University of Applied Sciences

HOCHSCHULE
EMDEN·LEER

Convolution Neural Networks for Image Analysis

Prof. Dr. E. Wings

Agenda

1. Image and Object Classification
2. How it works
3. Remarks
4. Example MNIST

Image and Object Classification

Unstructured Data: Images and Video

There are a number of application areas such defense and security, weather analysis, telecommunication, transportation, entertainment, etc., that generate high-dimensional data such as images and videos.

$\Rightarrow$ Use of Convolution Neural Networks

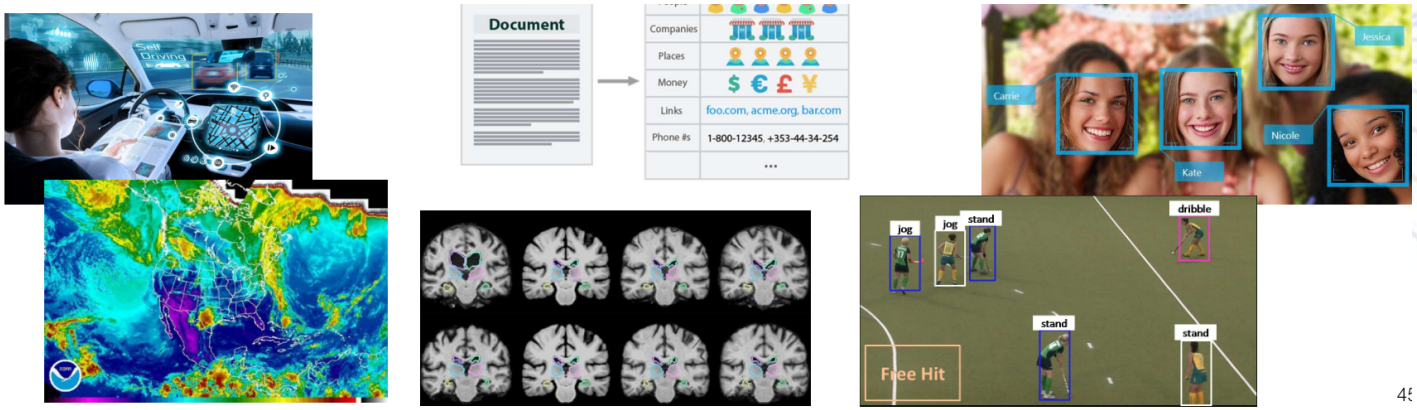


Image Classification

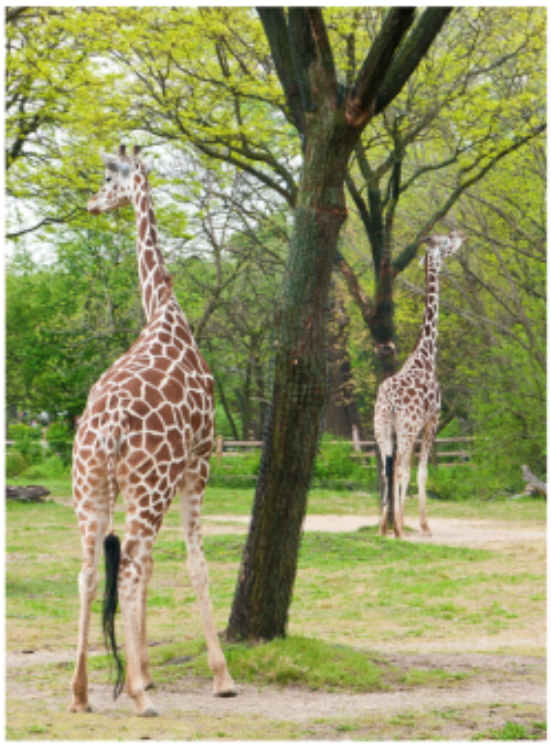
There are two basic ways to view images:

1. Description of an image: Image classification
2. Identification of objects in the images: Object classification

Image Classification

a soccer player is kicking a soccer ball

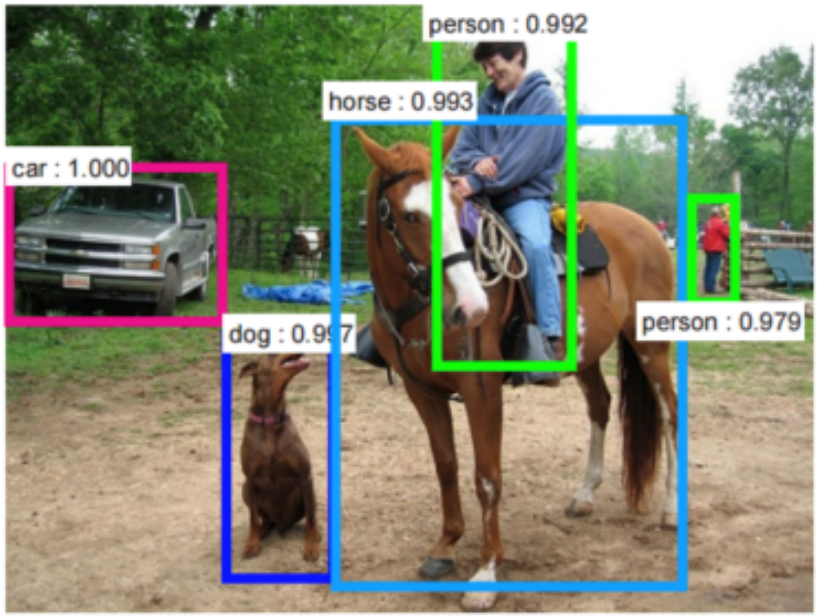
a street sign on a pole in front of a building

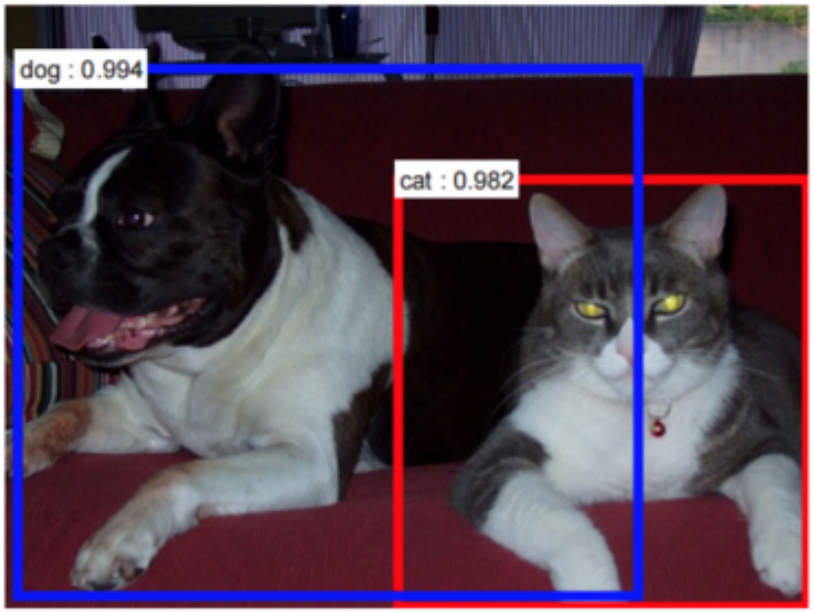


a couple of giraffe standing next to each other

Source: ujjwalkarn - An Intuitive Explanation of Convolutional Neural Networks, 2016

Object Classification





Source: ujjwalkarn - An Intuitive Explanation of Convolutional Neural Networks, 2016

Try it

Try YOLO: You only look once (YOLO) is a state-of-the-art, real-time object detection system.

YOLO

Digital Representation of Images

Computers "see" the world differently, in the format of digits.

How a Human sees an image

How a computer sees an image

```
[  9   1  29  70 114  76   0   8   4   5   5   0 111 162   9   8  62  62]
[  3   0  33  61 102 106  34   0   0   0   0  49 182 150   1  12  65  62]
[  1   0  40  54 123  90  72  77  52  51  49 121 205  98   0  15  67  59]
[  3   1  41  57  74  54  96 181 220 170  90 149 208  56   0  16  69  59]
[  6   1  32  36  47  81  85  90 176 206 140 171 186  22   3  15  72  63]
[  4   1  31  39  66  71  71  97 147 214 203 190 198  22   6  17  73  65]
[  2   3  15  30  52  57  68 123 161 197 207 200 179   8   8  18  73  66]
[  2   2  17  37  34  40  78 103 148 187 205 225 165   1   8  19  76  68]
[  2   3  20  44  37  34  35  26  78 156 214 145 200  38   2  21  78  69]
[  2   2  20  34  21  43  70  21  43 139 205  93 211  70   0  23  78  72]
[  3   4  16  24  14  21 102 175 120 130 226 212 236  75   0  25  78  72]
[  6   5  13  21  28  28  97 216 184  90 196 255 255  84   4  24  79  74]
[  6   5  15  25  30  39  63 105 140  66 113 252 251  74   4  28  79  75]
[  5   5  16  32  38  57  69  85  93 120 128 251 255 154  19  26  80  76]
[  6   5  20  42  55  62  66  76  86 104 148 242 254 241  83  26  80  77]
[  2   3  20  38  55  64  69  80  78 109 195 247 252 255 172  40  78  77]
[ 10   8  23  34  44  64  88 104 119 173 234 247 253 254 227  66  74  74]
[ 32   6  24  37  45  63  85 114 154 196 226 245 251 252 250 112  66  71]
```

People can see “selectively”:

▸ Concentration test

An Image is a Matrix of Pixel Values - Channels

Channel is a conventional term used to refer to a certain component of an image.

An image from a standard digital camera will have three channels:

- red
- green
- blue

You can imagine those as three 2D-matrices stacked over each other (one for each color), each having pixel values in the range 0 to 255. Sometimes it's called tensor.

An Image is a Matrix of Pixel Values - Grayscale Image

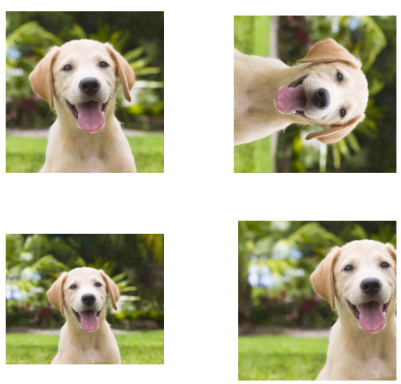
A grayscale image, on the other hand, has just one channel.

The value of each pixel in the matrix will range from 0 to 255. (zero indicating black and 255 indicating white)

Challenges of Working with Images

- High dimensionality
  - Size: 200 x 200 pixels = 40,000 pixels
  - Colour, 3 channels for red, greed and blue

  $\Rightarrow$ $40,000 \times 3 = 120,000$ attributes
- Rotation
- Scale
- Location



How it works

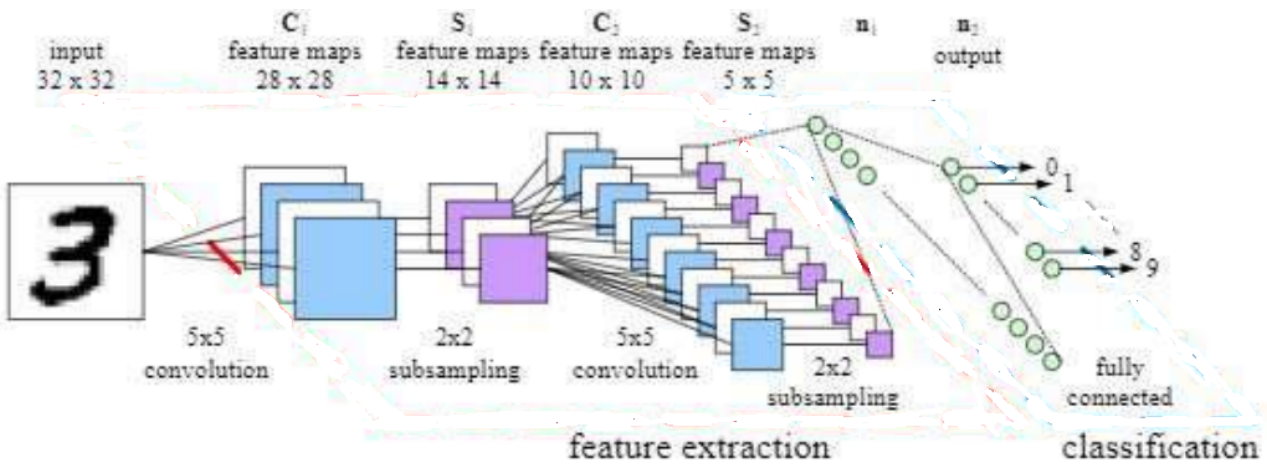
Workings of CNN

- A CNN is composed of two major parts:
  1. feature extraction and
  2. classification.
- Feature extraction is the process of automatically extracting features from the images.
- Features can be understood as attributes of the image, e.g.,
  1. an image of a cat might have features like whiskers, two ears, four legs etc..
  2. A handwritten digit image might have features as horizontal and vertical lines or loops and curves.
- Classification is the aspect related to prediction of an outcome.

Workings of CNN

A CNN consists of two sections:

1. Feature Extraction
   - Convolution
   - Non Linearity (ReLU)
   - Pooling or SubSampling
2. Classification (Fully Connected Layer)

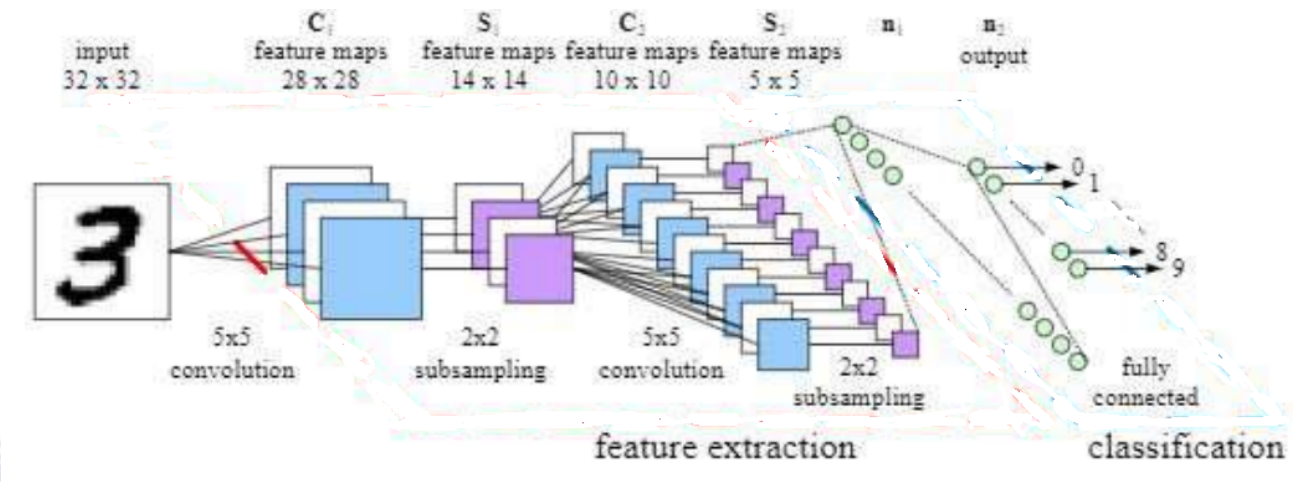


Source: Mesman et al. 2011

Workings of CNN

A CNN consists of two sections:

1. Feature Extraction
   - Convolution
   - Non Linearity (ReLU)
   - Pooling or SubSampling
2. Classification (Fully Connected Layer)

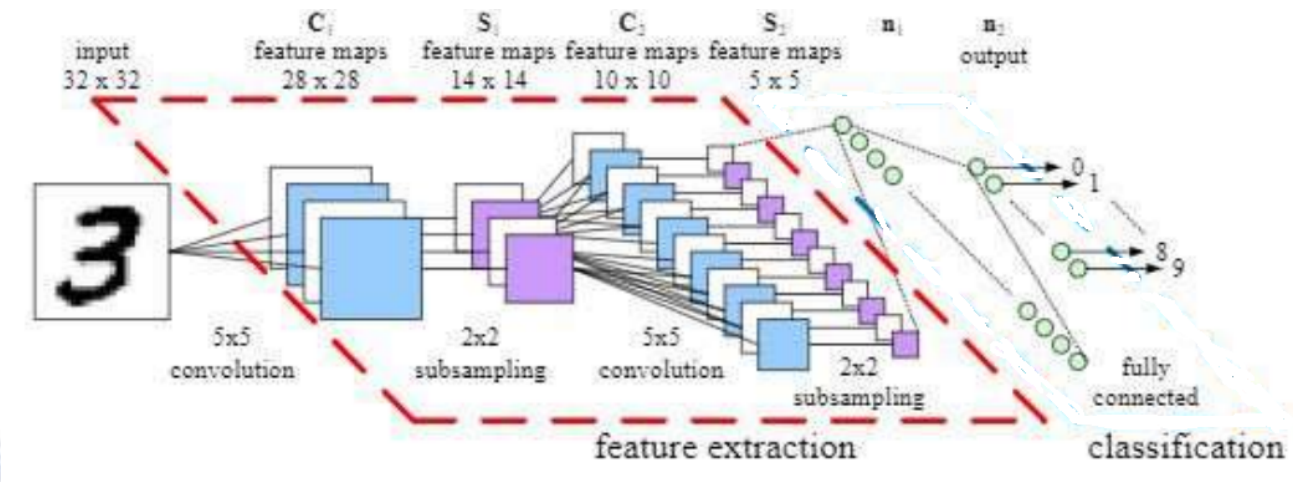


Source: Mesman et al. 2011

Workings of CNN

A CNN consists of two sections:

1. Feature Extraction
   - Convolution
   - Non Linearity (ReLU)
   - Pooling or SubSampling
2. Classification (Fully Connected Layer)

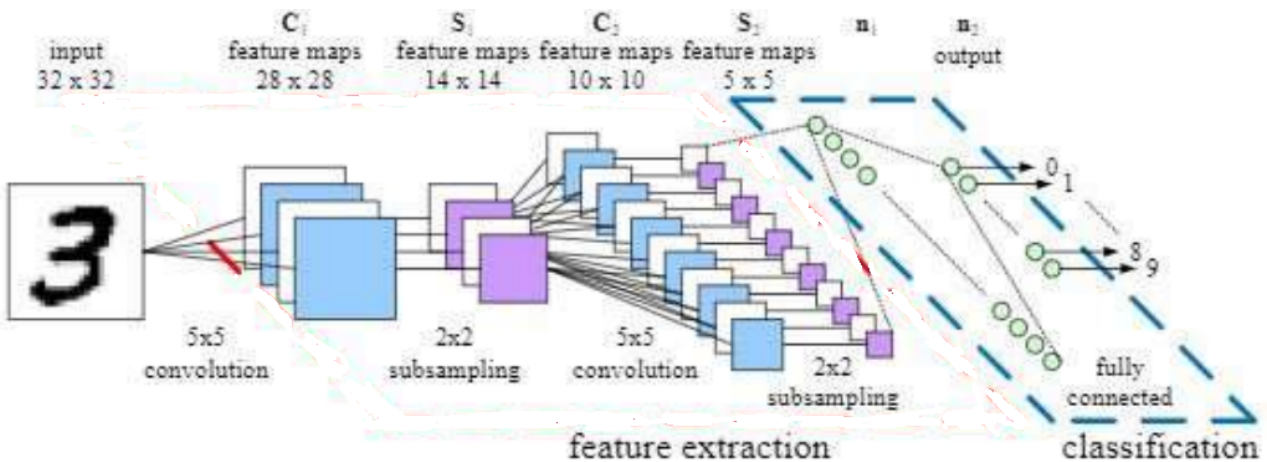


Source: Mesman et al. 2011

Workings of CNN

A CNN consists of two sections:

1. Feature Extraction
   - Convolution
   - Non Linearity (ReLU)
   - Pooling or SubSampling
2. Classification (Fully Connected Layer)



Source: Mesman et al. 2011

Convolutional Neural Network - Input

The input of a CNN is a tensor or a matrix, not a vector.

CNN - Convolutional Layer

In the first layer of a convolutional neural network a convolutional layer with 16 or 32 filters is mostly used, whose folded outputs are new matrices.

This first layer is usually followed by a second, equally constructed convolutional layer, which uses the new matrices from the convolution of the first layer as input. Then follows a Pooling Layer.

CNN - Convolutional Layer

The filter has a fixed weight for each point in its viewing window, and it calculates a result matrix from the pixel values in the current viewing window and these weights. The size of this result matrix depends on the size (kernel size) of the filter, the padding and especially on the step size.

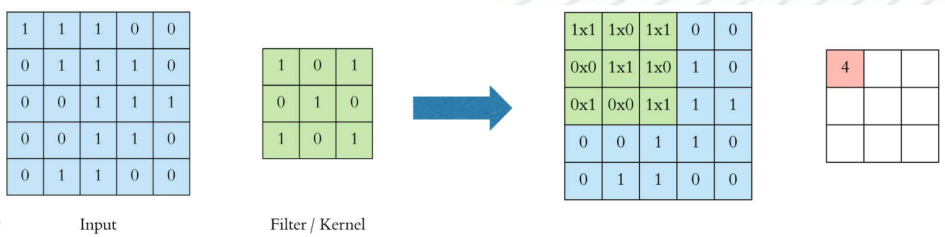
CNN - Convolutional Layer

A step size of 2 for a kernel size of 2×2, for example, halves the size of the result matrix per filter compared to the input matrix.

In this case, each pixel is no longer connected to the filter individually, but 4 pixels are connected to the filter at the same time (local connectivity). The input was then "folded" (convolution).

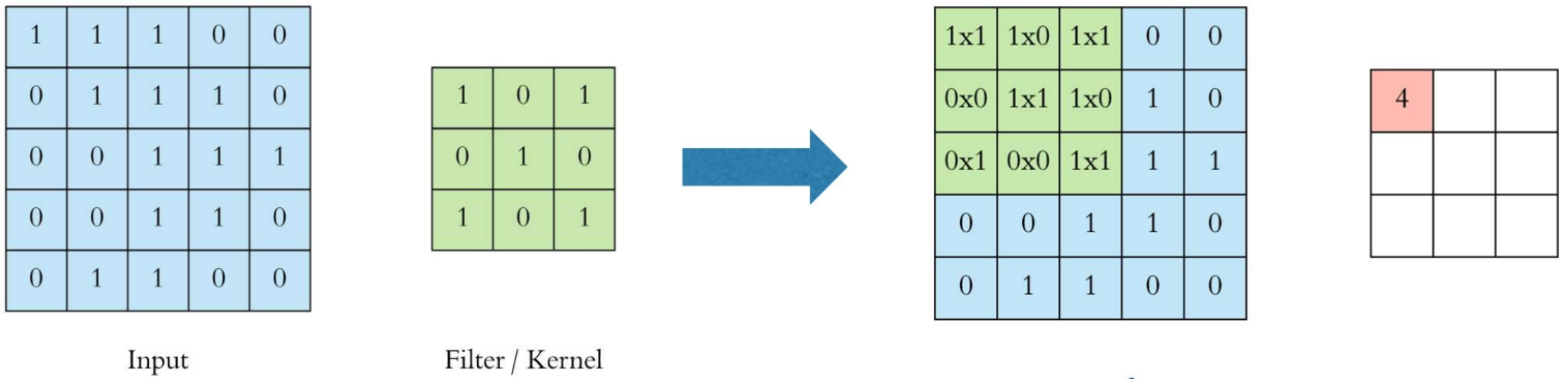
CNN: Feature Extraction

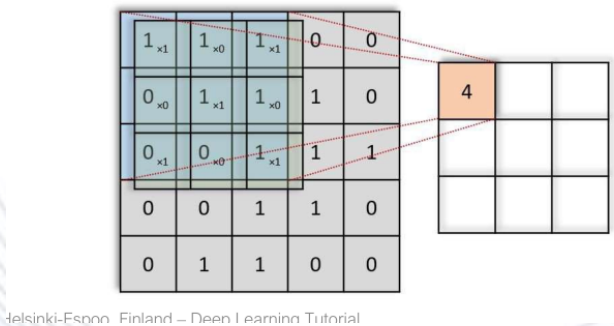
- Feature extraction of a CNN is conducted using three types of operations.
  1. Convolution,
  2. Non-Linearity and
  3. Pooling.
- Convolution is the mathematical operation which is central to the efficacy of this algorithm.
- Convolution operation is performed on an input image using a filter or a kernel.

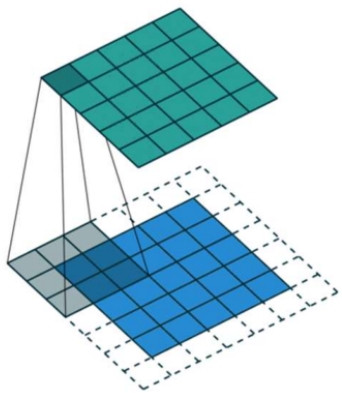


Source: Karkare 2019

CNN: Feature Extraction - Convolution Operation







Source: Kark

Source: Karkare 2019

CNN: Feature Extraction - Convolution Operation

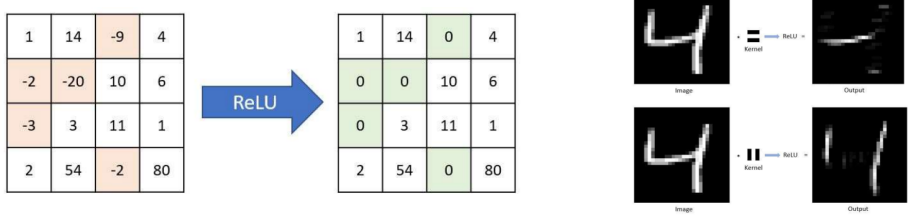
Horizontal Edge Filter

Vertical Edge Filter

Source: Karkare 2019

CNN: Feature Extraction - Non-Linearity

- After sliding our filter over the original image, the output which we get is passed through another mathematical function which is called an activation function.
- This is the same activation function ReLU (Rectified Linear Unit) we used for the Deep Neural Network.
- Makes this computationally inexpensive than the standard activation functions, such as sinh or tanh.



Source: Karkare 2019

CNN: Feature Extraction - Pooling

- After a convolution and ReLU operations, a common practice is to add a pooling layer in CNN.
- Pooling reduces the dimensionality to reduce the number of parameters and computation in the network.
- The most frequent type of pooling is max pooling, which takes the maximum value in a specified window

Pooling Layer

There are two types of pooling:

- Max Pooling
- Average Pooling

Max Pooling

56	12	53	94
78	67	64	24
42	62	73	83
43	51	14	57

$\xrightarrow{\text{Maximum}}$

78	94
62	83

In the example, a kernel of size $n \times n$ (2×2 in the above example) is moved across the matrix and for each position the max value is taken and put in the corresponding position of the output matrix.

Average Pooling

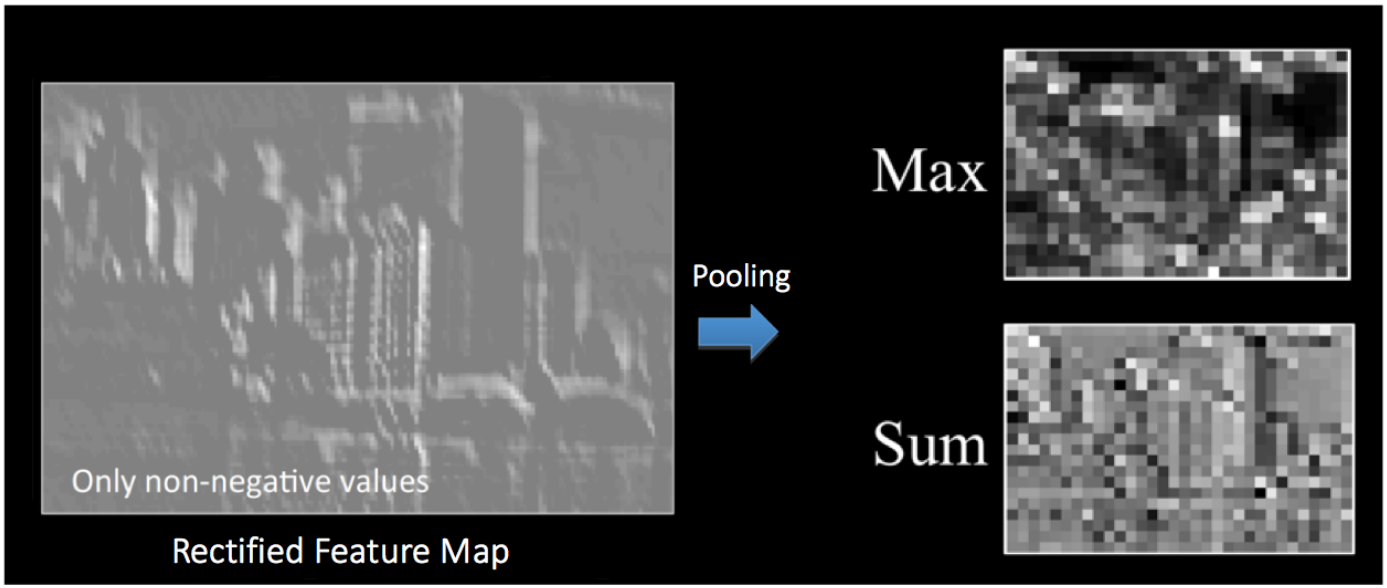
56	12	53	94
78	67	64	24
42	62	73	83
43	51	14	57

Average →

53	59
50	57

A kernel of size $n \times n$ is moved across the matrix and for each position the average is taken of all the values and put in the corresponding position of the output matrix.

Pooling


Only non-negative values
Rectified Feature Map
Pooling
Max
Sum

Pooling Layer

The main purpose of a pooling layer is to reduce the number of parameters of the input tensor:

- → Helps reduce overfitting
- → Extract representative features from the input tensor
- → Reduces computation and thus aids efficiency

The input to the pooling layer is a tensor.

Pooling Layer

This is repeated for each channel in the input tensor. And so we get the output tensor.

$\rightarrow$ Pooling downsamples the image in its height and width but the number of channels (depth) stays the same.

Flatten

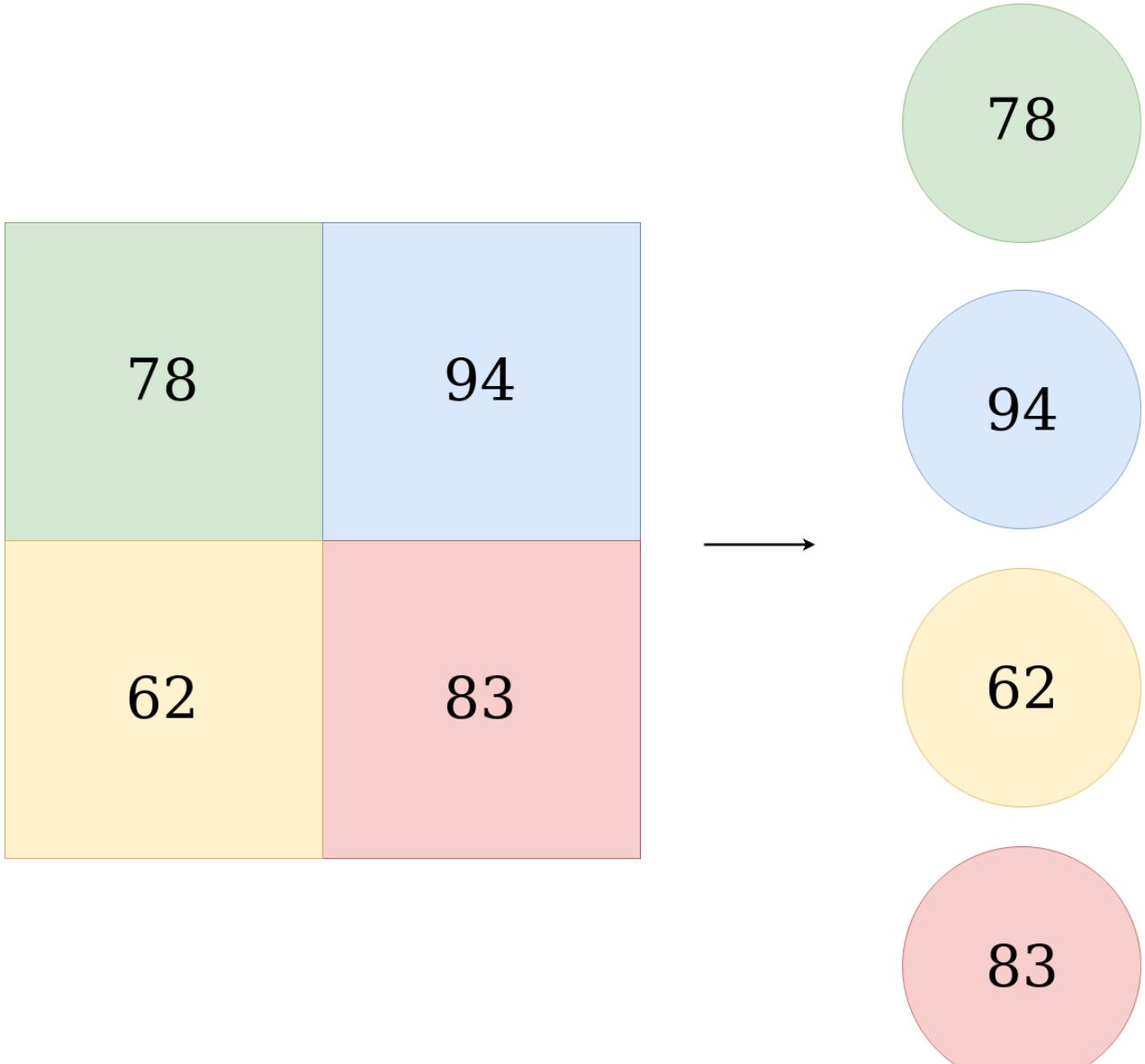
The previous input values are tensors or matrices; the neural network expects vectors.

$\Rightarrow$ Flatten converts to vectors.

Flatten

The output from the final (and any) Pooling and Convolutional Layer is a 3-dimensional matrix, to flatten that is to unroll all its values into a vector.

Flatten

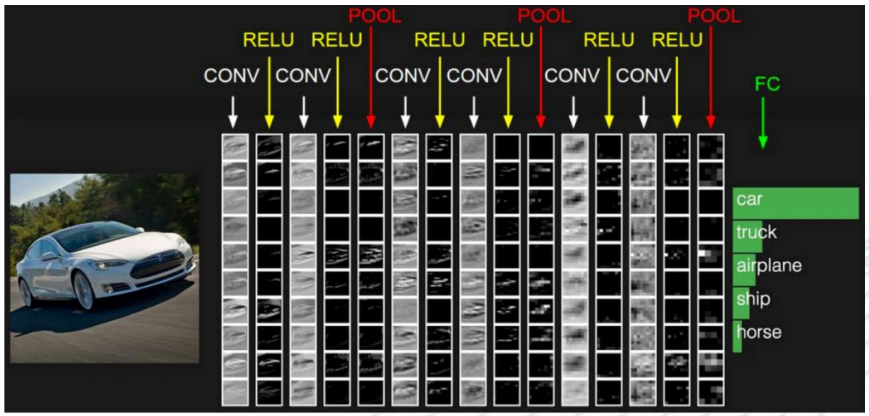


Flatten

This flattened vector is then connected to a few fully connected layers.

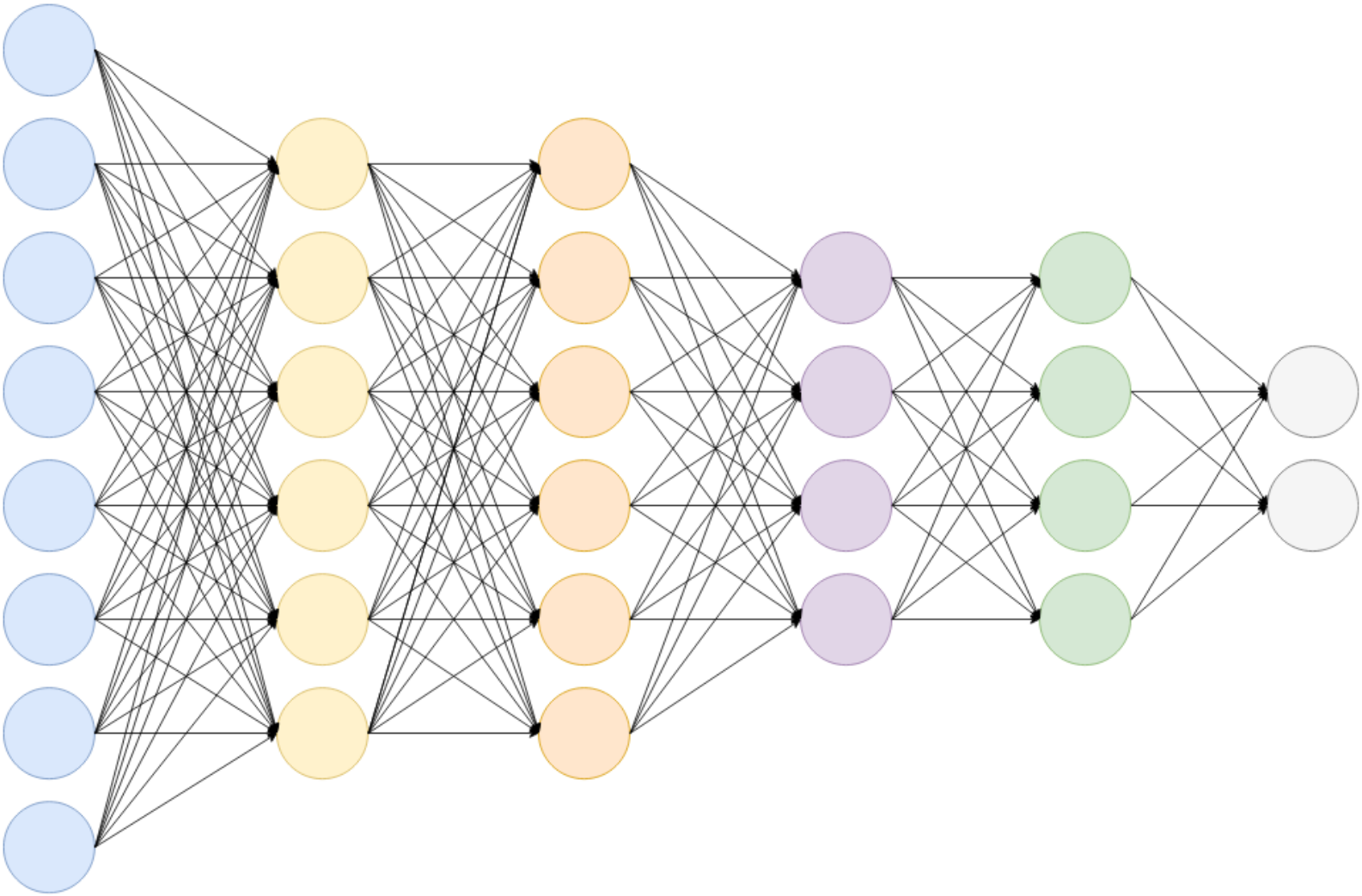
CNN: Classification

Now the extracted features are fed into a 2 or 3 hidden layers (similar to layers in a Deep Neural Network). This is called Fully Connected Layers in the CNN.



Source: Karkare 2019

Fully Connected Layer



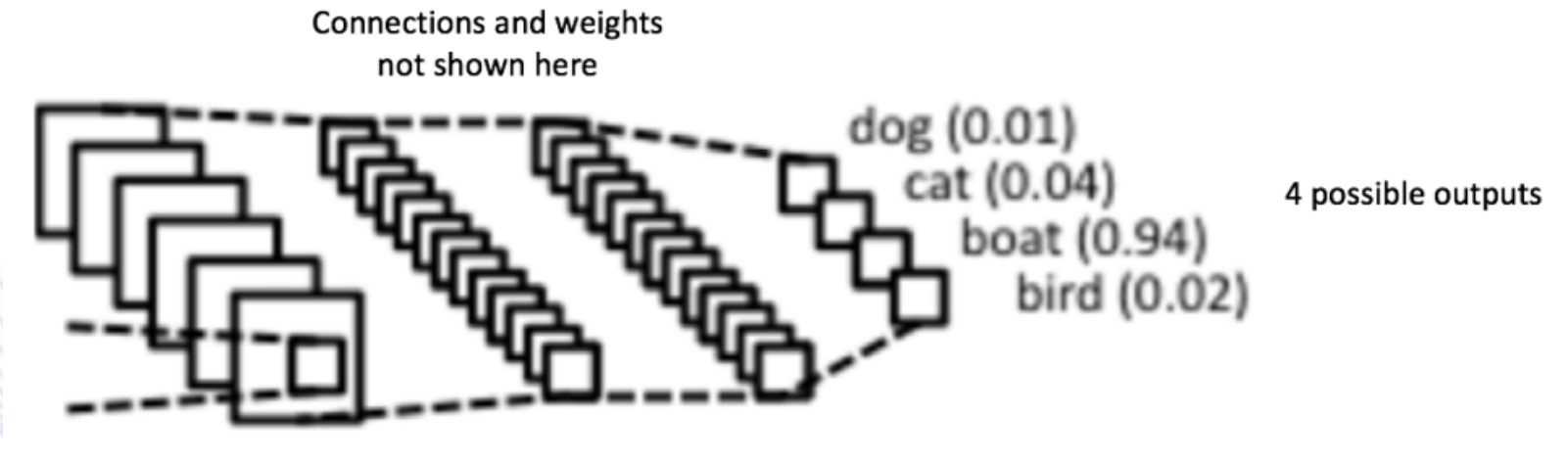
CNN - Fully Connected / Dense Layer

This object information is fed into one or more fully connected layers and connected to an output layer which, for example, has exactly the number of neurons corresponding to the number of different classes to be detected.

Fully Connected Layer

Fully Connected Layer consists simply of feed forward neural networks.

Fully Connected Layers form the last few layers in the network.



Neural Network Calculation for each Layer

For each layer of the Artificial Neural Network, the following calculation takes place:

$$F\left(W \cdot X + b\right)$$

- X - is the input vector with dimension $[p_\ell, 1]$
- W - is the weight matrix with dimensions $[p_\ell, n_\ell]$ where, p_ℓ is the number of neurons in the previous layer and n_ℓ is the number of neurons in the current layer.
- b - is the bias vector with dimension $[p_\ell, 1]$
- F - is the activation function, which is usually ReLU.

This calculation is repeated for each layer.

Softmax Activation Function

After passing through the fully connected layers, the final layer uses the softmax activation function (instead of ReLU) which is used to get probabilities of the input being in a particular class (classification).

$$\sigma(z_i) = \frac{e^{\beta z_i}}{\sum_{j=1}^{K} e^{-\beta z_j}}; \quad i = 1; \ldots; K; z \in \mathbb{R}^K$$

Softmax Activation Function

We have the probabilities of the object in the image belonging to the different classes.

CNN - Activation functions and optimization

In a convolutional neural network, the results of each layer are usually activated by a ReLU function. The ReLU function ensures that all values less than zero become zero and all values greater than zero remain [0; 1].

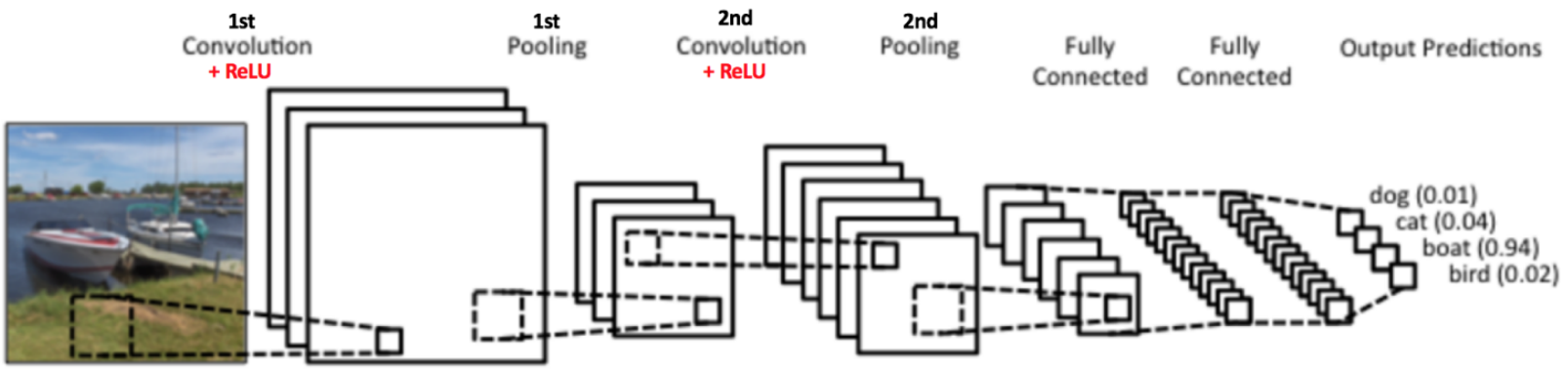
In the case of classification problems, the last layer receives a Softmax activation, i.e. the output of all output neurons is added to 1 and indicates the probability of the corresponding output.

CNN - Activation functions and optimization

The weights of the filters and the Fully Connected Layer are randomly selected at the beginning and then further optimized during the training by the known backpropagation.

In case of classification problems (e.g. which object can be seen on the picture) the Categorical Cross-Entropy (the negative natural logarithm of the calculated probability for the category) is used to measure the error.

CNN - Result


1st
Convolution
+ ReLU
1st
Pooling
2nd
Convolution
+ ReLU
2nd
Pooling
Fully
Connected
Fully
Connected
Output Predictions
dog (0.01)
cat (0.04)
boat (0.94)
bird (0.02)

Remarks

Convolutional neural network (CNN)

Convolutional neural networks show outstanding results in image and speech applications.

Yoon Kim in *Convolutional Neural Networks for Sentence Classification* describes the process and the results of text classification tasks using CNNs.
He presents a model built on top of word2vec[1], conducts a series of experiments with it, and tests it against several benchmarks, demonstrating that the model performs excellent.

[1] Family of functions in TensorFlow

Convolutional neural network (CNN)

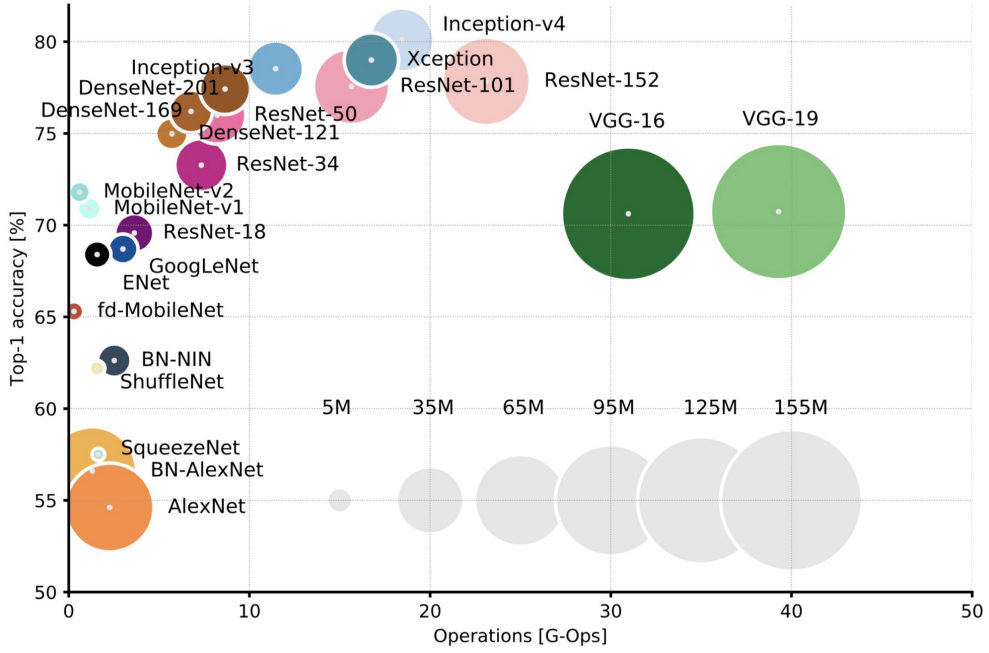
In Text *Understanding from Scratch*, Xiang Zhang and Yann LeCun, demonstrate that CNNs can achieve outstanding performance without the knowledge of words, phrases, sentences and any other syntactic or semantic structures with regards to a human language. Semantic parsing, paraphrase detection, speech recognition are also the applications of CNNs.

Remarks

In practice, a CNN learns the values of these filters on its own during the training process (although we still need to specify parameters such as number of filters, filter size, architecture of the network etc. before the training process).

The more number of filters we have, the more image features get extracted and the better our network becomes at recognizing patterns in unseen images.

Accuracy vs. Operations and Parameters



The size of the blobs is proportional to the number of network parameters; a legend is reported in the bottom right corner, spanning from $5 \cdot 10^6$ to $155 \cdot 10^6$ parameters.

Source: A. Canziani, A. Paszke, E. Culurciello - An Analysis of Deep Neural Network Models for Practical Applications, 2017

LeNet - 1994

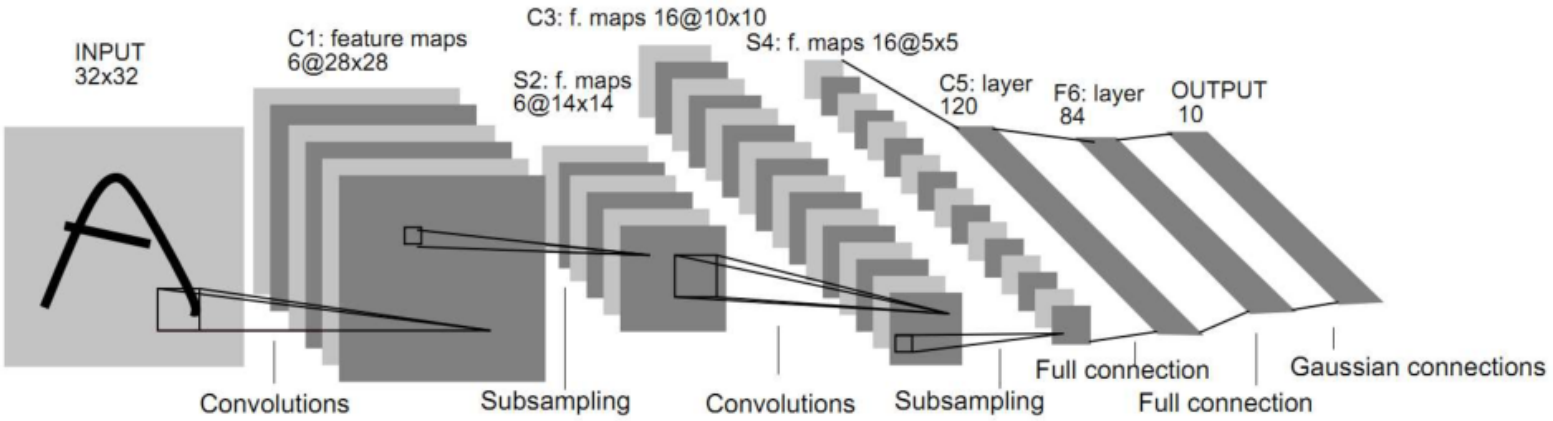


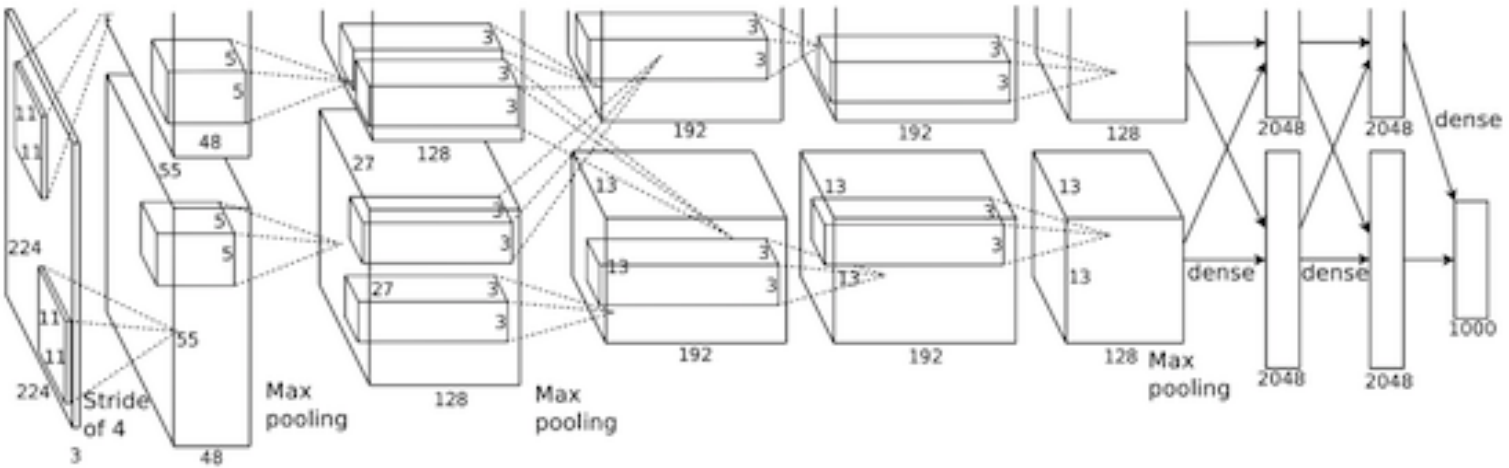
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeNet - Features

- A Convolutional neural network use sequence of 3 layers:
  1. convolution,
  2. pooling,
  3. non-linearity

  $\rightarrow$ This may be the key feature of Deep Learning for images since this paper!
- use convolution to extract spatial features
- subsample using spatial average of maps
- non-linearity in the form of tanh or sigmoids
- multi-layer neural network (MLP) as final classifier
- sparse connection matrix between layers to avoid large computational cost

AlexNet - 2012



Source: A. Alex Krizhevsky, I. Sutskever, G. E. Hinton - ImageNet Classification with Deep Convolutional Neural Networks, 2012

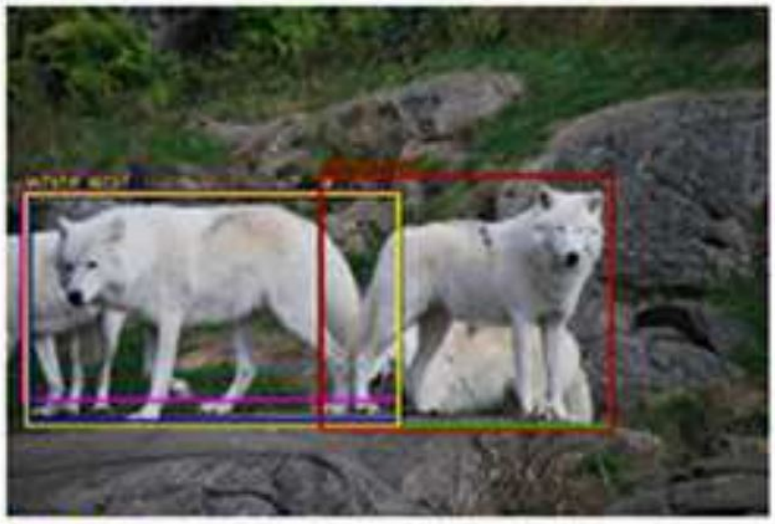
AlexNet - Features

AlexNet scaled the insights of LeNet into a much larger neural network that could be used to learn much more complex objects and object hierarchies.

The contributions of this work were:

- use of rectified linear units (ReLU) as non-linearities
- use of dropout technique to selectively ignore single neurons during training,
  $\rightarrow$ a way to avoid overfitting of the model
- overlapping max pooling,
  $\rightarrow$ avoiding the averaging effects of average pooling
- use of GPUs NVIDIA GTX 580 to reduce training time

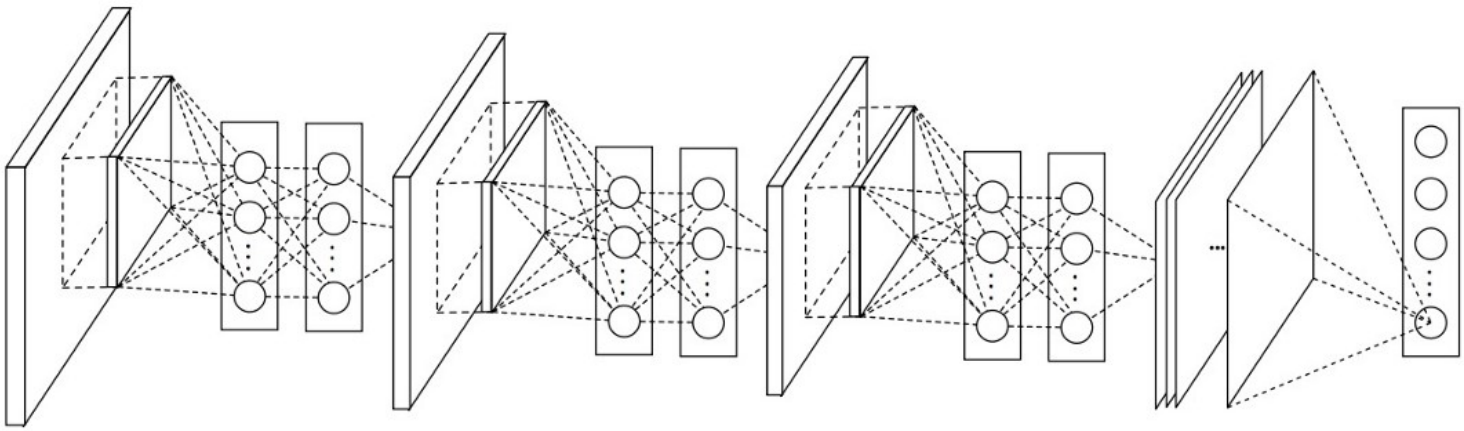
Overfeat - 2013



Source: P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, Y. LeCun - OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks

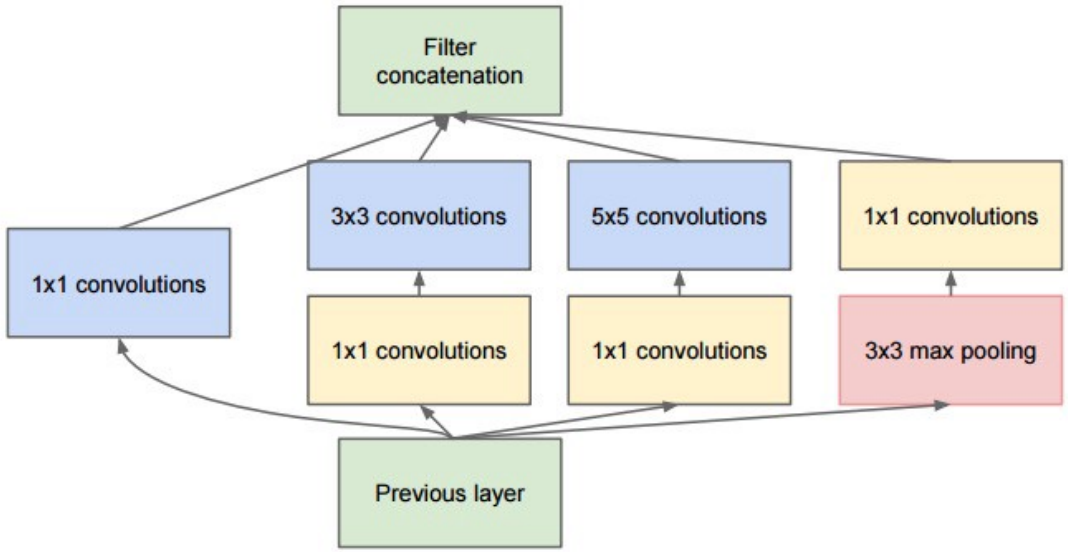
The article proposed learning bounding boxes.

Network in Network - 2014



Source: M. Lin, Q. Chen, S. Yan - Network In Network

GoogLeNet and Inception - 2014



Source: C. Szegedy et.al. - Going deeper with convolutions

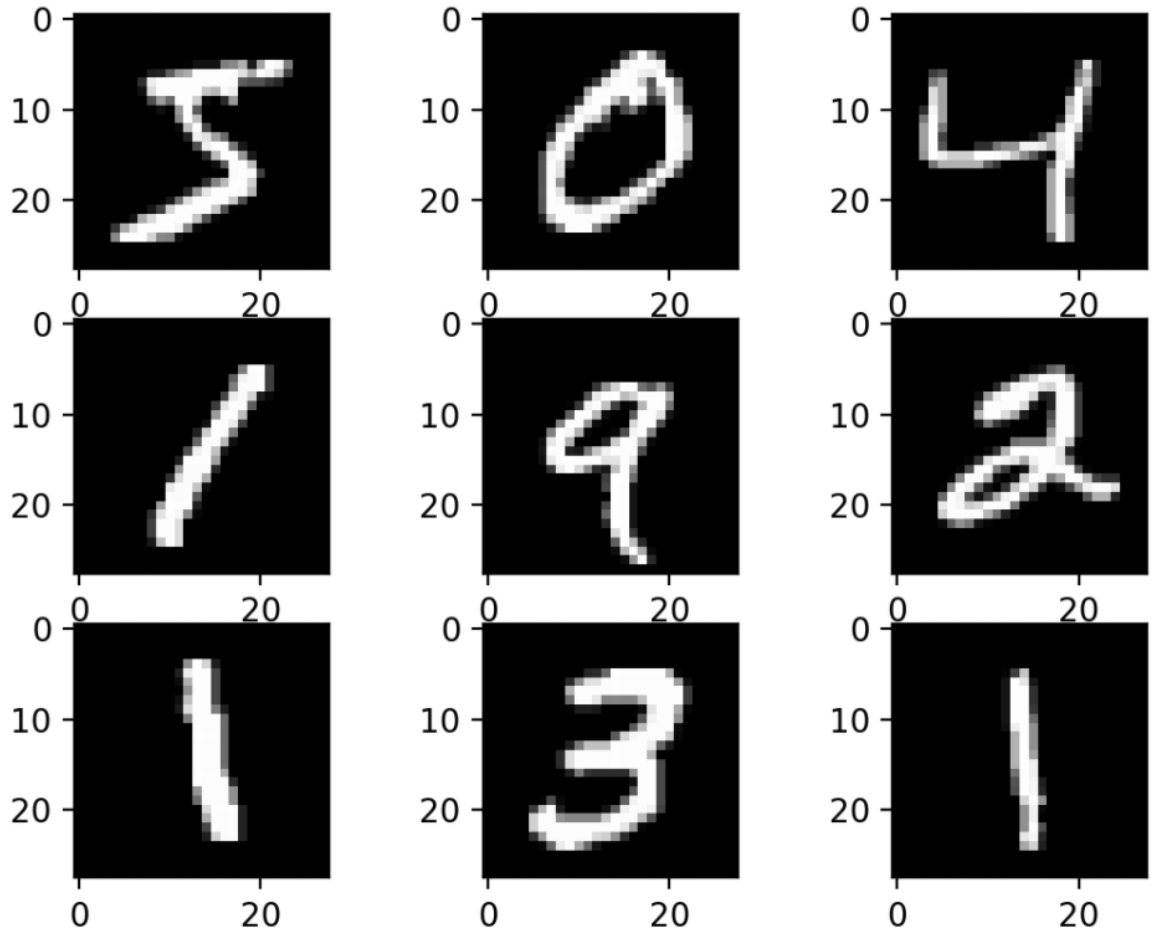
This is a very efficient network design.

Example MNIST

Dataset MNIST

- MNIST Handwritten Digit Classification Dataset
- The MNIST dataset is an acronym that stands for the Modified National Institute of Standards and Technology dataset.
- It is a dataset of
  - 60,000
  - square 28×28 pixel
  - grayscale images of handwritten single digits
  - between 0 and 9.
- The library Keras contains the dataset.

Dataset MNIST



Dataset MNIST

The task is to classify a given image of a handwritten digit into one of 10 classes representing integer values from 0 to 9, inclusively.

- It is a widely used and deeply understood dataset and, for the most part, is “solved”.
- Top-performing models are deep learning convolutional neural networks that achieve a classification accuracy of above 99%, with an error rate between 0.4% and 0.2% on the hold out test dataset.

Loading Dataset with Keras

```
# example of loading the mnist dataset
from keras.datasets import mnist
from matplotlib import pyplot
# load dataset
(trainX, trainy), (testX, testy) = mnist.load_data()
# summarize loaded dataset
print('Train:  X=%s, y=%s' % (trainX.shape, trainy.shape))
print('Test:  X=%s, y=%s' % (testX.shape, testy.shape))
# plot first few images
for i in range(9):
    # define subplot
    pyplot.subplot(330 + 1 + i)
    # plot raw pixel data
    pyplot.imshow(trainX[i], cmap=pyplot.get_cmap('gray'))
# show the figure
pyplot.show()
```

Running the Example

Running the example loads the MNIST train and test dataset and prints their shape.

Output:

```
Train:  X=(60000, 28, 28), y=(60000,)Test:  X=(10000, 28, 28), y=(10000,)
```

We can see that there are 60,000 examples in the training dataset and 10,000 in the test dataset and that images are indeed square with 28×28 pixels.

A plot of the first nine images in the dataset is also created showing the natural handwritten nature of the images to be classified.

Model Evaluation Methodology

The dataset already has a well-defined train and test dataset that we can use.

In order to estimate the performance of a model for a given training run, we can further split the training set into a train and validation dataset. Performance on the train and validation dataset over each run can then be plotted to provide learning curves and insight into how well a model is learning the problem.

Model Evaluation Methodology

The Keras API supports this by specifying the `validation_data` argument to the `model.fit()` function when training the model, that will, in turn, return an object that describes model performance for the chosen loss and metrics on each training epoch.

```
# record model performance on a validation dataset
during training
history = model.fit(..., validation_data=(valX,
valY))
```

Model Evaluation Methodology

In order to estimate the performance of a model on the problem in general, we can use k-fold cross-validation, perhaps five-fold cross-validation.
This will give some account of the models variance with both respect to differences in the training and test datasets, and in terms of the stochastic nature of the learning algorithm. The performance of a model can be taken as the mean performance across k-folds, given the standard deviation, that could be used to estimate a confidence interval if desired.

Model Evaluation Methodology

We can use the class `KFold` from the scikit-learn API to implement the k-fold cross-validation evaluation of a given neural network model. There are many ways to achieve this, although we can choose a flexible approach where the class `KFold` is only used to specify the row indexes used for each spit.

```
# example of k-fold cv for a neural net 0
data = ...
# prepare cross validation
kfold = KFold(5, shuffle=True, random_state=1)
# enumerate splits
for train_ix, test_ix in kfold.split(data):
    model = ...
...
```

Model Evaluation Methodology

We will hold back the actual test dataset and use it as an evaluation of our final model.

How to Develop a Baseline Model

This is critical as it both involves developing the infrastructure for the test harness so that any model we design can be evaluated on the dataset, and it establishes a baseline in model performance on the problem, by which all improvements can be compared.

How to Develop a Baseline Model

The design of the test harness is modular, and we can develop a separate function for each piece. This allows a given aspect of the test harness to be modified or inter-changed, if we desire, separately from the rest.

We can develop this test harness with five key elements. They are

- the loading of the dataset,
- the preparation of the dataset,
- the definition of the model,
- the evaluation of the model, and
- the presentation of results.

Load Dataset

We know some things about the dataset.

For example, we know that the images are all pre-aligned (e.g. each image only contains a hand-drawn digit), that the images all have the same square size of 28×28 pixels, and that the images are grayscale.

Therefore, we can load the images and reshape the data arrays to have a single color channel.

Load Dataset

```
# load dataset
(trainX, trainY), (testX, testY) = mnist.load_data()
# reshape dataset to have a single channel
trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
testX = testX.reshape((testX.shape[0], 28, 28, 1))
```

Load Dataset

We also know that there are 10 classes and that classes are represented as unique integers.

We can, therefore, use a one hot encoding for the class element of each sample, transforming the integer into a 10 element binary vector with a 1 for the index of the class value, and 0 values for all other classes. We can achieve this with the `to_categorical()` utility function.

```
# one hot encode target values
trainY = to_categorical(trainY)
testY = to_categorical(testY)
```

Load Dataset

The function `load_dataset()` implements these behaviors and can be used to load the dataset.

```
# load train and test dataset
def load_dataset():
    # load dataset
    (trainX, trainY), (testX, testY) = mnist.load_data()
    # reshape dataset to have a single channel
    trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
    testX = testX.reshape((testX.shape[0], 28, 28, 1))
    # one hot encode target values
    trainY = to_categorical(trainY)
    testY = to_categorical(testY)
    return trainX, trainY, testX, testY
```

Prepare Pixel Data

We know that the pixel values for each image in the dataset are unsigned integers in the range between black and white, or 0 and 255.

We do not know the best way to scale the pixel values for modeling, but we know that some scaling will be required.

A good starting point is to normalize the pixel values of grayscale images, e.g. rescale them to the range [0,1]. This involves first converting the data type from unsigned integers to floats, then dividing the pixel values by the maximum value.

Prepare Pixel Data

```
# convert from integers to floats
train_norm = train.astype('float32')
test_norm = test.astype('float32')
# normalize to range 0-1
train_norm = train_norm / 255.0
test_norm = test_norm / 255.0
```

Prepare Pixel Data

The function `prep_pixels()` implements these behaviors and is provided with the pixel values for both the train and test datasets that will need to be scaled.

```
# scale pixels
def prep_pixels(train, test):
    # convert from integers to floats
    train_norm = train.astype('float32')
    test_norm = test.astype('float32')
    # normalize to range 0-1
    train_norm = train_norm / 255.0
    test_norm = test_norm / 255.0
    # return normalized images
    return train_norm, test_norm
```

This function must be called to prepare the pixel values prior to any modeling.

Define Model

Next, we need to define a baseline convolutional neural network model for the problem.

The model has two main aspects: the feature extraction front end comprised of convolutional and pooling layers, and the classifier backend that will make a prediction.

Define Model

For the convolutional front-end, we can start with a single convolutional layer with a small filter size (3,3) and a modest number of filters (32) followed by a max pooling layer. The filter maps can then be flattened to provide features to the classifier.

Given that the problem is a multi-class classification task, we know that we will require an output layer with 10 nodes in order to predict the probability distribution of an image belonging to each of the 10 classes. This will also require the use of a softmax activation function. Between the feature extractor and the output layer, we can add a dense layer to interpret the features, in this case with 100 nodes.

Define Model

All layers will use the ReLU activation function and the He weight initialization scheme, both best practices.

Define Model

We will use a conservative configuration for the stochastic gradient descent optimizer with a learning rate of 0.01 and a momentum of 0.9. The categorical cross-entropy loss function will be optimized, suitable for multi-class classification, and we will monitor the classification accuracy metric, which is appropriate given we have the same number of examples in each of the 10 classes.

Define Model

The function `define_model()` will define and return this model.

```
# define cnn model
def define_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu',\
        kernel_initializer='he_uniform',\
        input_shape=(28, 28, 1)))
    model.add(MaxPooling2D((2, 2)))
    model.add(Flatten())
    model.add(Dense(100, activation='relu',\
        kernel_initializer='he_uniform'))
    model.add(Dense(10, activation='softmax'))
    # compile model
    opt = SGD(lr=0.01, momentum=0.9)
    model.compile(optimizer=opt,loss='categorical_crossentropy',\
        metrics=['accuracy'])
    return model
```

Evaluate Model

The model will be evaluated using five-fold cross-validation. The value of $k = 5$ was chosen to provide a baseline for both repeated evaluation and to not be so large as to require a long running time. Each test set will be 20% of the training dataset, or about 12,000 examples, close to the size of the actual test set for this problem.

Evaluate Model

The training dataset is shuffled prior to being split, and the sample shuffling is performed each time, so that any model we evaluate will have the same train and test datasets in each fold, providing an apples-to-apples comparison between models.

Evaluate Model

We will train the baseline model for a modest 10 training epochs with a default batch size of 32 examples. The test set for each fold will be used to evaluate the model both during each epoch of the training run, so that we can later create learning curves, and at the end of the run, so that we can estimate the performance of the model. As such, we will keep track of the resulting history from each run, as well as the classification accuracy of the fold.

Evaluate Model

The function `evaluate_model()` implements these behaviors, taking the training dataset as arguments and returning a list of accuracy scores and training histories that can be later summarized.

Evaluate Model

```
# evaluate a model using k-fold cross-validation
def evaluate_model(dataX, dataY, n_folds=5):
    scores, histories = list(), list()
    # prepare cross validation
    kfold = KFold(n_folds, shuffle=True, random_state=1)
    # enumerate splits
    for train_ix, test_ix in kfold.split(dataX):
        # define model
        model = define_model()
        # select rows for train and test
        trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix],\
            dataX[test_ix], dataY[test_ix]
        # fit model
        history = model.fit(trainX, trainY, epochs=10, batch_size=32,\
            validation_data=(testX, testY), verbose=0)
        # evaluate model
        _, acc = model.evaluate(testX, testY, verbose=0)
        print('> %.3f' % (acc * 100.0))
        # stores scores
        scores.append(acc)
        histories.append(history)
    return scores, histories
```

Present Results

There are two key aspects to present:

1. the diagnostics of the learning behavior of the model during training and
2. the estimation of the model performance.

These can be implemented using separate functions.

Present Results

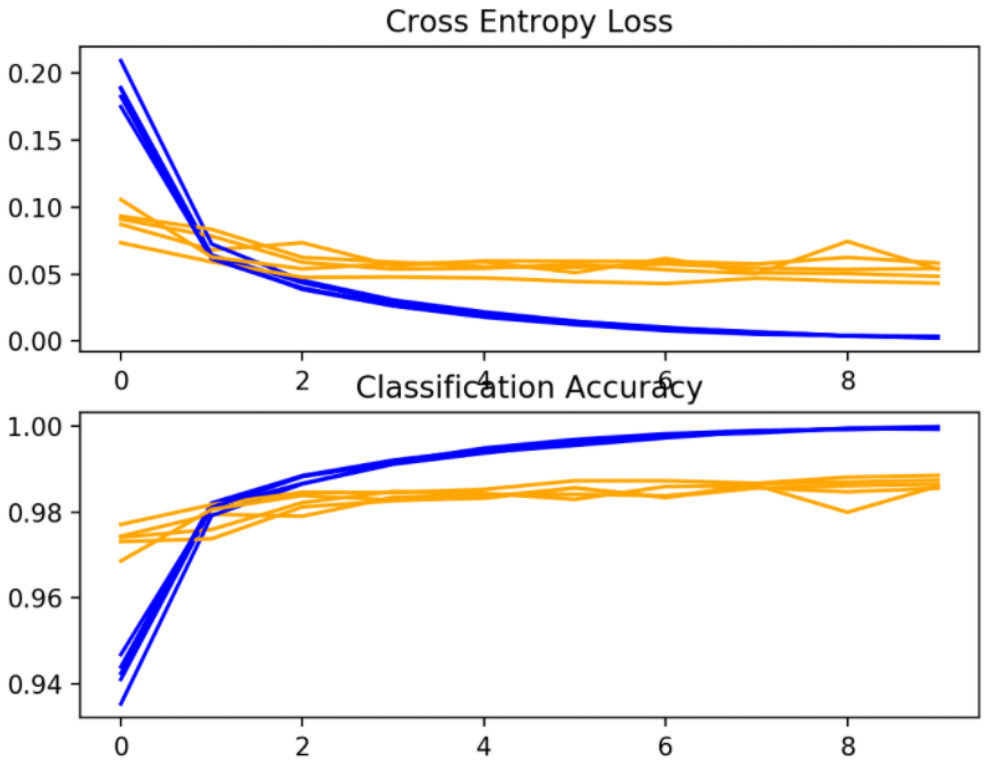
First, the diagnostics involve creating a line plot showing model performance on the train and test set during each fold of the k-fold cross-validation. These plots are valuable for getting an idea of whether a model is overfitting, underfitting, or has a good fit for the dataset.

We will create a single figure with two subplots, one for loss and one for accuracy. Blue lines will indicate model performance on the training dataset and orange lines will indicate performance on the hold out test dataset. The function `summarize_diagnostics()` creates and shows this plot given the collected training histories.

Present Results

```
# plot diagnostic learning curves
def summarize_diagnostics(histories):
    for i in range(len(histories)):
        # plot loss
        pyplot.subplot(2, 1, 1)
        pyplot.title('Cross Entropy Loss')
        pyplot.plot(histories[i].history['loss'], \
            color='blue', label='train')
        pyplot.plot(histories[i].history['val_loss'], \
            color='orange', label='test')
        # plot accuracy
        pyplot.subplot(2, 1, 2)
        pyplot.title('Classification Accuracy')
        pyplot.plot(histories[i].history['accuracy'], \
            color='blue', label='train')
        pyplot.plot(histories[i].history['val_accuracy'], \
            color='orange', label='test')
    pyplot.show()
```

Present Results



Loss and Accuracy Learning Curves for the Baseline Model During k-Fold Cross-Validation

Present Results

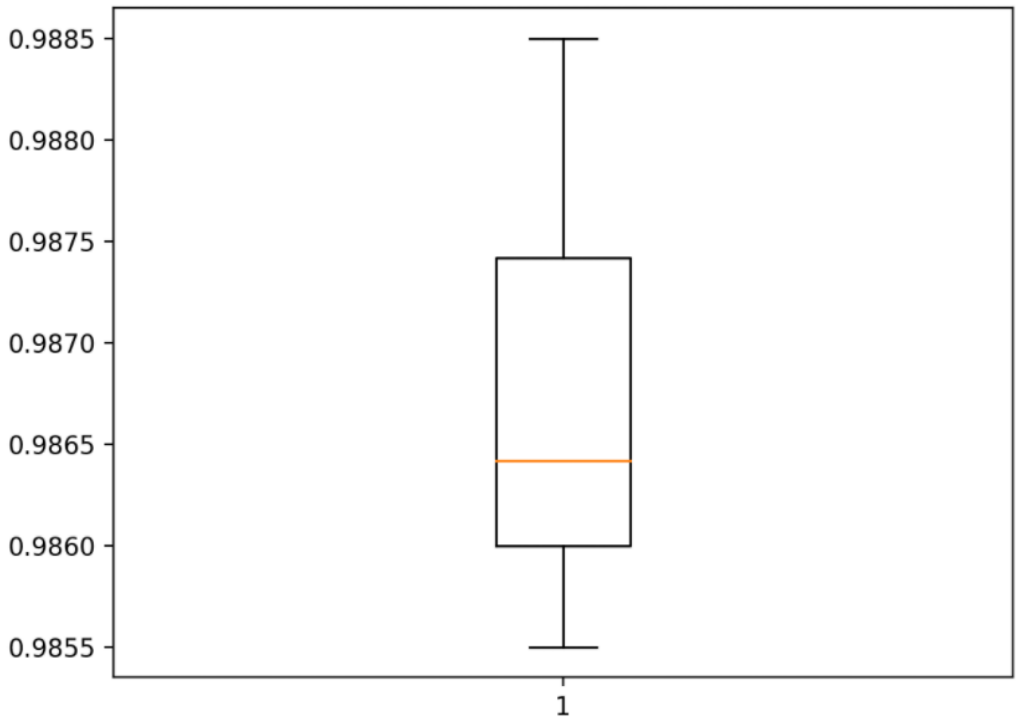
Next, the classification accuracy scores collected during each fold can be summarized by calculating the mean and standard deviation. This provides an estimate of the average expected performance of the model trained on this dataset, with an estimate of the average variance in the mean. We will also summarize the distribution of scores by creating and showing a box and whisker plot.

The function `summarize_performance()` implements this for a given list of scores collected during model evaluation.

Present Results

```
# summarize model performance
def summarize_performance(scores):
    # print summary
    print('Accuracy:  mean=%.3f std=%.3f, \
        n=%d' % (mean(scores)*100,\
        std(scores)*100, len(scores)))
    # box and whisker plots of results
    pyplot.boxplot(scores)
    pyplot.show()
```

Present Results



Summary of the model performance