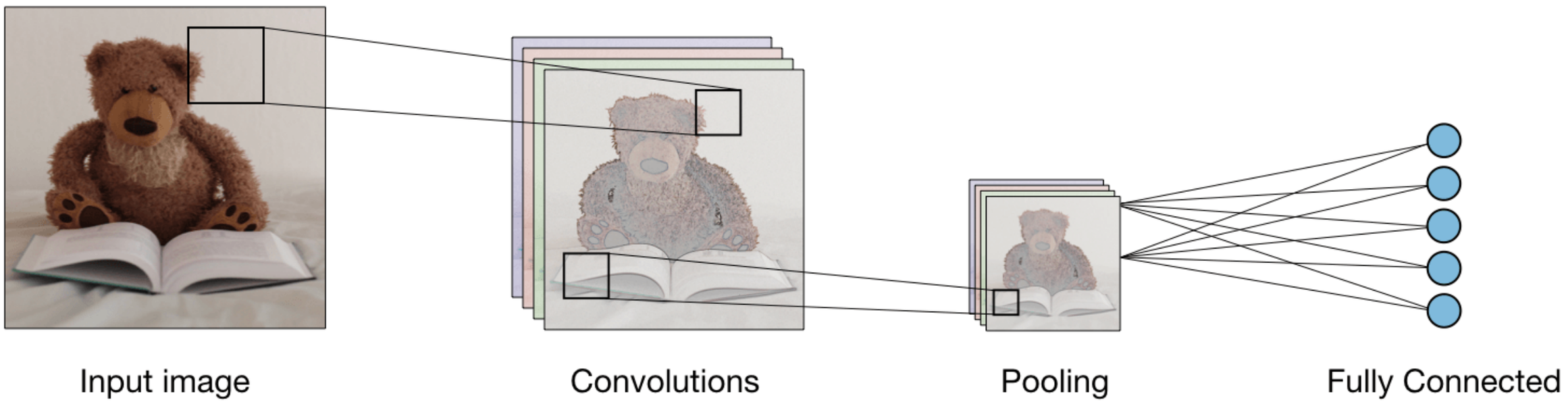
Selected topic 2:
MNIST classification with a simple convolutional neural network using Keras

Heidelberg University, WS 2020/21

Klaus Reygers (lectures)
Rainer Stamen, Martin Völkl (tutorials)

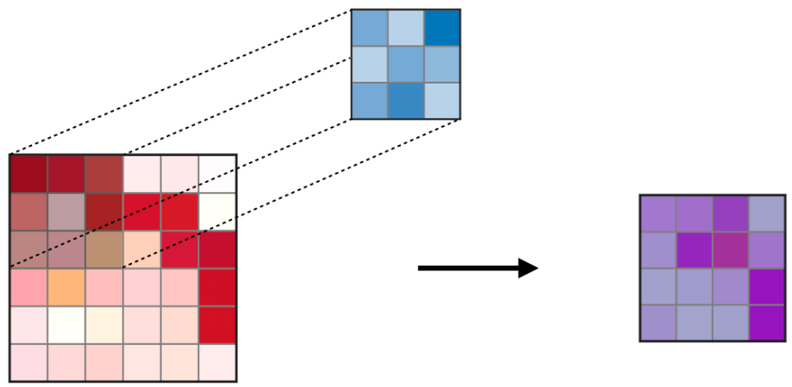
Basic architecture of a convolutional neural network

Afshine Amidi, Shervine Amidi
Convolutional Neural Networks cheatsheet
https://github.com/afshinea/stanford-cs-230-deep-learning/blob/master/en/cheatsheet-convolutional-neural-networks.pdf

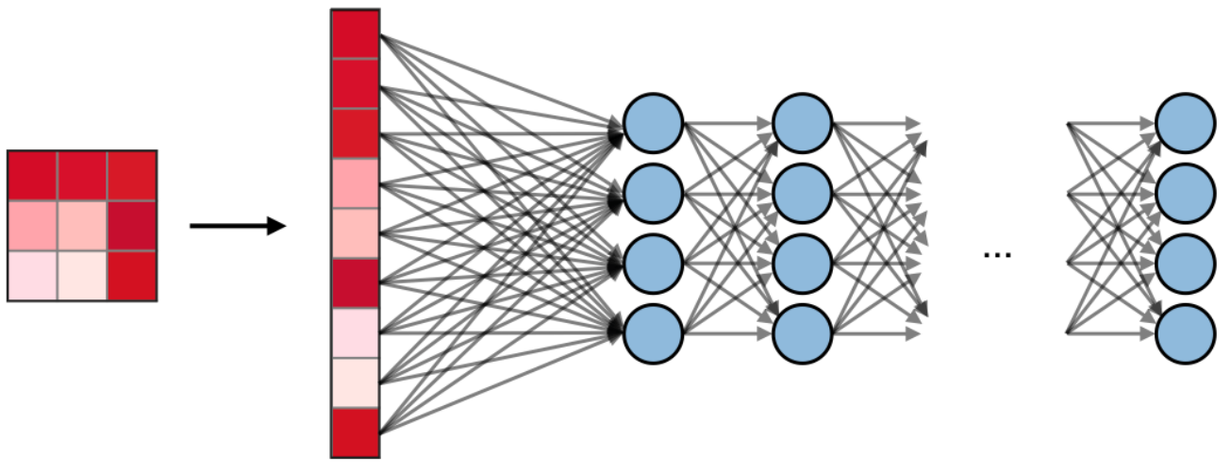


Different types of layers in a CNN

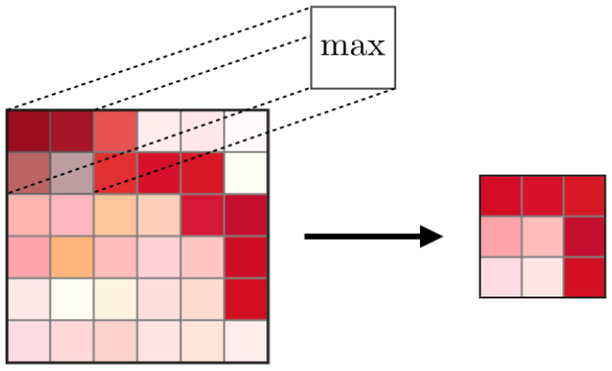
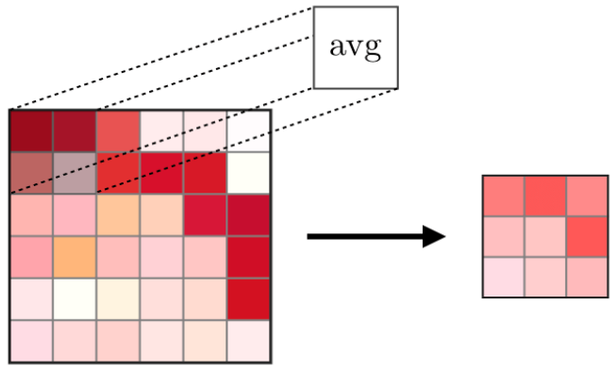
1. Convolutional layers



3. Fully connected layers



2. Pooling layers

	Max pooling	**Average pooling**
Purpose	Each pooling operation selects the maximum value of the current view	Each pooling operation averages the values of the current view
Illustration		
Comments	- Preserves detected features - Most commonly used	- Downsamples feature map - Used in LeNet

Tensorflow and Keras

Example code used in the following from
S. Wunsch, CERN IML TensorFlow/Keras Workshop
https://github.com/stwunsch/iml_tensorflow_keras_workshop

See also Keras website:
https://keras.io/examples/vision/mnist_convnet/

TensorFlow: Low-level implementation of operations needed to implement neural networks in multi-threaded CPU and multi GPU environments

Keras: High-level convenience wrapper for backend libraries, e.g. TensorFlow, to implement neural network models

S. Wunsch, https://github.com/stwunsch/iml_tensorflow_keras_workshop/blob/master/slides/slides.pdf

Defining the CNN in Keras

```
from keras.models import Sequential
from keras.layers import Dense, Flatten, MaxPooling2D, Conv2D, Input, Dropout

# conv layer with 8 3x3 filters

model = Sequential(
    [
        Input(shape=input_shape),
        Conv2D(8, kernel_size=(3, 3), activation="relu"),
        MaxPooling2D(pool_size=(2, 2)),
        Flatten(),
        Dense(16, activation="relu"),
        Dense(num_classes, activation="softmax"),
    ]
)
```

See mnist_keras_train.ipynb and mnist_keras_apply.ipynb on lecture web page.

For performance comparison: simple softmax regression in mnist_softmax_regression.ipynb.

CNN model summary

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 8)         80
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 8)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1352)              0
_________________________________________________________________
dense_2 (Dense)              (None, 16)                21648
_________________________________________________________________
dense_3 (Dense)              (None, 10)                170
=================================================================
Total params: 21,898
Trainable params: 21,898
Non-trainable params: 0
_________________________________________________________________
```

Model training

Compile the model

Using Keras, you have to `compile` a model, which means adding the loss function, the optimizer algorithm and validation metrics to your training setup.

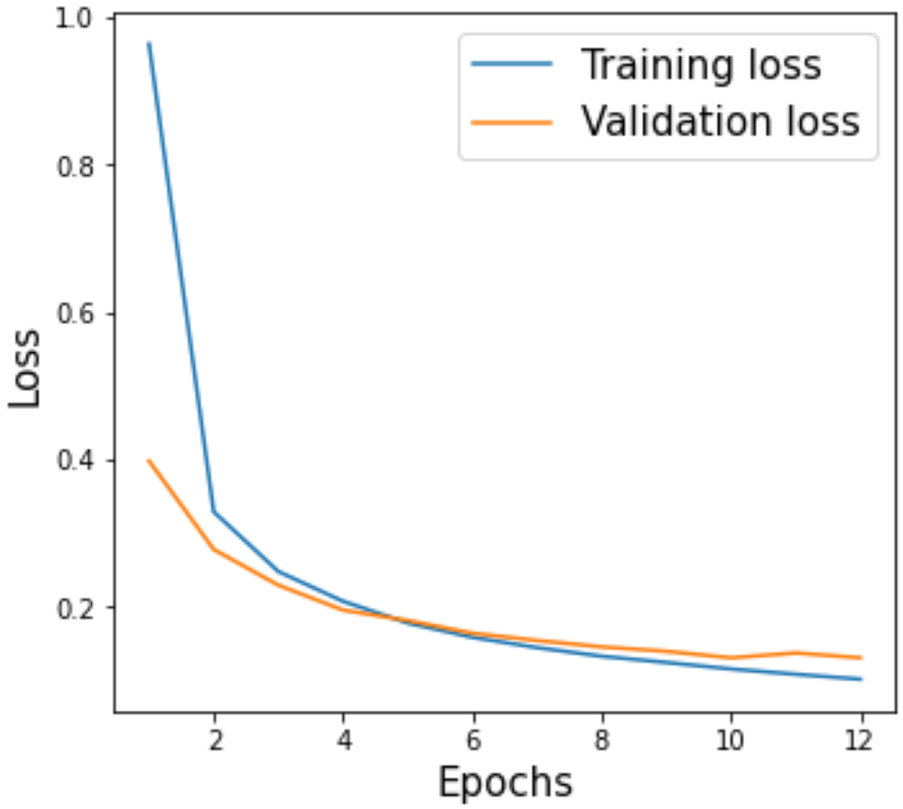
```
model.compile(loss="categorical_crossentropy",
        optimizer="adam",
        metrics=["accuracy"])
```

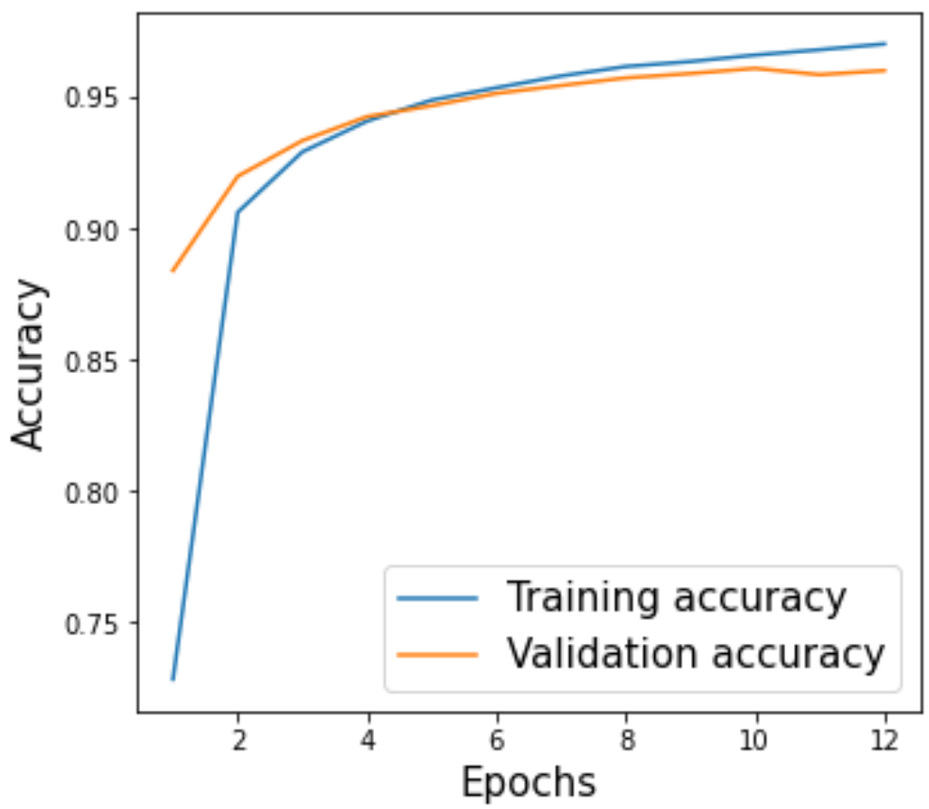
Train the model

```
from keras.callbacks import ModelCheckpoint, EarlyStopping

checkpoint = ModelCheckpoint(
             filepath="mnist_keras_model.h5",
             save_best_only=True,
             verbose=1)
early_stopping = EarlyStopping(patience=2)

history = model.fit(x_train, y_train, # Training data
             batch_size=200, # Batch size
             epochs=50, # Maximum number of training epochs
             validation_split=0.5, # Use 50% of the train dataset for validation
             callbacks=[checkpoint, early_stopping]) # Register callbacks
```

```
Epoch 1/50
150/150 [==============================] - 9s 51ms/step - loss: 1.5244 - accuracy: 0.5341 - val_loss:
0.3984 - val_accuracy: 0.8842

Epoch 00001: val_loss improved from inf to 0.39840, saving model to mnist_keras_model.h5
Epoch 2/50
150/150 [==============================] - 6s 43ms/step - loss: 0.3608 - accuracy: 0.8961 - val_loss:
0.2784 - val_accuracy: 0.9201
```

Loss and accuracy vs. number of epochs





Test the model

The prediction of unseen data is performed using the `model.predict(inputs)` call. Below, a basic test of the model is done by calculating the accuracy on the test dataset.

This simple CNN achieves about 97% accuracy. Adding one convolutional layer increases accuracy to about 99% (see Keras website) Accuracy with simple softmax regression is 92%.

```
# Get predictions on test dataset
y_pred = model.predict(x_test)

# Compare predictions with ground truth
test_accuracy = np.sum(
        np.argmax(y_test, axis=1)==np.argmax(y_pred, axis=1))/float(x_test.shape[0])

print("Test accuracy: {}".format(test_accuracy))
```

```
Test accuracy: 0.9655
```

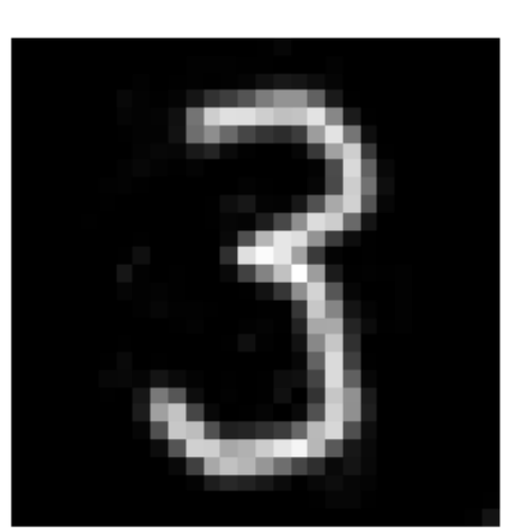
Test the trained model

Load the model

Loading a Keras model needs only a single line of code, see below. After this call, the model is back in the same state you stored it at the training step either by the `ModelCheckpoint` or `model.save(...)`.

```
model = load_model("mnist_keras_model.h5")
```

```
f = "mnist_my_digit_3.png"
image = np.zeros((1, 28, 28, 1), dtype=np.uint8)
pngdata = png.Reader(open(f, 'rb')).asDirect()
for i_row, row in enumerate(pngdata[2]):
    image[0, i_row, :, 0] = row

prediction_vector = model.predict(image)
prediction = np.argmax(prediction_vector)
print (f"Model prediction for each class: {prediction_vector}")
print (f"Predicted digit: {prediction}")
plt.axis('off')
plt.imshow(np.squeeze(image), cmap="gray");
```



```
Model prediction for each class: [[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]]
Predicted digit: 3
```