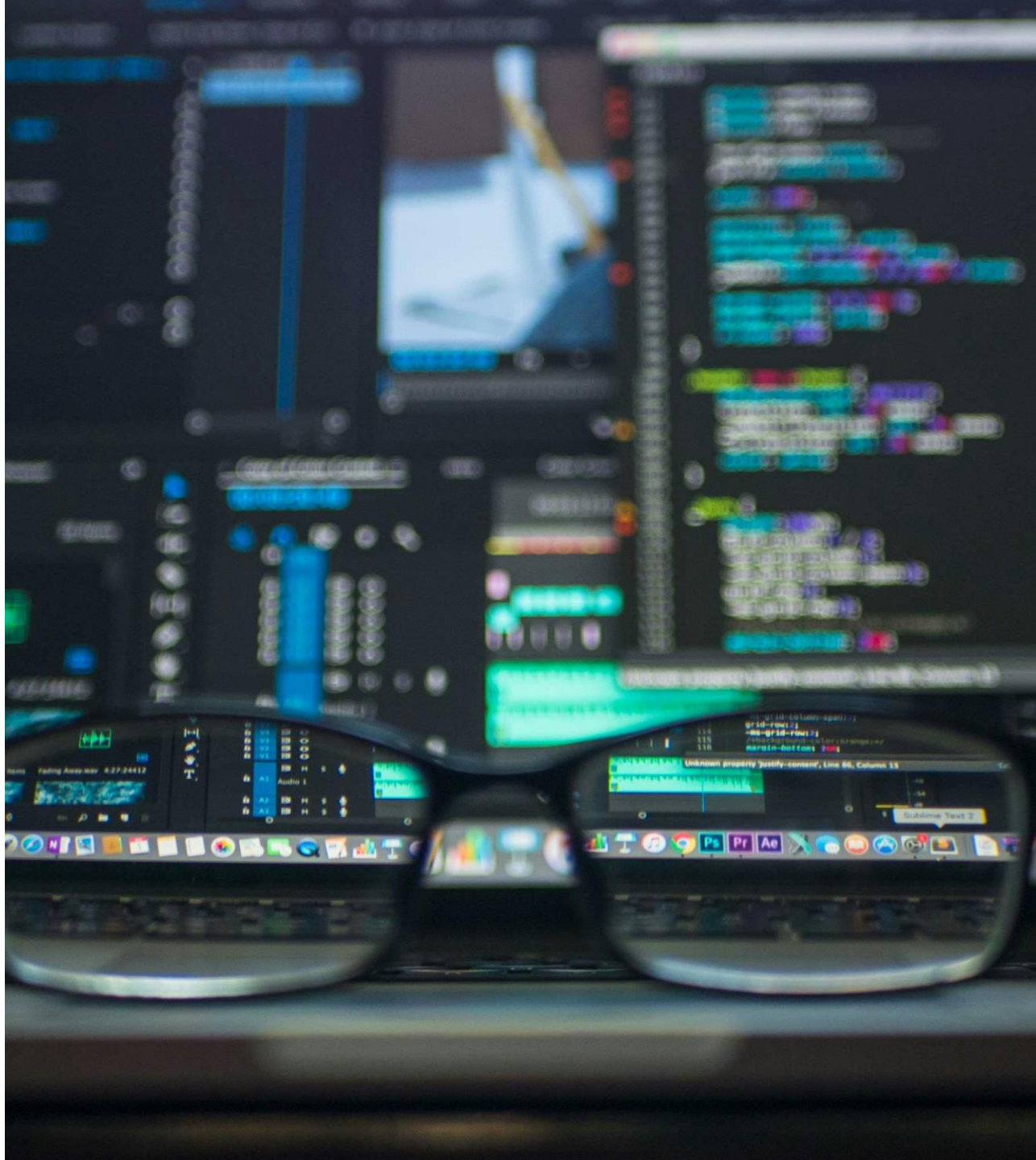
Solving CAPTCHAs with OCR

Maximillian Klar (8470260)
Moritz Landwehr (8476593)
Tristan Schwarz (8485182)
Stefan Cames (8471128)

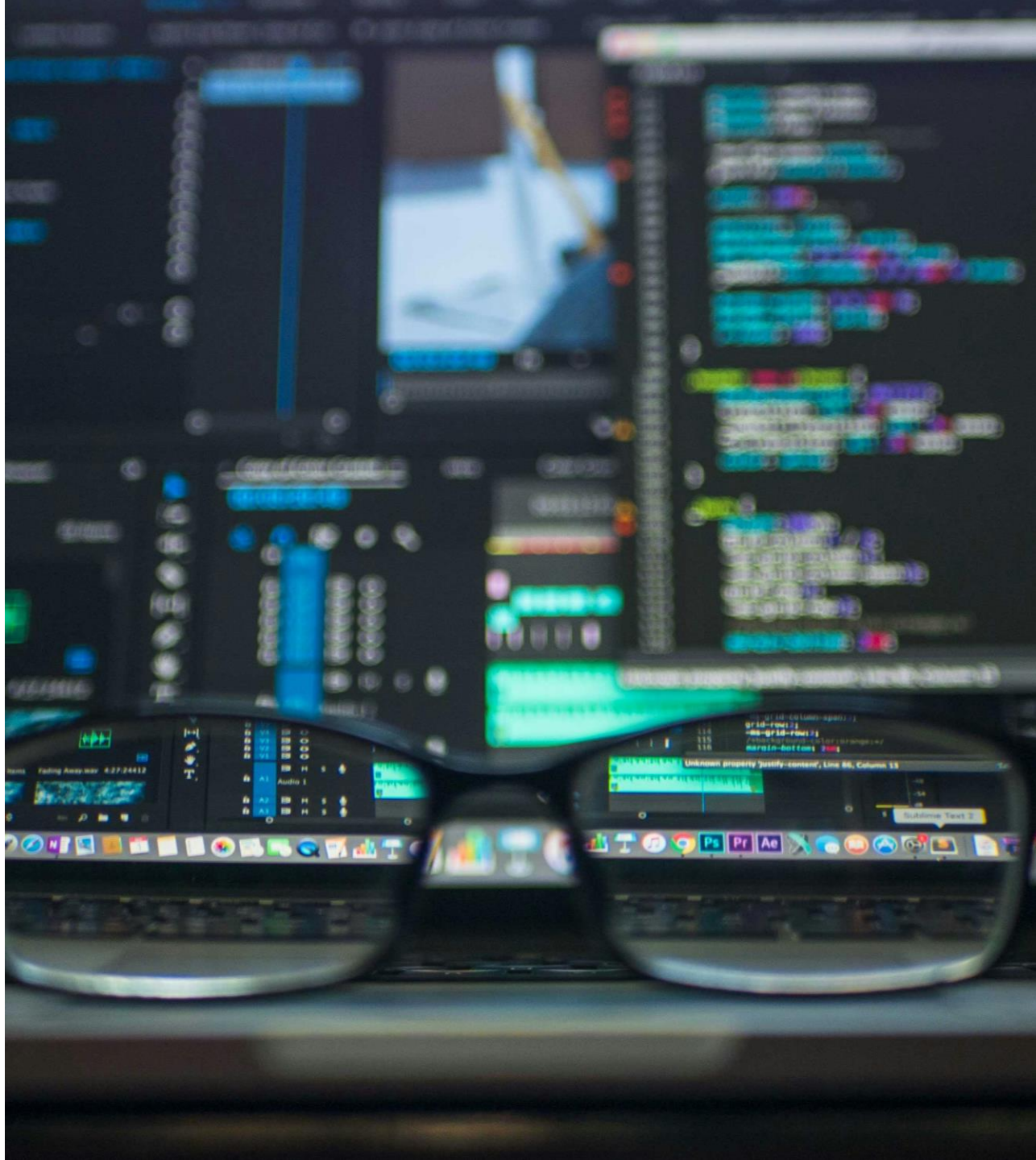
Frankfurt am Main,
25.09.2025

Agenda

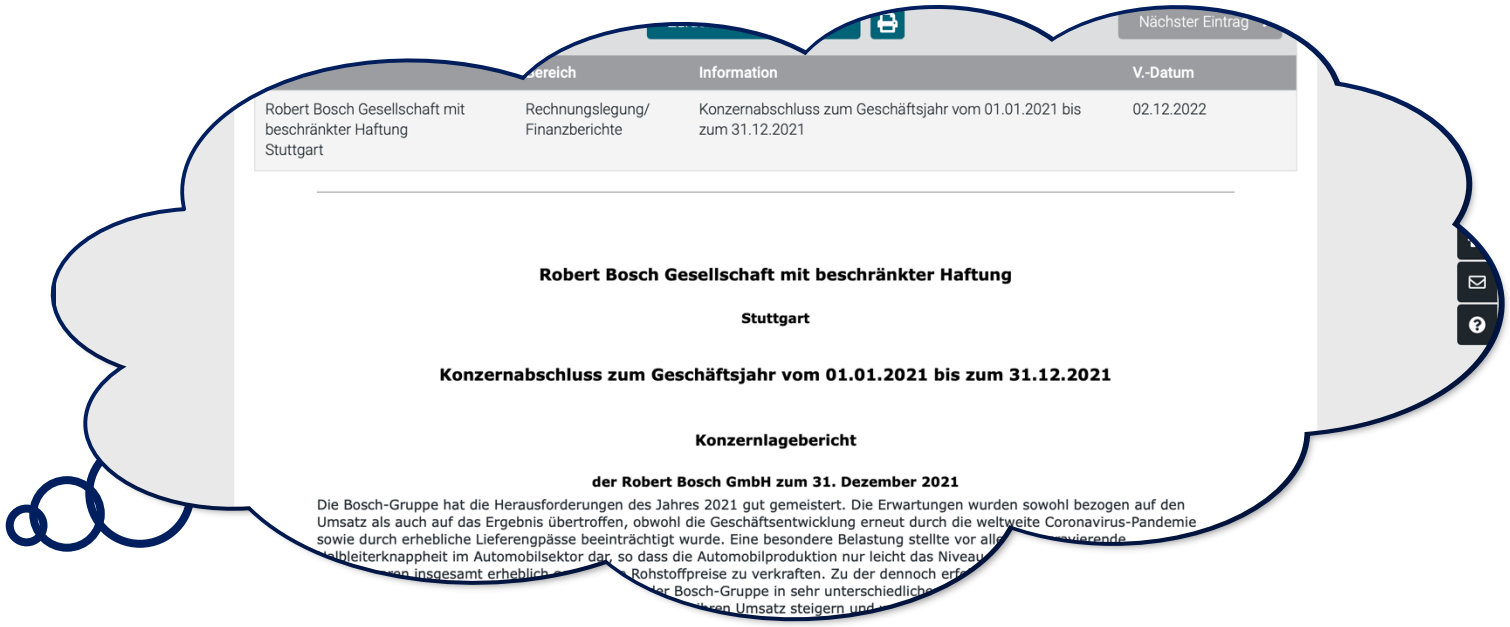
1. Goal & motivation
2. General idea of OCR
3. Data preparation
4. Our approach
5. Application
6. Conclusion



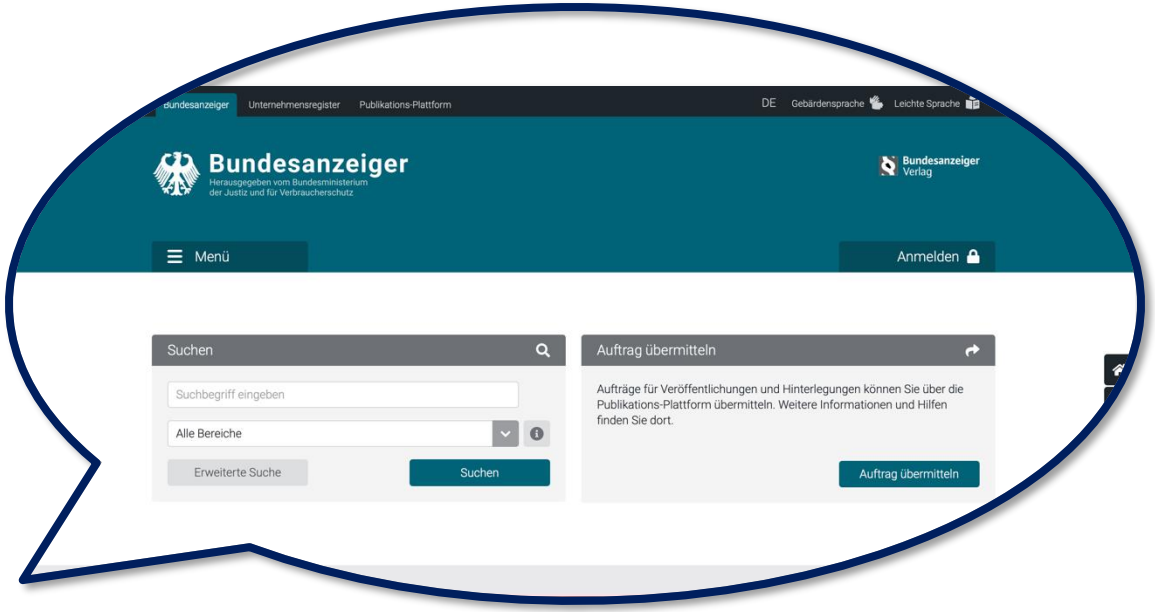
1 Goal & motivation



Where do we get annual reports for German companies from?

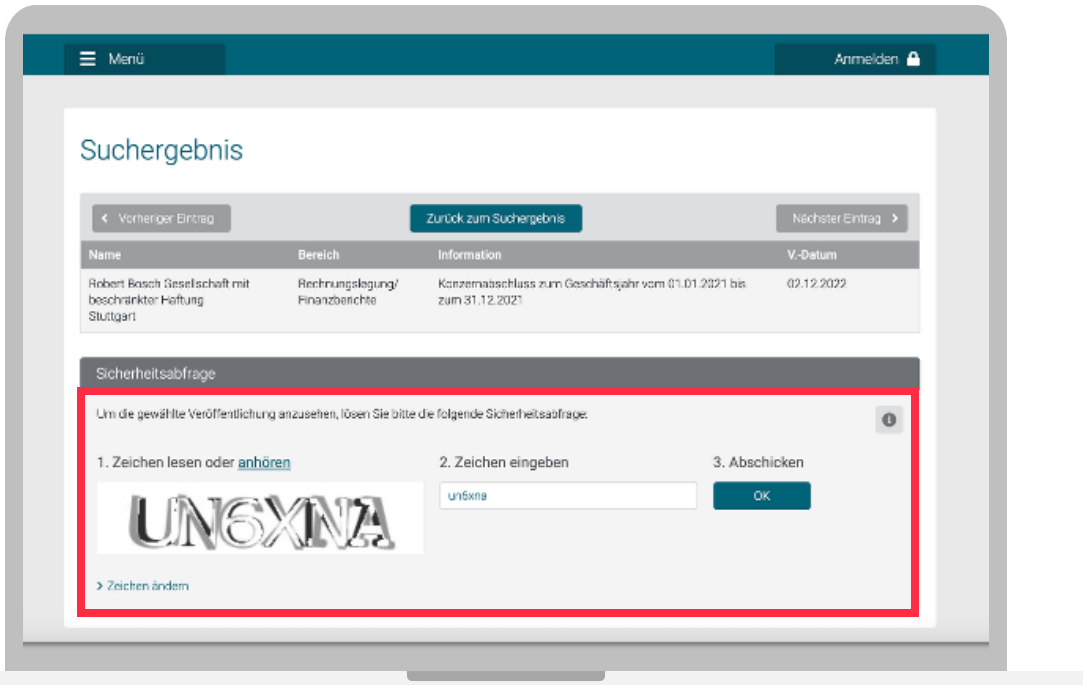


Where do we get annual reports for German companies from?

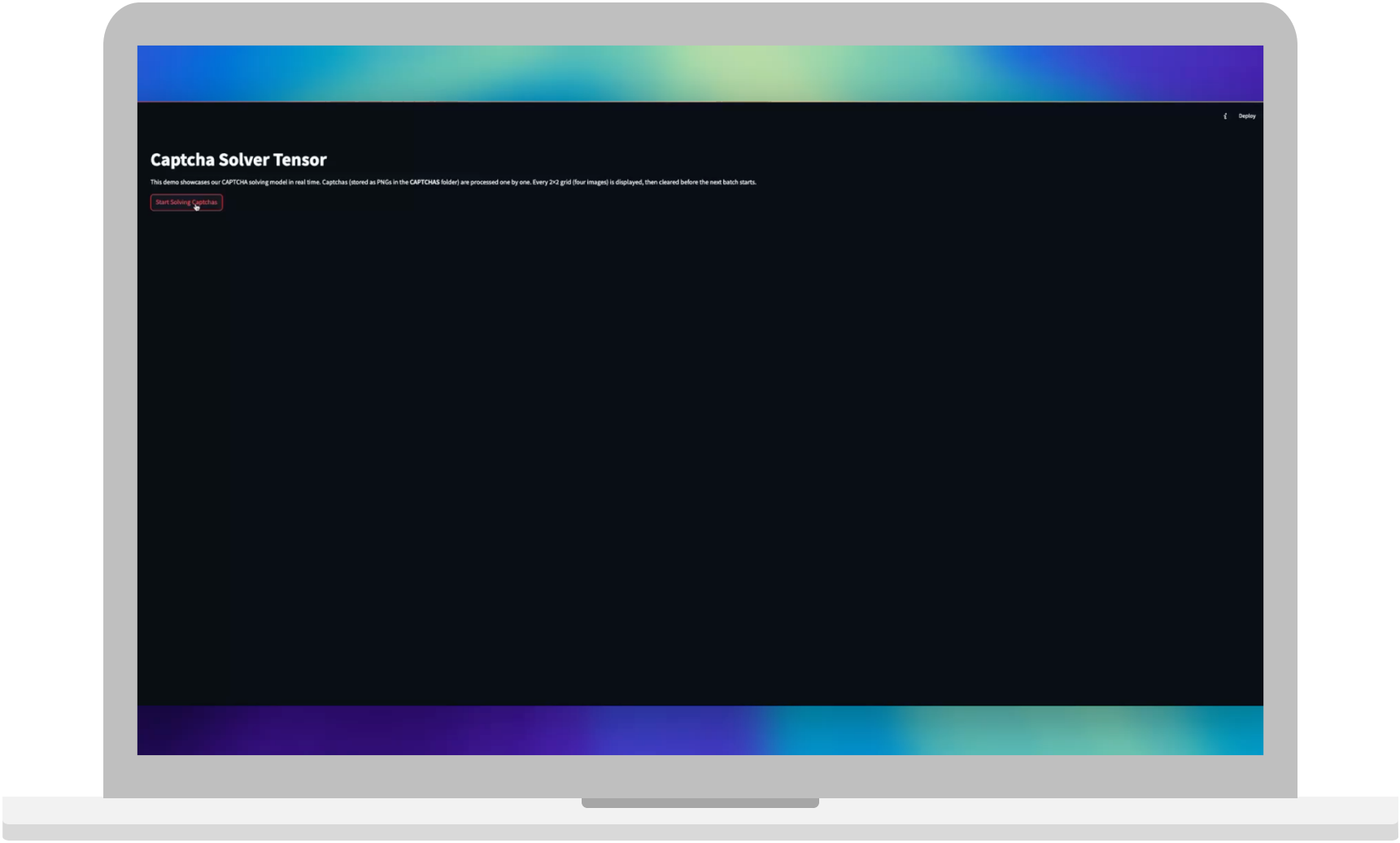


One key resource is Bundesanzeiger

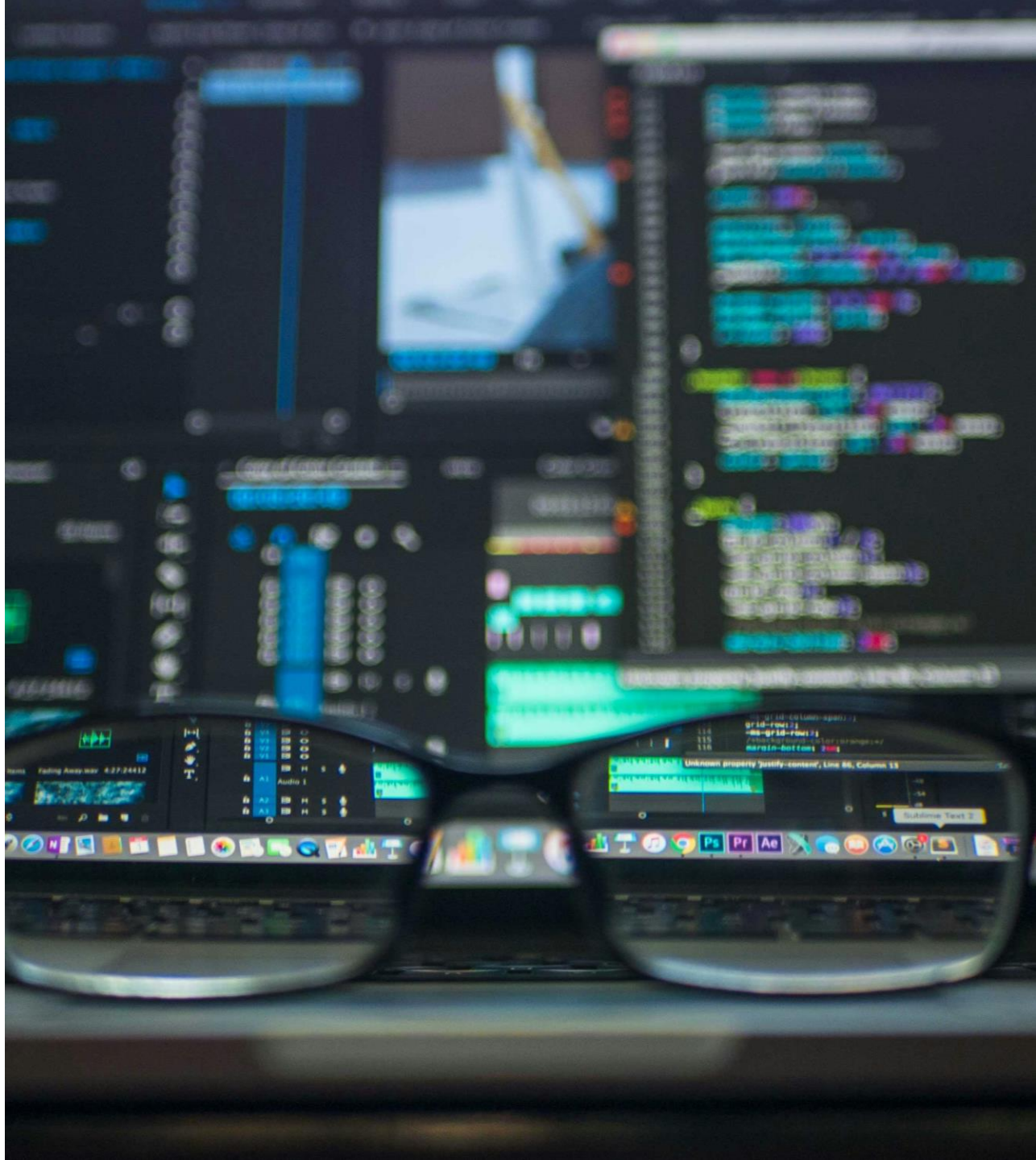
CAPTCHA!!!



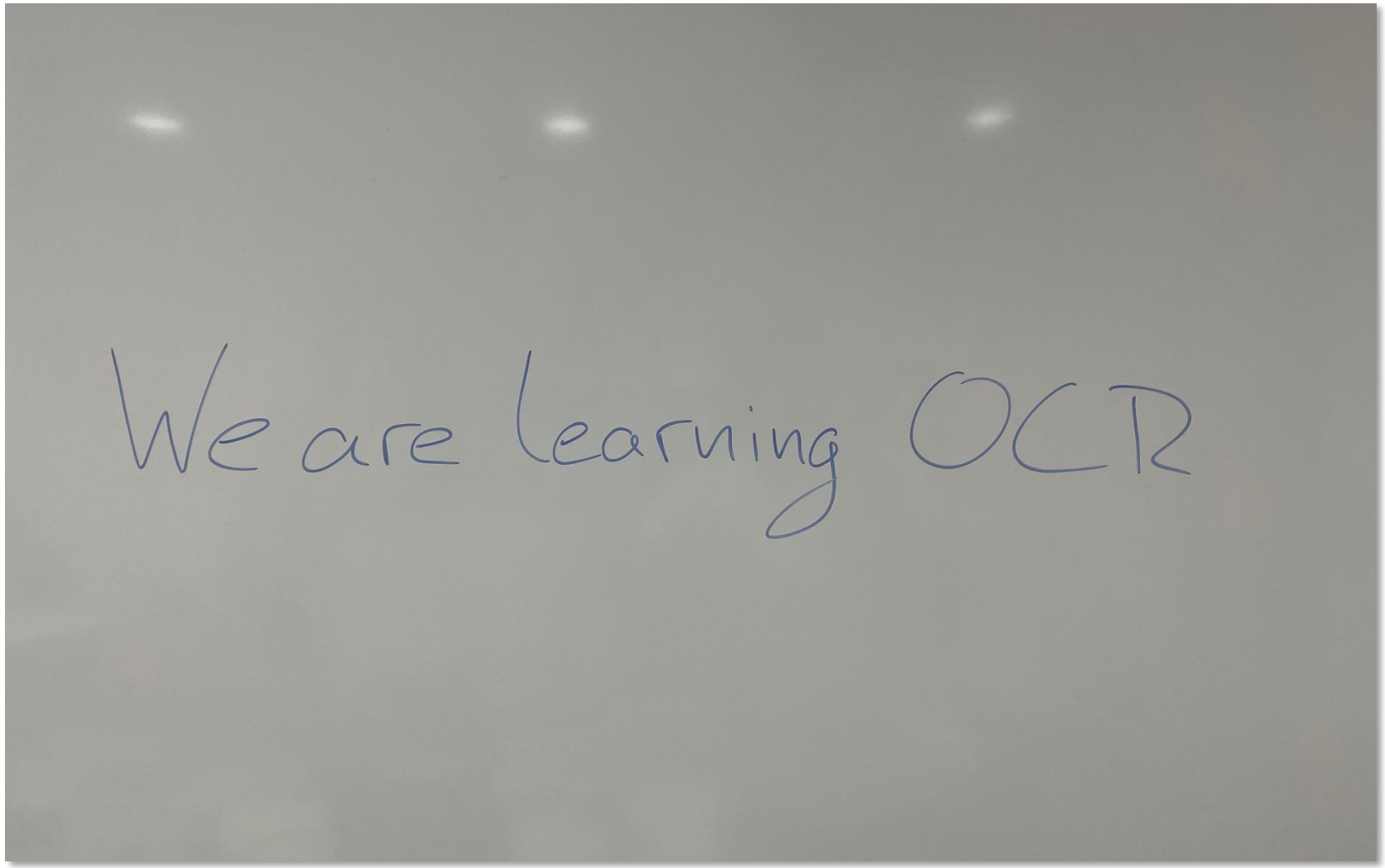
Therefore, automating the solving of CAPTCHA allows for faster workflows



2 General idea of OCR



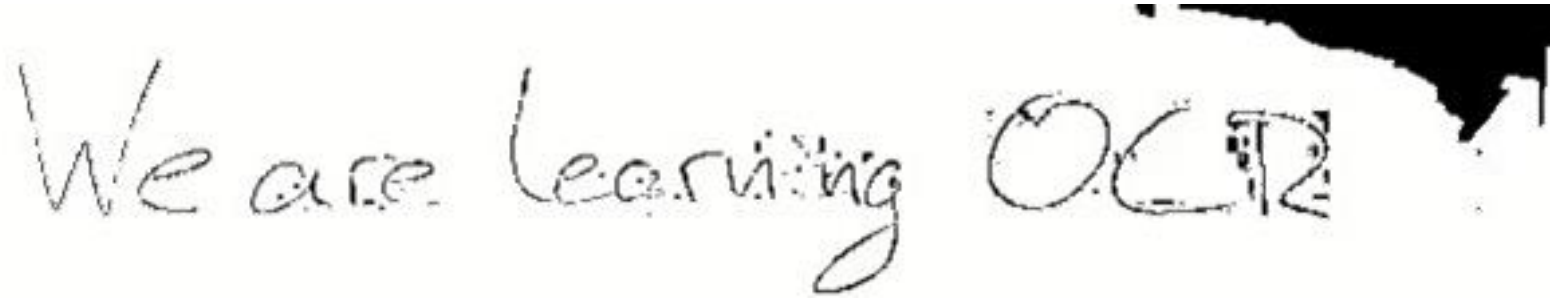
An ordinary picture is fed into the OCR algorithm…



- **Capture text** via a scanner, camera, or digital upload
- **Ensure sufficient resolution**, brightness, and contrast
- **Image quality directly effects OCR performance**

Sources: Klippa; IQT; Roboflow; Youtube – Computerphile; Youtube - IBM

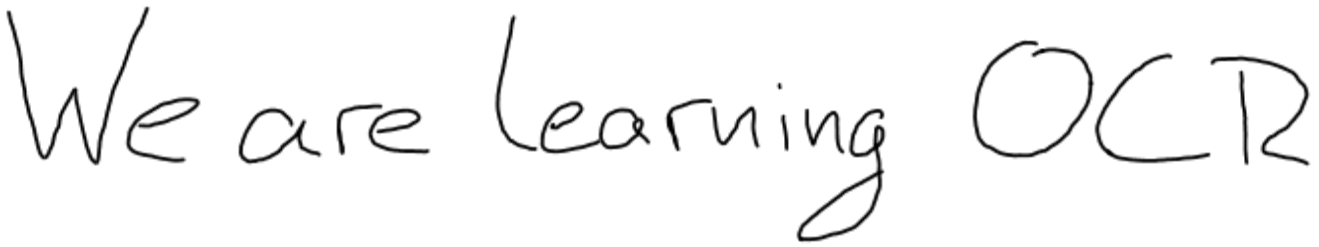
… the picture is then rendered into black and white…



- **Convert the images into binary** (black & white) form: Get differentiation between text versus the background
- Apply global or adaptive thresholding algorithms
- Reduce complexity (e.g., color) while preserving character structure

Sources: Klippa; IQT; Roboflow; Youtube – Computerphile; Youtube - IBM

… previous noise in the picture is reduced by further algorithms…



- **Eliminate visual artifacts** (speckles, dust, stains, …)
- Smooth character edges and **close broken strokes**

Sources: Klippa; IQT; Roboflow; Youtube – Computerphile; Youtube - IBM

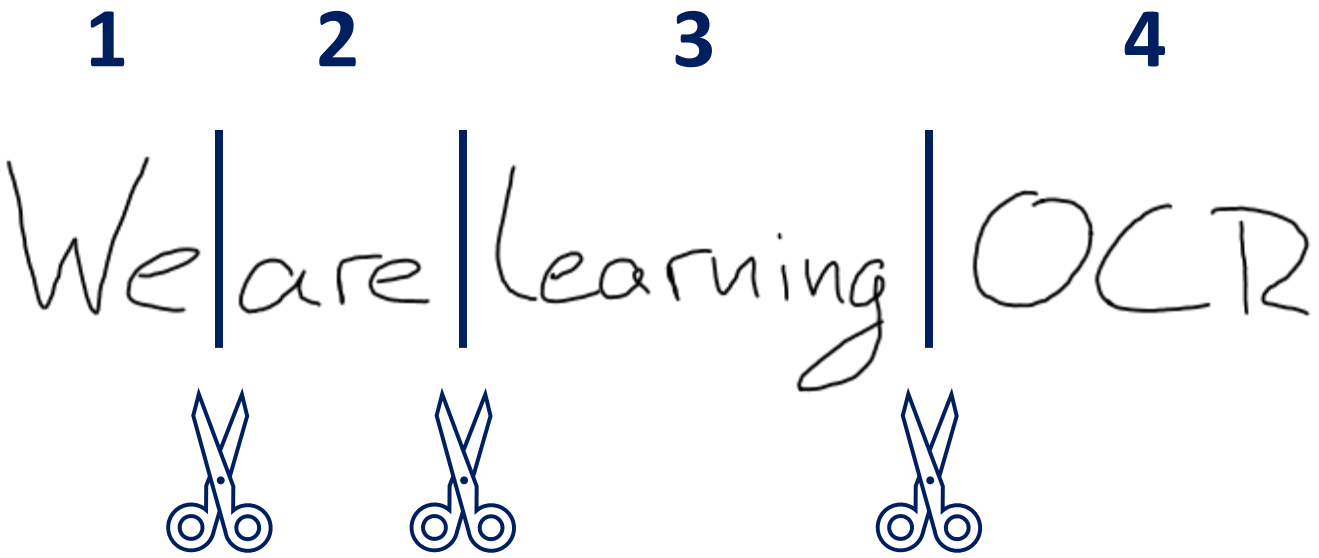
… each line of words is determined by the white space between the words …

1

- **Identify horizontal regions** containing text content (black vs. white regions)
- **Leverage pixel density** projections to detect line boundaries
- **Separate text lines** as foundation for further segmentation

Sources: Klippa; IQT; Roboflow; Youtube – Computerphile; Youtube - IBM

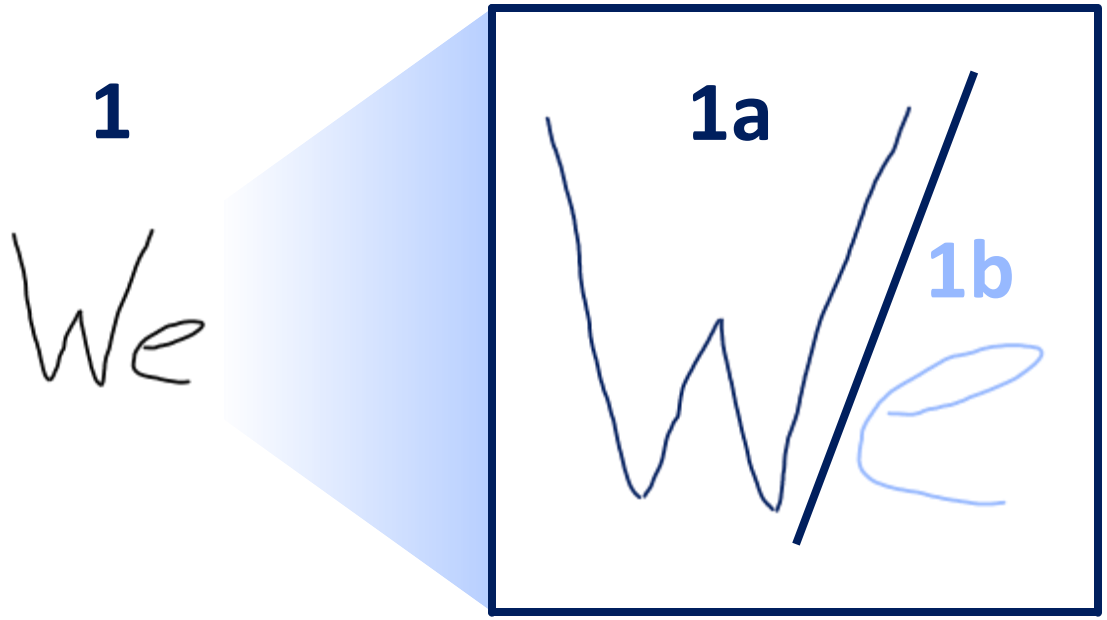
… in each line, the words are then split by horizontal white spaces…



- **Split each line into distinct word blocks**
- **Detect vertical whitespace** patterns between character groups
- **Maintain structural integrity** across different fonts and spacings

Sources: Klippa; IQT; Roboflow; Youtube – Computerphile; Youtube - IBM

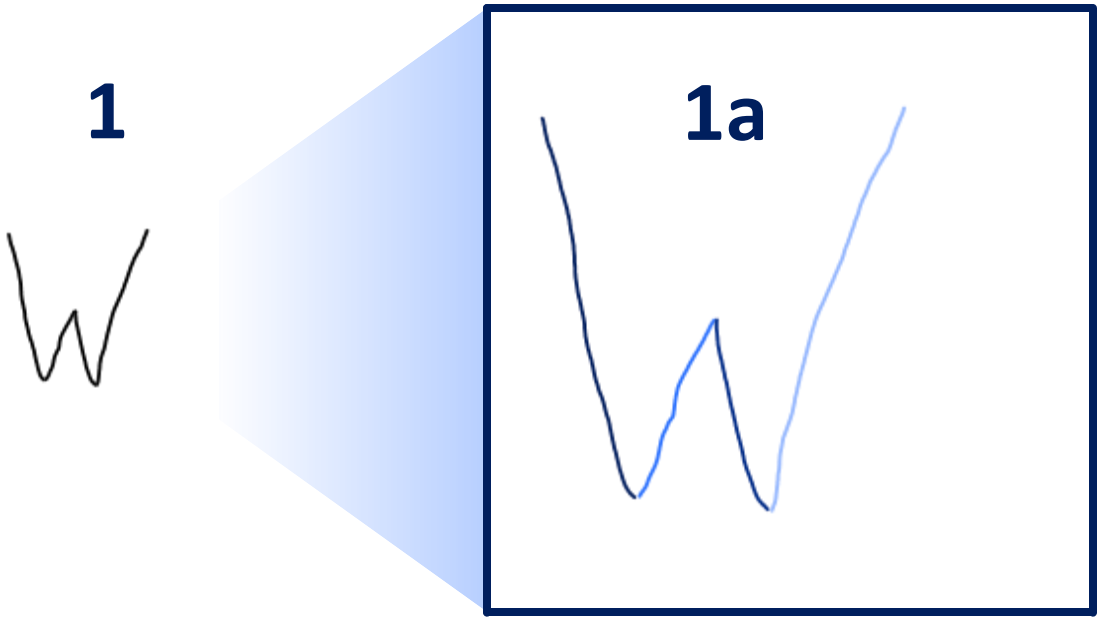
… then each letter is determined by further white spaces between the lines…



- **Segment characters by vertical projection profiles** (pixel valleys) or connected-component labeling (pixel blobs)
- **Address segmentation challenges**: touching pairs (e.g., “fi”), overlapping handwriting, distorted fonts (e.g., CAPTCHAs)
- Recognize that **segmentation quality is critical** as errors cascade into recognition mistakes
- This step is often done **implicitly by modern deep learning models**, as they can recognize full words/ sequences directly

Sources: Klippa; IQT; Roboflow; Youtube – Computerphile; Youtube - IBM

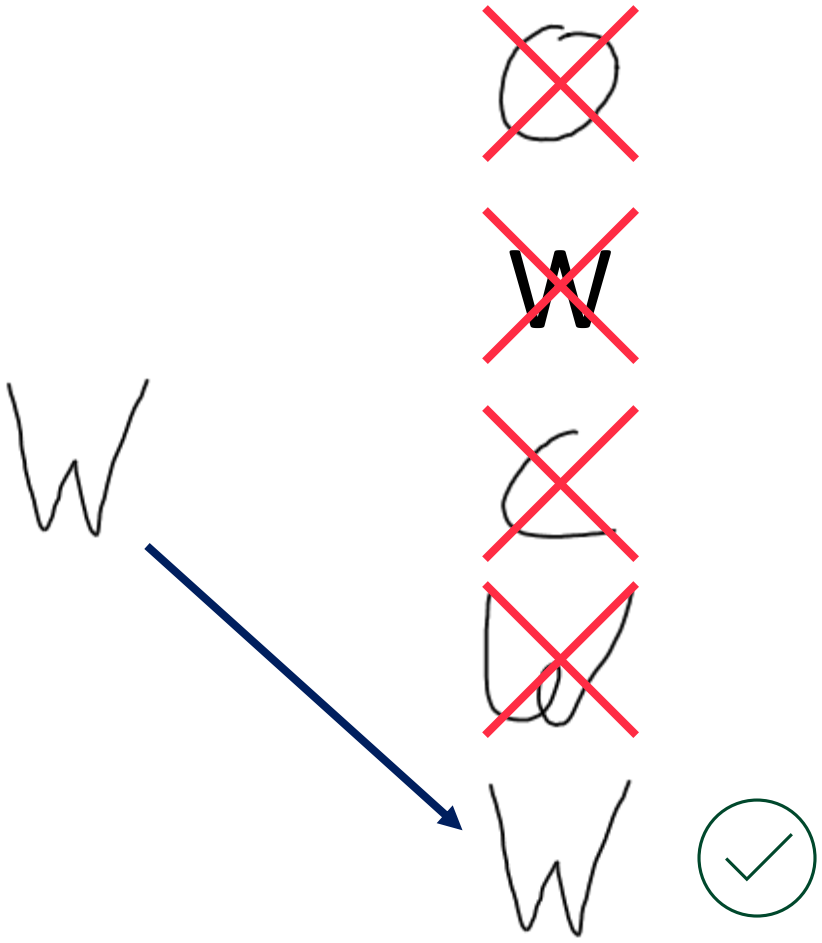
… lastly the features of each words are determined…



- **Transform each character image into a structured feature vector for classification**
- **Capture discriminative attributes**: pixel intensities, edges (Sobel/ gradients), corners, zoning densities, geometric traits (loops, symmetry)
- Ensure features encode a **unique "fingerprint" that separates similar characters** (e.g., "O" vs. "Q")
- In advanced OCR, **convolutional neural networks** (CNNs) **learn robust features automatically**, outperforming manual engineering

Sources: Klippa; IQT; Roboflow; Youtube – Computerphile; Youtube - IBM

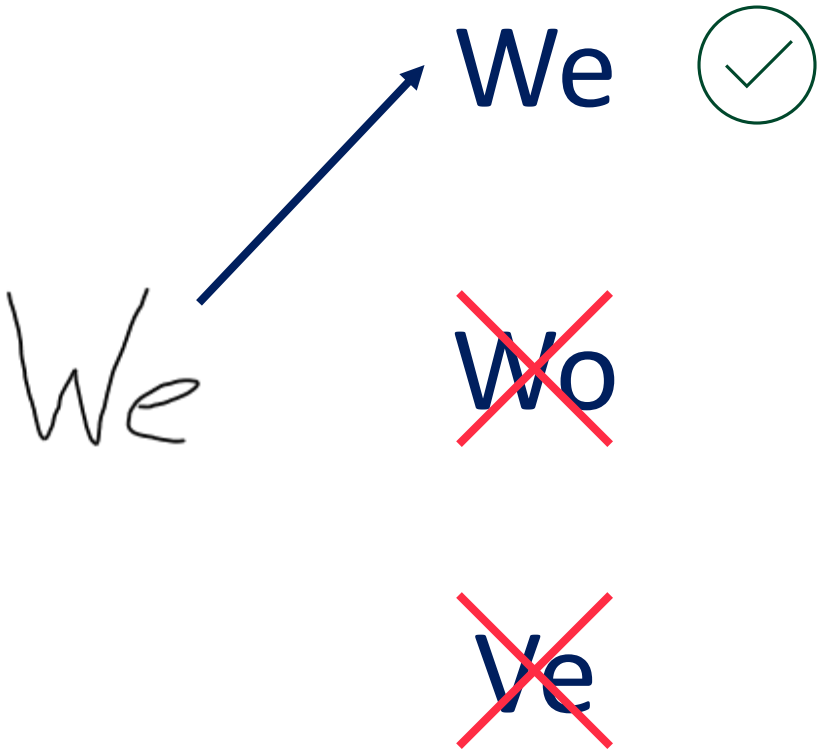
… these features are used to classify each letter based on an existing database…



- **Map feature vectors to predefined character classes/ dictionaries (A-Z, 0-9, symbols)**
- **Apply classifiers** ranging from template matching to statistical ML (k-NN, SVM, decision trees) to neural networks (CNNs, RNNs)
- **Output probabilistic predictions** with confidence scores (e.g., 70% W, 30% V)
- **Robust models handle noisy or distorted inputs**, making deep learning the standard in modern OCR

Sources: Klippa; IQT; Roboflow; Youtube – Computerphile; Youtube - IBM

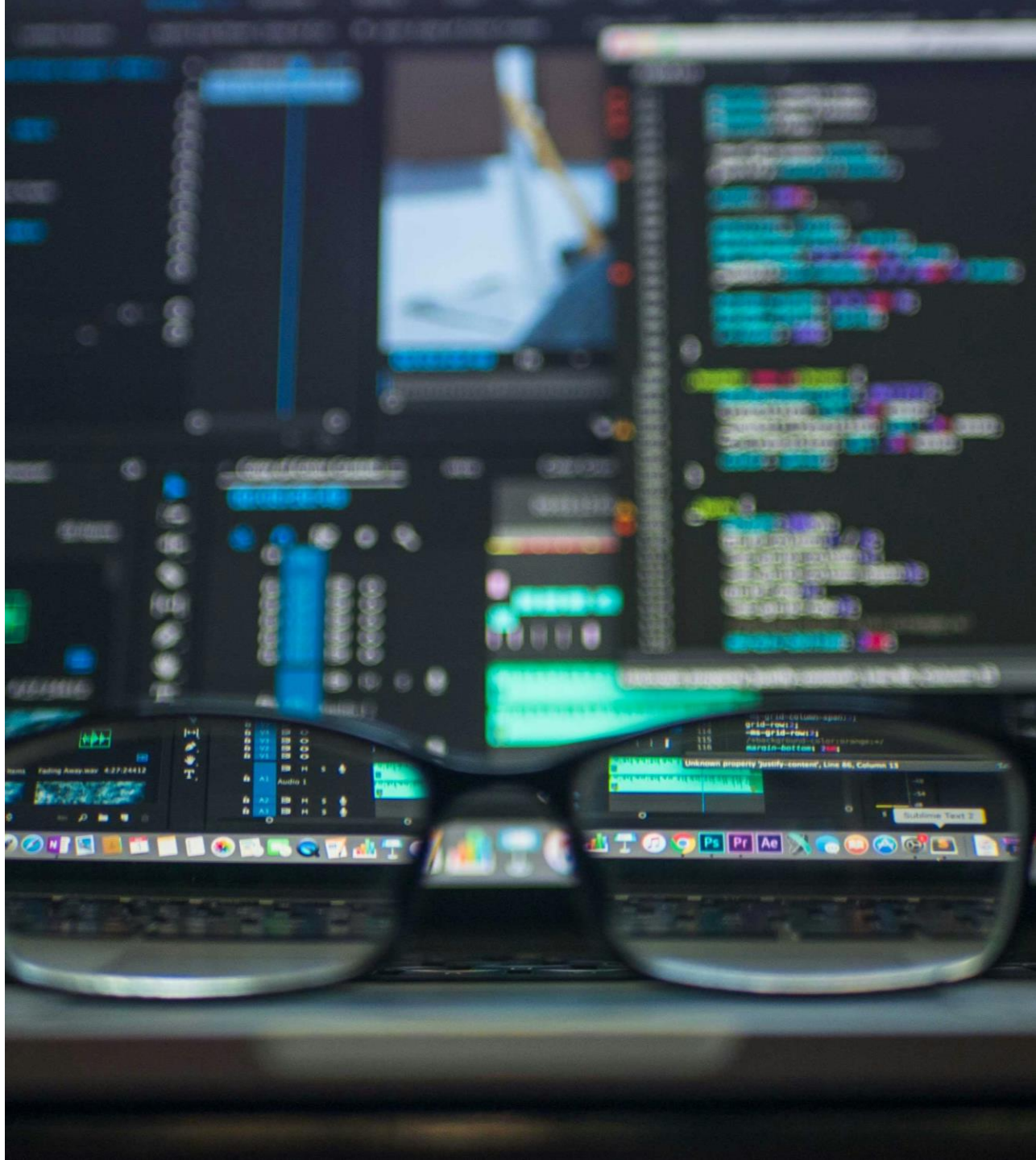
… to double check, the resulting words are checked against a dictionary.



- **Validate recognized characters against dictionaries and language rules** to ensure word plausibility (e.g., there is no “Wo” in the English language)
- **Correct frequent OCR confusion** (e.g., “O” vs. “0”) through contextual verification
- **Apply grammar or language models** to refine outputs (e.g., “c0mpany” -> “company”)
- **Incorporate domain-specific constraints** (license plates, structured codes, **CAPTCHAs**) to maximize reliability

Sources: Klippa; IQT; Roboflow; Youtube – Computerphile; Youtube - IBM

3 Data preparation



CAPTCHAs are solved by first using a general OCR to label initial data, which trains a baseline model that then generates further training data

01

Initial generation of training data using a **broad** but low-throughput **OCR model** primarily via 3rd-party providers

02

Cleaning of initial training data to eliminate noisy or mislabeled instances and transforming labels using **one-hot encoding**

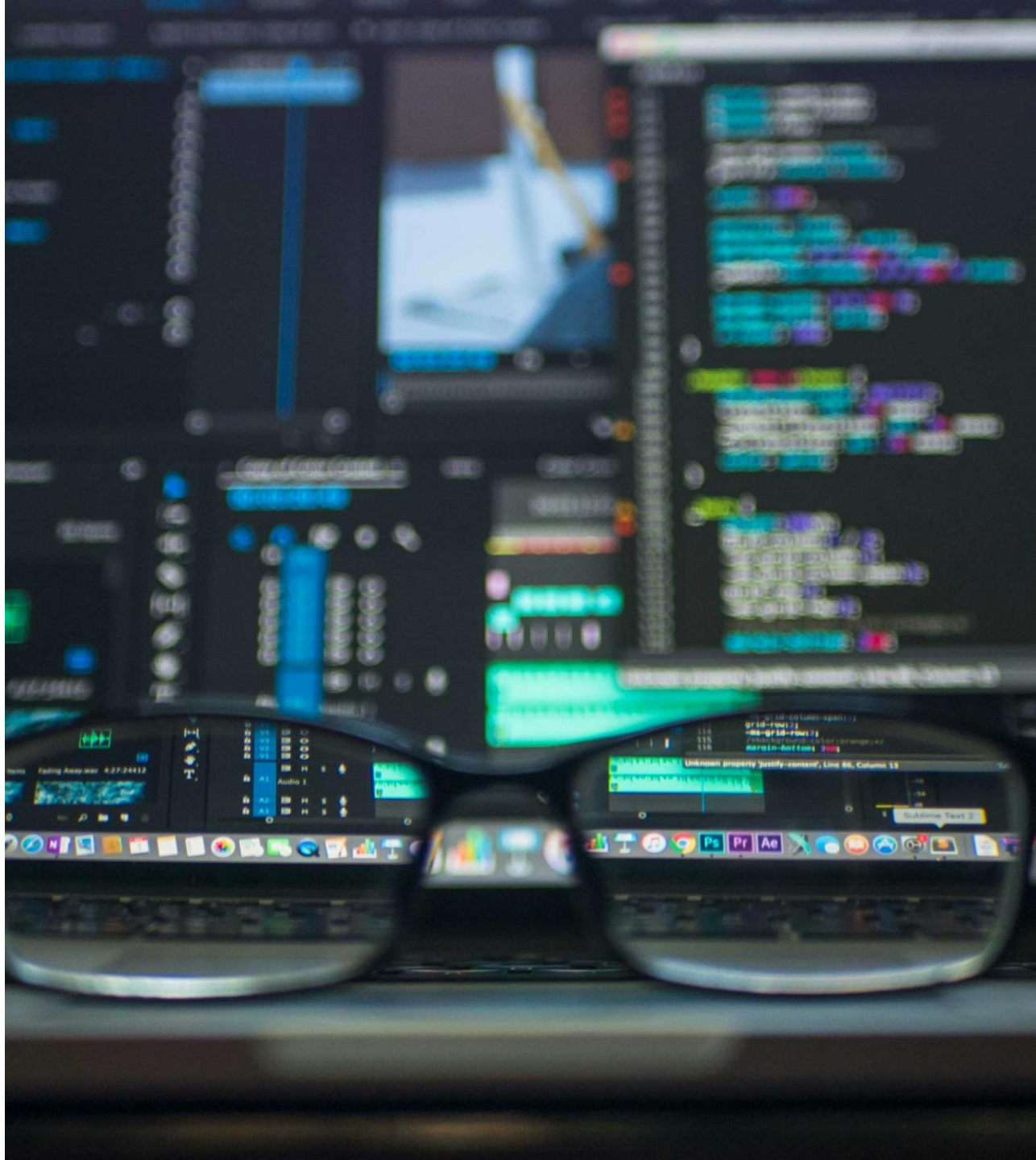
03

Training of an **initial baseline model** with **limited predictive accuracy**

04

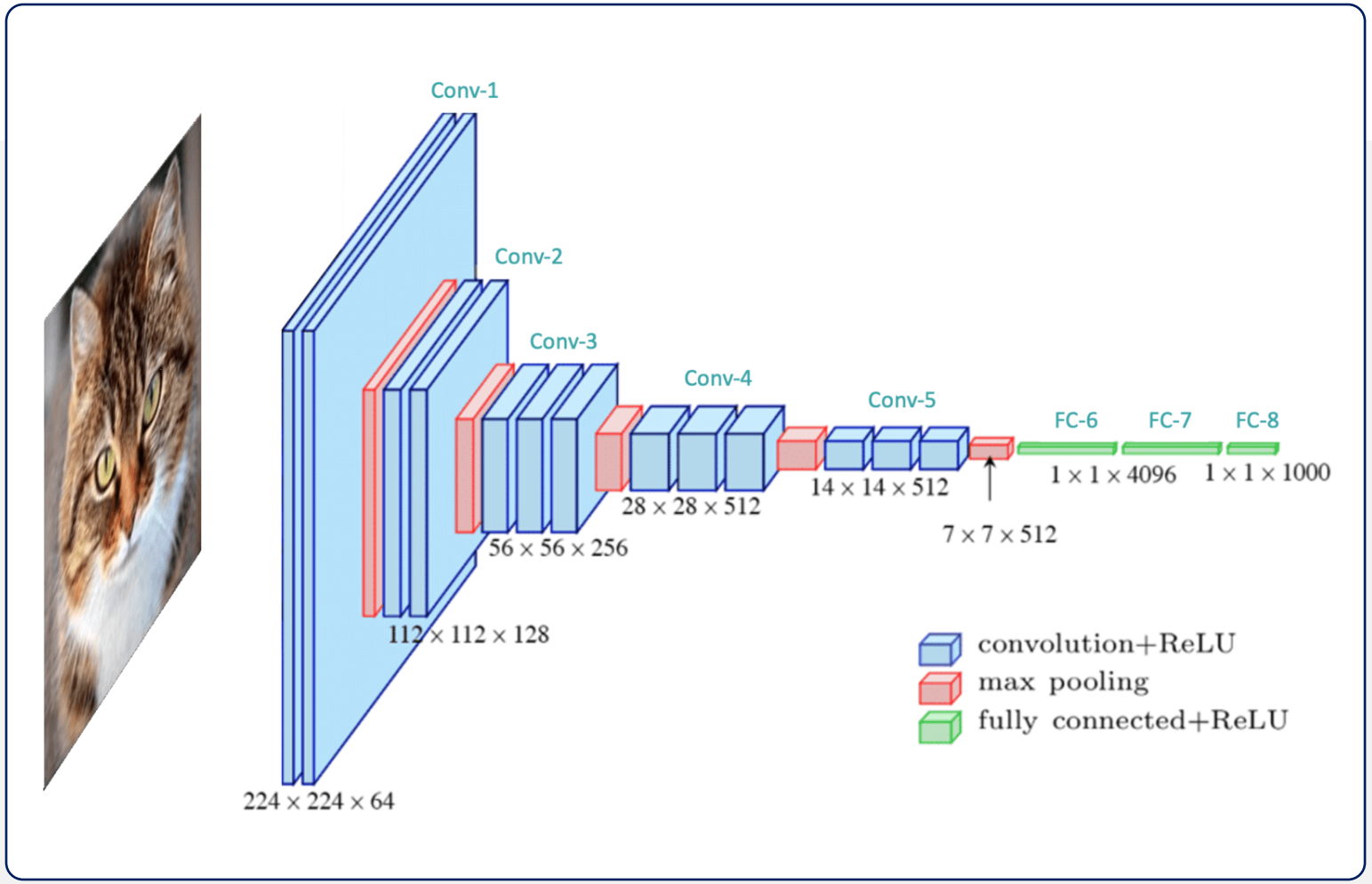
Using the model, if the **CAPTCHA is solved** successfully and **retrieves the annual report**, a **new labeled training instance** is created

4 Our approach



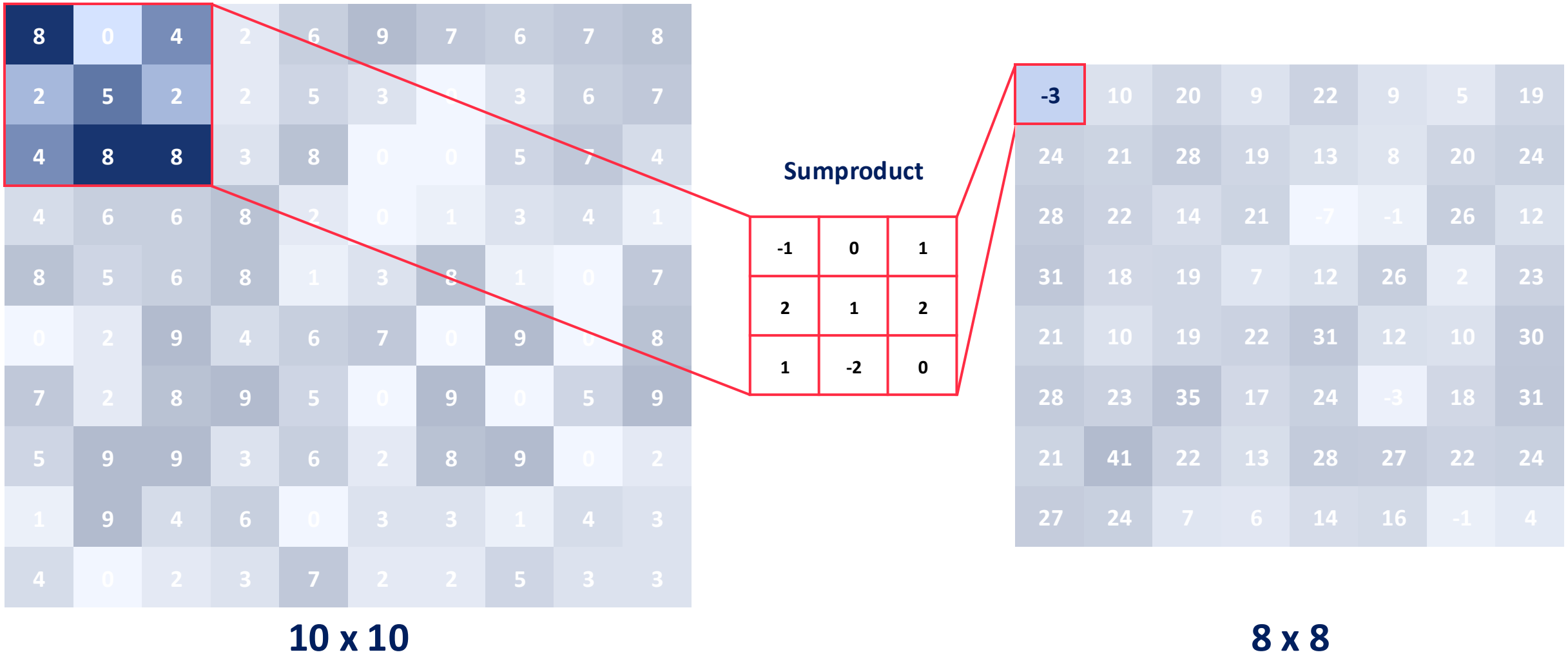
Convolutional neural networks use filters and data pooling to recognize patterns and features in images

- A convolutional neural network consists of several layers
- The **convolutional layer** runs filters to make the most important features of an image visible
- **Max pooling** downsizes the image data
- Finally, the compressed data is fed through a **fully connected feed forward neural network**



https://learnopencv.com/tag/convolutional-neural-network-diagram/

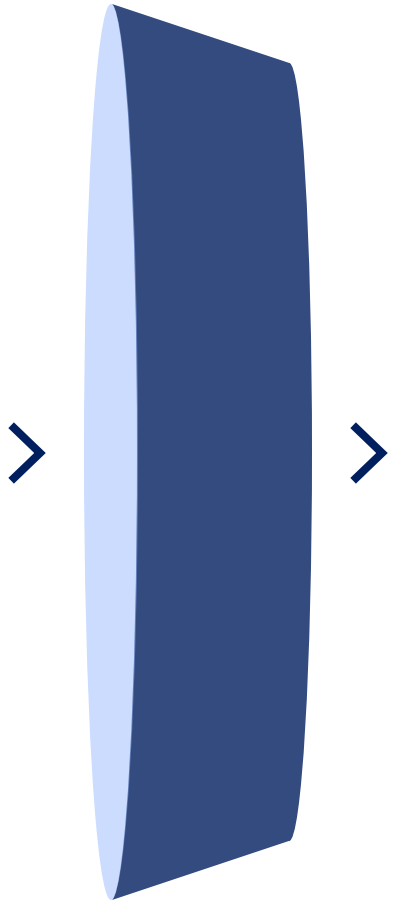
Convolutional layer



10 x 10

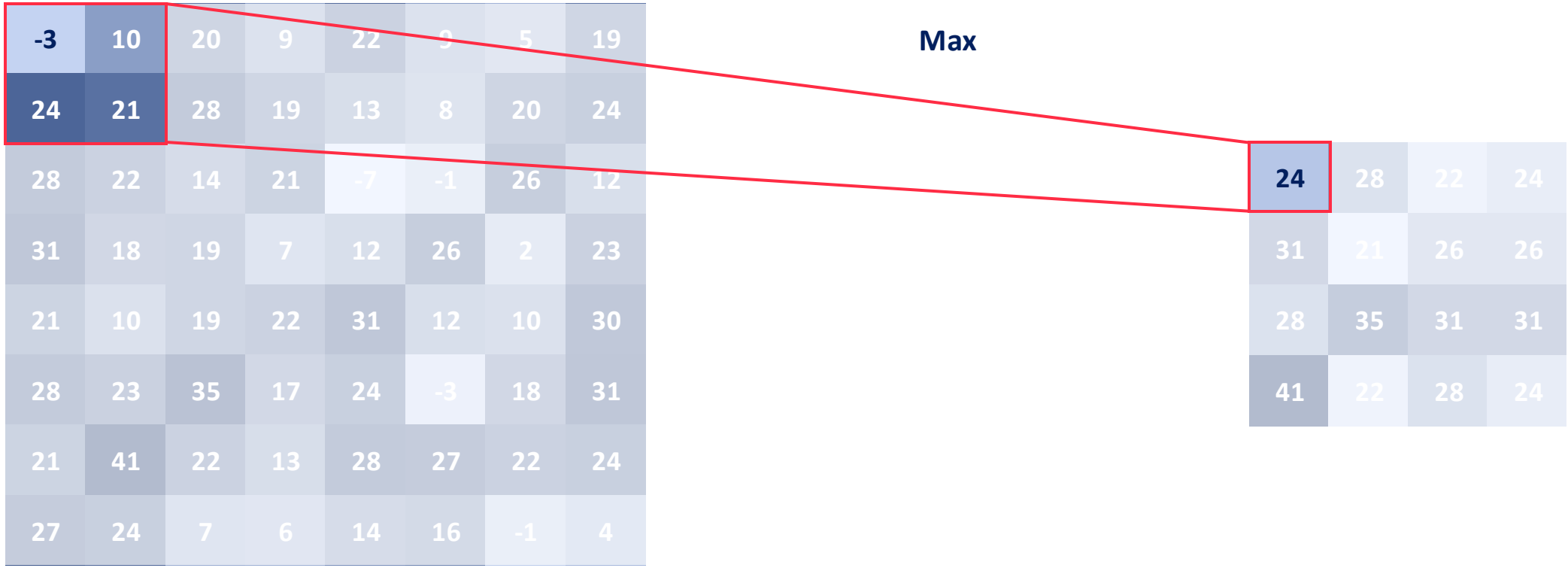
8 x 8

Convolutional layer



Sources: Github

Max Pooling

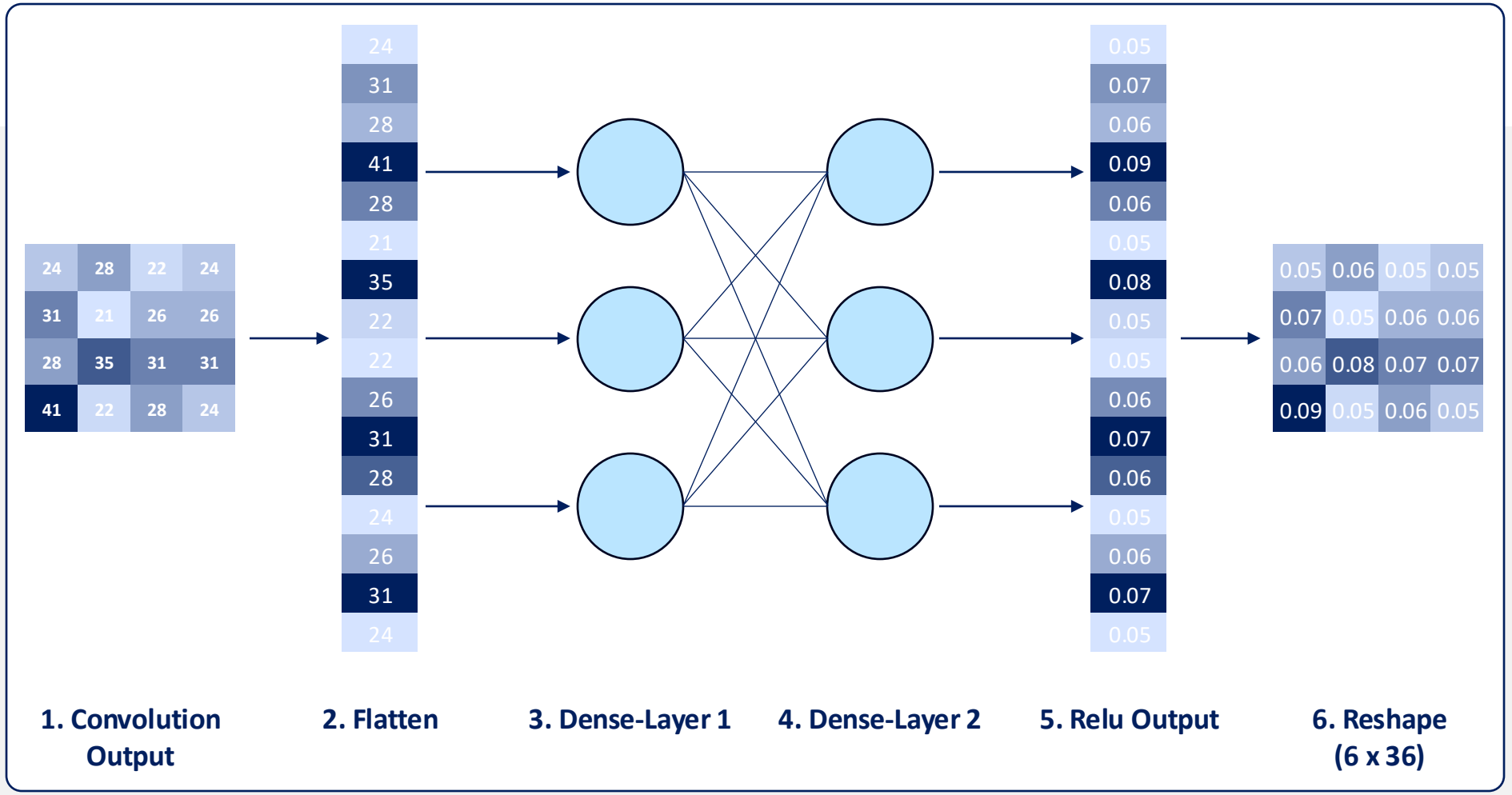


8 x 8

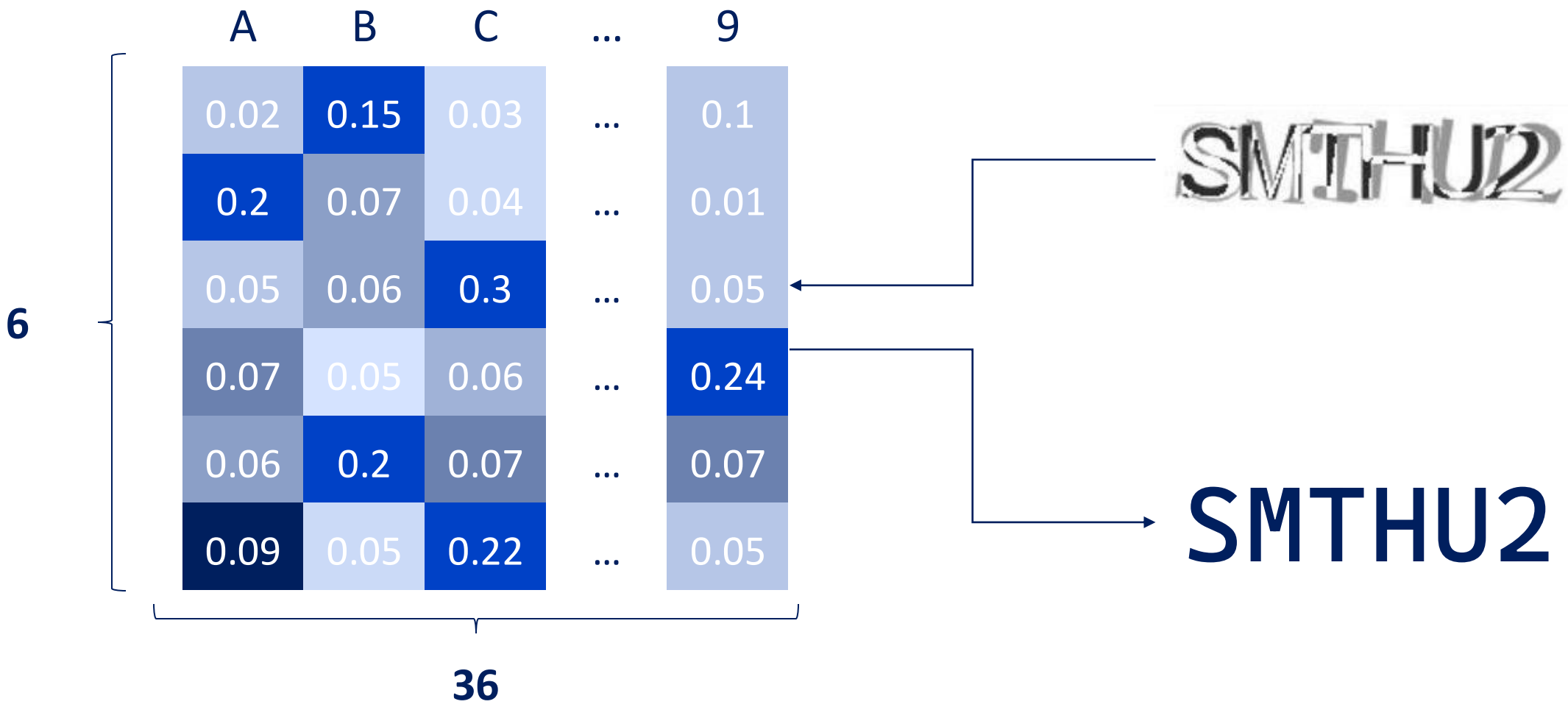
2 x 2

Feed forward neural network

- In the final step the filtered and pooled input is fed through a **fully connected feed forward neural network**
- The **dense layers** weigh the different parameters and create outputs
- These are then **trained for accuracy**



Final Output



The code

```python
# Modellarchitektur
input_shape = (88, 404, 1)  # Graustufen-Bild

model = models.Sequential([
    layers.Input(shape=input_shape),  # (88, 404, 1)
    layers.Conv2D(32, (3, 3), activation='relu'),  # (86, 402, 32)
    layers.MaxPooling2D((2, 2)),  # (43, 201, 32)
    layers.Conv2D(64, (3, 3), activation='relu'),  # (41, 199, 64)
    layers.MaxPooling2D((2, 2)),  # (20, 99, 64)
    layers.Conv2D(128, (3, 3), activation='relu'),  # (18, 97, 128)
    layers.MaxPooling2D((2, 2)),  # (9, 48, 128)
    layers.Conv2D(256, (3, 3), activation='relu'),  # (7, 46, 256)
    layers.MaxPooling2D((2, 2)),  # (3, 22, 256)
    layers.Flatten(),  # (3 * 22 * 256)
    layers.Dense(1024, activation='relu'),
    layers.Dropout(0.4),
    layers.Dense(6 * num_classes, activation='softmax'),
    layers.Reshape((6, num_classes))
])

# Modell kompilieren
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

The code

```
63  # Modellarchitektur
64  input_shape = (88, 404, 1)  # Graustufen-Bild
65
66  model = models.Sequential([
67      layers.Input(shape=input_shape),  # (88, 404, 1)
68      layers.Conv2D(32, (3, 3), activation='relu'),  # (86, 402, 32)
69      layers.MaxPooling2D((2, 2)),  # (43, 201, 32)
70      layers.Conv2D(64, (3, 3), activation='relu'),  # (41, 199, 64)
71      layers.MaxPooling2D((2, 2)),  # (20, 99, 64)
72      layers.Conv2D(128, (3, 3), activation='relu'),  # (18, 97, 128)
73      layers.MaxPooling2D((2, 2)),  # (9, 48, 128)
74      layers.Conv2D(256, (3, 3), activation='relu'),  # (7, 46, 256)
75      layers.MaxPooling2D((2, 2)),  # (3, 22, 256)
76      layers.Flatten(),  # (3 * 22 * 256)
77      layers.Dense(1024, activation='relu'),
78      layers.Dropout(0.4),
79      layers.Dense(6 * num_classes, activation='softmax'),
80      layers.Reshape((6, num_classes))
81  ])
82
83  # Modell kompilieren
84  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

- The **input shape of height = 88 and width = 404** is given by the loaded data in line 26
- The **scale is set to 1** – this means that it is grey scaled as we only have one single channel of visualization (RGB = 3)

The code

```
63  # Modellarchitektur
64  input_shape = (88, 404, 1)  # Graustufen-Bild
65
66  model = models.Sequential([
67      layers.Input(shape=input_shape),  # (88, 404, 1)
68      layers.Conv2D(32, (3, 3), activation='relu'),  # (86, 402, 32)
69      layers.MaxPooling2D((2, 2)),  # (43, 201, 32)
70      layers.Conv2D(64, (3, 3), activation='relu'),  # (41, 199, 64)
71      layers.MaxPooling2D((2, 2)),  # (20, 99, 64)
72      layers.Conv2D(128, (3, 3), activation='relu'),  # (18, 97, 128)
73      layers.MaxPooling2D((2, 2)),  # (9, 48, 128)
74      layers.Conv2D(256, (3, 3), activation='relu'),  # (7, 46, 256)
75      layers.MaxPooling2D((2, 2)),  # (3, 22, 256)
76      layers.Flatten(),  # (3 * 22 * 256)
77      layers.Dense(1024, activation='relu'),
78      layers.Dropout(0.4),
79      layers.Dense(6 * num_classes, activation='softmax'),
80      layers.Reshape((6, num_classes))
81  ])
82
83  # Modell kompilieren
84  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

- **Layers.Input** defines the input for the model
- **Layers.Conv2D** creates a 2D convolutional layer with the following parameters:
  - **32** output filters
  - **(3, 3)** size of the convolutional filter
  - **Activation = ‘relu’** as the activation function applied (this transforms all negative values to 0 which is necessary for the model to understand non-linearity and has performance benefits)

The code

```
63  # Modellarchitektur
64  input_shape = (88, 404, 1)  # Graustufen-Bild
65
66  model = models.Sequential([
67      layers.Input(shape=input_shape),  # (88, 404, 1)
68      layers.Conv2D(32, (3, 3), activation='relu'),  # (86, 402, 32)
69      layers.MaxPooling2D((2, 2)),  # (43, 201, 32)
70      layers.Conv2D(64, (3, 3), activation='relu'),  # (41, 199, 64)
71      layers.MaxPooling2D((2, 2)),  # (20, 99, 64)
72      layers.Conv2D(128, (3, 3), activation='relu'),  # (18, 97, 128)
73      layers.MaxPooling2D((2, 2)),  # (9, 48, 128)
74      layers.Conv2D(256, (3, 3), activation='relu'),  # (7, 46, 256)
75      layers.MaxPooling2D((2, 2)),  # (3, 22, 256)
76      layers.Flatten(),  # (3 * 22 * 256)
77      layers.Dense(1024, activation='relu'),
78      layers.Dropout(0.4),
79      layers.Dense(6 * num_classes, activation='softmax'),
80      layers.Reshape((6, num_classes))
81  ])
82
83  # Modell kompilieren
84  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

- **Layers.MaxPooling2D** operates maxpooling with the following parameters:
  - **(2, 2)** as the size of the pooling window
- The output shape has height **43** (86/2), width **201** (402/2) and **32** output filters (unchanged)

The code

```
63  # Modellarchitektur
64  input_shape = (88, 404, 1)  # Graustufen-Bild
65
66  model = models.Sequential([
67      layers.Input(shape=input_shape),  # (88, 404, 1)
68      layers.Conv2D(32, (3, 3), activation='relu'),  # (86, 402, 32)
69      layers.MaxPooling2D((2, 2)),  # (43, 201, 32)
70      layers.Conv2D(64, (3, 3), activation='relu'),  # (41, 199, 64)
71      layers.MaxPooling2D((2, 2)),  # (20, 99, 64)
72      layers.Conv2D(128, (3, 3), activation='relu'),  # (18, 97, 128)
73      layers.MaxPooling2D((2, 2)),  # (9, 48, 128)
74      layers.Conv2D(256, (3, 3), activation='relu'),  # (7, 46, 256)
75      layers.MaxPooling2D((2, 2)),  # (3, 22, 256)
76      layers.Flatten(),  # (3 * 22 * 256)
77      layers.Dense(1024, activation='relu'),
78      layers.Dropout(0.4),
79      layers.Dense(6 * num_classes, activation='softmax'),
80      layers.Reshape((6, num_classes))
81  ])
82
83  # Modell kompilieren
84  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

- **Layers.Flatten** converts the 3D output of the convolutional layers into a 1D vector of size **3 * 22 * 256** = 16,869
- **Layers.Dense** inputs the flattened data into **1024** functions where the combination of all functions outputs the probabilities for the image to contain a certain character at one of the six positions (again activation = **‘relu’** is used to eliminate negative numbers)

The code

```
63  # Modellarchitektur
64  input_shape = (88, 404, 1)  # Graustufen-Bild
65
66  model = models.Sequential([
67      layers.Input(shape=input_shape),  # (88, 404, 1)
68      layers.Conv2D(32, (3, 3), activation='relu'),  # (86, 402, 32)
69      layers.MaxPooling2D((2, 2)),  # (43, 201, 32)
70      layers.Conv2D(64, (3, 3), activation='relu'),  # (41, 199, 64)
71      layers.MaxPooling2D((2, 2)),  # (20, 99, 64)
72      layers.Conv2D(128, (3, 3), activation='relu'),  # (18, 97, 128)
73      layers.MaxPooling2D((2, 2)),  # (9, 48, 128)
74      layers.Conv2D(256, (3, 3), activation='relu'),  # (7, 46, 256)
75      layers.MaxPooling2D((2, 2)),  # (3, 22, 256)
76      layers.Flatten(),  # (3 * 22 * 256)
77      layers.Dense(1024, activation='relu'),
78      layers.Dropout(0.4),
79      layers.Dense(6 * num_classes, activation='softmax'),
80      layers.Reshape((6, num_classes))
81  ])
82
83  # Modell kompilieren
84  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

- Layers.Dropout randomly turns off **40%** of the neurons in our network during training
  - This is used to prevent overfitting (the model becoming too confident in certain patterns and relying only on few neurons)
- **Layers.Dense** is then used with **softmax** to convert the output variables to relative probabilities
- **Layers.reshape** creates a matrix of the size (number of characters in CAPTCHA, number of characters possible)

The code

```
63  # Modellarchitektur
64  input_shape = (88, 404, 1)  # Graustufen-Bild
65
66  model = models.Sequential([
67      layers.Input(shape=input_shape),  # (88, 404, 1)
68      layers.Conv2D(32, (3, 3), activation='relu'),  # (86, 402, 32)
69      layers.MaxPooling2D((2, 2)),  # (43, 201, 32)
70      layers.Conv2D(64, (3, 3), activation='relu'),  # (41, 199, 64)
71      layers.MaxPooling2D((2, 2)),  # (20, 99, 64)
72      layers.Conv2D(128, (3, 3), activation='relu'),  # (18, 97, 128)
73      layers.MaxPooling2D((2, 2)),  # (9, 48, 128)
74      layers.Conv2D(256, (3, 3), activation='relu'),  # (7, 46, 256)
75      layers.MaxPooling2D((2, 2)),  # (3, 22, 256)
76      layers.Flatten(),  # (3 * 22 * 256)
77      layers.Dense(1024, activation='relu'),
78      layers.Dropout(0.4),
79      layers.Dense(6 * num_classes, activation='softmax'),
80      layers.Reshape((6, num_classes))
81  ])
82
83  # Modell kompilieren
84  model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

- **Model.compile** determines how the model learns by specifying
  - The optimizer **adam** (Adaptive Moment Estimation) which combines momentum and adaptive learning rates
  - The loss function **categorical_crossentropy** which measures how far the model's predictions are from the true labels
  - The metrics **accuracy** that are extra measures to monitor performance during training and extracts the proportion of correct predictions

Best practices in layer structure

01 Progressive Feature Extraction

- (Early) layers should use small filters (e.g., 3x3, 5x5) to capture edges and simple features

02 Spatial Reduction

- Pooling to reduce resolution as depth increases
- Typical pattern: High resolution with few channels → lower resolution with more channels
- Rule of thumb: Double the number of channels each time you downsample (e.g., 64 → 128 → 256)

03 Activation

- Use ReLU activation for non-linearity

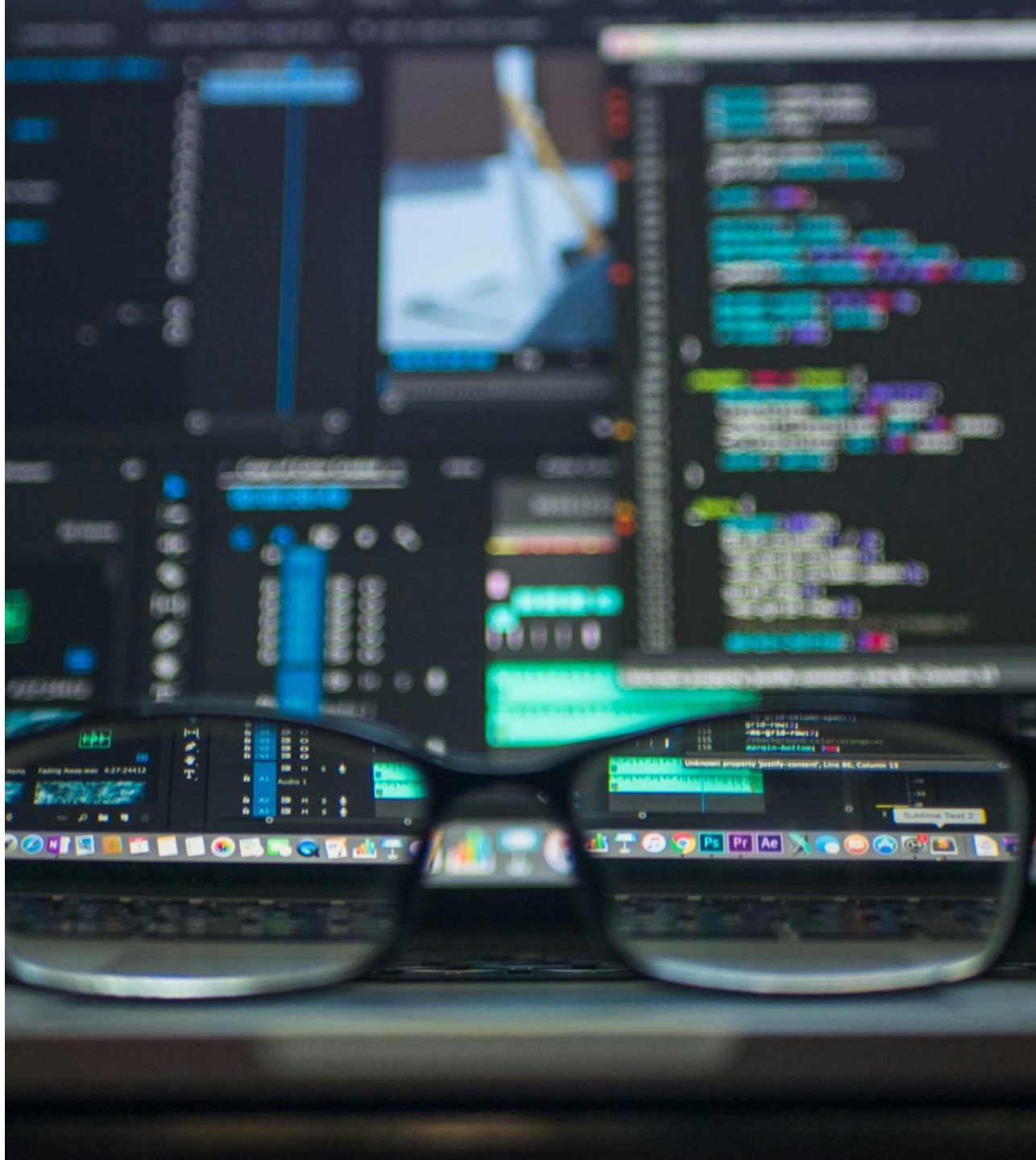
04 Regularization

- Use Dropout or other methods to prevent overfitting

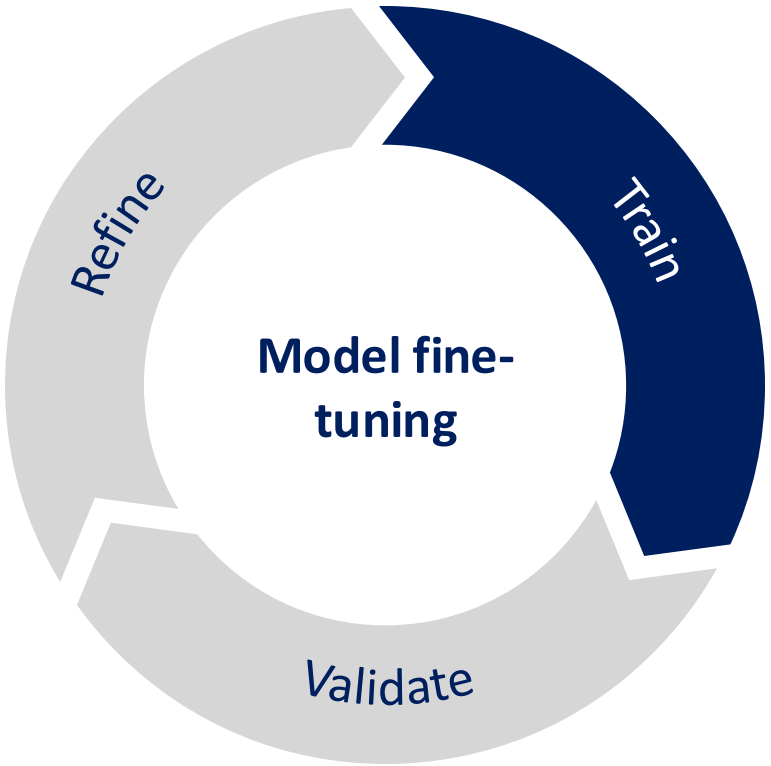
Talkin bout a convolution…

tinyurl.com/FolderCaptcha

5 Model fine-tuning



Model learns patterns from labeled CAPTCHA data and updates its parameters

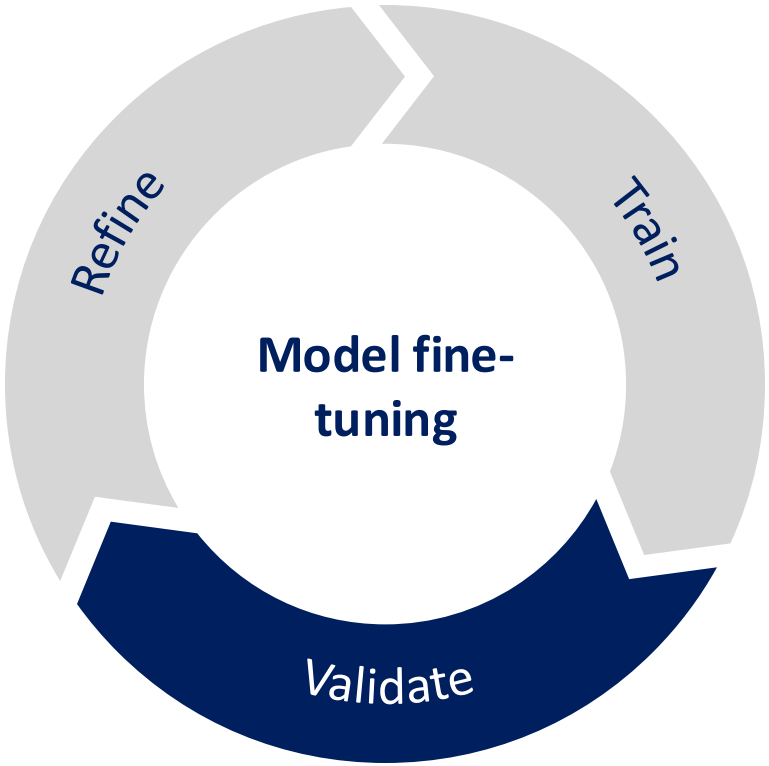


Training:

- **Learn from labeled CAPTCHA data** as training input
- **Update parameters** via optimizer using backpropagation
- **Iteratively improve model** accuracy across epochs

Sources: Peter Rossbach

Evaluate model performance on unseen data to detect overfitting



Training:

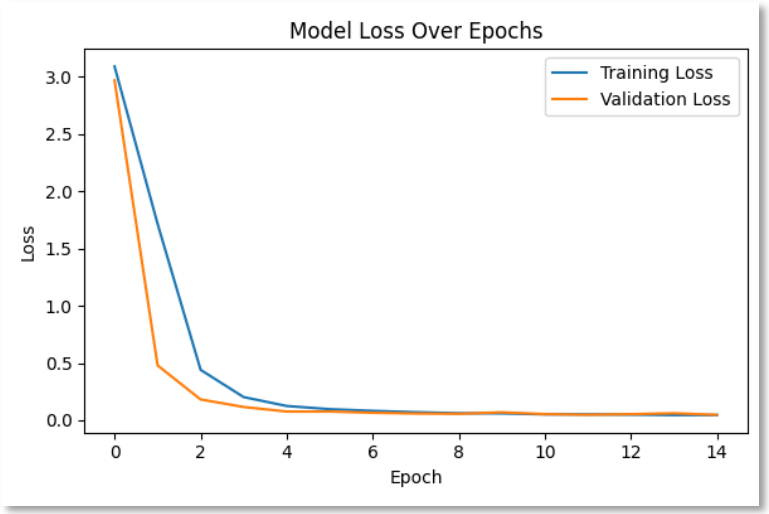
- **Learn from labeled CAPTCHA data** as training input
- **Update parameters** via optimizer using backpropagation
- **Iteratively improve model** accuracy across epochs

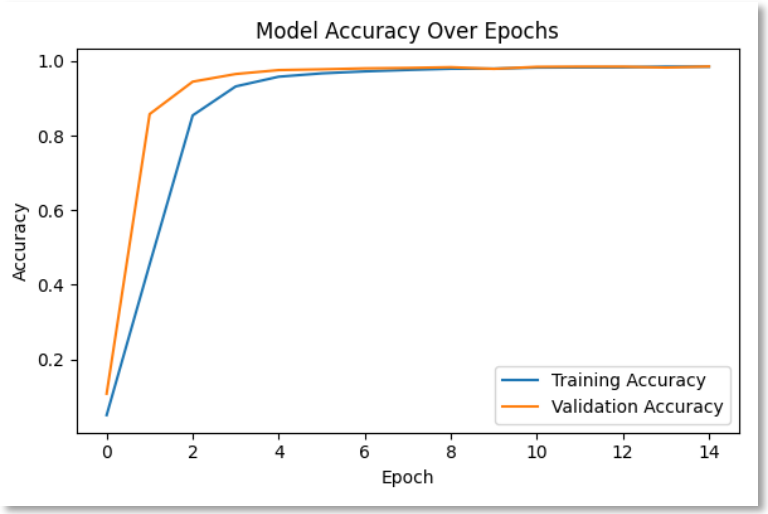
Validation:

- **Measure performance on unseen validation data** (accuracy)
- **Detect overfitting** by comparing training vs. validation curves
- **Guide refinement** by providing feedback for hyperparameter tuning

Sources: Peter Rossbach

Evaluate model performance on unseen data to detect overfitting





Training:

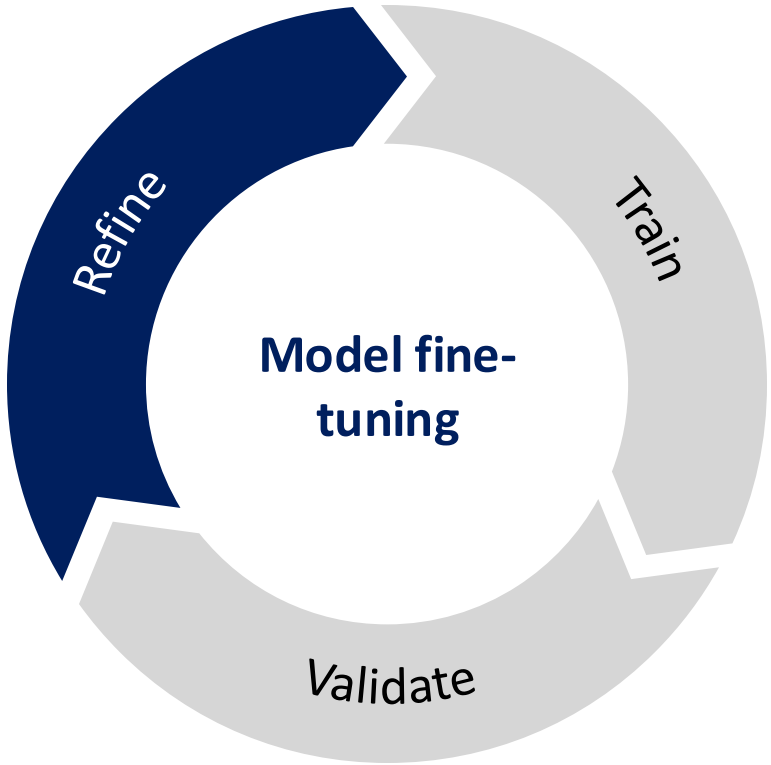
- **Learn from labeled CAPTCHA data** as training input
- **Update parameters** via optimizer using backpropagation
- **Iteratively improve model** accuracy across epochs

Validation:

- **Measure performance on unseen validation data** (accuracy)
- **Detect overfitting** by comparing training vs. validation curves
- **Guide refinement** by providing feedback for hyperparameter tuning

Sources: Peter Rossbach

Refining adjusts the model for better accuracy and generalization



Training:

- **Learn from labeled CAPTCHA data** as training input
- **Update parameters** via optimizer using backpropagation
- **Iteratively improve model** accuracy across epochs

Validation:

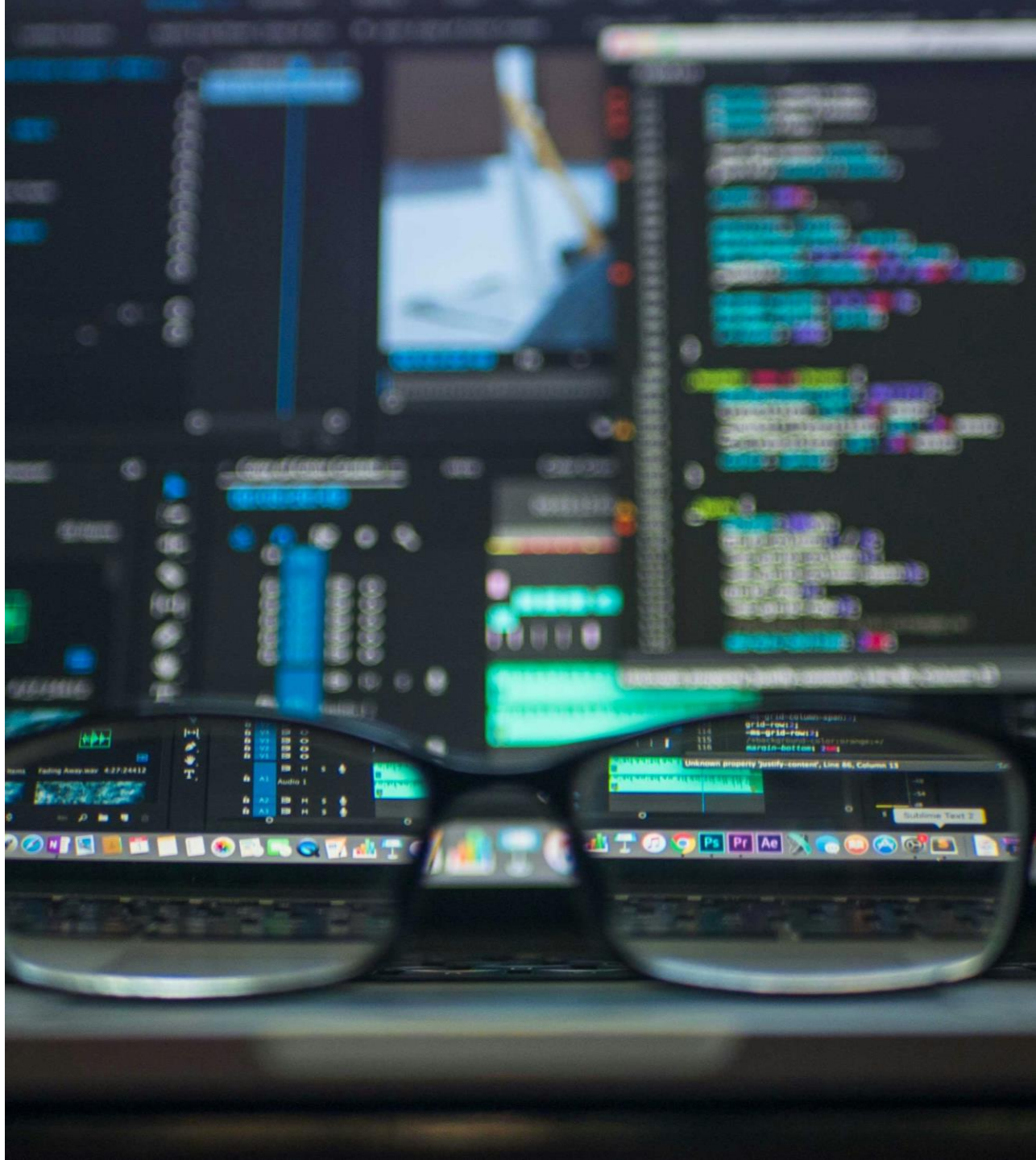
- **Measure performance on unseen validation data** (accuracy)
- **Detect overfitting** by comparing training vs. validation curves
- **Guide refinement** by providing feedback for hyperparameter tuning

Refining:

- **Adjust hyperparameters** (learning rate, batch size, etc.)
- **Optimize architecture** if needed (layers, filters, activation)
- **Iterate improvements** until stable, generalizable performance

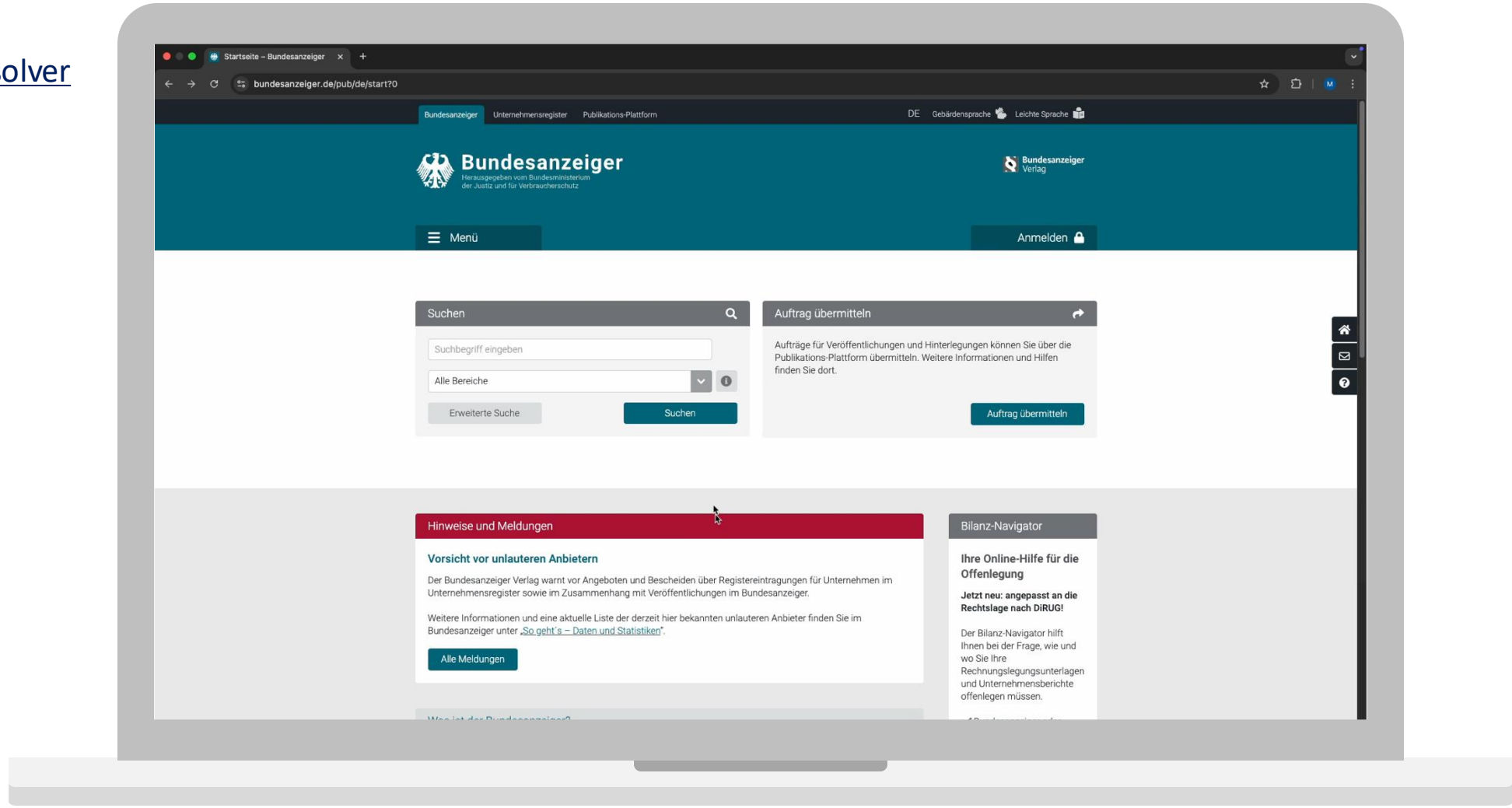
Sources: Peter Rossbach

6 Application

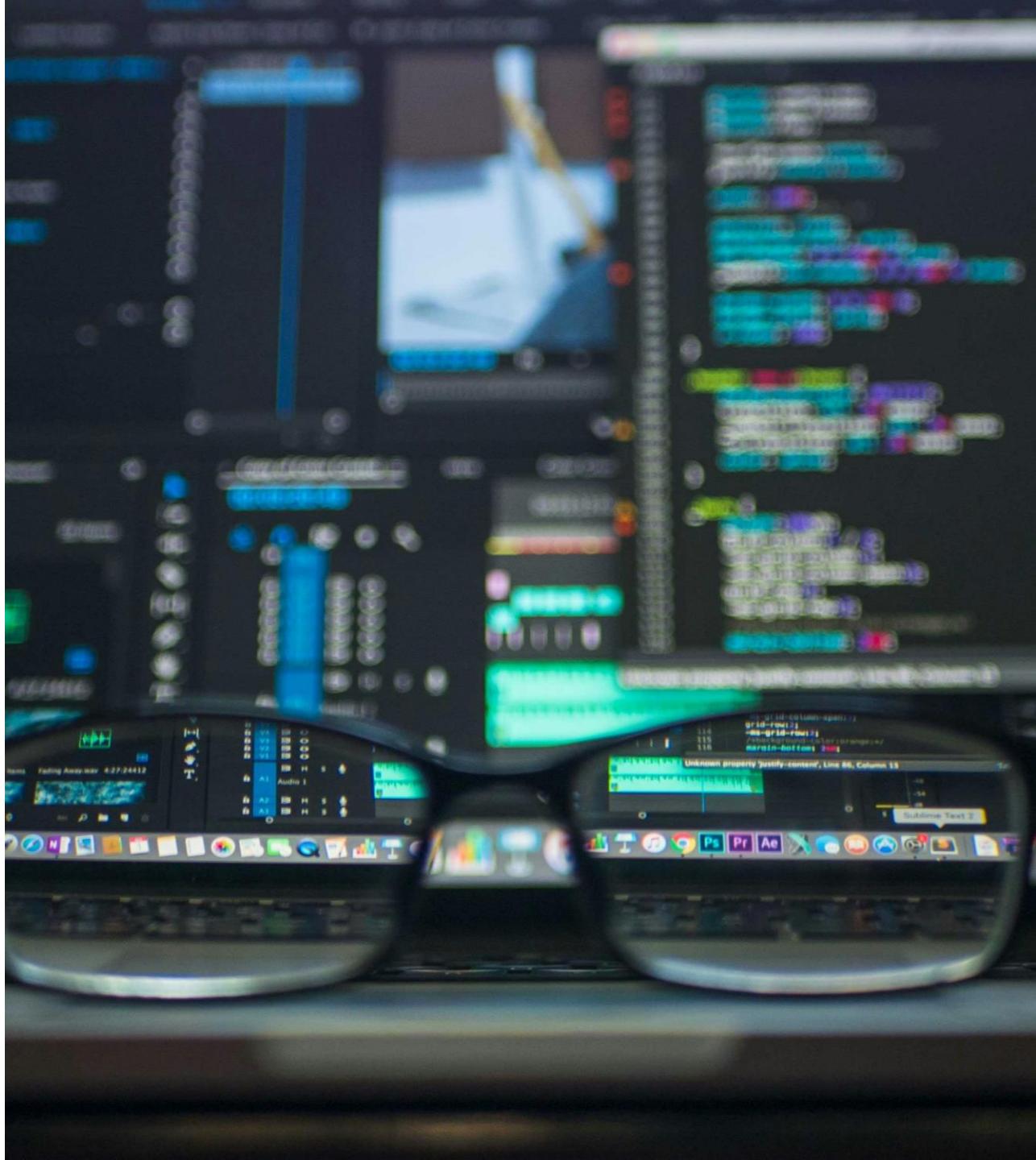


Compiling all this knowledge leads to this amazing Chrome Extension

tinyurl.com/FScaptchasolver



6 Conclusion



Conclusion

General idea of OCR
OCR converts images of text into machine-readable characters by detecting and classifying specific features/ patterns.

Convolution & Max Pooling
CNNs use filters and data pooling to recognize patterns and features in images.

Feed Forward Neural Network
weighs the different parameters and creates outputs.

Fine-tuning is an iterative process
Model fine-tuning cycles through training, validation, and refinement to improve accuracy and generalization.

Faster Bundesanzeiger research
Use OCR-extension to extract and make public filings searchable for quicker review and analysis.

Diverse use cases for further applications
Faster sign-up into accounts, buying tickets, or enable automations.

Contribution statement & Disclaimer

Every team member (Maximillian Klar, Moritz Landwehr, Tristan Schwarz & Stefan Cames) has contributed equally to this presentation.

All content of this presentation is prepared for research purposes only. No commercial use intended.