FoCV Report

Bartłomiej Krzepkowski

Task Overview

The entire task was divided into **4 main stages**:

1. **Dataset Design**
2. **Conversion to Spectrograms**
3. **Training and Validation**
4. **Fine-tuning on an Additional Person Labeled "allowed"**

---

1. Dataset Design

- **Merged the train and test folders** from `/VOiCES_devkit/distant-16k/speech`.
- **Selected 66 speakers** in total.
- **Balanced by gender**, then **assigned 6 speakers** to the **"allowed"** label for voice recognition. The remaining speakers were labeled as **"not allowed"**.
- For each group ("allowed" and "not allowed"):
    - **Split data into training, validation, and test sets.**
    - For each speaker, one of their two file types went into the training set; the rest were split between validation and test.
    - This ensures **no data leakage**: **the same text never appears in both training and validation/testing phases**.

- **Result**: Well-designed **training, validation, and test sets**.

---

2. Conversion to Spectrograms

The audio data undergoes the following processing steps:

1. **Load each WAV file** using `torchaudio.load`.
2. If stereo (**2 channels**), **convert to mono** by averaging.
3. **Normalize** the signal to **−20 dB RMS** for consistent loudness and stable learning.
4. Apply **Voice Activity Detection (VAD)** to **remove silence** from the beginning and end of each recording.
5. **Resample** audio to **16 kHz** if not already set, ensuring a **uniform time grid**.
6. Apply a **pre-emphasis filter** (0.97) to boost high frequencies and improve the **signal-to-noise ratio**.
7. **Generate a Mel spectrogram** with the following parameters:
    - **Window:** 25 ms
    - **Hop:** 10 ms
    - **Mel bands:** 80 (optimal for speech)
8. **Convert the power matrix to log scale** (dB).
9. **Save as .pt tensor files** for fast and secure loading during training.

Note:
Applied **augmentation methods** and **spectrogram normalization** are described in detail later in the report.

---

3. Training and Validation

- The dataset loads the **saved spectrograms**.

- **Cross Entropy Loss** uses class weights to compensate for the **1:10 imbalance** between “allowed” and “not allowed” speakers.
- During training:
    - **Randomly select a 5-second segment** of each audio sample.
    - During validation: **use the first 5 seconds** of each sample.
- **Moderate augmentations** are applied:
    - **Masking** random time and frequency regions,
    - **Cutting out rectangular areas,**
    - **Shifting** the spectrogram along the frequency axis.
    - **Normalize** the spectrogram using statistics from the first 5 seconds of all training samples.
- The model is a fully parameterizable **FlexibleCNN**:
    - Number of blocks, presence of batch normalization, dropout, skip connections, and activation types are all **configurable**.
    - Facilitates **experimentation** with various architectures.
- **Two optimizers tested: SGD** and **AdamW**.
- **Metrics monitored: precision, recall, F1 score**.
- **Early stopping** is based on the best F1 on the validation set (checkpoint is reverted to the best epoch).
- **Maximum training duration: 1000 epochs**.
- Every **50 epochs**, as well as at the end of training:
    - **Model and optimizer checkpoints are saved**.
    - The **best model (early stopping)** is saved (with patience set to 100 epochs).

---

4. Fine-tuning with a New Speaker (Label “1”)

- After training, the model can be **extended to recognize a new person** by **retraining the last layer** (while **freezing the backbone**).
- Uses a **small set of new samples** plus **10% of old data** per speaker.
- **Prevents catastrophic forgetting** of previously learned classes.
- **Details of this experiment** are provided at the end of this report.

I. Data Exploration (Visual Analysis of Spectrograms)

Analysis Approach

- As part of data exploration, I analyzed **spectrograms** generated from voice recordings **before and after preprocessing and augmentation**.
- The **goal** of this analysis was to:
    - Obtain a **general overview** of the data structure.
    - **Verify the quality** of the signal processing steps.

Gender Differences

- When comparing **spectrograms** from **male and female speakers**, **characteristic acoustic differences** were observed:
    - **Male voices**:
        - Tend to concentrate **energy in lower frequency bands**.
        - Feature **prominent, broad F1 and F2 formants**.
    - **Female voices**:
        - Tend to generate signals with **higher energy in mid and high frequencies**.
        - This appears as **denser high formants** and a **greater number of visible harmonics**.

- These differences can be **useful for speaker classification or gender recognition** based on the audio signal.

Impact of Signal Processing Operations

- **Several preprocessing operations** were applied before converting audio signals to spectrograms.
- **Analysis of spectrograms before and after these transformations revealed:**
    - **Normalization to -20 dB RMS**:
        - **Unified the loudness** of recordings, resulting in more **consistent brightness/intensity** in spectrograms.
        - This ensures the model learns the **signal content**, not just loudness.
    - **Voice Activity Detection (VAD)**:
        - **Removed silence** from the beginning and end of recordings.
        - Clearly **shortened some spectrograms** and reduced “dark” areas at the temporal edges.
        - Allows the model to **focus on the relevant signal portions**.
    - **Resampling to 16 kHz**:
        - Provided a **uniform time-frequency grid**.
        - All spectrograms have a **consistent frame and mel band layout**.
    - **Pre-emphasis (coefficient 0.97)**:
        - **Boosted higher frequencies**, increasing visibility of formants and harmonics in upper bands.
        - Improved **signal-to-noise ratio** and **input data quality** for the model.

Data Augmentation

- **Augmentation techniques** (such as **time and frequency masking**) were also applied, which:
    - **Simulated channel distortions** (e.g., loss of frequency bands).
    - **Randomly covered segments** of the utterance (**simulating noise or dropouts**).
    - **Forced the model to become more robust** to missing or corrupted information.

Summary

- **Visual analysis** of spectrograms before and after processing and augmentation **confirmed the effectiveness** of the applied procedures.
- As a result, the input data was:
    - **Normalized**
    - **Cleaned of silence**
    - **Standardized and enhanced** for deep learning purposes
- This **exploration** was a **crucial step** in validating the **data preparation pipeline** and the **quality of input representations**.

II. Several Techniques / Architectural Components

1. Model Architecture

- The initial model consists of **3 CNN blocks** with the following number of filters:
    - **16**
    - **32**
    - **32**
- After each **convolutional layer**, there is:
    - **ReLU activation function**

    - **Max Pooling layer**

- The output from the convolutional blocks is then passed to **two fully connected (linear) layers** with:
    - **64 neurons**
    - **2 neurons** (final output layer)

---

2. Weight Initialization

- **Custom weight initialization** is applied depending on the layer type:
    - For **convolutional layers**:
        - **He (Kaiming Normal) initialization** with **fan-out mode**
            - Helps ensure stable signal propagation with ReLU activations.
    - For **normalization layers**:
        - **Weights** initialized to **ones**
        - **Biases** initialized to **zeros**
            - Provides a **neutral effect** at the beginning of training.
    - For **linear (fully connected) layers**:
        - **Uniform initialization** within a range dependent on the number of outputs
            - **Limits excessive weight magnitudes** and **stabilizes gradients**.
        - **Biases** (if present) are initialized to **zeros**.

---

3. Learning Rate Selection

- Initially, both **SGD** and **AdamW** optimizers are evaluated to determine a **suitable learning rate**:

    - The goal is to find a value **large enough for efficient training**, but **not so large that training fails to converge**.

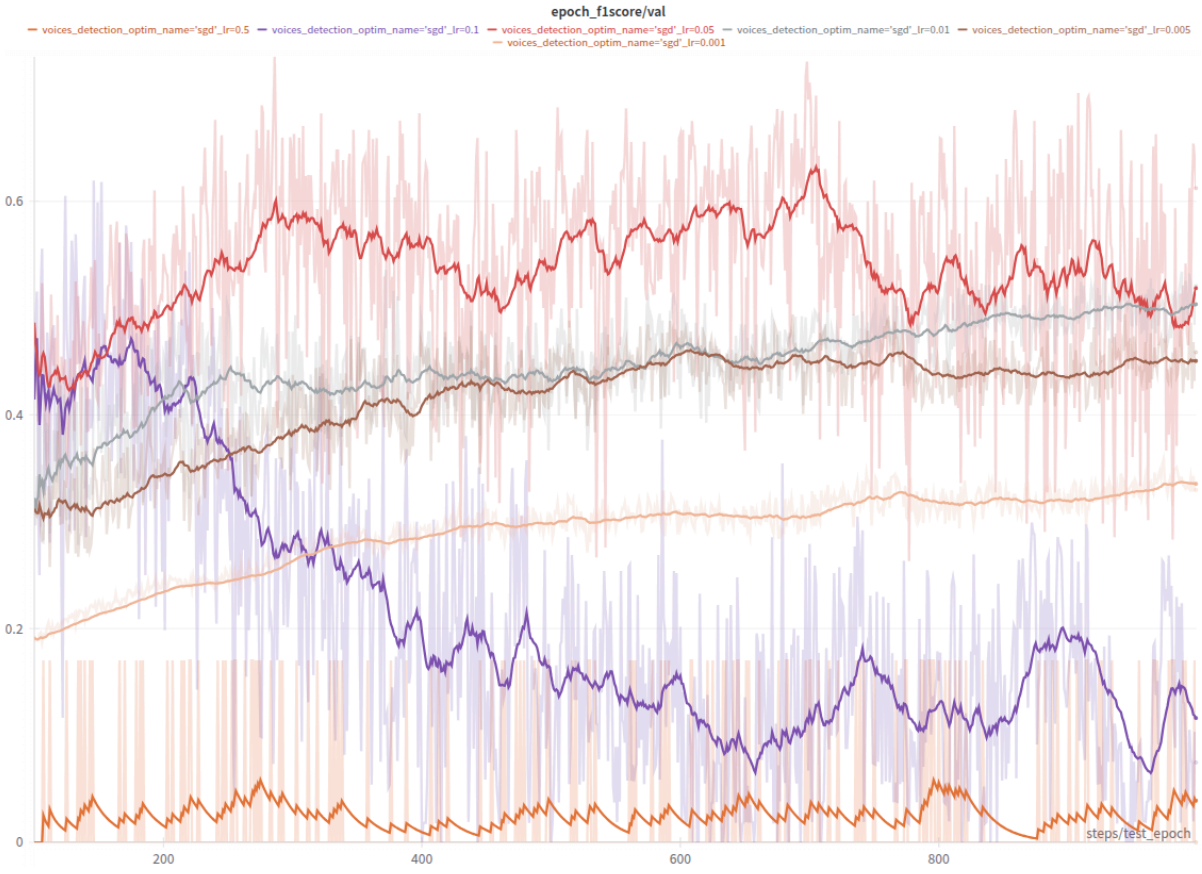
- In this experiment:
    - **Early stopping** is **disabled** (training does not stop automatically if there is no improvement).
    - The **best model checkpoint** encountered during training is saved.
    - At the end of training, the **F1 score** is reported on the **test set** using this best model.

3.1. SGD Results

- For **SGD**, the following learning rates were tested:
  [1e-3, 5e-3, 1e-2, 5e-2, 1e-1, 5e-1]
- The **best F1 score** achieved on the test set was:
    - **F1 = 0.7449**
    - **Precision = 0.7731**
    - **Recall = 0.7187**
    - For **learning rate (lr) = 5e-2**

3.2. Analysis of Learning Rate Impact

- **Analysis of F1 scores** on the **validation set** shows that:
    - **Learning rates higher** than **5e-2** do **not** lead to **stable solutions**.
    - **Lower learning rates** result in either **slower training** or **worse outcomes** (possibly due to sharp minima).


epoch_f1score/val
voices_detection_optim_name='sgd'_lr=0.5
voices_detection_optim_name='sgd'_lr=0.1
voices_detection_optim_name='sgd'_lr=0.05
voices_detection_optim_name='sgd'_lr=0.01
voices_detection_optim_name='sgd'_lr=0.005
voices_detection_optim_name='sgd'_lr=0.001
0.6
0.4
0.2
0
200
400
600
800
steps/test_epoch

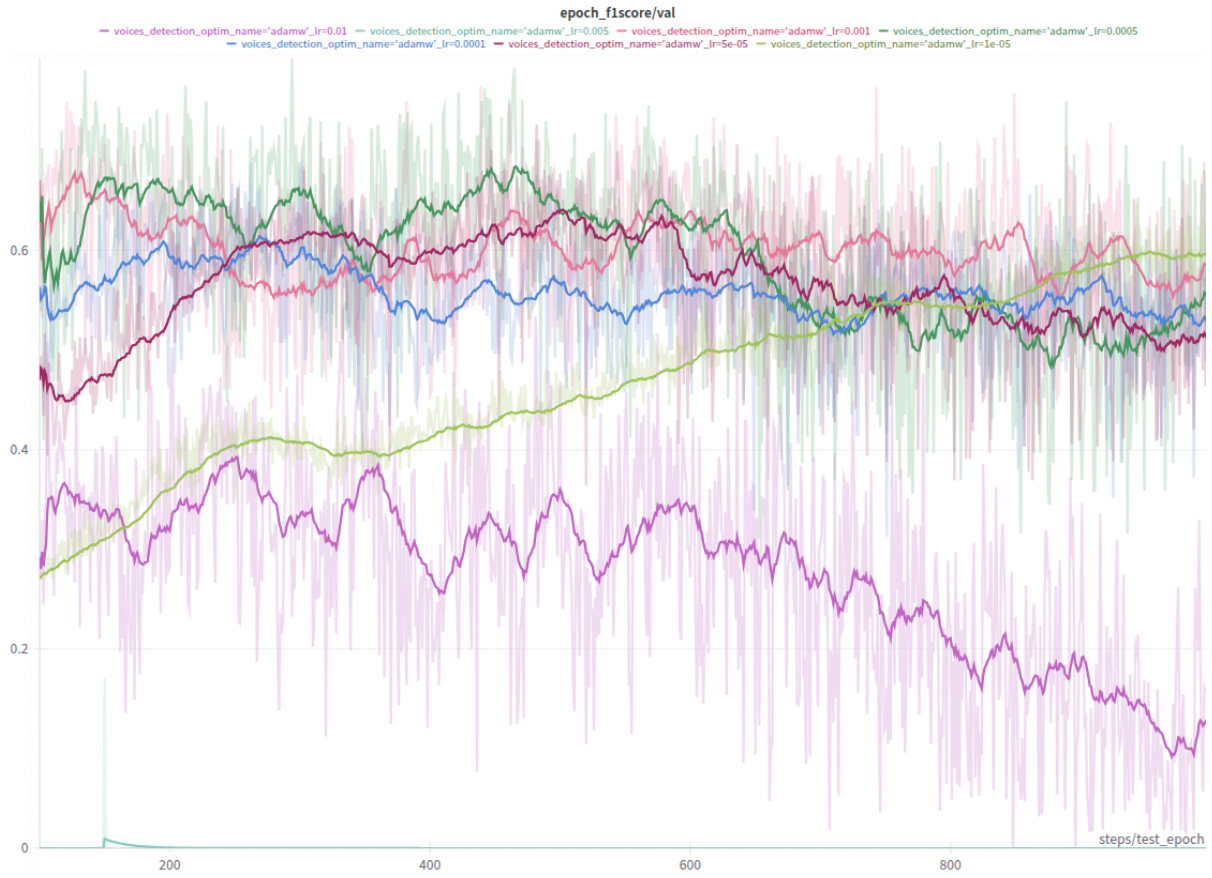

epoch_f1score/test
voices_detection_optim_name='sgd'_lr=0.5
voices_detection_optim_name='sgd'_lr=0.1
voices_detection_optim_name='sgd'_lr=0.05
voices_detection_optim_name='sgd'_lr=0.01
voices_detection_optim_name='sgd'_lr=0.005
voices_detection_optim_name='sgd'_lr=0.001
999: 0.74494 voices_detection_optim_name='sgd'_lr=0.05
999: 0.63188 voices_detection_optim_name='sgd'_lr=0.1
999: 0.51646 voices_detection_optim_name='sgd'_lr=0.01
999: 0.45476 voices_detection_optim_name='sgd'_lr=0.005
999: 0.32222 voices_detection_optim_name='sgd'_lr=0.001
999: 0.2515 voices_detection_optim_name='sgd'_lr=0.5
0.7
0.6
0.5
0.4
0.3
500
1k
1.5k
steps/test_epoch

3.3. AdamW Results

- For the **AdamW optimizer**, the **best F1 score** achieved on the test set was:
    - **F1 = 0.7607**
    - **Precision = 0.9254**
    - **Recall = 0.6458**
    - For **learning rate (lr) = 1e-3**

3.4. Analysis of Learning Rate Impact (AdamW)

- Similar to the analysis for **SGD**:
    - **Learning rates higher** than **1e-3** do **not** lead to **stable solutions**.
    - **Lower learning rates** cause **slower training** or **worse results**.


epoch_f1score/val
voices_detection_optim_name='adamw'_lr=0.01
voices_detection_optim_name='adamw'_lr=0.005
voices_detection_optim_name='adamw'_lr=0.001
voices_detection_optim_name='adamw'_lr=0.0005
voices_detection_optim_name='adamw'_lr=0.0001
voices_detection_optim_name='adamw'_lr=5e-05
voices_detection_optim_name='adamw'_lr=1e-05
0.6
0.4
0.2
0
200
400
600
800
steps/test_epoch


epoch_f1score/test
voices_detection_optim_name='adamw'_lr=0.01
voices_detection_optim_name='adamw'_lr=0.005
voices_detection_optim_name='adamw'_lr=0.001
voices_detection_optim_name='adamw'_lr=0.0005
voices_detection_optim_name='adamw'_lr=0.0001
voices_detection_optim_name='adamw'_lr=5e-05
voices_detection_optim_name='adamw'_lr=1e-05
999: 0.76074 voices_detection_optim_name='adamw'_lr=0.001
999: 0.75038 voices_detection_optim_name='adamw'_lr=0.0005
999: 0.66667 voices_detection_optim_name='adamw'_lr=0.0001
999: 0.63636 voices_detection_optim_name='adamw'_lr=5e-05
999: 0.63158 voices_detection_optim_name='adamw'_lr=1e-05
999: 0.44687 voices_detection_optim_name='adamw'_lr=0.01
999: 0.17021 voices_detection_optim_name='adamw'_lr=0.005
0.7
0.6
0.5
0.4
0.3
0.2
500
1k
1.5k
steps/test_epoch

3.5. Observations on Loss Function and Overfitting

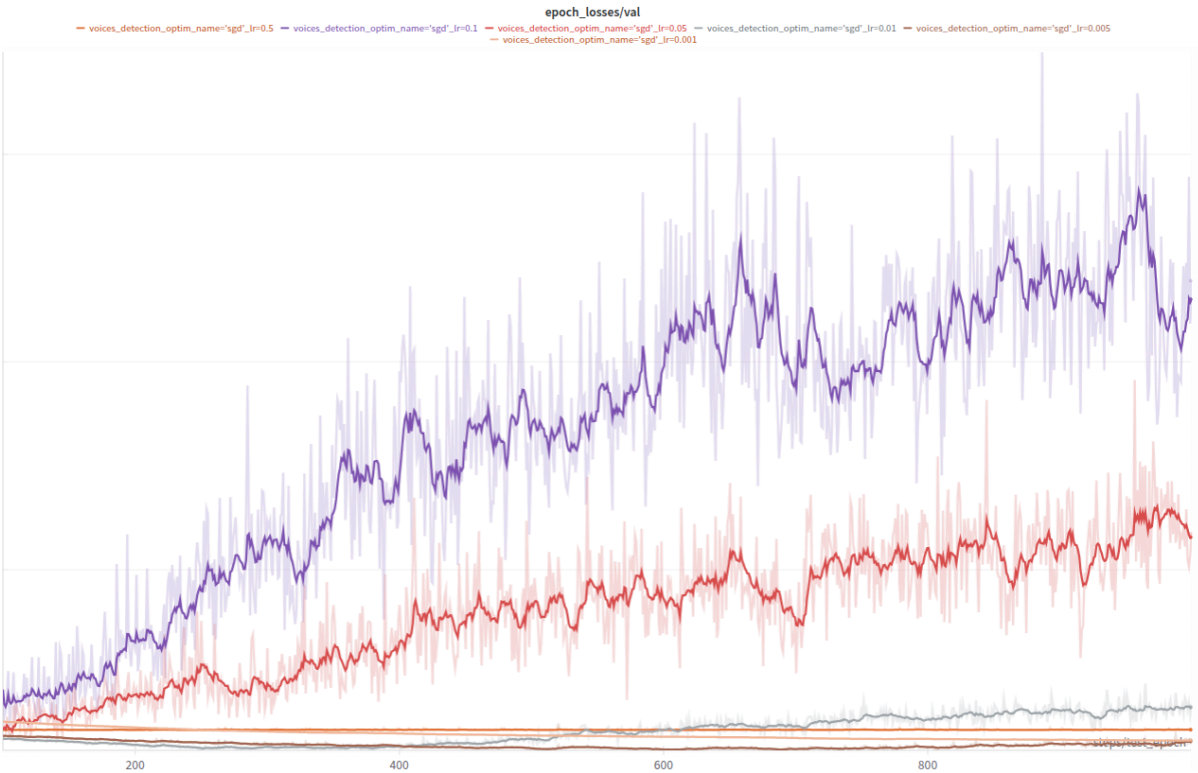
- **Monitoring the loss function** on the **validation set** revealed that, for **both SGD and AdamW**, the model **overfits very quickly** during each training run.

- This observation indicates a **need for stronger regularization**—even **more** than the spectrogram augmentation techniques already applied.

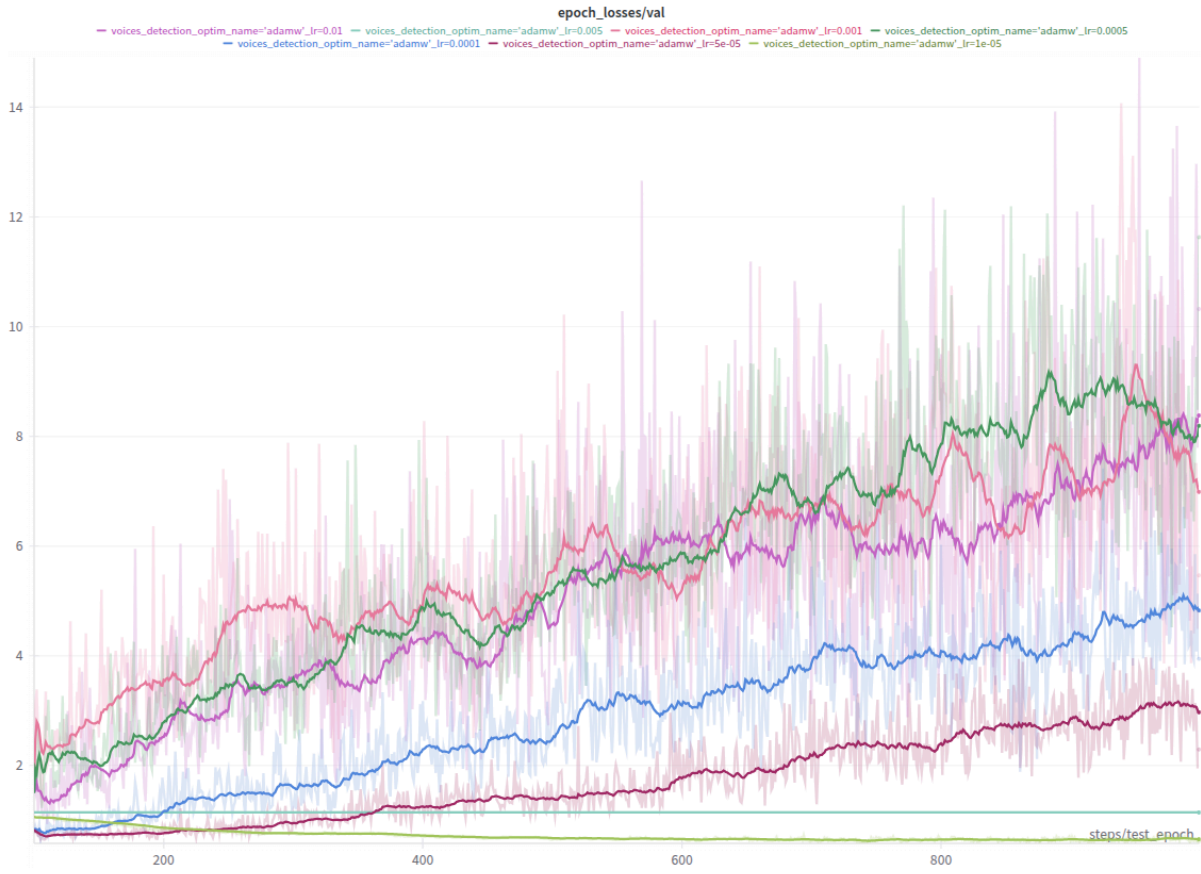
3.6. Loss Curves

- **Loss curves** were monitored for both **SGD** and **AdamW** during training:

    - **SGD losses:**



- **AdamW losses:**



3.7. Optimizer Comparison

- The results clearly show that the **AdamW optimizer** requires a **smaller learning rate** than **SGD** to guide the model toward stable solutions.
- This is due to the **adaptive gradient scaling** mechanism in **AdamW**.

3.8. Selection for Further Experiments

- For all **further experiments** with these two optimizers,
  the **chosen learning rates** are those identified as optimal in the previous tests:
  - **SGD:** `lr = 5e-2`
  - **AdamW:** `lr = 1e-3`

---

4. Different Optimizers (SGD, AdamW), Learning Rate Schedules, and Weight Decay

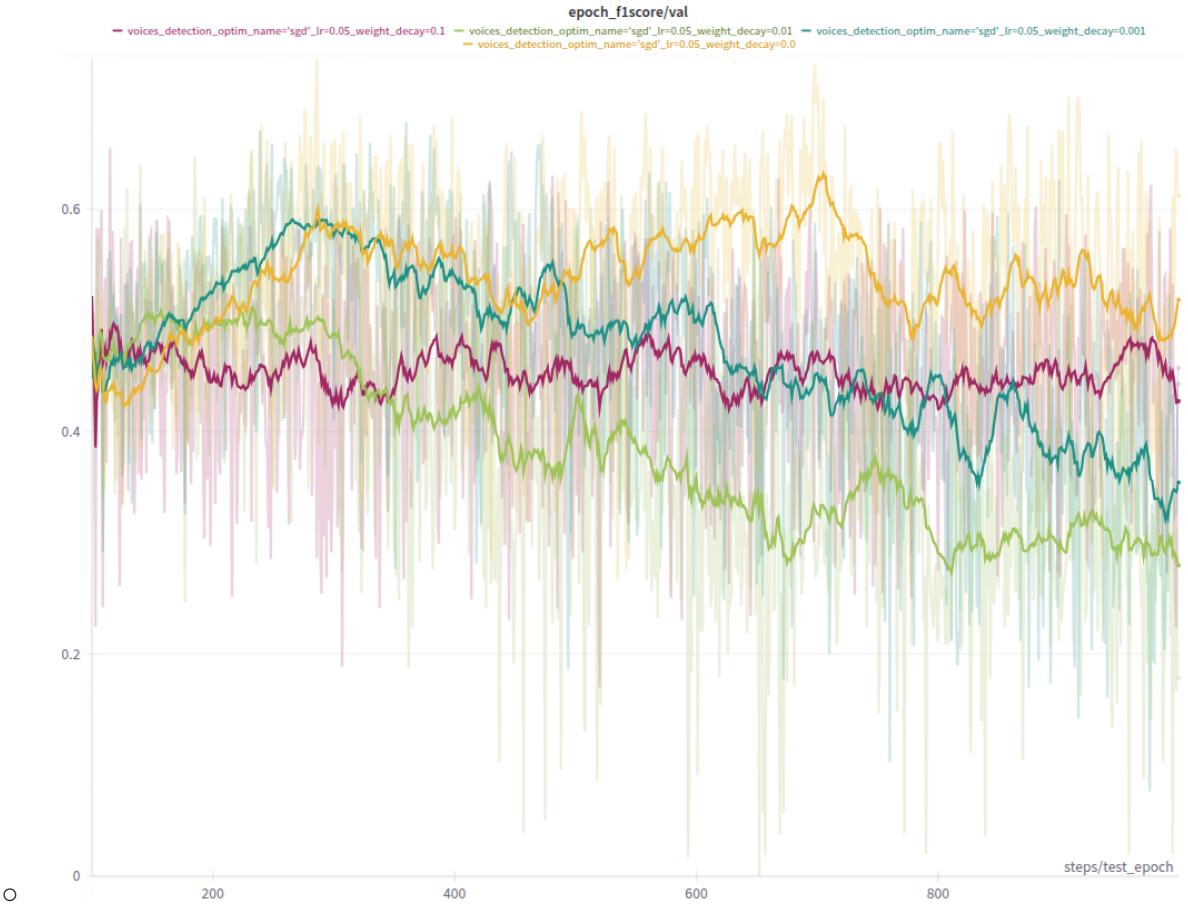
---

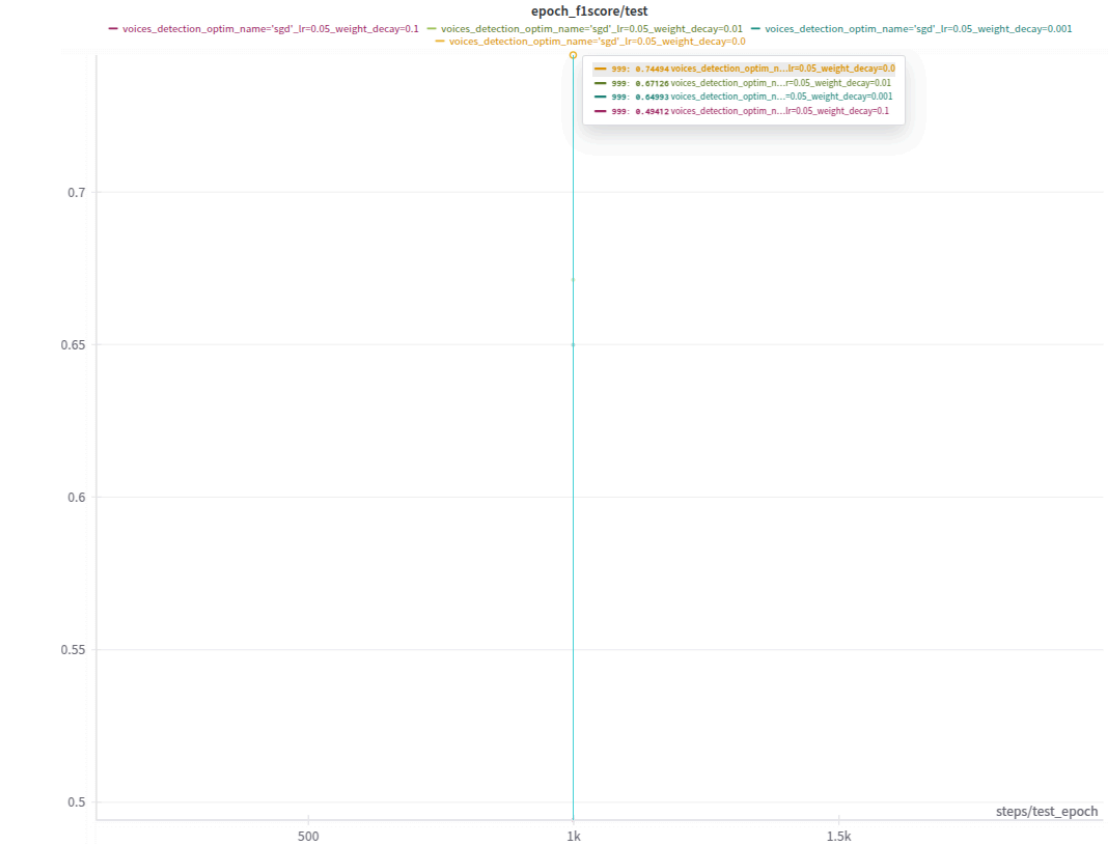
4.1. Experiment Setup

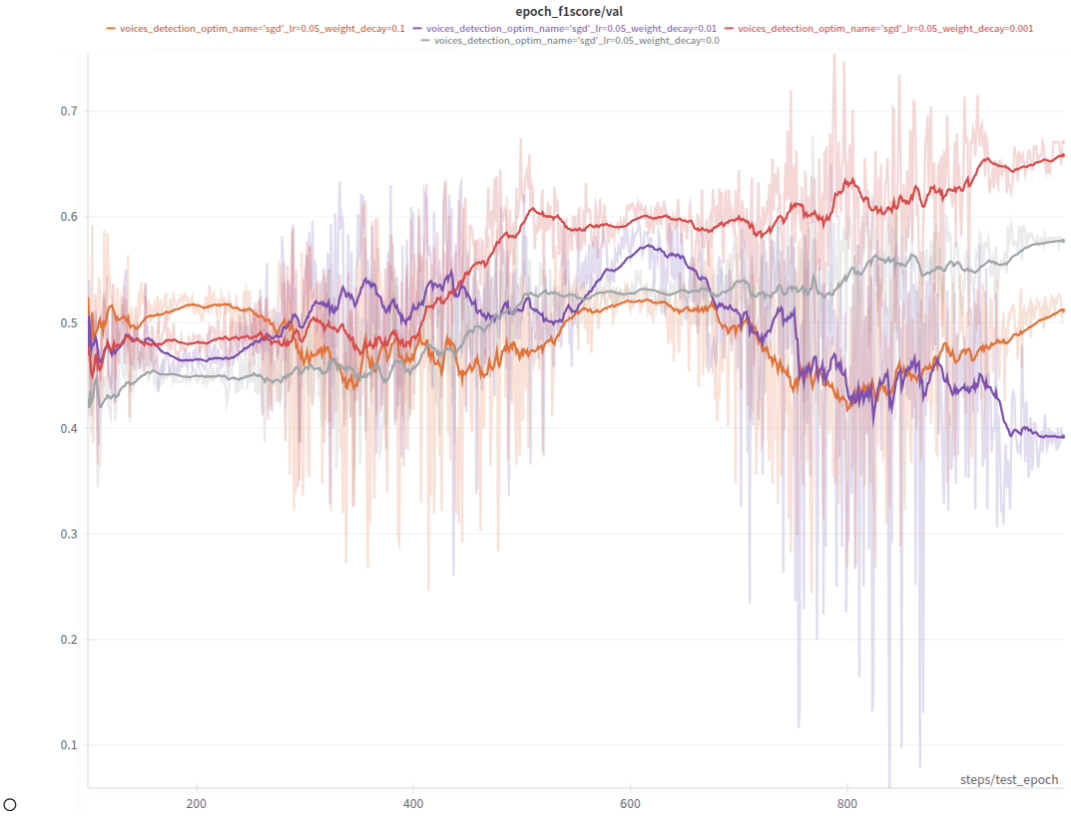
- In this experiment, for both **SGD** and **AdamW**, the following parameters were explored:
    - **Learning rate schedule**:
        - **None** (no scheduler)
        - **Cosine annealing** (**T_max = 200**)
        - **Reduce on plateau** (**patience = 50 epochs**, **factor = 0.8**)
    - **Weight decay** (**wd**):
        - **[0.0, 1e-3, 1e-2, 1e-1]**

4.2. SGD Results

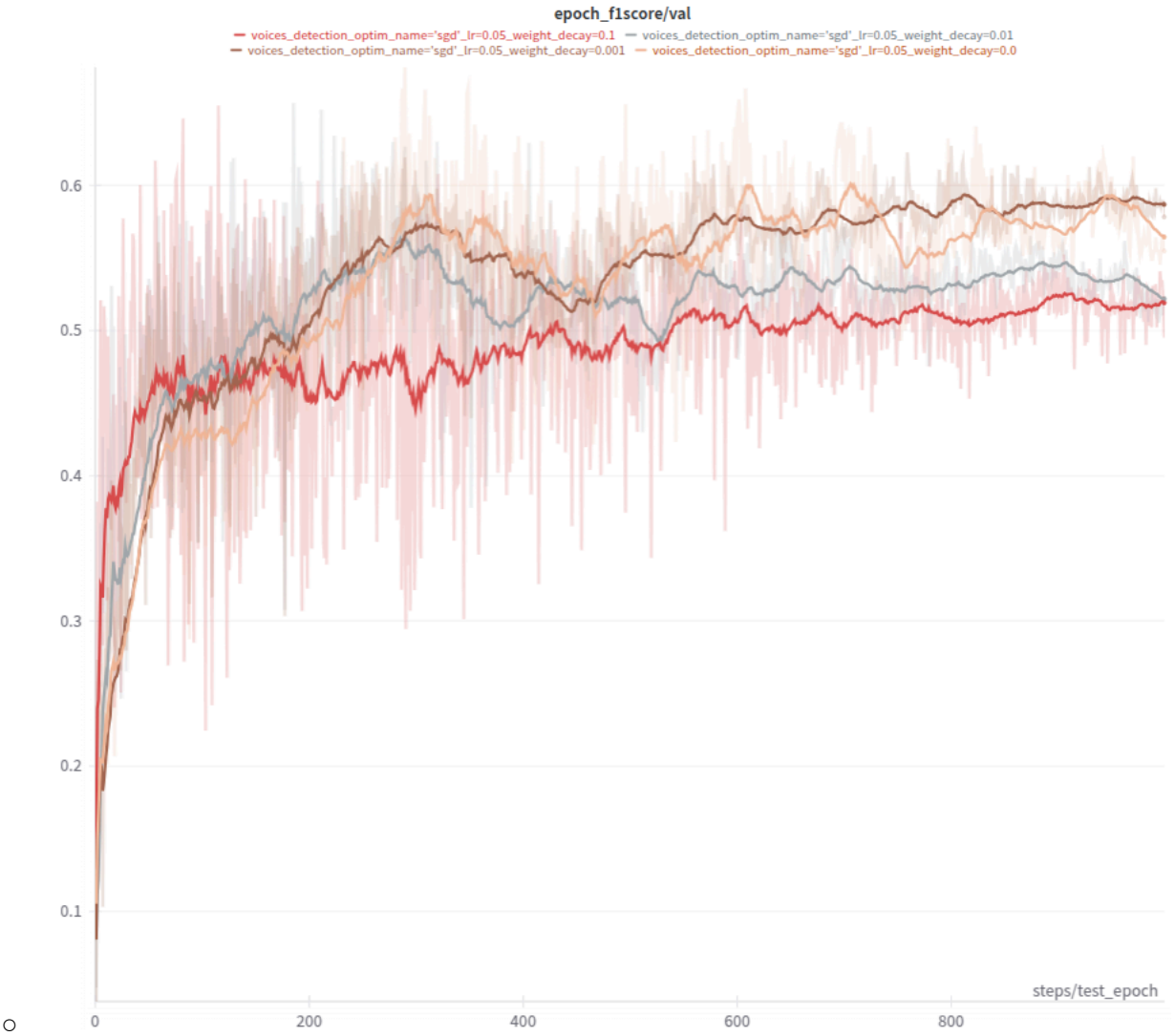
- **scheduler = None**:
    - **Best F1 = 0.7449** (**precision = 0.7731**, **recall = 0.7187**) for **wd = 0.0**


epoch_f1score/val
voices_detection_optim_name='sgd'_lr=0.05_weight_decay=0.1
voices_detection_optim_name='sgd'_lr=0.05_weight_decay=0.01
voices_detection_optim_name='sgd'_lr=0.05_weight_decay=0.001
voices_detection_optim_name='sgd'_lr=0.05_weight_decay=0.0
0.6
0.4
0.2
0
200
400
600
800
steps/test_epoch


epoch_f1score/test
voices_detection_optim_name='sgd'_lr=0.05_weight_decay=0.1
voices_detection_optim_name='sgd'_lr=0.05_weight_decay=0.01
voices_detection_optim_name='sgd'_lr=0.05_weight_decay=0.001
voices_detection_optim_name='sgd'_lr=0.05_weight_decay=0.0
999: 0.74494 voices_detection_optim_n...lr=0.05_weight_decay=0.0
999: 0.67126 voices_detection_optim_n...r=0.05_weight_decay=0.01
999: 0.64993 voices_detection_optim_n...=0.05_weight_decay=0.001
999: 0.49412 voices_detection_optim_n...lr=0.05_weight_decay=0.1
0.7
0.65
0.6
0.55
0.5
500
1k
1.5k
steps/test_epoch

- **scheduler = Cosine**:
  - **Best F1 = 0.7318** (**precision = 0.9042**, **recall = 0.6149**) for **wd = 1e-3**
    - 

    - 

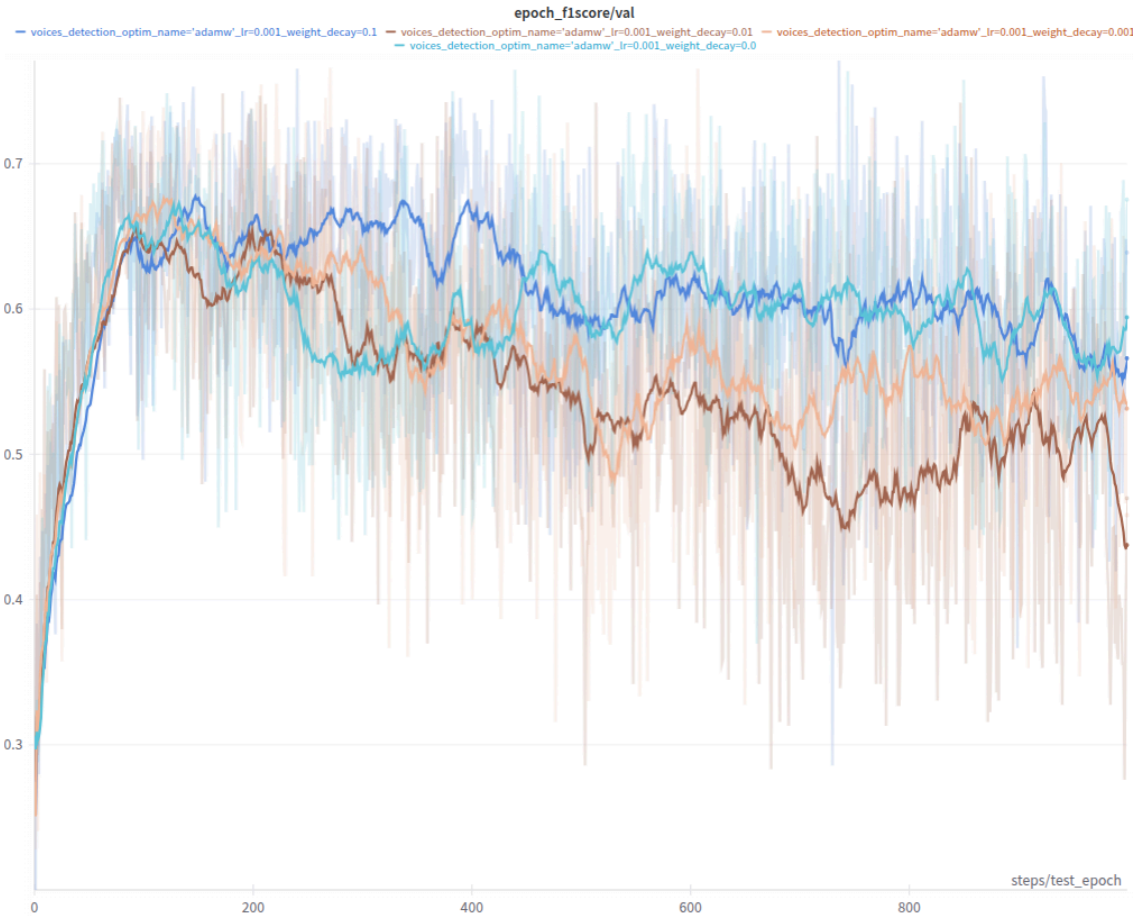
- **scheduler = Reduce on Plateau**:
  - **Best F1 = 0.6461** (**precision = 0.7407**, **recall = 0.5729**) for **wd = 0.0**
  - 

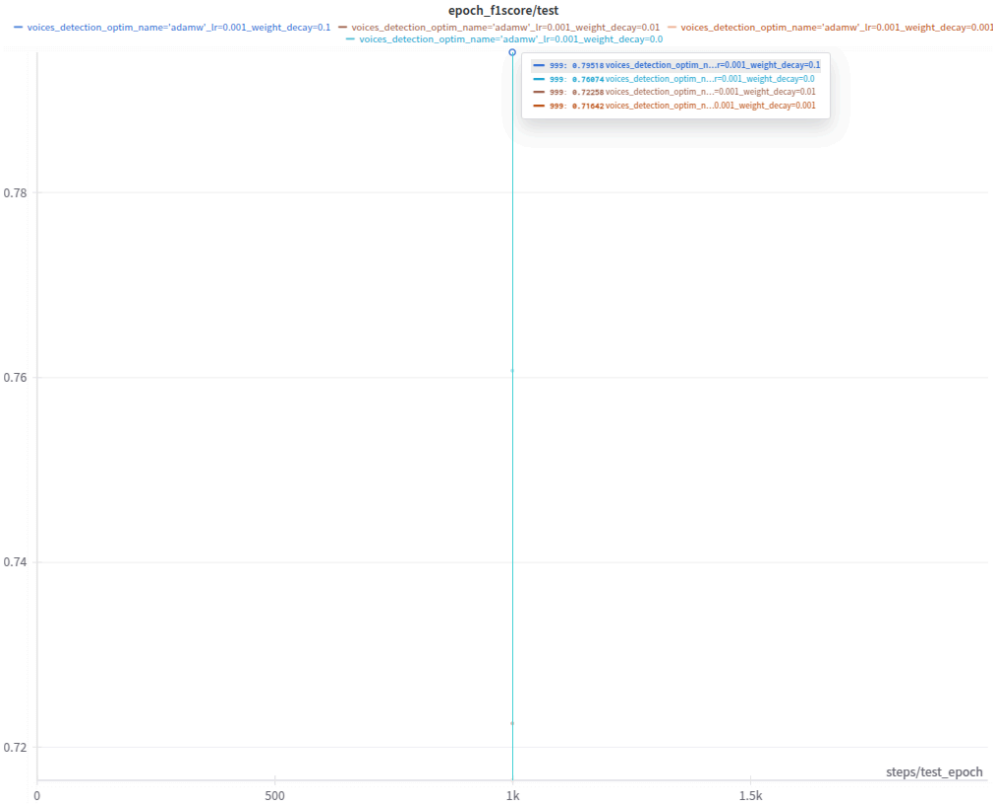
Observations:

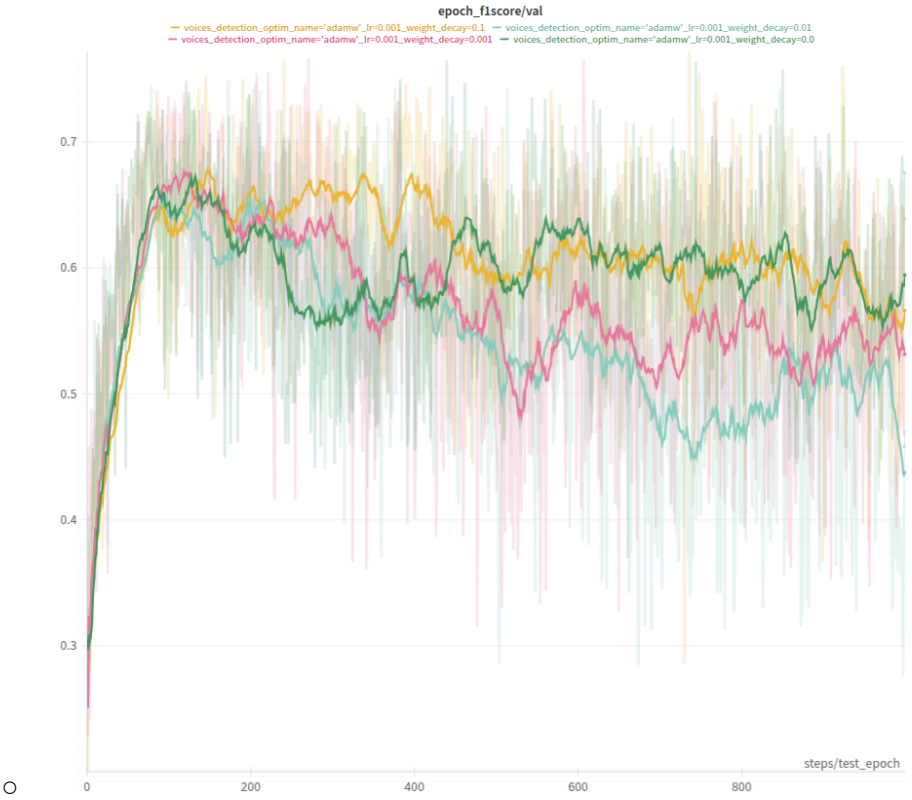
- Applying **schedulers** with **SGD** leads to **worse models** compared to using no scheduler.
- Among the schedulers, **cosine annealing** performs **better than reduce on plateau**.
- **Non-zero weight decay** only helps when **scheduler = cosine**.

4.3. AdamW Results

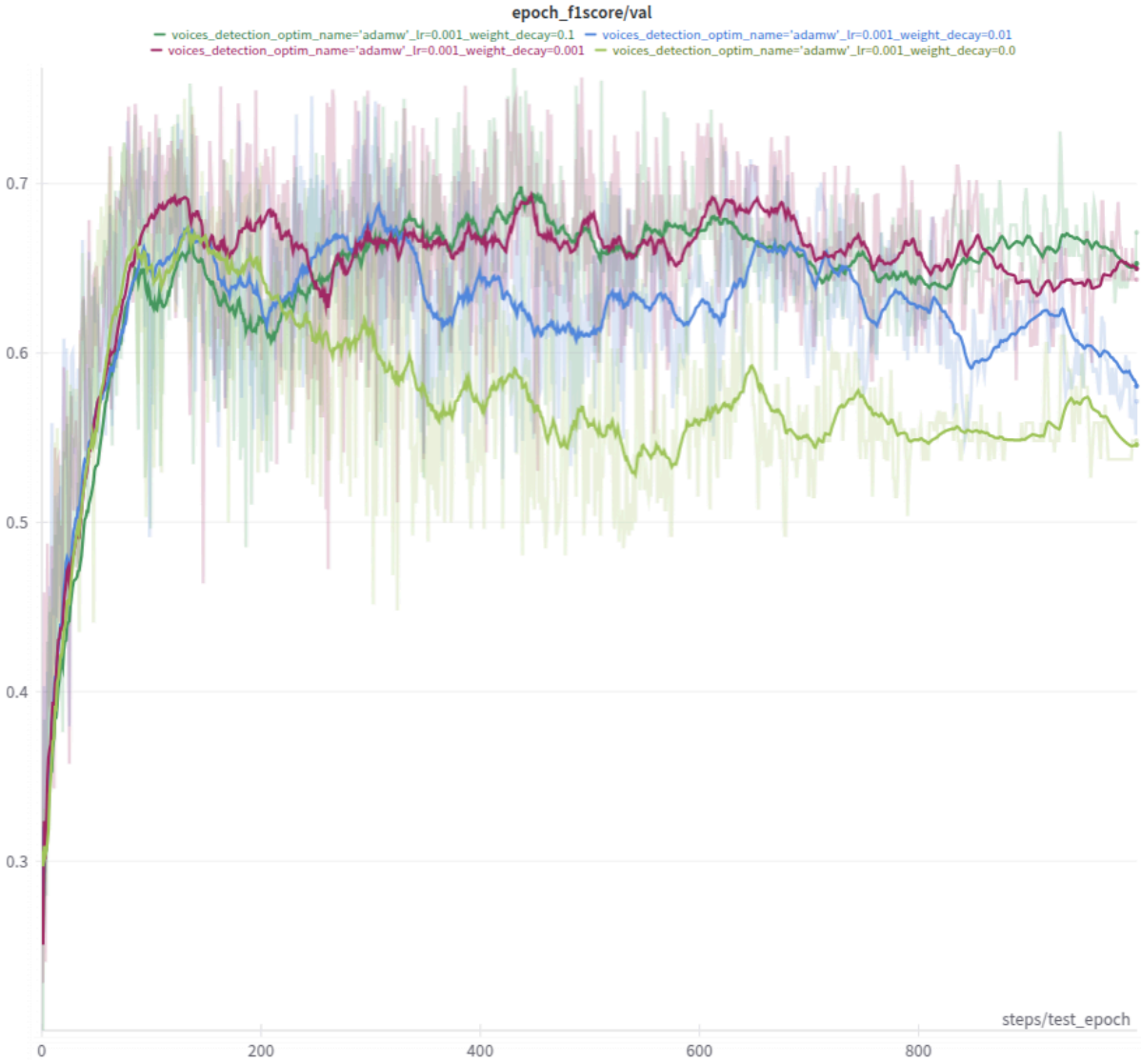
- **scheduler = None**:
    - **Best F1 = 0.7952** (**precision = 0.9429**, **recall = 0.6875**) for **wd = 1e-1**


epoch_f1score/val
voices_detection_optim_name='adamw'_lr=0.001_weight_decay=0.1
voices_detection_optim_name='adamw'_lr=0.001_weight_decay=0.01
voices_detection_optim_name='adamw'_lr=0.001_weight_decay=0.001
voices_detection_optim_name='adamw'_lr=0.001_weight_decay=0.0
0.7
0.6
0.5
0.4
0.3
0
200
400
600
800
steps/test_epoch


epoch_f1score/test
voices_detection_optim_name='adamw'_lr=0.001_weight_decay=0.1
voices_detection_optim_name='adamw'_lr=0.001_weight_decay=0.01
voices_detection_optim_name='adamw'_lr=0.001_weight_decay=0.001
voices_detection_optim_name='adamw'_lr=0.001_weight_decay=0.0
999: 0.79518 voices_detection_optim_n...r=0.001_weight_decay=0.1
999: 0.76074 voices_detection_optim_n...r=0.001_weight_decay=0.0
999: 0.72258 voices_detection_optim_n...=0.001_weight_decay=0.01
999: 0.71642 voices_detection_optim_n...0.001_weight_decay=0.001
0.78
0.76
0.74
0.72
0
500
1k
1.5k
steps/test_epoch

- **scheduler = Cosine**:
  - **Best F1 = 0.7819** (**precision = 0.9227**, **recall = 0.6975**) for **wd = 1e-1**
  - 

  - 

- **scheduler = Reduce on Plateau**:
  - **Best F1 = 0.7831** (**precision = 0.9286**, **recall = 0.6771**) for **wd = 1e-1**
  - 

Observations:

- As with SGD, **schedulers** tend to **worsen the test F1 score** compared to no scheduler.
- However, in all cases, **weight decay = 1e-1** yields the best result with **AdamW**.

4.4. Summary & Conclusions

- **The best result overall** was achieved with:
  - **AdamW optimizer**

    - **No learning rate scheduler**
    - **Weight decay = 1e-1**

- For further experiments, the **reduce on plateau scheduler** is chosen, as its result is close to the best, and - as in previous experiments - **the model overfits very quickly** in every setup.

- Using a **scheduler in combination with regularization** may be helpful to improve generalization.

4.5. Precision vs. Recall

- In all experiments so far, **precision is higher than recall**.
    - This means the model is **conservative**: it allows only the most obvious candidates, but **misses some correct ones**.

- **AdamW models** consistently achieve **precision > 90%** (compared to **SGD**, where it is below 80%).
    - meaning they **rarely make mistakes when granting access** — if someone is let in, they are almost always authorized.
    - However, the **recall remains relatively low (below 70%)**, which suggests that the model **fails to recognize a significant number of eligible individuals**.

---

5. Experiments with Batch Normalization, Dropout, and Skip Connections (Extended Results)

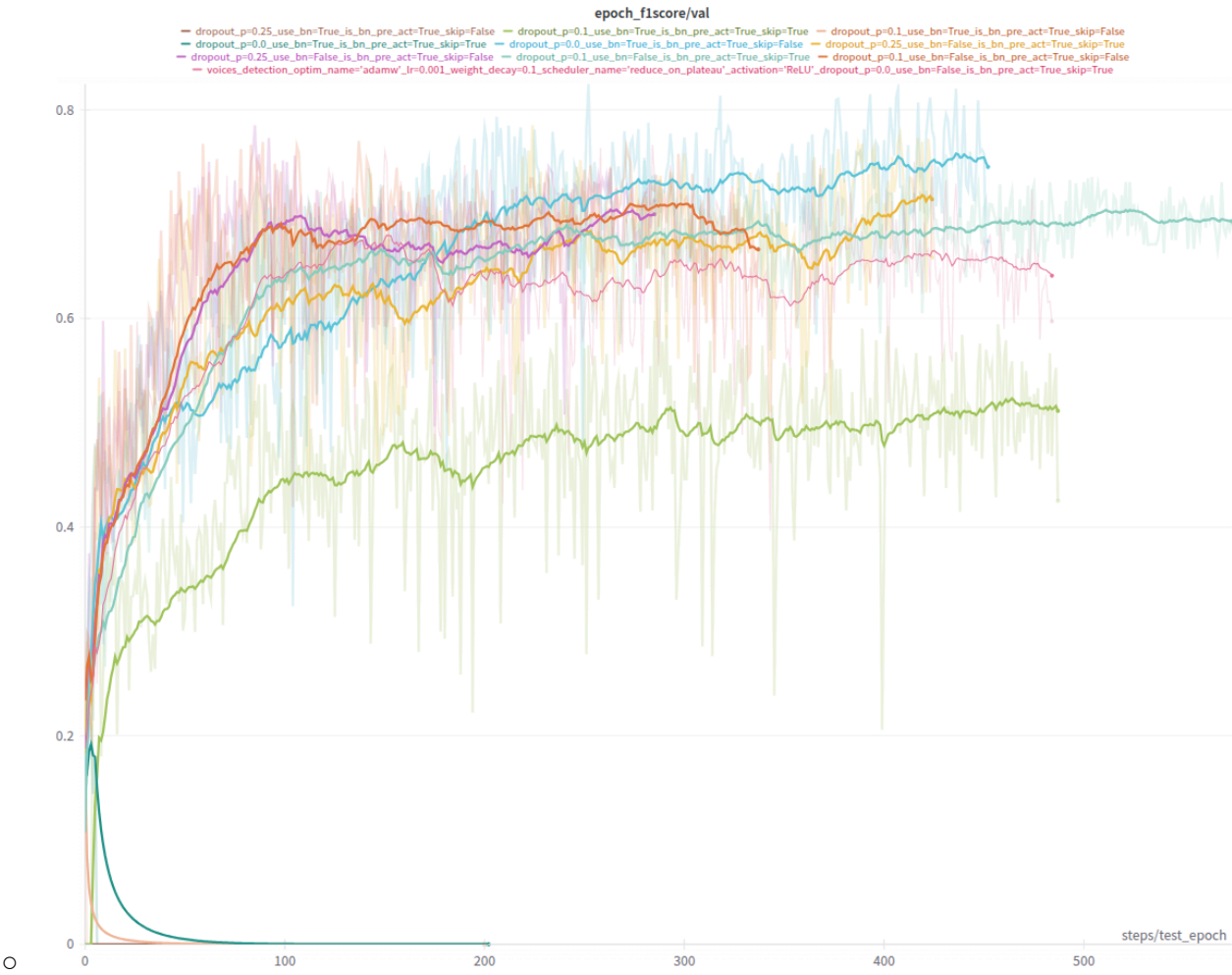
Experiment Setup

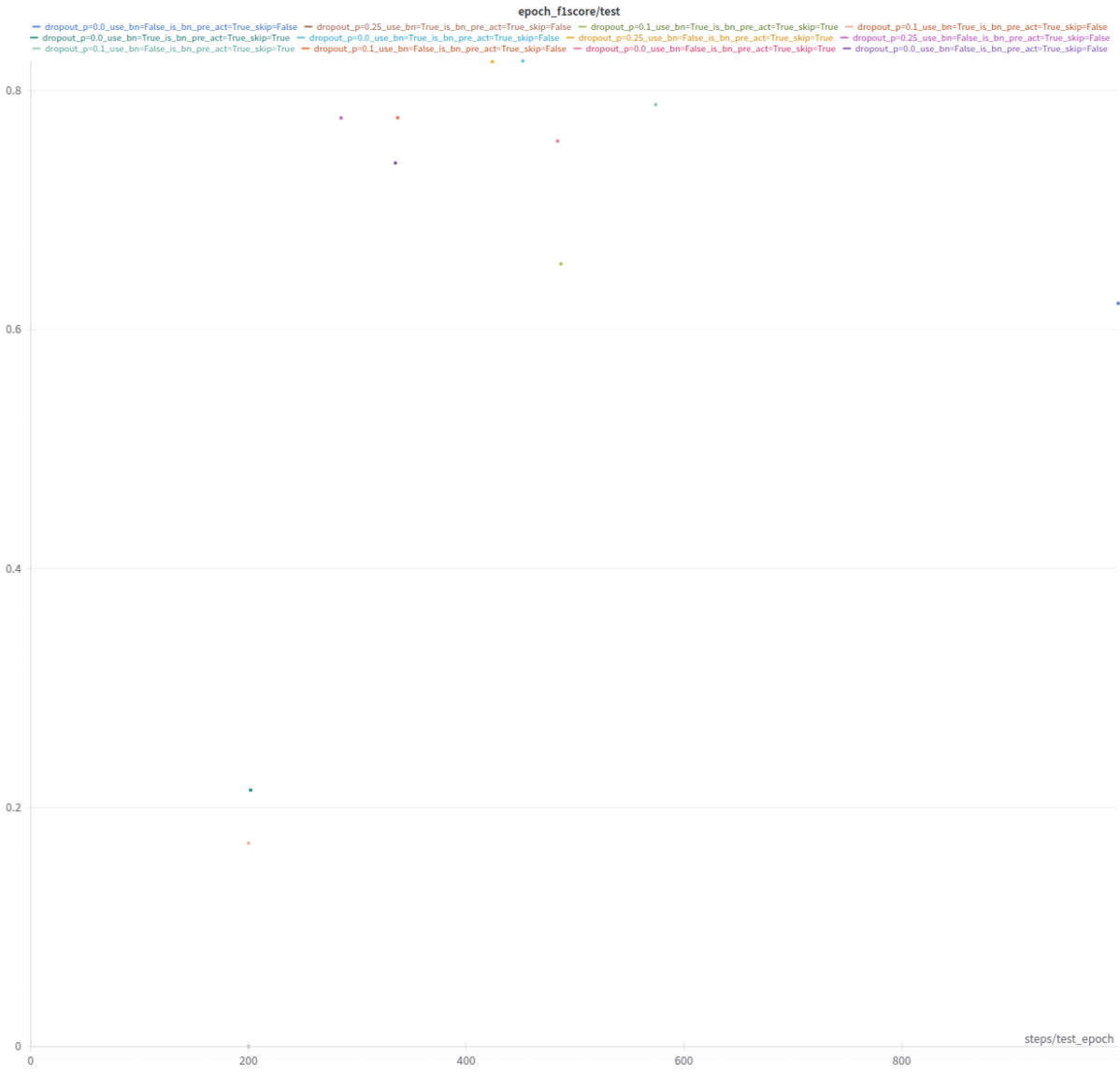
- In this group of experiments, I examined the use of popular **regularization techniques**:
    - **Batch normalization**
    - **Dropout**
    - **Skip connections**

- **Early stopping** was enabled, with **patience set to 100 epochs**.
- During experiments with **batch normalization**:
    - **Batch normalization parameters** were **excluded from weight decay**.
    - The **ReLU activation function** was placed **immediately after** each batch normalization layer.
- Explored variants:
    - **Batch normalization** with standard parameters (enabled/disabled)
    - **Skip connections** (enabled/disabled)
    - **Dropout** in **[0.00, 0.1, 0.25]** (grid search)

Best Results

- **Best F1 score = 0.8249**
    - **Precision = 0.9012**
    - **Recall = 0.7604**
- Configuration:
    - **Dropout = 0.0**
    - **Skip connections: disabled**
    - **Batch normalization: enabled**


epoch_f1score/val
dropout_p=0.25_use_bn=True_is_bn_pre_act=True_skip=False
dropout_p=0.1_use_bn=True_is_bn_pre_act=True_skip=True
dropout_p=0.1_use_bn=True_is_bn_pre_act=True_skip=False
dropout_p=0.0_use_bn=True_is_bn_pre_act=True_skip=True
dropout_p=0.0_use_bn=True_is_bn_pre_act=True_skip=False
dropout_p=0.25_use_bn=False_is_bn_pre_act=True_skip=True
dropout_p=0.25_use_bn=False_is_bn_pre_act=True_skip=False
dropout_p=0.1_use_bn=False_is_bn_pre_act=True_skip=True
dropout_p=0.1_use_bn=False_is_bn_pre_act=True_skip=False
voices_detection_optim_name='adamw'_lr=0.001_weight_decay=0.1_scheduler_name='reduce_on_plateau'_activation='ReLU'_dropout_p=0.0_use_bn=False_is_bn_pre_act=True_skip=True
0.8
0.6
0.4
0.2
0
0
100
200
300
400
500
steps/test_epoch



Key Observations

1. **Dropout**:
    - **Improves final performance** as the dropout probability increases, **only if** other regularization (batch normalization) is **not available**, or only **skip connections** are used.
    - **Worsens performance** if **batch normalization is already used**.
2. **Batch normalization + Skip connections**:
    - When **both batch normalization and skip connections** are enabled, **training only succeeded** for **dropout = 0.1** (higher/lower values failed).

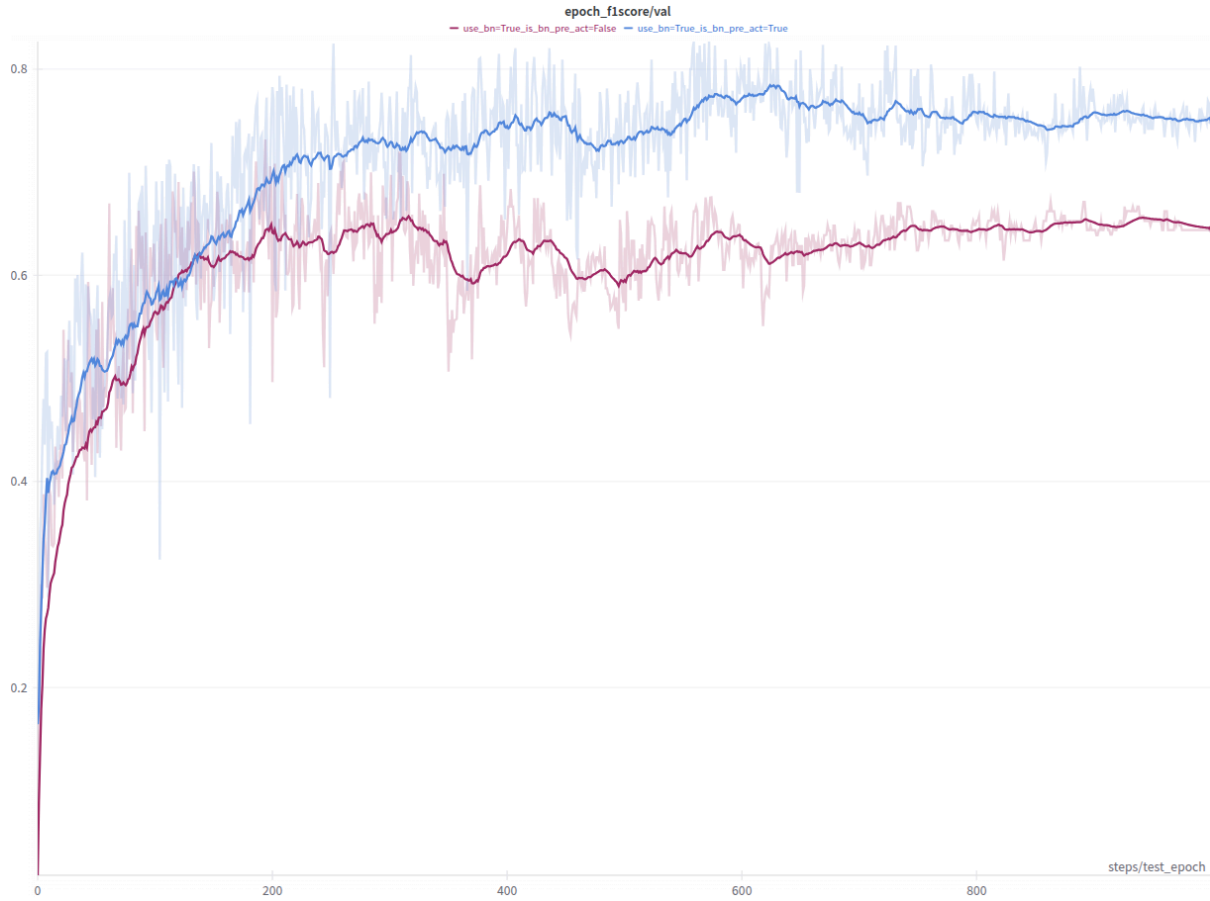
3. **Increase in recall**
   As **a result**, we can observe that applied **regularization increased recall**.

Additional Observations: Skip Connections and Batch Normalization Placement

- In the case of such a **shallow network** (without blocks),
  no substantial benefit from skip connections was observed.

- Additionally, for the selected **best model**, I tested what happens if the **activation function is placed before the batch normalization layer**.

    - The result was **worse performance**.



- The probable reason is that, in this setup, the **batch normalization layer learns the distribution** of values **already transformed by the activation function** (in this case, a function that does **not return negative values**, e.g., ReLU).
    - As a result, the **range of distribution shifts** becomes limited,
    - Which **reduces the expressive power of the data**.

---

6. Adding a New Speaker to the Allowed Class

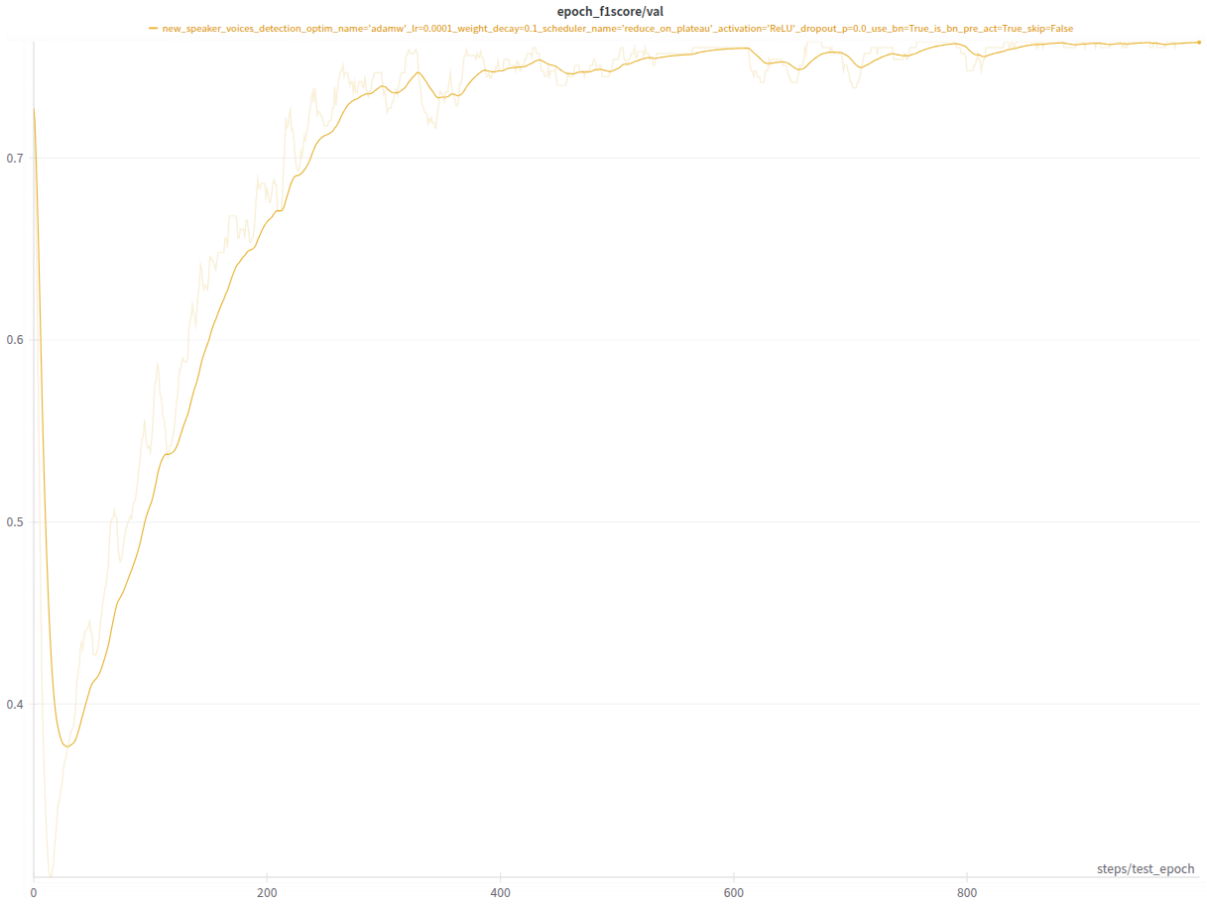
Procedure

- For this task, the **best-performing network** (from previous regularization experiments) was **fine-tuned** using the recordings of a **new speaker**.
- **Validation** was performed on the remaining recordings, using a data split **similar to the one described at the beginning of this report**.

- Additionally, **10% of data from each other speaker** was **added to the training set**.
    1. This acts as a **replay buffer** to prevent catastrophic forgetting.
- The network was then validated on:
    1. The **original validation set**
    2. A **validation set consisting of the new speaker's recordings**
    3. The **combined validation set** (original + new speaker)
- The **backbone** of the model was **frozen—only the head was trained**.
- The **learning rate was reduced by an order of magnitude** for this fine-tuning step.

Results

- The **F1 scores** achieved were as follows:
    - **Original test set: F1 = 0.7677**
    - **Test set of the new speaker: F1 = 0.7692**
    - **Combined test sets: F1 = 0.7678**


epoch_f1score/val
new_speaker_voices_detection_optim_name='adamw'_lr=0.0001_weight_decay=0.1_scheduler_name='reduce_on_plateau'_activation='ReLU'_dropout_p=0.0_use_bn=True_is_bn_pre_act=True_skip=False
0.7
0.6
0.5
0.4
steps/test_epoch
0
200
400
600
800


```python
data_params = {
    'dataset_name' : 'voices_spectograms',
    'dataset_params': {
        'custom_root': '/net/pr2/projects/plgrid/plggdnnp/datasets/VOiCES_devkit',
        'df_train_path': 'data/train_spectogram_df.csv',
        'df_val_path': 'data/val_spectogram_df.csv',
        'df_test_path': 'data/test_spectogram_df.csv',
        'use_transform': True, # if True, then use transforms for the base dataset (per side [CIFAR10 at this point])
    },
    'loader_params': {'batch_size': 128, 'pin_memory': True, 'num_workers': 12}
}
loaders = prepare_loaders(data_params)
test_loader = loaders['test']

import torch.nn.functional as F
from sklearn.metrics impor(parameter) test_loader: Any
def evaluate_model(model, test_loader):
    model.eval()
    all_preds = []
    all_labels = []
    with torch.no_grad():
        for data in tqdm(test_loader):
            x_true, y_true = data
            inputs, labels = x_true.to(device), y_true.to(device)
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    f1 = f1_score(all_labels, all_preds, average='binary')  # lub 'macro' dla wieloklasowej
    return f1

f1 = evaluate_model(model, test_loader)
print(f"Model F1-score on test set: {f1 * 100:.2f}%")
```

✓ 58.6s

```
100%|██████████| 9/9 [00:56<00:00,  6.30s/it]
Model F1-score on test set: 76.77%
```

```python
from src.utils.utils_data import prepare_loaders
data_params = {
    'dataset_name' : 'voices_spectograms',
    'dataset_params': {
        'custom_root': '/net/pr2/projects/plgrid/plggdnnp/datasets/VOiCES_devkit',
        'df_train_path': 'data/new_speaker_train_spectogram_df.csv',
        'df_val_path': 'data/new_speaker_val_spectogram_df.csv',
        'df_test_path': 'data/new_speaker_test_spectogram_df.csv',
        'use_transform': True, # if True, then use transforms for the base dataset (per side [CIFAR10 at this point]
    },
    'loader_params': {'batch_size': 128, 'pin_memory': True, 'num_workers': 12}
}
loaders = prepare_loaders(data_params)
test_loader = loaders['test']

import torch.nn.functional as F
from sklearn.metrics import f1_score
def evaluate_model(model, test_loader):
    model.eval()
    all_preds = []
    all_labels = []
    with torch.no_grad():
        for data in tqdm(test_loader):
            x_true, y_true = data
            inputs, labels = x_true.to(device), y_true.to(device)
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    f1 = f1_score(all_labels, all_preds, average='binary')  # lub 'macro' dla wieloklasowej
    return f1

f1 = evaluate_model(model, test_loader)
print(f"Model F1-score on test set: {f1 * 100:.2f}%")
```

(variable) labels: Any

✓ 4.9s

```
100%|██████████| 1/1 [00:04<00:00,  4.93s/it]
Model F1-score on test set: 76.92%
```