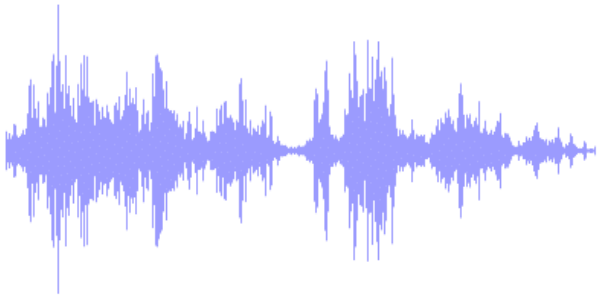
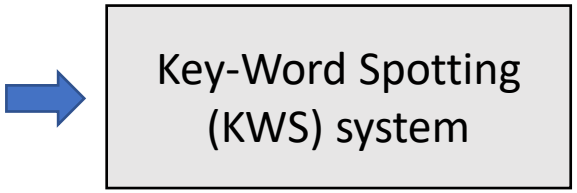
Analysis of keyword spotting techniques

Simone Ceccato - simone.ceccato.1@studenti.unipd.it
Federico Venturini - federico.venturini@studenti.unipd.it

The key-word spotting task



Input raw signal



Keyword 1 in position x_1

Keyword 2 in position x_2

…

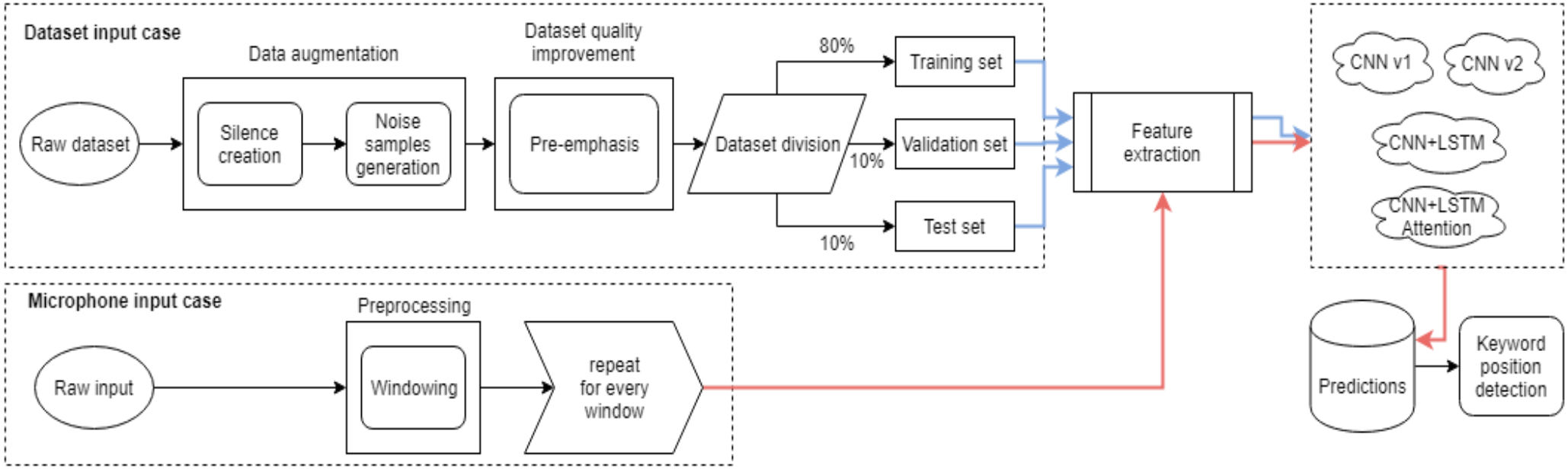
Keyword *n* in position x_n

So, let's see a possible implementation

Python implementation

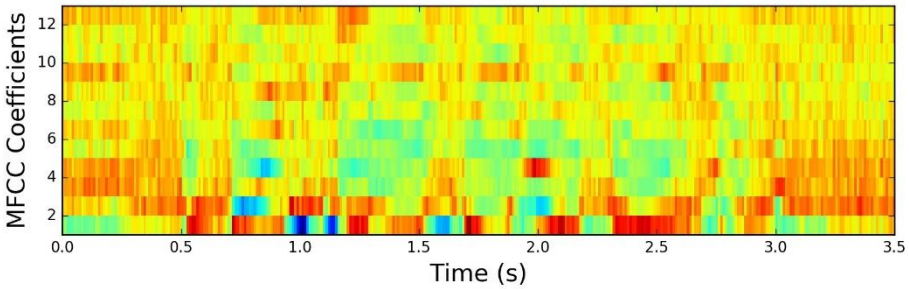
The processing pipeline

How we achieved such result?



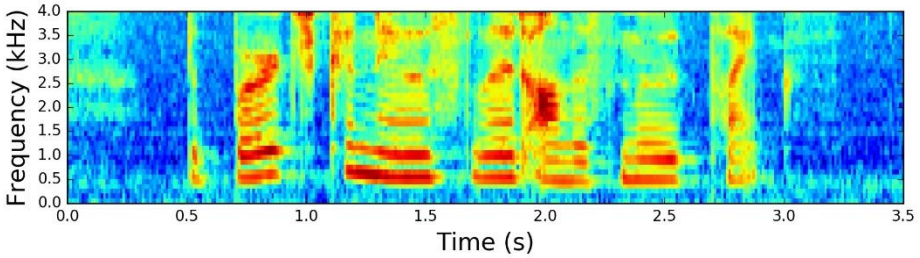
Feature extraction

Mel-frequency cepstral coefficients (MFCC)

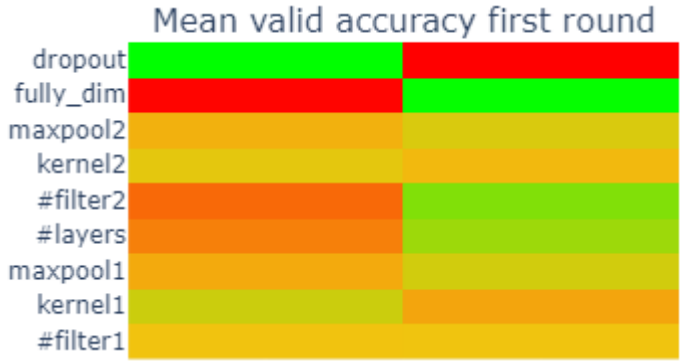


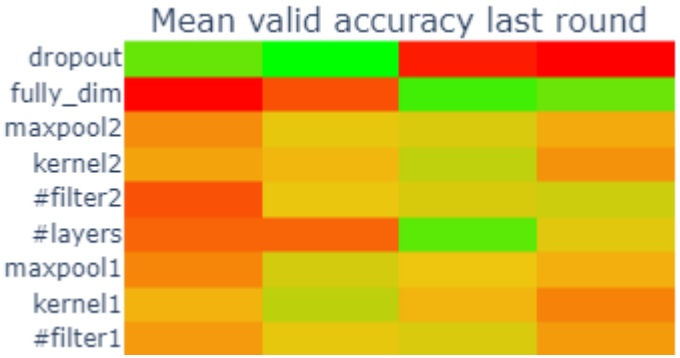
MFCC with Delta and Delta-Delta

Log - Filterbanks



Automatic hyperparameters initialization: IAHOS





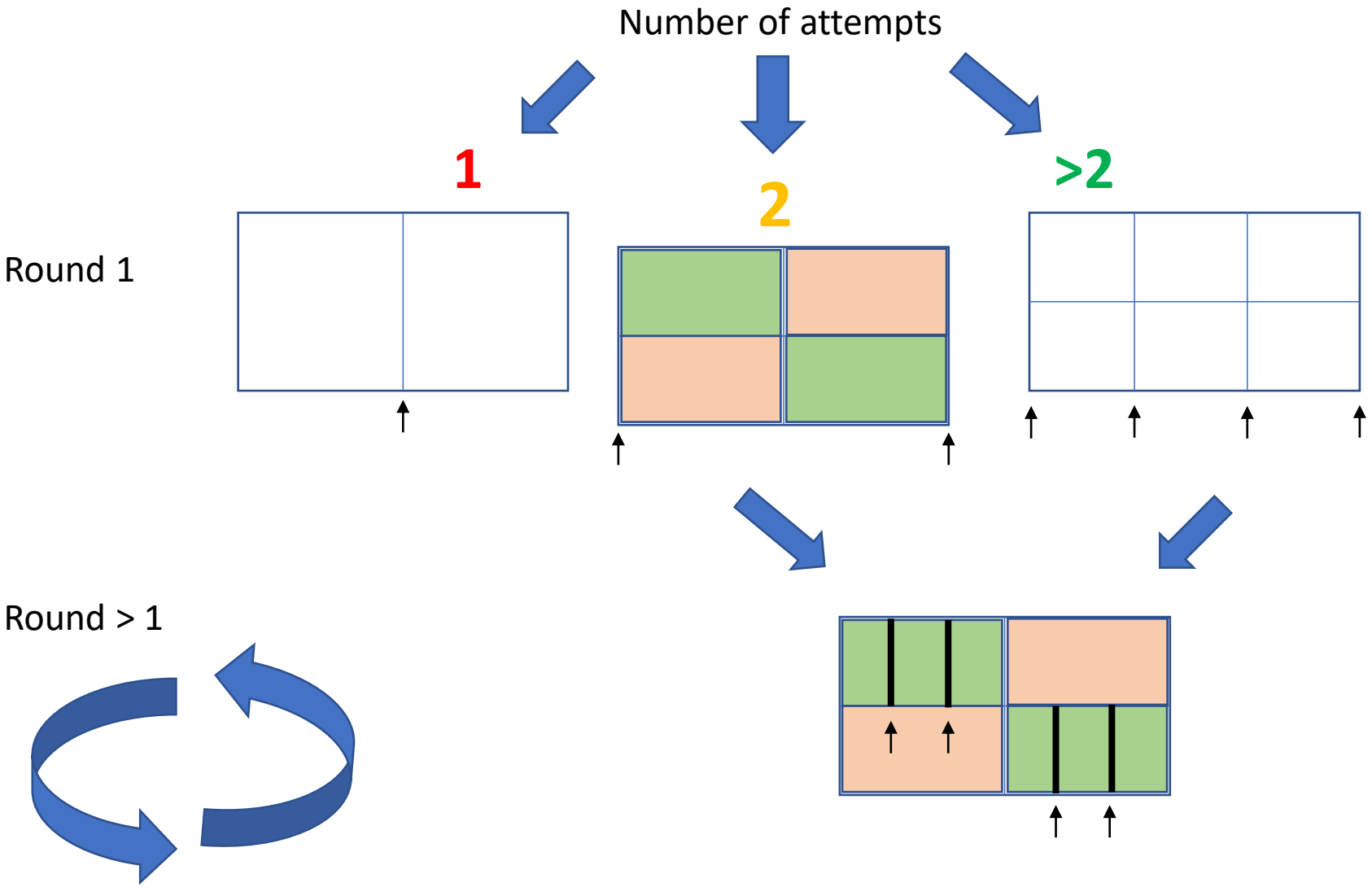
Selection of the external limits of the possible values of the hyperparameters

Creation of the hyperparams table

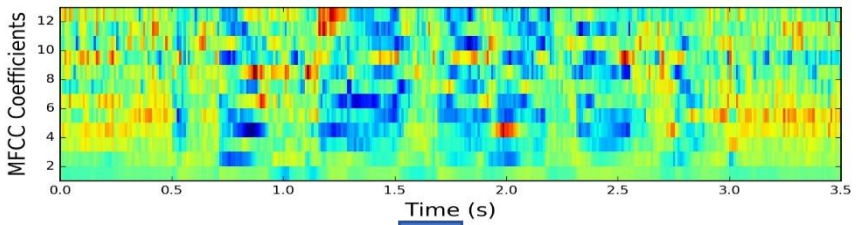
Computation of the training and validation accuracy on all the possible combinations of the new values table for 1 epoch with adam optimizer.

Turn to step 1 but with new limits values that are selected around the best region found until now on each hyperparameter.

Automatic hyperparameters initialization: IAHOS



Models: CNN version 1



= hyperparams studied by IAHOS

Conv2D layer (IAHOS on filters, kernel_size2)

Dropout layer (IAHOS on rate)

MaxPooling2D layer (IAHOS on size)

Conv2D layer (IAHOS on filters, kernel_size2)

Dropout layer

MaxPooling2D layer (IAHOS on size)

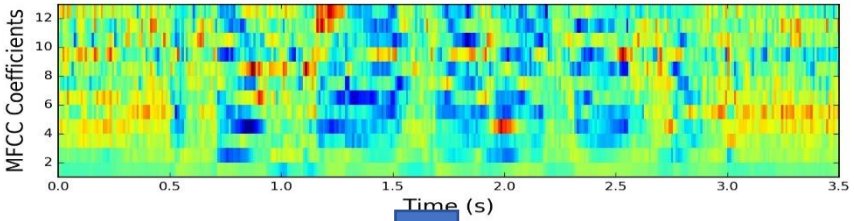
i+1 times

Flatten layer

Fully connected (IAHOS on number of neurons)

Fully connected (softmax activation)

Models: CNN version 2



= hyperparams studied by IAHOS

Conv2D layer (IAHOS on filters, kernel_size1, kernel_size2)

Dropout layer (IAHOS on rate)

MaxPooling2D layer (IAHOS on size1, size2)

Conv2D layer (IAHOS on filters, kernel_size1)

Dropout layer

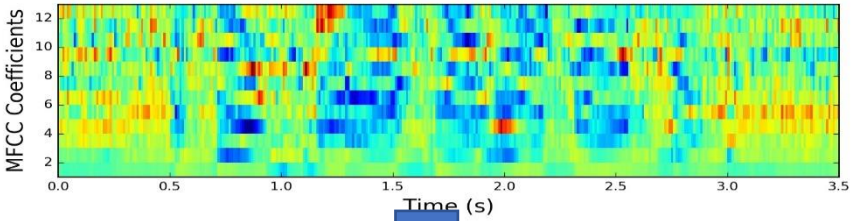
MaxPooling2D layer

Flatten layer

Fully connected (IAHOS on number of neurons)

Fully connected (softmax activation)

Models: CNN + LSTM



Conv2D layer (10, (5, 1))

Batch normalization layer ()

Conv2D layer (1, (5,1)

Batch normalization layer ()

Lamba layer for squeezing

Bidirectional(LSTM(64))

Bidirectional(LSTM(64))

Flatten

laye

Dense(64, relu) + Dense(32, linear) + Dense(n_classes, softmax)

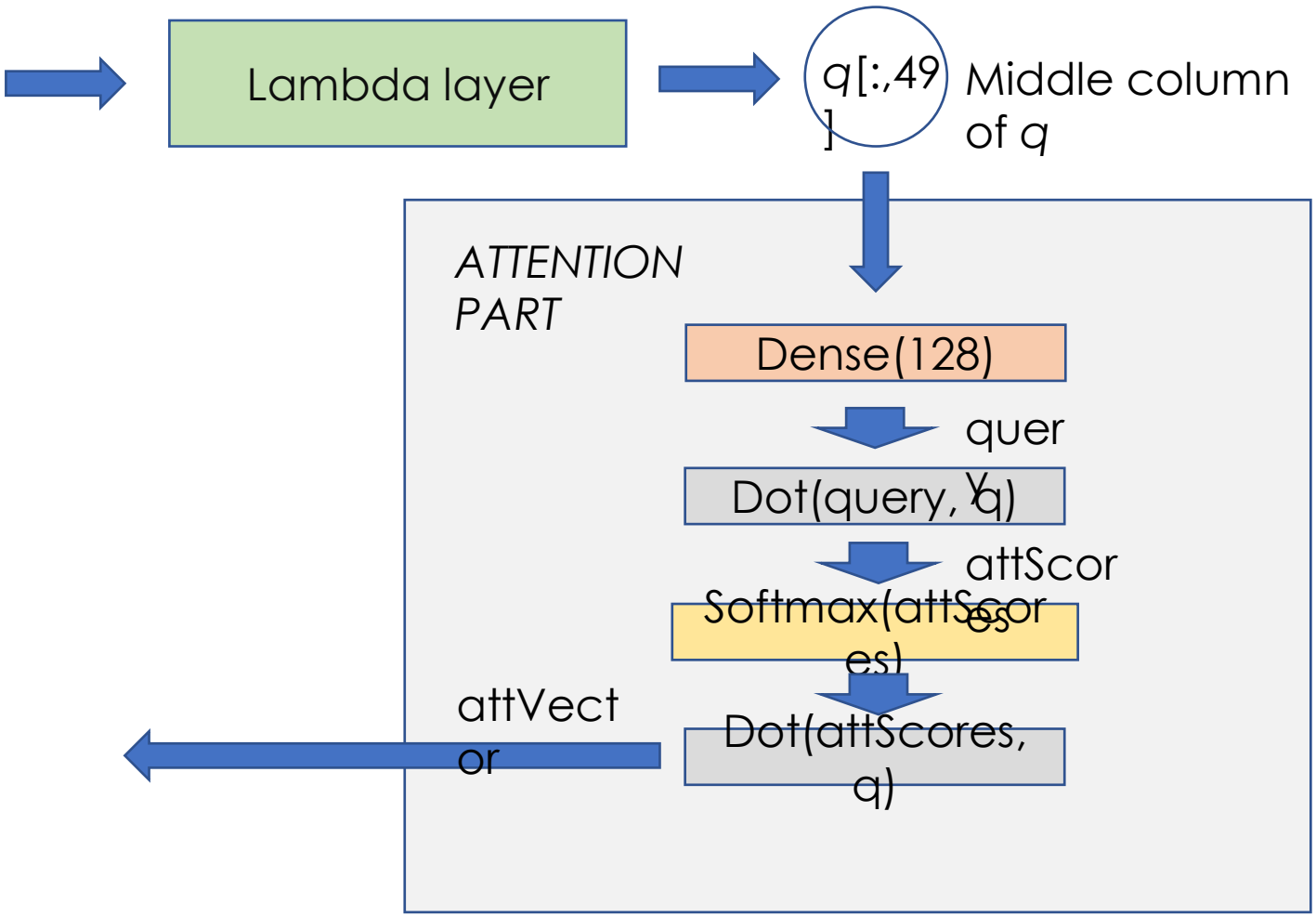
Models: CNN + LSTM with attention

Same architecture as the previous network but:

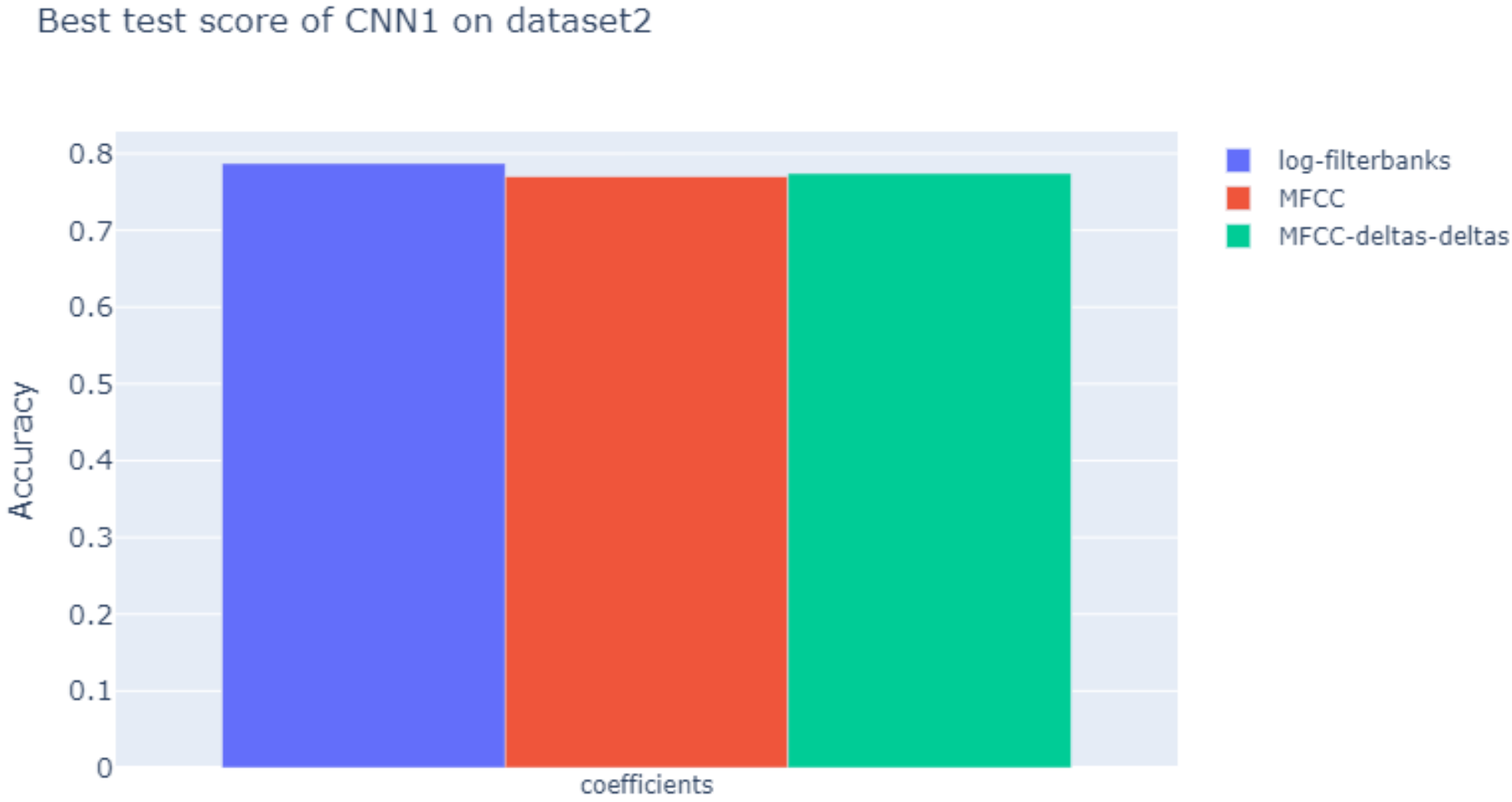
Output of the second Bidirectional LSTM *q*

Final part of the previous network

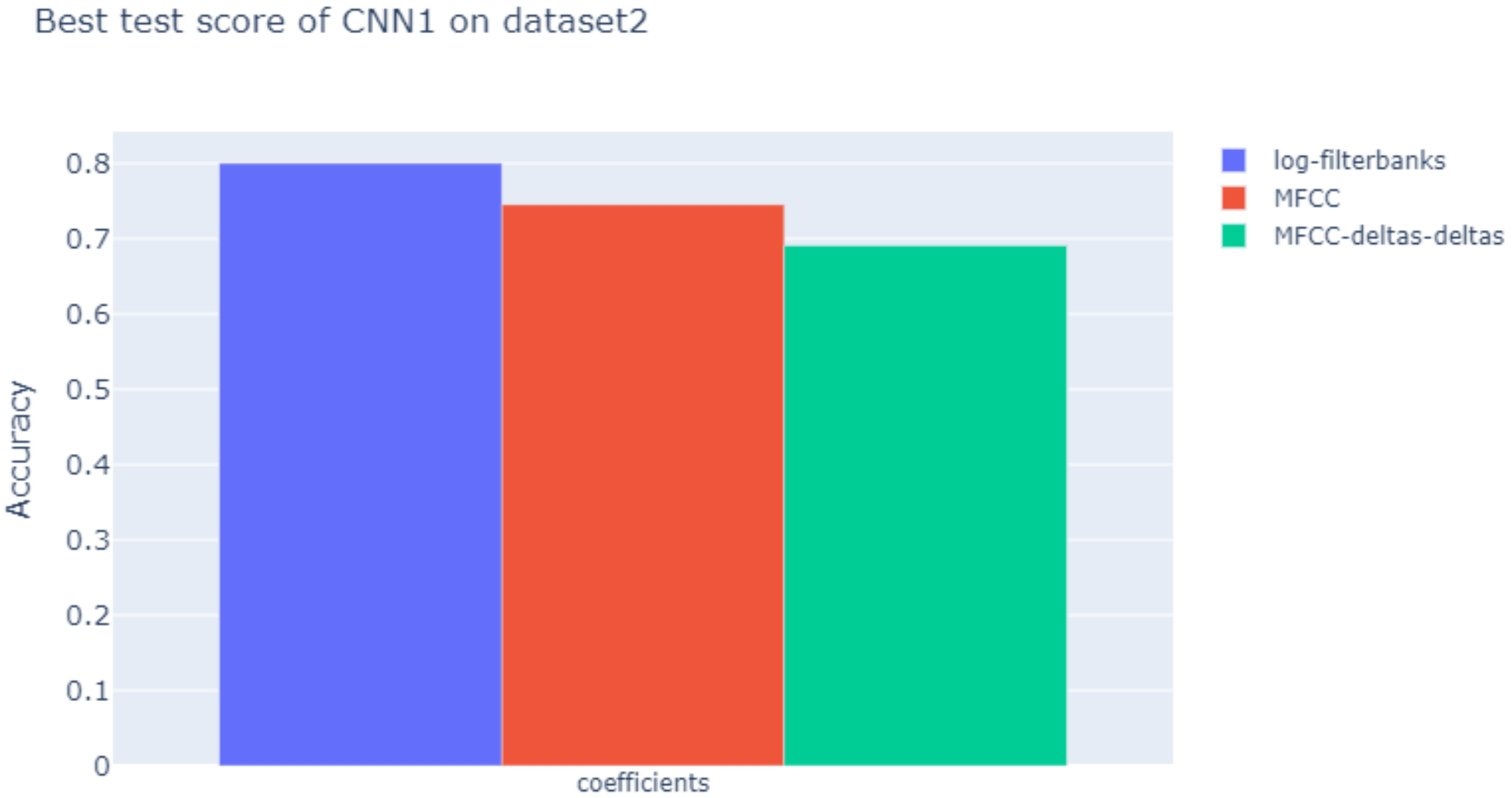
Dense(64, relu) + Dense(32, linear) + Dense(n_classes, softmax)



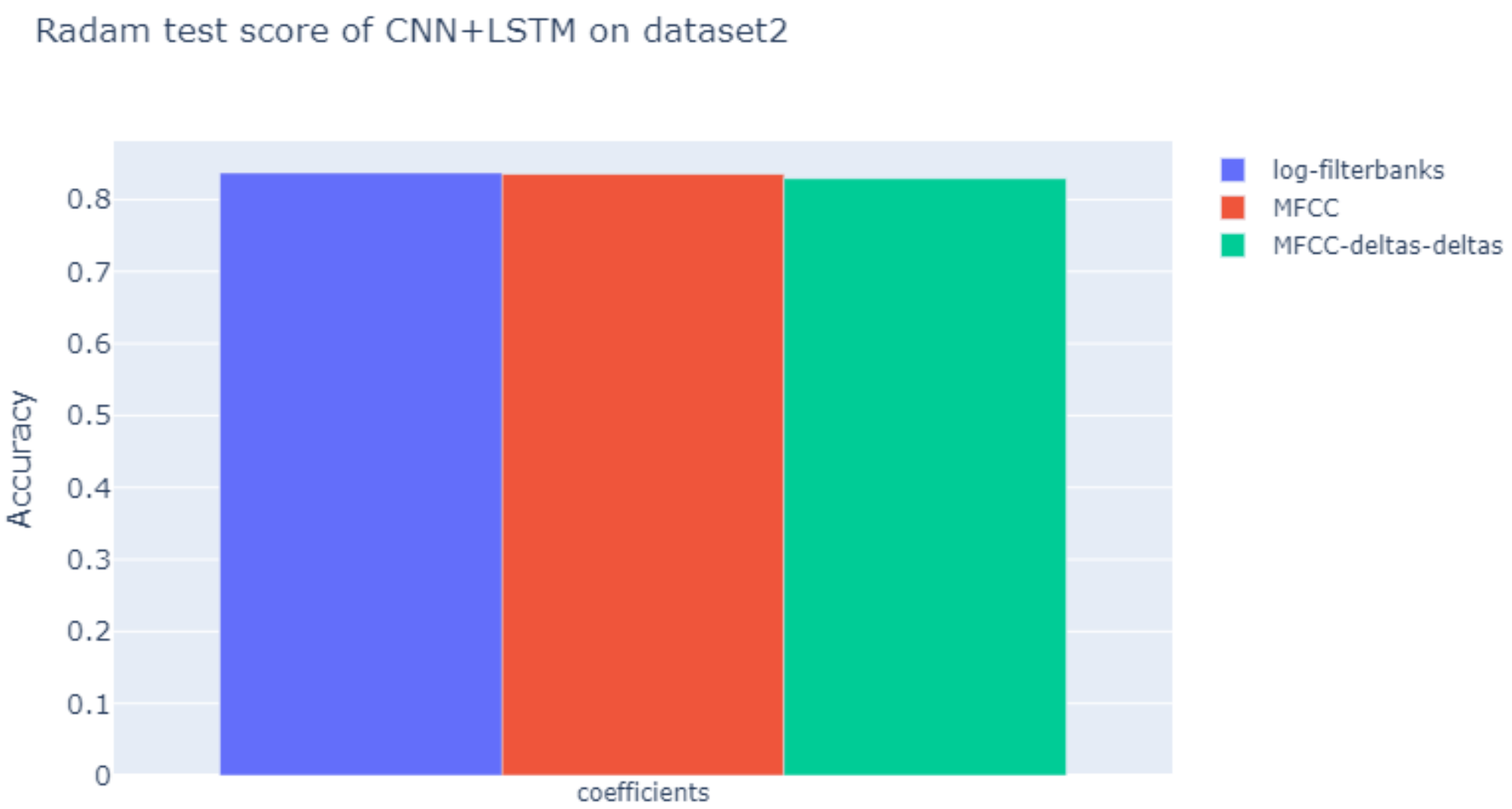
Results: CNN1



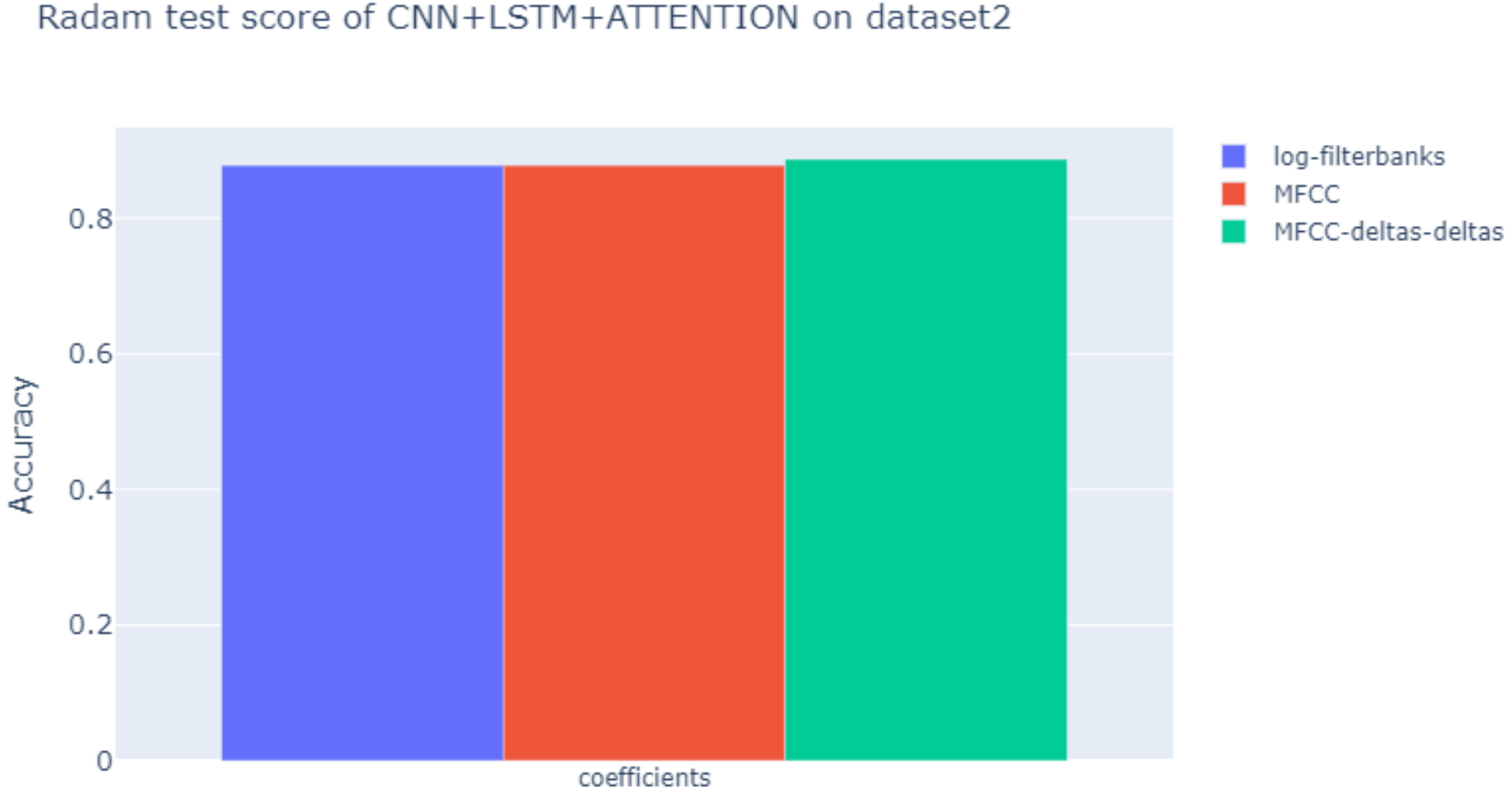
Results: CNN2



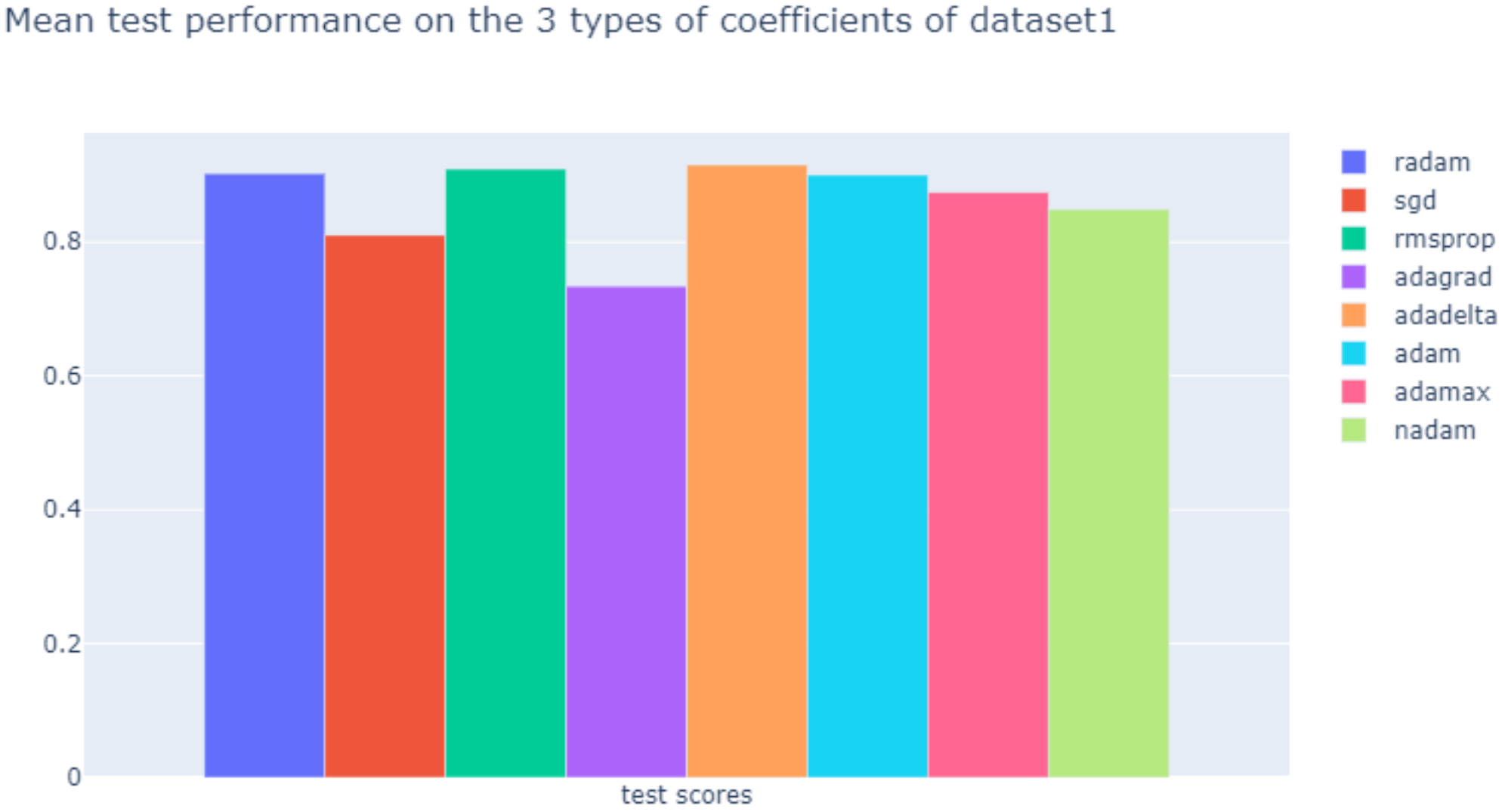
Results: CNN+LSTM



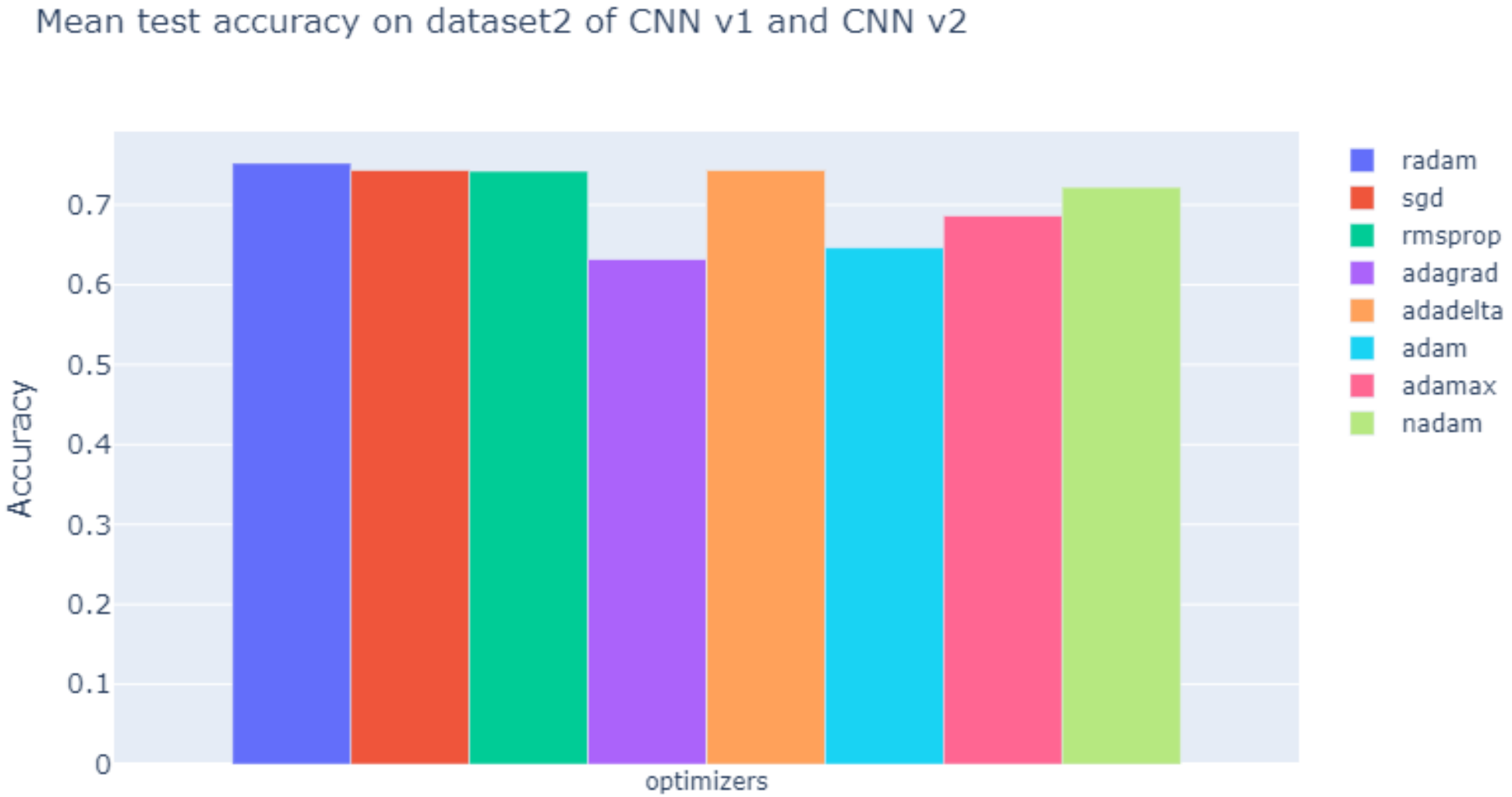
Results: CNN+LSTM+ATTENTION



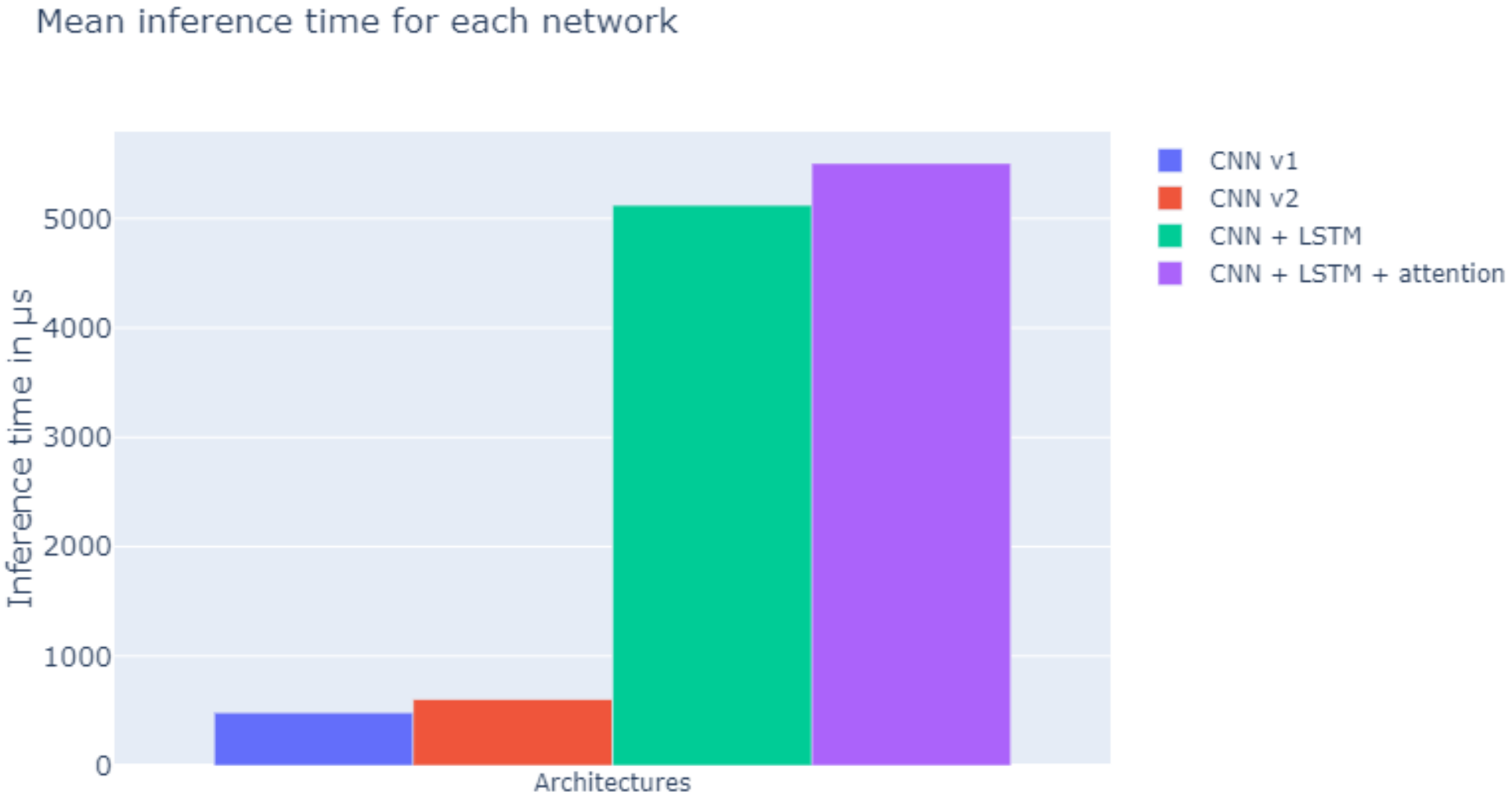
Results: the optimizers



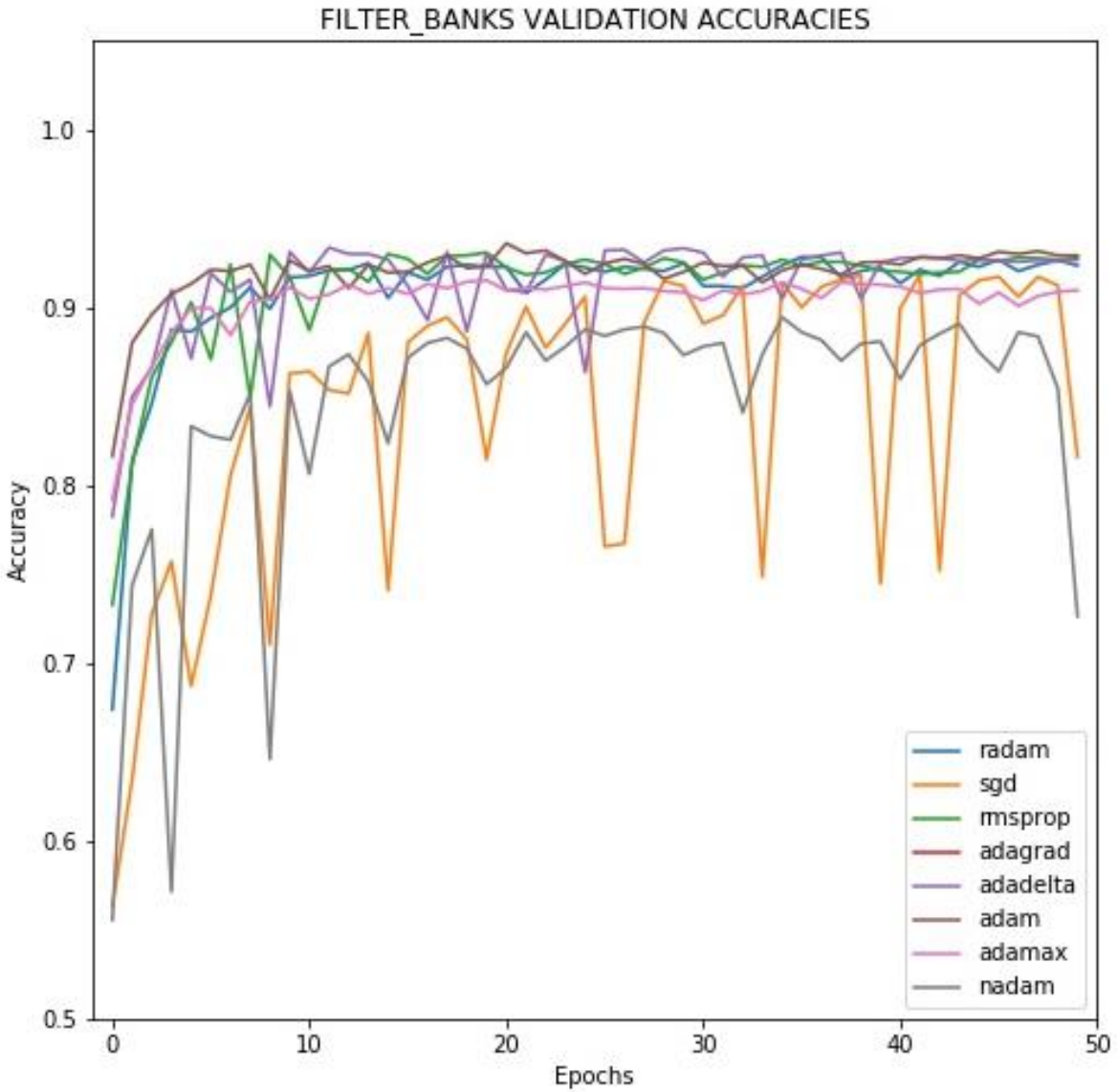
Results: the optimizers



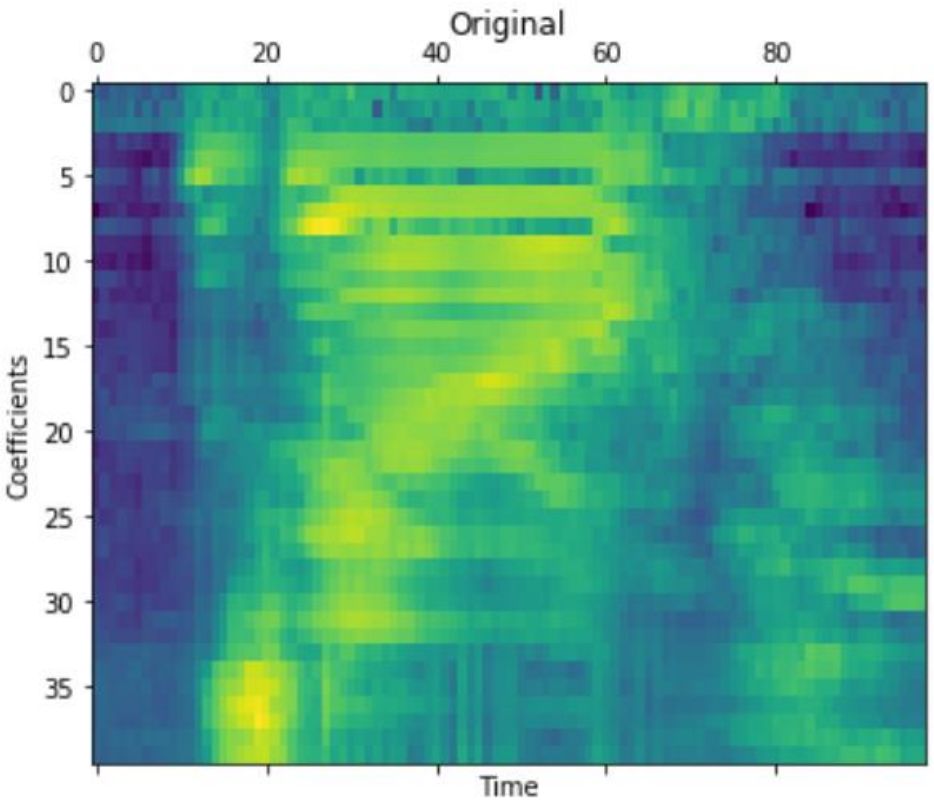
Results: inference time

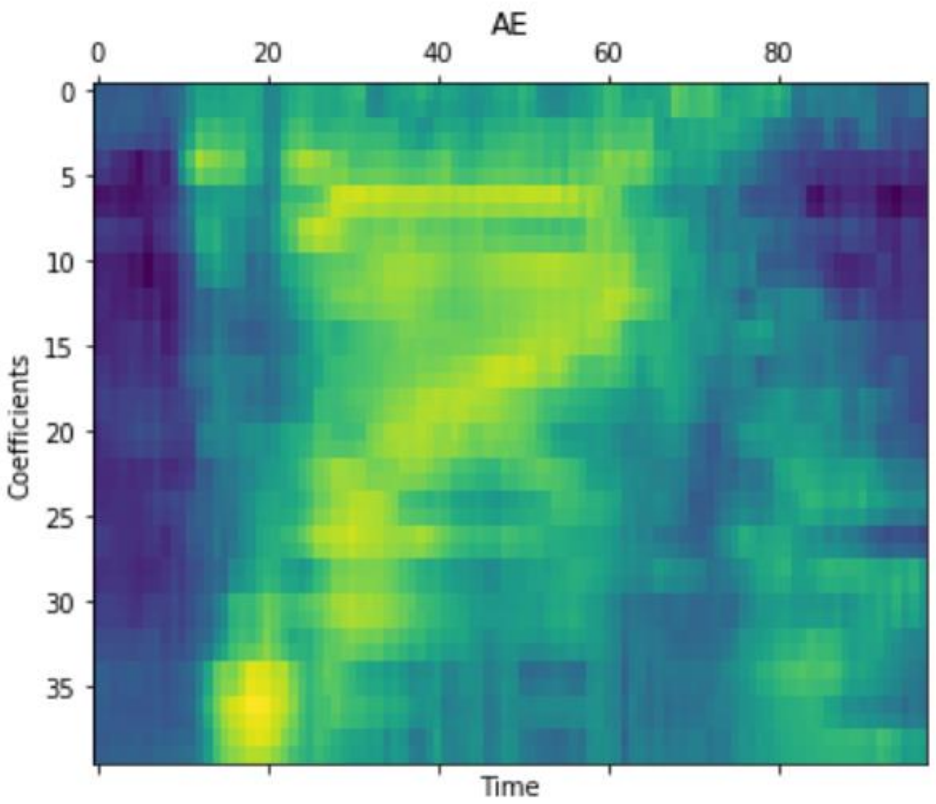


Results: early stopping and overfitting (5 classes dataset, CNN2)



What didn't work: Autoencoder





Conclusions and final remarks

What we learned

- Small accuracy difference between MFCC and MFCC-Deltas/Log Filterbanks.
- Use ML techniques and dig a little bit in the DeepLearning.
- Tradeoff accuracy – inference time.
- LSTM based networks robust to the 3 types of coefficients; instead the first 2 networks showed that LogFilterbanks outperformed the other 2 type, especially with 30 classes.
- The benefits of RNNs and their building blocks.
- The importance of the attention mechanism.
- Goodness of Radam

What we can learn

- Make the autoencoder learn from data, trying to use inside it some of the features used in the last 2 networks: attention, LSTM, bidirectional RNNs.
- Construct a Denoising Autoencoder.
- Exploit also the SSC coefficients.
- Apply IAHOS also on the last 2 networks and study it deeply, discovering its possible real power on clusters.
- Try Ranger=Radam+Lookahead

Thanks for your '*attention*'!