Image Processing for Document Automation

Group2 - Assignments

in the context of “Image Processing for Document Automation (WS23/24)”
(University of Koblenz)

Submitted by:
Ritik Gupta (221100538)
Prajna Shetty (222100479)

1. Fill out the following overview table according to the state of your programming:

Exercise	Base-Exercise	Bonus	Measure
1 – Classification	Yes	No	Accuracy = 92.19
2 – Segmentation	Yes	No	Accuracy = 97.26
3 – De-skewing	Yes	Part	----------
4 – Cleaning	Yes	Yes	Accuracy = 96.43
5 – OCR	Yes	Part	----------
6 – End-to-end	Yes	No	Recall on correct words =0

TASK-1 ID Card Classification

1. What network structure did you choose and why?
   Convolutional Neural Network (CNN) architecture with 2 convolutional layers followed by 2 max-pooling layers. This architecture is a common choice for image classification tasks due to its ability to capture hierarchical features from images. Here's why this structure is chosen:
   - **Feature Extraction**: CNNs are well-suited for learning hierarchical representations of images by convolving filters across the input image and capturing local patterns in the initial layers, then gradually learning more abstract features in deeper layers.
   - **Max-Pooling**: Max-pooling layers help in reducing the spatial dimensions of the feature maps, thus decreasing the computational complexity and controlling overfitting by providing translation invariance.

2. What hyperparameters and loss function did you use, and why?

   - **Loss Function (Binary Crossentropy**): Binary crossentropy is a suitable choice for binary classification tasks, where the model predicts the probability of belonging to one of two classes. It measures the difference between the predicted probabilities and the actual binary labels, encouraging the model to produce high probabilities for the correct class and low probabilities for the incorrect class.

   - **Optimizer (RMSprop):** RMSprop is an adaptive learning rate optimization algorithm that adjusts the learning rate for each parameter based on the average of recent gradients for that parameter. It is often used in neural network training to adaptively adjust the learning rates, leading to faster convergence and better performance.

3. Report your training and test accuracies

Training:

```
Epoch 1/50
4/4 [==============================] - 58s 12s/step - loss: 0.2405 - accuracy: 0.9375
Epoch 2/50
4/4 [==============================] - 43s 11s/step - loss: 0.0930 - accuracy: 0.9844
Epoch 3/50
4/4 [==============================] - 46s 12s/step - loss: 0.0318 - accuracy: 1.0000
Epoch 4/50
4/4 [==============================] - 43s 10s/step - loss: 0.3282 - accuracy: 0.8750
Epoch 5/50
4/4 [==============================] - 41s 10s/step - loss: 11.7837 - accuracy: 0.6094
Epoch 6/50
4/4 [==============================] - 41s 11s/step - loss: 1.3713 - accuracy: 0.8125
Epoch 7/50
4/4 [==============================] - 42s 10s/step - loss: 0.3687 - accuracy: 0.8750
Epoch 8/50
4/4 [==============================] - 42s 10s/step - loss: 0.2501 - accuracy: 0.9219
```

Test:

```
13/13 [==============================] - 44s 3s/step
13/13 [==============================] - 43s 3s/step - loss: 0.2858 - accuracy: 0.9250
```

4. Provided the required data, what would you have to change in your model to classify not only the presence of German ID cards, but also ID Cards of other countries?

   To extend the model to classify ID cards of other countries, several changes may be necessary:

   - **Data Augmentation**: You may need to collect a dataset containing images of ID cards from different countries. Data augmentation techniques such as rotation, flipping, and scaling can be applied to increase the diversity of the dataset and improve the model's generalization ability.
   - **Model Adaptation**: Depending on the differences in the appearance and characteristics of ID cards from different countries, you may need to adjust the architecture of the CNN, such as adding more layers or modifying the filter sizes, to capture relevant features.
   - **Training Strategy**: The model may require retraining on the combined dataset containing ID cards from multiple countries. Depending on the size and complexity of the dataset, you may need to adjust the learning rate, batch size, and number of epochs to ensure optimal training performance.

5. Provide a (tensorflow model) summary of your network (layers, parameters, etc.)

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 510, 510, 64)      1792

 max_pooling2d (MaxPooling2  (None, 255, 255, 64)      0
 D)

 conv2d_1 (Conv2D)           (None, 253, 253, 32)      18464

 max_pooling2d_1 (MaxPoolin  (None, 126, 126, 32)      0
 g2D)

 flatten (Flatten)           (None, 508032)            0

 dense (Dense)               (None, 1)                 508033

=================================================================
Total params: 528289 (2.02 MB)
Trainable params: 528289 (2.02 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

TASK-2 ID Card Segmentation

1. What network structure did you choose and why?

   The convolutional neural network architecture U-Net is used. U-Net is commonly recommended for Image segmentation tasks, as it captures both low-level and high-level feature. And is suitable for segmentation of Objects with mostly clear boundaries, such as ID cards.

2. What hyperparameters and loss function did you use, and why?

   Learning rate = 0.001
   Optimizer = Adam, as it helps with faster convergence.
   Loss function = binary cross-entropy, as it is recommended for binary classification tasks.
   Early stopping , to prevent overfitting.

3. Report your training and test accuracies/losses

   Training:

```
Epoch 3/10
13/13 [==============================] - ETA: 0s - loss: 0.3142 - accuracy: 0.8973 WARNING:tensorflow:E
13/13 [==============================] - 139s 11s/step - loss: 0.3142 - accuracy: 0.8973
Epoch 4/10
13/13 [==============================] - ETA: 0s - loss: 0.2416 - accuracy: 0.8973 WARNING:tensorflow:E
13/13 [==============================] - 149s 11s/step - loss: 0.2416 - accuracy: 0.8973
Epoch 5/10
13/13 [==============================] - ETA: 0s - loss: 0.1397 - accuracy: 0.8973 WARNING:tensorflow:E
13/13 [==============================] - 140s 11s/step - loss: 0.1397 - accuracy: 0.8973
Epoch 6/10
13/13 [==============================] - ETA: 0s - loss: 0.1142 - accuracy: 0.9278 WARNING:tensorflow:E
13/13 [==============================] - 140s 11s/step - loss: 0.1142 - accuracy: 0.9278
Epoch 7/10
13/13 [==============================] - ETA: 0s - loss: 0.1046 - accuracy: 0.9618 WARNING:tensorflow:E
13/13 [==============================] - 140s 11s/step - loss: 0.1046 - accuracy: 0.9618
Epoch 8/10
13/13 [==============================] - ETA: 0s - loss: 0.1532 - accuracy: 0.9369 WARNING:tensorflow:E
13/13 [==============================] - 140s 11s/step - loss: 0.1532 - accuracy: 0.9369
Epoch 9/10
13/13 [==============================] - ETA: 0s - loss: 0.0941 - accuracy: 0.9597 WARNING:tensorflow:E
13/13 [==============================] - 140s 11s/step - loss: 0.0941 - accuracy: 0.9597
Epoch 10/10
13/13 [==============================] - ETA: 0s - loss: 0.0715 - accuracy: 0.9726 WARNING:tensorflow:E
13/13 [==============================] - 140s 11s/step - loss: 0.0715 - accuracy: 0.9726
<keras.src.callbacks.History at 0x7a904b0ee380>
```

Test:

Accuracy= 94.23

Loss=0.561

4. Provided the required data, what would you have to change in your model to segment multiple ID cards per image (multiple solution strategies)

   One way to segment multiple ID cards in an image would be using a sliding window approach. In the Sliding window approach, you move a fixed-sized window across an Image. Images in each window are segmented independently and combined in the end to reconstruct the final Image.

5. Provide a (tensorflow model) summary of your network (layers, parameters, etc.)

```
Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
input_1 (InputLayer)        [(None, 512, 512, 3)]     0         []

conv2d (Conv2D)             (None, 512, 512, 32)      896       ['input_1[0][0]']

conv2d_1 (Conv2D)           (None, 512, 512, 32)      9248      ['conv2d[0][0]']

max_pooling2d (MaxPooling2  (None, 256, 256, 32)      0         ['conv2d_1[0][0]']
D)

conv2d_2 (Conv2D)           (None, 256, 256, 64)      18496     ['max_pooling2d[0][0]']

conv2d_3 (Conv2D)           (None, 256, 256, 64)      36928     ['conv2d_2[0][0]']

max_pooling2d_1 (MaxPoolin  (None, 128, 128, 64)      0         ['conv2d_3[0][0]']
g2D)

conv2d_4 (Conv2D)           (None, 128, 128, 128)     73856     ['max_pooling2d_1[0][0]']

conv2d_5 (Conv2D)           (None, 128, 128, 128)     147584    ['conv2d_4[0][0]']

max_pooling2d_2 (MaxPoolin  (None, 64, 64, 128)       0         ['conv2d_5[0][0]']
g2D)

conv2d_6 (Conv2D)           (None, 64, 64, 256)       295168    ['max_pooling2d_2[0][0]']

conv2d_7 (Conv2D)           (None, 64, 64, 256)       590080    ['conv2d_6[0][0]']

conv2d_transpose (Conv2DTr  (None, 128, 128, 128)     131200    ['conv2d_7[0][0]']
anspose)

concatenate (Concatenate)   (None, 128, 128, 256)     0         ['conv2d_transpose[0][0]',
                                                                 'conv2d_5[0][0]']

conv2d_8 (Conv2D)           (None, 128, 128, 128)     295040    ['concatenate[0][0]']

conv2d_9 (Conv2D)           (None, 128, 128, 128)     147584    ['conv2d_8[0][0]']

conv2d_transpose_1 (Conv2D  (None, 256, 256, 64)      32832     ['conv2d_9[0][0]']
Transpose)

concatenate_1 (Concatenate  (None, 256, 256, 128)     0         ['conv2d_transpose_1[0][0]',
)                                                                'conv2d_3[0][0]']

conv2d_10 (Conv2D)          (None, 256, 256, 64)      73792     ['concatenate_1[0][0]']

conv2d_11 (Conv2D)          (None, 256, 256, 64)      36928     ['conv2d_10[0][0]']

conv2d_transpose_2 (Conv2D  (None, 512, 512, 32)      8224      ['conv2d_11[0][0]']
Transpose)

concatenate_2 (Concatenate  (None, 512, 512, 64)      0         ['conv2d_transpose_2[0][0]',
)                                                                'conv2d_1[0][0]']

conv2d_12 (Conv2D)          (None, 512, 512, 32)      18464     ['concatenate_2[0][0]']

conv2d_13 (Conv2D)          (None, 512, 512, 32)      9248      ['conv2d_12[0][0]']

conv2d_14 (Conv2D)          (None, 512, 512, 1)       33        ['conv2d_13[0][0]']

==================================================================================================
Total params: 1925601 (7.35 MB)
Trainable params: 1925601 (7.35 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

TASK-3 De-skewing

1. How did you implement your method?

   (cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY)). Then, it applies morphological operations to enhance the edges of the document. It first dilates the grayscale image using a kernel of size (25, 25) (cv2.dilate(gray, kernel, iterations=2)) and then erodes it using the same kernel size (cv2.erode(dilation, kernel, iterations=1)). Next, it performs an element-wise multiplication between the eroded image and the original grayscale image (new = erosion * gray). Afterward, it sets all pixels with value 0 to 255 (new[new == 0] = 255). This step seems to be aimed at handling regions with no information, possibly the background. Following this, the Canny edge detector (cv2.Canny(erosion, 50, 150, apertureSize=3)) is used to detect edges in the eroded image. The Hough Line Transform (cv2.HoughLines()) is then applied to detect lines in the edge-detected image. The angles of the detected lines are extracted, and a majority voting approach is used to determine the most common lines and angles. Finally, the image is rotated to deskew it is using the calculated angle (cv2.getRotationMatrix2D() and cv2.warpAffine()).

2. What are advantages/disadvantages you notice about your method?

   Advantages:

   The method is based on simple image processing techniques, making it computationally efficient.

   It does not rely on complex feature extraction methods like SIFT, which might require additional computational resources or might not be suitable for all types of images.

   Disadvantages:

   The method might not work well in cases where the document edges are not well-defined or when there are significant distortions or perspective effects.

It assumes that the document is the dominant feature in the image and might fail if there are other prominent lines or edges in the scene.
It might not handle rotation angles close to 90 or 180 degrees well, as the majority voting approach might not provide accurate results in such cases.

3. Are there any skew angles that are (un)favourable for your method?

   The method relies on the Hough Line Transform and a majority voting approach to determine the skew angle. It should work reasonably well for skew angles that are not too close to 90 or 180 degrees. However, angles close to these values might pose challenges, as the majority voting approach might not accurately identify the correct angle in such cases.

4. How would you implement upside-down detection without SIFT?

   Without using SIFT, an alternative approach to detect the orientation of text or documents in an image could involve using other feature extraction methods such as Harris corner detection or gradient-based methods. These methods can help identify key points or regions in the image that are likely to correspond to corners or edges of the document. Once these key points are detected, the orientation of the document can be estimated based on the distribution of these points.

5. What are options to segment and de-skew ID cards in one step?

   Options to segment and deskew ID cards in one step could involve using deep learning-based object detection and recognition models. These models can be trained to detect and localize ID cards in an image and then apply image processing techniques to segment the detected ID cards and deskew them. Alternatively, advanced image processing techniques such as contour detection and perspective transformation can be combined to first segment the ID card from the background and then deskew it to correct any perspective distortions.

Bonus:

1. Improve your method with image pre-processing
   To improve the method with image preprocessing, we have included additional steps such as smoothing, thresholding, and morphological operations to enhance the quality of the image and improve the effectiveness of the dilation and erosion operations.

   Here's an improved version of the method with image preprocessing steps:

   - **Convert to Grayscale**: Convert the input image to grayscale to simplify processing.
   - **Gaussian Blurring:** Apply Gaussian blur to smooth out noise and small variations in intensity.
   - **Morphological Operations** (Dilation and Erosion): Use dilation to expand the foreground regions and fill in gaps, followed by erosion to remove noise and fine-tune the shapes.

TASK-4 Cleaning

1. What network structure did you choose and why?
   The network structure we chose is U-Net. Because U-Nets are effective in differentiating between foreground text such as information on the ID card and background noise.

2. What hyperparameters and loss function did you use, and why?
   Optimizer = Adam, as it helps in faster convergence.
   Loss function = binary cross-entropy, as it is recommended for binary classification tasks.

3. Report your training and test accuracies
   Training:

```
Epoch 1/10
5/5 [==============================] - 575s 105s/step - loss: 0.4237 - accuracy: 0.9645
Epoch 2/10
5/5 [==============================] - 524s 101s/step - loss: 0.1549 - accuracy: 0.9611
Epoch 3/10
5/5 [==============================] - 477s 99s/step - loss: 0.1173 - accuracy: 0.9608
Epoch 4/10
5/5 [==============================] - 594s 129s/step - loss: 0.1107 - accuracy: 0.9625
Epoch 5/10
5/5 [==============================] - 754s 155s/step - loss: 0.1078 - accuracy: 0.9607
Epoch 6/10
5/5 [==============================] - 472s 82s/step - loss: 0.1005 - accuracy: 0.9605
Epoch 7/10
5/5 [==============================] - 700s 155s/step - loss: 0.0905 - accuracy: 0.9624
Epoch 8/10
5/5 [==============================] - 697s 140s/step - loss: 0.0800 - accuracy: 0.9593
Epoch 9/10
5/5 [==============================] - 464s 94s/step - loss: 0.0625 - accuracy: 0.9673
Epoch 10/10
5/5 [==============================] - 549s 108s/step - loss: 0.0561 - accuracy: 0.9695
```

   Test:

   Accuracy= 0.9243

   Loss= 0.0681

4. What is a fundamental issue with this approach that some text has to be removed, while other text is to be preserved?

   The fundamental issue of this approach that we faced is that it treats all content equally. It doesn't distinguish between information that should be removed and information that should be preserved. Manual intervention or other techniques can be used to get the expected results.

   a. What could be a reason as to why the approach does work anyway?

   This approach works anyways as it is designed to work well with segmentation like tasks. Resulting in most background/noise being removed.

   b. What would be an approach that is more “aligned” with how image based CNNs work?

   Approaches like multi-task learning could be more aligned. Providing contextual information into the model architecture can also be used.

5. Provide a (tensorflow model) summary of your network (layers, parameters, etc.)

```
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
input_1 (InputLayer)         [(None, 512, 512, 3)]     0         []

conv2d (Conv2D)              (None, 512, 512, 32)      896       ['input_1[0][0]']

conv2d_1 (Conv2D)            (None, 512, 512, 32)      9248      ['conv2d[0][0]']

max_pooling2d (MaxPooling2   (None, 256, 256, 32)      0         ['conv2d_1[0][0]']
D)

conv2d_2 (Conv2D)            (None, 256, 256, 64)      18496     ['max_pooling2d[0][0]']

conv2d_3 (Conv2D)            (None, 256, 256, 64)      36928     ['conv2d_2[0][0]']

max_pooling2d_1 (MaxPoolin   (None, 128, 128, 64)      0         ['conv2d_3[0][0]']
g2D)

conv2d_4 (Conv2D)            (None, 128, 128, 128)     73856     ['max_pooling2d_1[0][0]']

conv2d_5 (Conv2D)            (None, 128, 128, 128)     147584    ['conv2d_4[0][0]']

max_pooling2d_2 (MaxPoolin   (None, 64, 64, 128)       0         ['conv2d_5[0][0]']
g2D)

conv2d_6 (Conv2D)            (None, 64, 64, 256)       295168    ['max_pooling2d_2[0][0]']

conv2d_7 (Conv2D)            (None, 64, 64, 256)       590080    ['conv2d_6[0][0]']

conv2d_transpose (Conv2DTr   (None, 128, 128, 128)     131200    ['conv2d_7[0][0]']
anspose)

concatenate (Concatenate)    (None, 128, 128, 256)     0         ['conv2d_transpose[0][0]',
                                                                  'conv2d_5[0][0]']

conv2d_8 (Conv2D)            (None, 128, 128, 128)     295040    ['concatenate[0][0]']

conv2d_9 (Conv2D)            (None, 128, 128, 128)     147584    ['conv2d_8[0][0]']

conv2d_transpose_1 (Conv2D   (None, 256, 256, 64)      32832     ['conv2d_9[0][0]']
Transpose)

concatenate_1 (Concatenate   (None, 256, 256, 128)     0         ['conv2d_transpose_1[0][0]',
)                                                                 'conv2d_3[0][0]']
```

```
conv2d_10 (Conv2D)          (None, 256, 256, 64)         73792     ['concatenate_1[0][0]']

conv2d_11 (Conv2D)          (None, 256, 256, 64)         36928     ['conv2d_10[0][0]']

conv2d_transpose_2 (Conv2D  (None, 512, 512, 32)         8224      ['conv2d_11[0][0]']
Transpose)

concatenate_2 (Concatenate  (None, 512, 512, 64)         0         ['conv2d_transpose_2[0][0]',
)                                                                   'conv2d_1[0][0]']

conv2d_12 (Conv2D)          (None, 512, 512, 32)         18464     ['concatenate_2[0][0]']

conv2d_13 (Conv2D)          (None, 512, 512, 32)         9248      ['conv2d_12[0][0]']

conv2d_14 (Conv2D)          (None, 512, 512, 1)          33        ['conv2d_13[0][0]']

==================================================================================================
Total params: 1925601 (7.35 MB)
Trainable params: 1925601 (7.35 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

Bonus:

1. Compare your neural solution against a selection of classical models from the lecture.

   U-Net for Image Cleaning:

   - **Effectiveness**: Very effective for noise reduction and cleaning, especially in complex scenarios.
   - **Computational** Efficiency: Can be computationally intensive due to its deep architecture.
   - **Ease of Implementation**: Requires knowledge of deep learning frameworks.
   - **Robustness**: Robust across different types of noise and image variations.

   Morphological Filters:

   - **Effectiveness**: Effective for basic noise reduction but may struggle with complex patterns.
   - **Computational Efficiency**: Fast and computationally efficient.
   - **Ease of Implementation**: Easy to implement using libraries like OpenCV.
   - **Robustness**: Less robust to complex noise patterns compared to deep learning models.

   Comparison:

   - **Complexity of Noise**: U-Net performs better with complex noise patterns.
   - **Preservation of Details**: U-Net preserves fine details better.

- **Computational Overhead**: U-Net has higher computational overhead.
- **Generalization**: U-Net generalizes well across different scenarios, while morphological filters may require more manual adjustment.

2. What are advantages / downsides of neural and classical models?

   Neural Models:

   - **Advantages**: Highly effective due to their ability to learn complex patterns and relationships in data, adaptable to different data distributions.
   - **Downsides**: Computationally intensive, requiring large amounts of labeled data for training, and can be challenging to interpret.

   Classical Models:

   - **Advantages**: Computationally efficient, interpretable, and often simpler to implement.
   - **Downsides**: Limited expressiveness, requiring manual feature engineering and may struggle with complex data distributions.

TASK-5 OCR & Task-6 End-to-End

1. Report your recall on correct words w.r.t. the test set.
2. And Recall: #words in GT found / #words in GT

   We are encountering an issue with my performance on task 6. It seems we unable to correctly identify words after the "cleaning" process from task 4, which leads to vague results in the task 6 dataset. Interestingly, Model performed well on the task 4 test set the cleaning step

3. What did you have to change to plug individual components together?

To plug individual components together and create a pipeline, we need to ensure that each component is correctly loaded, integrated, and executed in sequence. Here's what you need to do:

- **Load Trained Models:** need to save and load each trained model separately. This typically involves saving the trained model weights and architecture to files and then loading them when needed. You can use frameworks like TensorFlow or PyTorch to save and load models.
- **Integration**: Once the models are loaded, integrate them into pipeline by defining functions for each component (e.g., classification, segmentation, deskewing). These functions should encapsulate the logic for each component and accept input data as arguments.
- **Execution**: Define a function or script to execute the pipeline. This function/script should call each component function in sequence, passing the output of one component as input to the next. Ensure proper error handling and logging to handle any issues that may arise during execution.

4. How does your end-to-end pipeline handle the sample ids?

   Overall: consistent sample ID usage throughout the pipeline.

   - **Task 1 (Classification):** IDs work well, correctly identified as true.
   - **Task 2 (Segmentation):** IDs seem functional, but performance drops compared to individual model run.
   - **Task 3 (Deskewing):** IDs work Perfect, achieving accurate deskewing for all images.
   - **Task 4 (Text extraction):** Cut out of IDs, but cleaning step removes them and text, leading to mostly white outputs. We also tried morphological filters, but results remain poor.

5. What are problems, what worked directly, what do you notice?

Problems:

- **Pipeline performance:** Individually well-performing models underperform when combined in the pipeline.
- **Image resolution:** Limited by our laptop, models are trained on 512x512 pixels only, which might be insufficient.

Observations:

- Models perform well individually.
- Limited image resolution might be a contributing factor.