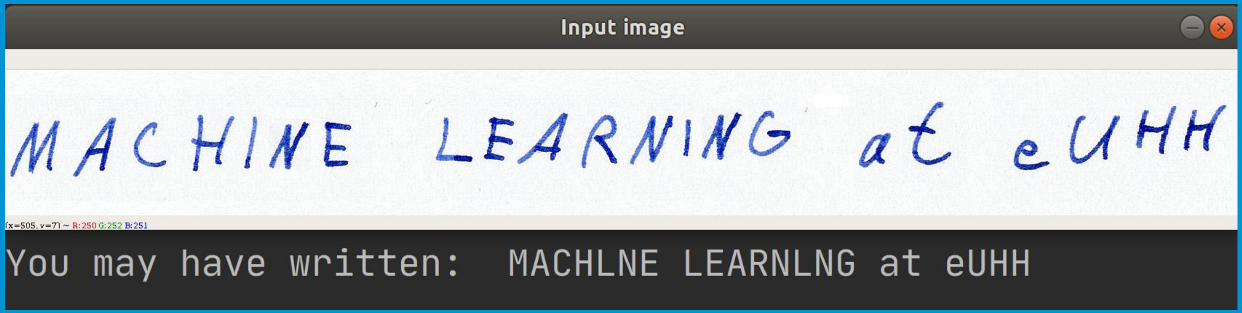
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

FAKULTÄT
FÜR MATHEMATIK, INFORMATIK
UND NATURWISSENSCHAFTEN



Project 3
Ismail Cucevic, Jannik Eggers, Alban Kryeziu, Julius Ziemann

Handwriting Recognition

Outline

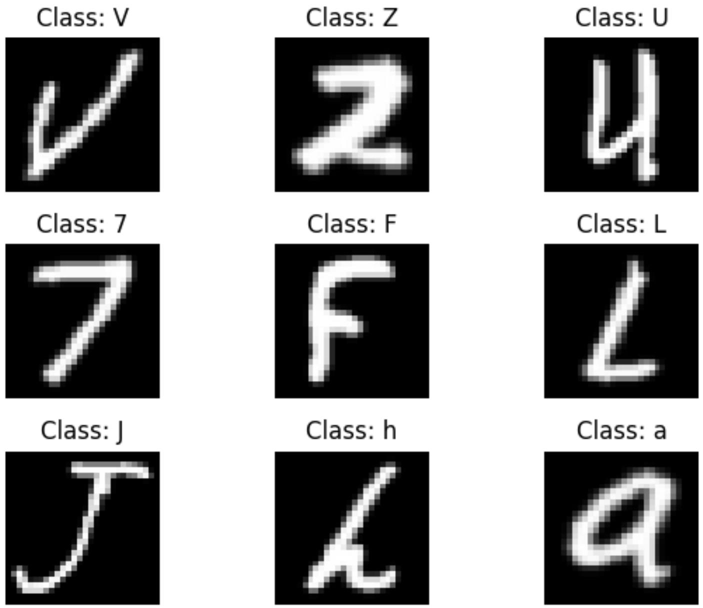
1. Goals
2. Datasets
3. Model selection
4. Neural Network: Training and performance
5. Image processing
6. Final product
7. Lessons learned

1. Goals

1. **Create Program** with following features:
   - Recognize single digit from a 28x28 image
   - Create and save own data
   - Recognize single letters from an image
   - Recognize multiple letters from image
2. **Learn** more about **implementing NN** for handwriting recognition
3. **Learn** more about the **theory of NN** needed for handwriting recognition
4. **Learn** how **to work on** machine learning **projects**

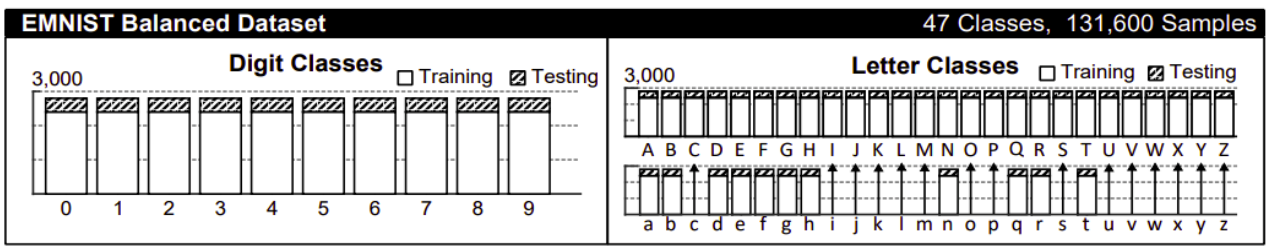
2. Datasets

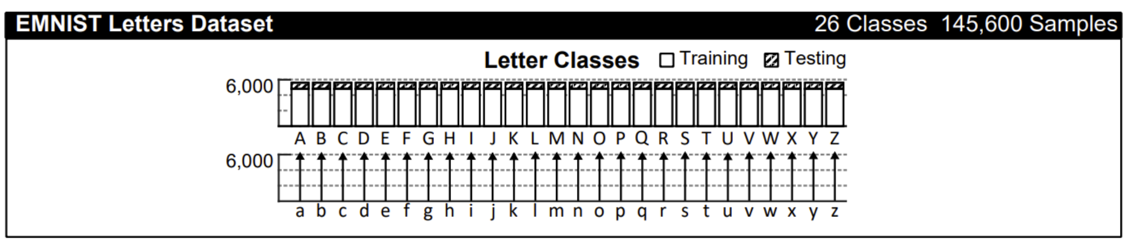
- EMNIST [4] database
- 28 x 28 greyscale
- Values 0 to 255
- Flattened image to 784 dimensional vector



Datasets and Classification

dataset	contains	classes
MNIST	digits	10
letters	letters	26
balanced	digits and letters	47



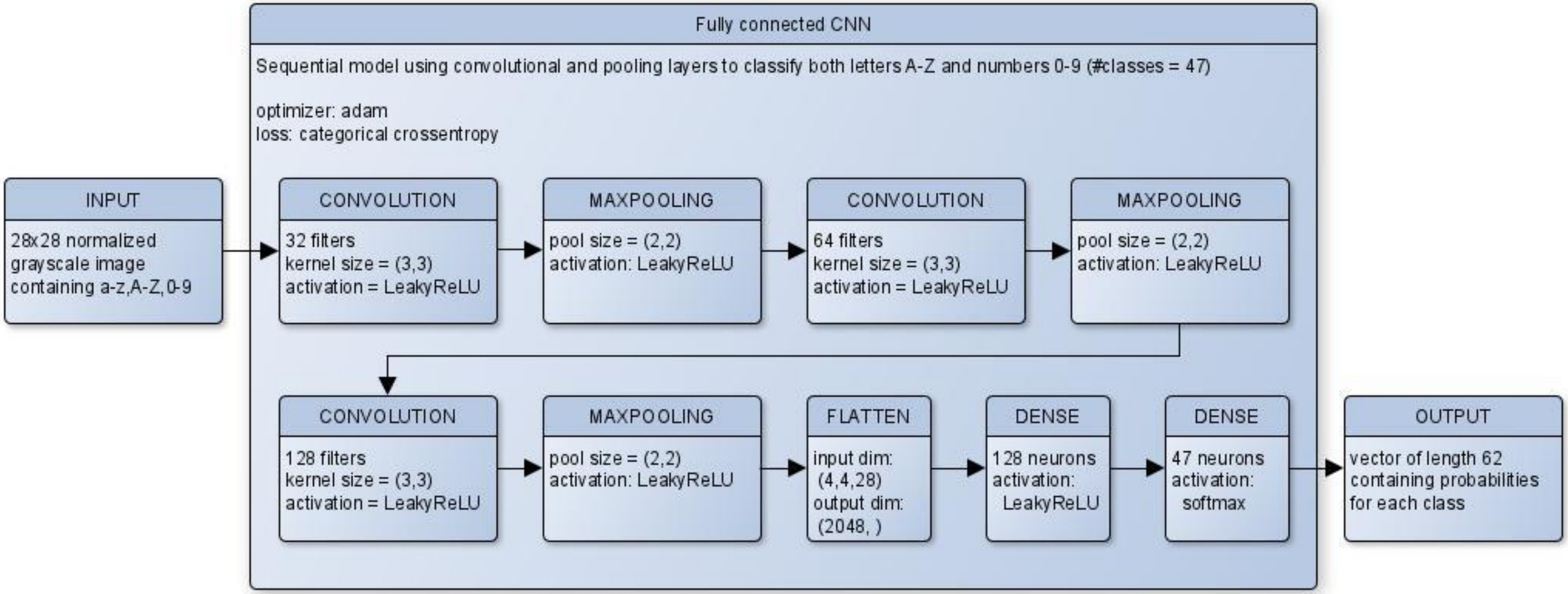


3. Model selection

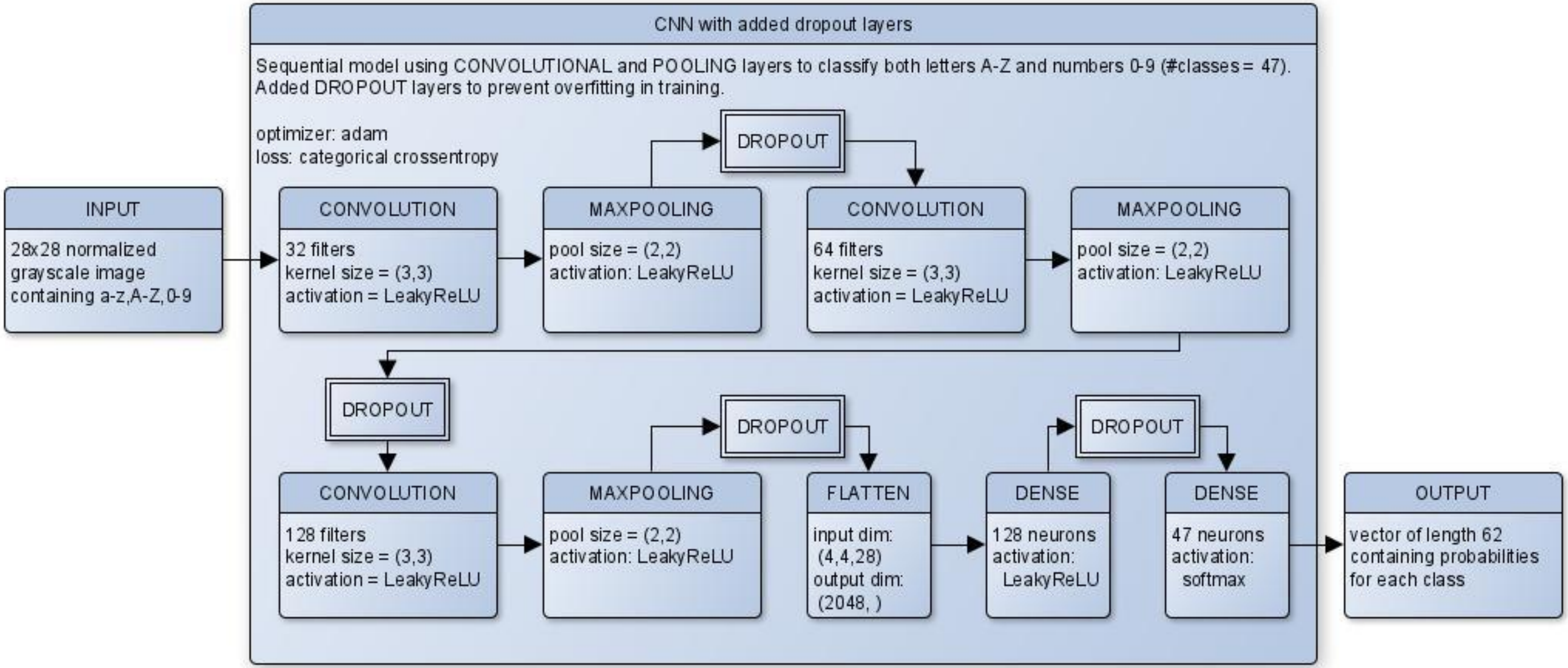
3.1 Why CNN?

- CNN are effective for parameter reduction without losing quality of the models.
- Good to identify edges of any object in image.
- All layers of a CNN have multiple convolutional filters working and scanning the complete feature matrix and carry out the dimensionality reduction.
- Alternative: SVMs, but they work better on fewer classes

3.2 Neural Network: Architecture



3.2 Neural Network: Architecture



4. Neural Network: Training and performance

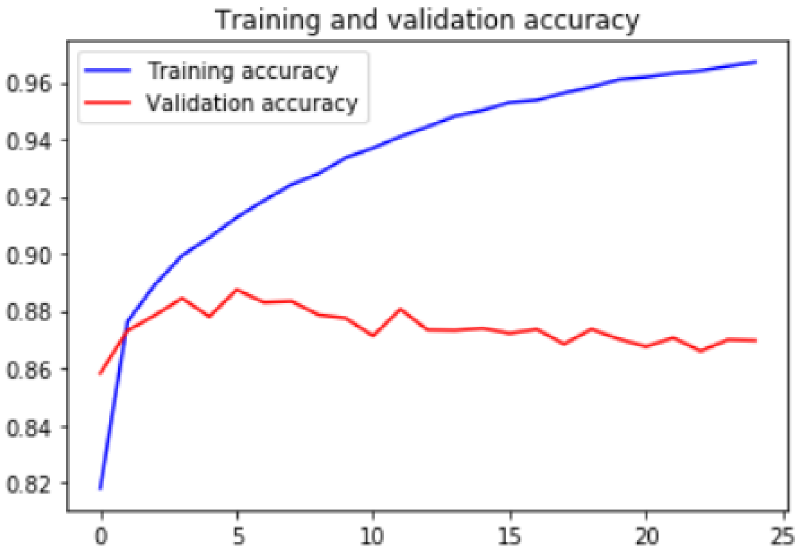
Question: How many epochs give best results?

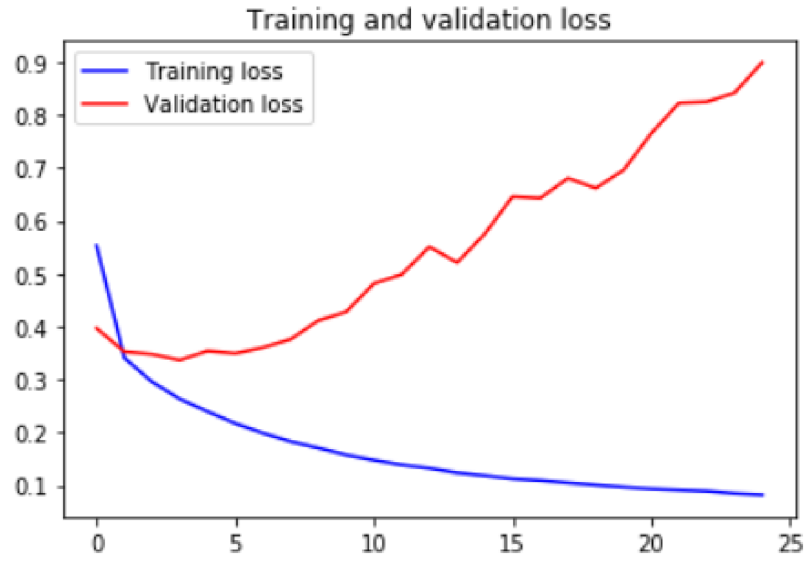
CNN with dropout:

```
Total params: 358,298
Trainable params: 358,298
Non-trainable params: 0
```

- Training on all 3 datasets
- Average training time per epoch: 4 minutes
- All training done with batch size 32

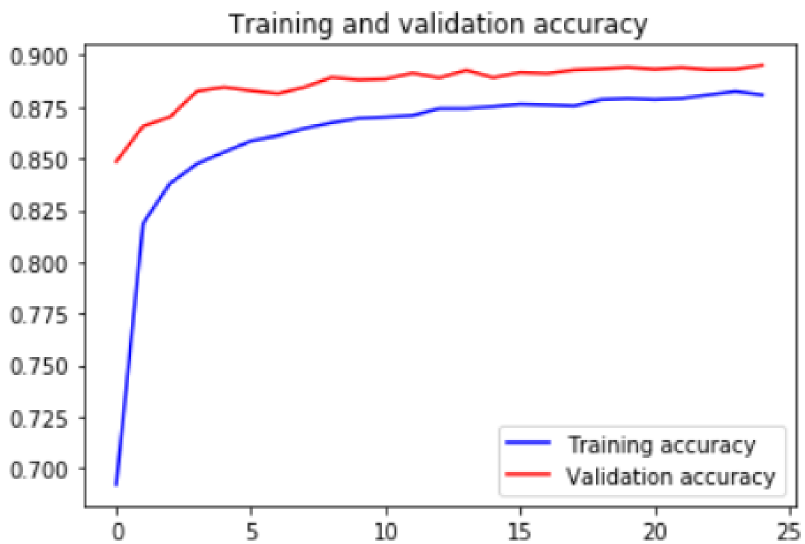
Fully connected CNN on balanced dataset

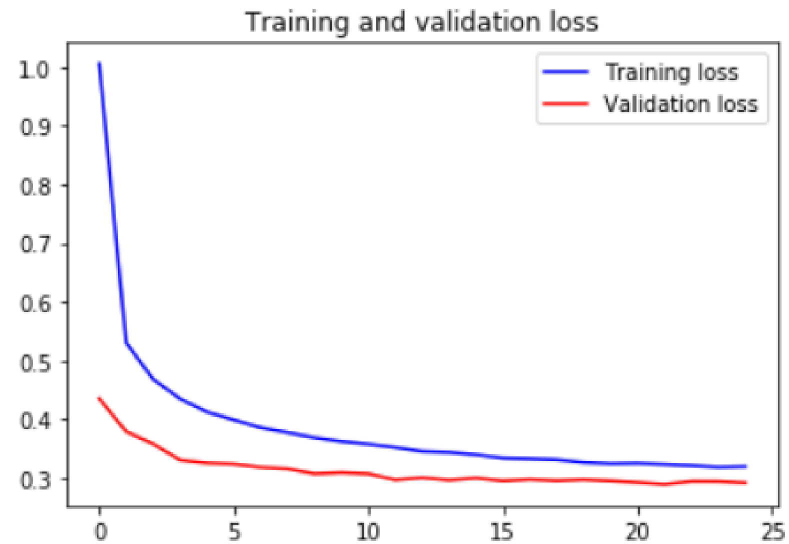




- Overfitting!

Adding dropout layers





- Problem solved?

Both architectures compared

	No dropout	With dropout
Optimal #epochs:	~4	~12
Test evaluation:	Test loss: 0.3325982670898133 Test accuracy: 0.88569146	Test loss: 0.317814768641553 Test accuracy: 0.8891489505767822

Dropout: yes or no?

- Much longer training required
- No big difference in accuracy on testing data
- Slightly improved loss

Other datasets…

Dataset	No dropout	With dropout
Letters	`Test loss: 0.17473136057826474` `Test accuracy: 0.9441827`	`Test loss: 0.1698504756536197` `Test accuracy: 0.943365395069122 3`
MNIST	`Test loss: 0.02518391303855460 3` `Test accuracy: 0.9922000169754028`	`Test loss: 0.03363668061224743 5` `Test accuracy: 0.9900000095367432`

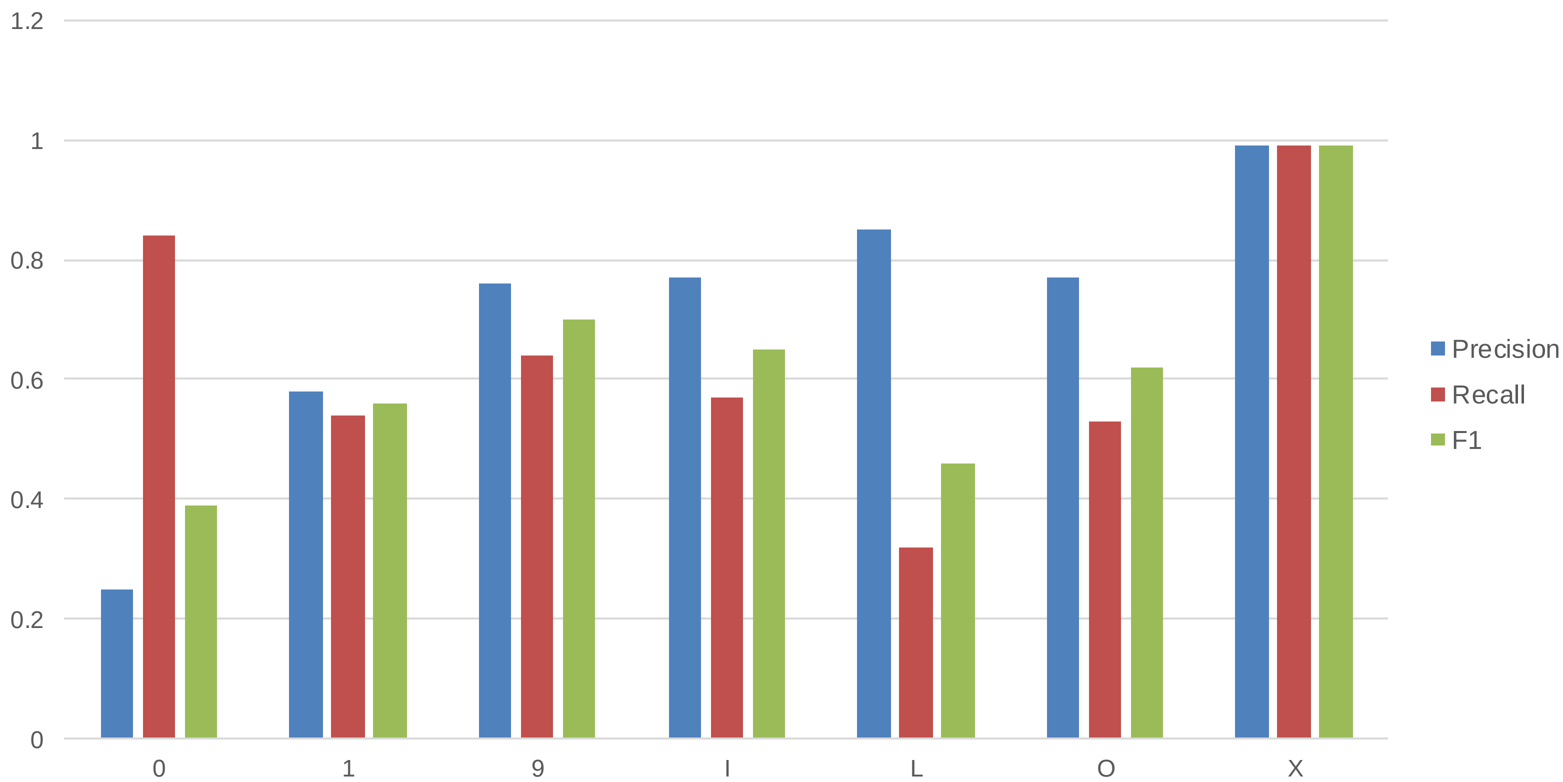
- Letters: trained for 4 and 14 epochs
- MNIST: trained for 3 and 3 epochs

Classification Report

- **TN / True Negative:** when a case was negative and predicted negative
- **TP / True Positive:** when a case was positive and predicted positive
- **FN / False Negative:** when a case was positive but predicted negative
- **FP / False Positive:** when a case was negative but predicted positive

Precision = TP/(TP + FP)
Recall = TP/(TP+FN)
F1 Score = 2*(Recall * Precision) / (Recall +Precision)

Classification Report



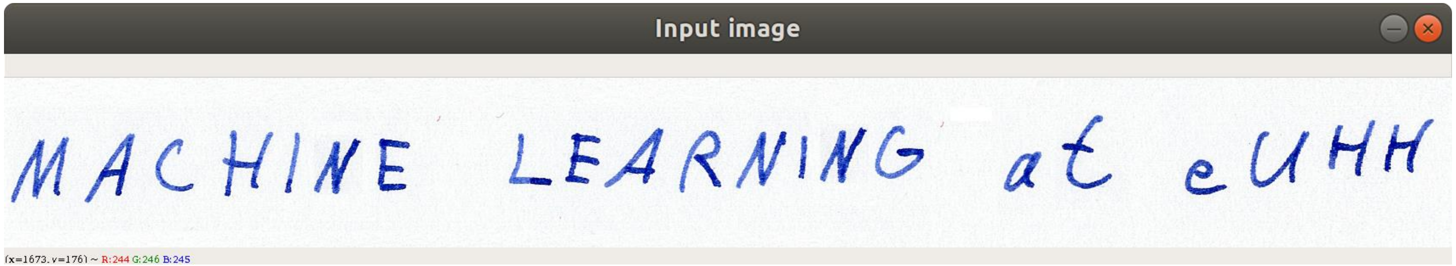
5. Image Processing

How to extract single letters (input to NN) from input image?

Solution:

1. Make image binary
2. Find contours using the open-source library OpenCV [3]
3. Extract rectangles containing contours
4. Fit rectangles to 28x28 greyscale image

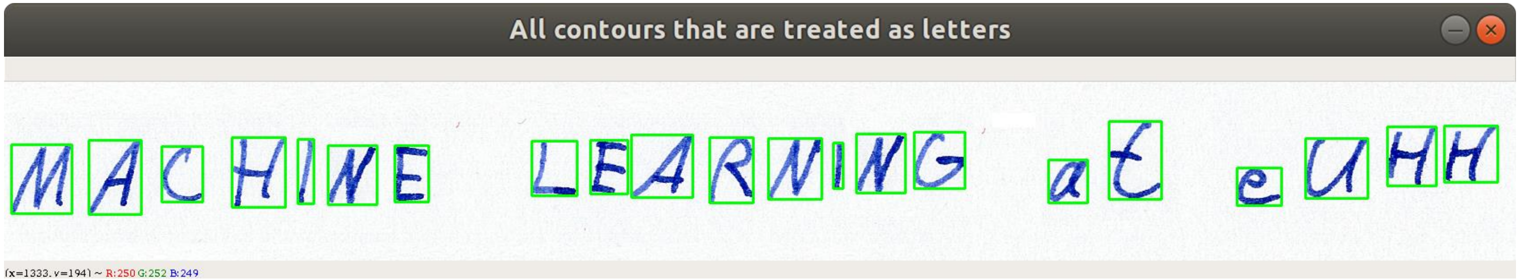
5.1 Make image binary



- Convert from RGB to greyscale
- Invert image
- Apply threshold setting values to 0 or 255

5.2 Find contours

- Use OpenCV's function **`findcontours()`** to find contours
- Disregard contours with area smaller than some threshold
- Draw rectangle around contours
- Assign rectangles to words based on distance
- Draw rectangles on original image



5.3 Extract rectangles from input image

- Smoothen input image, make binary
- Extract rectangle from image (with width w, height h)

5.4 Fit rectangles to 28x28 greyscale image

- Center rectangle in square matrix of size $\max(w, h)$
- Resize matrix to 26x26, center in matrix of size 28x28

Extracted letters

Input to neural network

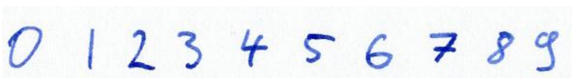
6. Final product

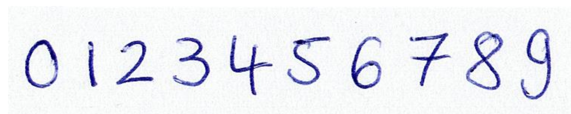
Command line tool with three input parameters:

- Path to input image
- Modeltype: Choose between balanced, letters or MNIST
- Option to show plots (default off)

```
jannik@jannik-ThinkPad-ubuntu:~/HandwritingRecognition/src/image_handwriting_recognition$ python3 image_handwriting_recognition.p
y --img data/ML_at_eUHH.png --modeltype balanced
Imagepath:  data/ML_at_eUHH.png
Modelpath:  models/cnn_emnist_balanced_ep3.model
Found 21 contours that are treated as letters.
Seperated the image into 4 words.
You may have written:  MACHLNE LEARNLNG at eUHH
```

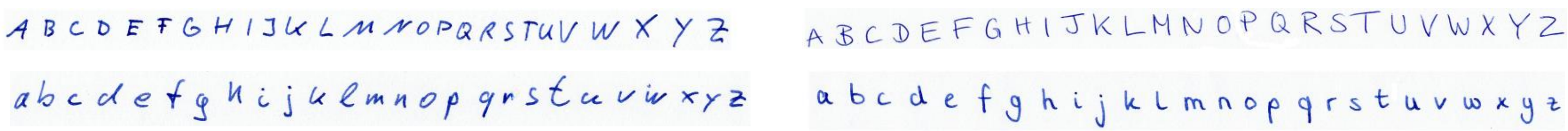
Product performance on handwritten digits





Input	balanced	MNIST
Digits 1	QL2345G78g 4 wrong	0123456789 0 wrong
Digits 2	0Y2345G78G 3 wrong	0123456788 1 wrong

Product performance on handwritten letters



Input	balanced	letters
Caps 1	ABCDEFGHLJULMMQPQRSTUWWKYZ 6 wrong	ABCDEFGHHZKLMMDPQRSFUVWXYZ 5 wrong
Caps 2	ABCDEFGHYJKGMW0PQRSTUVWXYZ 4 wrong	ABCDEFGKKJKLMNQPQRSTUVWXYZ 3 wrong
Small 1	abCdefgMGJUBMn0PYMStUUWKYZ 9 wrong	ABCDEFGKCDUBMMOPQRSTUVWXYZ 6 wrong
Small 2	abCdefgKKJKKMA0P9rStUVWKYZ 7 wrong	UBCDEFGHKDKKMNOPQFSTUVWXYZ 5 wrong

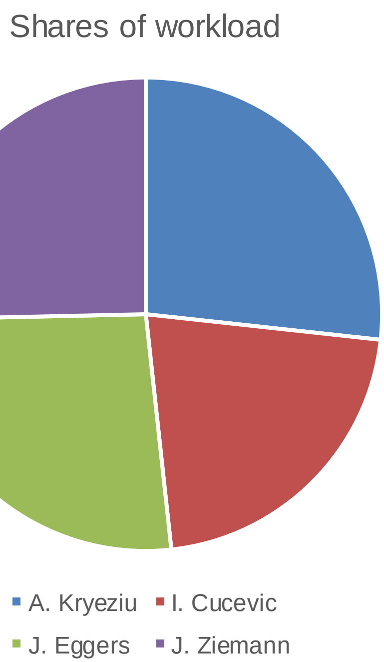
Product Performance

- Digits: 10-class MNIST model works very well
  47-class balanced produces some errors
- Letters: 26-class letters model works slightly better than 47-class balanced model
- Typical false classifications: 1, I ➔ L, H, Y; n, N ➔M, W; O, 0 ➔ Q, H➔K , g ←➔9,

7. Lessons learned

- Don‘t underestimate training time and data preprocessing
- Write down decisions / plans / ideas etc.
- Be adaptable, find compromises
- Try different approaches to tasks
- Get a good PC.

Who did what?



- A. Kryeziu:
  - Project Management
  - CNN Architecture (Keras, tensorflow)
  - Performance Analysis (Charts and model report)
- I. Cucevic:
  - Research alternative approaches (different dataset, RNN)
  - Presentation design and uniformity
- J. Eggers:
  - Implement neural network from scratch
  - Input processing (separating and predicting letters/digits)
- J. Ziemann:
  - Documentation (Gantt, charts and diagrams)
  - Training and testing tensorflow networks

References

[1] Abadi, Martín, Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J. et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[2] Bengio, Y., Courville, A., Goodfellow, I. (2016). Deep Learning. MIT Press. See https://www.deeplearningbook.org/

[3] Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools.

[4] Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. Retrieved from http://arxiv.org/abs/1702.05373 .

[5] Goyal, P., Pandey, S., Jain, K. (2018). Deep Learning for Natural Language Processing. Apress.

[6] Nielsen, M. A. (2018). Neural Networks and Deep Learning. Determination Press. See http://neuralnetworksanddeeplearning.com/

Thank you for listening!