Data augmentation for speech classification

Emanuele Zangrando
emanuele.zangrando@studenti.unipd.it

1. Introduction

The problem I decided to tackle in this project is the audio classification one.
The main goal is to classify a recorded audio according to the word spoken in it.
Having efficient models for this kind of problem can be important in order to tackle more difficult scenarios (like audio to text transcription) or it can be even useful as it is for single word vocal commands recognition (on smartphones for example). Audio classification has also other application: a recent example is the SARS-CoV-2 detection from the recording of a cough.

This report outlines several tests I decided to do in order to find a good model to tackle this problem. In particular, after finding an appropriate neural network architecture, I focused on studying the effects of the various kind of data augmentation.

2. Datasets

The dataset used in this project is a reduced version of the Tensorflow speech command one, it contains 1600 training examples and 109 validation samples (test samples in the code) . The total number of labels is 8 and the training dataset is perfectly balanced (there are exactly 200 examples for each label). It is made of couples audio-spoken word, where every audio signal (time vs amplitude) is represented as a vector in $\mathbb{R}^{16.000}$.
The first useful preprocessing was to represent the recordings not as a time series, but using the log-Mel spectrogram (as suggested). This representation suggests also that we can use methods coming from image classification.
I also splitted the training set in two parts ($80 - 20\%$ proportion, uniformly in all labels), the first one was used for training and the second one as a validation set for early stopping. The provided validation set was instead used as unbiased test set. In the following I will refer to this set as test set and not to the one provided for kaggle evaluation.

3. Method

The best performances I've been able to get was by using Neural Networks. In particular, I decided to use a CNN on the spectrogram for some reasons:

- the "meaningful part" can be placed in different positions of the recorded audio, and this makes the problem more difficult for a fully connected architecture. Moreover, the interaction between the features tends to be sparse;
- using CNN can significantly reduce the dimensionality of the parameter space with an equal (or even better) performance. This allows also to use deeper models without an exponential growth of the number of weights;

After having tried some shallow architectures for a discriminative model $p_\theta(y = \text{label} \,|\, x = \text{spectrogram})$, I came up with the one represented in Table 1.

Layer type	activation	out shape	# pars
Conv2D, kersize = 4, filters=32	Relu	(29,29,32)	544
MaxPool , pool_size = 3		(9,9,32)	0
Conv2D, kersize = 3,filters = 16	Relu	(7,7,16)	4624
MaxPool, pool_size = 2		(3,3,16)	0
Flatten		144	0
Dropout(p = 0.4)		144	0
Dense	sigmoid	20	2900
Dense	Softmax	8	168
Total parameters			8236

Table 1. Architecture of the main model. Early stopping patience was of 15 epochs on validation loss.

In the analysis I included also a dense architecture and given their simplicity I decided to try also Knn and Svm's , at least as matter of reference of the performance. It worths noticing that in general Knn is not suited for this kind of applications, since one usually wants this kind of classifications to be done in real time.

3.1. Loss for the CNN

Since the label distribution conditioned on the spectrogram is a multinoulli, the most natural choice is to use a softmax activation function in the output layer and consequently the negative log-likelihood as loss. In Tensorflow there's a loss equivalent from an optimization point of view (the cross-entropy), so I used that one.

3.2. Regularization

In order to avoid overfitting, the neural networks had to be regularized. In particular, I regularized the main model by using Dropout and early stopping (1). I decided to use Dropout regularization also to avoid the model to attach to some particular feature for classifying and this hopefully helped in getting a more robust model.
The Dropout idea comes from ensemble methods, in practice we use a random mask to "forget" some nodes and all its connections at every optimization step. Thus, what we are actually optimizing in this context is the expected value of the loss over all masks, $\tilde{J}(\theta) = \mathbb{E}_{\mu}[J_{\mu}(\theta)]$. This in practice allows to train an exponentially big ensemble of neural networks with a reasonable cost.
The optimization algorithm I used (Adam) is a modified version of the minibatch stochastic gradient descent, it exploits the idea of adaptive moment estimation of the loss' gradient [3].

4. Experiments

At first I decided to explore the robustness of the main model with respect to groups of "natural" transformations of the samples. By "natural transformations" I mean transformations from the data manifold to itself that preserve also the labels, and we would like our model $p_\theta(y|x)$ to be (at least approximately) invariant to them.

Geometrically, every group $(f_\lambda)_{\lambda\in\Lambda\subset\mathbb{R}^l}$ of transformations of this kind gives us an l dimensional piece of data manifold around each point all contained in the same label region (parametrized by the map $\lambda \mapsto f_\lambda(x)$). These transformations usually come from an a priori knowledge about the nature of the data representation.

Once we decide which relevant family to use, we just need to enforce in some way that our classifier should be approximately constant along each one of the previously mentioned pieces of manifold. This can be done in different ways :

- the easiest one is probably data augmentation;
- another idea is to add another term to the loss that penalizes neural network parameters that do not produce a classification almost constant on these patches. If the family of parameters $\lambda \in \Lambda$ is finite, we can add a penalty term like $\sum_{\lambda\in\Lambda}\sum_{i=1}^{n} ||\frac{\partial}{\partial\lambda}\vec{p}_\theta(\bullet\,|\,f_\lambda(x^{(i)}))||_F^2$.

In this project I explored the first possibility.
As group of transformations I chose to study time translations, since they can be performed directly on the spectrogram.
In the other experiments I checked if the model's performance was improving by using other kind of data augmentations, in particular I tried different kinds of noise adding and frequency masking.

4.1. Time translation augmenting

In this experiment I decided to check the robustness of various models with respect to time translations. The first thing I did was to evaluate all the models on a translated version of the test set for various translation parameters (t is the amount of pixels translation, negative for translations to the left) and I plotted the accuracies against the amount of translation (Figure 1). In this plot we can notice that the convolutional architecture seems to be the most robust one with respect to translations.
After that I trained all the models again by using an augmented training set containing also slightly translated samples and produced the same plot as before to compare the results (Figure 2).
By comparing the two plots we can notice that the injection of these new samples enforced the robustness with respect to translations and it also improved the performance in all the models under consideration. The best accuracy had been reached by CNN in both scenarios.
For big translations the accuracy drops pretty fast, this happens probably because the majority of information had been translated out of the window.
The best test accuracy I was able to attain (95.4%) was with this kind of augmentation and the CNN architecure represented in (1).

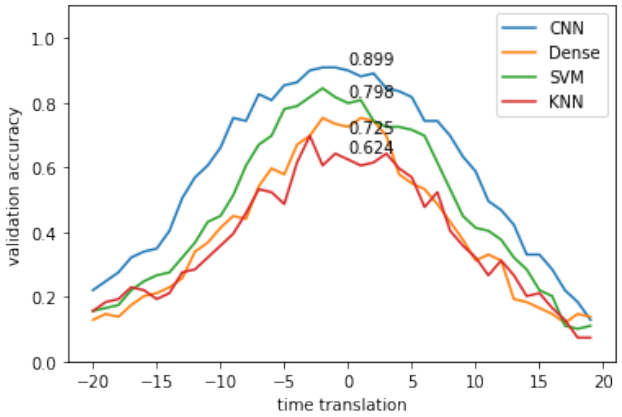

Figure 1. test accuracy of time translated dataset vs amount of translation for the various models. The training set has not been augmented with translated images. The highlighted values are the one attained on the non translated test set.

4.2. Noise injection an frequency masking

Often in real audios there's background noise present, so another important thing to check is the robustness with respect to it. Noise adding unfortunately is more complicated than time translations, if one wants to add a "real" back-

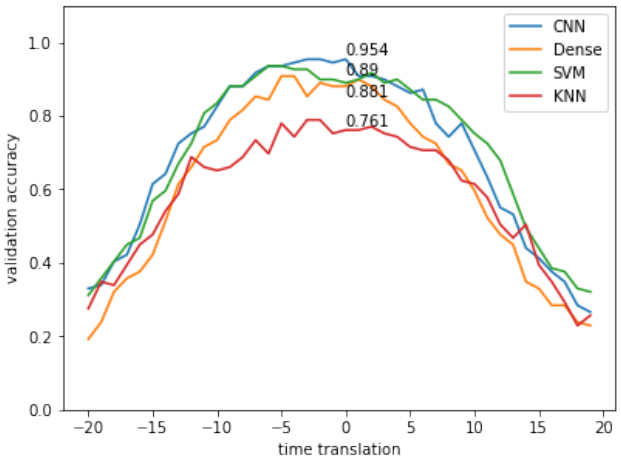

Figure 2. test accuracy of time translated dataset vs amount of translation for the various models after training on the augmented dataset. The highlighted values are the one attained on the non translated test set.

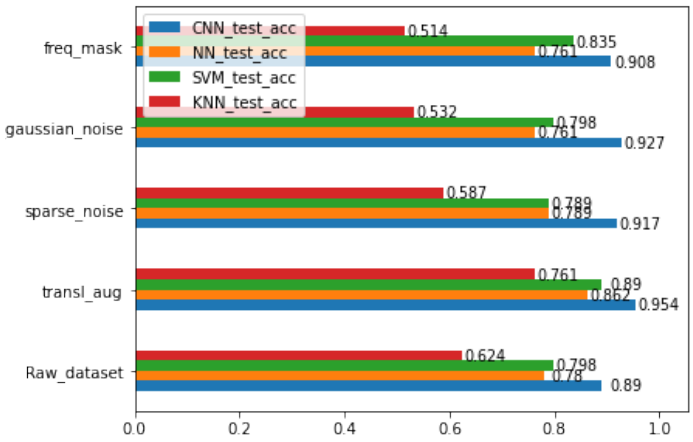

Figure 3. test accuracy of all models after training on different augmented training sets. The best accuracy had been reached by the Convolutional neural net on the dataset augmented with time translated samples.

ground noise it's required to add it to the time signal and then recompute the spectrogram.
In this experiment I instead checked the robustness with respect to noise injected directly on the spectrogram, even if in this way the meaning of this noise in the "audio sense" is lost.
I injected a sparsified gaussian noise in the spectrograms in a way that the expected Frobenius norm of the noise was of order $\mathcal{O}(1)$ (I had to compensate concentration of measure).
The second kind of noise I tried was salt and pepper: I randomly selected a low percentage p of entries and masked them. Another strategy one can use is adding a noise layer in the neural network instead of doing batch data augmentation.
In both scenarios the test performance of the CNN improved (92.66% and 91.74% test accuracy for salt and pepper and gaussian noises respectively) but it wasn't able to overcome the translation-augmented version.
Frequency masking instead attained a 90.83% of test accuracy, so the improvement was not that significant. In Figure (3) I reported all the test performances.

4.3. Final evaluation on the augmented dataset

Finally, I tried also to train the model by using all these augmentations together. This model performed 93.58% accuracy on test set (so worse than the one with only translation augmentation), but all the other measures were better (test/validation accuracies and test/train/validation losses). One can prefer this model with respect to the other for this higher homogeneity of accuracies in all three datasets.

5. Conclusion

These experiments are meant to show that the right data augmentation can improve the performance of a model and in our case the robustness with respect to time translations. The first kind of augmentation performed better because it is intrinsically different from the others: the first one actually gives us meaningful local directions in which we want our classifier to be constant. Gaussian noise instead enforces the classifier to be "smoother" and masking augmentation forces the model to not focus on some particular restricted subset of entries of the spectrogram (just like Dropout does), thus it is reasonable that these last two improved less the performance.
Apart from the main model, probably the second best choice given the simplicity and the performances in (3) would be SVMs (with Radial basis kernel) on the translation-augmented dataset.
Lastly, NN is performing worse than CNN in all scenarios despite its bigger number of parameters.

5.1. Further questions

Given the results on the translation augmented dataset it would be interesting to study other families of interesting data transformations, for example time dilatation or others based on frequency.
It would be also interesting to study how the regularization approach works, since it has the advantage of not actually augmenting the dataset (but it has the disadvantage of having to calculate the jacobian of the NN with respect to the input for regularization).
Finally, it would be interesting to see the performance on noisy samples after data augmenting with real audio noise, cause test samples seemed to be noiseless.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[4] Pete Warden. Speech commands: A public dataset for single-word speech recognition. 2017.