Benchmarking Neural Network Architectures for Environmental Sounds Classification

Marco Chiloiro, Michail Sapkas

***Abstract*—In this paper, we are benchmarking multiple neural network architectures using the ESC-10 dataset. Although some of the models used may seem trivial, we think it is necessary in order to better understand their sound classification capabilities and weaknesses. We will also use elaborate combinations of the simple models, invoking both supervised and unsupervised methods and augmentation of data.**

***Index Terms*—Environmental Sound Classification, Benchmarking Neural Networks architectures, Data Augmentation**

I. Introduction

Environmental sound classification is a crucial task in various applications, including smart cities, wildlife monitoring, urban planning, underwater exploration and surveillance. The ability to accurately classify environmental sounds can lead to significant advancements in automated systems, contributing to better decision-making and enhanced situational awareness. In recent years, neural networks have demonstrated remarkable performance in various domains of pattern recognition and classification, with examples like Apple's Siri, Google Assistant, Amazon's Alexa and more, spurring interest in applications to environmental sound classification.

We explore several neural network architectures, including Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and hybrid models. Each of these architectures brings unique advantages to the table: CNNs excel in capturing pixel patterns in audio spectrograms, RNNs are proficient in handling temporal dependencies, and hybrid models hopefully will combine the strengths (or weaknesses) of both approaches. By benchmarking these models against the standard dataset of ESC-10, we aim to provide a first approach analysis of their accuracy, both aggregate and on different classes, by computing the confusion matrices. We also invoke some basic hyperparameter tuning on our models, using the Keras tuner module.

II. Related Work

We will expand on the work done by Karol J. Piczak [1] on compiling the Environmental Sounds Classification dataset. In this cited work, the sub-field of recognizing sounds of the environmental scene is born, benchmarking baseline classifiers like k-nearest neighbors (k-NN), random forests, and support vector machines (SVMs). Preliminary results indicate that while traditional classifiers achieve moderate accuracy, CNNs demonstrate superior performance due to their ability to capture intricate audio patterns.

Furthermore, in Karol J. Piczak's github repository [2] there are a lot of Neural Network Architectures deployed during the 7 years that this dataset has been introduced. As stated above, CNNs dominate in various architectural forms and preprocessing of the inputs.

Another noteworthy reference is the GitHub repository [3], which served as inspiration for a CNN architecture used with the ESC-10 dataset.

III. Data and Preprocessing

A. The ESC Dataset

The ESC-50 and ESC-10 datasets are curated collections of environmental sound recordings designed for research in sound classification. Both datasets are derived from recordings available through the Freesound project and include various types of common sound events.

The ESC-10 dataset is a simplified subset of the ESC-50, containing 400 clips across 10 classes, with 40 clips per class. This subset includes a mix of transient/percussive sounds (e.g., sneezing, dog barking), sounds with strong harmonic content (e.g., crying baby, crowing rooster), and structured noise/-soundscapes (e.g., rain, sea waves). The ESC-10 is designed to provide an easier benchmark for classification tasks, with more pronounced differences between sound classes.

Since our focus is on deploying Neural Network Architectures, we are going to perform our benchmarks using the ESC-10. This allows us to keep data complexity to a minimum and to better detect the strengths and weaknesses of the models with respect to the transient type of the audio data inputs.

B. Preprocessing

1) Mel Spectrograms: Unprocessed audio signals are basically timeseries of the intensity of sounds reaching the microphone, sampled in a specific sampling rate. In the field of audio recognition using NNs, instead working with the raw audio timeseries it is standard practice to transform the data into Spectrograms.

A Spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time. Dividing the whole sample into time windows, we perform a Fourier Transform on each time window, thus decomposing each temporal chunk into a set of frequencies. In practice, the frequencies are grouped into ranged bins, and the result is a new representation of the raw signal as a matrix containing in each column, a time window, in each row the frequency bins corresponding to a time window, and effectively each value represents the intensity of the specific frequency range on that time window.

A standard convention is to impose one more rule on the Fourier transformation of the raw audio, which is to use a non-linear set of ranged bins, called "Mel filterbanks" attributed to the name of the researcher that first implemented them. The motivation of using Mel's filterbanks is to approximate the human perception of sound. This comes from heuristic findings that humans perceive better variations in low frequencies and much less in high audio frequencies.

The result of this preprocessing are Mel Spectrograms, and the whole process invokes the use of some parameters that affect the final representations. The most important are:

- The Sampling Rate actually used for processing the audio sample. We used 16 kHZ.
- The number of Mel filterbanks to use. We used 124.
- The size of the time window, in which we used 1024 samples.
- The number of samples between successive frames: 512. Given that the FFT window length is 1024 samples, this results in overlapping windows with a 50% overlap.
- The size of the time windows and how much they overlap define the total length of all samples. In our case, we end up with 156 time frames, i.e. each frame lasts 32 ms.

Once we have converted all audio samples into Mel Spectrograms each one of 124 rows of Mel filterbanks and 156 time steps, now we also convert the intensity of the signals into the logarithmic power spectrum using as reference level the minimum value of each sample. In this way, samples represent intensity of sound in the way humans perceive it, and each sample is normalized on its own relevant scale.

Finally, we invoke the *MinMaxScaler* module from *sklearn* to rescale all samples between -1 and 1. All Neural Networks that we are going to use not only benefit numerically from this, but we discovered that if we don't use some kind of scaling, most of them are impossible to train.

Rescaling from -1 to 1 all pixel values, help RNNs since they invoke the hyperbolic tangent activation function multiple times, ergo working between these values is extremely beneficial. For this study we are going to use the same rescaling for all architectures, although it is possible that different Neural Networks could benefit from a different specific rescaling of the input.

2) Data Augmentation: It is immediately evident that the ESC-10 Dataset is extremely small, especially if we consider that there is going to be a train - test split, not to mention possible validation sets to monitor overfitting during training. In order to help the models to generalize, we will induce augmentations on the training set. For this task, we will invoke a python library build to do exactly this, called *audiomentations*. The audiomentations library uses randomness to apply user specified augmentations on the raw audio data (meaning before the Mel Spectrogram conversion). We specifically used the following augmentations for the audio samples:

- TimeStretch - (min_rate=0.8, max_rate=1.2)
- PitchShift - (min_semitones=-4, max_semitones=4)
- Shift - (min_shift=-0.1, max_shift=0.1)

Randomness is introduced at two levels in the augmentation process. First, there is the probability p that a specific augmentation will be applied. Second, if the augmentation is applied, a random value within a defined interval is selected for the augmentation's parameters. For instance, consider the time shift (Shift) augmentation: with a probability p, a random shift value within a specified minimum and maximum range is chosen for each sample. To ensure reproducibility and interpretability, we set $p = 1$. This means that the augmentation will always be applied, eliminating the first level of randomness. Furthermore, each augmentation is applied once per sample, resulting in an augmented training set that is four times larger than the original one (three augmented versions plus the original). The same preprocessed data are used for training all the considered models, ensuring consistency in the training process.

C. Train - Test Folds

To fully utilize the dataset's potential, we leverage its existing 5-fold split. Specifically, we perform 5-fold cross-validation by using 4 folds as the training set and the remaining fold as the test set. For the training set, we also include the augmented data. However, for the test set, we only use the original, unaugmented data. This approach ensures a robust evaluation while maximizing the training data available to the models.

Finally, to obtain comprehensive statistics on the training results, we aggregate the outputs from each fold of the 5-fold cross-validation and compute their average. This approach allows us to assess the overall performance and effectiveness of the model across multiple training sets using the same Dataset. This technique is appropriate when dealing with limited size Datasets.

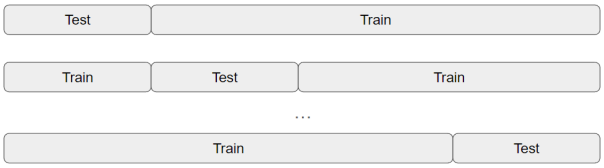

Fig. 1: Visual representation of the 5-folds cross-validation train-test splitting. The dataset is divided into five equal parts. Each part is used as the test set once, while the remaining parts form the training set. This process is repeated five times.

IV. Neural Network Architectures

In this section, we are presenting the Neural Network architectures that we used. Following the picture, is a brief explanation of the architecture and the motivation to use it. Then we will present training metrics. But, before we delve into the architectures, let us give some information about the optimizer and the loss function that we used.

1) Optimizer: Adam optimizer has become the industry standard due to its efficiency in various scenarios. We utilized Adam optimizer with a batch size of 32 across all models.

2) Loss Function: In all models that result to a Dense Feedforward Neural Network as Classifier we used "Categorical Cross-Entropy" which serves as a probability measure when using numbered classes (eg. '0' = 'dog bark', etc).

For Autoencoders we used the "Mean Squared Error" Loss function, which is appropriate for reconstruction of image-like input-output.

3) Hyperparameter Tuning - Keras Tuner: We employed the Keras Tuner module in order to perform hyperparameter tuning to most models. The technique used was 'Bayesian Optimization' which has proven to converge quickly to optimal values. Depending on the model, the hyperparameters optimized where:

- The final number of neurons of the Dense Layer Classifier
- The activation function of that layer
- Dropout before the 10 neuron classifier layer
- The number of kernels for each distinct CNN layer
- The size of the kernel on that layer
- The size of each Max Pooling layer
- The number of the hidden state neuron on the LSTM/-GRU
- The amount of Dropout used in the LSTM/GRU layer

This will also explain why, in some cases, the number of neurons may seem "random". Let's start reviewing the architectures.

A. Feedforward (Dense) Neural Network (FNN)

Feedforward neural networks (FFNs) serve as a valuable benchmark for general classification tasks due to their straightforward architecture and effectiveness in learning complex patterns from data. Their simplicity and ability to handle a wide range of datasets make them a reliable baseline model against which more complex architectures can be compared. This makes FFNs a preferred choice for establishing a baseline performance in classification problems before exploring more advanced neural network architectures. In Figure 2, the depicted architecture illustrates the neural network configuration used.

As shown in Figure 3, the model converges on the validation set after just a few epochs, while it continues to improve on the training set. Extending the training for additional epochs would only result in increased overfitting. From this, we can clearly see the limitations in generalizing outside the training set of such architecture. This clearly highlights the architecture's limitations in generalizing beyond the training set, as FNNs cannot efficiently capture spatial and temporal dependencies.

B. Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) are highly effective for classifying images due to their ability to capture spatial hierarchies and patterns in the data. By using local receptive fields and pooling layers, CNNs can extract and condense relevant features from Mel spectrograms, making them robust for audio classification tasks. These characteristics make CNNs particularly powerful and efficient for Mel spectrogram classification. Figure 4 illustrates the neural network architecture.

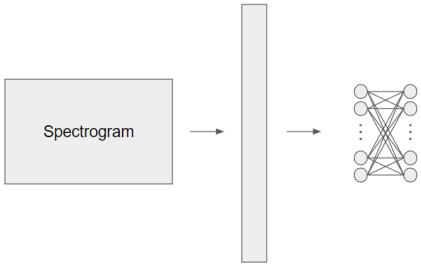

Fig. 2: Structure of a Feedforward Neural Network (FNN) using dense layers. Input spectrograms are flattened and sequentially processed through three dense layers: the first with 1088 neurons using ReLU activation followed by a dropout of 0.4, the second with 1984 neurons and a dropout of 0.3 using hyperbolic tangent activation, and the third with 480 neurons and a dropout of 0.3 using ReLU activation. The final layer consists of 10 neurons with a softmax activation function for classification.

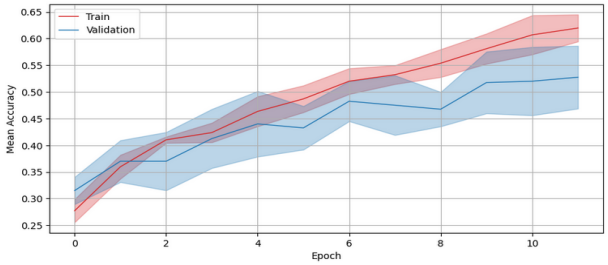

Fig. 3: Average training history of the FNN architecture over 12 epochs.

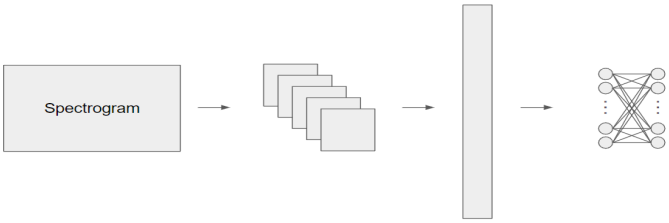

Fig. 4: Structure of a convolutional neural network (CNN) for spectrograms classification. The network consists of three convolutional layers with filter sizes 5x5, 5x5, and 3x3, and respective numbers of filters 24, 36, and 48. Between the convolutional layers, max pooling with a kernel size of 3x3, 2x2 and a stride of 3, 2 are applied, with appropriate zero padding to maintain spatial dimensions. Following the convolutional layers, the output is flattened and fed into a multilayer perceptron (MLP) comprising a Dense layer with 256 neurons. The final layer uses softmax activation for 10 output neurons corresponding to class probabilities. ReLU activation functions are used throughout. Dropout and L2 regularization are applied to mitigate overfitting.

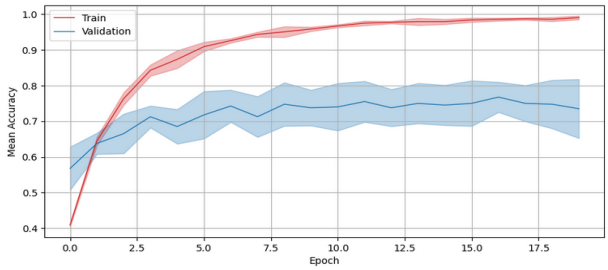

Fig. 5: Average training history of the CNN architecture over 20 epochs.

As shown in Figure 5, this CNN architecture outperforms the simpler FNN, even with fewer trainable parameters. This improvement is due to the CNN's ability to efficiently capture spatial features. The model achieves nearly 100% accuracy after just a few epochs, indicating overfitting due to the limited amount of available data, resulting in an overly complex model.

To mitigate this overfitting, we propose an alternative CNN architecture with reduced complexity. Instead of directly feeding the output of the convolutional layers to the MLP by flattening it, we apply a global average pooling layer. This pooled output is then passed to the MLP. Consequently, the MLP's hidden layer size is reduced from 256 neurons to 64 neurons, reflecting the smaller output shape from the global average pooling layer. This modification aims to maintain performance while reducing the risk of overfitting by simplifying the model.

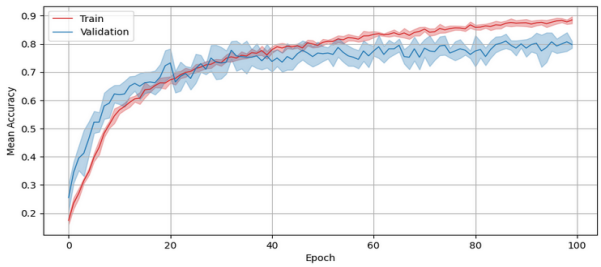

Fig. 6: Average training history of the revised CNN architecture over 100 epochs.

As illustrated in Figure 6, the revised model demonstrates improved performance over the previous architecture while significantly reducing overfitting. Unlike the initial model, which reached nearly 100% training accuracy after just a few epochs, this new model's training accuracy is now more closely aligned with its validation accuracy. This indicates a better balance between model complexity and generalization, leading to more robust performance on unseen data. The application of global average pooling and the reduction in the MLP hidden layer size effectively curbed the overfitting, resulting in a model that performs consistently well on both training and validation sets. However, this improved performance and reduced overfitting come with the need for more training epochs. The revised model requires additional epochs to achieve optimal results, reflecting its more gradual and balanced learning process.

C. Long Short Term Memory (LSTM)

Long Short-Term Memory (LSTM) models are particularly useful due to their ability to effectively capture temporal dependencies and long-range dependencies in sequential data. Mel spectrograms, which represent the frequency content of audio signals over time, often exhibit complex temporal patterns that LSTM models can learn and leverage for classification tasks. Figure 7 illustrates the neural network architecture.

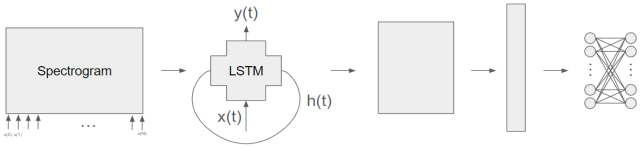

Fig. 7: LSTM architecture composed by: LSTM layer (512 units, 20% dropout, returning sequences), followed by flattening, a dense layer (448 units, tanh activation, dropout 20%), and a final dense layer (10 units, softmax activation) for classification.

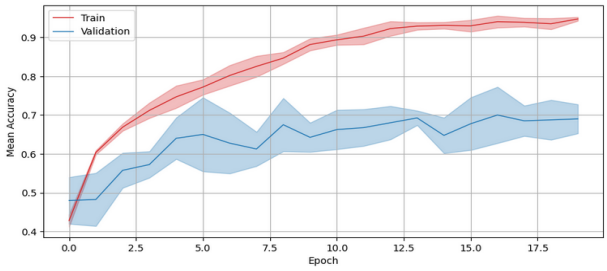

Fig. 8: Average training history of the LSTM architecture over 20 epochs.

From Figure 8, overfitting is evident. Moreover, the performance is not significantly better than the FNN, suggesting that applying LSTM directly on spectrograms fails to capture essential information for generalization. This limitation points to the necessity for a more sophisticated approach.

To address this issue, we propose combining CNN and LSTM in the next models.

D. CNN - Reshaped Features - LSTM

The first architecture we consider which combines CNN and LSTM architectures is illustrated in Figure 9. The key feature of this model is its processing pipeline: after the initial convolutional layers, we stack all the filters (or channels) such that, for each time step, all the values at that time step from all the filters are combined. This stacked sequence is then passed to an LSTM, which extracts temporal correlations. The LSTM outputs a sequence of hidden states, which are flattened and subsequently passed through a final MLP for classification. This method leverages the strengths of both

CNNs and LSTMs, capturing intricate spatial and temporal patterns within the audio data.

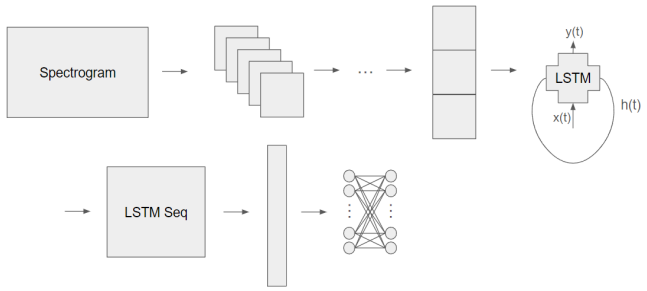

Fig. 9: Structure of a CNN-LSTM Model for Audio Classification using convolutional layers followed by an LSTM. The architecture includes the following layers with ReLU activation: a CNN layer with 80 filters, a kernel size of 3x3; max pooling with a 2x2 pool size; a second CNN layer with 128 filters, a kernel size of 4x4; max pooling with a 5x5 pool size. The output is then permuted and reshaped. This reshaped data is passed to an LSTM layer with 32 units and 30% dropout. The data is then flattened and passed through a dense layer matching the input shape with ReLU activation, followed by another dense layer with 160 units and tanh activation, and a dropout of 30%. The final layer consists of 10 units with a softmax activation function for classification.

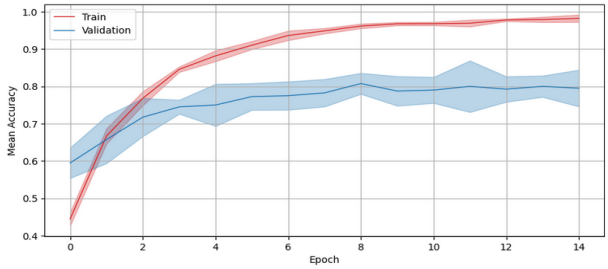

Fig. 10: Average training history of the CNN-RF-LSTM architecture over 15 epochs.

As observed in Figure 10, the overfitting persists, evidenced by the training accuracy reaching 100% after just a few epochs. Despite this, the model exhibits significantly improved validation performance compared to a simpler LSTM applied directly to spectrograms. This phenomenon suggests that while the model is capable of classifying the training data to a high degree, it generalizes better to unseen data during validation. The integration of CNNs prior to the LSTM likely plays a crucial role in this improvement. The convolutional layers effectively extract spatial features from the spectrograms, which are then processed by the LSTM to capture temporal dependencies. This hierarchical approach enables the model to learn both intricate spatial patterns and temporal correlations within the audio data. Therefore, despite the overfitting during training, the validation results indicate that the CNN-LSTM architecture is effectively leveraging its combined strengths to achieve better generalization performance compared to a standalone LSTM model. Adjustments such as regularization techniques or dropout could potentially mitigate the overfitting issue further, while continuing to harness the benefits of both CNNs and LSTMs in audio data analysis.

E. CNN - Time Distributed LSTM

In this revised CNN-LSTM architecture (see Figure 11), we adopt a different approach from the previous model. Instead of directly passing the stacked output of the CNN to the LSTM, we apply a Time Distributed LSTM operation for each individual filter. After processing through these Time Distributed LSTM, the final hidden states are aggregated. This aggregated representation, which synthesizes the temporal information learned by each LSTM, is then forwarded to the MLP for the classification task.
This modified architecture aims to treat each filter's temporal evolution independently.

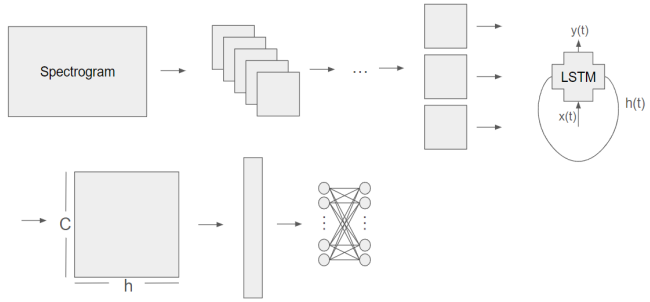

Fig. 11: Structure of a CNN-Time Distributed LSTM model. The architecture begins with a CNN layer with 64 filters and a kernel size of 3, using ReLU activation, followed by a MaxPooling2D layer with a pool size of 2x2. The second CNN layer has 56 filters and a kernel size of 2, also using ReLU activation, followed by a MaxPooling2D layer with a pool size of 3x3. The third CNN layer has 48 filters and a kernel size of 2 with ReLU activation, followed by a MaxPooling2D layer with a pool size of 4x4. The output is then passed through a Time Distributed LSTM layer with 56 units and 10% dropout. The data is then flattened and passed through a dense layer with 192 units and tanh activation and 20% dropout. The final dense layer consists of 10 units with a sigmoid activation function for classification.

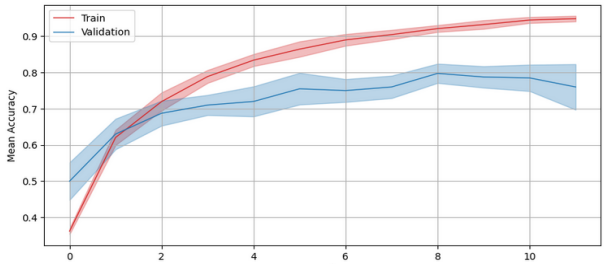

Fig. 12: Average training history of the CNN-TD-LSTM architecture over 12 epochs.

Figure 12 reveals that the training behavior of the CNN-Time Distributed LSTM model is quite similar to that of the previous CNN-LSTM architecture. The number of parameters in both models is of the same order of magnitude, approximately 5×10^5, indicating comparable model complexity. While the validation accuracy between the two models does not show a significant difference, an important observation is that the overfitting appears to be less pronounced in the CNN-Time Distributed LSTM model. This reduced overfitting suggests that the Time Distributed LSTM approach may be more effective in learning generalizable features from the data. By processing the temporal sequences separately, the model might be benefiting from a more focused temporal feature extraction, reducing the risk of overfitting compared to a single LSTM handling all stacked features simultaneously.

F. LSTM - CNN

Another possible combination of LSTM and CNN architectures involves reversing the order of their application. This architecture, illustrated in Figure 13, leverages the strengths of both LSTMs and CNNs but in a novel sequence. The LSTM initially processes the raw spectrogram data, capturing long-term dependencies and temporal patterns. By storing the hidden states in a matrix, the temporal dynamics are converted into a form that a CNN can effectively process. The CNN then applies its convolutional layers to detect spatial patterns and correlations within the matrix of hidden states.

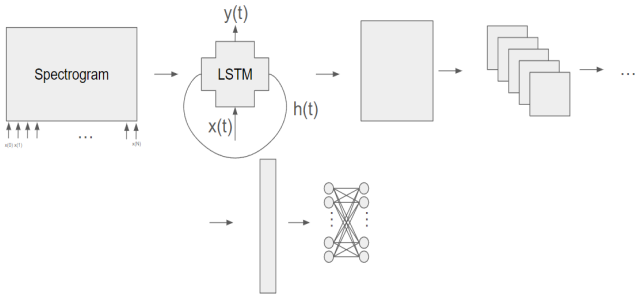

Fig. 13: Structure of a LSTM-CNN Model for Audio Classification. The architecture begins with an LSTM layer with 64 units, 20% dropout, that returns hidden state sequences. The output is then reshaped to adjust data dimensions for convolution. This is followed by a CNN layer with 80 filters and a kernel size of 5, using ReLU activation, and a MaxPooling2D layer with a pool size of 4x4. Another CNN layer with 48 filters and a kernel size of 5, also with ReLU activation, follows, along with a MaxPooling2D layer with a pool size of 5x5. The data is then flattened and passed through a dense layer with 96 units and ReLU activation, with 30% dropout. The final dense layer consists of 10 units with a softmax activation function for classification.

Figure 14 illustrates that overfitting remains, and the performance of this LSTM-CNN architecture is worse compared to the previous two models. One possible explanation for this

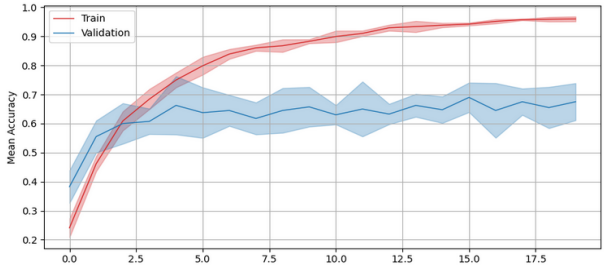

Fig. 14: Average training history of the LSTM-CNN architecture over 20 epochs.

performance decline is the order of the layers. In the LSTM-CNN model, the LSTM layer processes the spectrogram sequences first, capturing temporal dependencies and generating hidden states. These hidden states are then reshaped and processed by the CNN layers. However, this approach might not be optimal for capturing the spatial features present in the spectrograms. By contrast, the CNN-LSTM architectures apply convolutional layers first to extract spatial patterns from the spectrograms. These spatial features are then fed into the LSTM, which models the temporal correlations. This sequence leverages the CNN's ability to handle spatial dependencies and the LSTM's strength in processing temporal sequences, resulting in better overall performance. Thus, the diminished performance of the LSTM-CNN model suggests that the initial extraction of spatial features by CNNs is crucial. Convolutional layers are designed to capture local patterns and structures in the data, which can then be more effectively modeled over time by LSTM layers. Reversing this order may hinder the model's ability to fully understand the spatial characteristics of the input, leading to suboptimal performance in classification tasks.

G. CNN Autoencoder

Autoencoders are used for our task because they effectively learn important features from the data in an unsupervised manner. This approach simplifies the high-dimensional spectrograms, highlighting relevant patterns and reducing noise without needing labeled data.

The autoencoder encodes the input spectrograms into a lower-dimensional latent space and then decodes them back to their original form. During this process, the autoencoder captures essential characteristics of the audio data, providing a compact, informative representation of the spectrograms. These representations serve as inputs to the supervised classifier.

By employing autoencoders, it is possible to address challenges related to the high dimensionality and complexity of spectrograms, making the classification task more manageable. The unsupervised learning phase of the autoencoder may allow the utilization of large amounts of unlabeled data to pretrain the model, which is particularly beneficial when labeled data is limited or expensive to obtain.

Overall, the use of autoencoders aims to provide more discriminative and compact features, expected to improve the efficiency and accuracy of the classifier.

The first architecture employed is an autoencoder utilizing Convolutional Neural Network, as shown in Figure 15. This approach is particularly advantageous for capturing spatial correlations within data like spectrograms. CNNs are well-suited for tasks where spatial relationships are important, as they operate on local input patches and learn hierarchical representations through convolutional and pooling layers. In the context of spectrograms, which exhibit significant spatial correlation between adjacent frequency and time bins, CNNs can effectively capture these dependencies.

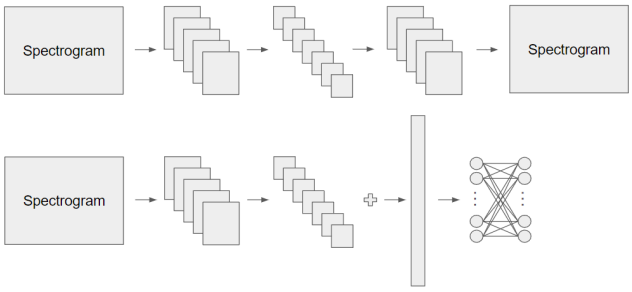

Fig. 15: The autoencoder architecture (figure above) consists of a series of convolutional and transposed convolutional layers designed to encode and decode the input spectrograms. Initially, it uses two Conv2D layers, each with 16 filters and a (2,3) kernel size, followed by two Conv2D layers with 32 filters and a (2,3) kernel size, where the second Conv2D layer uses strides of (2,3). Next, there are two Conv2D layers with 64 filters and a kernel size of 3, with the first one using strides of 3. A MaxPooling2D layer with a pool size and strides of (2,2) follows. In the decoding phase, an UpSampling2D layer with a size of (2,2) is applied, followed by two Conv2DTranspose layers with 64 filters and a kernel size of 3. Then, there are two Conv2DTranspose layers with 32 filters and a (2,3) kernel size, with the second one using strides of (2,3). Finally, two Conv2DTranspose layers with 16 filters and a (2,3) kernel size are followed by a Conv2DTranspose layer with a single filter and a (2,3) kernel size, using tanh activation. The trained encoder serves as the initial layer for the classifier (scheme below), which consists of a feedforward neural network (FNN). This FNN includes two hidden dense layers with 256 and 128 neurons respectively. The output layer of the classifier comprises 10 neurons with softmax activation for classification.

As shown in Figure 16, the training process of the CNN autoencoder model demonstrates convergence after only a few epochs. This rapid convergence suggests that the model may be overfitting to the training data, as it quickly adapts to the training set but potentially at the cost of generalization to new, unseen data. Despite the signs of overfitting, the CNN autoencoder still achieves a validation accuracy that is significantly higher than that of the benchmark FNN model. This improved performance indicates that the CNN autoencoder is more effective at extracting relevant features from the data.

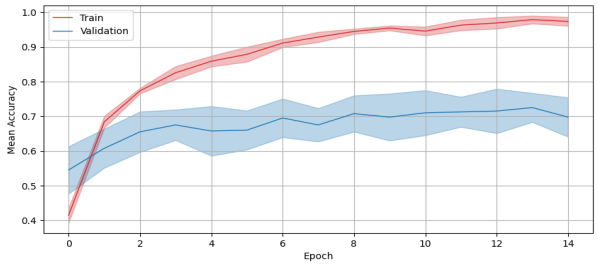

Fig. 16: The training history for the CNN Autoencoder classifier over 15 epochs.

H. Gated Recurrent Unit (GRU) Autoencoder

Another possible approach involves implementing a sequence-to-sequence autoencoder using Gated Recurrent Unit (GRU) layers, as shown in Figure 17. Mel spectrograms, which represent the frequency content of audio signals over time, are inherently sequential data. By leveraging GRU-based autoencoders, we can effectively capture and encode the temporal dependencies within these spectrograms. GRUs are particularly advantageous due to their ability to retain long-term dependencies in sequential data while mitigating the vanishing gradient problem, which can hinder the training of traditional recurrent neural networks (RNNs).

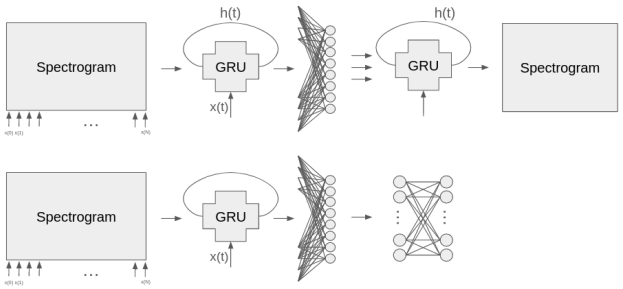

Fig. 17: The architecture comprises a sequence-to-sequence model using Gated Recurrent Unit (GRU) layers and dense connections (scheme above). The encoder section includes GRU layers with 256 and 128 units respectively, with the latter followed by a dense layer of 128 units and tanh activation. A RepeatVector operation prepares the encoder output for the decoder, which consists of GRU layers (128 and 256 units) configured to return sequences. The final TimeDistributed Dense layer with tanh activation operates independently on each time step, facilitating sequence-based tasks. The trained encoder serves as the initial layer for the classifier (scheme below), which consists of a feedforward neural network (FNN). This FNN includes two hidden dense layers with 128 and 64 neurons, respectively. The output layer of the classifier comprises 10 neurons with softmax activation for classification.

As shown in Figure 18, the GRU autoencoder achieves performance on the test set that is comparable to the CNN autoencoder, albeit slightly lower. However, one notable advantage of the GRU autoencoder is its reduced overfitting behavior

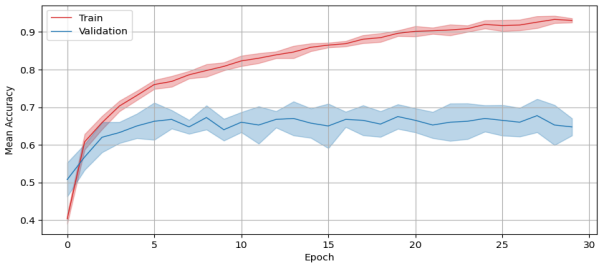

Fig. 18: The training history for the GRU Autoencoder classifier over 30 epochs.

compared to the CNN autoencoder. In the case of the CNN autoencoder, overfitting is more pronounced, which can lead to decreased generalization ability on new data. In comparison, a dense FNN lacks the specialized mechanisms to capture temporal or spatial features effectively. Dense FNNs treat input data as independent and identically distributed, which limits their ability to model sequential or spatial dependencies. As a result, they generally perform worse on tasks where such dependencies are crucial. Overall, the results suggest that for this specific task, leveraging temporal and/or spatial features using GRU or CNN autoencoders leads to better performance compared to solely applying a dense feedforward neural network.

V. Results

In this section, we present the results, shown in Table 1, and the average accuracies for each class in Table 2.

Model	Validation Accuracy	Trainable Params
FNN	0.53 ± 0.06	24,705,194
CNN	0.73 ± 0.08	1,257,198
CNN (GlobalPooling)	0.80 ± 0.01	41,410
LSTM	0.69 ± 0.04	36,870,986
CNN-RF-LSTM	0.80 ± 0.05	640,202
CNN-TD-LSTM	0.76 ± 0.06	557,714
LSTM-CNN	0.68 ± 0.06	203,898
CNN AE	/	161,905
CNN AE clf	0.70 ± 0.06	1,656,458
GRU AE	/	889,600
GRU AE clf	0.65 ± 0.02	25,418

TABLE 1: Averaged validation accuracy over 5-fold cross-validation and trainable parameters of different models.

From the analysis of confusion matrices across various sound classification models, distinct performance patterns emerge across different sound categories: transient/percussive (‘dog’, ‘sneezing’), harmonic (‘crying baby’, ‘rooster’), and noise/soundscape (‘rain’, ‘sea waves’, ‘clock tick’, ‘helicopter’, ‘chainsaw’). Among the noise and soundscape sounds, sea waves emerges consistently as the sound most effectively classified by the majority of the models considered.

The benchmark model, FNN, generally exhibits low accuracies with significant variability in cross-validation, except for a notable exception in its relatively better performance in classifying ‘sneezing’ sounds. This anomaly warrants further investigation into why FNNs excel specifically in this category while struggling with others.

Both CNN architectures demonstrate robust capabilities in classifying harmonic sounds and percussive sounds. However, they encounter challenges in accurately classifying noisy sounds. The second CNN architecture notably enhances accuracies across most classes and shows particular proficiency in classifying noisy sounds such as ‘chainsaw’ and ‘crackling fire’. Despite these improvements, the noisy sounds remain among the most challenging sounds to classify accurately.

The LSTM model demonstrates strong performance in classifying harmonic sounds, indicating its capability to effectively capture the temporal patterns characteristic of these sounds. It also performs well in classifying percussive sounds, although not as consistently as with harmonic sounds. However, the LSTM model struggles more with classifying noise sounds. This discrepancy highlights the LSTM’s strength in handling sequential and rhythmic data, but also its limitations when dealing with more irregular and non-repetitive sound patterns typical of noise and soundscapes. This suggests that while LSTMs are well-suited for tasks involving clear temporal dependencies, they might benefit from being combined with other architectures, such as CNNs, to improve performance on more complex and diverse sound categories.

In fact, the combination of CNN and LSTM performs significantly better than using LSTM alone. From Table 2, it is evident that the LSTM-CNN architecture achieves superior performance. Specifically, this architecture excels in classifying harmonic and percussive sounds, while also showing improved accuracy in classifying certain noise sounds like ‘crackling fire’ and ‘chainsaw’.

The architecture with feature reshaping, where a single LSTM is applied to the sequences output by the CNN, demonstrates notably higher accuracies. This suggests that the integration of CNN and LSTM allows the model to capture both spatial and temporal features more effectively. On the other hand, the architecture employing time-distributed LSTMs shows comparatively lower accuracies overall. This could be due to the increased complexity that comes with distributing the convoluted images to the same LSTM module, which might dilute its effectiveness in learning distinct features from the data. Additionally, the main difference is that the reshaped features CNN-LSTM model takes advantage of the whole sequence of encodings while the TimeDistributed model uses only the last hidden state which, in practice, seems like it is carrying less useful information for the classification. The combination of CNN and LSTM, particularly with feature reshaping, proves to be a more robust approach for classifying a diverse range of sounds.

Interestingly, by switching the order to LSTM-CNN, we observe that the accuracies on the harmonic and percussion classes remain almost the same, but the performance on the noise classes decreases. This indicates that while the order of LSTM and CNN does not significantly impact the model’s ability to classify harmonic and percussive sounds, it does affect the classification of noise sounds. When the LSTM is

Model	Dog Bark	Rain	Sea Waves	Baby Cry	Clock Tick	Person Sneeze	Helicopter	Chainsaw	Rooster	Fire Crackling
FNN	47.5 ± 12.25	47.5 ± 18.37	60.00 ± 22.91	45.00 ± 16.96	25.00 ± 19.36	80.00 ± 15.00	30.00 ± 25.74	55.00 ± 16.96	70.00 ± 25.74	67.50 ± 35.00
CNN	80.00 ± 15.0	62.50 ± 15.81	90.00 ± 9.35	75.00 ± 22.36	57.50 ± 25.74	82.50 ± 10.00	65.00 ± 27.84	57.5 ± 12.75	92.50 ± 6.12	72.50 ± 21.51
CNN (GP)	70.00 ± 10.00	72.50 ± 25.50	82.50 ± 12.75	90.00 ± 9.35	62.50 ± 7.91	90.00 ± 14.58	70.00 ± 32.21	82.50 ± 15.00	92.50 ± 10.00	82.00 ± 9.35
LSTM	75.00 ± 11.18	65.00 ± 12.25	72.5 ± 9.35	92.50 ± 10.00	50.00 ± 13.69	80.00 ± 16.96	50.00 ± 20.92	52.50 ± 14.58	90.00 ± 12.25	62.50 ± 22.36
CNN-RF-LSTM	82.50 ± 16.96	70.00 ± 18.71	82.50 ± 18.71	87.50 ± 7.91	57.5 ± 18.71	82.50 ± 12.75	67.50 ± 20.31	92.50 ± 6.12	90.00 ± 9.35	82.50 ± 12.75
CNN-TD-LSTM	75.00 ± 20.92	62.50 ± 20.92	67.50 ± 18.71	92.50 ± 10.00	55.00 ± 12.75	87.50 ± 13.69	62.50 ± 27.39	85.00 ± 5.00	87.50 ± 11.18	85.00 ± 18.37
LSTM-CNN	82.50 ± 19.96	52.50 ± 9.35	82.50 ± 6.12	87.50 ± 19.36	45.00 ± 12.75	82.50 ± 12.75	50 ± 20.92	50.00 ± 15.81	77.50 ± 16.58	65.00 ± 28.94
CNN AE	65.00 ± 9.35	62.50 ± 25.00	75.00 ± 11.18	77.5 ± 16.58	52.5 ± 26.69	75.00 ± 17.68	57.50 ± 28.06	72.50 ± 12.25	92.50 ± 10.00	67.50 ± 12.75
GRU AE	77.50 ± 14.58	52.50 ± 21.51	77.50 ± 18.37	80.00 ± 18.71	32.50 ± 6.12	80.00 ± 16.96	52.50 ± 28.94	60 ± 12.25	80.00 ± 16.96	55.00 ± 32.21

TABLE 2: Accuracy of the Models with respect to the different Classes of the ESC-10 Dataset

applied before the CNN, the model might lose some of the temporal dependencies that are better captured by the CNN first. This loss is particularly evident for noisy classes, which may rely more on capturing intricate temporal and spatial features than the CNN can handle. Therefore, the order of integrating these two architectures plays a crucial role in optimizing performance across different types of sounds, with the CNN-LSTM combination providing a more balanced and effective approach for diverse sound classification tasks.

Regarding the models involving an autoencoder training, it is interesting to note that for all considered cases, the best accuracies are achieved for the classes 'sea waves', 'sneezing', and 'rooster'. This pattern suggests that these specific sounds have distinctive features that the autoencoders can effectively capture and encode, making them easier to classify accurately.

The CNN autoencoder better captures the relevant features, resulting in increased accuracy across all classes compared to the FNN model. However, despite this improvement, the model still suffers from overfitting, as discussed in the previous section. While the CNN autoencoder demonstrates the potential for enhanced feature extraction, its susceptibility to overfitting underlines the need for further refinement and techniques to improve generalization.

Finally, the GRU autoencoder significantly improves classification performance for harmonic and percussion sounds, achieving higher accuracies than the previously discussed autoencoder model. However, it still struggles considerably with noisy sounds. This indicates that while the GRU autoencoder is effective in capturing temporal dependencies and enhancing the recognition of structured sound patterns, it faces challenges in distinguishing between noise/soundscape classes where the temporal and harmonic features are less distinct. This highlights the need for further advancements or complementary techniques to address the complexities associated with noisy sound classification.

VI. Concluding Remarks

From this analysis, the importance of denoising processes becomes evident. Noisy classes consistently posed difficulties across various classification models. The introduction of Dense denoising autoencoder to work cooperatively with the other models could be an interesting extension over our models. Another, more 'rigid' approach, could be the further preprocessing of the Spectrograms by performing PCA directly before the inputs, thus reducing the noise components.

It is observed that classes with clear harmonic content generally resulted in better performance across all models evaluated. Specifically, models such as the CNN-LSTM combination and the GRU-autoencoder demonstrated notable improvements in accuracy for these harmonic-rich classes. Robust feature extraction methods, enhance the capability of models to discern and classify subtle acoustic patterns, ultimately improving overall classification outcomes.

Our biggest challenge was the limited amount of available data, which caused our models to overfit on the training data. In most models, we tried to used techniques to avoid overfit, using regularization, dropout and data augmentation. Especially in the cases where we used unsupervised learning to train autoencoders, the models could greatly benefit from more training data. In the future, it would be very interesting to explore how overfitting on a small dataset can ultimately affect real world model deployment.

In conclusion, integrating different feature extraction techniques into sound classification workflows not only mitigates challenges posed by noise but also optimizes the acoustic classification task.

References

[1] K. J. Piczak, "ESC: Dataset for Environmental Sound Classification," in *Proceedings of the 23rd Annual ACM Conference on Multimedia*, pp. 1015–1018, ACM Press.

[2] K. J. Piczak, "ESC: Dataset for Environmental Sound Classification." https://github.com/karolpiczak/ESC-50, 2017.

[3] https://github.com/mariostrbac/environmental-sound-classification.