IANNWTF Final Project report

Group members:

- Niklas Bub, 977216 (nbub@uni-osnabrueck.de)
- Tim Bax, 979026 (tbax@uni-osnabrueck.de)
- Jasper Oldach, 982481 (Joldach@uni-osnabrueck.de)

Tutors involved with this project:

- Leon Schmid
- Matthis Pink

Link to Github repository:

- https://github.com/ElBuberino/Tensorflow_Final_Project_Beatbox

Note:

- This report describes and analyzes two tasks: An audio-image segmentation task and an audio-image classification task, with the former being our main project idea. As our model for this task had some unresolvable issues and we still wanted to have some presentable data results, we performed a simpler classification task described below.

Usage of project for educational purposes:

- We consent to the usage of this project for educational purposes of the IANNwTF course.

Meeting summary:

- Unfortunately, we forgot to take notes during the many meetings we had. But it would be a 20-page document considering the amount of meetings, so maybe it is for the best.

Project:

Audio-Image Segmentation Task of Amateur Vocal Percussion Dataset

Motivation:

The idea for the project originated from a general interest in music. Although music is a widely appreciated phenomenon that almost everyone interacts with on a regular basis in some shape or form, the barrier to entry for creating music is still a substantial obstacle.

Instead of having to do a deep dive to understand various music production programs, we wanted something that allows users to create quickly and without a lot of background knowledge. This would not only be beneficial for novices but also for musical professionals who want to quickly capture an idea. This led us to the idea of an automatic transcription application where the user would be able to vocally produce the beats and melody lines of a song, which would then be converted to a digital audio file that could be played by different electronic instruments.

For this project, we decided to focus on the beat and rhythm aspect of the application, i.e., training a model to recognize and classify beat-boxing audio files, notating the type of drum played, as well as the onset of the beat.

We thought that this would be a good choice for our final project as we would be able to apply what we learnt in the course. Additionally, we would learn how to use audio data and begin to understand what is possible in the field of music information retrieval (MIR).

Related Literature

Automatic transcription from audio to MIDI is a well-recognized task in the field of Music Information Retrieval (MIR), and vocal percussion transcription is a subtask that has been of interest to researchers for the past two decades.

There have been numerous studies in this area that have utilized various methods to detect and classify different sound events. For instance, Hazan (2005) proposed a three-part approach that involved detecting onsets, extracting relevant feature information, and using a KNN approach for drum beat classification. Gillet and Richard (2004) utilized hidden Markov models and support vector machines to transcribe drum loops, while Tanghe, Degroeve, and Baets (2005) worked on real-

time transcription of drum events in polyphonic music using support vector machines. Sinyor et al. (2005) achieved remarkable results using decision trees, with a beatbox classification accuracy of 95%. Additionally, Miron, Davies, and Gouyon (2013) developed a drum transcription system that employed an energy-based onset detector followed by a KNN machine learning approach for sound event classification. Another approach that produced quite promising results outperforming the inbuilt function in Ableton Live (Music production software) also split the task into three components, onset detection, feature extraction and a KNN machine learning approach for classification (Ramires, 2017).

Popular music production software, such as FL Studio, Ableton Live, and Logic X, have an audio to MIDI function built-in, which can be useful for musicians and producers. However, these tools may struggle with novice beatboxing recordings. Therefore, it is essential to consider specialized approaches for accurate transcription.

In order for us to attempt this as our final project we needed a dataset, unfortunately many of the datasets that were used by the above mentioned papers were not made publicly available. Fortunately we found the AVP dataset which was made publicly available by Delgado et al. (2019).

To begin with our approach we needed to determine the most effective way to represent the audio data. We discovered that the fast Fourier transform is a commonly used technique in machine learning applications that work with audio data. By converting the wav file into a spectrogram representation, we were able to use convolutional layers in our network. After further research, we realized that the U-Net architecture can be useful as it is useful for performing object localization, allowing us to identify beat events in the spectrogram at precise locations (beat onsets).

Unfortunately, we encountered difficulties with this approach and decided to develop a new approach for the task. When researching other beat transcription approaches we found that many use three distinct stages: onset detection, feature extraction, and classification. With our first approach, we attempted to accomplish all three stages simultaneously. In our second approach we decided to simplify the task and focus solely on the classification of each beat event.

1. A. Hazan, “Towards automatic transcription of expressive oral percussive performances,” in Proceedings of the 10th International Conference on Intelligent User Interfaces, IUI ’05, (New York, NY, USA), pp. 296–298, ACM, 2005.
2. [23] O. Gillet and G. Richard, “Drum track transcription of polyphonic music using noise subspace projection,” in In Proc. of ISMIR, pp. 92–99, 2005

3. K. Tanghe, S. Degroeve, and B. D. Baets, “An algorithm for detecting and labeling drum events in polyphonic music,” in In Proc. of First Annual Music Information Retrieval Evaluation eXchange, pp. 11–15, 2005.
4. O. Gillet and G. Richard, “Automatic transcription of drum loops,” in 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing, vol. 4, pp. iv–269–iv–272 vol.4, May 2004.
5. E. Sinyor, C. Mckay, R. Fiebrink, D. Mcennis, and I. Fujinaga, “BEATBOX CLASSIFICATION USING ACE,” in Proceedings of the International Conference on Music Information Retrieval, pp. 672–675, 2005.
6. M. Miron, M. E. P. Davies, and F. Gouyon, “An open-source drum transcription system for pure data and max msp,” in 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 221–225, May 2013
7. A. Delgado, N. Xu, S. McDonald and M Sandler, “A New Dataset for Amateur Vocal Percussion Analysis” , In Audio Mostly (AM’19), https://dl.acm.org/doi/10.1145/3356590.3356844, 2019
8. A. F. S. Ramires, Automatic Transcription of Drums and Vocalised percussion, FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO, 2017.

Methods

Data and preprocessing

The AVP dataset is composed of 4873 sound events produced through vocal percussion, which were recorded by 28 individuals and annotated with four labels: kick drum, snare drum, closed hi-hat, and open hi-hat. Each participant contributed four files, with each file containing repetitions of vocal percussion sounds of the same class. Additionally, one file from each participant corresponds to a freestyle improvisation using these sounds. For each of the sound files stored in wav format, there is a corresponding CSV file containing information on the onset time and the vocal type (i.e closed hi-hat) (see Figure 1).

P1_Improvisation_Personal

0	kd	p	ə
0.307664399582766	hho	t	i
0.993378684582766	kd	p	ə
1.35328798158277	kd	p	ə
1.70195011358277	hho	t	i
2.43229024958277	kd	p	ə
2.77043083858277	kd	p	ə
3.12725623558277	sd	t	e
3.45687074858277	hho	t	x
3.79791383258277	hhc	tʃ	I
4.55111111158277	kd	p	ə
4.91827664358277	kd	p	ə
5.56698412658277	hho	t	i
6.24907029458277	kd	p	ə
6.59736961458277	kd	p	ə
6.96235827658277	hho	t	i
7.33133786858277	hhc	tʃ	x
7.67909297058277	hhc	tʃ	x

Figure: CSV-file for the corresponding improvisation file for participant one. 0: Onset times for the vocal percussion; kd: onset label.

Audio data is typically in the time domain, which means that it consists of amplitude values over time. However, for neural network processing, it is often beneficial to convert audio data into spectrograms, which are in the frequency domain. Spectrograms represent the distribution of frequencies present in the audio signal as a function of time, and they provide a much richer and more detailed representation of the audio data.

Converting audio data into spectrograms allows neural networks to take advantage of convolutional neural network (CNN) architectures, which are widely used for image recognition tasks. Spectrograms have a similar structure to images, and can therefore be fed into a CNN as input. This enables the neural network to learn and extract features from the audio data in a more effective way, leading to improved performance on audio-related tasks such as speech recognition, music classification, and sound event detection.

U-Net architecture

The network architecture used is called a ‘U-Net’, originally proposed by Ronneberger et al. (2015), for biomedical image segmentation. It is a convolutional neural network commonly used for multiclass image segmentation tasks that is characterized by its ‘U’ shape (see Figure 2). It is composed of encoder-decoder blocks and is connected by a bridge. Figure 1 depicts the network structure as

proposed by Ronneberger et al., which includes a contracting path (encoder) on the left and an expansive path (decoder) on the right. The contracting path adheres to the standard convolutional network architecture, featuring the repetition of two 3x3 convolutions (without padding), followed by a ReLU and 2x2 max pooling operation with a stride of 2 for downsampling. With each downsampling step, the number of feature channels is doubled while the image size is halved. Each step in the expansive path comprises an upsampling of the feature map, a 2x2 convolution (known as "up-convolution") that reduces the number of feature channels by half, a concatenation with the corresponding cropped feature map from the contracting path (skip connection), and two 3x3 convolutions with ReLU. Cropping is necessary to address the loss of border pixels during each convolution. The final layer employs a 1x1 convolution to map each 64-component feature vector to the desired number of classes. Overall, the network comprises 23 convolutional layers.

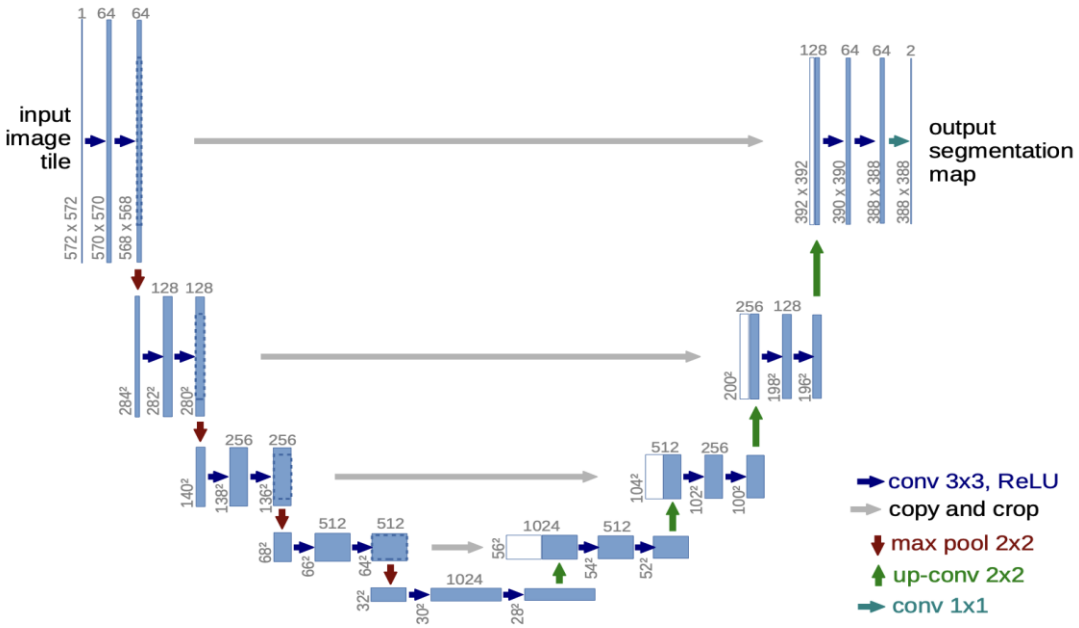

Figure: Example of a U-Net Architecture (image from Ronneberger et al., 2015, p. 235)

Implementation

Preprocessing

Input images

After collecting all sorted wav- and CSV files in two separate lists, we used the librosa package to read the wav files and convert them into spectrograms. As the audio files were of different length, the spectrograms had different sizes. In order to

have the same size for all of our input we used the largest file as a reference for the x-axis, fixing the size of each spectrogram to 128 x 2048. The resulting spectrograms were normalized to values between [0,1], by finding the minimum and maximum values and changing x according to:

```
x = (x - min_value) / (max_value - min_value)
```

As the dataset was very small we had to augment our data to naturally expand our dataset. Despite using a convolutional neural network, classical image augmentation techniques like tilting, blurring, etc. were not useful as we are working with audio data. Hence, we used the pitch_shift and add_noise function from the librosa package to alter our data. All the generated and original spectrograms were appended to an input list.

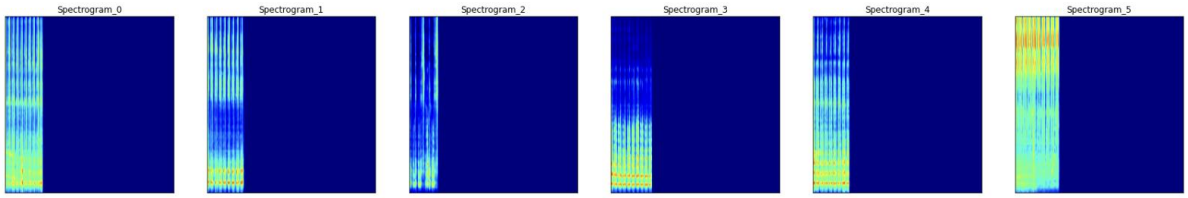

Figure: Visualization of five spectrograms, set to a fixed length

Targets

Our targets were four (for each category) 1-dimensional arrays of length 2048 (time length of the spectrogram), hence a (1 x 2048 x 4) matrix, where each onset time was marked as a 1 in the corresponding category channel. While files of the same class only contained target values other than 0 in one of the four category channels, the improvisation files could have onset targets in all of the four channels. This approach was possible as only one sound could be made at the same time in the dataset and consequently, there was no overlap in onset times. To create the targets, we used the CSV-file and converted it into a numpy array and saved the first row (onset times) and the second row (sound category). We created a dummy target of the desired shape (1 x 2048 x 4) and added the target values as described above and appended them to a target list.

Datasets

The input list and the target list were conververted to Tensorflow Datasets and zipped. After mapping the targets to float32 values, the dataset was shuffled, batched and prefetched. The resulting dataset had the shape `(Batch_size, 128, 2048, 1)` for the inputs and `(Batch_size, 1, 2048, 4)` for the targets.

Model

The employed model is the U-Net architecture described above. The encoder started with 8 filters for the original input and worked as explained in the abovementioned section. Only two differences to the original architecture were applied: We used ‘same’ instead of ‘valid’ padding in order to assure the right dimensionality for the concatenation with the decoder path and the corresponding skip connection and we added a batch normalization layer after every convolution in order to avoid overfitting. The result of each step in the contracting path were saved for a later skip connection (s) to be concatenated to the expanding path (see Figure 4). The filters in the decoder were doubled until the bridge with 128 filters before reducing them by half and increasing the image size in the decoder up to the original image size. A convolution with four filters (one for each category), kernel size (1,1) and a softmax activation function was used to label each pixel accordingly.

As we were just interested in the onset times and vocal category our targets were of shape (1 x 2048 x 4) instead of (128 x 2048 x 4). We consequently run a final convolution with kernel size (128,1) in order to crop the image to four 1-dimensional arrays. We trained our model using the ‘Adam’ optimizer and the ‘categorical_crossentropy’ loss function.

```python
def build_unet(input_shape):
  inputs = Input(input_shape)

  """Encoder"""
  s1, p1 = encoder_block(inputs, 8)
  s2, p2 = encoder_block(p1, 16)
  s3, p3 = encoder_block(p2, 32)
  s4, p4 = encoder_block(p3, 64)

  """Bridge"""

  b1 = conv_block(p4, 128)

  """Decoder"""

  d1 = decoder_block(b1, s4, 64)
  d2 = decoder_block(d1, s3, 32)
  d3 = decoder_block(d2, s2, 16)
  d4 = decoder_block(d3, s1, 8)

  """Output"""

  outputs = Conv2D(4, (1,1), padding = 'same', activation = 'softmax')(d4)

  outputs = Conv2D(4, (128,1), padding = 'valid', activation = 'sigmoid')(outputs)
  # outputs = AvgPool2D((128,1))(outputs)

  model = Model(inputs, outputs, name = 'U-Net')
  return model
```

Figure: U-Net architecture

Results

Sadly, we were not able to get a working model. We guess it has something to do with the loss-function, as this one did not properly compute the loss. After 10 epochs, the U-Net achieved a training accuracy of 1.19% and a training loss values of 0.0233. However, the first epoch started with a loss of 0.0236 and an accuracy of 65.12%, which indicates a massive error at some point. Sadly, we were not able to figure it out. We tried several different loss functions, but none yielded any useful results.

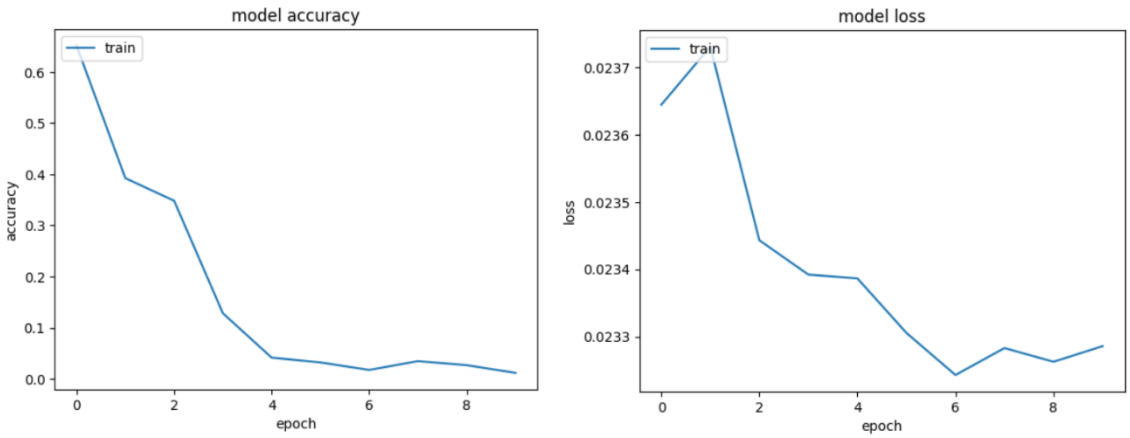

Figure: U-Net training accuracy and loss

As we could not figure out the error in the model, we did not even start to compare it to a validation loss/accuracy, as the loss-function itself was not working porperly.

At this point we decided to simplify the task to obtain at least some presentable results.

Image classification task

As the proposed network in the original audio-image segmentation task had some unresolvable issues regarding its learning, we decided to dissect the original spectrograms to get one image for each sound utterance and classify them. We produced one image x for each onset time(x) to the onset time(x+1).

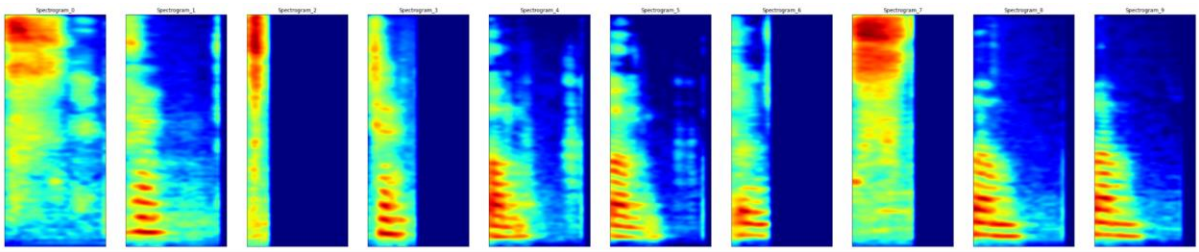

Figure: Visualization of spectrograms after splitting them into their respective instrument (split from onset to next onset)

By dissecting each utterance and using the librosa pitch shift function to augment our data we yielded a much bigger dataset of simpler sepctograms with almost 20.000 data points (see Figure). We created one-hot targets of length four, for our four voice categories. Other than that the preprocessing was similar to the preprocessing of the original task.

Model

We created two models, a DenseCNN and a BasicConv Net. The ConvNet consists of three Convolutional layers each followed by a BatchNormalization layer, a dropout layer and except for the last one a MaxPooling layer. For the convolutions same padding and relu activations were chosen, the kernel size decreased form 5 in the first layer to 3 in the following layers. After the final convolution a GlobalMaxPooling layer flattened the image array to be processed by a Dense Layer of depth 64 and again a relu activation function. An output layer of depth 4 (one for each category) and softmax activation, to assign each outcome a probability, is the final network output. The 102,148 trainable parameters were trained using the Adam optimizer and categorical crossentropy.

For the architecture of the DenseNet, we were heavily inspired by the DenseNet from the IANNwTF lecture (see Schmid, L. (2022)). Therefore, we also used our own training loop. The model is split nested into a DenseCNN model, and two DenseCNN layers. In the model, we defined three Conv-Blocks, and one final output layer. The blocks were then computed in the first CNN-layer, which performed a list comprehension. In the list comprehension, the actual convolutional layers were defined and calle. To reduce overfitting, we used regularization, dropout-layers, batch-normalization and weight initialization.

Results

The results of the audio-image segmentation task using the Convolutional Neural Network model (Conv Model) for the Amateur Vocal Percussion Dataset are presented in this section. The model architecture and training parameters are detailed in the methods section.

Conv-Net with data augmentation (n_augmented = 6):

Even after splitting up all the different onsets into spectrograms, the dataset was quite sparse. That’s why we decided to augment every spectrogram 6 times via pitch shifting, to get an absolute number of almost 20,000 data samples. Running these data thorugh the training loop, the Conv-Net achieved a training accuracy of 99.77% and a validation accuracy of 51.86% after 25 epochs. The training and validation loss values were 0.3836 and 1,4573 respectively, after the last epoch.

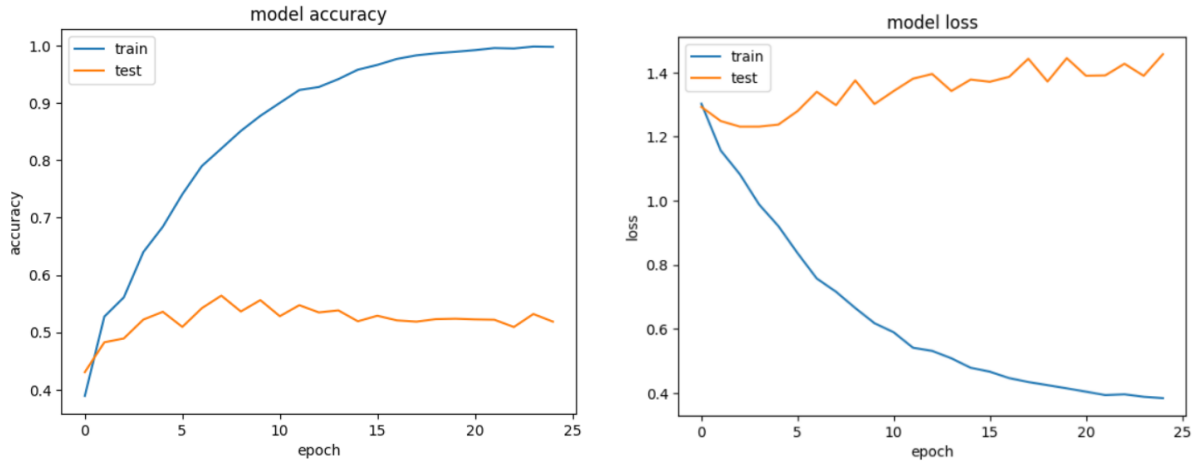

Figure: Conv-Net accuracy and and loss with data augmentation

Conv-Net without data augmentation:

As the model was overfitting so heavily, we decided to give it a try without agumentation, und just use the roughly 5000 data points we had. Surprisingly, the model performed way better with less given data. The Conv-Net achieved a training accuracy of 99.90% and a validation accuracy of 98.66% after 50 epochs. The training and validation loss values were 0.3646 and 0.4032 respectively, after the last epoch.

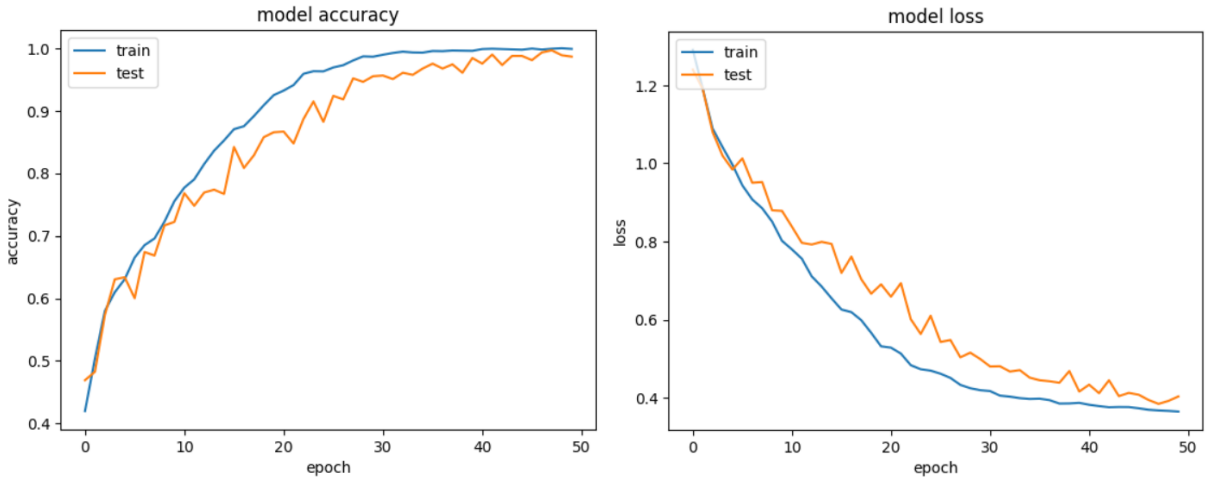

Figure: Conv-Net accuracy and loss without data augmentation

Dense-Net without data augmentation:

Compared to the Conv-Net, the Dense-Net performed way worse, and also took way longer to compute. While the Conv-Net took less than one minute for 50 epochs, the DenseNet took 16 minutes for training. After the last epoch, the Dense-Net achieved

a training accuracy of 84.45% and a validation accuracy of 63.38%. The training and validation loss were 0.4864 and 1.376, respectively.

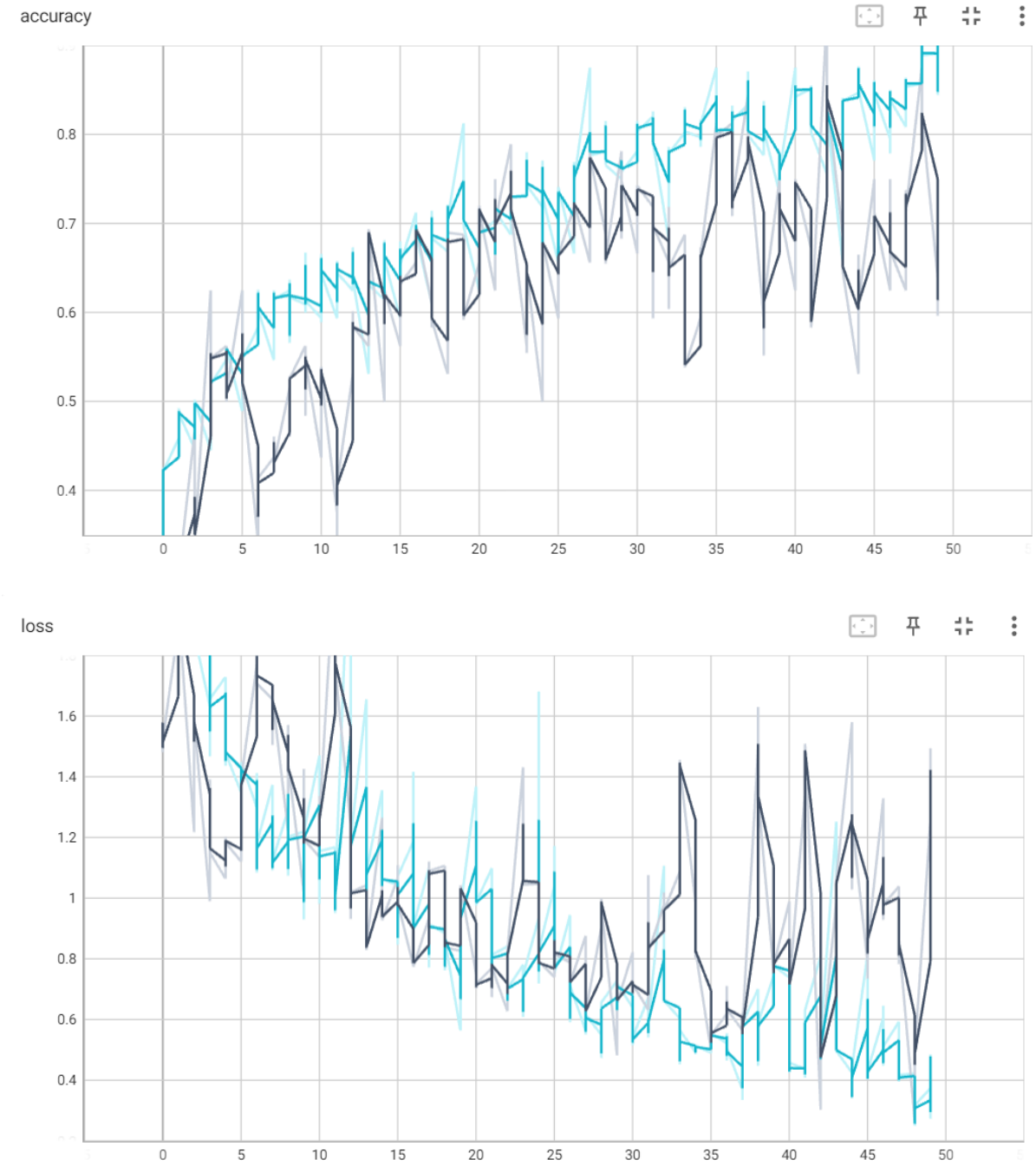

Figure: Dense-Net accuracy and loss without augmentation

Dense-Net with data augmentation (n_augmented = 6):

With data augmentation, the Dense-Net performed even worse, as it was overfitting even more than the Conv-Net. After the last epoch, the Dense-Net achieved a training accuracy of 91.11% and a validation accuracy of 38.43%. The training and validation loss were 0.2910 and 3.4312, respectively.

Discussion

One of the most significant findings of our project was the difficulty of image segmentation. Despite having a labeled dataset, we were unable to generate a functional network. Given that image segmentation was not discussed during the course, our project goal was ambitious, but we were not expecting it to be so challenging. The scarcity of the dataset we worked with was one of the primary issues we encountered. Therefore, we opted to use a U-Net instead of a Transformer model and had to research how to build this type of architecture. Although the U-Net architecture worked in the end, our model did not learn, which could have been related to the loss function, although we are uncertain.

We also discovered that creating a simple Conv-Net for image classification was easy after we were unable to get the U-Net working. In contrast to the U-Net, the Conv-Net performed well and quickly, without requiring many layers or complex forms of architecture. As the image classification task worked well with the amount of data, we still believe that the U-Net image segmentation task should work in principle. We were simply out of ideas at the end.

Comparing the results of two models in the image classification task, it was fascinating that the basic Conv-Net outperformed the Dense-Net. The speed of computation was unsurprisingly faster, but the significant improvement in performance was surprising. Our best hypothesis is that the Dense-Net's architecture has many more parameters, resulting in a greater risk of overfitting, which was the issue we encountered with the DenseNet. Another reason could be that the different classes in the split spectrograms were too simple to detect for the network, and the DenseNet's additional connections led to overfitting. The third reason could be that we did not tune the hyperparameters of the Dense-Net well enough, leading to worse performance than the basic Conv-Net.

Another surprising discovery was that data augmentation resulted in significant overfitting in both the basic Conv-Net and the Dense-Net. Perhaps we augmented too much data, resulting in the images being too similar to one another, causing the model to fail to generalize to new, unseen data. The fact that our general data pool was so small compared to our augmented data pool supports this hypothesis. Alternatively, the type of augmentation we used (pitch shifting) may not have been appropriate for the spectrograms, resulting in worse performance as the model struggled to generalize with all of these augmentations.

Both the differences in augmentation and model architecture were intriguing and unexpected and could be explored further. However, since we already have a highly performing image classification Conv-Net, it is not necessary. The primary focus for further research would be on getting the U-Net image segmentation working, perhaps even with the help of the image classification network we built as an

alternative. Having now classified the different instruments, we could integrate this information into the U-Net's preprocessing. This would undoubtedly be the next step for this network.

Bibliography

Ronneberger, O., Fischer, P., & Brox, T. (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 (pp. 234-241). Springer International Publishing. https://doi.org/10.48550/arXiv.1505.04597

Alejandro Delgado Luezas. (2021). AVP-LVT Vocal Percussion Dataset [Data set]. Zenodo. https://doi.org/10.5281/zenodo.5578744

Mohan, A. (2021, 12. Dezember). An Overview On U-net Architecture - DataDrivenInvestor. Medium. https://medium.datadriveninvestor.com/an-overview-on-u-net-architecture-d6caabf7caa4

API Documentation | TensorFlow v2.12.0. (o. D.). https://www.tensorflow.org/api_docs

ChatGPT. (o. D.). https://chat.openai.com/chat

librosa — librosa 0.10.0 documentation. (o. D.). https://librosa.org/doc/latest/index.html

Stack Overflow - Where Developers Learn, Share & Build Careers. (o. D.). https://stackoverflow.com/

Brownlee, J. (2022, 5. August). *Display Deep Learning Model Training History in Keras*. MachineLearningMastery.com. https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/

Idiot Developer. (2019, 2. Februar). *UNET Segmentation in Keras TensorFlow | Semantic Segmentation | Deep Learning* [Video]. YouTube. https://www.youtube.com/watch?v=M3EZS__Z_XE

Schmid, L. (2022). *Universität Osnabrück - Universität Osnabrück*. Universität Osnabrück: StudIP - IANNwTF courseware. https://studip.uni-osnabrueck.de/dispatch.php/course/courseware/?cid=9a82df906557ec20607b94edba3c5ff6#/structural_element/139750