MEDNARODNA
PODIPLOMSKA ŠOLA
JOŽEFA STEFANA

INFORMATION AND COMMUNICATION TECHNOLOGIES
PhD study programme

Data Mining and Knowledge Discovery

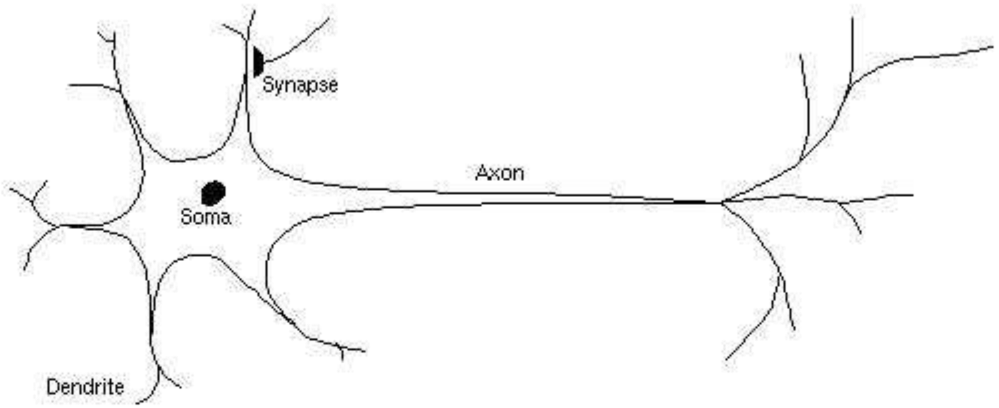
Petra Kralj Novak

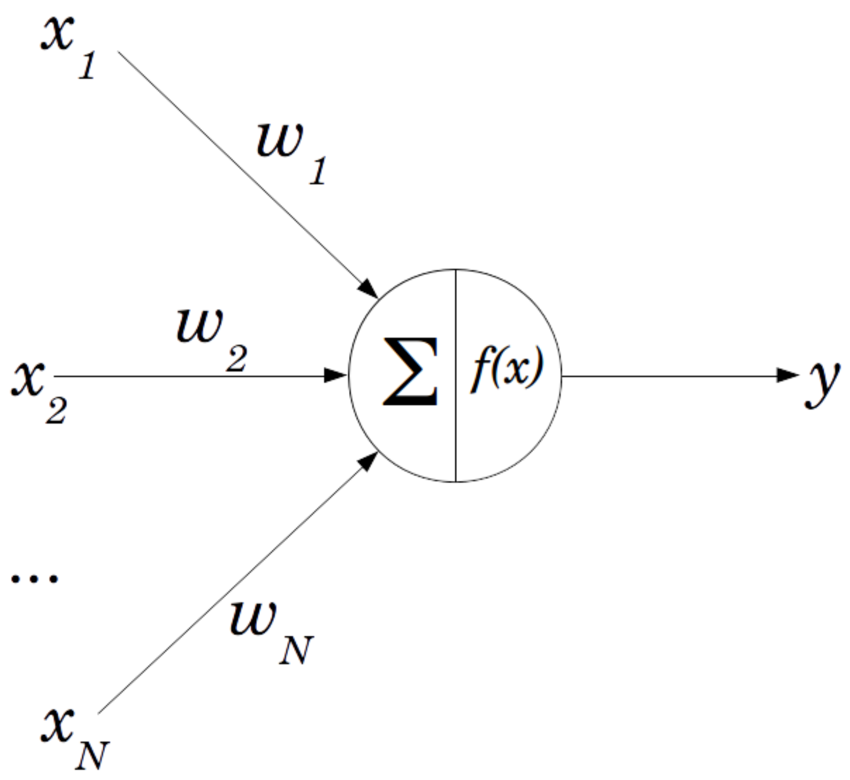
January 15, 2020

http://kt.ijs.si/petra_kralj/dmkd3.html

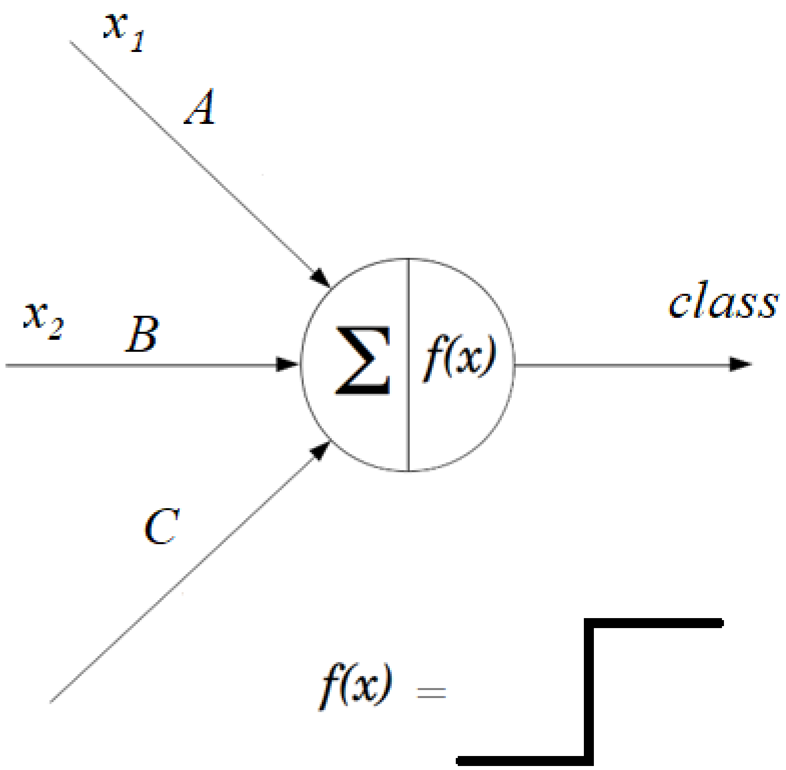
Neural networks

Neuron, perceptron





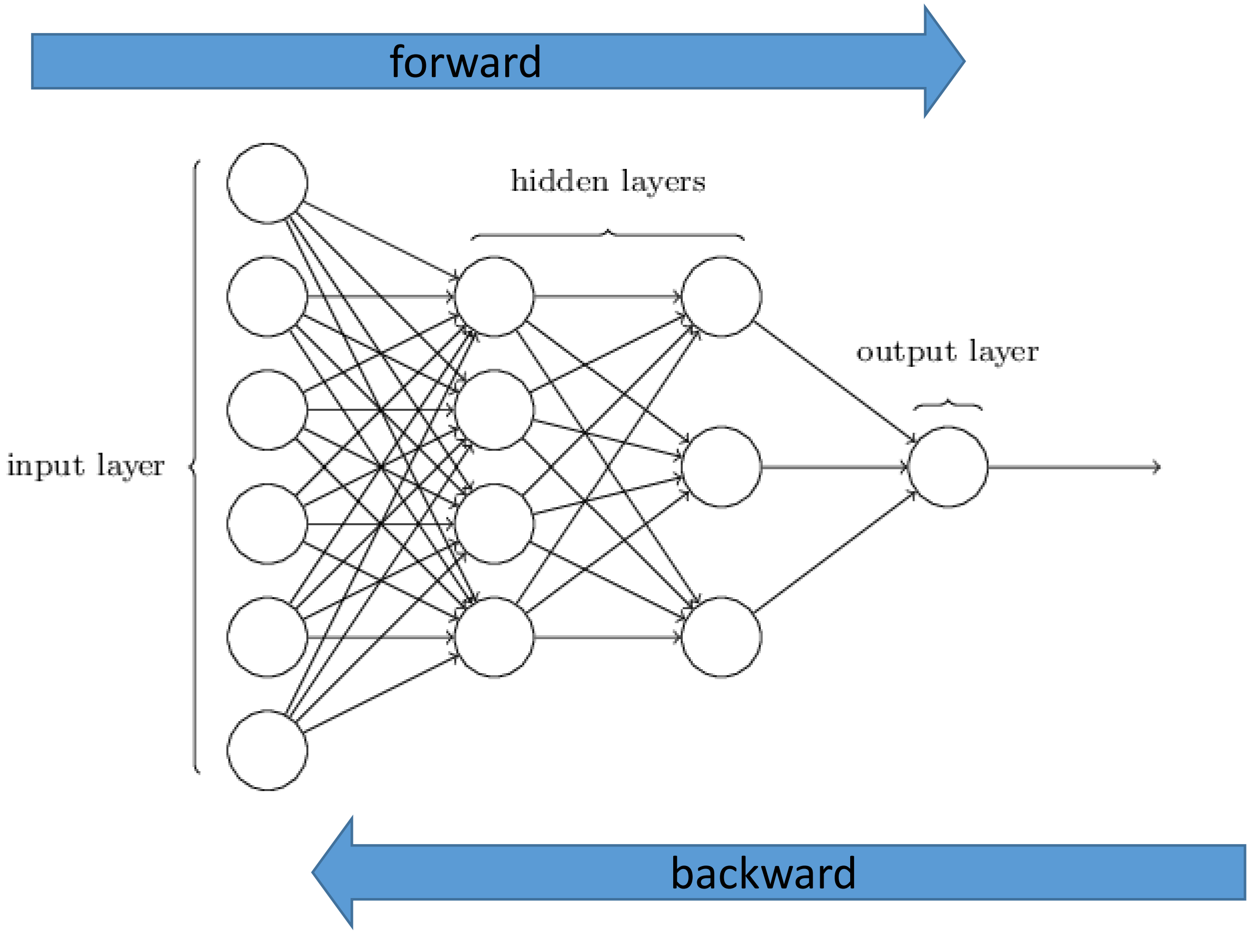
The **perceptron** is a mathematical model of a biological neuron



- A single perceptron can separate linearly.

$$\text{Output of P} = \begin{cases} 1 \text{ if } A\ x_1 + B\ x_2 > C \\ 0 \text{ if } A\ x_1 + B\ y_2 < = C \end{cases}$$

Neural network



Train

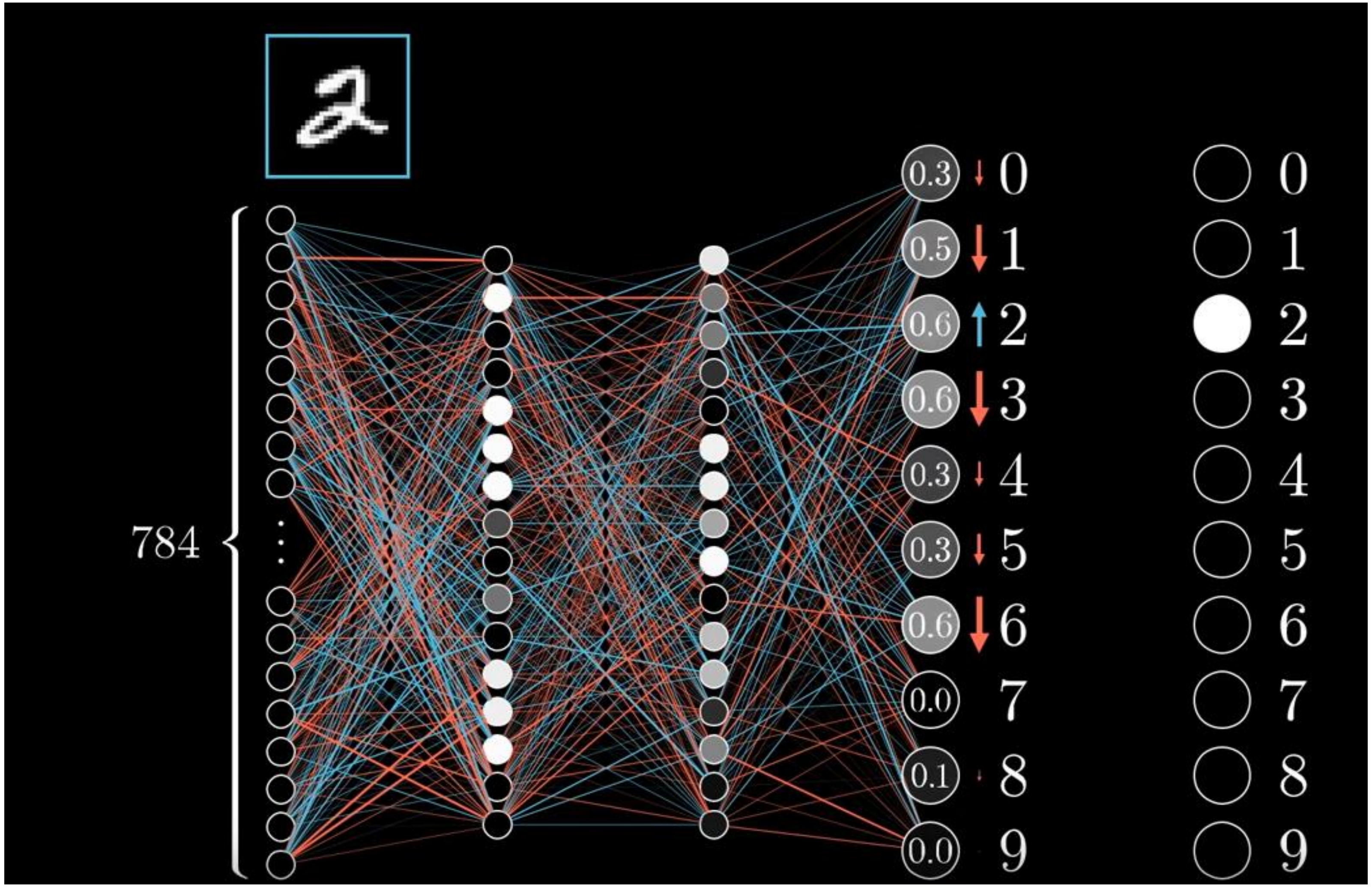
- **Forward propagation** (check performance)
  - **Loss function** is an error metric between actual and predicted
  - absolute error, sum of squared errors, …
- **Backpropagation** (direction of parameter/weight change)
  - How much the total error will change if we change the internal weight of the neural network with a certain small value **Δw (gradient)**
  - Backpropagate the errors using the derivatives of these functions: auto-differentiation
- **Optimization** (change weights based on learning rate, gradient descent)

```
New weight = old weight – Derivative Rate * learning rate
```

  - **Batch size** is a hyperparameter that controls the number of training samples to work through before the model's internal parameters are updated.
  - The number of **epochs** is a hyperparameter that controls the number of complete passes through the training dataset.

[What is backpropagation really doing?]

Deep learning, chapter 3



Hands on Neural Networks in Keras

Predictive model

- Architecture
  - Define
  - Compile
- Train (fit)
  - Forward
  - Backward
  - Optimize
- Evaluate & Predict
  - Forward

Keras: The Python Deep Learning library

- Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano.
- Google's Tensorflow: is a low-level framework that can be used with Python and C++.

Exercise

1. Install packages:
   - tensorflow
   - keras

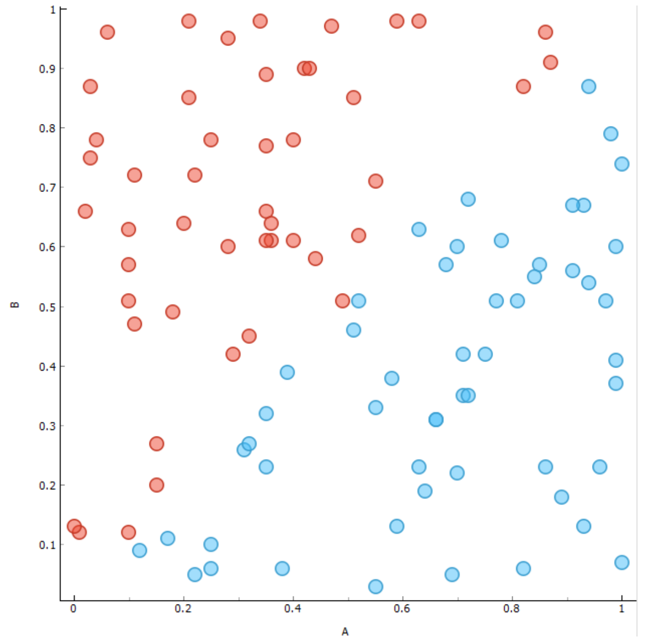
2. Train a simple one-layer network for the “A>B” problem.

- Start from gitlab:
  - http://source.ijs.si/pkraljnovak/DM_course
  - 8_neural_nets-perceptron.py

A	B	C	A>B
0.953725	0.544997	0.854959	True
0.490541	0.953735	0.200973	False
0.987391	0.524999	0.092299	True
0.074883	0.145092	0.158558	False
0.215517	0.003417	0.441095	True

…

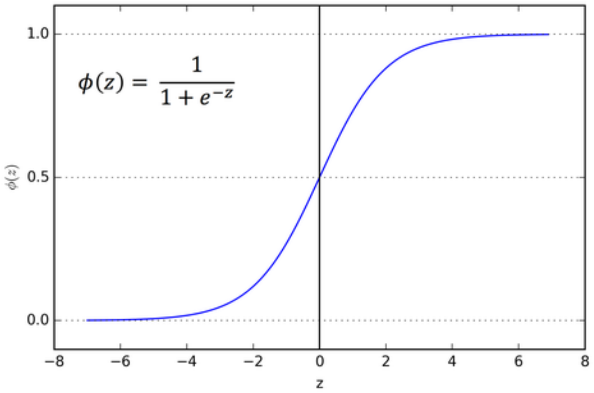
data shape: (1000, 4)

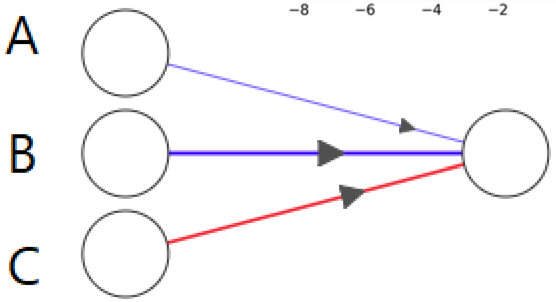


Network architecture

- One layer
  - 3 real-values inputs
  - 1 real-valued output
  - Activation function: sigmoid

- What do the weights need to be to get the desired output

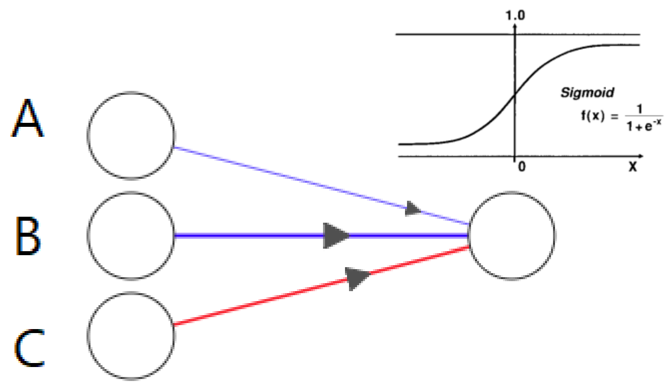




$$Y = \begin{cases} 1; \text{ if } A>B \\ 0; \text{ if } A<=B \end{cases}$$

Network architecture

8_neural_nets-perceptron.py




```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(input_dim=3, output_dim=1, init='uniform', activation='sigmoid'))
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

# Fit the model
model.fit(X_train, y_train, validation_data=(X_validation, y_validation), epochs=10, batch_size=64, verbose=0)

# Predict
y_pred = model.predict(X_test)
print(" Actual    Predicted    Difference")
for i in range(10):
    print("{0:6.2f}  {1:8.2f}     {2:8.2f}".format(y_test[i], y_pred[i][0], y_test[i]- y_pred[i][0]))

# Model perfomance
scores = model.evaluate(X_test, y_test, verbose=0)
print("Test set error: ", scores)
```

Questions

1. What is the error of the model (MAE, MSE)
2. Set the verbose parameter in model.fit to 2

```
model.fit(X_train, y_train, validation_data=(X_validation, y_validation),
      epochs=10, batch_size=64, verbose=2)
```

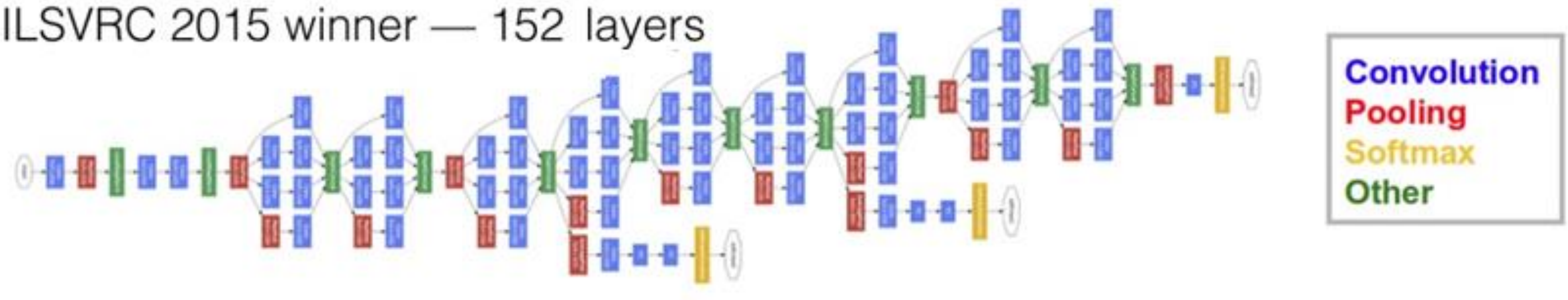
- What happens with the mean absolute error in each epoch on the validation set?
- What do we need to change for the model to perform better?

Deep learning

Neural networks and Deep learning

A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers.

Example:



ILSVRC 2015 Task 2a: Classification +localization with provided training data

Szegedy C. , et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

Exercise

Handwritten Digit Recognition using Convolutional Neural Networks in Python with Keras

Full tutorial: https://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/

MINST – handwritten digits

- Each image is a 28 by 28 pixel square (784 pixels total).
- Normalized in size and centered
- A standard spit of the dataset is used to evaluate and compare models, where 60,000 images are used to train a model and a separate set of 10,000 images are used to test it.

Exercise

- Load the MNIST dataset in Keras.
- Train and evaluate a **baseline neural network** model for the MNIST problem.
- Train and evaluate a simple **Convolutional Neural Network** for MNIST.
- Implement a **close to state-of-the-art deep learning** model for MNIST.

- The code is available in the github repository

Baseline neural network model for MNIST

Input layer with 784 neurons

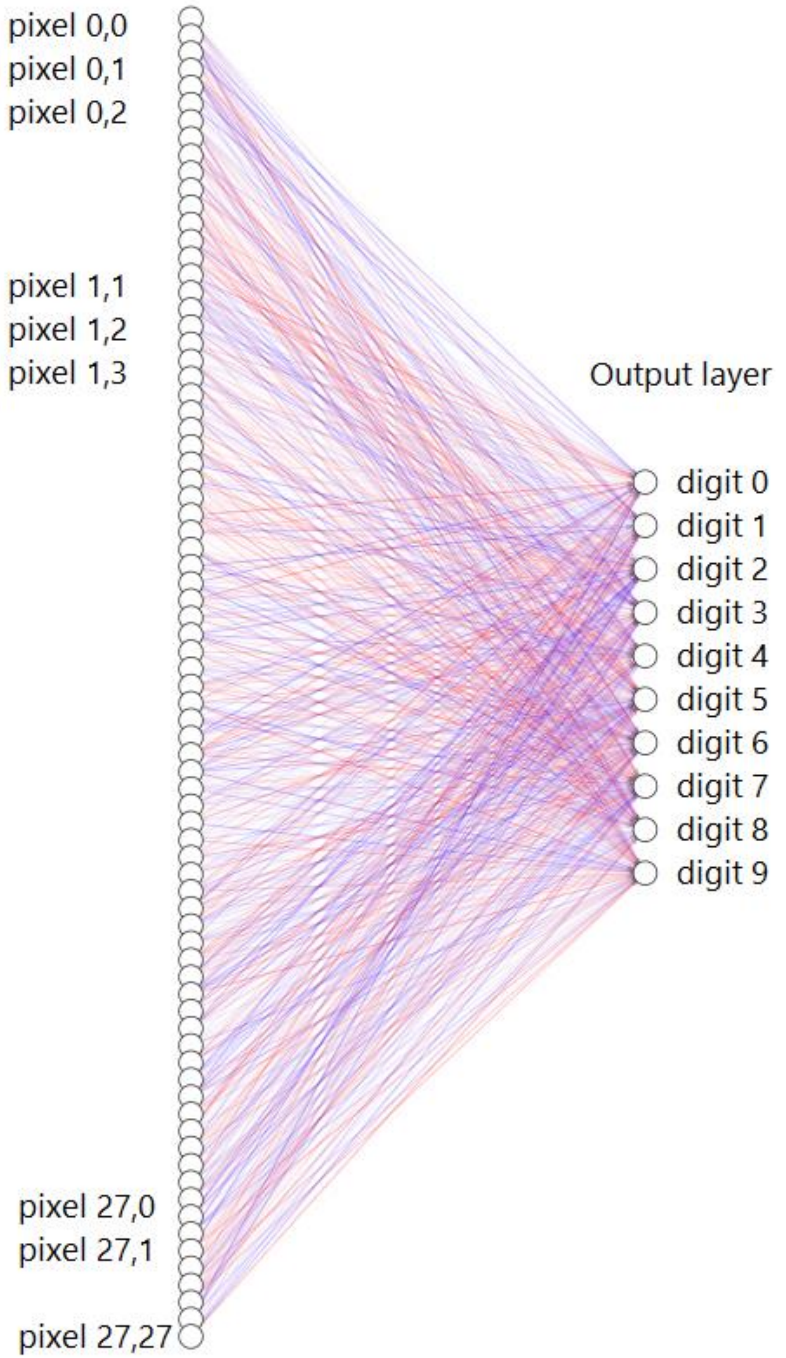
image size = 28x28

Output layer with 10 neurons

number of classes = 10

No hidden layers

How many weights are there between the input and the output layer?



Load the data

9_neural_nets-0-load_data.py

```
from keras.datasets import mnist
import matplotlib.pyplot as plt

# Plot ad hoc mnist instances

(X_train, y_train), (X_test, y_test) = mnist.load_data()    # Dataset of 60,000 28x28
grayscale images of the 10 digits, along with a test set of 10,000 images.
# plot 4 images as gray scale
plt.subplot(221)
plt.imshow(X_train[0], cmap=plt.get_cmap('gray'))
plt.subplot(222)
plt.imshow(X_train[1], cmap=plt.get_cmap('gray'))
plt.subplot(223)
plt.imshow(X_train[2], cmap=plt.get_cmap('gray'))
plt.subplot(224)
plt.imshow(X_train[3], cmap=plt.get_cmap('gray'))
# show the plot
plt.show()
```

Prepare data

9_neural_nets-1-perceptron.py

```
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)

# load data
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# flatten 28*28 images to a 784 vector for each image
num_pixels = X_train.shape[1] * X_train.shape[2]
X_train = X_train.reshape(X_train.shape[0], num_pixels).astype('float32')
X_test = X_test.reshape(X_test.shape[0], num_pixels).astype('float32')

# train-validation split
X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train, test_size=0.1, random_state=42)


# normalize inputs from 0-255 to 0-1
X_train = X_train / 255
X_validation = X_validation / 255
X_test = X_test / 255

# one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_validation = np_utils.to_categorical(y_validation)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
```

One-hot Encoding
for Multi-label and multi-target prediction

```
# one-hot encoding class labels

from keras.utils import np_utils

y_train[:10]
```

```
array([5, 0, 4, 1, 9, 2, 1, 3, 1, 4], dtype=uint8)
```

```
y_train_OneHotEncoding = np_utils.to_categorical(y_train)
y_train_OneHotEncoding[:10]
```

```
array([[ 0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.],
       [ 0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.]])
```

0 1 2 3 4 5 6 7 8 9

Define + compile, fit, predict

9_neural_nets-1-perceptron.py

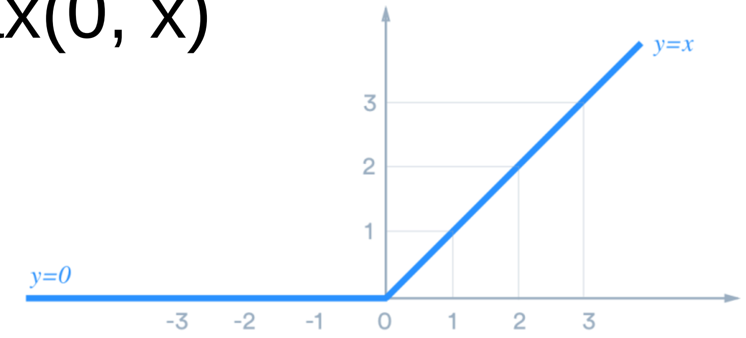
```
# define baseline model
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(num_pixels, input_dim=num_pixels, kernel_initializer='normal', activation='relu
    model.add(Dense(num_classes, kernel_initializer='normal', activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model


# build the model
model = baseline_model()
# Fit the model
model.fit(X_train, y_train, validation_data=(X_validation, y_validation), epochs=10, batch_size=200

# Final evaluation of the model
print("Final evaluation of the model")
scores = model.evaluate(X_test, y_test, verbose=1)
print("Baseline Error: %.2f%%" % (100 - scores[1] * 100))
```

Activation functions

- relu(x) = max(0, x)



- Softmax
  - After applying softmax, each component will be in the interval [0,1], and the components will add up to 1
  - The softmax function is frequently used as the final activation function in neural networks for classification problems.
  - Maps the non-normalized output of a network to a probability distribution over predicted output classes.

Loss function: categorical_crossentropy

- Multi-class classification tasks
- Must be combined with Softmax

$$L(y, \hat{y}) = -\sum_{j=0}^{M} \sum_{i=0}^{N} (y_{ij} * log(\hat{y}_{ij}))$$

- $\hat{y}_{ij}$ is the predicted value
- y_{ij} is the actual (correct) value

Architecture

- Layers: type, inicialization, regularization
  - Dense
  - Convolutional
  - Pooling
  - Dropout – for regularization
  - Recurrent
  - Embedding
- Activation functions
  - relu
  - softmax (output layer)
- Loss function
  - Classification
    - **categorical_crossentropy**, categorical_hinge, sparse_categorical_crossentropy, binary_crossentropy, …
  - Numeric prediction
    - **mean_squared_error**, mean_absolute_error, mean_absolute_percentage_error, mean_squared_logarithmic_error, cosine_proximity, …

- **Model.compile**

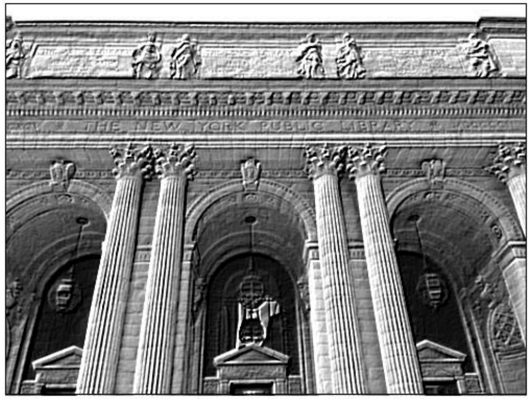
Convolution on images

Convolving the original image with an appropriate filter kernel produces the filtered image.

Linear filtering can improve images in many ways: sharpening the edges of objects, reducing random noise, correcting for unequal illumination, deconvolution to correct for blur and motion, etc.

Examples from: http://setosa.io/ev/image-kernels/

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



Emboss

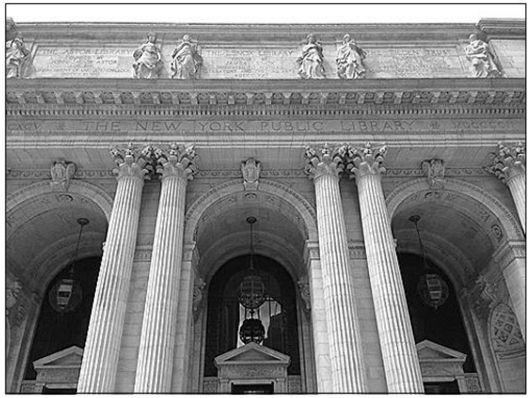
-1	0	1
-2	0	2
-1	0	1

Right Sobel

-1	-1	-1
-1	8	-1
-1	-1	-1

Outline

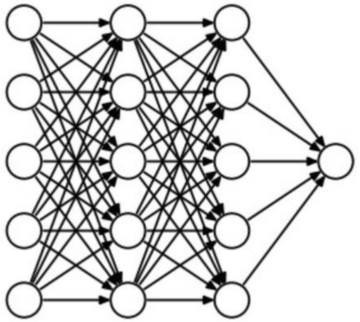
0	-1	0
-1	5	-1
0	-1	0



Sharpen

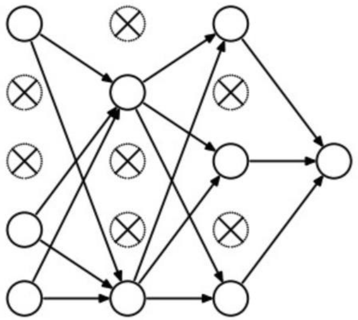
Types of layers (1)

Dense



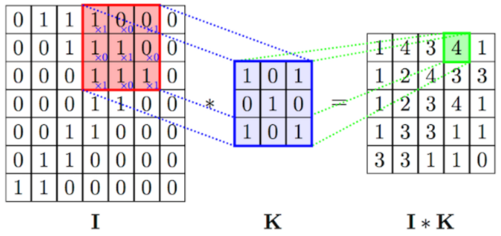
Fully connected.

Dropout



During training, some neurons on a particular layer will be deactivated. This improves generalization because it forces the layer to learn with different neurons the same "concept“.

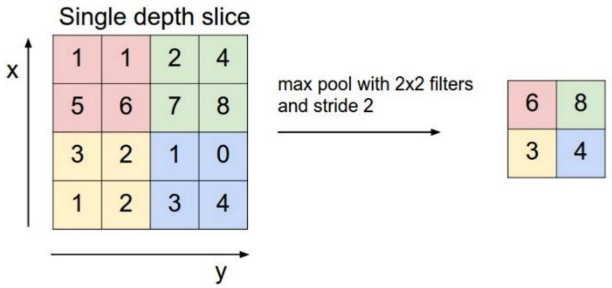
Convolutional



The convolution layer comprises of a set of independent filters. Each filter is independently convolved with the image.

Example: link

Pooling

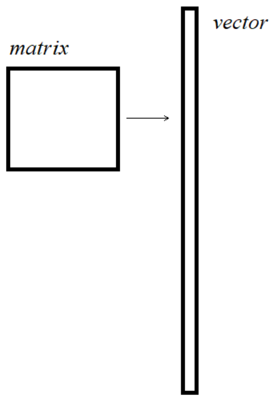


A max-pooling layer takes the maximum of features over small blocks of a previous layer.

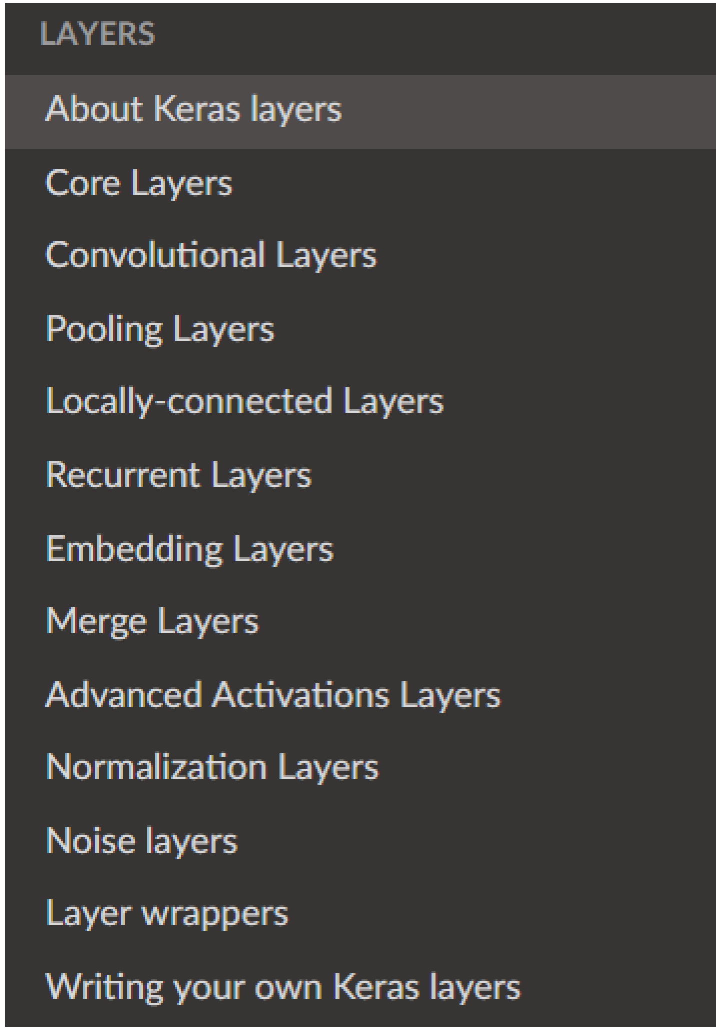
Edge detection example
https://youtu.be/puxHUGpuOVw

Types of layers (2)

Flatten



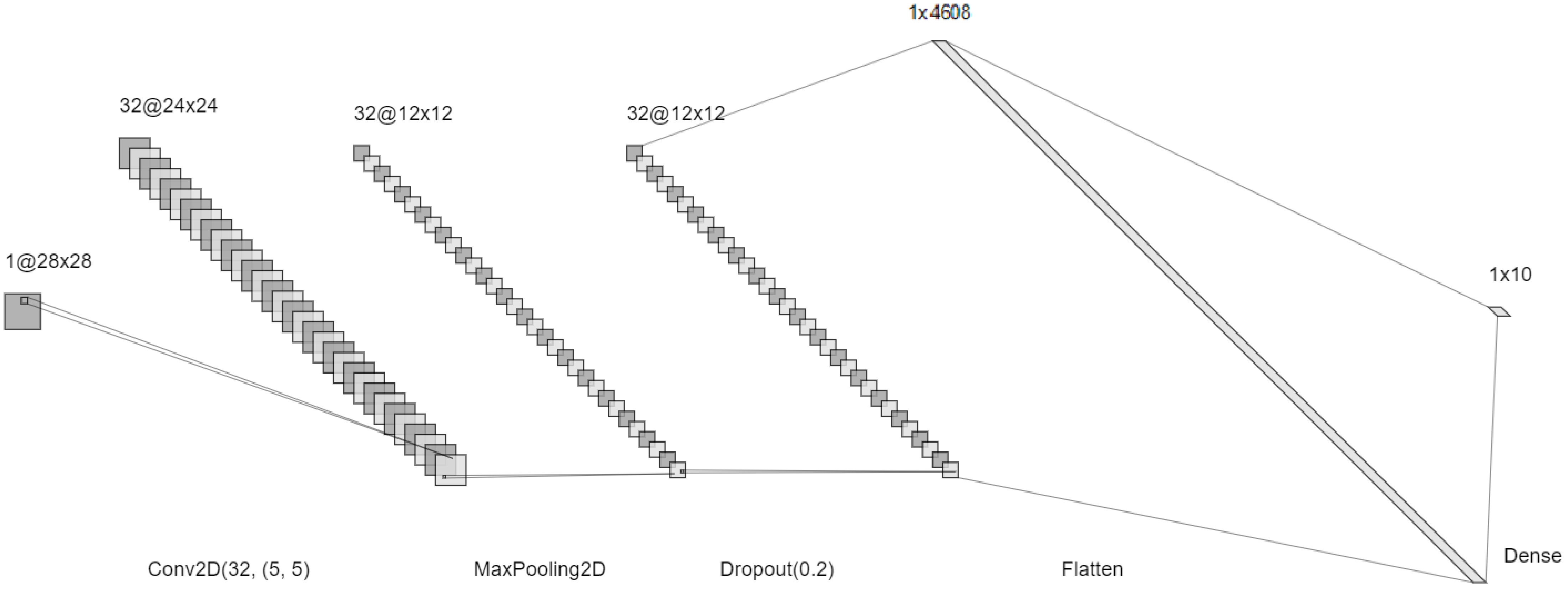
Fully connected.



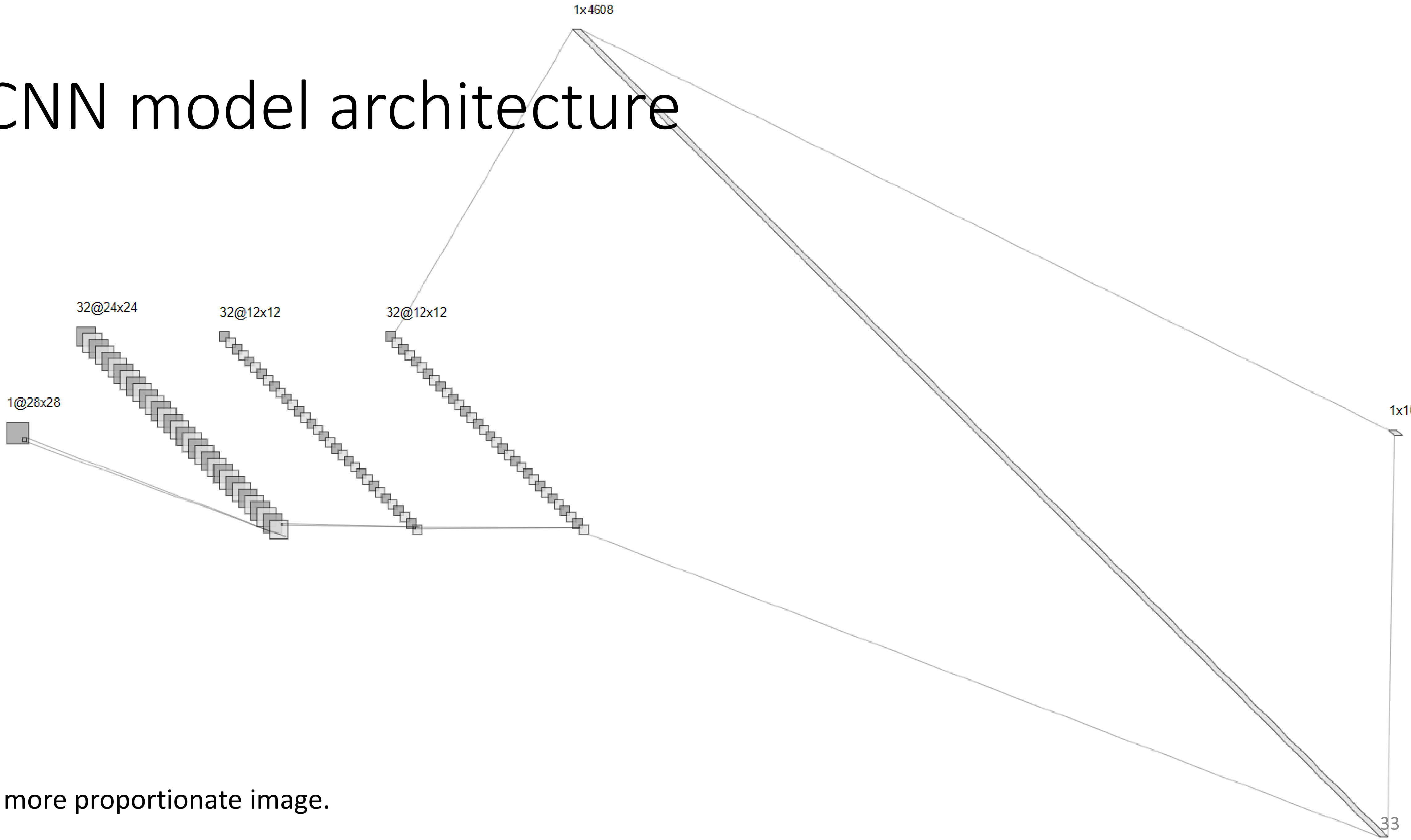
Convolutional model architecture

1. Convolutional layer: Convolution2D. The layer has 32 feature maps, which with the size of 5×5 and a rectifier activation function.
2. Pooling layer that takes the max called MaxPooling2D. It is configured with a pool size of 2×2.
3. Dropout: regularization layer
4. Flatten: converts the 2D matrix data to a vector
5. Dense layer with 128 neurons
6. Output layer has 10 neurons (for the 10 classes)

CNN model architecture



CNN model architecture



A more proportionate image.

Convolutional model 1

9_neural_nets-1-perceptron.py

```
def baseline_model():
    # create model
    model = Sequential()
    model.add(Conv2D(32, (5, 5), input_shape=(1, 28, 28), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

Convolutional model 2

9_neural_nets-2-convolutional.py

```
def larger_model():
    # create model
    model = Sequential()
    model.add(Conv2D(30, (5, 5), input_shape=(1, 28, 28), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(15, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(50, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```