Recovering information from pixelized credentials

Viktor Garske[1] Andreas Noack[2]

Abstract: Pixelation is a common technique to redact sensitive information like credentials in images. In this paper, we propose a system that is able to recover information from pixelized text. Our contribution consists of a neural network as well as a generic pipeline that generates a realistic training dataset considering flexible specifications including wordlists, fonts, font sizes and letter spacings. The contributed neural network is a composition of a Convolutional Neural Network (CNN), a Recurrent Neural Network (RNN) using Long short-term memory (LSTM) and a Connectionist Temporal Classification (CTC) layer to decode sequences of characters. With our approach, we achieve a Label Error Rate (LER) under 50% when taking pixelation block sizes of up to 8×8 pixels on a 22pt font into account. Thereby, our results indicate that pixelation of sensitive data does not satisfy common privacy standards.

Keywords: Neural networks, privacy, machine learning, computer vision, password, credentials, pixelized

1 Introduction

"A picture is worth a thousand words" – this applies to many situations in everyday life. We use pictures to support our explanations because they can massively accelerate the time we need to understand complex issues. Screenshots showing a misbehavior of an application on GitHub or Stack Overflow as well as linked images on Reddit are common examples how pictures are used on the internet today. On E-commerce platforms like eBay, pictures are mainly used to draw attraction to a specific product.

Fig. 1: Pixelized credential sticker on an Access Point

In most cases, the uploaders of pictures take care of their own privacy, by applying a pixelization (see Fig. 1) on confidential information like postal addresses, public IP addresses, usernames or passwords. Image manipulation tools like *GIMP* provide functions

[1] University of Applied Sciences Stralsund, Department of Electrical Engineering and Computer Science, Zur Schwedenschanze 15, 18435 Stralsund, Germany, viktor.e.garske@hochschule-stralsund.de
[2] University of Applied Sciences Stralsund, Department of Electrical Engineering and Computer Science, Zur Schwedenschanze 15, 18435 Stralsund, Germany, andreas.noack@hochschule-stralsund.de

doi:10.18420/sicherheit2022_08

to redact information in photos or screenshots conveniently. From a security point of view, optically redacted data is in many cases less secure than uploaders think.

This paper focuses on recovering information from images of credentials that are redacted by pixelation. Our contribution is a system incorporating neural networks that allows to recover credential information from cropped redacted images. In our setup, the redacted images are allowed to have a height of only four pixels in the minimum case.

The organization of this paper is as follows: Section 2 deals with related and previous work on the topic of recovering information from redacted images. We give an overview about different types of pixelation in Section 3. Section 4 introduces our approach to gain information from pixelized character sequences using a system of neural networks. In Section 5, we demonstrate how accurate our solution is. Section 6 concludes this paper and provides an outlook.

2 Related Work

The obfuscation of credentials in images has already been investigated in the literature. For instance, McPherson et al. [MSS16] studied the influence of deep learning techniques against image obfuscation. Their work focuses on mosaicing, pixelation and blurring of specific datasets, including the MNIST dataset. Different from our work, McPherson et al. concentrate on individual MNIST digits and do not take sequences of characters, like natural words or passwords, into account.

Also in 2016, Hill et al. [Hi16] presented their in-depth study of mosaicing and blurring as tools for document redaction with respect to character sequences. They use Hidden Markov Models (HMM) to model the underlying text of pixelized images as the hidden states. The trained models are used to infer the text from other pixelized images. Hill et al. note that the effort needed to adjust the parameters like font, font size and pixelation offset is high and furthermore, a parameter mismatch would reduce the recognition accuracy substantially.

Although HMMs constitute a promising approach to handle sequences, Graves et al. [Gr06] choose a different path for training such models. They utilize Recurrent Neural Networks (RNN) and remove additional preprocessing and postprocessing for the up to that point necessary segmentation steps. Their approach using RNNs in combination with the introduced Connectionist Temporal Classification (CTC) reduces training effort and is more robust to temporal and spatial noise. The approach developed by Graves et al. [Gr06] was primarily utilized in the field of speech recognition.

Studies of information recovery in pixelized text images have also been researched from a non-privacy-motivated point of view as Optical Character Recognitions (OCR) face a similar problem. In the OCR case, it is assumed that images have a bad quality which is induced unintentionally. Gilbey et al. [GS21] cover the recognition of ultra-low-resolution printed

text images that were scanned with a resolution of as little as 60 dpi. They follow a human-vision inspired approach that takes into account how our brain deals with low-resolution images. Gaussian Blur or bicubic interpolation reduce the higher-frequency information in low-resolution images and lead to a mean character level accuracy of 99.7%. As typical for OCR, Tesseract is incorporated with a custom model to provide this functionality. Tesseract uses a special RNN architecture called Long short-term memory (LSTM) [HS97] in combination with the CTC presented by Graves et al. Our approach differs because we do not focus on English texts but rather common passwords, which may also contain special characters. Thus, transfer learning with pre-trained Tesseract language models as utilized by Gilbey et al. cannot be applied to our use case directly.

Another application for deciphering low-resolution or pixelized images is the recognition of license plates. Kaiser et al. [Ka21] used CNNs to evaluate opportunities for information recovery relating to low-quality JPEG images from a forensic perspective.

There are two related open-source projects available covering depixelation of passwords. Mellema [Me20] published a tool called *Depix* which tries to resemble the content of pixelized images optically. Schatz [Sc21] implemented the HMM-based approach of Hill et al. [Hi16] in his tool *DepixHMM*. It should be noted that the term *depixelation* refers to the recovery of information in pixelized text as the pixelation itself cannot be inverted.

3 Pixelation Techniques

Pixelation is one of several methods to obfuscate text in an image. The goal is to reduce information up to a point where no human can recognize the redacted text without raising too much visual attention. A pixelation algorithm rasterizes the image with a lower resolution and averages all pixels.

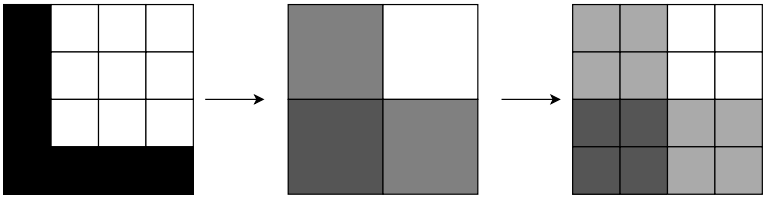


Fig. 2: Method of operation of a character pixelation with a block size of 2×2 pixels. The image with 4×4 pixels (1) will be re-rasterized and the pixel color values will be averaged (2). In the last step (3), the color values of each rectangle in (2) will be assigned to all pixels that lie in this rectangle, i.e. that contributed to the average.

Fig. 2 illustrates that averaging a rectangle of pixels is a non-reversible lossy operation which leaves certain structures behind. Another way to create a similar pixelation effect is

to scale the image down based on the block size and to scale it up using nearest neighbor interpolation again. Both methods are used in practical implementations:

- The open-source image editing software *GIMP* uses GEGL for its operations. Its *pixelize* filter implements the aforementioned averaging technique.
- *ImageMagick* or the popular *Pillow* package in Python do not support pixelation directly, so that developers are going to implement pixelation by using the double resizing technique.

In addition, a pixelation effect occurs unintentionally when a document is photographed with a too low resolution or on a too high distance.

4 Recovery of pixelized information

Pixelation is a common technique to hide credentials in photos while preserving the impression of the whole photo. It reduces the amount of information in an image but does not erase them. Our approach consists of two main components that aim to extract as much information about the underlying text of pixelized images as possible:

1. A **preprocessor** that (a) converts pixelized images to grayscale and (b) aligns the image to left and center, and crops it to a uniform height. This is an important step to increase the accuracy of the following neural network.
2. A **neural network** including a CNN and RNN with the CTC loss function builds the core of our approach.

The proposed system generates key candidates that restrict the search space in order to optimize key recovery attacks.

In the following, we begin with the description of the depixelation training pipeline, which is a generic approach that is also compatible with other solutions. Then, we describe our proposed preprocessing, and the internal structure and details about our neural network. Finally, we add some technical remarks that explain how we increase our systems performance furthermore.

4.1 Training Pipeline

In order to create a good classifier for pixelized texts, a model has to be trained. Our goal is to provide a training concept that is efficient but also fault-tolerant. Fig. 3 shows our training pipeline, which works as follows:

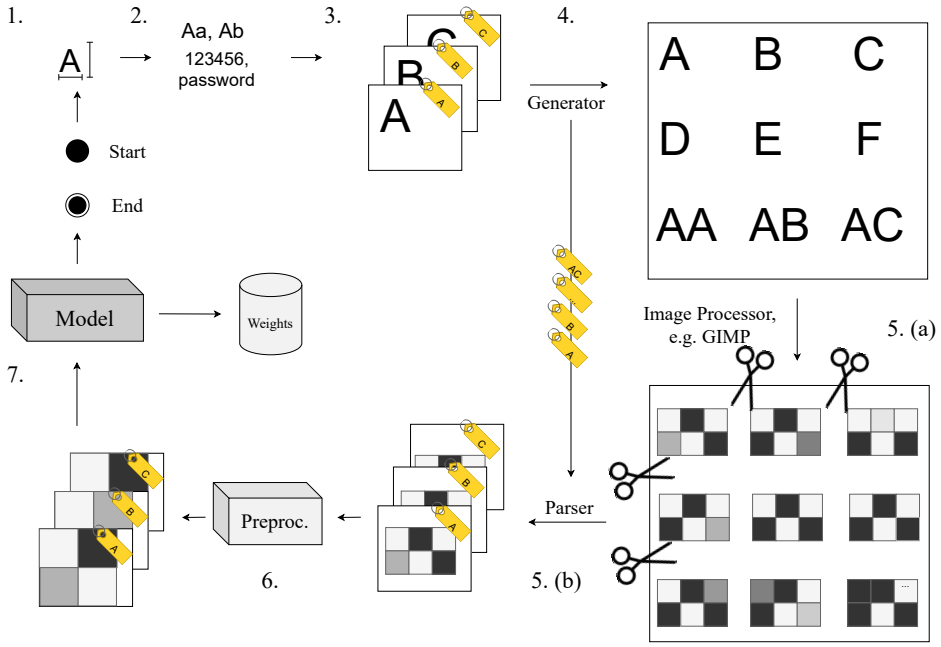


Fig. 3: Pipeline for training the model

1. **Specify parameters**. Select font, font size and pixelation block size.
2. **Generate dictionary**. Permute a predefined alphabet or extract words from a dictionary.
3. **Generate images**. Sequences of words are synthesized on a canvas with a predefined size to make it compatible to the input layer of subsequent analysis steps. The text is truncated if it exceeds the canvas size. To make the system perform more generic and fault-tolerant, noise can be added in an optional step. Further details are explained in a moment.
4. **Concatenate images**. All images of sequences are randomly shuffled and concatenated into a large collage image to allow more efficient handling.
5. **Pixelize collage**. The collage image is (a) fed to an image processor like *GIMP* or *Pillow* in order to apply the pixelation. The resulting image will be cut into single images again and (b) relabeled based on their respective preimage.
6. **Preprocess dataset**. This is an optional step that improves the quality of the pixelized images. Section 4.2 describes our contributed algorithm.
7. **Train the analysis system**. A model is trained with the pixelized images and their corresponding labels.

It is crucial to mimic the environment of the pixelized as good as possible. We assume that high-quality images like screenshots and scans act as the preimage for most pixelations. Thus, we can synthesize the training data similarly. We suggest adding Gaussian noise to train the analysis system with both full-quality and lower-quality images. The noise will be added prior to the pixelation in order to diversify the pixelation results. Additionally, any pixelation result depends on the exact position of the letters so that we also suggest several variations of letter position along the x and y axis to be included in the training set.

4.2 Preprocessing

The preprocessor is applied to pixelized images before training as well as to every test or validation image before it is fed to the neural network. Our main goal for the proposed preprocessor is to align and fit the pixelized letters in the image which are later forwarded to the neural network. First, all images are converted to grayscale with e.g. 256 discrete values. Secondly, we align and scale the image so that it fits on a canvas with a uniform height and width.

In the following, we only describe our aligning and scaling algorithms because the grayscale conversion is done by a standard function. Our algorithm takes roughly cropped pixelized images and removes empty borders first. In order to do so, it scans horizontally and vertically through the image to detect nearly empty borders around the supposed text. If any of the values in each line or column exceeds a threshold value, it will be treated as the first or, respectively, last line or column. After removing the borders, the image is scaled to the desired height using a linear standard function. To further improve the classification accuracy, the image is blurred with a Gaussian blur filter as suggested by Gilbey; Schönlieb [GS21]. Finally, the resulting image is placed on a fixed-width canvas to truncate words that exceed our character limit on the right side.

4.3 Neural Network

Neural networks (NNs) are particularly suitable for classification or regression tasks, as many publications show over the last decades [GR01][MV15][GBC16]. A typical application is image classification, where the network accepts the pixels of an image on the input layer and outputs probabilities for a given number of classifications on the output layer. Trained with a given data set, the neural network is later capable of finding classifications for unknown input data, whereby its accuracy depends on the suitability of the training data and the internal structure of the neural network.

For the depixelation use case, a neural network architecture with two convolutional and pooling layers performs reliably with images of single pixelized characters as shown by McPherson et al. [MSS16]. In the case of longer character/letter sequences, the accuracy

of the naive approach decreases dramatically. There are, however, different approaches to counter the accuracy loss:

- **Pre- and Postprocessing**. An extensive preprocessing of pixelized character sequences produces single images of pixelized characters that can then be fed to the neural network. A postprocessor combines the predicted labels of the neural network with respect to their corresponding probabilities to character sequences, again.
- **Connectionist Temporal Classification**. The CTC loss function that is used in combination with the Recurrent Neural Network (RNN) was proposed by Graves et al. [Gr06]. In the area of speech recognition, research results indicate that it outperforms the Hidden Markov Model (HMM) approach. HMM is one of the most promising approaches to deal with our use case, namely depixelation, as shown by Hill et al. [Hi16].

For this paper, we choose the Connectionist Temporal Classification (CTC) approach by Graves et al. [Gr06] and adapt it to our use case, the depixelation of credentials.

The core of a CTC-fitted recurrent neural network (RNN) is the loss function that does not only consider the raw input data but also takes the length of the input into account. The CTC-RNN implementation by Soullard et al. [SRP19] uses the edit distance, which is explained later, in order to calculate the similarity between the predicted and the real label.

After preprocessing, we proceed with one channel grayscale images in the dimension of 192×32 pixels that are fed into the CNN. The neural network architecture was inspired by ResNet [He15] and DenseNet [HLW16], and we make use of skip connections to enable more efficient training. In comparison to ResNet, we use fewer layers, again for efficiency reasons. Fig. 6 (Appendix) gives a structural overview about the complete neural network architecture. To avoid overfitting, Gaussian noise is added at the beginning of the training.

Intermediate layers prepare the output of the CNN for the following LSTM by flattening and reshaping. Two LSTM layers process the output and feed it into the CTC layer. This CTC layer combines the output of the LSTMs together with the content width of the pixelized image, and the value and length of its corresponding label sequence.

4.4 Technical Remarks

We design our solution to be as close as possible to a real world pixelation scenario by deploying standard fonts and font sizes as well as standard tools like *GIMP* for the pixelation process.

For realistic results, we use a standard wordlist from the *SecLists* project by Miessler et al. [MHg21]. The selected wordlist is named `10-million-password-list-top-10000.txt` and

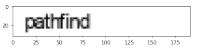
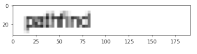
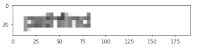
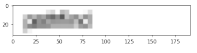
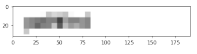
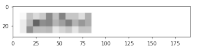
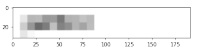
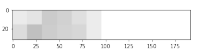
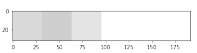
contains the top 10,000 entries of a collection compiling the 10 million most used passwords. Entries include combinations of letters, numbers and special characters in one word.

We noticed that a pixelation result does not only depend on the image but also on its position in the coordinate system, i.e. after realigning two pixelations from the same image at different positions, the result differs. To improve the prediction results, we let our image synthesizer generate text images at several positions on the canvas to cover the variations in the pixelation results.

The text image synthesis will be performed by the Python *Pillow* package [LCC21]. Furthermore, we modify the functionality of the `ImageDraw` class because we found out that it did not take letter spacings into account. In our pixelation pipeline, we call *GIMP* in batch mode using the *PNG* file format with compression level 6, whereby level 1 has the best speed and level 9 the best compression rate.

5 Results

The performance of the proposed information recovery system depends on the used font size relative to the size of the blocks that are created in the pixelation step. We use the Arial font and define a font size of 22pt for our tests and evaluate the error rate in dependency of pixel block sizes between 2×2 and 32×32 pixels, whereby our canvas has a maximum height of 32 pixels. As you can see in Tab. 1, a pixelation block size of 8×8 pixels results in a height of four blocks. The last two block sizes, 16×16 pixels and 32×32 demonstrate edge cases.

Block Size	Example Image	Label Error Rate	Sequence Error Rate
2×2		2.99%	17.97%
3×3		16.00%	57.00%
4×4		20.47%	65.63%
5×5		36.17%	87.50%
6×6		28.92%	78.90%
7×7		41.17%	90.63%
8×8		49.93%	97.66%
16×16		61.09%	99.22%
32×32		69.25%	99.22%

Tab. 1: Error rate of the proposed depixelation system, see Fig. 4 for chart illustration.

For comparability reasons, we use two established metrics, the Label Error Rate (LER) proposed by Graves et al. [Gr06] and the Sequence Error Rate (SER) by Soullard et al. [SRP19].

The Label Error Rate (LER) metric measures the **mean normalized** edit distances between classifications and preimage labels and is defined as follows:

$$LER(h, S') = \frac{1}{|S'|} \sum_{(\mathbf{x},\mathbf{z}\in S')} \frac{ED(h(\mathbf{x}), \mathbf{z})}{|\mathbf{z}|} \tag{1}$$

whereby the LER is computed for a temporal classifier h (i.e. the model) and a given test set S'. x stands for the input sequences and z for corresponding labels. Both are compared with the edit distance function $ED(\cdot, \cdot)$ which is the number of characters that must be changed to make both parameters equal. The LER results in Tab. 1 approve our assumption that the error rate increases proportional to the block size, Fig. 4 (Appendix) shows even better that there is a linear relationship.

As a second metric, we choose the Sequence Error Rate (SER) that reveals the percentage of **completely correct** predictions in the test set. Our test results show that the probability to predict a pixelized text without any errors drops dramatically with increasing information loss due to larger pixelation block sizes.

As a consequence of different approaches and data representations, a comparison to previous publications is a challenging task. When comparing to [MSS16], we have to keep in mind that this experiment focused on individual pictures of digits in the MNIST dataset and not on sequences of letters, digits and special characters, which emerges as an easier problem. We can show that a reasonable recognition rate is still realistic, even if character sequences instead of single digits are investigated.

The HMM approach by Hill et al. [Hi16] considers character sequences. Their goal is to specifically train an HMM-based recognizer for a given set of specifications. This includes finding the correct offset, as well as other parameters like font size or font type. As a result, this approach requires high effort for each image sample but outperforms more-general approaches like ours, especially at high mosaic grid sizes. Unfortunately, we could not examine and compare our results in detail as their code and data set for the claimed accuracy is not public. Our results indicate that our system is slightly more robust to offset variations than the HMM approach as presented in Fig. 5 (Appendix).

It becomes clear that the probability for a successful information recovery depends on the block size and training. Higher block sizes result in higher LERs which in turn lead to higher privacy probabilities for the redacted text. Proper training and dataset selection, however, is vital to enable low LERs. Consequently, our network design leads to suitable results in less than 15 epochs of training using a dataset of about 25,000 samples. Nonetheless, training has to be monitored with respect to loss metrics to ensure a high quality.

Predictions of the network have to be assessed by a human operator in order to evaluate the plausibility in respect to the supposed content. We examined our neural network structure for superfluous elements and could observe an essential role of the CNN part. When removed or replaced by more dense layers, the LER rises up to 50 percentage points.

6 Conclusion and Future Work

In this paper we have analyzed the privacy of pixelation whereby our research is especially focused on the obfuscation of credentials like passwords. We make two major contributions, firstly, (1) a NN-based information recovery system for pixelized text in images with a competitive accuracy compared to previous HMM-based approaches and secondly, (2) a generic and flexible training pipeline for depixelation systems.

The proposed system consists of a combination of Convolutional Neural Network (CNN) and Long short-term memory (LSTM) network together with a Connectionist Temporal Classification (CTC) layer. Thereby, we reach a mean error rate, namely the Label Error Rate (LER), better than 50% with pixelation block sizes up to 8×8 pixels. Our results indicate that the proposed system still works even when humans are no longer able to depixelize images anymore. This leads to a substantial privacy impact on the security of pixelation algorithms used in standard software. However, the performance of our approach decreases if sufficient high block sizes are used. We discovered that the label error rate rises above 70% if a block is at least as big as the pixelized character.

Furthermore, we developed a complete training pipeline for obfuscation problems that comprises a generic approach for various depixelation concepts, e.g. based on HMMs or NNs, and is able to employ standard software as subroutines. The training pipeline allows efficient and automated training, while it is flexible in terms of parameter variations.

In the future, we plan to investigate several techniques like ensemble methods or the separation of image classification and content recognition to further improve the error rate of our system. Additionally, our network architecture is going to be revised with the idea to find alternatives for the CNNs or LSTMs in order to either decrease the training effort or improve the accuracy.

References

[GBC16] Goodfellow, I.; Bengio, Y.; Courville, A.: Deep learning. MIT press, 2016.

[GR01] Giacinto, G.; Roli, F.: Design of effective neural network ensembles for image classification purposes. Image and Vision Computing 19/9-10, pp. 699–707, 2001.

[Gr06] Graves, A.; Fernández, S.; Gomez, F.; Schmidhuber, J.: Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In: Proceedings of the 23rd International Conference on Machine Learning. ICML '06, Association for Computing Machinery, Pittsburgh, Pennsylvania, USA, pp. 369–376, 2006, ISBN: 1595933832, URL: https://doi.org/10.1145/1143844.1143891.

[GS21] Gilbey, J.; Schönlieb, C.-B.: An end-to-end Optical Character Recognition approach for ultra-low-resolution printed text images. arXiv preprint arXiv:2105.04515/, 2021.

[He15] He, K.; Zhang, X.; Ren, S.; Sun, J.: Deep Residual Learning for Image Recognition. CoRR abs/1512.03385/, 2015, arXiv: 1512.03385, URL: http://arxiv.org/abs/1512.03385.

[Hi16] Hill, S.; Zhou, Z.; Saul, L.; Shacham, H.: On the (in) effectiveness of mosaicing and blurring as tools for document redaction. Proceedings on Privacy Enhancing Technologies 2016/4, pp. 403–417, 2016.

[HLW16] Huang, G.; Liu, Z.; Weinberger, K. Q.: Densely Connected Convolutional Networks. CoRR abs/1608.06993/, 2016, arXiv: 1608.06993, URL: http://arxiv.org/abs/1608.06993.

[HS97] Hochreiter, S.; Schmidhuber, J.: Long Short-term Memory. Neural computation 9/, pp. 1735–80, Dec. 1997.

[Ka21] Kaiser, P.; Schirrmacher, F.; Lorch, B.; Rieß, C.: Learning to Decipher License Plates in Severely Degraded Images. In: MultiMedia FORensics in the WILD - accepted for publication. Jan. 11–11, 2021, URL: https://faui1-files.cs.fau.de/public/publications/mmsec/2021-Schirrmacher-LTD.pdf.

[LCC21] Lundh, F.; Clark, A.; Contributors: Pillow: The friendly PIL fork (Python Imaging Library) repository, Version 8.3.1, Apr. 1, 2021, URL: https://github.com/python-pillow/Pillow/tree/8.3.1.

[Me20] Mellema, S.: Depix, 2020, URL: https://github.com/beurtschipper/Depix.

[MHg21] Miessler, D.; Haddix, J.; g0tmi1k: SecLists, Mar. 1, 2021, URL: https://github.com/danielmiessler/SecLists.

[MSS16] McPherson, R.; Shokri, R.; Shmatikov, V.: Defeating Image Obfuscation with Deep Learning, 2016, arXiv: 1609.00408 [cs.CR].

[MV15] McDonnell, M. D.; Vladusich, T.: Enhanced image classification with a fast-learning shallow convolutional neural network. In: 2015 International Joint Conference on Neural Networks (IJCNN). IEEE, pp. 1–7, 2015.

[Sc21] Schatz, J.: DepixHMM, 2021, URL: https://github.com/JonasSchatz/DepixHMM.

[SRP19] Soullard, Y.; Ruffino, C.; Paquet, T.: CTCModel: a Keras Model for Connectionist Temporal Classification, 2019, arXiv: 1901.07957 [cs.LG].

A Additional Information

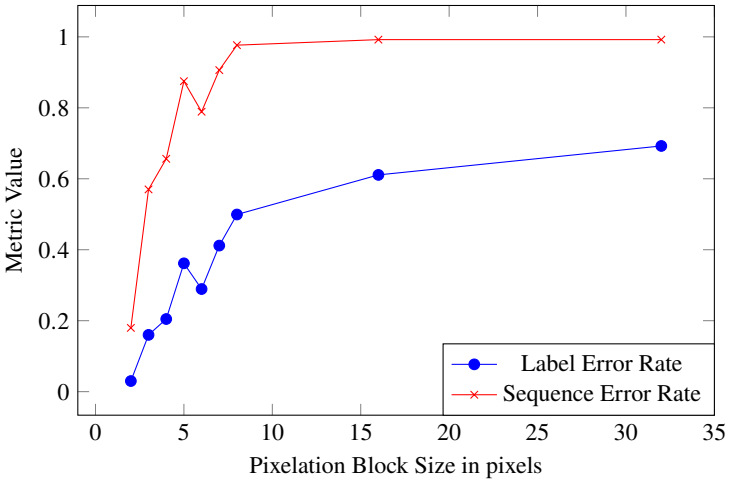


Fig. 4: Error rates in relation to the pixelation block size

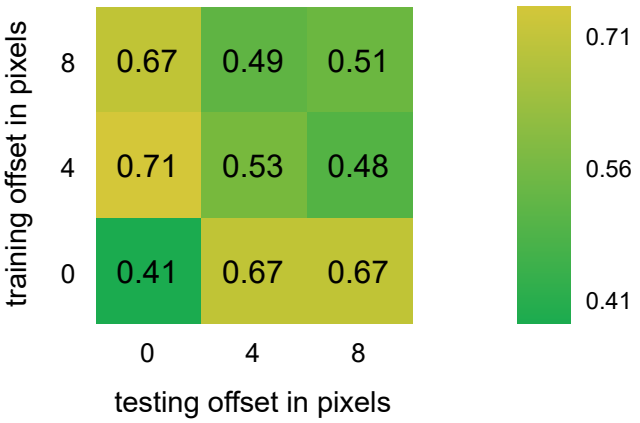
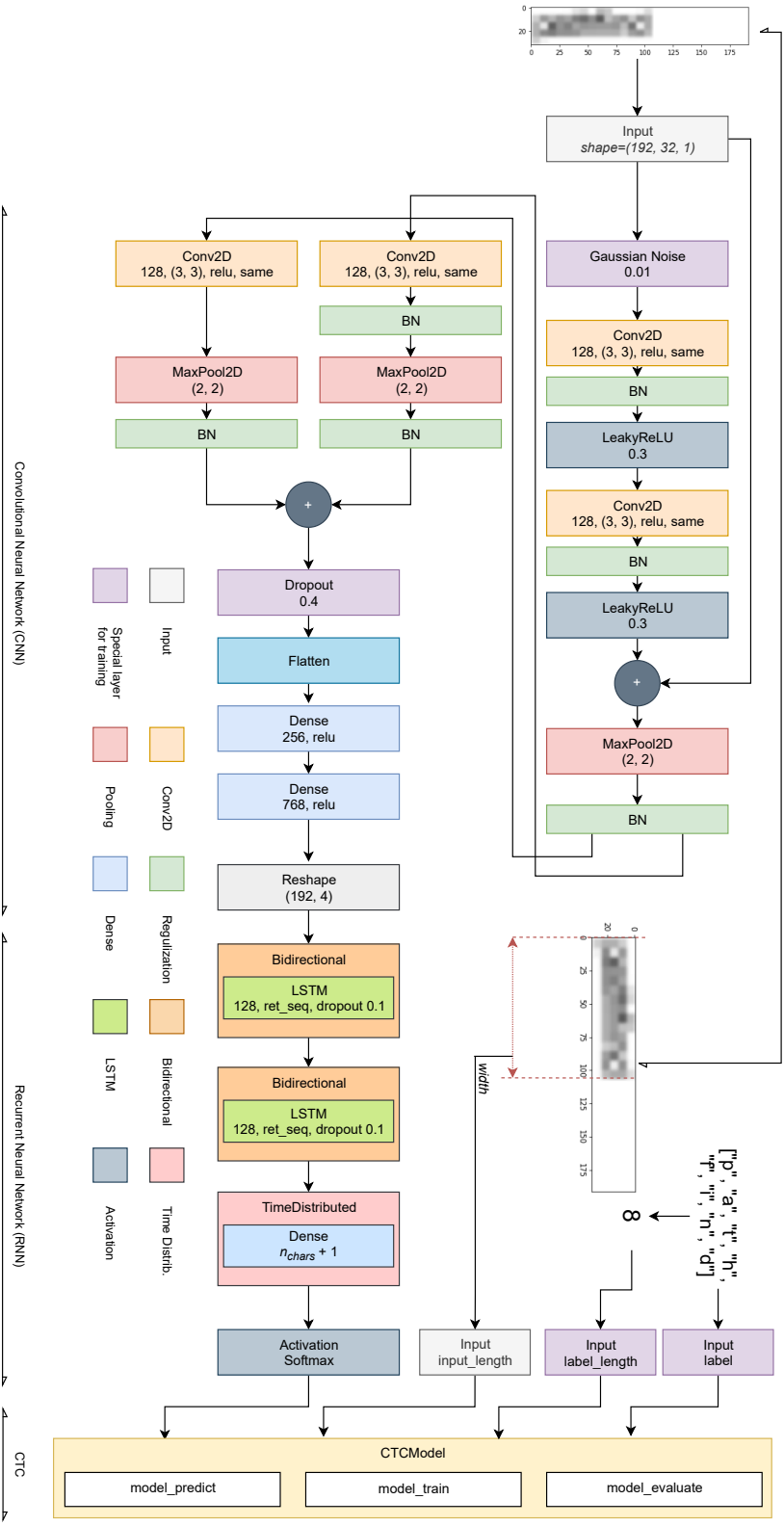


Fig. 5: A LER confusion matrix for offset variations when using a pixelation block size of 6×6 pixels


Input
shape=(192, 32, 1)
Conv2D
128, (3, 3), relu, same
Conv2D
128, (3, 3), relu, same
BN
MaxPool2D
(2, 2)
MaxPool2D
(2, 2)
BN
BN
Gaussian Noise
0.01
Conv2D
128, (3, 3), relu, same
BN
LeakyReLU
0.3
Conv2D
128, (3, 3), relu, same
BN
LeakyReLU
0.3
MaxPool2D
(2, 2)
BN
Dropout
0.4
Flatten
Dense
256, relu
Dense
768, relu
Reshape
(192, 4)
Bidirectional
LSTM
128, ret_seq, dropout 0.1
Bidirectional
LSTM
128, ret_seq, dropout 0.1
TimeDistributed
Dense
n_{chars} + 1
Activation
Softmax
Input
input_length
Input
label_length
Input
label
width
8
["p", "a", "t", "h", "n", "p"]
Convolutional Neural Network (CNN)
Recurrent Neural Network (RNN)
CTC
Special layer for training
Input
Pooling
Conv2D
Dense
Regulzation
LSTM
Bidirectional
Activation
Time Distrib.
CTCModel
model_predict
model_train
model_evaluate