Denoising Audio with Urban Sounds

Judith Kohler, Ronja Lindemann, Marieke-Kristin Schmiesing

Lecture: Implementing ANNs with TensorFlow

Term: Winter Term 22/23

Date: 31.03.23

Contents

1 Motivation **3**

2 Literature **3**

2.1 Deep Learning in Speech Enhancement 3

2.2 Types of Neuronal Networks in audio denoising 4

2.3 The U-Net-Architecture in Speech Enhancement 5

3 Methods **6**

3.1 Data 6

3.2 U-Net 7

3.3 Spectograms 7

4 Implementation **9**

4.1 Preprocessing 9

4.2 Model 10

4.3 Postprocessing 11

5 Results **11**

5.1 Validation Loss 11

5.2 Influence of Learning Rate 13

5.3 Influence of Dropout Layers 13

5.4 Optimizers and Loss Function 13

5.5 Increasing the Receptive Field 16

5.6 Unknown Data 17

6 Discussion **17**

1 Motivation

When deciding on a project, two aspects were especially important to us. We wanted a project that had practical applications and that offered the opportunity to us to learn about a new aspect of neuronal networks. We found that audio denoising fulfilled both. The unintelligibility of noisy speech and the unpleasantness of noisy music are problems that we can relate to, as we encounter them in everyday life, e.g., when making phone calls, watching videos, or listening to radio broadcasts, scratched records, or cassette tapes. There are many applications of audio denoising: It can be used for audio restoration (Moliner and Välimäki, 2022) and is essential for designing hearing aids. It is also often a part of the development of audio technical devices like mobile communication systems and headphones and of automatic speech recognition (Nossier et al., 2020, p. 1). Additionally, it offered the opportunity to learn something new, as audio data is something we never worked with before and U-Nets are a CNN structure we were unfamiliar with.

2 Literature

2.1 Deep Learning in Speech Enhancement

A look at the literature shows that there are many very different approaches to audio denoising in Deep Learning (for more detailed reviews, compare: Yuliani et al., 2021 and Nossier et al., 2020). This versatility does not only result from the fact that Deep Learning is still a constantly evolving field: Different network architectures have different advantages and disadvantages and the choice of architecture depends on the specific task, the data structure, and given resources. On the one hand, there is a trade-off between efficiency and performance. Audio denoising is a challenging task, especially when it comes to non-stationary noise, multiple noise sources, and noise that has speech features like babble noise and reverberation (ibid., pp. 1, 15, 24). This is why models are often quite deep or even combine different architectures. But complex architectures include a lot of hyperparameters which leads to a challenging training, a lack of comprehensibility, and a need for great computational resources. Those are not only expensive but often not available during practical, real-time application. Especially when it comes to small mobile devices like hearing aids (ibid., pp. 24–25). On the other hand, there is a trade-off between speech intelligibility and noise reduction: networks that thoroughly remove noise often seem to affect the speech features as well (ibid., p. 16).

2.2 Types of Neuronal Networks in audio denoising

The **Multilayer Perceptron (MLP)** is one of the architectures commonly used for speech enhancement. This fully-connected feed-forward network learns a single complex underlying mapping function. While it has had some improvements compared to purely statistical methods, it has two main drawbacks: 1. It uses no spatial or temporal information and is thus not able to have contextual knowledge and generalize beyond the data set. 2. It uses a lot of weights and has a great computational cost. (Nossier et al., 2020, p. 5)

Long Short-Term Memory (LSTM) and other **recurrent neural network (RNN)** architectures were shown to outperform Deep Neural Network as they can make use of the contextual information of audio sequences, but they are very complex to compute and train (Yuliani et al., 2021, p. 22).

The **Convolutional Neural Network (CNN)** is either used with 1D-convolutions or directly on wave inputs or 2D-convolutions on spectrograms. In general, CNNs gained a lot of attention in speech enhancement as they are more efficient than RNNs through parameter sharing and more effective than MLPs because they also consider contextual information. (ibid., p. 22)

A **Denoising Autoencoder (DAE)** learns unsupervised just from labeled input data. Usually it is quite effective in noise removal as during the compression process of the encoder relevant speech features are condensed. But it is a challenge to reconstruct the full sound quality in the decoder. (Nossier et al., 2020, p. 6)

This is a potential advantage of the **Generative Adversarial Network (GAN)**. Like in a DAE, the generator of the GAN tries to generate clean data, but the discriminator of GAN directly compares input and output. The learning target of the generator is to generate realistic, input-like data. But training the two interdependent networks can be challenging, and GANs were not proven to be very effective in speech enhancement. (ibid., p. 7)

The learning objective of these denoising networks can be differentiated as masking and mapping tasks: Masking aims to learn noise masks which can be used to distinguish speech and noise in audios. Masking is a classification task. Mapping on the other hand is a regression task that aims to map the noisy input directly to a clean output.

As noise and speech features can differ a lot, overfitting is a common problem in audio denoising. The generalization capability of a network is tested by feeding it unseen data, potentially testing other languages and noise. (ibid., p. 32)

Another challenge is the evaluation of network performance. The human perception of clarity, intelligibility, and sound quality are not easily mathematically representable. There are different complex methods like Perceptual Evaluation of Speech Quality (PESQ) and

Short-Time Objective Intelligibility (STOI), which try to model human sound perception and understanding. These are often additionally combined with subjective evaluations. (Xing Luo, 2019, p. 9)

2.3 The U-Net-Architecture in Speech Enhancement

A very efficient and quite effective network architecture for audio denoising tasks is the U-Net (Nustede and Anemüller, 2021) which was originally proposed for biomedical image segmentation tasks and was named after the U-like shape of its architecture (Ronneberger, Fischer, and Brox, 2015). A U-net is a convolutional autoencoder. It is used either with spectrograms (e.g. Yuliani et al., 2021, Moliner and Välimäki, 2022, Grzywalski and Drgas, 2018) or wave-data (e.g. Kang, Huang, and Lu, 2022, Macartney and Weyde, 2018, Ali, Brutti, and Falavigna, 2020). While recent studies use mainly 1D-wave-convolutions as they have achieved better performances for complex noise like reverberation (Stoller, Ewert, and Dixon, 2018), spectrogram-U-nets work well, even with less complex, smaller, and shallower networks (Nustede and Anemüller, 2021, p. 2).

The following figure (Figure 1) is used to illustrate the basic architecture of a U-net:

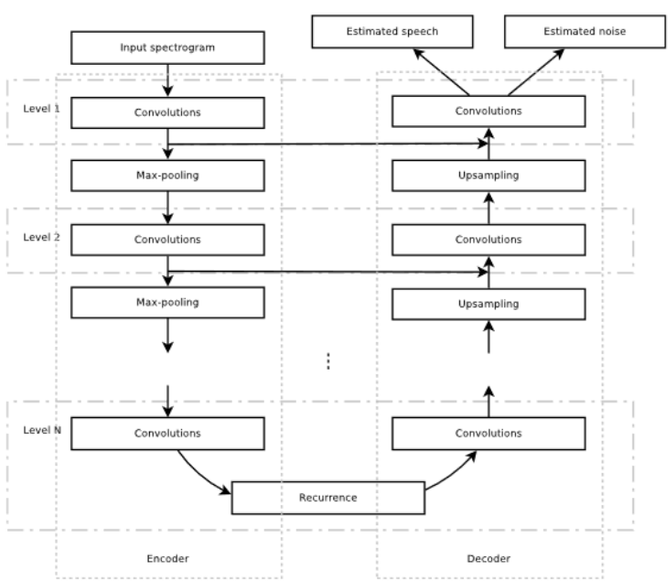

Figure 1: A U-Net for denoising spectrograms through masking -
Source: Grzywalski and Drgas, 2018, p. 1

On the left side of the U-net is the encoder. The purpose of the encoder is to down-sample the image while extracting important features. The objective of the right side of the U-net is to reconstruct the feature map through up-sampling. Each expansive block draws data from a contracting block on the same "height" and concatenates it with the current data. The last part is the bridge - a convolution layer that connects the two blocks with one another. (Ronneberger, Fischer, and Brox, 2015, p. 4)

The skip connections between each block of the encoder to the decoder are employed to

maintain the spatial or temporal information that each step of the encoder has. Through this process, information that might get lost in the compression process can be reintroduced in the decoding (Ronneberger, Fischer, and Brox, 2015, p. 4). Strong indications were found that those skip-connections improve the performance of U-Nets (Grzywalski and Drgas, 2018, p. 82).

An important feature that is discussed in U-Net research is the design of receptive fields of each layer in the autoencoder: In a convolutional autoencoder, the receptive field refers to the region of the input that is used to compute the output of a particular neuron in each step of the network. Thus, the receptive field of a neuron determines the contextual frame (ibid., p. 84). Some U-Net architectures in audio denoising use dilations in the encoder to increase the receptive field (e.g. Nustede and Anemüller, 2021, Stoller, Ewert, and Dixon, 2018, Ali, Brutti, and Falavigna, 2020 and Kang, Huang, and Lu, 2022). This figure servers to visualize dilations:

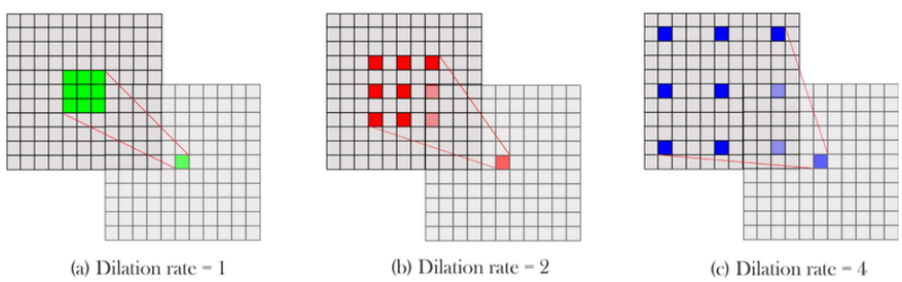

Figure 2: Visualizing dilation - Source: Negahbani et al., 2021, p. 6

There are many extensions of the U-Net in speech enhancement in order to further improve the denoising, like variational approaches (Nustede and Anemüller, 2021), self-attention mechanisms, a combination with LSTM or compressed sensing loss (Kang, Huang, and Lu, 2022, p. 1).

3 Methods

3.1 Data

A first, suitable dataset for our work was found in the LibriSpeech ASR corpus (Panayotov et al., 2023): The audios are in English, mainly novels read by volunteers with different accents. As a first proof of concept, Gaussian noise is added to the original data. Later, to provide some challenge and optimize the network, we also worked with the Microsoft Scalable Noisy Speech Dataset (MS-SNSD) (Reddy et al., 2019). It draws its clean speech from the Edinburgh 56 speaker dataset (about 400 sentences per speaker) and the Pitch Tracking Database from Graz University of Technology (20 speakers, in total 4720 recorded sentences). The dataset also provides code to add a variety of background noises to the data. This includes everyday sounds like air conditioners, domestic sounds like vacuum cleaners,

and even more challenging noise with speech features. We used 42 noise files with 8 different types of noise.

3.2 U-Net

Taking the available computational resources into consideration, as well as our limited time and experience, we wanted to work with an efficient network that does not consist of too many hyperparameters and is not too complex to train and optimize. This is why we chose to work with a basic spectrogram-U-Net. We decided to base the specific architecture of our U-Net on the paper "Towards speech enhancement using a variational U-Net architecture" (Nustede and Anemüller, 2021), as it also tries to denoise single channel audio with a spectrogram-U-Net (ibid., p. 1), although we work with a simplified non-variational version of the proposed architecture. Another difference is that our network maps the noisy data directly to the clean targets, so we work with a supervised regression task. For insight into the specific architecture of our model, see Figure 3.

3.3 Spectograms

Audio is determined by magnitude and phase changes over time (Cryer, 1986). Spectrograms are a way to visually represent an audio signal's magnitude as it varies over time. We use them as our model input. They are calculated via the Short-Time Fast Fourier Transforms (STFT). Fast Fourier Transforms analyze the frequency of a signal. But since our audio signal is non-periodic, it also changes with time. Therefore, we use the short-time FFT – it computes the FFT on multiple overlapping time-windows of the audio. (Roberts, 2023).
It was shown that the magnitude is crucial for our speech intelligibility (Wang and Lim, 1982). There are different ways to compute the magnitude on the basis of wave data. The mel-spectrogram, for example, tries to model the human perception of the magnitude (Roberts, 2023). As the mel-spectrogram is computationally more efficient than the sfts-magnitude (McFee et al., 2020), we trained our network on mel-spectrograms.
Good estimations of a denoised magnitude are intelligible, but sound rather unpleasant. To have better sound quality, we have to combine our magnitude with the phase. Unfortunately, phase estimation is very complicated. Additionally, a bad phase estimation sounds significantly worse than a bad original noisy phase, and a good estimation often barely improves it (Wang and Lim, 1982). Therefore, we settled for combining our magnitude estimation with the noisy phase, which was also done in the underlying architecture (Nustede and Anemüller, 2021, p. 3). Like Nustede and Altmüller we feed the magnitude-spectrogram without the phase into the network (ibid., p. 3).

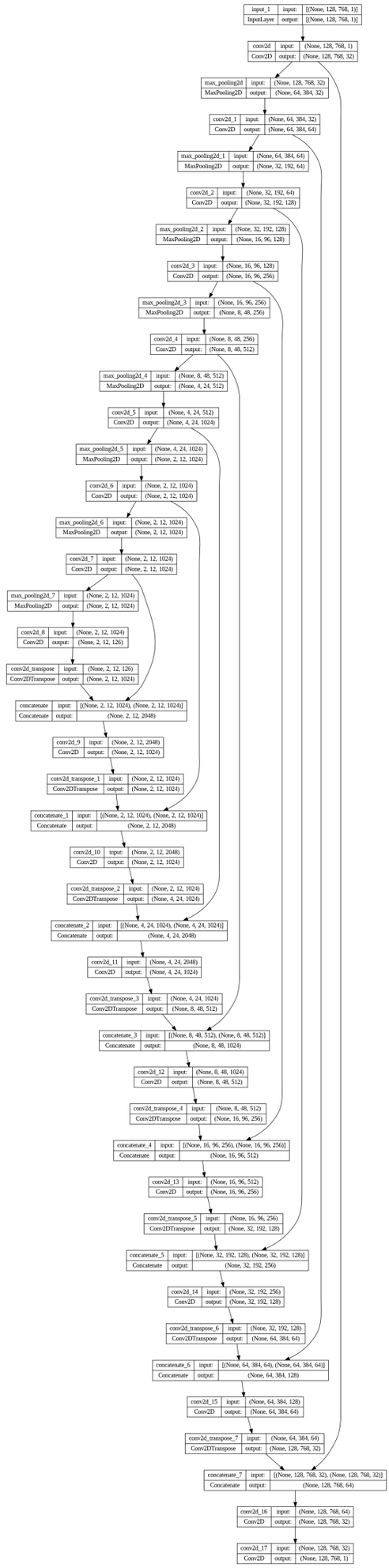

Figure 3: our implementation of the U-Net - Source: Visualization of our model output taken on 03/16/23.

4 Implementation

As we did not have any experience working with audio data before, we based our first draft roughly on this implementation of a DAE (tanishkaa31, 2023), which we then changed and adapted for our project.

4.1 Preprocessing

We first get the data in 125 (or 38 for the Gaussian noise) snippets of about 10–15s length, either as a .wav or .flac. As such, a relevant part of preprocessing is formatting and resizing the files so that the U-Net can work with it (a size that can be divided by two six times). We first convert the audio into a np.array and then cut or expand all np.arrays such that they have the same length. If we wanted to add Gaussian noise we would do it now. With the MS-SNSD noise, the noise is already added to the uploaded data.
Then we convert the audio into a spectrogram. A problem with this is, that when we convert the mel-spectrogram back into audio data, the reconstructed audio is recognizable but sounds significantly worse than the original. As this is true not only for the estimated but also for the clean spectrograms, it is clear that there is an information loss in the mel-spectrogram: The mel-spectrogram depends only on the amplitude of the audio frequencies. To have a complete representation of audio data information, we would also need information about frequency phases. This can be represented in a complex spectrogram - each time frame and frequency bin assigns a complex value. The phase is represented by the angle of the vector, which is described by both the imaginary and real part of that value, and the amplitude is represented by the magnitude of that vector.

Our first solution to this was not to work with a melspretrogram, but to compute a complex spectrogram with the librosa-sfts-function to then create a spectrogram with only the absolute values. The computed sfts-magnitude-spectrogram can be used to estimate the phase with the Griffin-Lim algorithm, which takes the given magnitude-spectrogram and randomly initializes the phase. It inverts the estimated complex spectrogram to an audio signal and then converts the audio signal back to a complex spectrogram. After, it updates the estimated phase based on the phase aspect of that complex spectrogram. This is repeated for a chosen amount of iterations (Librosa-development-team, 2023). The problem with this solution was that the sampling size necessary to provide good audio quality led to spectrogram sizes that were too large for our hardware to handle. Thus, we returned to mel-spectrograms We found the librosa.feature.inverse.mel_to_stft-function (McFee et al., 2020), which converts a mel-spectrogram into a sfts-spectrogram with only absolute values. The sfts-spectrogram can then be combined with the phase information of the noisy input to reconstruct the complex values. This produces decent output.

The final part of the preprocessing is the usual CNN preprocessing. We add a mock color channel and normalize the data to be between -1 and 1. Then we convert spectrograms into an actual tf.dataset, split it randomly into training and validation data, and batch it.

4.2 Model

This pseudocode demonstrates the specific structure of the encoder-decoder-blocks in our implementation:

```
# Model input
input = layers.Input(shape=(frequency bins, time frames, color channels))

# 1. Encoder block:
conv1=Convolution2D(filters, kernel_size, activation, padding, dilation)(input)
dropout1 = layers.Dropout(rate=0.2)(conv1)
pool1=MaxPooling2D(pool_size, strides, padding)(dropout1)

# Further encoder blocks...

# Bridge from Encoder to Decoder:

conv9=Convolution2D(filters, kernel_size, activation, padding, dilation)(pool8)

# Decoder blocks...

# 8. Decoder block:

upconv8 = TransposedConvolution2D(filters, kernel_size, activation, padding)(convD7)
concat8 = Concatenate()([upconv8, dropout1])
convD8 = Convolution2D(filters, kernel_size, activation, padding, dilation)(concat8)

# Model output
output = Convolution2D(1, kernel_size, activation, padding)(convD8)
```

In the actual model, we use 8 encoder/decoder pairs plus one bridge, with the channel dimensions (64, 128, 256, 512, 1024, 1024, 1024, 1024). We reduce the input from its original size of (128x768) down to a (2x12)-representation in the latent space.
Each encoder block consists of a Conv2D layer with the current feature size, and a 3x3 kernel. Then we have a dropout layer with a dropout rate of 0.2, followed by a MaxPooling2D layer with a pool_size of 3 and a stride of 2 in the blocks where we are still down-sampling.
After the encoder we have the bridge which connects the encoder and decoder and consists of another convolution layer. The decoder block consists of a Conv2DTranspose layer with the appropriate filter size and, as soon as we start up-sampling, a stride of 2, which increases the size of the input. Then we have the special part of a U-Net: a concatenation layer.

This concatenation layer adds input from the encoder block of the same "height" to the up-sampling chain. The following Conv2D layer helps to extract the important information from those concatenated layers. Its parameters are equivalent to those of the transposed Conv2D layer. The output layer is another convolution layer with a filter size of one.

To save ourselves time and debugging effort, we decided to implement the model with the tf.fit function. This made it easy to make changes to the model and also has a built-in loss output. In the process of trying to figure out why our validation loss was so low, we did build a version for which we wrote our own train and test functions, but as it offered no insight and was more difficult to work with we stuck to the original.

4.3 Postprocessing

Since audio denoising is very dependent on human perception, we did not want to rely on the TensorBoard to test the quality of our model.
During the testing phase, we did look at different data points to get an impression of the performance. To get an easily comparable output and to better demonstrate the progress in this paper, we later decided to send the same two, unseen audios through the trained network. One with generic crackling in the background, and one with talking as background noise, as so-called "babble" is considered particularly difficult to denoise.
We then reverse the model-specific preprocessing and visualize the spectrogram we predicted, as well as the noisy and original spectrogram for comparison. To improve our evaluation, we also converted the spectrogram back to audio and listened to it using the Ipython library.
We also implemented the possibility to run completely different, new data through the dataset.

5 Results

5.1 Validation Loss

As a first assessment of the efficacy of our model, we added basic Gaussian noise to the data. Except for the issue with the phase information we outlined above, this worked very well, even with very few epochs and using default hyperparameters (batch_size=10, optimizer=Adam, loss=0.001, loss_function=mean_squared_error, epochs=4). However, compared to the training loss, the validation loss was much better. We attributed this to either the insufficient quantity or quality of the dataset at our disposal. To tackle this issue, we experimented with a more intricate dataset and substituted Gaussian noise with background noise from the Microsoft Scalable Noisy Speech Dataset (Reddy et al., 2019).

As expected, since the data is much more complex than before, the output was considerably worse and unfortunately, the issue with the validation loss persisted, if less pronounced. We doubled the sample size, which improved both the predictions and the issue with the validation loss a little. It did not fix it though, so in another effort to counteract it, we tried to supplement our models with BatchNorm() layers after each convolutional layer to try to smooth out the loss landscape. This wielded rather strange results, regardless of the learning rate. Figure 4 and Figure 5 show the results for a learning rate of 0.0007 and 0.005, respectively.

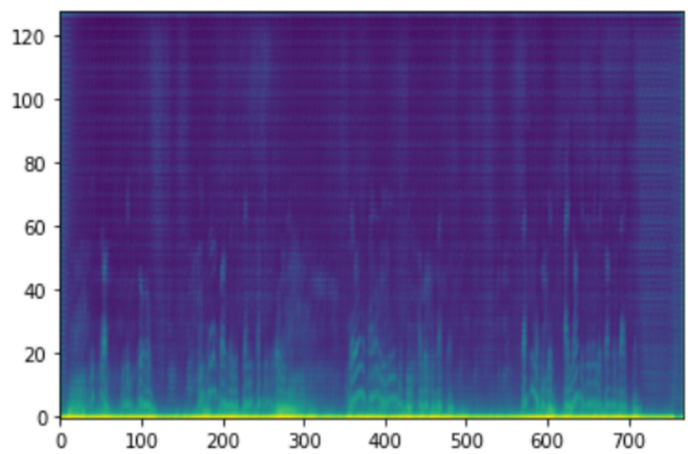

Figure 4: with BatchNorm(), Learning Rate = 0.0007 - Source: screenshot of our model output

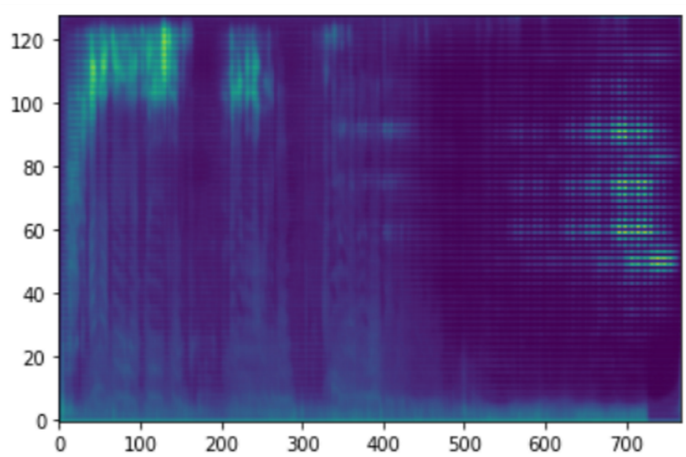

Figure 5: with BatchNorm(), Learning Rate = 0.005 - Source: screenshot of our model output

It is likely that the variance of the input data is too high between batches, causing over-regularization and the introduction of noise to the spectrogram. Unfortunately, RAM limitations did not allow us to fine-tune it any further, and we had to give up on using batch normalization layers.

To make sure that it was not just an issue of the validation data being somehow simpler than the training data, we shuffled the data before the split. However, this did not change anything, indicating that the problem has another cause. Our changes to the batch size, which we will discuss later, also had no significant effects on the relation between validation

loss and training loss. What did help a little was increasing the number of epochs, but the validation loss was still higher in the first few epochs. The underlying issue here might just be that the validation and training data are from the same dataset and both speech and added noise are very similar. In addition to that, the training set is not particularly large.

5.2 Influence of Learning Rate

Now, to optimize the model output, we started by adjusting the learning rate. After some experimenting, we found that a learning rate of 0.0006 garnered the best results. This learning rate is relatively small, but cautious steps at the beginning of the learning are beneficial in our case, especially as the noise of each of the audio clips is so different. As shown in Figure 6 this already leads to a relatively clear output.

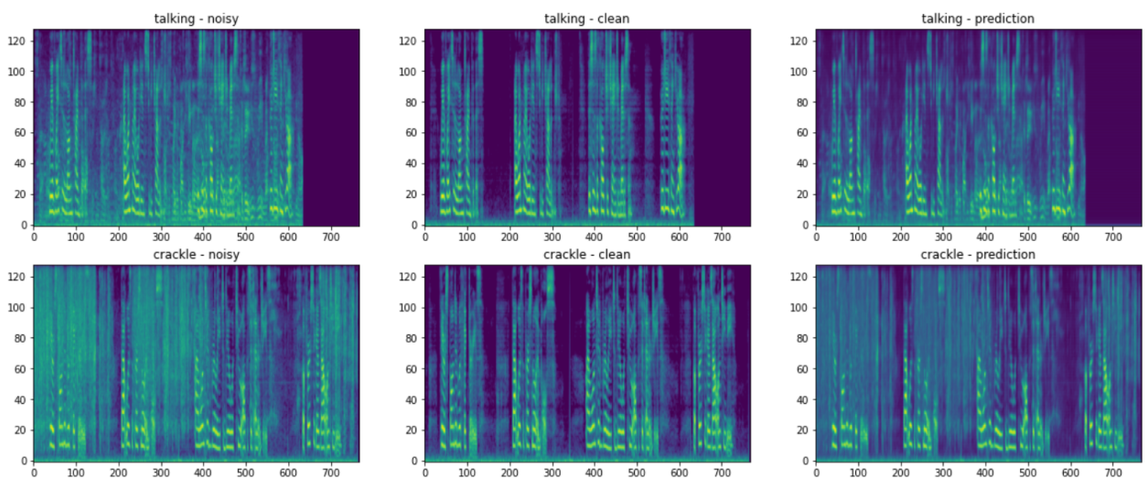

Figure 6: Mean Squared Error, Learning Rate = 0.0006 - Source: screenshot of our model output

5.3 Influence of Dropout Layers

In the next step, we added some dropout layers to our model, to see whether this would improve its performance. As the noise of the inputs is so varied, it is vital that the model does not start to rely too much on specific features. We started with a dropout rate of 0.2, which also proved to be the ideal dropout rate in later experiments. While the loss did not change much, adding the dropout layers did improve the output prediction, likely because it helps the model to generalize better. The output is less fuzzy, but unfortunately also quieter.

5.4 Optimizers and Loss Function

Just to be certain that Adam really is the ideal optimizer for our model, we let it run with both RMSprop and Adagrad. As expected, both showed a significantly worse output, especially Adagrad, which you can see in Figure 7 and Table 1. We continued using Adam afterwards.

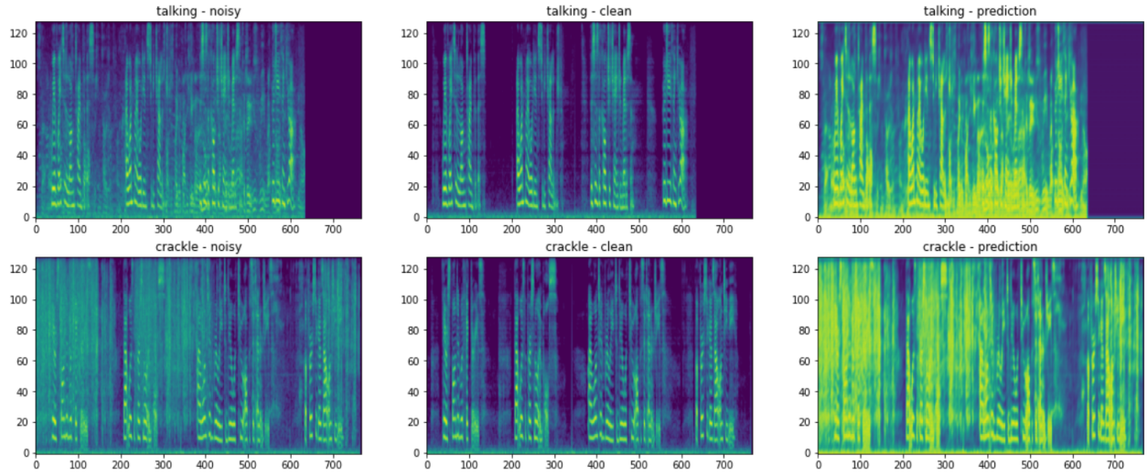

Figure 7: Mean Squared Error, Optimizer = Adagrad - Source: screenshot of our model output

Optimizer	**Training Loss**	**Validation Loss**
Adam	0.0909	0.0858
RMSprop	0.1051	0.1364
Adagrad	0.1987	0.2036

Table 1: comparison of loss when using different optimizers

After the optimizers, we focused on finding the best loss function for our problem. First, we implemented a custom noise-to-signal ratio function, as we believed this might be well-fitted for denoising an image. But while the loss seemed promising, even after 20 epochs the output itself was still very noisy, as you can see in Figure 8.

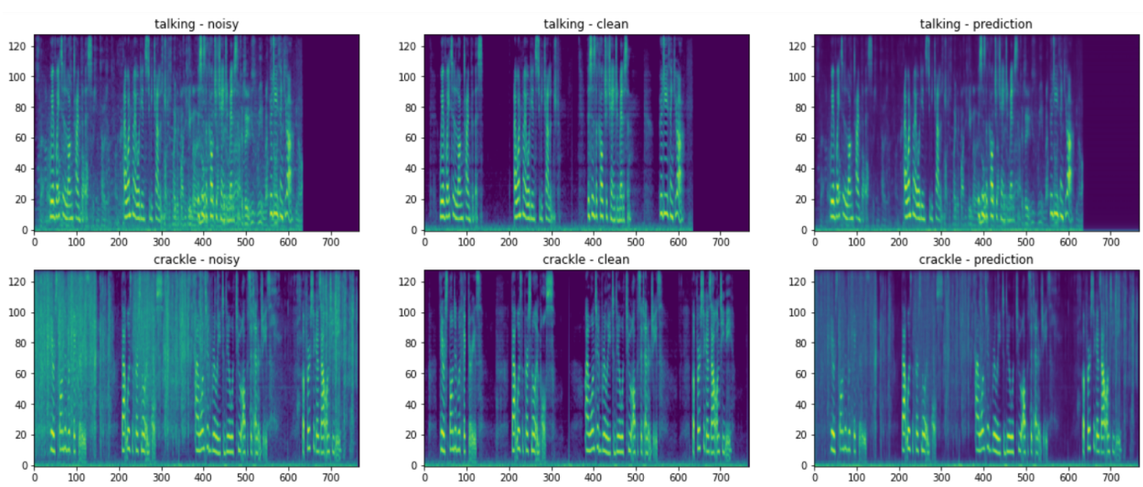

Figure 8: Noise-to-signal loss function - Source: screenshot of our model output

Then we tried a custom signal-to-noise ratio function, which resulted in a much better output. Still, as it was supposed to be both more stable and computationally efficient, and to exhaust all the options, we also tried using Mean Absolute Error. Figure 9 shows the changes in loss for each of the different functions. MAE was a significant improvement compared to all previous options, and led to our best output yet, as you can also see in Figure 10. This makes sense, as our dataset has rather little variance between batches but many outliers, and MSE penalizes large deviations more heavily than small ones (relatively). Just to be

certain, we also tried both lower and higher learning rates again, but 0.0006 still remained the most effective.

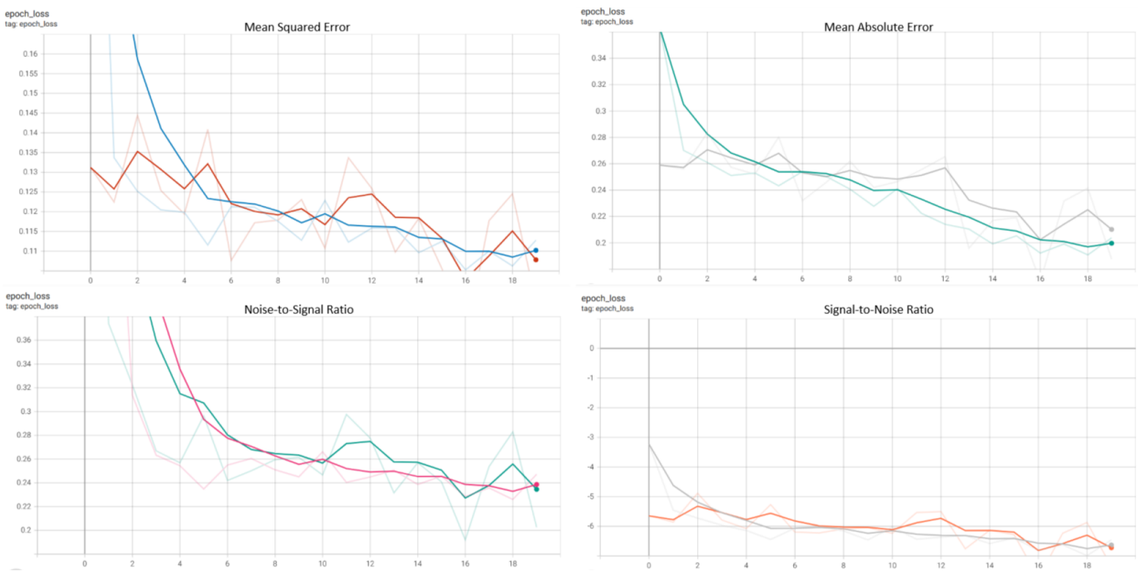

Figure 9: Plots for four different loss functions. Note that due to the different loss ranges the scaling is different - Source: screenshot of tensorboard showing our model output

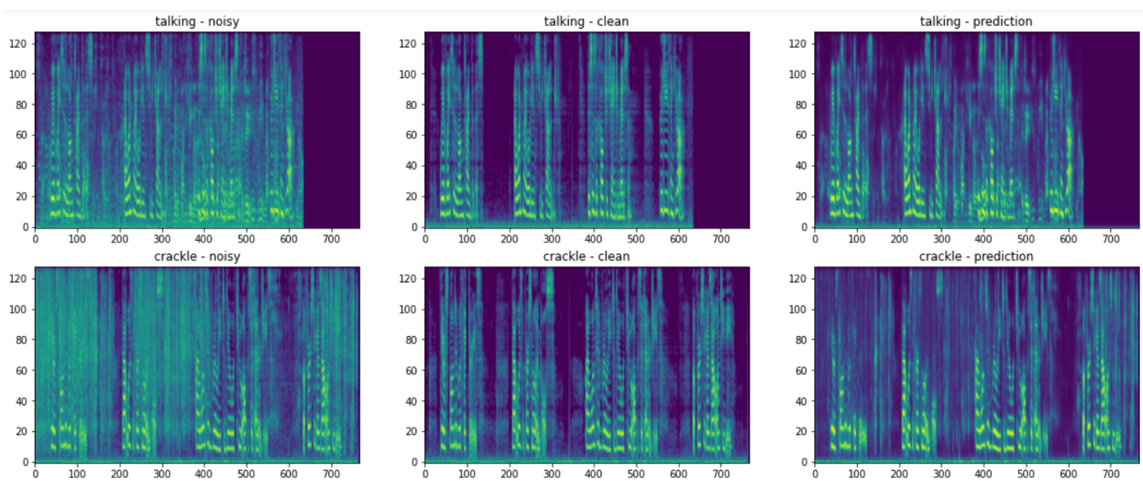

Figure 10: Mean Absolute Error, Learning Rate = 0.0006 - Source: screenshot of our model output

Next, we wanted to optimize our batch size. We had started with a batch size of 16, as this was used in the previously mentioned paper by Nustede and Anemüller, 2021, but now we wanted to make sure that this really was the optimal batch size for our model. Increasing the batch size had rather unfortunate results, but a reduced batch size actually contributed to a cleaner output. It is likely that, due to the large diversity between the different kinds of noise, it is an advantage to keep some amount of randomness between batches so the model does not start to overfit. Figure 11 shows the output spectrogram with a batch size of eight.

As the difference at introduction was only so slight, we now tried running the model with these parameters but without the dropout layers. The loss improved a little, as would be expected since now all units are used in training. However, there was not much of a discernible difference when listening to the audio or looking at the spectrograms, as can be seen in Figure 12. If anything, it was a bit worse and more fuzzy than before.

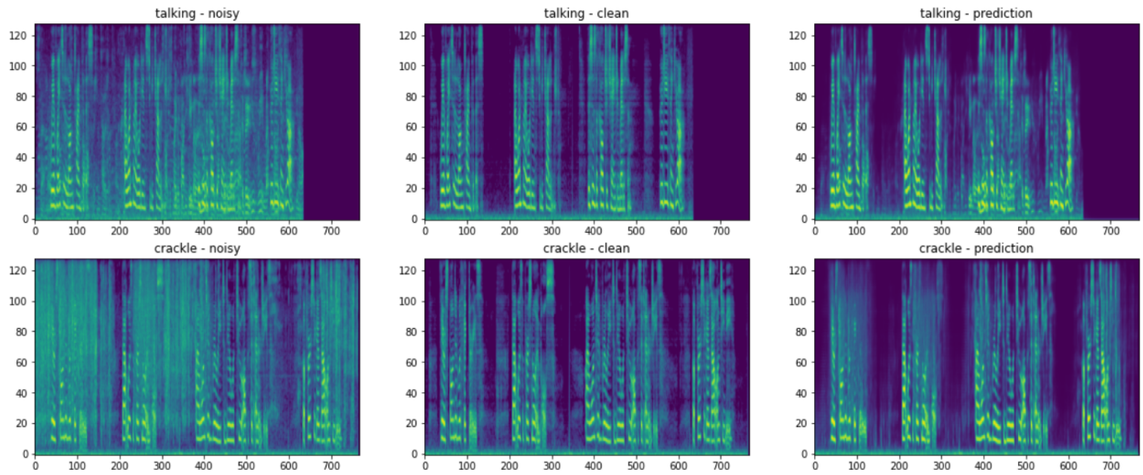

Figure 11: Mean Absolute Error, Learning Rate = 0.0006, Batch Size = 8 - Source: screenshot of our model output

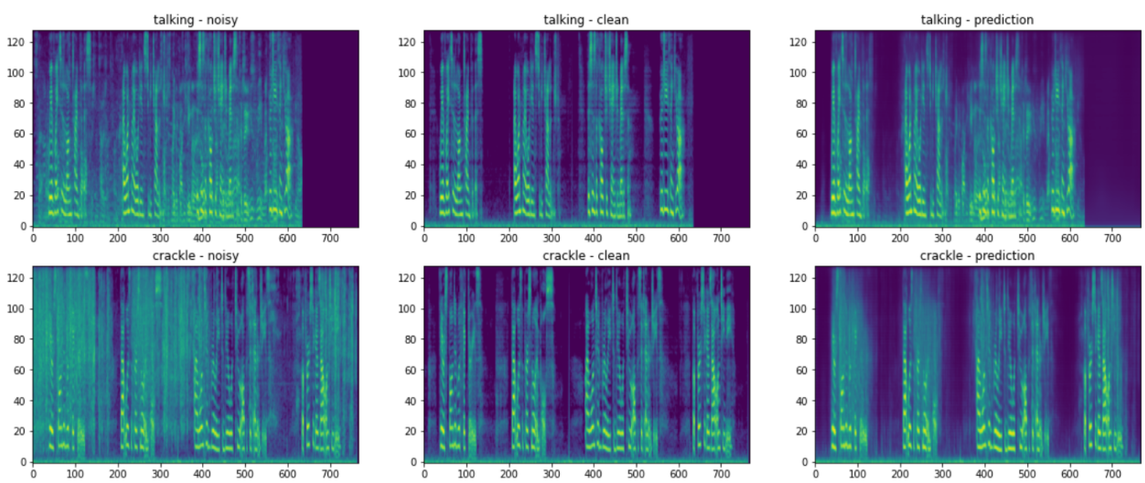

Figure 12: Mean Absolute Error, Learning Rate = 0.0006, Batch Size = 8, no Dropout - Source: screenshot of our model output

To make sure that we were still using the ideal amount of epochs for the current parameters, we tried increasing the epochs of the model with the current parameters to 30. But as in the beginning, there was very little to no improvement after the 20th epoch.

5.5 Increasing the Receptive Field

To test if an enlargement of the receptive field of a single neuron would help our network to gain more contextual knowledge, we tested different dilation rates. We started with an increasing dilation rate in the encoder, as is usually chosen in audio-denoising research (like in the paper that our architecture is based on). Unfortunately, the prediction was very noisy. This result made sense to us: If we consider that our spectrogram is sampled down from (128x768) to (2x12), we lose information in the deeper layers of the encoder. The 8th block already gets a (2x12)-input, and we had a 4,4-dilation in the convolutional layer there. With that dilation rate and a kernel size of 3x3, we would need six zeroes as padding in the left-right direction to ensure that the output feature map has the same spatial dimensions as the input. Thus, we would blur our information through a lot of padding.

After recognizing that, we tried to use the dilation earlier, in the second encoder block. The output still sounded very noisy. It is important to keep in mind that when using dilations, we trade resolution for receptive field size, which is often desirable. In our case, however, increasing the receptive field did not yield good results. This could have two explanations:

1. Our feature maps are quite small, so we already have both a low time resolution and a low frequency resolution.
2. Our network does not need to be sensitive to such a large area, as we have quite constant noise and just one speaker for each audio frame.

5.6 Unknown Data

Finally, as we wanted to see how well our model generalizes to unfamiliar data, we created a small dataset to run through the trained model. First, we added some of the known noise from the MS-SNSD dataset to audio clips in Arabic and Chinese. The audio in Arabic sounded pretty good, but there was some static noise in the background, which was different from the noise before. With the audio in Chinese, this was even worse. We then tried to use an unknown speaker and added both simple and more complex background noise, which we created ourselves. The model performed fine on the simple audio, even if it was a little quieter, but on the complex audio some of the more sudden noise remained. We also created audio with multiple speakers and added familiar noise, and the noise we made ourselves. The familiar noise was denoised well, but the model took away too much of the speakers' voices. Regarding the noise we created ourselves, the U-Net actually performed well on the easy noise, but very badly on the more complex one.

It seems, that the model overfits to the noise more so than it does to the speakers' voices or the language. Our architecture also had difficulty with multiple speakers, but as that is a well-known issue with audio denoising algorithms, it was unsurprising to us.

6 Discussion

In general, we would claim, that U-Nets work quite well for denoising audio, even with complex data and a limited dataset. Our output was quite clean and distinct, just the audio quality suffered a bit and it performed worse with unseen data.

There are some obvious improvements that could be made to our model and preprocessing. One clear issue we had, was the RAM-limitations. With better hardware, it would be possible to train the model on larger spectrograms. This would be interesting, both as there would be more information for the model to work with and potentially we could properly reconstruct the phase. Additionally, a clear, loss-free output, would also make it possible to fine-tune

the model more reliably.

Another improvement would be to increase the size of the dataset. This might properly challenge the complexity of the model. It might also be interesting to train the model on a dataset with even more challenging noise. While we do have background speech as noise in our dataset, which is considered quite challenging, we only use stationary audio. An improvement to our dataset might be to add stereo audio and experiment with reverberation as added noise. Testing how our network with unknown data, the most challenging for the network seemed to be unknown noise. This indicates that our network overfits to our noise. So it would be especially interesting to use a larger, more diverse noise dataset.

An unsupervised learning scheme would also lead to more diverse noise and additionally realistic training data. In unsupervised learning we could also profit from real-life effects like the Lombard-effect which describes that people tend to talk louder when it is noisy (Zollinger and Brumm, 2011). It would also be good to use unseen noise for our validation set, this would probably also increase the confusingly low validation loss. Our network performance also got slightly worse with multiple speakers which is not a problem when just being used for audio book denoising but would make it less applicable for other tasks.

References

Ali, Mohamed Nabih, Alessio Brutti, and Daniele Falavigna (2020). "Speech enhancement using dilated wave-u-net: an experimental analysis". In: *2020 27th Conference of Open Innovations Association (FRUCT)*. IEEE, pp. 3–9.

Cryer, Jonathan D (1986). *Time series analysis*. Vol. 286. Duxbury Press Boston.

Grzywalski, Tomasz and Szymon Drgas (2018). "Application of recurrent U-net architecture to speech enhancement". In: *2018 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*. IEEE, pp. 82–87.

Kang, Zheng, Zhihua Huang, and Chenhua Lu (2022). "Speech Enhancement Using U-Net with Compressed Sensing". In: *Applied Sciences* 12.9, p. 4161.

Librosa-development-team (2023). *denoising-audio-signal*. accessed 20-March-2023. URL: `https://librosa.org/doc/main/generated/librosa.griffinlim.html`.

Macartney, Craig and Tillman Weyde (2018). "Improved speech enhancement with the wave-u-net". In: *arXiv preprint arXiv:1811.11307*.

McFee, Brian et al. (2020). *librosa: 0.8.1*. URL: `https://doi.org/10.5281/zenodo.592898`.

Moliner, Eloi and Vesa Välimäki (2022). "A two-stage u-net for high-fidelity denoising of historical recordings". In: *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 841–845.

Negahbani, Farzin et al. (Apr. 2021). "PathoNet introduced as a deep neural network backend for evaluation of Ki-67 and tumor-infiltrating lymphocytes in breast cancer". In: *Scientific Reports* 11.

Nossier, Soha A et al. (2020). "An experimental analysis of deep learning architectures for supervised speech enhancement". In: *Electronics* 10.1, p. 17.

Nustede, Eike J and Jörn Anemüller (2021). "Towards speech enhancement using a variational U-Net architecture". In: *2021 29th European Signal Processing Conference (EUSIPCO)*. IEEE, pp. 481–485.

Panayotov, Vassil et al. (2023). *LibriSpeech ASR corpus*. accessed 20-March-2023. URL: `http://www.openslr.org/12`.

Reddy, Chandan KA et al. (2019). "A Scalable Noisy Speech Dataset and Online Subjective Test Framework". In: *Proc. Interspeech 2019*, pp. 1816–1820.

Roberts, Leland (2023). *Understanding the Mel Spectrogram*. accessed 20-March-2023]. URL: `https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram-fca2afa2ce53`.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-net: Convolutional networks for biomedical image segmentation". In: *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer, pp. 234–241.

Stoller, Daniel, Sebastian Ewert, and Simon Dixon (2018). "Wave-u-net: A multi-scale neural network for end-to-end audio source separation". In: *arXiv preprint arXiv:1806.03185*.

tanishkaa31 (2023). *denoising-audio-signal*. accessed 20-March-2023. URL: https://github.com/tanishkaa31/denoising-audio-signal.

Wang, Dequan and Jae Lim (1982). "The unimportance of phase in speech enhancement". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 30.4, pp. 679–681.

Xing Luo, Oscar (2019). *Deep Learning for Speech Enhancement: A Study on WaveNet, GANs and General CNN-RNN Architectures*.

Yuliani, Asri Rizki et al. (2021). "Speech Enhancement Using Deep Learning Methods: A Review". In: *Jurnal Elektronika dan Telekomunikasi* 21.1, pp. 19–26.

Zollinger, Sue Anne and Henrik Brumm (2011). "The lombard effect". In: *Current Biology* 21.16, R614–R615.