Deep Learning - Mini Project AVS8

The report contains part 1 and 2 of the Deep Learning Mini Project from Aalborg University. The mini project focuses on the COVID-19 Audio Classification Dataset: https://www.kaggle.com/datasets/andrewmvd/covid19-cough-audio-classification/code

To view the code: https://www.kaggle.com/code/simonnnb/audio-classification-resnet?kernelSessionId=170493426

Simon Baks, Jasa Basl, Alexander Nielsen

Part 1 - Exploration of the dataset

How many samples does the dataset contain?

27550 total samples including non-classified samples, however only 16244 of them are labeled.

What does the class distribution look like?

The dataset consists of three classes: healthy, symptomatic, and COVID-19.

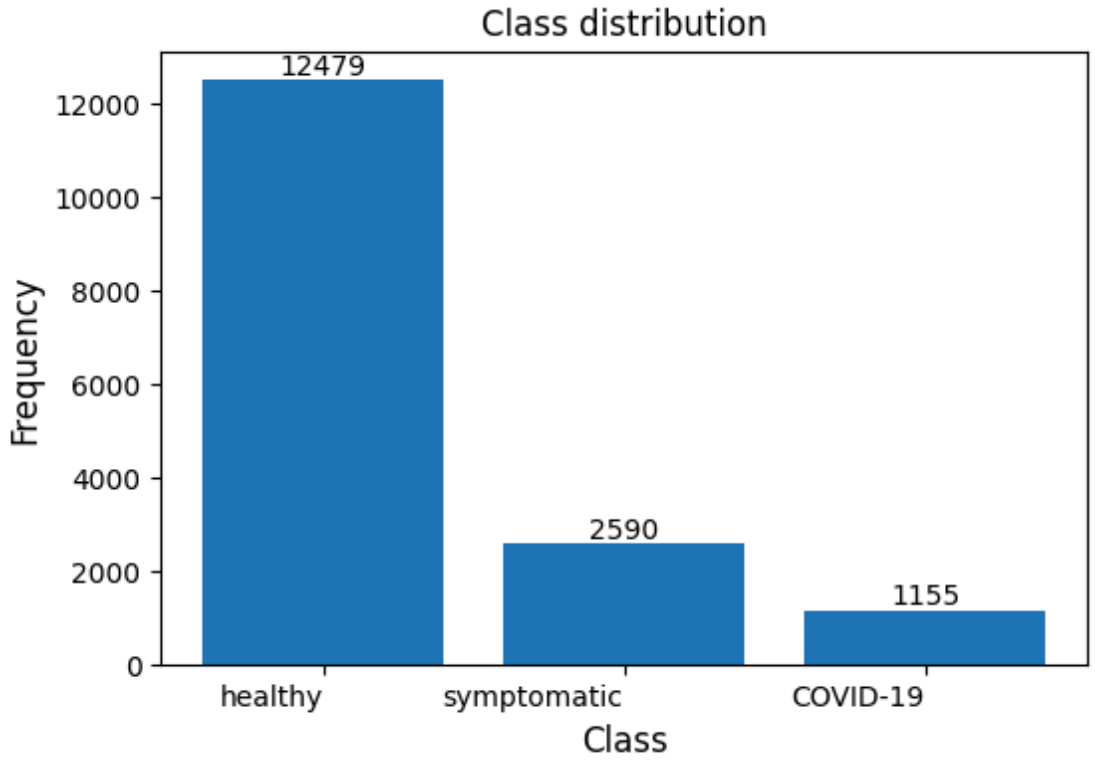


Figure 1. Class distribution

Playing random audio samples

After playing a few samples we observed that the cough_detected, is most likely the output from some cough detection model. Where the value represents the confidence of a cough being in the cough recording.

There is somewhat a clear distinction between the healthy coughs. A "forced healthy cough" is definitely more distinct than the symptomatic and COVID-19 coughs.

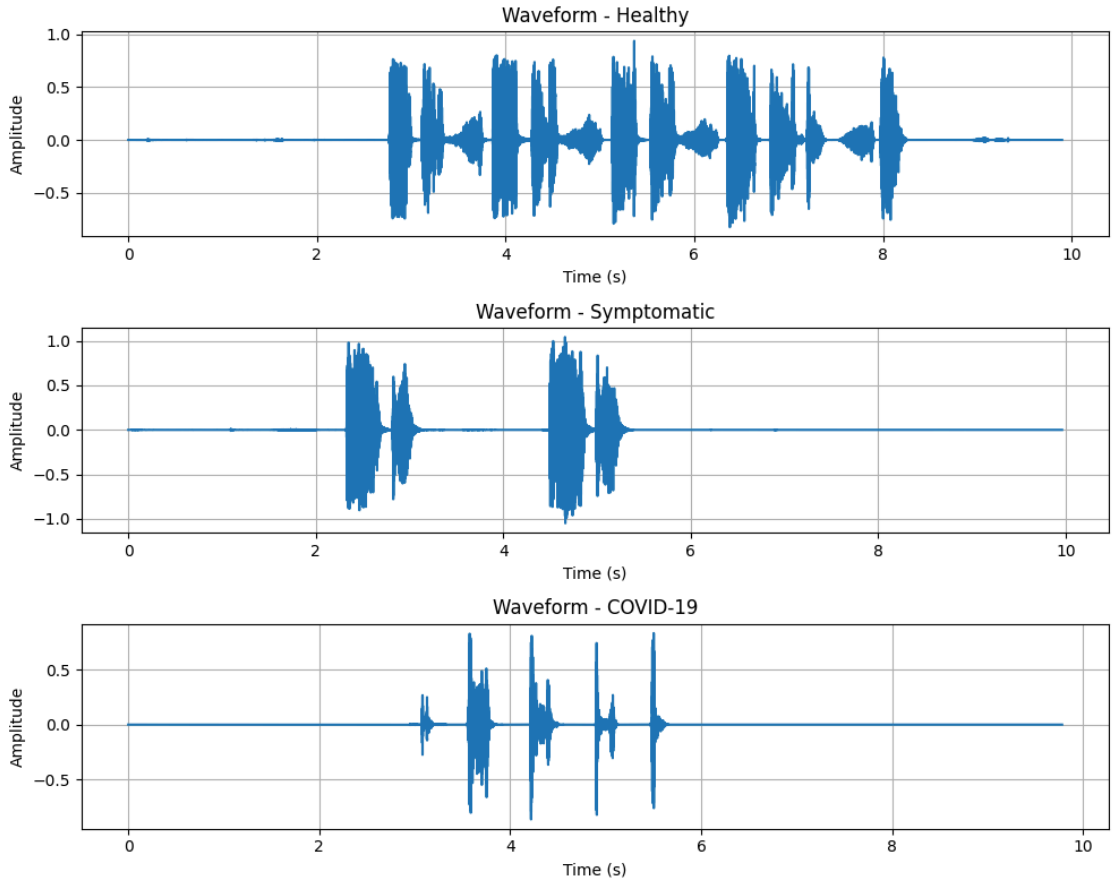


Figure 2. Original waveforms from each class.

Data distribution's effect on classifier

The classifier will most likely be biased towards the healthy class, due to the amount of healthy data compared to the others. It would probably be better to undersample the healthy samples to somewhat match the number of symptomatic and COVID-19 samples. Maybe even trying to oversample symptomatic and COVID-19 by creating synthetic data, could also be a possibility.

Another approach to try would be training the model with weights. Pytorch has a handy function called WeightedRandomSampler, which sample elements with given probabilities, which means we could put more emphasis on the minority classes. This results in the random sampler selecting random samples based on these probabilities.

Initial feature selection

It may not make much sense to use the samples with a ‘cough_detected’ below a certain threshold, as they don’t actually include a cough in those and only consist of background noise. Likewise the samples that don’t have a ‘status’/class appointed to them can’t be used. In our case we simply chose a threshold of 80% so all the samples less than 80% are removed. Additionally as seen from the waveforms above, there is a lot of silence in the audio samples which we decided to remove as it may introduce noise when training.

It may also be interesting to keep the age, gender, and SNR features, which may have some relation between the coughs and whether they are healthy, symptomatic, or have COVID-19. However, it may be beneficial to remove some of the outliers within the data such as a low number of samples of the ages and the “other” gender. As such it was decided that for the ages, samples fewer than a 100 would be removed as well as the "other" gender.

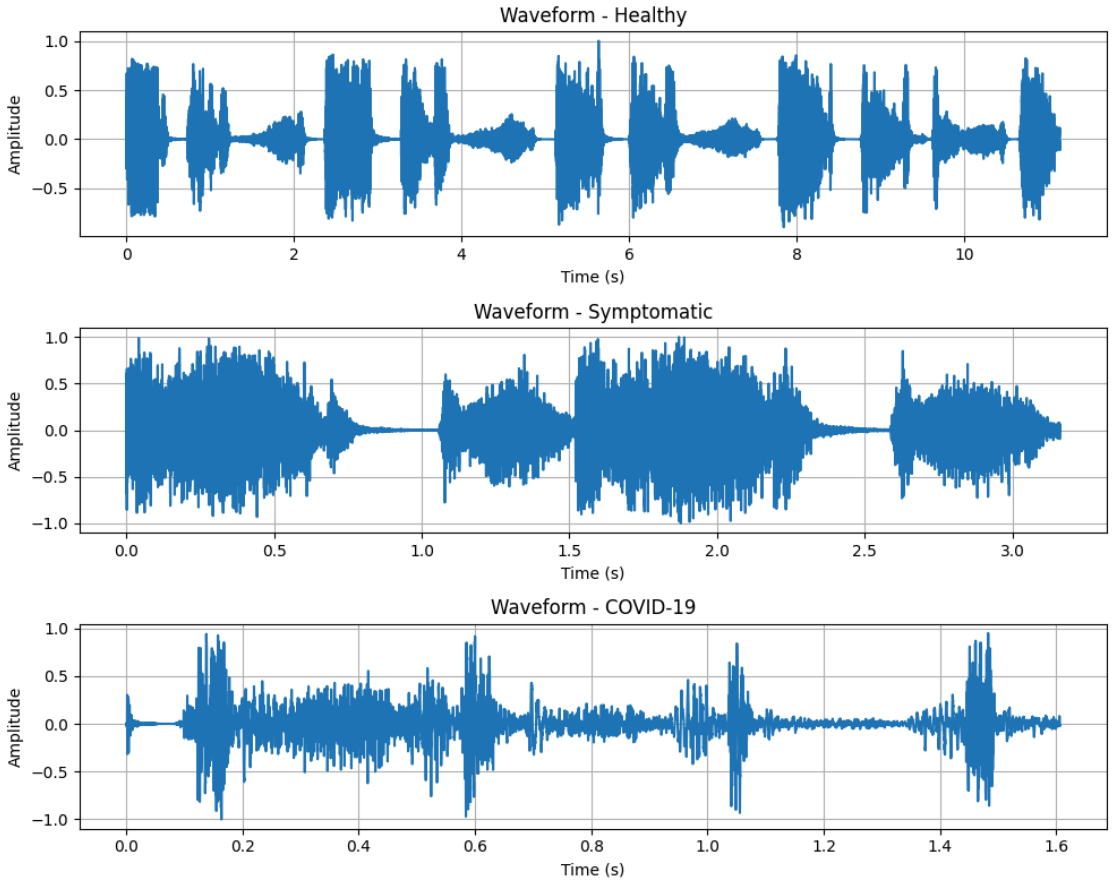


Figure 3. Filtered waveforms with silence removed.

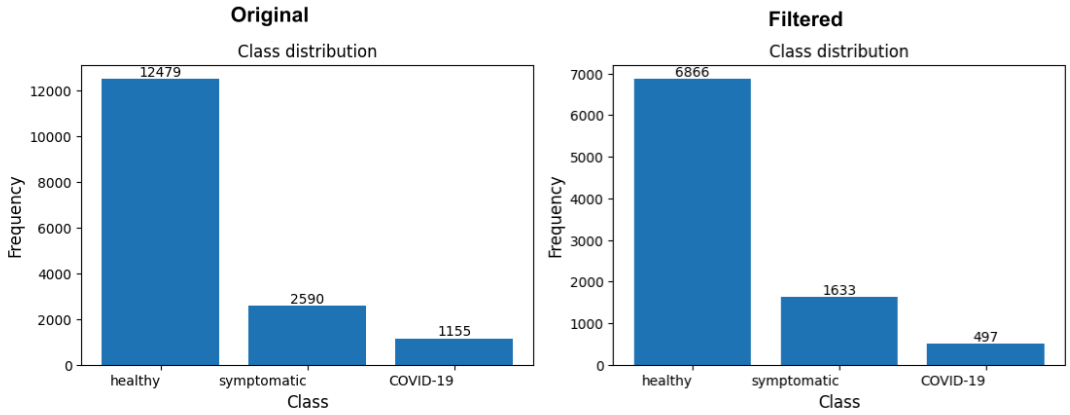


Figure 4. Original and filtered class distribution.

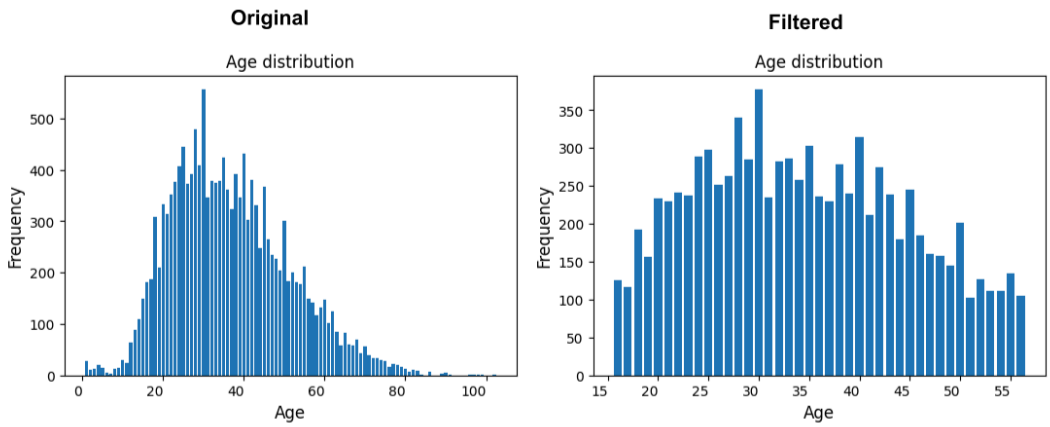


Figure 5. Original and filtered age distribution.

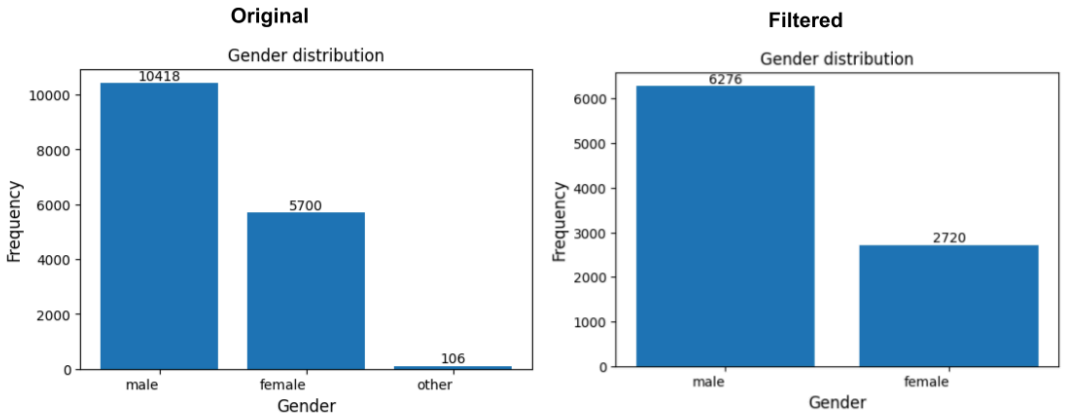


Figure 6. Original and filtered gender distribution.

Part 2 - Training a neural network

For part 2 of the mini project the following network architectures were chosen arbitrarily: the two residual convolutional networks ResNet18, ResNet34, and the audio spectrogram transformer network ASTModel.

The ASTModel is based on the transformer architecture which is based on the vision transformer. This model purely utilizes the attention mechanism and no convolutions or convolution-like methods.

Original paper : https://arxiv.org/abs/2104.01778

Relevant hyper-parameters

- **Learning rate:**

Learning rate determines the size of the step taken during the optimization process.

A higher learning rate leads to faster convergence, since larger steps are taken in the parameter space. However, it can potentially cause divergence if the steps are too large. Lower learning rate can on the

other hand provide more precise updates to the model parameters, but can result in slower convergence.

If the learning rate is too high the optimization algorithm might oscillate around the minimum or even diverge. If the learning rate is too low, the optimization might get stuck in local minimum.

Additionally, a too high of a learning rate may also cause *exploding gradients* which is a term that is used when large error gradients accumulate and result in very large updates to the network model weights during training. These large updates can in turn result in an unstable network. In extreme cases the value of the weights can become so large that it overflows and result in NaN (Not-a-Number) values. If the extreme case occurs it means that the network cannot learn from the training data and results in NaN weight values that can no longer be updated.

In some cases it could be beneficial to adjust the learning rate during training. Techniques like rate decay (the learning rate decreases over time).

Different learning rates must also be considered for different types of data, model architectures and the complexity of the problem. For example, noisy or sparse data may require more careful tuning of the learning rates. Deeper networks or networks with complex structures might benefit from smaller learning rates to avoid overshooting or divergence. More complex problems might require finer adjustments to the learning rate.

- Batch size:

Batch size refers to a subset of the entire dataset. This subset is limited by either available memory or by design. The batch size may affect the outcome of training as the resulting loss of each sample is summed together across the batch. With this summed up loss the backwards pass is then performed which adjusts all the parameters. As batches are mostly picked at random from a shuffled dataset it is possible for a batch to contain only 1 class or mostly 1 class. This may result in training that tends towards overfitting to single classes

or create unbalanced inference later on. This means a small batch size may decrease the ability for generalization in the end. Opposite this, the batch size is also limited by available memory as an entire batch is loaded into memory at once resulting in large input samples that may limit the possible batch size (mini-batch gradient descent).

- Epochs:

The number of epochs affects the quality and speed of the training process. If we use too few epochs, the neural network may not learn enough from the data and underfit (it will perform badly on both training and the test data). If we use too many epochs, the NN may overfit, meaning that it will memorize the training data and not be able to generalize to new unseen data. How many epochs one should use depends on size and complexity of data, the architecture and capacity of the model, the learning rate and optimization algorithm, the regularization and dropout techniques.

One common way of finding the right number of epochs is using a validation set (subset of training data that is not used for updating the model's parameters). Validation loss and accuracy can be observed over time, which shows learning of the model and when it starts to overfit. Usually the training is stopped when the validation loss stops decreasing and starts increasing, which indicates that the model has reached its peak performance and further training is no longer needed. This technique is called early stopping.

Another technique is to use a learning curve. Learning curve shows how the model behaves during the training process and how it converges to the stable state. A good learning curve shows a steady decrease in the training and validation loss and steady increase in the training and validation accuracy, until they reach a plateau. If the learning curve shows large differences between the training and validation metrics, or a quick increase in the validation loss, it means that the model is overfitting.

But one of the best ways to choose the number of epochs is to experiment with different values and compare the results. Start with a small number of epochs and gradually increase the number (be

careful with experimenting, because it can be time-consuming and computationally expensive).

- Dropout rate:

Drop out rate is a value, usually between 0.1 and 0.5 that describes the chance for a parameter in a model to be set to 0. This is a regularization method to help avoid overfitting by reducing the reliance on a few singular parameters throughout the model. This helps distribute the contribution from more parameters resulting in better generalization. Even with this parameter's ability to help generalize and reduce overfitting it does have consequences by setting it too high. With too high a drop out rate you are essentially training multiple submodels within the same model meaning that by training on a subset of the parameters you may experience that the model suddenly is unable to learn more complex relations and patterns within the training data. A high dropout rate will also result in a slower convergence resulting in the need for more epochs or a generally larger training set as some parameters are only exposed to a subset of the training set as they sometimes are forced to become 0.

- Train-test split ratio:

The split ratio between training and test data is an important hyper-parameter to set properly as more training data results in less test data which both needs to be a proper size depending on application. Generally too much training data is not problem but forcing more training data by reducing the amount of test data in the total set means that less data is available to evaluate the models performance properly. On the other hand too much test data giving a strong evaluation of the models performance risks having too little training data meaning the model might being underfit and not being able to predict anything meaningful from the data provided. An often acceptable split is 70%-80% training data with the rest going to test data. This split might however change a lot depending on application and the amount of available data.

Experimentation with batch size, learning rates, and network layers

In the table below the different types of hyper-parameters experimented with can be seen.

ID	Architecture	Batch size	Learning rate
Alexander	ResNet34	16	0.001
Alexander	ResNet18	8	0.01
Alexander	ResNet34	16	0.01
Alexander	Resnet18	16	0.01
Jasa	ResNet18	8	0.001
Jasa	ResNet18	16	0.001
Jasa	ResNet34	16	0.0001
Jasa	ResNet18	16	0.0001
Simon	ResNet34	8	0.01
Simon	ResNet34	8	0.001
Simon	ResNet34	8	0.0001
Simon	ResNet18	8	0.0001

Table 1. Experiments with different hyper-parameters.

Data augmentation techniques

For data augmentation we decided to perform spectrogram augmentation as specified in the paper: "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition" - https://arxiv.org/abs/1904.08779

In the paper they propose that instead of performing data augmentation on the raw audio, performing it on spectrograms instead could yield better performance. The spectrogram augmentation techniques consist of a series of techniques: time warping, frequency and time masking, and is applied to the spectrogram before its fed into the network.

First the spectrograms are computed from the raw audio. Then time warping is applied to the spectrograms, which involves warping the spectrogram along the time axis to introduce small stretches or compressions. During time masking, random blocks of consecutive time steps are masked (set to zero), which helps encouraging the model to be more invariant to temporal variations. The same happens for frequency masking, but makes it more invariant to irrelevant frequency variations.

Below in figure 7, a visualization of the different steps can be seen. In our case the spectrograms vary in time so padding is needed in order to pass them to the network.

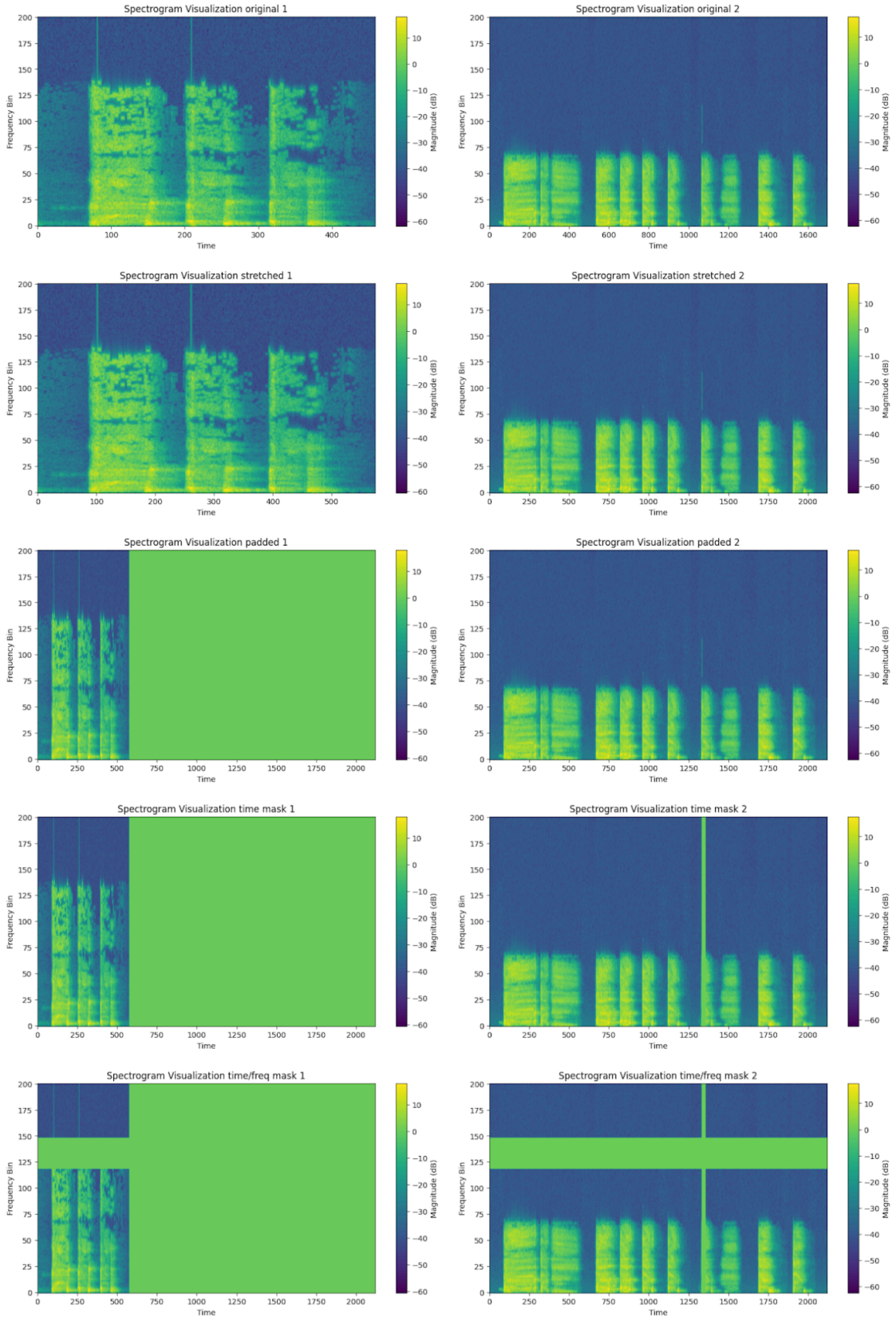


Figure 7. Visualization of the different data augmentation techniques used on two spectrograms.

▾ Influence of memory

Size of the dataset directly impacts memory usage, because storing a large dataset requires more memory. This includes original cough waveforms and generated spectograms. In the process we needed to reduce the number of samples in some of the classes, which also reduced the memory consumption (even though that was not why we decided to reduce the number of samples in some classes as discussed above).

We could potentially use a model architecture that requires fewer parameters and less memory. Architecture specifically designed for audio classification could be used, which could be more efficient in our case.

Size and resolution of spectograms can also affect the memory consumption. Experiments with different spectogram revolutions to find a balance between memory usage and classification performance could be performed. Lower resolution could still be sufficient for our task while consuming less memory.

Augmentation techniques: limit the number and complexity of augmentation techniques used during training to reduce memory consumption. Additionally we should pre-generate augmented data and store it to the disk to save the memory during the training.

▾ Additional features extracted

In addition to the augmented spectrograms, age, gender and SNR, we decided to include additional features: zero-crossing rate (ZCR), root-mean-square (RMS), and spectral centroid (SC), which are also commonly used features for audio classification. The ZCR measure the rate at which the signal changes its sign, from positive to negative or vice versa. The RMS measures the average power of the signal or the "loudness" or amplitude. The SC is a representation of the "cener of mass" or the mean frequency of the signal's power spectrum. It indicates where the average frequency of the signal lies.

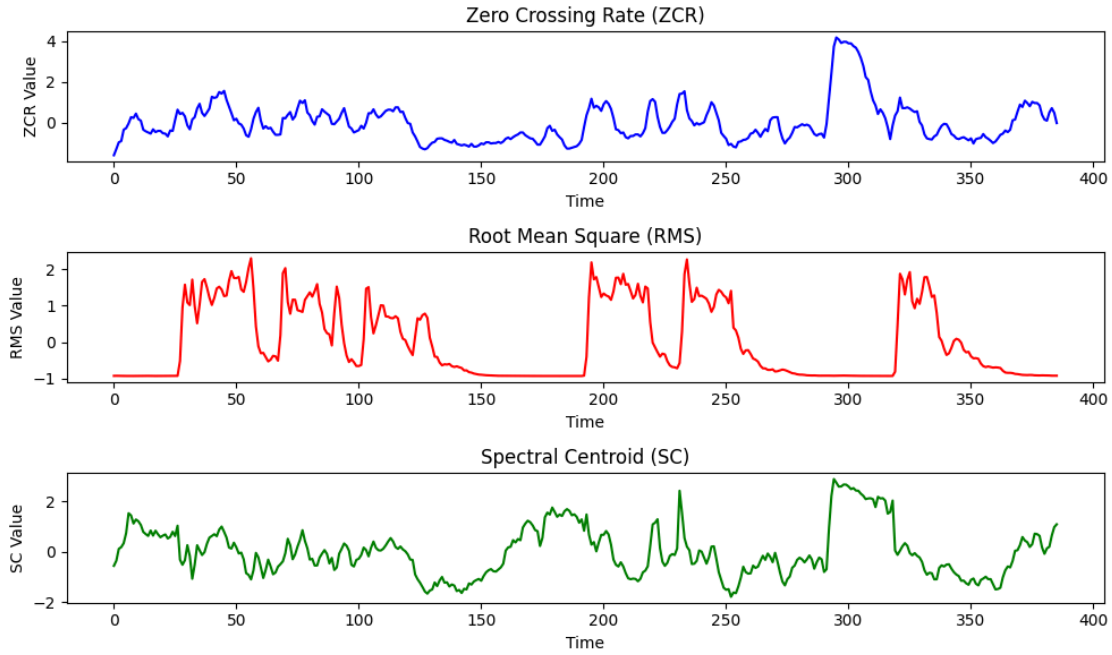
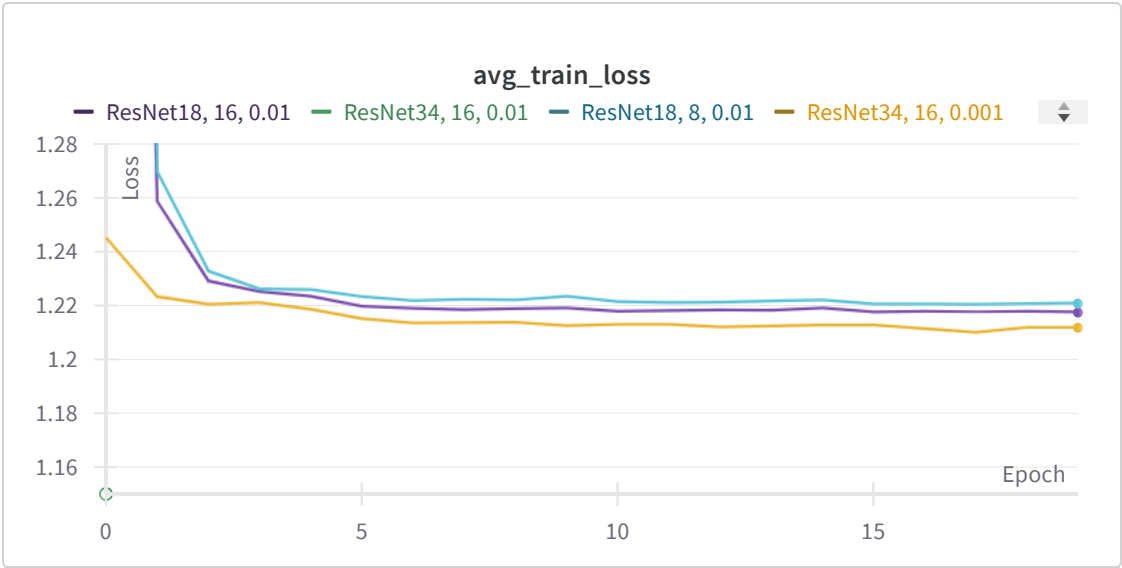
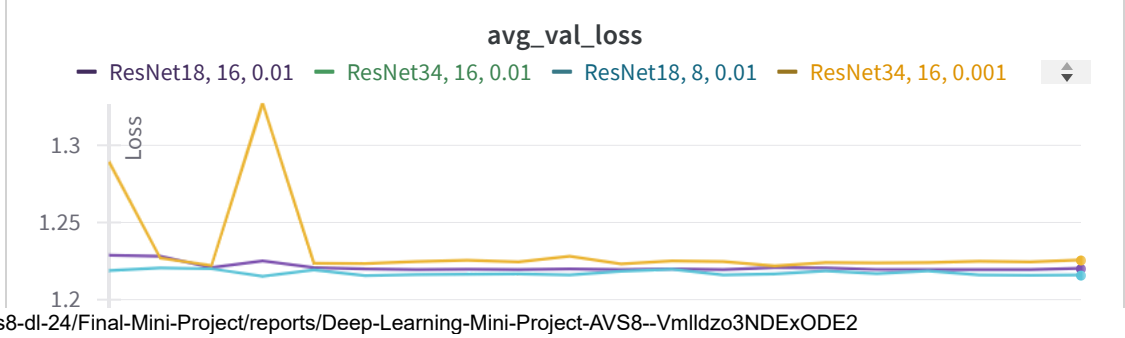


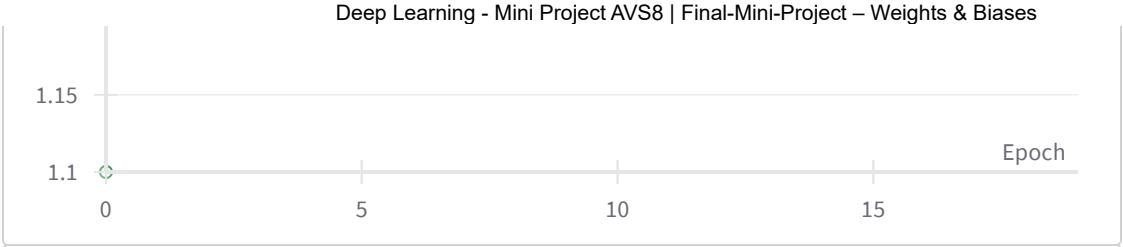
Figure 8. Visualization of the ZCR, RMS, and SC.

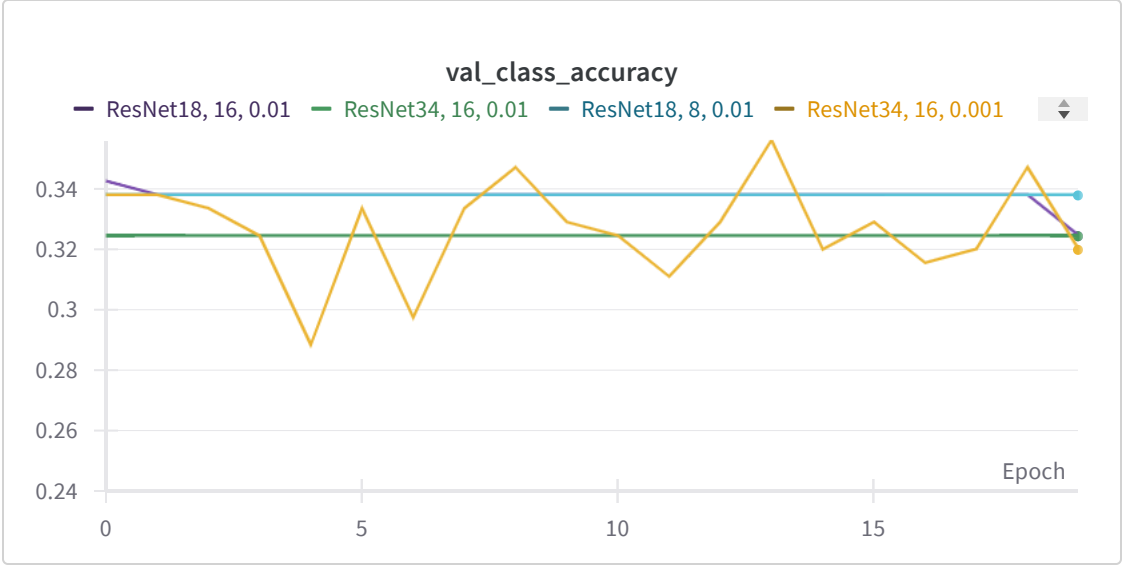
Training and validation results

These are the training and validation metrics for Alexander:

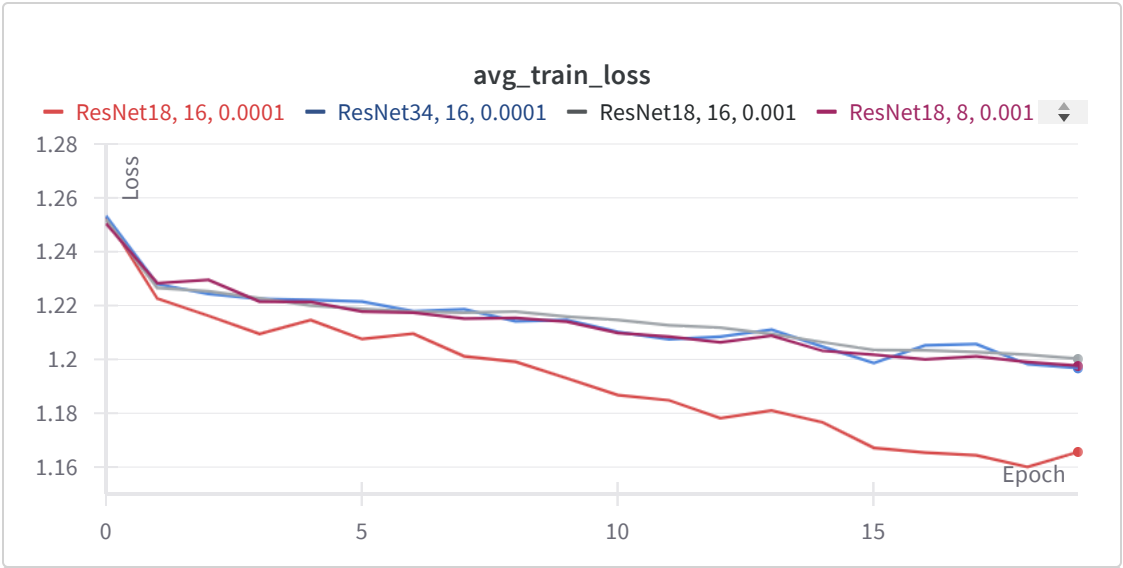


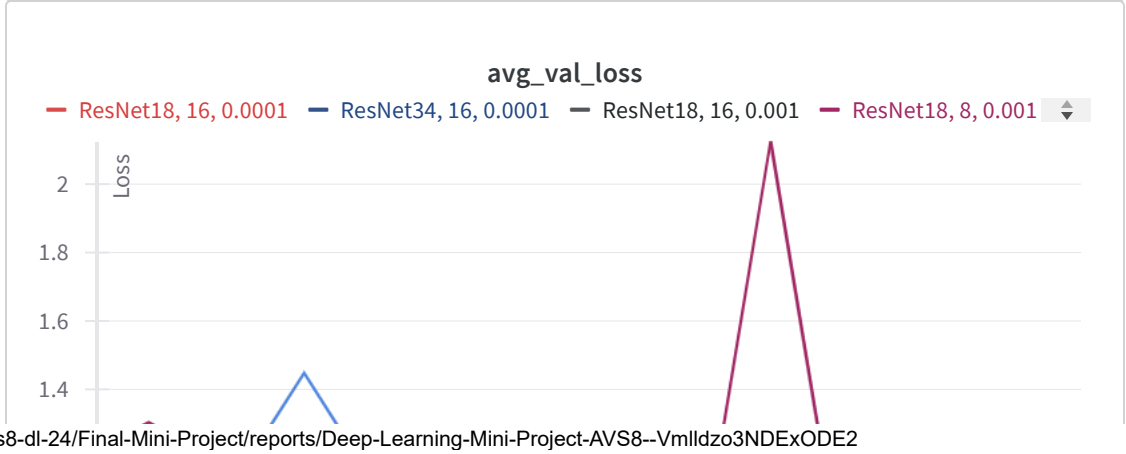


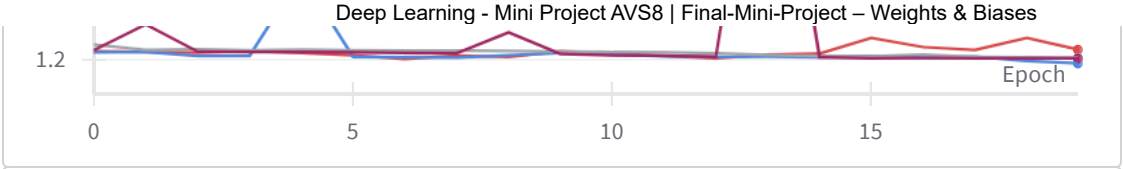


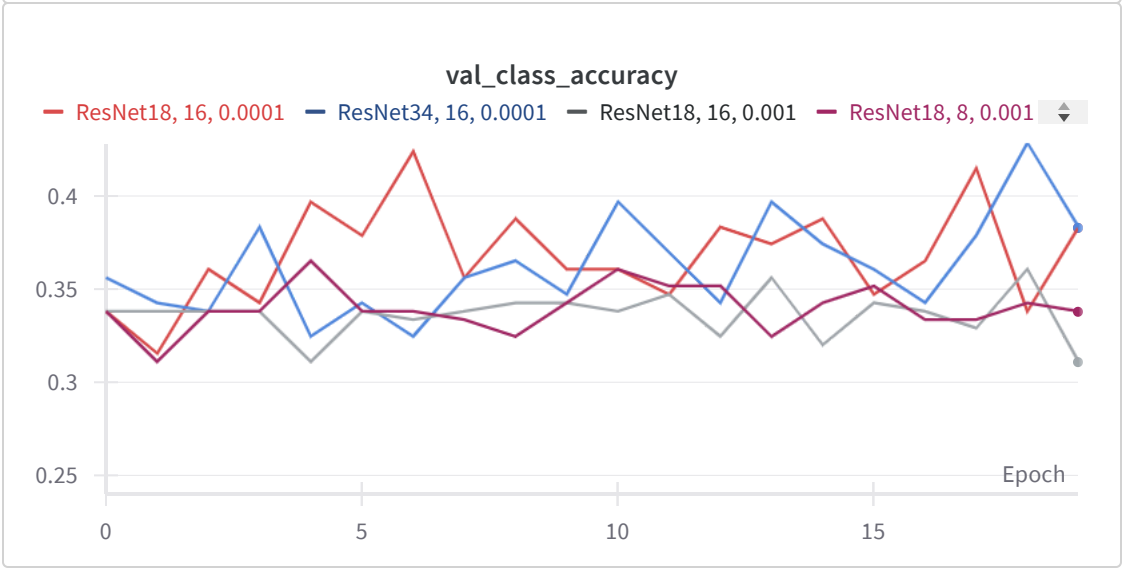


These are the training and validation metrics for Jasa:

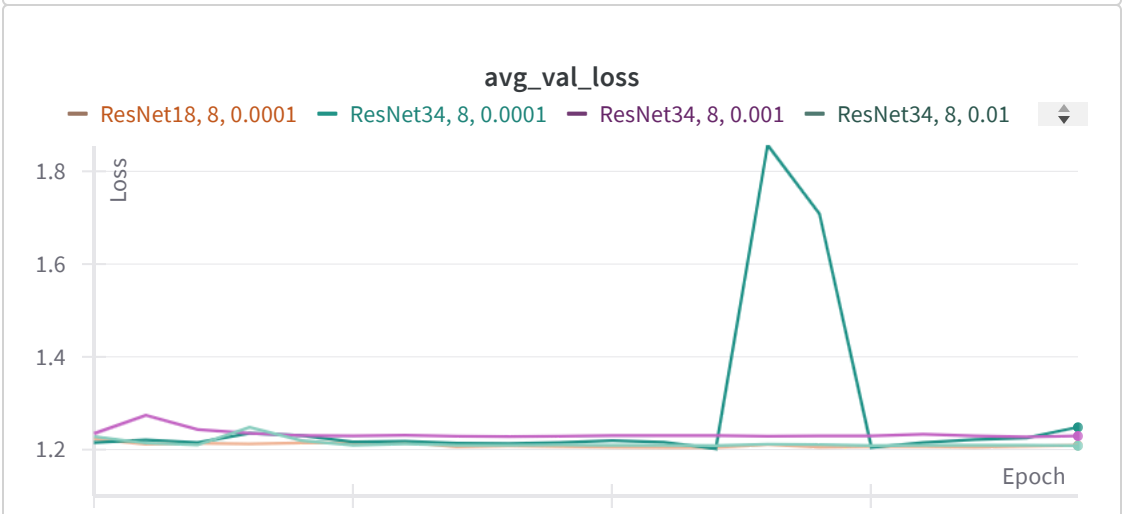


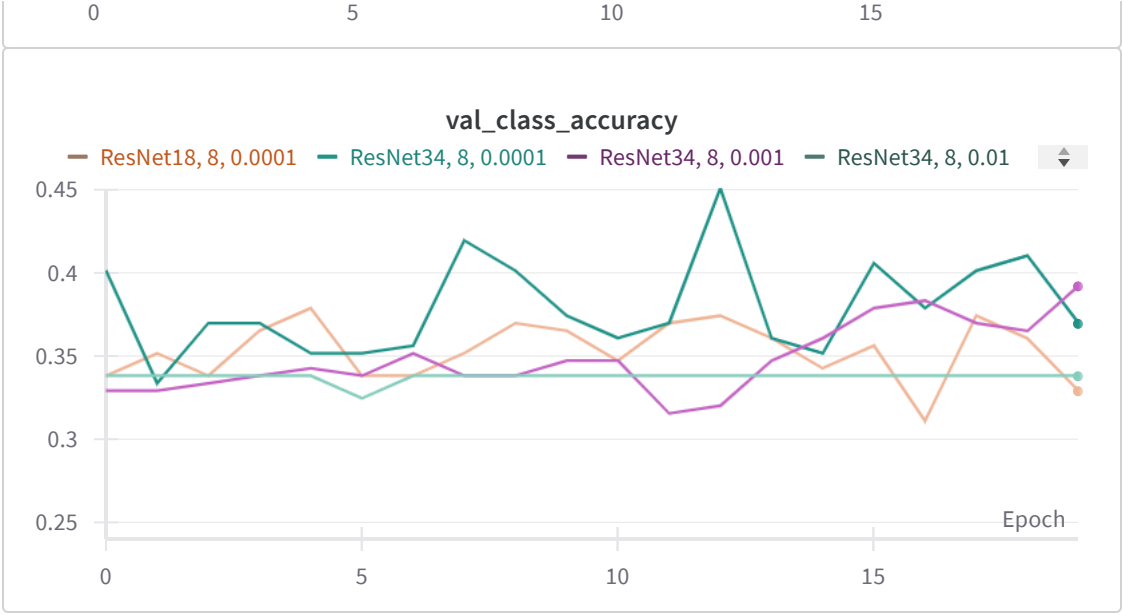






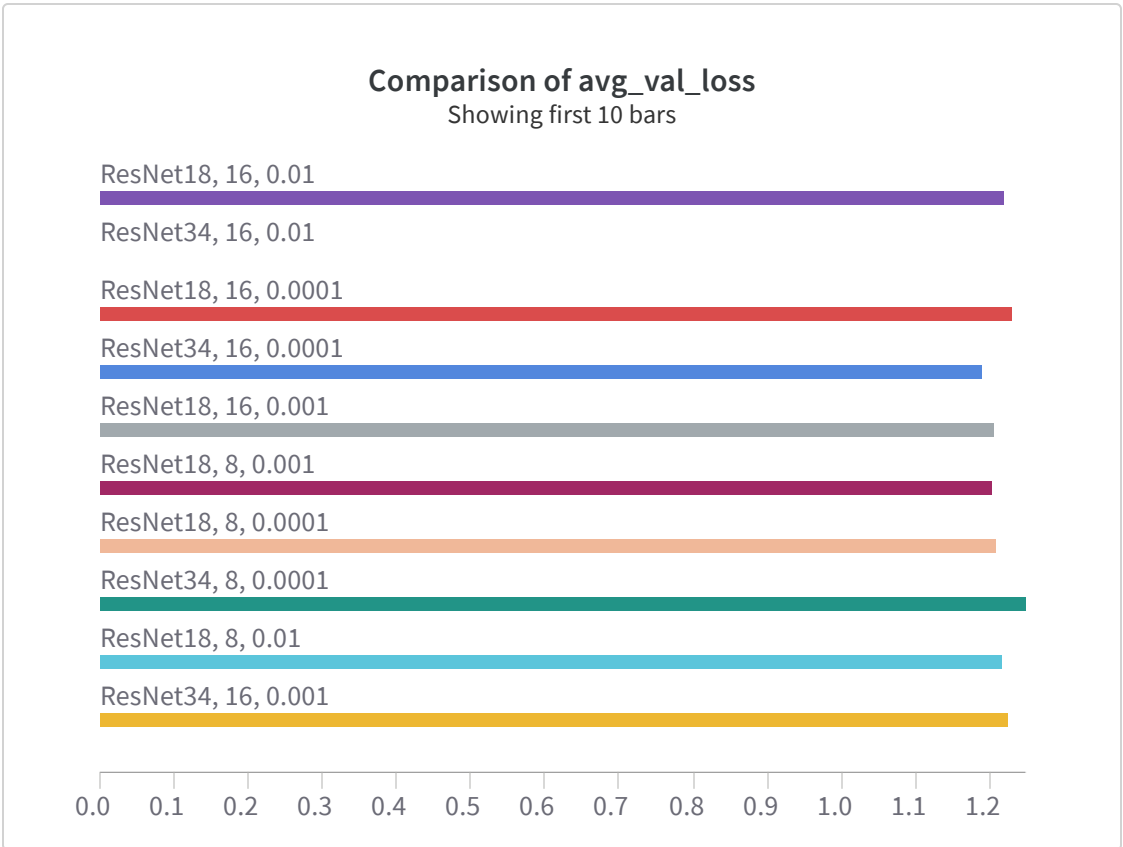
These are the training and validation metrics for Simon:

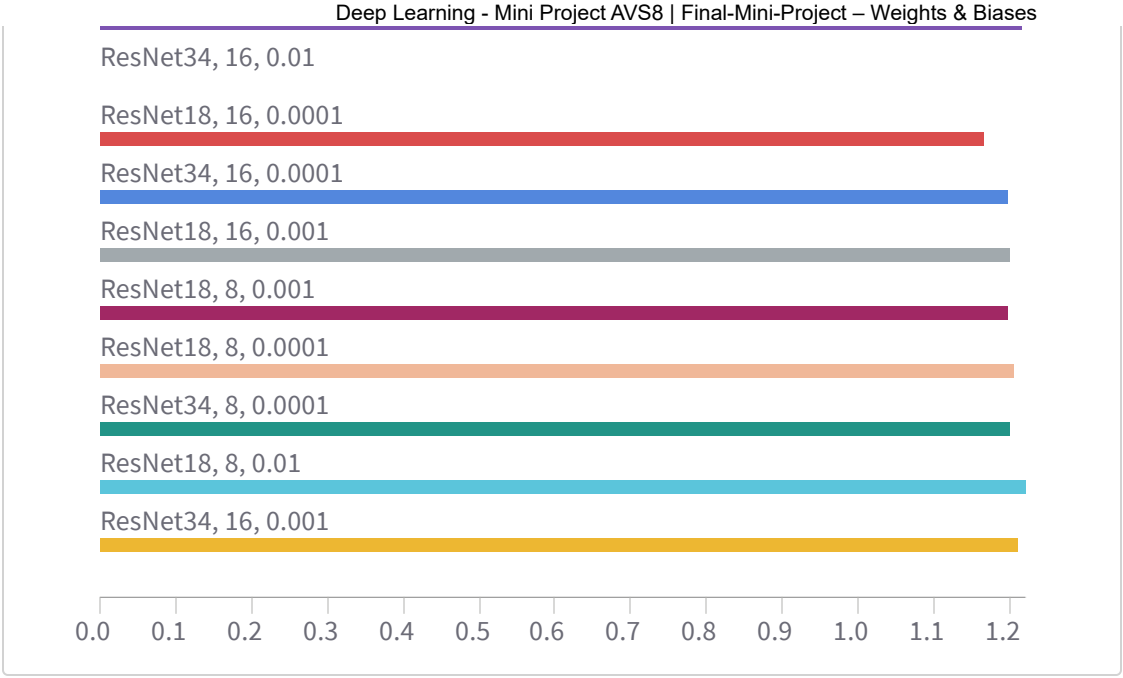




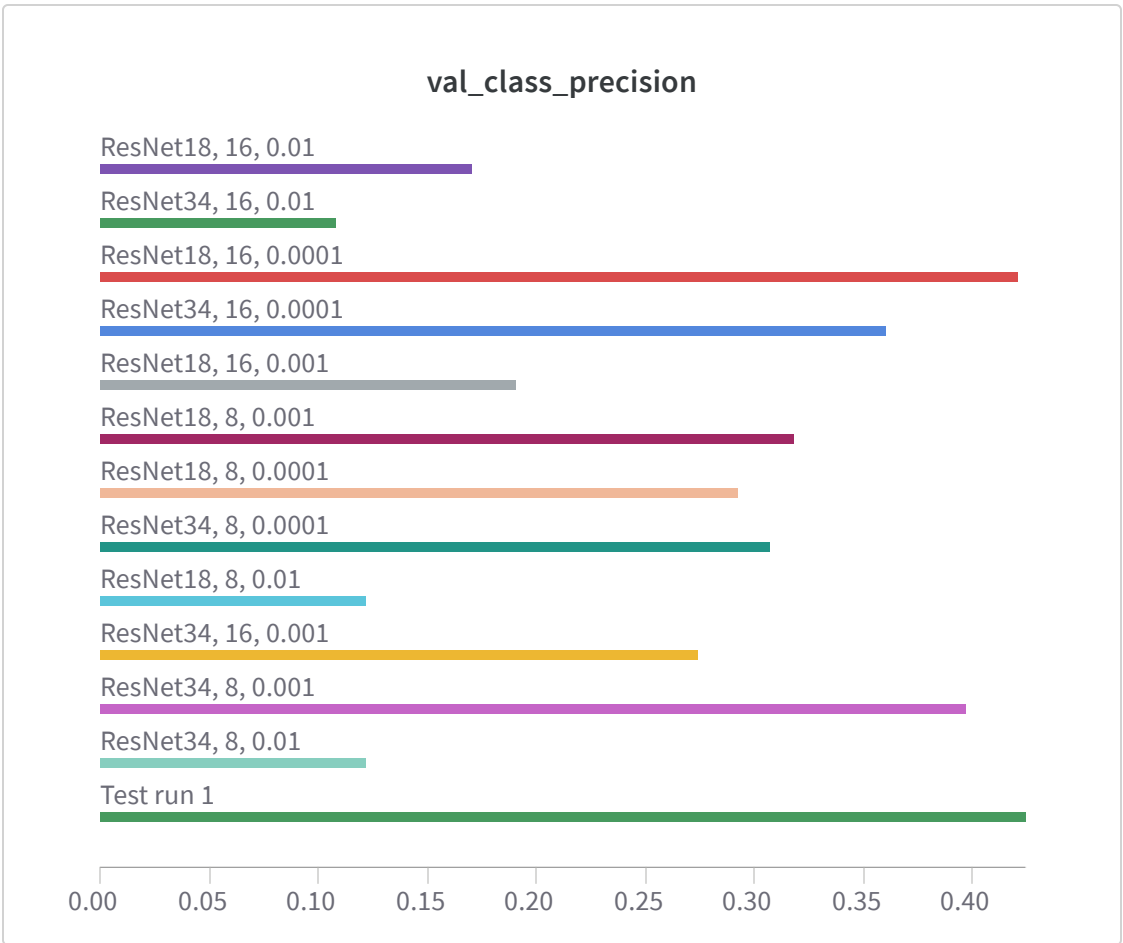
Summary of results

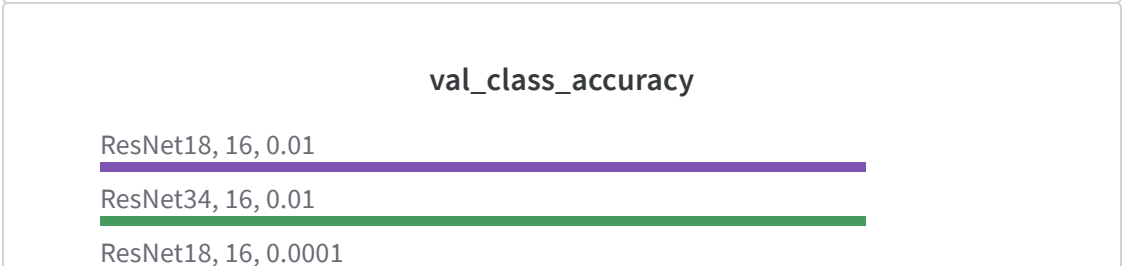
Comparison of average training and validation loss

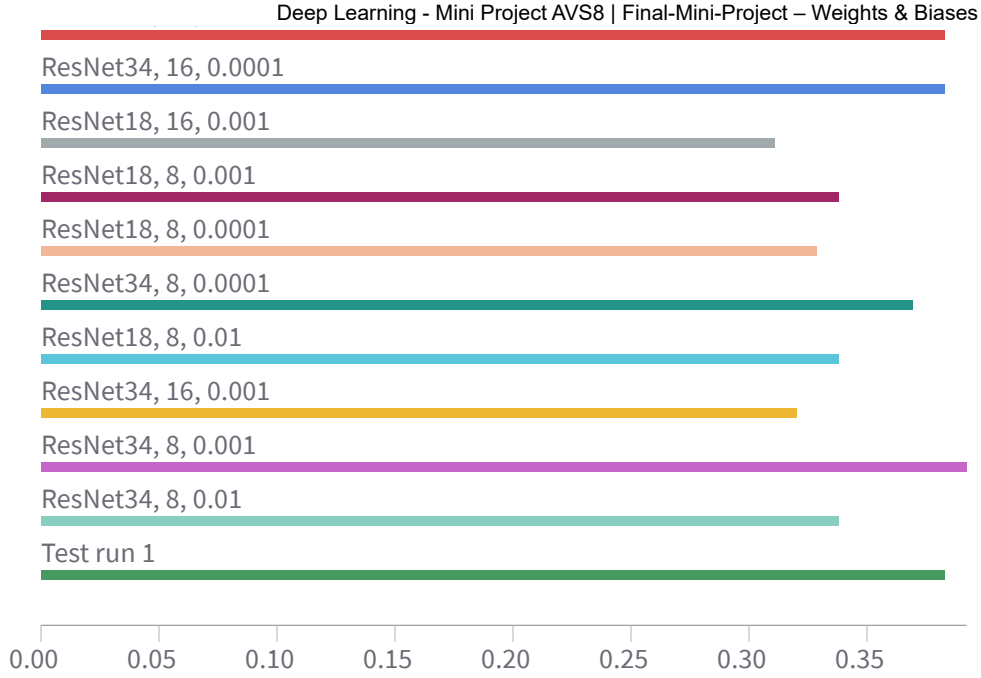




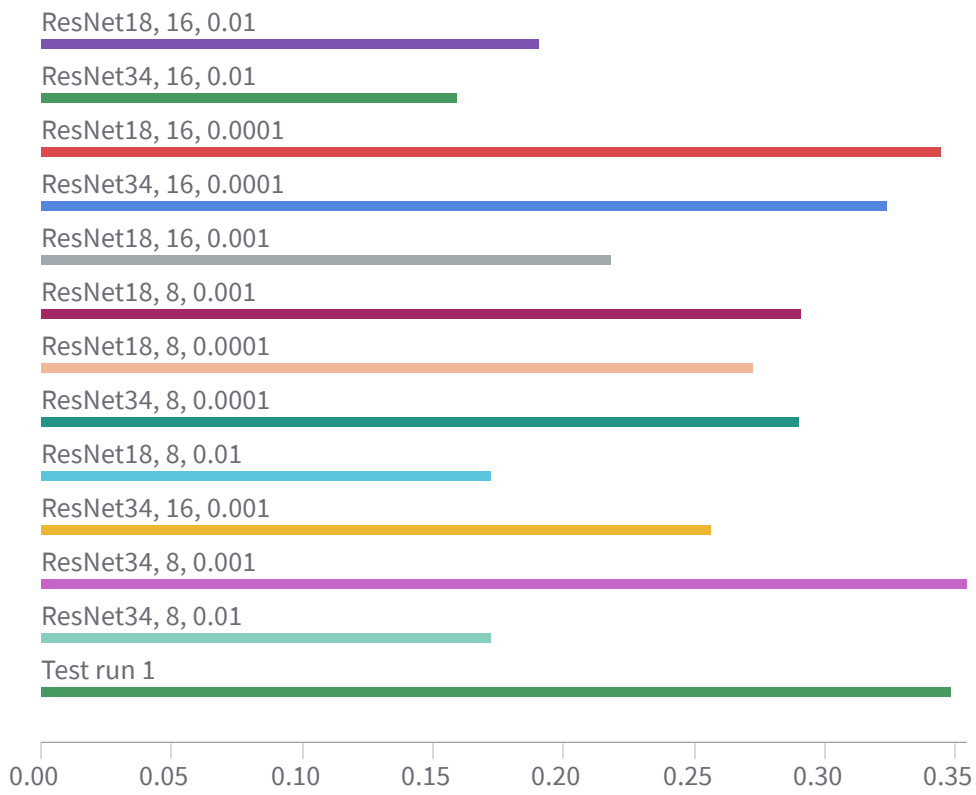
Comparison of performance metrics



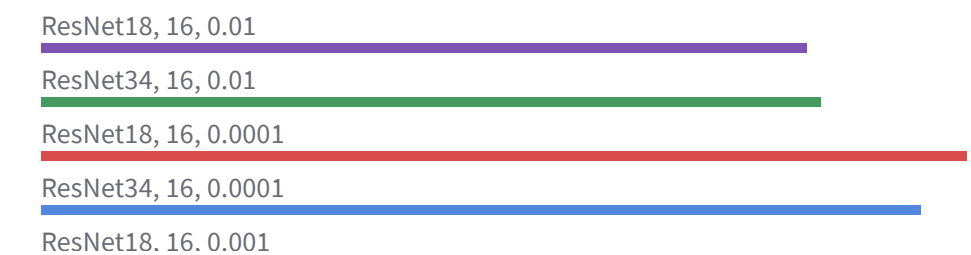


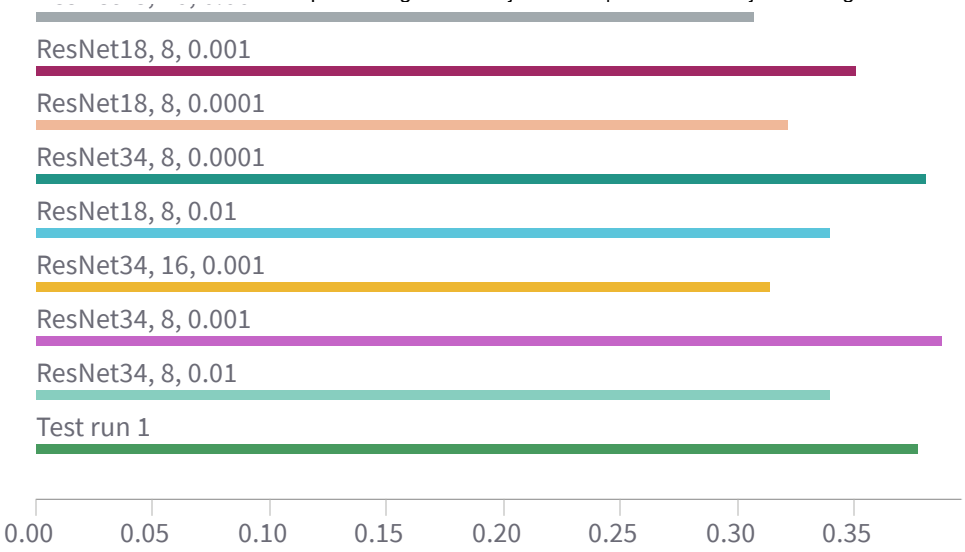


val_class_f1_score



val_class_recall





Confusion matrices of the best performing models

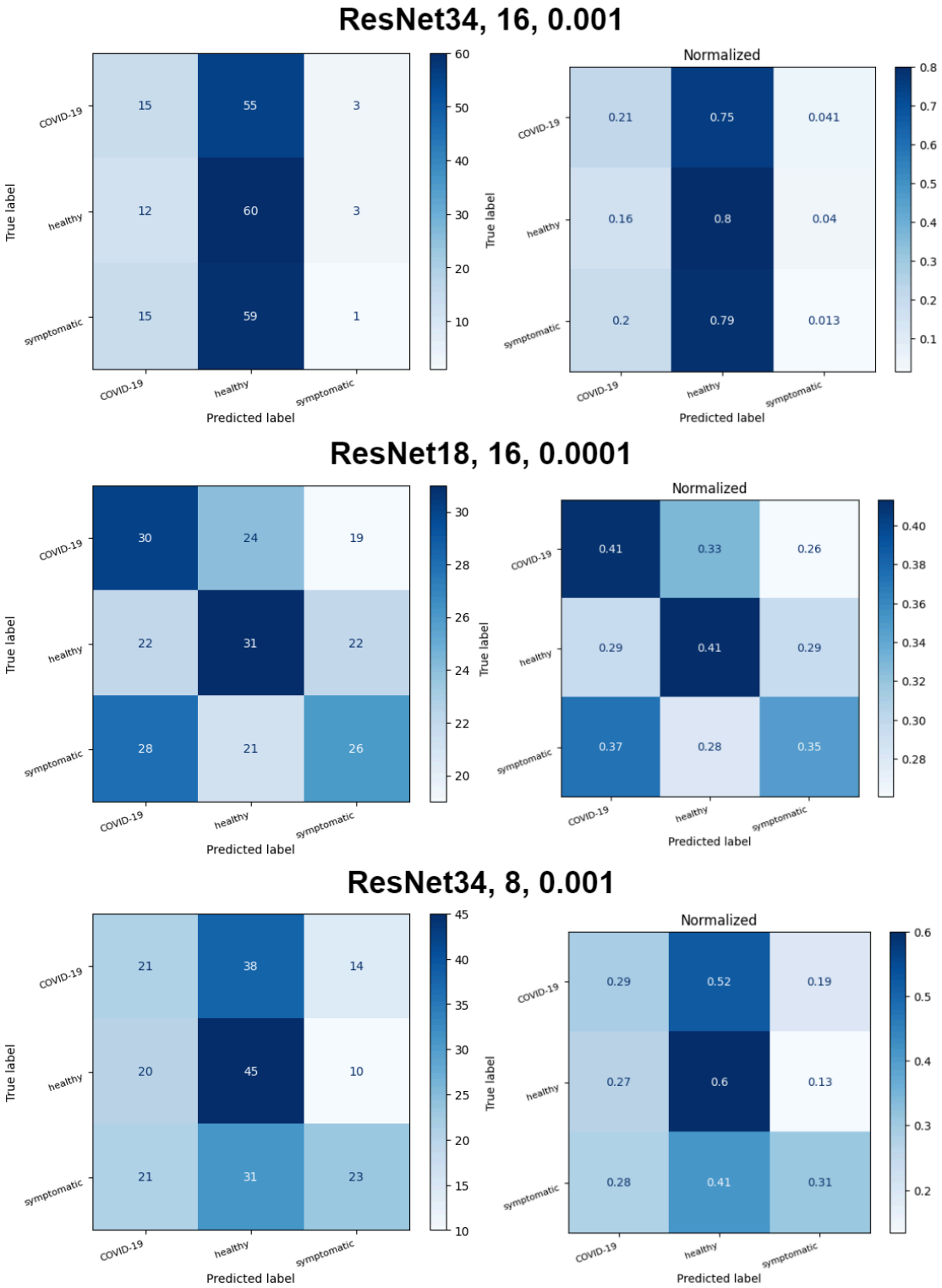


Figure 9. Confusion matrices of the best performing models.

Discussion / Breakdown of results

This project set out to classify coughs into three classes, being healthy, symptomatic, and COVID-19. For this kaggle was used for the training and development of the software handling the data processing and analysis of results with weights and biases used for plotting and tracking of results throughout the project.

The task of performing classification in this case proved to be difficult due to a multitude of reasons. One of these reasons being the imbalance of classes having a relation between the class with most and least samples being 10.8:1.

Another difficulty classifying this dataset is the big variety in both size of each sample but also the big variety in samples with varying amounts of silence and the location of the cough in each sample also being different. This means that hidden similarities in timing of coughs may interfere with more meaningful extraction of data. For this a lot of silent parts in samples were removed to make the content of each sample more similar.

In the end the results still suggest that minimal learning is happening as the accuracy on the validation set is consistently around the same value across all epochs during training. This can be due to many factors but the similar performance across all tests suggest that it is not likely to be any of the parameters adjusted or tested that on its own causes this problem. This is likely a problem with either the complexity of the problem itself or the representation used as input. The spectrograms may discard info that could be important for this classification such as phase.

With all this taken into consideration the last reason worth mentioning is the architecture of the models used. These models are originally designed for image classification tasks with the

initialisation of weights not being random but pre-trained on imagenet. This could prove problematic because even though the data used is a spectrogram which is an image the structure of it is not directly relatable to “normal” images. These images are sharper with more abstract visible features. An example of this being an image of a dog. In this case there is a strong relation between features allowing for the outline of the dog to be identified but this is not as prominent in a spectrogram that is more visibly chaotic. A model that could have been more suitable for a task like this is the VGGish model. This is a CNN-based model designed specifically for audio classification tasks. It extracts embeddings from audio spectrograms and has been pre-trained on a large-scale audio dataset called AudioSet.

Conclusion

This project aimed at classifying whether or not a person was sick with COVID-19 based on their cough. This task proved to be challenging with difficult data that turned out to be difficult for the chosen models to classify. This is both due to the chosen representation and challenging data with small variations between classes and potentially discarded information from the chosen representation of audio. Even with these challenges the results still prove that this approach is not effective and other approaches will need to be investigated instead. Some of the approaches that could be considered is using the audio in its raw format or finding a latent representation that does not discard phase information like spectrograms does.

Regardless of this the best performing model configuration seems to be the Resnet18 architecture with a batch size of 16 samples and a

learning rate of 0.0001. This model has the highest accuracy with the best results as seen in the confusion matrices shown.

Created with 💔 on Weights & Biases.

https://wandb.ai/avs8-dl-24/Final-Mini-Project/reports/Deep-Learning-Mini-Project-AVS8--Vmlldzo3NDExODE2