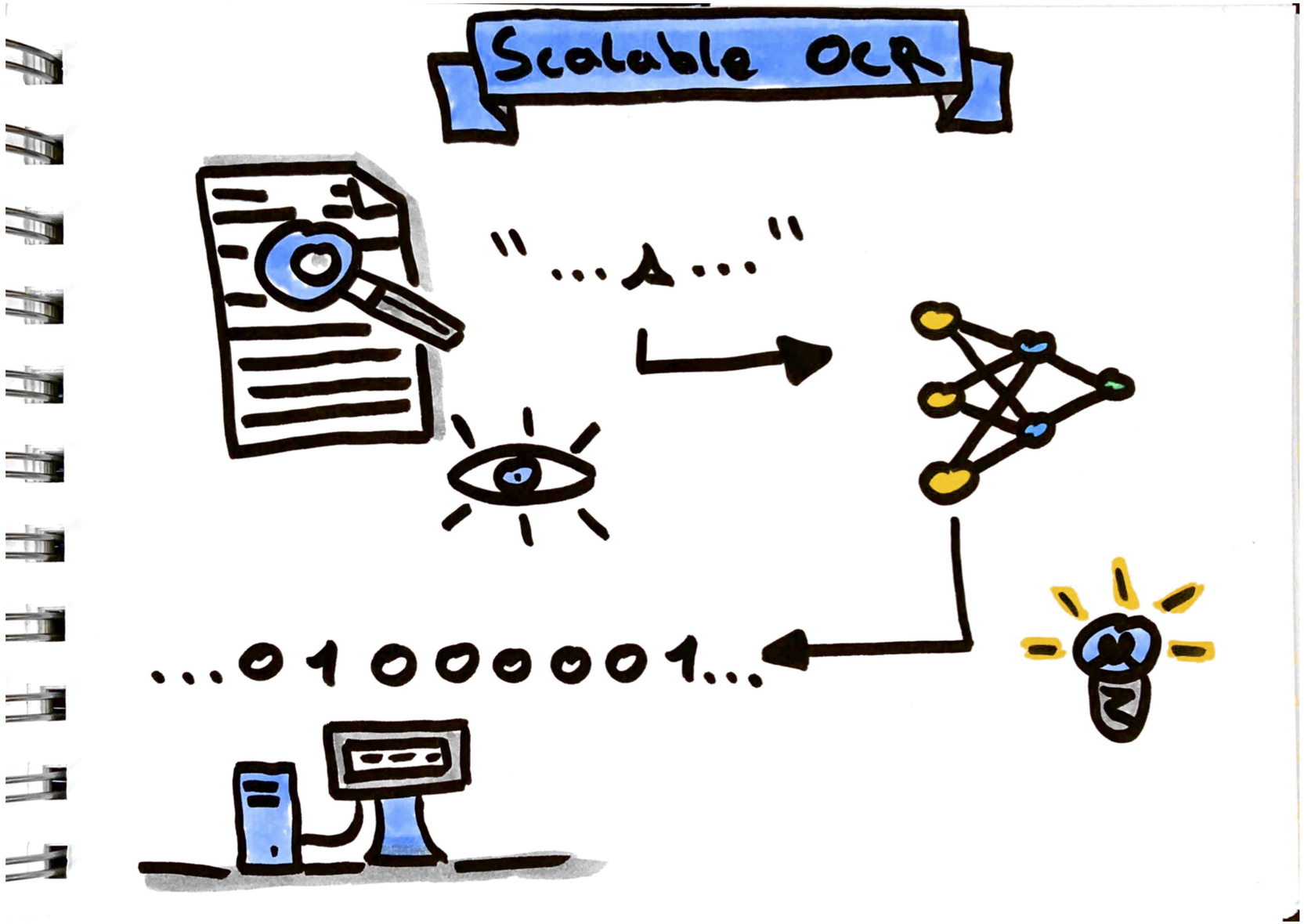
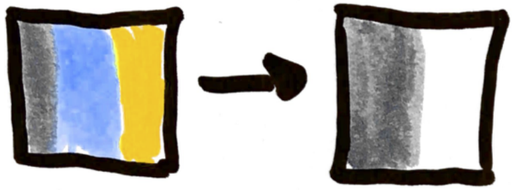

Scalable OCR
"...A..."
...01000001...

Text Rotation

1. Binarize



2. Find Lines

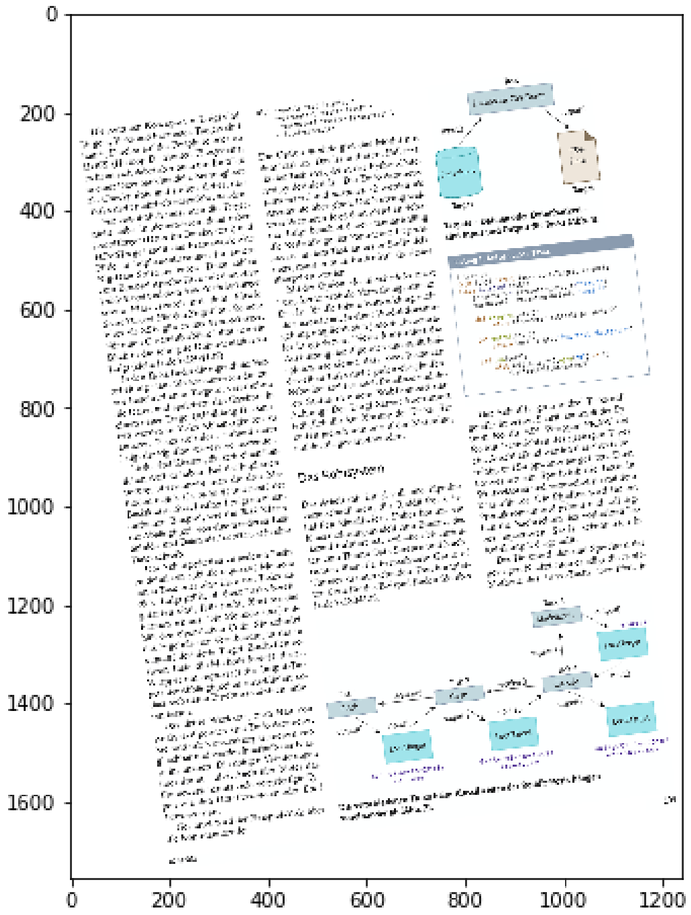
3. Calc. Avg. Angle

$\sum \alpha / \text{No. Lines}$

→ Rotate!

In [4]:
```
file = 'documents/skewed.tiff'
im = cv2.imread(file)
plt.figure(figsize=(8, 8))
plt.imshow(im)
```

Out[4]:
```
<matplotlib.image.AxesImage at 0x10b47d3d0>
```

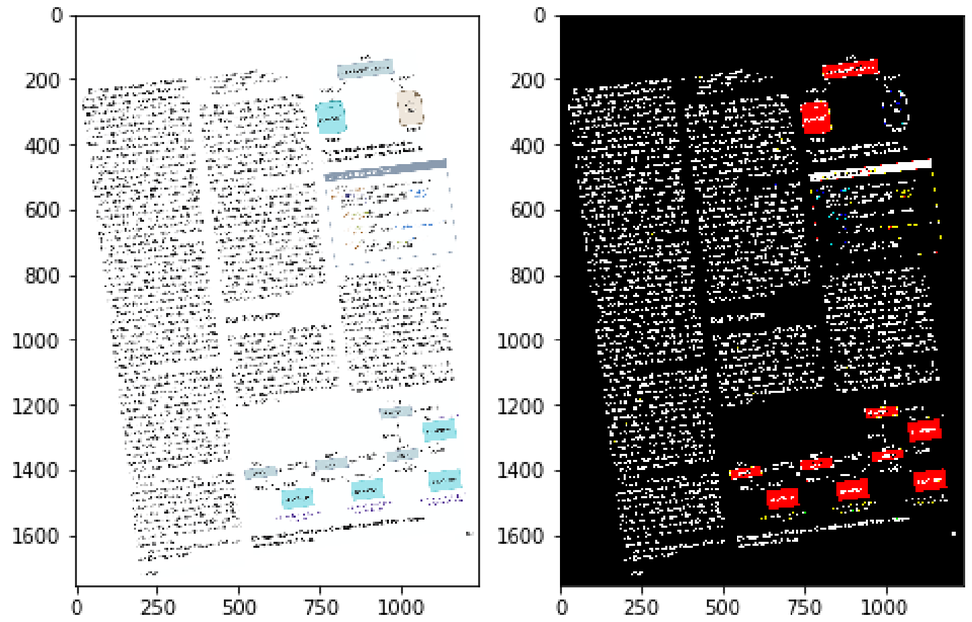


Thresholding

In [6]:
```
_, binary_thresh = cv2.threshold(im, 200, 255, cv2.THRESH_BINARY_INV)

fig = plt.figure(figsize=(8, 8))
fig.add_subplot(1, 2, 1)
plt.imshow(im)
fig.add_subplot(1, 2, 2)
plt.imshow(binary_thresh)
```

Out[6]:
```
<matplotlib.image.AxesImage at 0x10b8043d0>
```



Detect lines

In [8]:
```
lines = cv2.HoughLinesP(binary_thresh, 1, numpy.pi/180, 100, minLineLength= 600/2.
0, maxLineGap=20)

angle = 0
for line in lines:
    x1, y1, x2, y2 = line[0]
    r = numpy.arctan2(y2 - y1, x2 - x1)
    angle += numpy.arctan2(y2 - y1, x2 - x1)

avg_radian = angle / len(lines)
avg_angle = avg_radian * 180 / numpy.pi

print "Average angle is %f degrees" % avg_angle
```

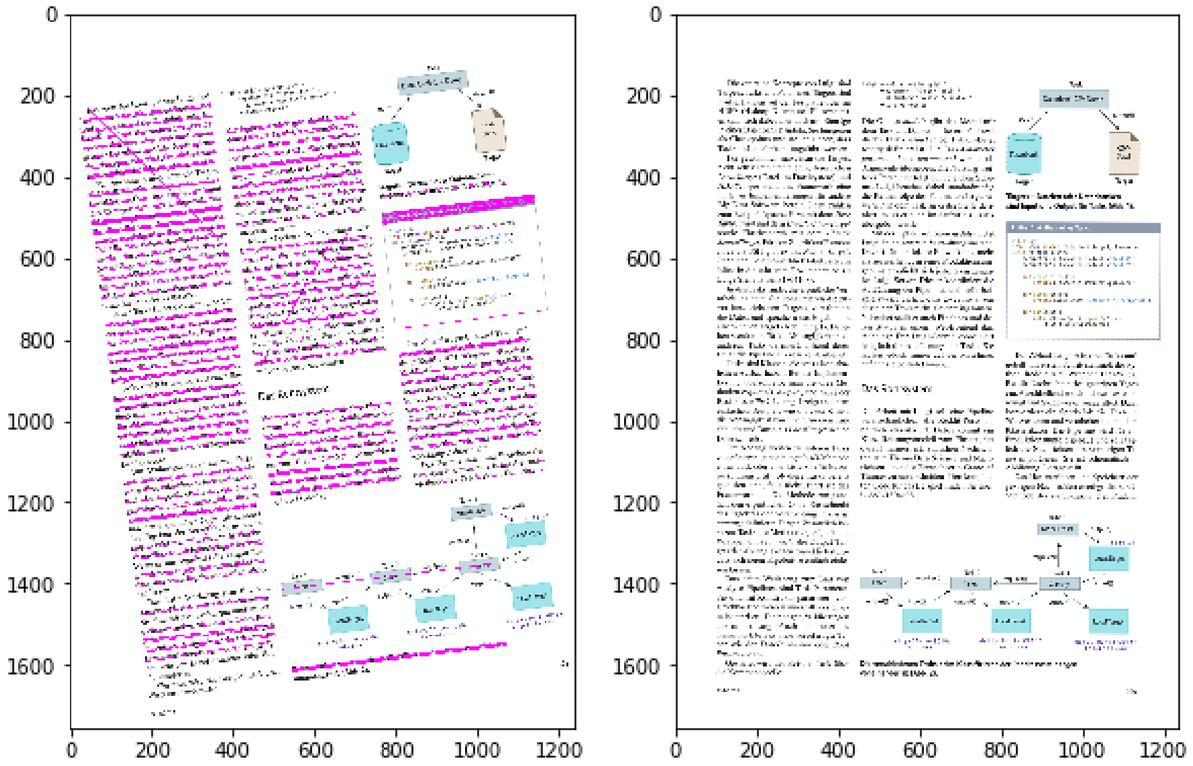
```
Average angle is -7.209056 degrees
```

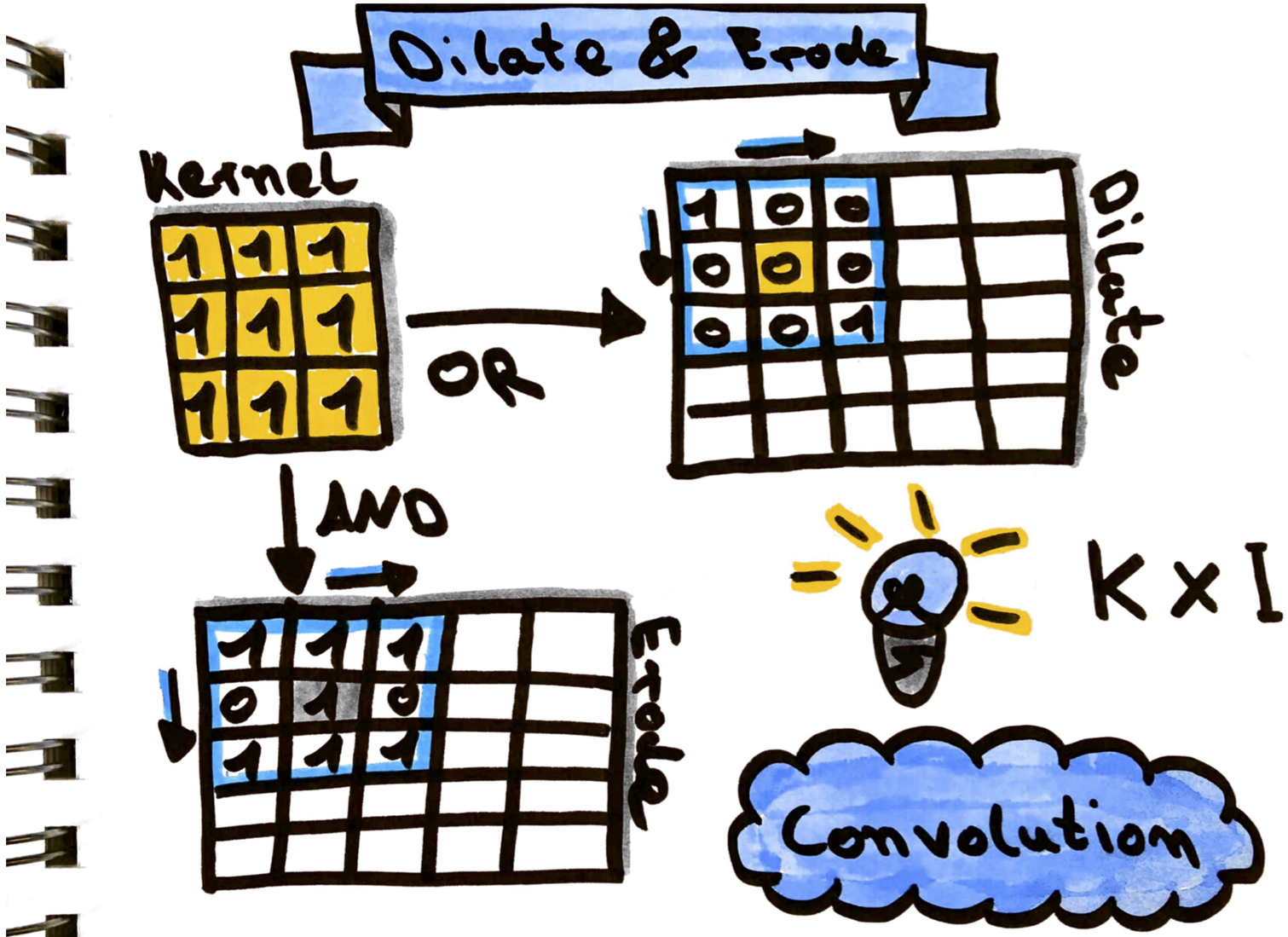
In [11]:
```
fig = plt.figure(figsize=(10, 10))

fig.add_subplot(1, 2, 1)
plt.imshow(vis)

fig.add_subplot(1, 2, 2)
plt.imshow(rotated)
```

Out[11]:
```
<matplotlib.image.AxesImage at 0x11296cb10>
```



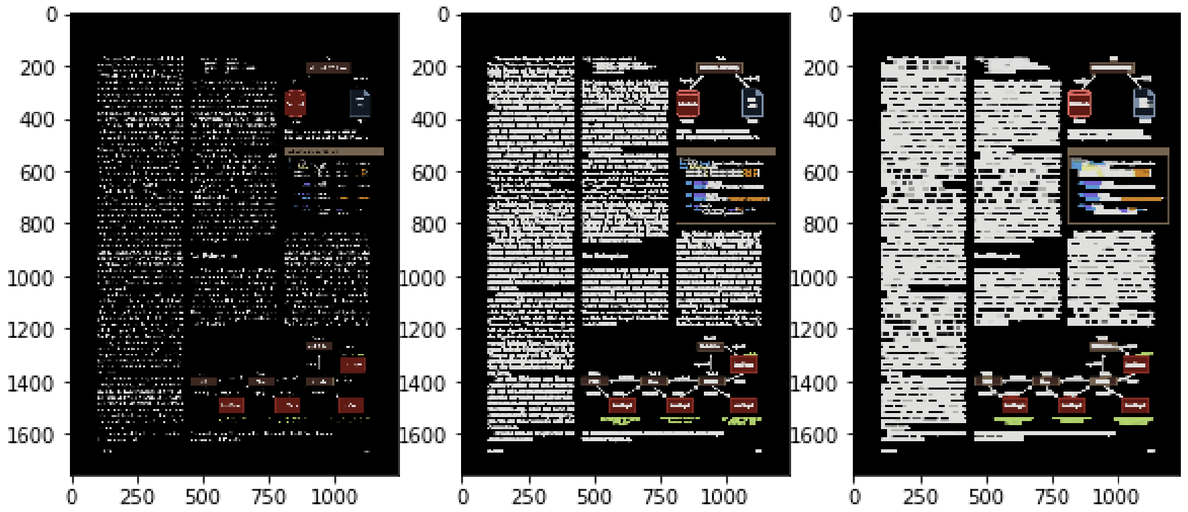

Dilate & Erode
Kernel
OR
Dilate
AND
Erode
K x I
Convolution

In [16]:
```
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
dilate = cv2.morphologyEx(neg, cv2.MORPH_DILATE, kernel)

kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (10, 5))
connected = cv2.morphologyEx(dilate, cv2.MORPH_CLOSE, kernel, iterations=2)

fig = plt.figure(figsize=(10, 10))
fig.add_subplot(1, 3, 1)
plt.imshow(neg)
fig.add_subplot(1, 3, 2)
plt.imshow(dilate)
fig.add_subplot(1, 3, 3)
plt.imshow(connected)
```

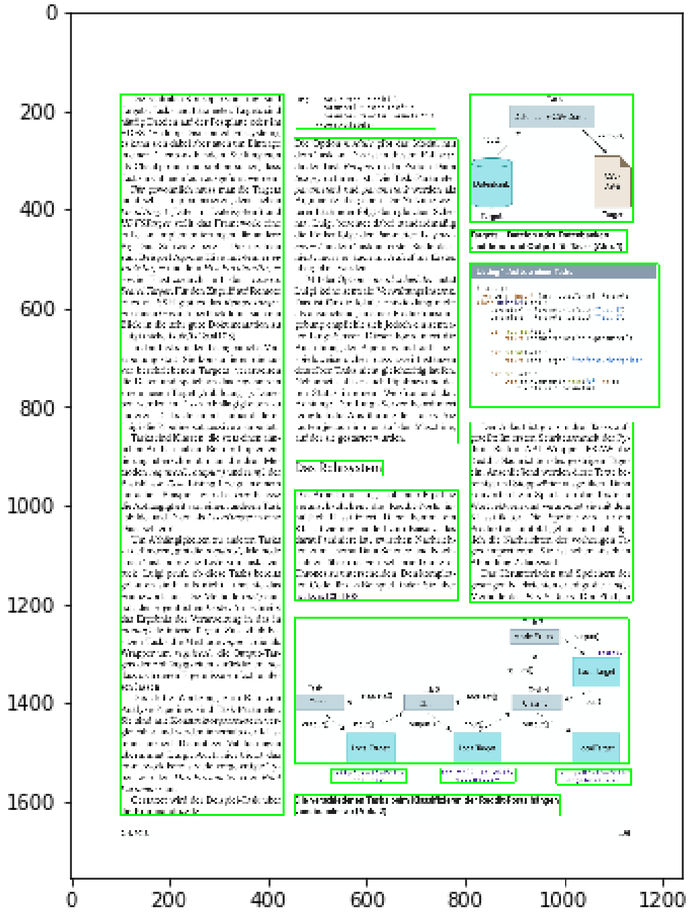
Out[16]:
```
<matplotlib.image.AxesImage at 0x116ce3810>
```

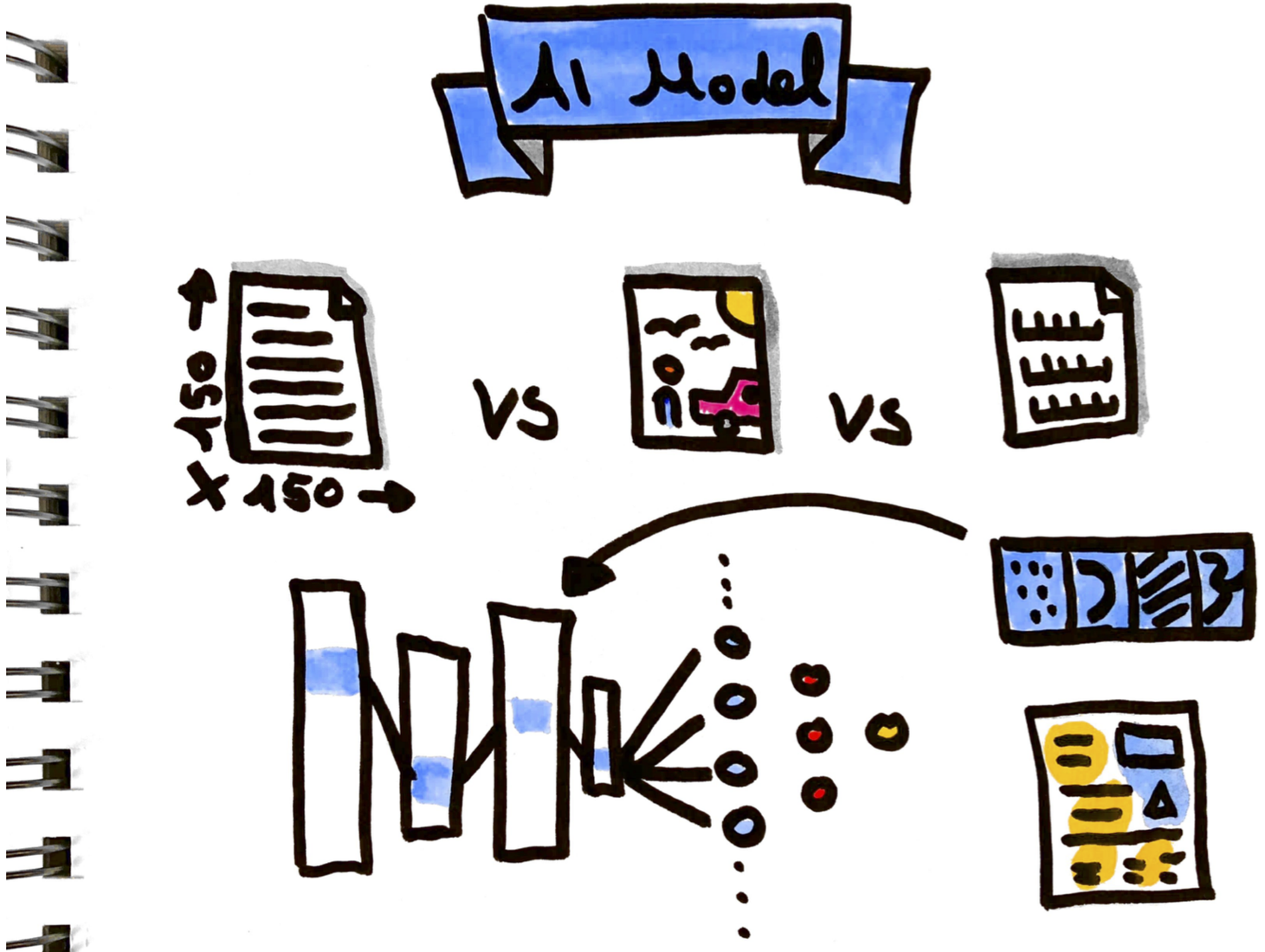


In [18]:
```
pic, contours, hierarchy = cv2.findContours(connected, cv2.RETR_EXTERNAL, cv2.CHAI
N_APPROX_SIMPLE)
boxes = map(lambda c: cv2.boundingRect(c), contours)
filtered = filter(lambda b: b[2] > 20 and b[3] > 25, boxes)
```

In [20]:
```
plt.figure(figsize=(8, 8))
plt.imshow(vis)
```

Out[20]:
```
<matplotlib.image.AxesImage at 0x118324150>
```



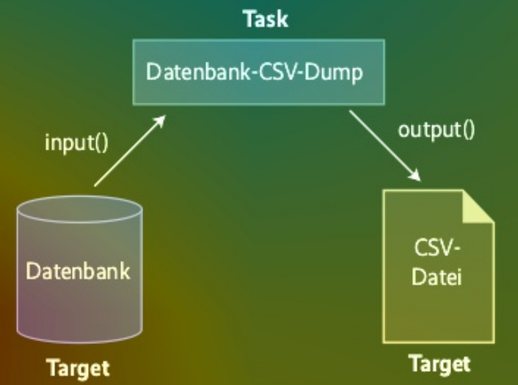

AI Model
150
X 150
VS
VS

Die zentralen Konzepte von Luigi sind Targets, Tasks und Parameter. Targets sind häufig Dateien auf der Festplatte oder im HDFS (Hadoop Distributed Filesystem), es kann sich dabei aber auch um Einträge in einer Datenbank handeln. Sie fungieren als Checkpoints und stellen sicher, dass Tasks nicht mehrfach ausgeführt werden.

Für gewöhnlich muss man die Targets nicht selbst implementieren, denn neben *LocalTarget* (Datei im Dateisystem) und *HDFSTarget* stellt das Framework eine Fülle an Implementierungen für andere Big-Data-Software bereit. Dazu zählen zum Beispiel Apache Hive mit dem *Hive*

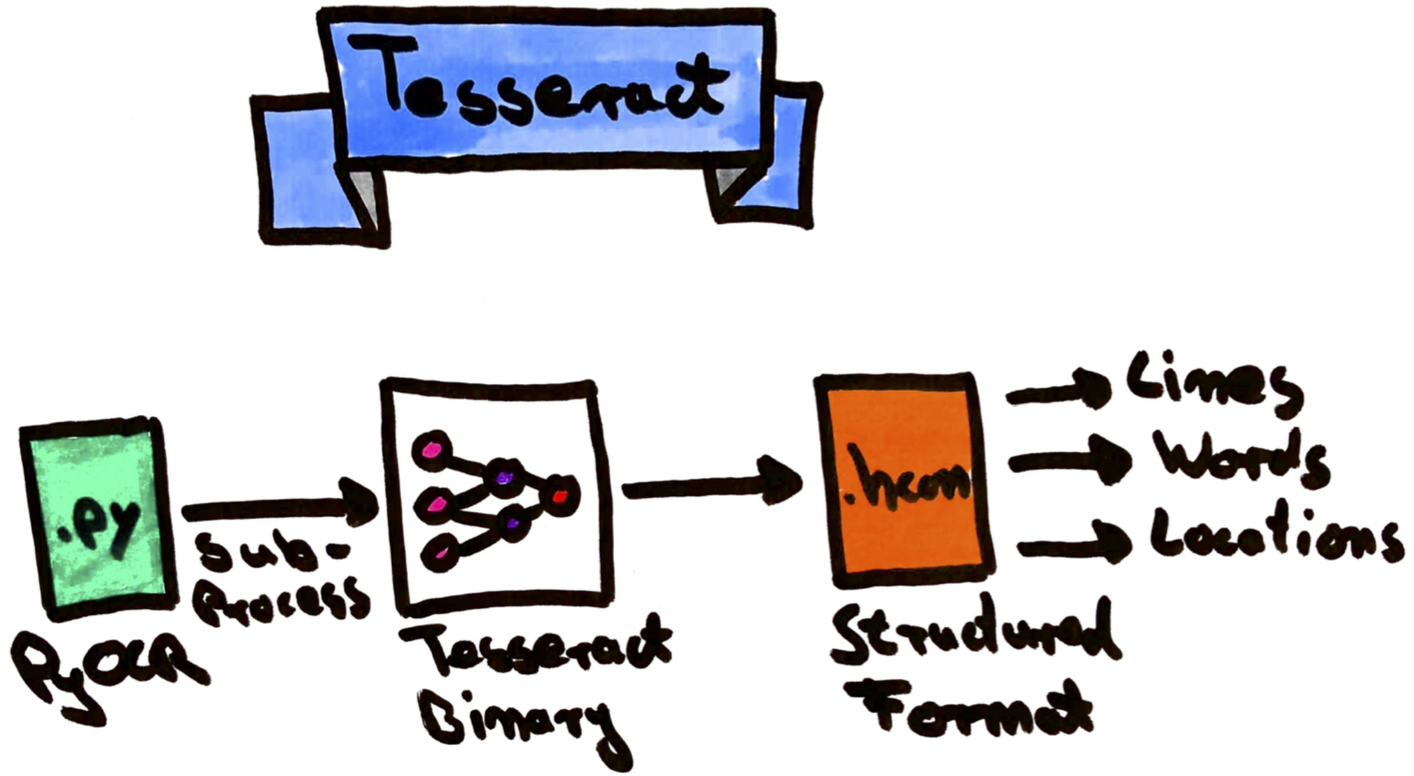
```
luigi --module test Beispiel \
      --parameter1 "Erster Test!" \
      --parameter2 "zweiter Parameter!" \
      --local-scheduler
```

Die Option *module* gibt das Modul mit dem Task an. Das ist in diesem Fall *test*, da der Task *Beispiel* in der Python-Datei *test.py* definiert ist. Die Task-Parameter *parameter1* und *parameter2* werden als Argumente übergeben. Die Nutzung weiterer Parameter folgt dem gleichen Schema. Luigi beachtet dabei standardmäßig die Reihenfolge der Parameter: Ist *parameter1* in dem Task an erster Stelle definiert, muss er auch im Aufruf als Erstes



Targets – Dateien oder Datenbanken – sind Input und Output für Tasks (Abb. 1).


RIVIERA
ENTRANCE

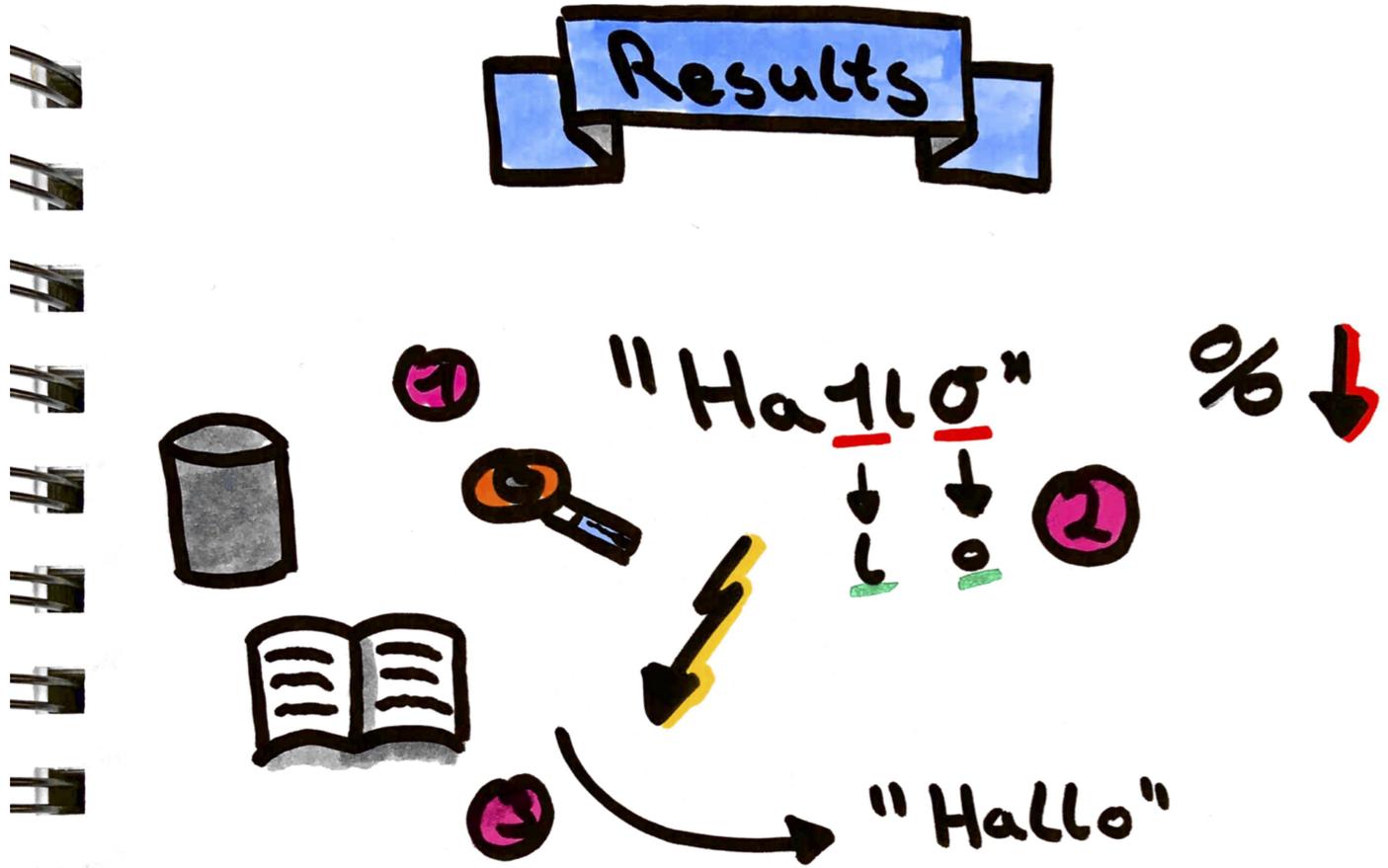

Tesseract
.py
Sub-
Process
.hocr
Lines
Words
Locations
PyOCR
Tesseract
Binary
Structured
Format

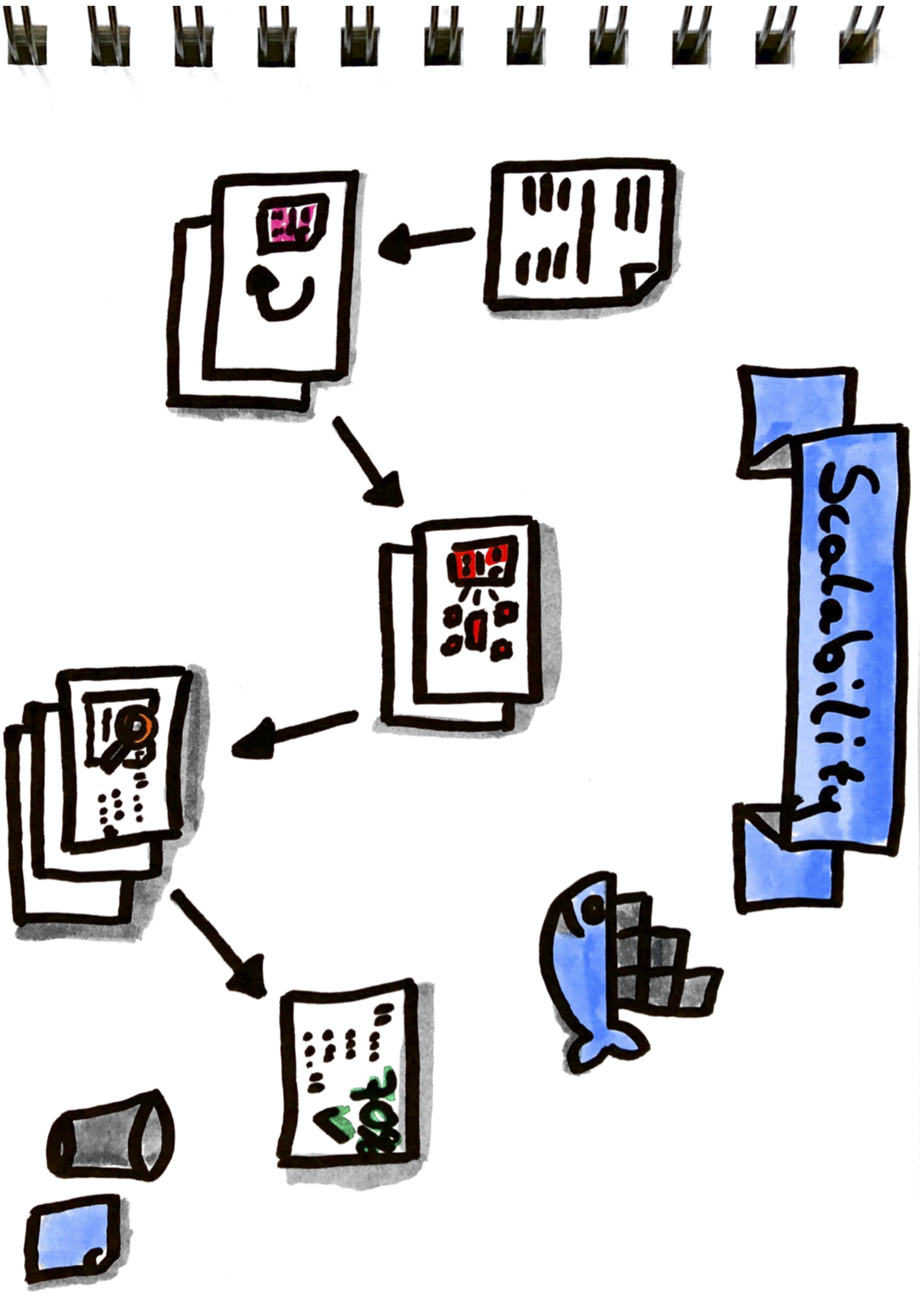
In [161]:
```
builder = pyocr.builders.LineBoxBuilder()
builder.tesseract_flags.extend(["--psm","11",
                                "--oem", "1",
                                "-c", "tessedit_create_hocr=1"])

im = Image.open(open("documents/dokument.tiff", 'rb')).convert('L')
lines = tool.image_to_string(im, lang=lang, builder=builder)

for l in lines[:5]:
    print l.content
```

```
Die zentralen Konzepte von Luigi sind
Targets, Tasks und Parameter. Targets sind
häufig Dateien auf der Festplatte oder im
HDFS (Hadoop Distributed Filesystem),
es kann sich dabei aber auch um Einträge
```


Results
"Hallo"
%
"Hallo"


Scalability

Contact

Twitter: @kein_mark

Mail: mark.keinhoerster@codecentric.de