# Atrial Fibrillation Classification

The goal of this exercise is to train different neural networks to discriminate between atrial fibrillation and normal sinus rhythm from a sequence of interbeat intervals. We use interbeat intervals extracted from the Long Term AF Database (https://physionet.org/content/ltafdb/1.0.0/).

We will train the following models on windows of interbeat intervals:

- Logistic regression
- Multi-layer perceptron
- Convolutional neural network
- Recurrent neural network

In addition, the first two models will also be trained on simple features derived from each window of interbeat intervals.

First, we import all the required packages, define global constants, and seed the random number generators to obtain reproducible results.

In [1]:
```
%matplotlib widget


import itertools
import logging
import operator
import pathlib
import warnings

import IPython.display
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pytorch_lightning as pl
import scipy.special
import seaborn as sns
import sklearn.metrics
import sklearn.model_selection
import torch


DATA_FILE = pathlib.Path('../data/ltafdb_intervals.npz')
LOG_DIRECTORY = pathlib.Path('../logs/af_classification')


# Disable logging for PyTorch Lightning to avoid too long outputs.
logging.getLogger('pytorch_lightning').setLevel(logging.ERROR)

# Seed random number generators for reproducible results.
pl.seed_everything(42)
```

```
Seed set to 42
```

Out[1]: 42

Then, we load the windows of interbeat intervals and the corresponding labels. We also load the record identifiers. They will help to avoid using intervals from the same record for both training and testing.

In [2]:
```
def load_data():
    with np.load(DATA_FILE) as data:
        intervals = data['intervals']
        labels = data['labels']
        identifiers = data['identifiers']
    return intervals, labels, identifiers


intervals, labels, identifiers = load_data()
targets = (labels == 'atrial_fibrillation').astype('float32')[:, None]
window_size = intervals.shape[1]

print(f'Number of windows: {intervals.shape[0]}')
print(f'Window size: {window_size}')
print(f'Window labels: {set(labels)}')
```

```
Number of windows: 25064
Window size: 32
Window labels: {'atrial_fibrillation', 'normal_sinus_rhythm'}
```

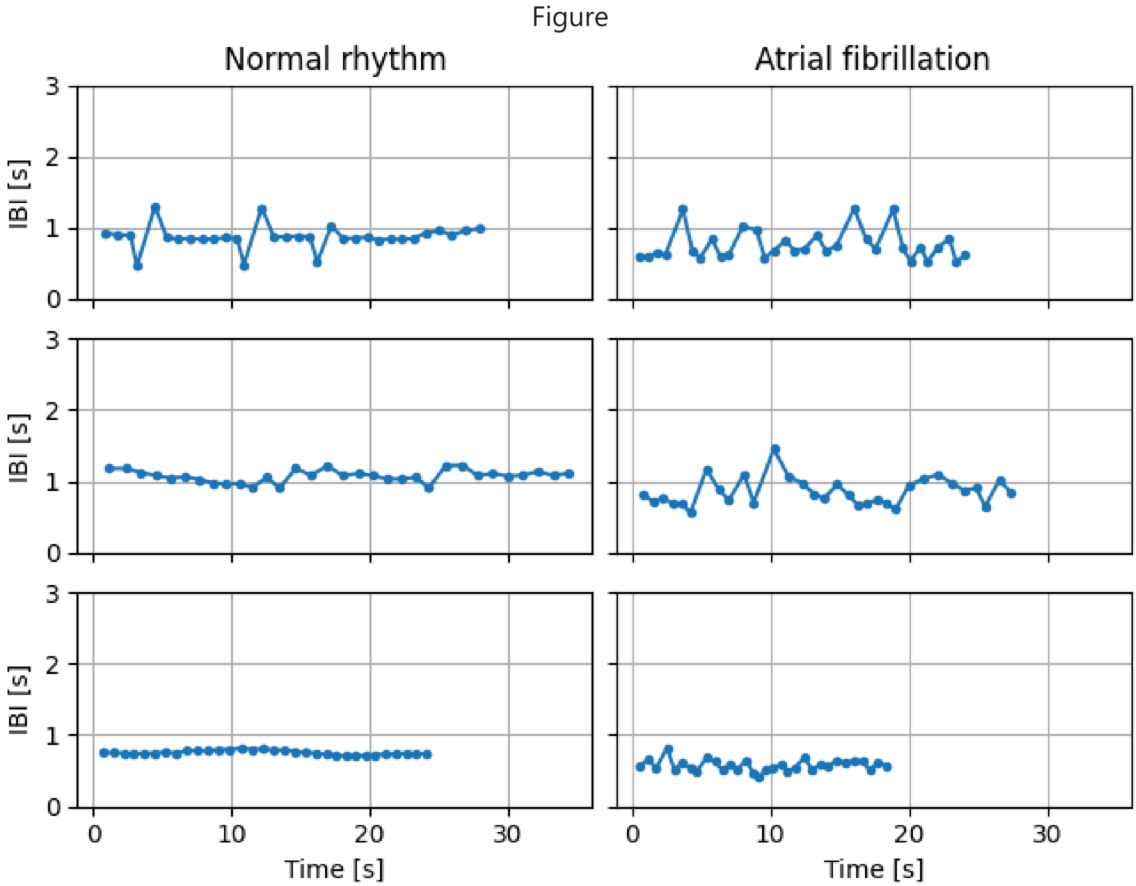
Here are a few examples of windows of interbeat intervals.

In [3]:
```
def plot_interval_examples(intervals, targets, n_examples=3):
    normal_indices = np.random.choice(np.flatnonzero(targets == 0.0), n_examples
    af_indices = np.random.choice(np.flatnonzero(targets == 1.0), n_examples, re

    def plot_intervals(ax, index):
        ax.plot(np.cumsum(intervals[index]), intervals[index], '.-')
        ax.grid(True)

    fig, axes = plt.subplots(n_examples, 2, sharex='all', sharey='all', squeeze=
    for i in range(n_examples):
        plot_intervals(axes[i, 0], normal_indices[i])
        plot_intervals(axes[i, 1], af_indices[i])
    plt.setp(axes, ylim=(0.0, 3.0))
    plt.setp(axes[-1, :], xlabel='Time [s]')
    plt.setp(axes[:, 0], ylabel='IBI [s]')
    axes[0, 0].set_title('Normal rhythm')
    axes[0, 1].set_title('Atrial fibrillation')


plot_interval_examples(intervals, targets)
```




**Question 1**

Visually, what are the differences between the examples of the two classes?

**Answer:** During atrial fibrillation, the average interbeat interval is lower than during normal rhythm and features significantly more variation.

The next step is to split the dataset into subsets for training, validation, and testing stratified by labels. We use the record identifiers to avoid using windows from the same record in more than one subest.

In [4]:
```
def split_data(identifiers, intervals, targets):
    splitter = sklearn.model_selection.StratifiedGroupKFold(n_splits=5)
    indices = list(map(operator.itemgetter(1), splitter.split(intervals, targets
    i_train = np.hstack(indices[:-2])
    i_val = indices[-2]
    i_test = indices[-1]

    assert not (set(identifiers[i_train]) & set(identifiers[i_val]))
    assert not (set(identifiers[i_train]) & set(identifiers[i_test]))
    assert not (set(identifiers[i_val]) & set(identifiers[i_test]))
    assert set(identifiers[i_train]) | set(identifiers[i_val]) | set(identifiers

    return i_train, i_val, i_test


i_train, i_val, i_test = split_data(identifiers, intervals, targets)


def build_summary(subsets, targets):
```

```
    data = []
    for subset, y in zip(subsets, targets):
        data.append({
            'subset': subset,
            'total_count': y.size,
            'normal_count': np.sum(y == 0.0),
            'af_count': np.sum(y == 1.0),
            'normal_proportion': np.mean(y == 0.0),
            'af_proportion': np.mean(y == 1.0),
        })
    return pd.DataFrame(data)


IPython.display.display(build_summary(('training', 'validation', 'testing'), (ta
```

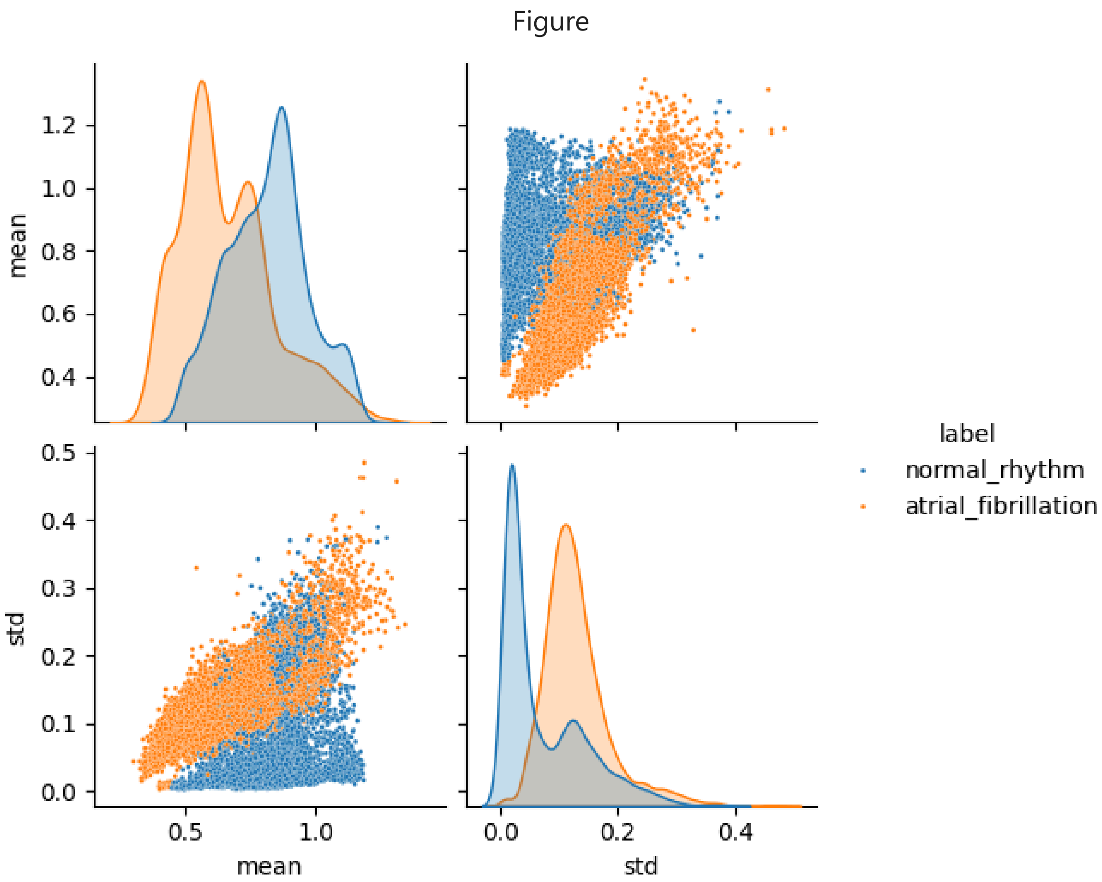
| | subset | total_count | normal_count | af_count | normal_proportion | af_proportion |
|---|---|---|---|---|---|---|
| **0** | training | 15000 | 6919 | 8081 | 0.461267 | 0.538733 |
| **1** | validation | 4964 | 2365 | 2599 | 0.476430 | 0.523570 |
| **2** | testing | 4964 | 2365 | 2599 | 0.476430 | 0.523570 |

To better understand the dataset, we extract two features from each window of interbeat intervals: the mean and the standard deviation. We then plot these two features for the two classes.

In [5]:
```
features = np.vstack((
    np.mean(intervals, axis=1),
    np.std(intervals, axis=1),
)).T


def plot_features(f, y):
    data = pd.DataFrame({
        'label': y.ravel(),
        'mean': f[:, 0],
        'std': f[:, 1],
    })
    data['label'] = data['label'].map({0.0: 'normal_rhythm', 1.0: 'atrial_fibril
    sns.pairplot(data, hue='label', plot_kws={'s': 4})


plot_features(features[i_train], targets[i_train])
```




**Question 2**

Would it be possible to discriminate between the two classes with these features and a linear classifier?

**Answer:** No. Although the main mode of the mean and std distribution of each class are clearly separated, there is a significant overlap between the distributions. There is no hyperplane that can entirely separate the two set of points.

To classify atrial fibrillation and normal rhythm, we define three models: a multi-layer perceptron (MLP), a convolutional neural network (CNN), and a recurrent neural network (RNN).

Make sure you understand the differences between these models.

In [6]:
```
class MlpModel(torch.nn.Module):

    def __init__(self, input_size, output_size, n_hidden_layers=1, n_units=128):
        super().__init__()
        self.input_size = input_size
        self.output_size = output_size
        self.n_hidden_layers = n_hidden_layers
        self.n_units = n_units
        self.layers = self._build_layers()

    def _build_layers(self):
        sizes = [self.input_size]
        sizes.extend(itertools.repeat(self.n_units, self.n_hidden_layers))
        layers = []
```

```
        for i in range(self.n_hidden_layers):
            layers.append(torch.nn.Linear(sizes[i], sizes[i + 1]))
            layers.append(torch.nn.ReLU())
        layers.append(torch.nn.Linear(sizes[-1], self.output_size))
        return torch.nn.Sequential(*layers)

    def forward(self, x):
        return self.layers(x)


class CnnModel(torch.nn.Module):

    def __init__(self,
                 input_size,
                 output_size,
                 n_layers=3,
                 n_initial_channels=8,
                 kernel_size=3):
        super().__init__()
        self.input_size = input_size
        self.output_size = output_size
        self.n_layers = n_layers
        self.n_initial_channels = n_initial_channels
        self.kernel_size = kernel_size
        self.layers = self._build_layers()

    def _build_layers(self):
        layers = []

        # Build convolutional layers. Each layer is composed of a convolution,
        # a ReLU activation, and max pooling (except for the last convolutional
        # layer that uses global average pooling).
        n_output_channels = 1
        for i in range(self.n_layers):
            n_input_channels = n_output_channels
            n_output_channels = self.n_initial_channels * 2 ** i
            layers.append(torch.nn.Conv1d(
                in_channels=n_input_channels,
                out_channels=n_output_channels,
                kernel_size=self.kernel_size,
                padding='same',
            ))
            layers.append(torch.nn.ReLU())
            if i < self.n_layers - 1:
                layers.append(torch.nn.MaxPool1d(kernel_size=2))
            else:
                layers.append(torch.nn.AdaptiveAvgPool1d(1))
        layers.append(torch.nn.Flatten())

        # Build output layer.
        layers.append(torch.nn.Linear(
            in_features=self.n_initial_channels * 2 ** (self.n_layers - 1),
            out_features=self.output_size,
        ))

        return torch.nn.Sequential(*layers)

    def forward(self, x):
        x = x.view(x.shape[0], 1, x.shape[1])
        return self.layers(x)
```

```python
class RnnModel(torch.nn.Module):

    def __init__(self, input_size, output_size, hidden_size=64):
        super().__init__()
        self.input_size = input_size
        self.output_size = output_size
        self.hidden_size = hidden_size
        self.recurrent_layer = torch.nn.GRU(
            input_size=1,
            hidden_size=self.hidden_size,
            num_layers=1,
            batch_first=False,
        )
        self.output_layer = torch.nn.Linear(self.hidden_size, self.output_size)

    def forward(self, x):
        x = x.transpose(0, 1)[..., None]
        y, h = self.recurrent_layer(x)
        return self.output_layer(y[-1])
```

We also define a class to manage different models and a few functions to train and evaluate models.

In [7]:
```python
class Classifier(pl.LightningModule):

    def __init__(self, config):
        super().__init__()
        self.save_hyperparameters()
        self.config = config
        self.model = self._build_model()
        self.example_input_array = torch.zeros((1, self.model.input_size))

    def _build_model(self):
        name = self.config['model']['name']
        config = self.config['model'].get('config', {})
        if name == 'mlp':
            return MlpModel(**config)
        elif name == 'cnn':
            return CnnModel(**config)
        elif name == 'rnn':
            return RnnModel(**config)
        else:
            raise ValueError(f'unknown model: {name!r}')

    def configure_optimizers(self):
        name = self.config['optimizer']['name']
        config = self.config['optimizer'].get('config', {})
        if name == 'sgd':
            return torch.optim.SGD(self.parameters(), **config)
        elif name == 'adam':
            return torch.optim.Adam(self.parameters(), **config)
        else:
            raise ValueError(f'unknown optimizer: {name!r}')

    def forward(self, x):
        return self.model(x)
```

```
    def training_step(self, batch, batch_idx):
        return self._run_step(batch, 'train')

    def validation_step(self, batch, batch_idx):
        self._run_step(batch, 'val')

    def test_step(self, batch, batch_idx):
        self._run_step(batch, 'test')

    def predict_step(self, batch, batch_idx, dataloader_idx=0):
        x, y = batch
        return self.model(x)

    def _run_step(self, batch, subset):
        x, y = batch
        logits = self.model(x)
        loss = torch.nn.functional.binary_cross_entropy_with_logits(logits, y)
        acc = ((logits > 0.0).float() == y).float().mean()
        self.log_dict({f'{subset}_loss': loss, f'{subset}_acc': acc},
                      on_step=False, on_epoch=True, prog_bar=True)
        return loss


def build_loader(*tensors, batch_size=100, shuffle=False, n_workers=0):
    dataset = torch.utils.data.TensorDataset(*map(torch.Tensor, tensors))
    return torch.utils.data.DataLoader(
        dataset=dataset,
        batch_size=batch_size,
        shuffle=shuffle,
        num_workers=n_workers,
    )

def train_model(name, config, x, y, i_train, i_val, n_epochs=10, batch_size=100)
    train_loader = build_loader(x[i_train], y[i_train], batch_size=batch_size, s
    val_loader = build_loader(x[i_val], y[i_val], batch_size=batch_size)
    classifier = Classifier(config)
    print(pl.utilities.model_summary.ModelSummary(classifier, max_depth=-1))

    with warnings.catch_warnings():
        warnings.simplefilter('ignore')
        trainer = pl.Trainer(
            default_root_dir=LOG_DIRECTORY,
            logger=pl.loggers.TensorBoardLogger(LOG_DIRECTORY, name),
            enable_model_summary=False,
            max_epochs=n_epochs,
        )
        trainer.fit(classifier, train_loader, val_loader)

    return classifier


def compute_metrics(targets, predictions, threshold=0.5):
    targets = np.ravel(targets) > threshold
    predictions = np.ravel(predictions) > threshold
    c = sklearn.metrics.confusion_matrix(targets, predictions)
    tp = c[1, 1]
    tn = c[0, 0]
    fp = c[0, 1]
    fn = c[1, 0]
    return {
```

```
        'count': c.sum(),
        'acc': (tp + tn) / (tp + tn + fp + fn),
        'tpr': tp / (tp + fn),
        'tnr': tn / (tn + fp),
        'ppv': tp / (tp + fp),
        'npv': tn / (tn + fn),
        'f1': 2 * tp / (2 * tp + fp + fn),
    }


def evaluate_model(model, x, y, i_train, i_val, i_test, batch_size=100):
    loader = build_loader(x, y, batch_size=batch_size)

    with warnings.catch_warnings():
        warnings.simplefilter('ignore')
        trainer = pl.Trainer(
            default_root_dir=LOG_DIRECTORY,
            logger=False,
            enable_progress_bar=False,
            enable_model_summary=False,
        )
        z = trainer.predict(model, loader)
    z = np.vstack([u.numpy() for u in z])
    z = scipy.special.expit(z)  # Convert logits to probabilities.

    metrics = []
    for subset, indices in (('train', i_train), ('val', i_val), ('test', i_test)
        metrics.append({
            'subset': subset,
            **compute_metrics(y[indices], z[indices]),
        })

    return pd.DataFrame(metrics)
```

The final step before training and evaluating models is to define the configurations of the different models.

We will train the following models:

- Features as inputs (input size = 2):
  - Logsitic regression
  - Multi-layer perceptron
    - Dense layer (output size = 128)
    - ReLU activation
    - Dense layer (output size = 128)
    - ReLU activation
    - Dense layer (output size = 1)
- Interbeat intervals as inputs (input_size = 32):
  - Logsitic regression
  - Multi-layer perceptron
    - Dense layer (output size = 128)
    - ReLU activation
    - Dense layer (output size = 128)
    - ReLU activation

        - Dense layer (output size = 1)
    - Convolutional neural network
        - Convolutional layer (kernel size = 3, output size = 32, output channels = 8)
        - ReLU activation
        - Max pooling (output size = 16, output channels = 8)
        - Convolutional layer (kernel size = 3, output size = 16, output channels = 16)
        - ReLU activation
        - Max pooling (output size = 8, output channels = 16)
        - Convolutional layer (kernel size = 3, output size = 8, output channels = 32)
        - ReLU activation
        - Global average pooling (output size = 32)
        - Dense layer (output size = 1)
    - Recurrent neural network
        - GRU layer (output size = 64)
        - Dense layer (output size = 1)

In [8]:
```
n_epochs = 40
batch_size = 200

configs = {
    'features_logistic': {
        'model': {
            'name': 'mlp',
            'config': {
                'input_size': features.shape[1],
                'output_size': targets.shape[1],
                'n_hidden_layers': 0,  # A muli-layer perceptron without hidden
            },
        },
        'optimizer': {
            'name': 'adam',
            'config': {'lr': 0.01},
        },
    },
    'features_mlp': {
        'model': {
            'name': 'mlp',
            'config': {
                'input_size': features.shape[1],
                'output_size': targets.shape[1],
                'n_hidden_layers': 2,
                'n_units': 128,
            },
        },
        'optimizer': {
            'name': 'adam',
            'config': {'lr': 0.001},
        },
    },
    'logistic': {
        'model': {
            'name': 'mlp',
            'config': {
```

```
                'input_size': intervals.shape[1],
                'output_size': targets.shape[1],
                'n_hidden_layers': 0,  # A muli-layer perceptron without hidden
            },
        },
        'optimizer': {
            'name': 'adam',
            'config': {'lr': 0.001},
        },
    },
    'mlp': {
        'model': {
            'name': 'mlp',
            'config': {
                'input_size': intervals.shape[1],
                'output_size': targets.shape[1],
                'n_hidden_layers': 2,
                'n_units': 128,
            },
        },
        'optimizer': {
            'name': 'adam',
            'config': {'lr': 0.001},
        },
    },
    'cnn': {
        'model': {
            'name': 'cnn',
            'config': {
                'input_size': intervals.shape[1],
                'output_size': targets.shape[1],
                'n_layers': 3,
                'n_initial_channels': 8,
                'kernel_size': 3,
            },
        },
        'optimizer': {
            'name': 'adam',
            'config': {'lr': 0.001},
        },
    },
    'rnn': {
        'model': {
            'name': 'rnn',
            'config': {
                'input_size': intervals.shape[1],
                'output_size': targets.shape[1],
                'hidden_size': 64,
            },
        },
        'optimizer': {
            'name': 'adam',
            'config': {'lr': 0.001},
        },
    },

}
```

To visualize the loss and accuracy during training, we start TensorBoard.

If you prefer to view TensorBoard in a separate window, you can open http://localhost:6006/ in your web browser.

In [9]:
```
%reload_ext tensorboard
%tensorboard --logdir ../logs/af_classification --port 6006
```

# Index de C:\

| Nom | Taille | Date de modification |
|---|---|---|
| $Recycle.Bin/ | | 23/11/2023 20:06:09 |
| AppData/ | | 09/12/2023 12:20:38 |
| Config.Msi/ | | 22/11/2024 19:32:34 |
| DATA/ | | 23/11/2023 21:31:22 |
| Dokumente und Einstellungen/ | | 23/11/2023 17:25:38 |
| gurobi1003/ | | 24/05/2024 19:45:05 |
| hp/ | | 02/09/2023 15:07:32 |
| Kairos/ | | 09/12/2023 12:21:05 |
| OneDriveTemp/ | | 13/05/2024 09:21:25 |
| PerfLogs/ | | 07/05/2022 07:24:50 |
| Program Files/ | | 12/12/2024 09:39:35 |
| Program Files (x86)/ | | 03/10/2024 15:05:28 |
| ProgramData/ | | 22/11/2024 19:32:39 |
| Programme/ | | 23/11/2023 17:25:38 |
| Recovery/ | | 23/11/2023 19:49:01 |
| System Volume Information/ | | 09/12/2024 19:44:33 |
| System.sav/ | | 03/09/2023 02:01:15 |
| Users/ | | 23/11/2023 20:57:48 |
| Windows/ | | 13/11/2024 03:36:39 |
| DumpStack.log.tmp | 12.0 kB | 22/11/2024 19:32:37 |
| hiberfil.sys | 6.3 GB | 12/12/2024 09:38:15 |
| install.log | 1015 kB | 09/12/2023 12:21:15 |
| pagefile.sys | 31.0 GB | 09/12/2024 20:19:31 |
| Rapport-batterie.html | 83.5 kB | 02/12/2024 11:32:23 |
| swapfile.sys | 16.0 MB | 22/11/2024 19:32:37 |

Finally, we train the different models. It will take a few minutes.

In [10]:
```
models = {}
for name, config in configs.items():
    print(f'Training {name!r} model')
```

```
    x = features if name.startswith('features') else intervals
    y = targets
    models[name] = train_model(name, config, x, y, i_train, i_val, n_epochs, bat
    print('\n\n\n')  # Add blank lines to separate models in the output.
```

```
Training 'features_logistic' model
  | Name           | Type       | Params | Mode  | In sizes | Out sizes
----------------------------------------------------------------------------
0 | model          | MlpModel   | 3      | train | [1, 2]   | [1, 1]
1 | model.layers   | Sequential | 3      | train | [1, 2]   | [1, 1]
2 | model.layers.0 | Linear     | 3      | train | [1, 2]   | [1, 1]
----------------------------------------------------------------------------
3         Trainable params
0         Non-trainable params
3         Total params
0.000     Total estimated model params size (MB)
3         Modules in train mode
0         Modules in eval mode
Sanity Checking: |          | 0/? [00:00<?, ?it/s]
Training: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
Training 'features_mlp' model
  | Name           | Type       | Params | Mode  | In sizes | Out sizes
------------------------------------------------------------------------------
0 | model          | MlpModel   | 17.0 K | train | [1, 2]   | [1, 1]
1 | model.layers   | Sequential | 17.0 K | train | [1, 2]   | [1, 1]
2 | model.layers.0 | Linear     | 384    | train | [1, 2]   | [1, 128]
3 | model.layers.1 | ReLU       | 0      | train | [1, 128] | [1, 128]
4 | model.layers.2 | Linear     | 16.5 K | train | [1, 128] | [1, 128]
5 | model.layers.3 | ReLU       | 0      | train | [1, 128] | [1, 128]
6 | model.layers.4 | Linear     | 129    | train | [1, 128] | [1, 1]
------------------------------------------------------------------------------
17.0 K    Trainable params
0         Non-trainable params
17.0 K    Total params
0.068     Total estimated model params size (MB)
7         Modules in train mode
0         Modules in eval mode
Sanity Checking: |          | 0/? [00:00<?, ?it/s]
Training: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]



Training 'logistic' model
  | Name           | Type       | Params | Mode  | In sizes | Out sizes
----------------------------------------------------------------------------
0 | model          | MlpModel   | 33     | train | [1, 32]  | [1, 1]
1 | model.layers   | Sequential | 33     | train | [1, 32]  | [1, 1]
2 | model.layers.0 | Linear     | 33     | train | [1, 32]  | [1, 1]
----------------------------------------------------------------------------
33        Trainable params
0         Non-trainable params
33        Total params
0.000     Total estimated model params size (MB)
3         Modules in train mode
0         Modules in eval mode
Sanity Checking: |          | 0/? [00:00<?, ?it/s]
Training: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
Training 'mlp' model
  | Name           | Type       | Params | Mode  | In sizes | Out sizes
---------------------------------------------------------------------------
0 | model          | MlpModel   | 20.9 K | train | [1, 32]  | [1, 1]
1 | model.layers   | Sequential | 20.9 K | train | [1, 32]  | [1, 1]
2 | model.layers.0 | Linear     | 4.2 K  | train | [1, 32]  | [1, 128]
3 | model.layers.1 | ReLU       | 0      | train | [1, 128] | [1, 128]
4 | model.layers.2 | Linear     | 16.5 K | train | [1, 128] | [1, 128]
5 | model.layers.3 | ReLU       | 0      | train | [1, 128] | [1, 128]
6 | model.layers.4 | Linear     | 129    | train | [1, 128] | [1, 1]
---------------------------------------------------------------------------
20.9 K    Trainable params
0         Non-trainable params
20.9 K    Total params
0.083     Total estimated model params size (MB)
7         Modules in train mode
0         Modules in eval mode
Sanity Checking: |          | 0/? [00:00<?, ?it/s]
Training: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]



Training 'cnn' model
   | Name            | Type              | Params | Mode  | In sizes    | Out siz
es
--------------------------------------------------------------------------------
----------
0  | model           | CnnModel          | 2.0 K  | train | [1, 32]     | [1, 1]
1  | model.layers    | Sequential        | 2.0 K  | train | [1, 1, 32]  | [1, 1]
2  | model.layers.0  | Conv1d            | 32     | train | [1, 1, 32]  | [1, 8,
32]
3  | model.layers.1  | ReLU              | 0      | train | [1, 8, 32]  | [1, 8,
32]
4  | model.layers.2  | MaxPool1d         | 0      | train | [1, 8, 32]  | [1, 8,
16]
5  | model.layers.3  | Conv1d            | 400    | train | [1, 8, 16]  | [1, 16,
16]
6  | model.layers.4  | ReLU              | 0      | train | [1, 16, 16] | [1, 16,
16]
7  | model.layers.5  | MaxPool1d         | 0      | train | [1, 16, 16] | [1, 16,
8]
8  | model.layers.6  | Conv1d            | 1.6 K  | train | [1, 16, 8]  | [1, 32,
8]
9  | model.layers.7  | ReLU              | 0      | train | [1, 32, 8]  | [1, 32,
8]
10 | model.layers.8  | AdaptiveAvgPool1d | 0      | train | [1, 32, 8]  | [1, 32,
1]
11 | model.layers.9  | Flatten           | 0      | train | [1, 32, 1]  | [1, 32]
12 | model.layers.10 | Linear            | 33     | train | [1, 32]     | [1, 1]
--------------------------------------------------------------------------------
----------
2.0 K     Trainable params
0         Non-trainable params
2.0 K     Total params
0.008     Total estimated model params size (MB)
13        Modules in train mode
0         Modules in eval mode
Sanity Checking: |          | 0/? [00:00<?, ?it/s]
Training: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]


Training 'rnn' model
  | Name                  | Type     | Params | Mode  | In sizes    | Out sizes
--------------------------------------------------------------------------------
-------------------
0 | model                 | RnnModel | 12.9 K | train | [1, 32]     | [1, 1]
1 | model.recurrent_layer | GRU      | 12.9 K | train | [32, 1, 1]  | [[32, 1, 6
4], [1, 1, 64]]
2 | model.output_layer    | Linear   | 65     | train | [1, 64]     | [1, 1]
--------------------------------------------------------------------------------
-------------------
12.9 K    Trainable params
0         Non-trainable params
12.9 K    Total params
0.052     Total estimated model params size (MB)
3         Modules in train mode
0         Modules in eval mode
Sanity Checking: |          | 0/? [00:00<?, ?it/s]
Training: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

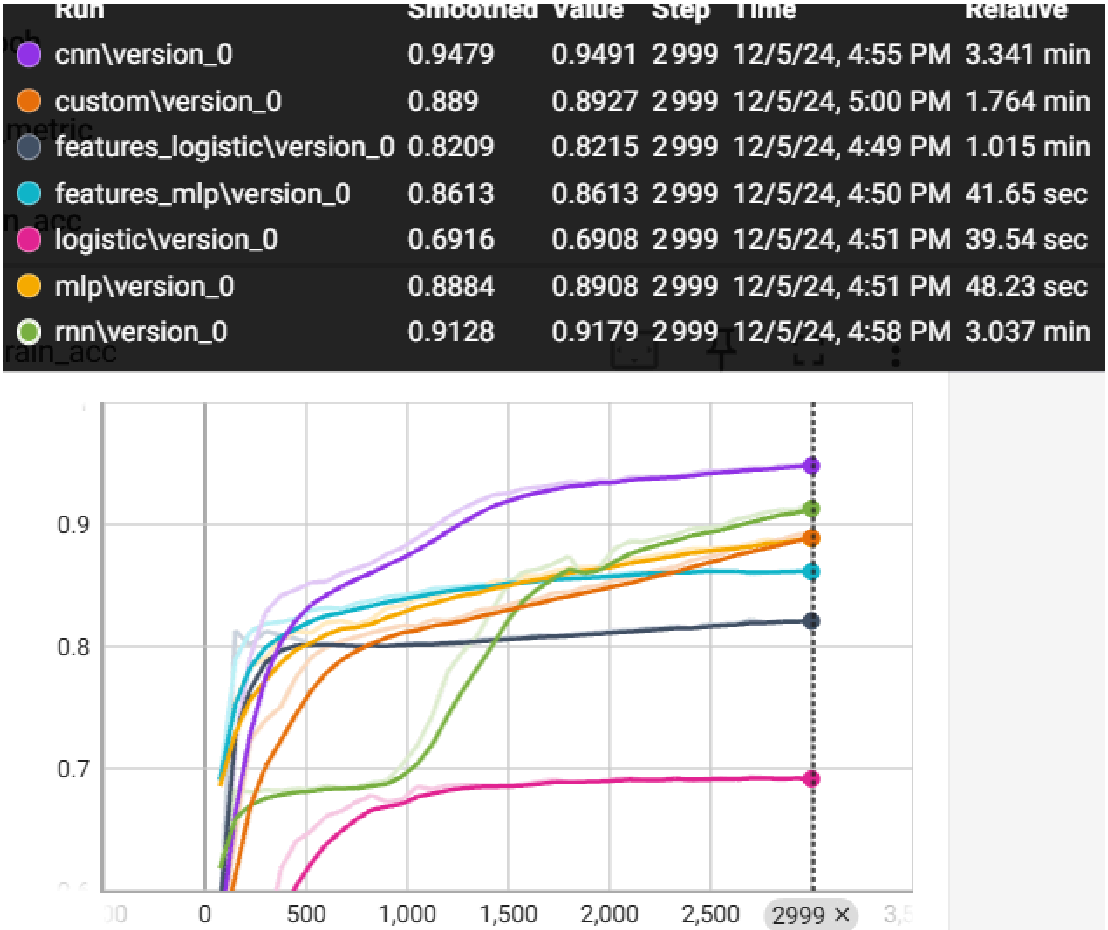
```
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

**Question 3**

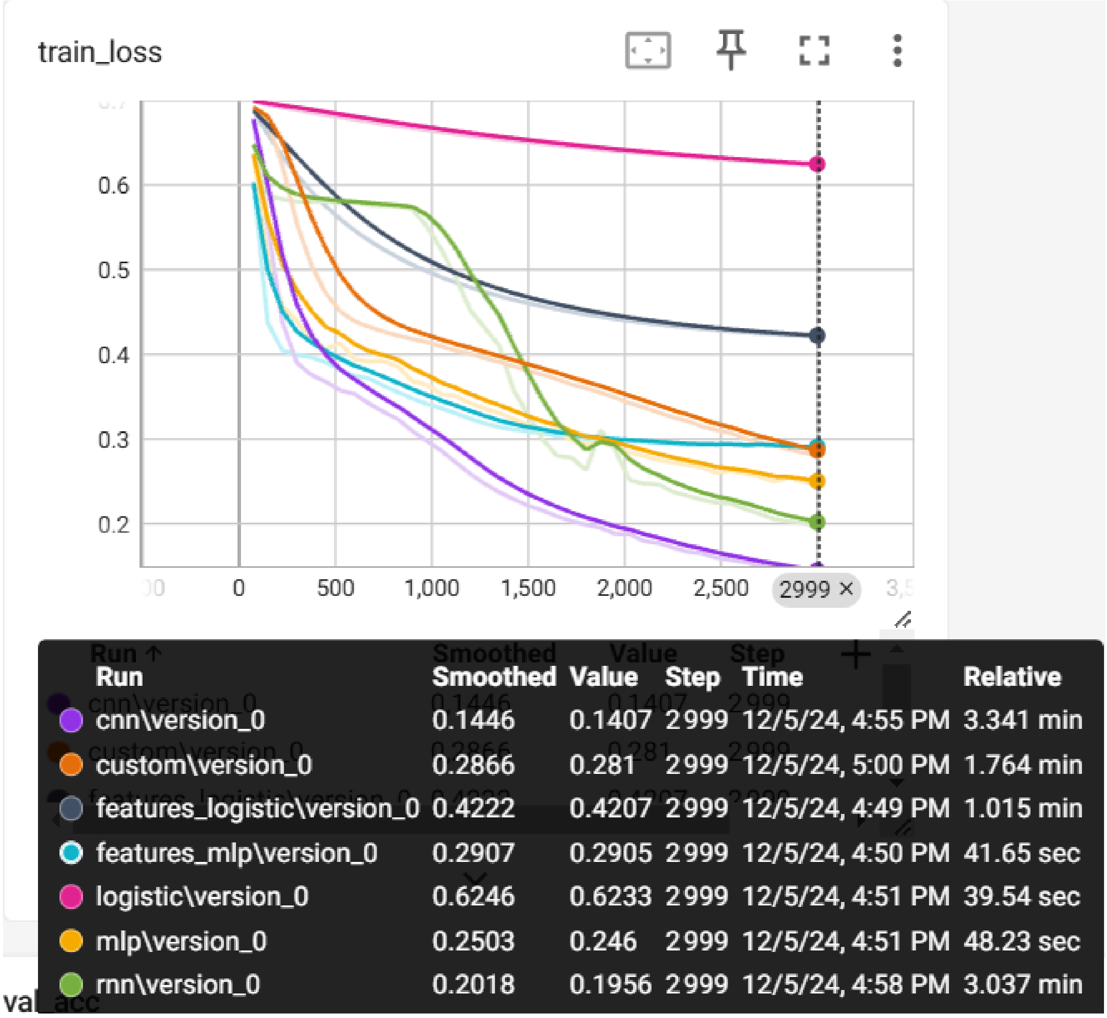
Based on the model summaries printed above and the metrics shown in TensorBoard, answer the following questions:

1. After training is finished, you can see the number of parameters for each model. Why is the number of parameters for the CNN model much lower than for the MLP model?
2. What can you about the loss and accuracy of the different models on the training and validation subsets? Do some model overfit?
3. Why does the logistic regression that takes features as inputs performs much better than the logistic regression that takes raw interbeat intervals as inputs?
4. Are there models that would benefit from training for more epochs?

# Train Accuracy

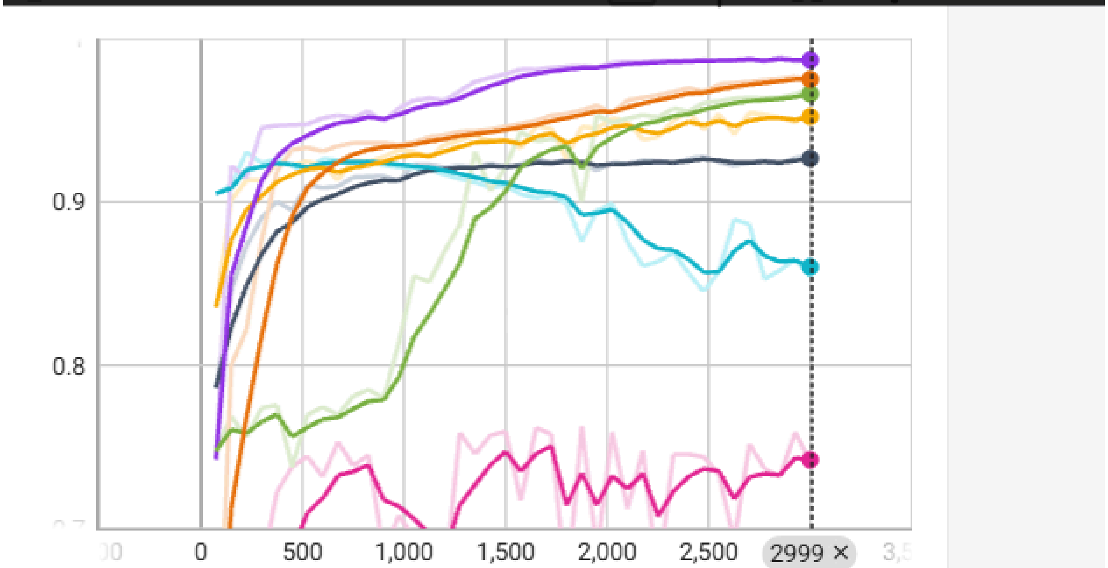

Run Smoothed Value Step Time Relative
cnn\version_0 0.9479 0.9491 2999 12/5/24, 4:55 PM 3.341 min
custom\version_0 0.889 0.8927 2999 12/5/24, 5:00 PM 1.764 min
features_logistic\version_0 0.8209 0.8215 2999 12/5/24, 4:49 PM 1.015 min
features_mlp\version_0 0.8613 0.8613 2999 12/5/24, 4:50 PM 41.65 sec
logistic\version_0 0.6916 0.6908 2999 12/5/24, 4:51 PM 39.54 sec
mlp\version_0 0.8884 0.8908 2999 12/5/24, 4:51 PM 48.23 sec
rnn\version_0 0.9128 0.9179 2999 12/5/24, 4:58 PM 3.037 min
0.9
0.8
0.7
0
500
1,000
1,500
2,000
2,500
2999

# Train Loss

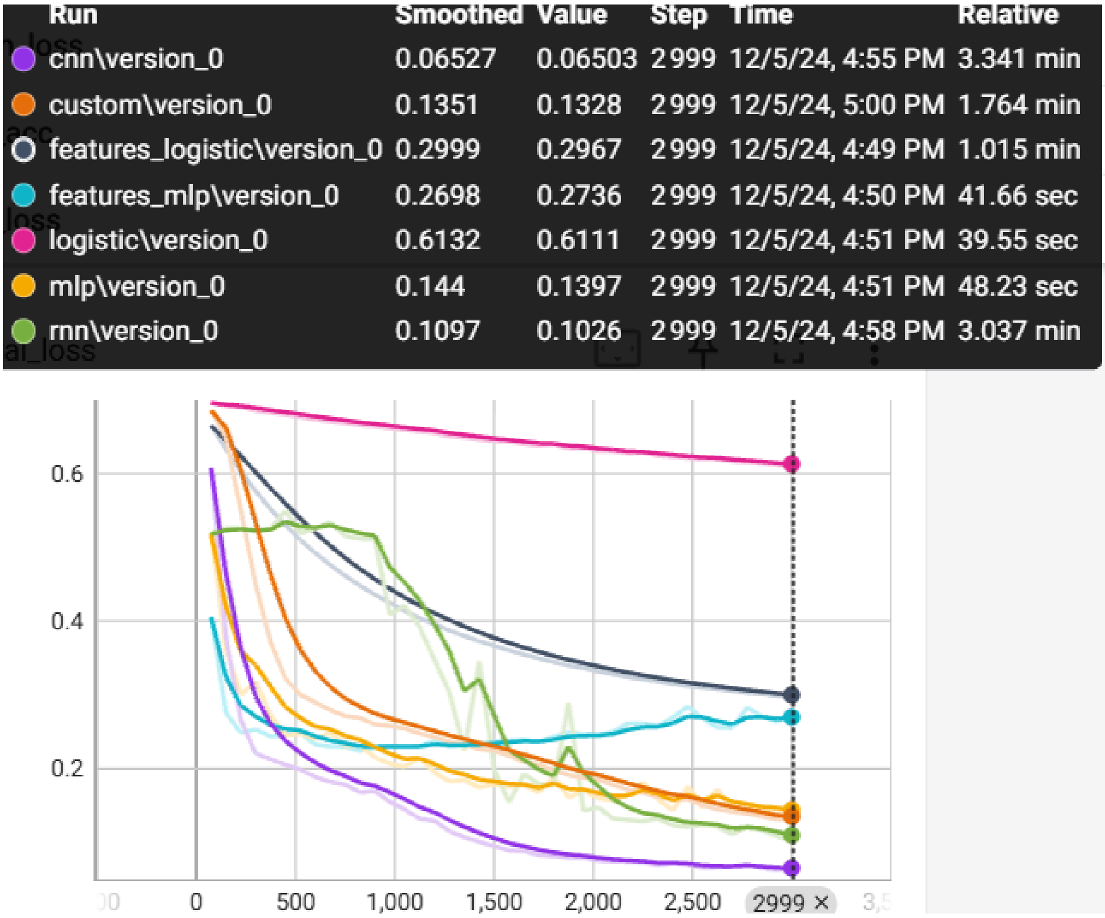



# Validation Accuracy

| Run | Smoothed | Value | Step | Time | Relative |
|---|---|---|---|---|---|
| cnn\version_0 | 0.9869 | 0.9869 | 2999 | 12/5/24, 4:55 PM | 3.341 min |
| custom\version_0 | 0.975 | 0.975 | 2999 | 12/5/24, 5:00 PM | 1.764 min |
| features_logistic\version_0 | 0.9268 | 0.9289 | 2999 | 12/5/24, 4:49 PM | 1.015 min |
| features_mlp\version_0 | 0.8602 | 0.8544 | 2999 | 12/5/24, 4:50 PM | 41.66 sec |
| logistic\version_0 | 0.7422 | 0.7405 | 2999 | 12/5/24, 4:51 PM | 39.55 sec |
| mlp\version_0 | 0.9523 | 0.9553 | 2999 | 12/5/24, 4:51 PM | 48.23 sec |
| rnn\version_0 | 0.9663 | 0.9692 | 2999 | 12/5/24, 4:58 PM | 3.037 min |




## Validation Loss



| Run | Smoothed | Value | Step | Time | Relative |
|---|---|---|---|---|---|
| cnn\version_0 | 0.06527 | 0.06503 | 2999 | 12/5/24, 4:55 PM | 3.341 min |
| custom\version_0 | 0.1351 | 0.1328 | 2999 | 12/5/24, 5:00 PM | 1.764 min |
| features_logistic\version_0 | 0.2999 | 0.2967 | 2999 | 12/5/24, 4:49 PM | 1.015 min |
| features_mlp\version_0 | 0.2698 | 0.2736 | 2999 | 12/5/24, 4:50 PM | 41.66 sec |
| logistic\version_0 | 0.6132 | 0.6111 | 2999 | 12/5/24, 4:51 PM | 39.55 sec |
| mlp\version_0 | 0.144 | 0.1397 | 2999 | 12/5/24, 4:51 PM | 48.23 sec |
| rnn\version_0 | 0.1097 | 0.1026 | 2999 | 12/5/24, 4:58 PM | 3.037 min |



**Answer:**

1. A convolution is a special case of a linear transformation. While a linear layer is defined by the $m \times n$ coefficients of the associated matrix, where $m$ and $n$ are the output and input dimensions respectively, a convolutional layer is defined by its kernel, which is much smaller than the input and output dimensions. This allows the CNN model to have less parameters in total despite having more layers, as each layer has much less parameters.
2. All models show a validation accuracy higher than their training accuracy, suggesting that they are not overfitting. The exception is the features_MLP because the Validation Loss is increasing at later epochs, indicating overfitting. Overall the best model is the CNN model and the worst is the Logistic one. The CNN and RNN are able to capture the temporal dependencies between subsequent data samples thanks to their architectures.
3. The logistic model cannot learn non-linear features such as the variance, which we have seen to be significantly different between atrial fibrillation and normal fibrillation. Hence, the model performs much better when these features are used as inputs.
4. Training more epochs will only benefit models that have not yet reached a plateau in their loss curves. The CNN reaches a plateau in its validation loss and the `features_mlp` clearly has an increasing validation loss at later epochs. Therefore, those models will not benefit from additional epochs. We guess that the MLP and RNN will continue improving with further training.

Now that all models are trained we can evaluate them on the subsets for training, validation, and testing. For each model, we compute the following metrics:

- Accuracy (ACC)
- True positive rate (TPR)
- True negative rate (TNR)
- Positive predictive value (PPV)
- Negative predictive value (NPV)
- F1 score (F1)

You can find more information about these metrics on Wikipedia.

In [11]:
```
metrics = []
for name, model in models.items():
    x = features if name.startswith('features') else intervals
    y = targets
    df = evaluate_model(model, x, y, i_train, i_val, i_test, batch_size=batch_si
    df.insert(0, 'model', name)
    metrics.append(df)
metrics = pd.concat(metrics, axis=0, ignore_index=True)
metrics = metrics.set_index(['model', 'subset'])
index = metrics.index.get_level_values(0).unique()
columns = pd.MultiIndex.from_product([metrics.columns, metrics.index.get_level_v
metrics = metrics.unstack().reindex(index=index, columns=columns)
IPython.display.display(metrics)
```

| | count | | | acc | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **subset** | **train** | **val** | **test** | **train** | **val** | **test** | **train** | **val** | |
| **model** | | | | | | | | | |
| **features_logistic** | 15000 | 4964 | 5100 | 0.823400 | 0.928888 | 0.917647 | 0.875139 | 0.989227 | 0.9 |
| **features_mlp** | 15000 | 4964 | 5100 | 0.860467 | 0.854351 | 0.889804 | 0.921297 | 0.851866 | 0.8 |
| **logistic** | 15000 | 4964 | 5100 | 0.691333 | 0.740532 | 0.653922 | 0.726519 | 0.724509 | 0.6 |
| **mlp** | 15000 | 4964 | 5100 | 0.887733 | 0.955278 | 0.933922 | 0.955699 | 0.973836 | 0.9 |
| **cnn** | 15000 | 4964 | 5100 | 0.948400 | 0.986906 | 0.969216 | 0.958050 | 0.982301 | 0.9 |
| **rnn** | 15000 | 4964 | 5100 | 0.920200 | 0.969178 | 0.950588 | 0.939611 | 0.969604 | 0.9 |

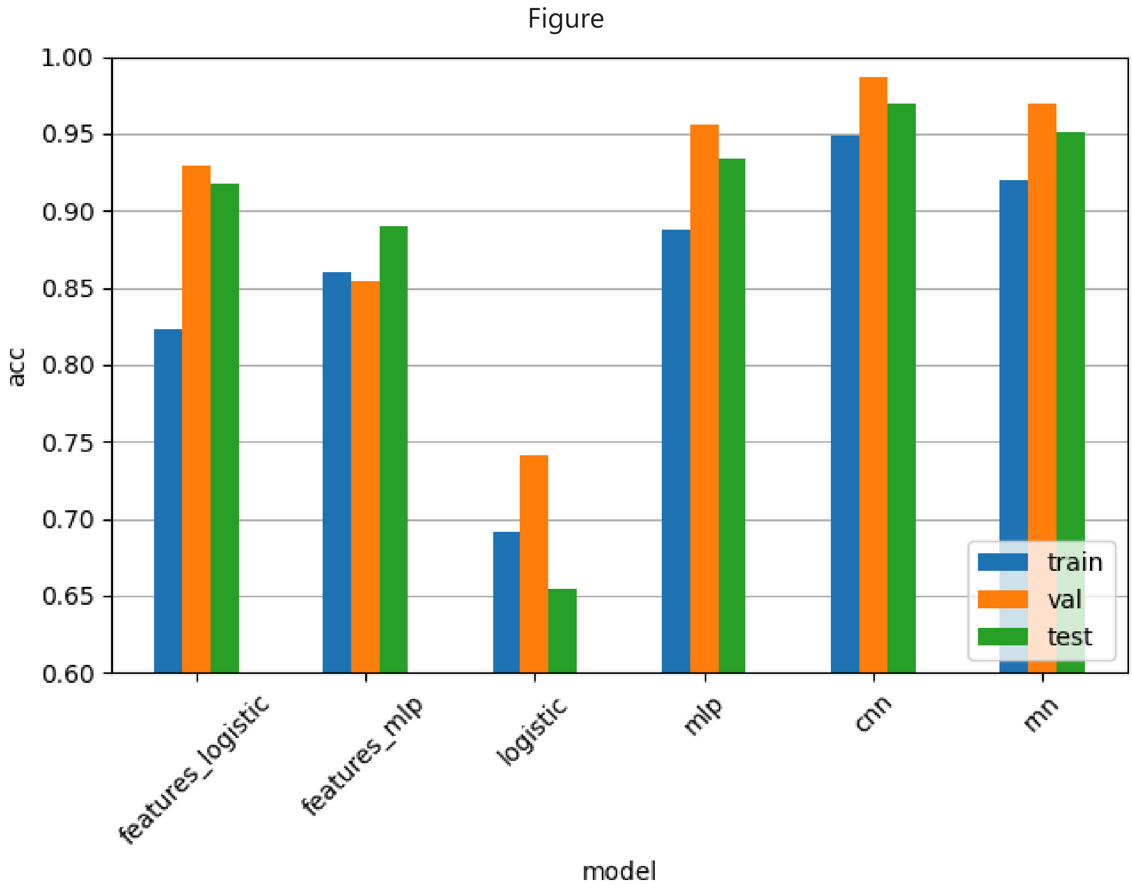
6 rows × 21 columns

We can also plot the different metrics.

In [12]:
```
def plot_metrics(data):
    for metric in data.columns.get_level_values(0).unique():
        if metric == 'count':
            continue
        df = data[metric]
        plt.figure(constrained_layout=True)
        plt.gca().set_axisbelow(True)
        df.plot(kind='bar', ylabel=metric, ax=plt.gca())
```

```
        plt.grid(axis='y')
        plt.ylim(0.6, 1.0)
        plt.legend(loc='lower right')
        plt.gca().xaxis.set_tick_params(rotation=45)


plot_metrics(metrics)
```



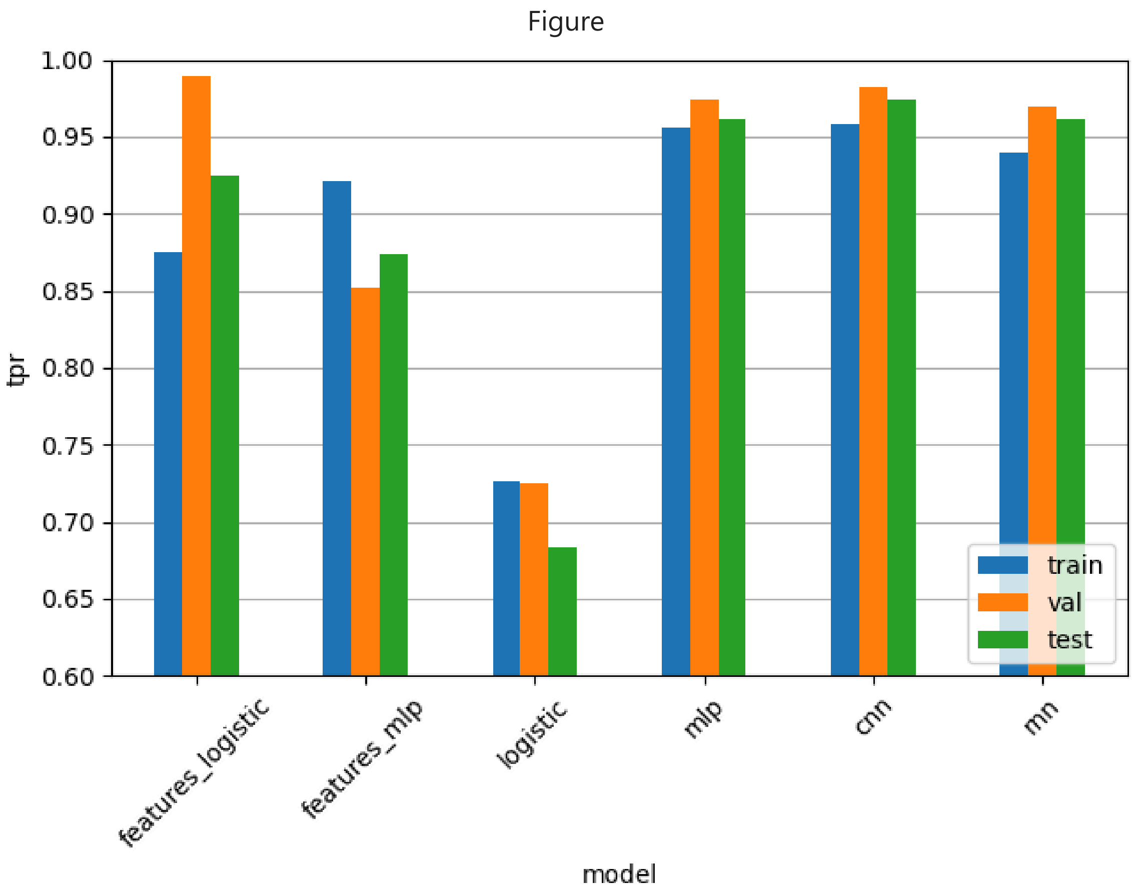


Figure
tpr
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
features_logistic
features_mlp
logistic
mlp
cnn
rnn
model
train
val
test

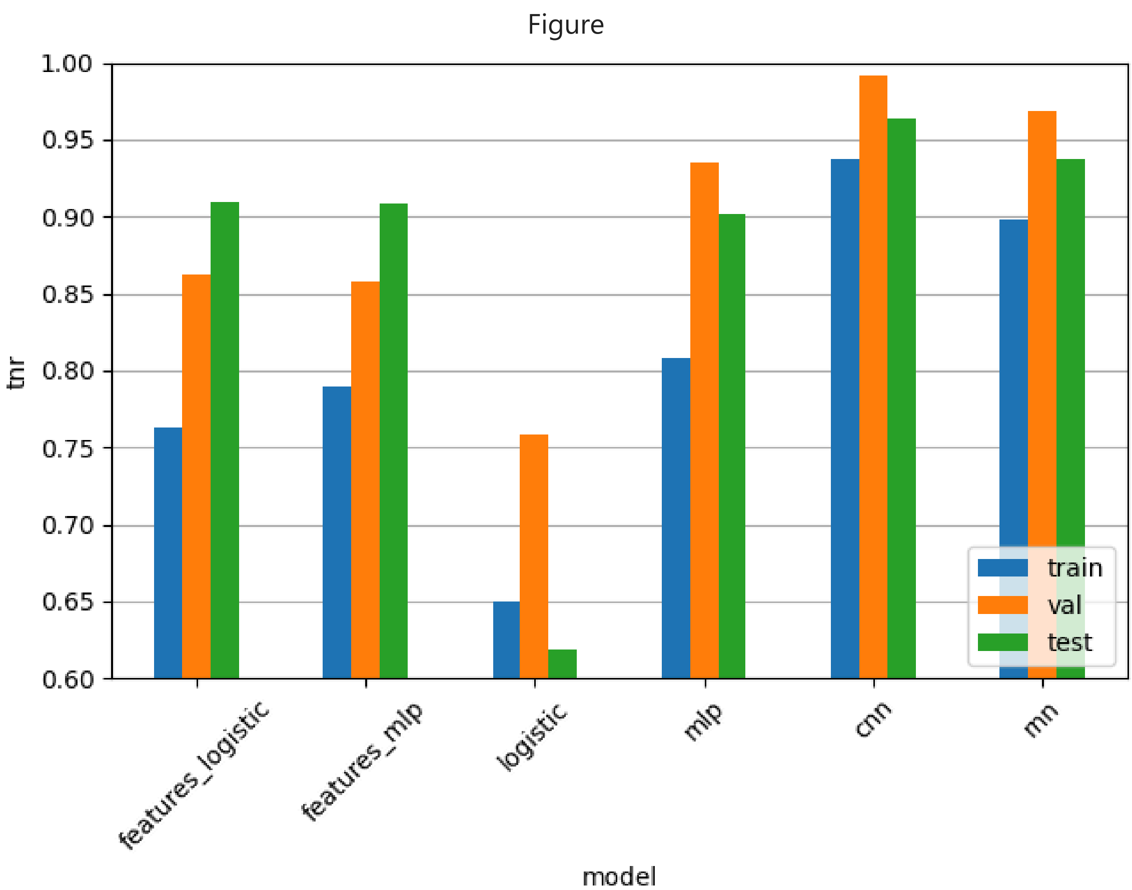


Figure
tnr
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
features_logistic
features_mlp
logistic
mlp
cnn
rnn
model
train
val
test

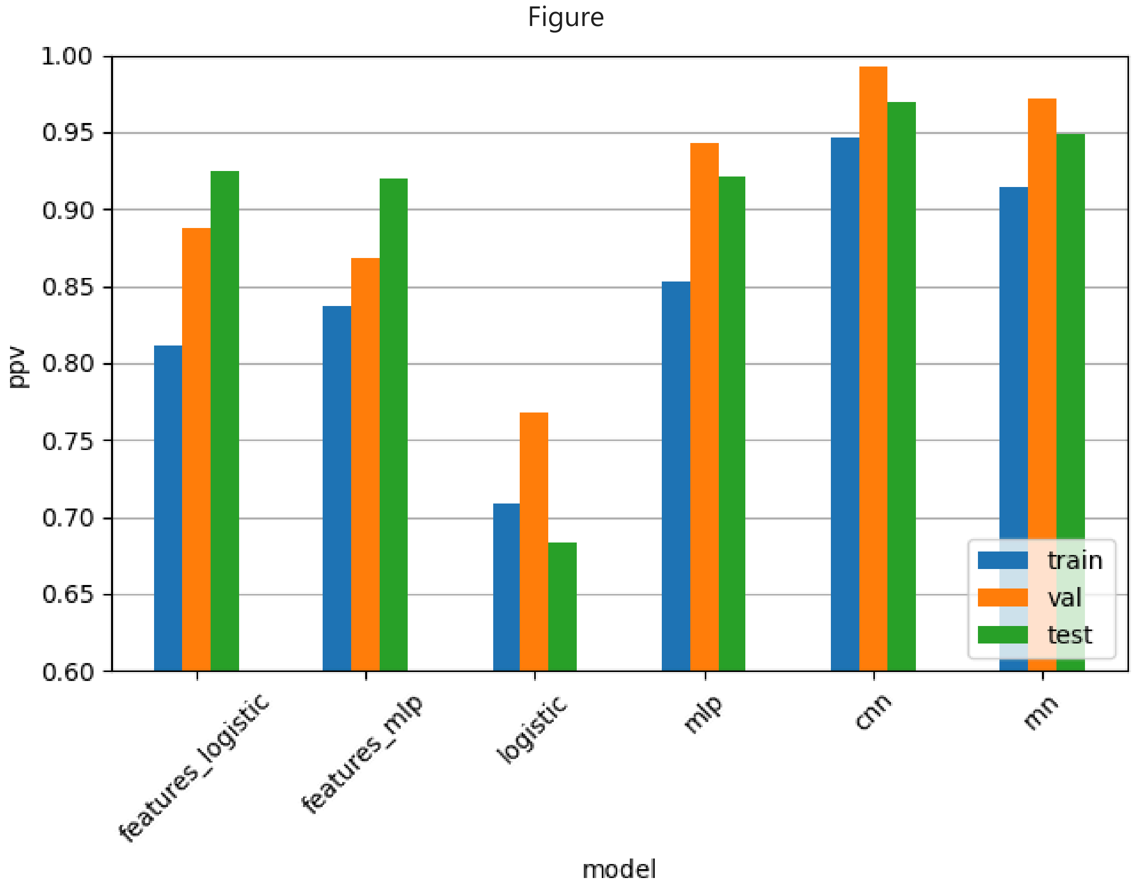


Figure
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
ppv
train
val
test
features_logistic
features_mlp
logistic
mlp
cnn
rnn
model

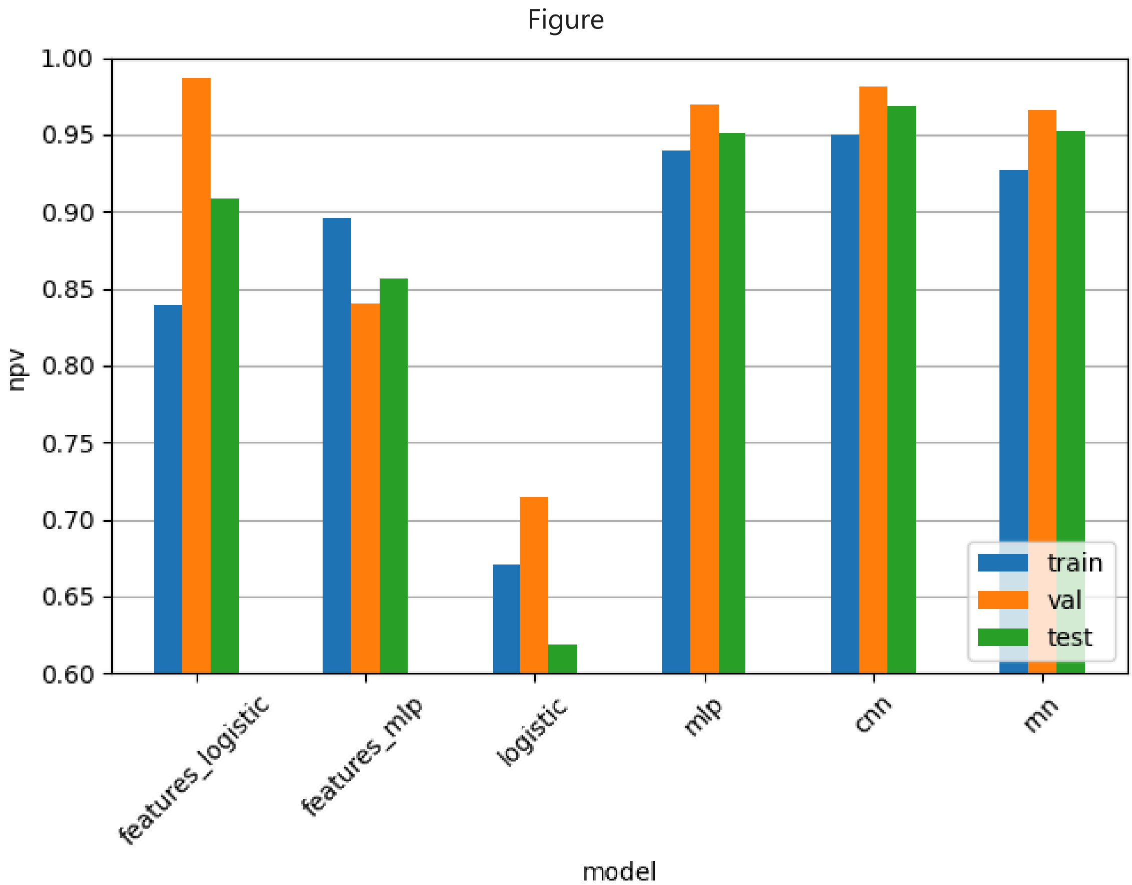
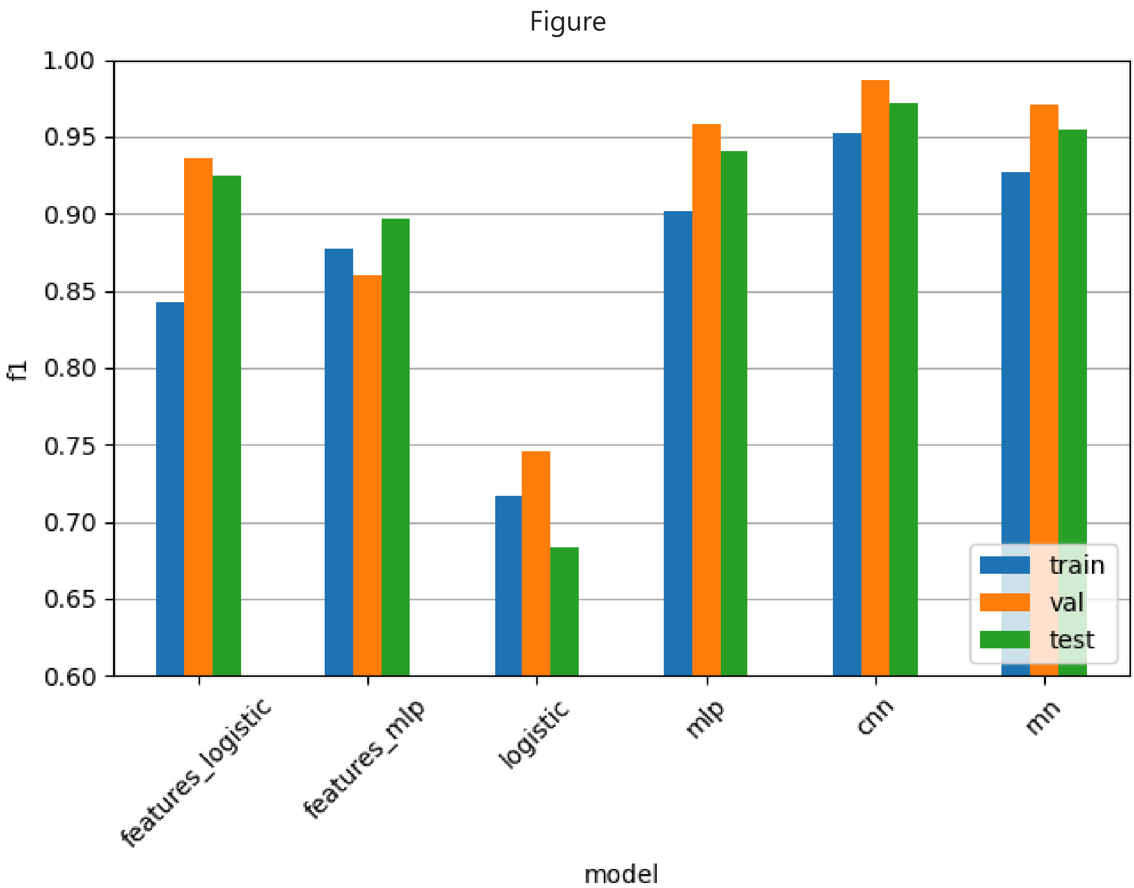


Figure
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
npv
train
val
test
features_logistic
features_mlp
logistic
mlp
cnn
rnn
model




**Question 4**

Based on the performance metrics shown above, answer the following questions:

1. Which are the best models?
2. What can you say about the models using features as inputs?
3. Bonus question: do you think it is *honest* to compare several models based on metrics computed on the test set in order to select the best one ?

**Answer:**

1. From the F1-score, the CNN is the best model. The RNN, MLP and `features_logistic` are all slightly worse, but still significantly better than `features_mlp` and `logistic`.
2. `features_logistic` is tremendously better than the `logistic` model. However, the MLP performs worse when using features as inputs. This may be because the MLP can learn better features from the raw data, thanks to its architecture.
3. It would be better to do so using metrics computed on the validation set. Indeed, the test set should only be used as an indicator of the model's performance when presented with data it has never seen before.

**Question 5**

Train and evaluate a model with a custom configuration. Can you obtain better performance than the previous models?

In [13]:
```
# Customize the model configuration below.
custom_config_name = 'custom'
custom_config = {
    'model': {
        'name': 'cnn',
        'config': {
            'input_size': intervals.shape[1],
            'output_size': targets.shape[1],
            'n_layers': 3,
            'n_initial_channels': 16,
            'kernel_size': 5,
        },
    },
    'optimizer': {
        'name': 'adam',
        'config': {'lr': 0.0001},
    },
}

custom_model = train_model(custom_config_name, custom_config, intervals, targets
custom_metrics = evaluate_model(custom_model, intervals, targets, i_train, i_val
IPython.display.display(custom_metrics)
```

```
   | Name             | Type               | Params | Mode  | In sizes     | Out siz
es
--------------------------------------------------------------------------------------
----------
0  | model            | CnnModel           | 13.1 K | train | [1, 32]      | [1, 1]
1  | model.layers     | Sequential         | 13.1 K | train | [1, 1, 32]   | [1, 1]
2  | model.layers.0   | Conv1d             | 96     | train | [1, 1, 32]   | [1, 16,
32]
3  | model.layers.1   | ReLU               | 0      | train | [1, 16, 32]  | [1, 16,
32]
4  | model.layers.2   | MaxPool1d          | 0      | train | [1, 16, 32]  | [1, 16,
16]
5  | model.layers.3   | Conv1d             | 2.6 K  | train | [1, 16, 16]  | [1, 32,
16]
6  | model.layers.4   | ReLU               | 0      | train | [1, 32, 16]  | [1, 32,
16]
7  | model.layers.5   | MaxPool1d          | 0      | train | [1, 32, 16]  | [1, 32,
8]
8  | model.layers.6   | Conv1d             | 10.3 K | train | [1, 32, 8]   | [1, 64,
8]
9  | model.layers.7   | ReLU               | 0      | train | [1, 64, 8]   | [1, 64,
8]
10 | model.layers.8   | AdaptiveAvgPool1d  | 0      | train | [1, 64, 8]   | [1, 64,
1]
11 | model.layers.9   | Flatten            | 0      | train | [1, 64, 1]   | [1, 64]
12 | model.layers.10  | Linear             | 65     | train | [1, 64]      | [1, 1]
--------------------------------------------------------------------------------------
----------
13.1 K    Trainable params
0         Non-trainable params
13.1 K    Total params
0.052     Total estimated model params size (MB)
13        Modules in train mode
0         Modules in eval mode
Sanity Checking: |          | 0/? [00:00<?, ?it/s]
Training: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

| | subset | count | acc | tpr | tnr | ppv | npv | f1 |
|---|---|---|---|---|---|---|---|---|
| **0** | train | 15000 | 0.896400 | 0.899641 | 0.892615 | 0.907276 | 0.883927 | 0.903442 |
| **1** | val | 4964 | 0.975020 | 0.963063 | 0.988161 | 0.988937 | 0.960543 | 0.975828 |
| **2** | test | 5100 | 0.950392 | 0.946576 | 0.954998 | 0.962099 | 0.936757 | 0.954274 |

Yes. We obtain better performance (accuracy) by adding an additional layer.

# Gait Classification

The goal of this exercise is to classify three types of gaits from windows of stride intervals. The stride intervals we will use are extracted from the Gait in Aging and Disease Database (https://physionet.org/content/gaitdb/1.0.0/).

This database includes stride intervals collected from 15 subjects: 5 healthy young adults, 5 healthy old adults, and 5 older adults with Parkinson's disease. Windows of 32 stride intervals (with 50% overlap) are already extracted from the raw data.

First, we import all required packages, define global constants, and seed the random number generators to obtain reproducible results.

In [1]:
```
%matplotlib widget


import itertools
import logging
import operator
import pathlib
import warnings

import IPython.display
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pytorch_lightning as pl
import seaborn as sns
import sklearn.metrics
import sklearn.model_selection
import sklearn.preprocessing
import torch


DATA_FILE = pathlib.Path('../data/gaitdb_intervals.npz')
LOG_DIRECTORY = pathlib.Path('../logs/gait_classification')


# Disable logging for PyTorch Lightning to avoid too long outputs.
logging.getLogger('pytorch_lightning').setLevel(logging.ERROR)

# Seed random number generators for reproducible results.
pl.seed_everything(42)
```

```
Seed set to 42
```

Out[1]: 42

Then, we load the windows of stride intervals and the corresponding labels.

In [2]:
```
def load_data():
    with np.load(DATA_FILE) as data:
        intervals = data['intervals']
        labels = data['labels']
```

```
        subjects = data['subjects']
    return intervals, labels, subjects


intervals, labels, subjects = load_data()

print(f'Number of windows : {intervals.shape[0]}')
print(f'Window size       : {intervals.shape[1]}')
print(f'Classes           : {np.unique(labels)}')
print(f'Subjects          : {np.unique(subjects)}')
```

```
Number of windows : 550
Window size       : 32
Classes           : ['old' 'parkinson' 'young']
Subjects          : ['o1' 'o2' 'o3' 'o4' 'o5' 'pd1' 'pd2' 'pd3' 'pd4' 'pd5' 'y1'
'y2' 'y3'
 'y4' 'y5']
```

We plot a few examples of stride interval windows for the different classes.

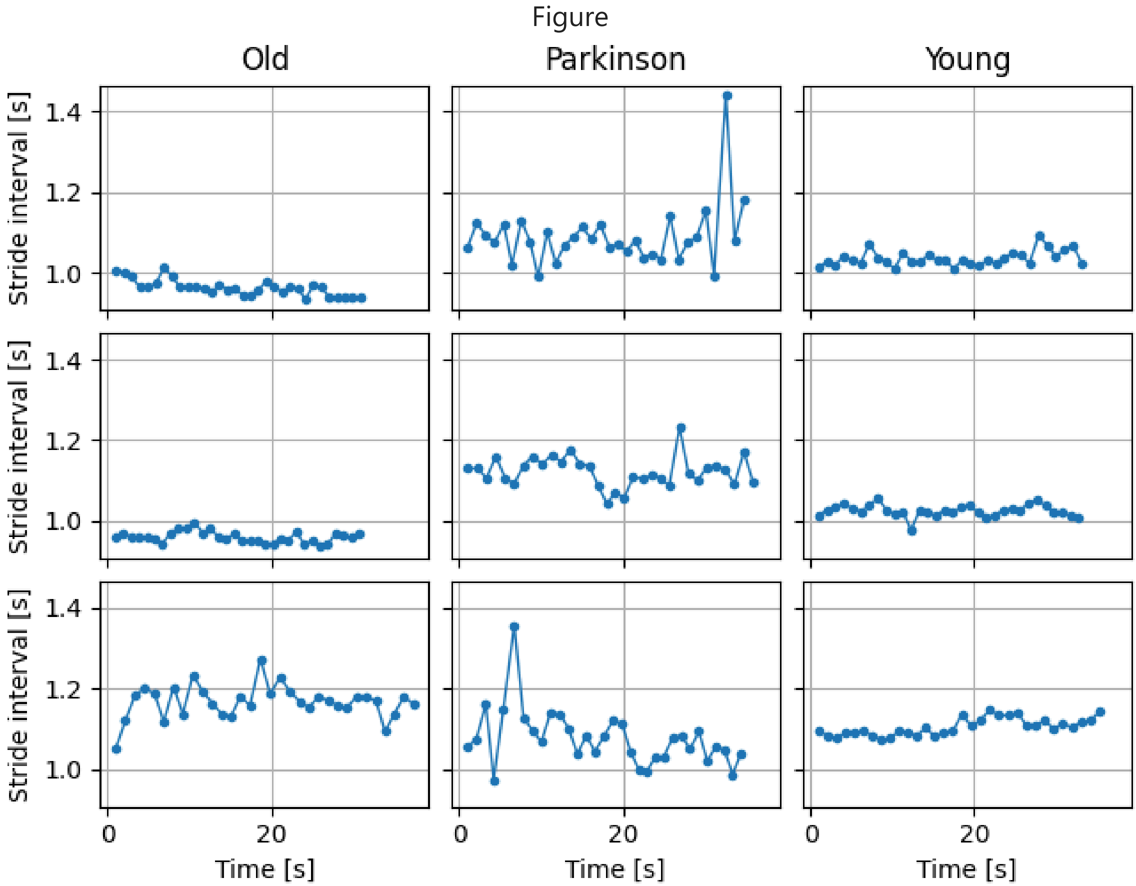
In [3]:
```
def plot_stride_examples(intervals, labels, n=3):
    classes = np.unique(labels)
    fig, axes = plt.subplots(n, len(classes), squeeze=False,
                             sharex='all', sharey='all', constrained_layout=True


    def plot_intervals(ax, x):
        t = np.cumsum(x)
        ax.plot(t, x, '.-', linewidth=1)
        ax.grid(True)

    for i, cls in enumerate(classes):
        indices = np.flatnonzero(labels == cls)
        indices = np.random.choice(indices, size=n)
        for j in range(n):
            if j == 0:
                axes[j, i].set_title(cls.capitalize())
            plot_intervals(axes[j, i], intervals[indices[j]])

    plt.setp(axes[:, 0], ylabel='Stride interval [s]')
    plt.setp(axes[-1, :], xlabel='Time [s]')


plot_stride_examples(intervals, labels)
```




**Question 1**

Are there any visible differences between the three classes in these examples?

**Answer:**

There is no visible difference between old and young subjects, but the Parkinson subjects have much more variance in their stride intervals.

Then, we split data into subsets for training (60%), validation (20%), and testing (20%) stratified by labels.

In [4]:
```
def split_data(labels):
    n = labels.size
    splitter = sklearn.model_selection.StratifiedKFold(n_splits=5, shuffle=True)
    indices = list(map(operator.itemgetter(1), splitter.split(np.zeros((n, 1)),
    i_train = np.hstack(indices[:-2])
    i_val = indices[-2]
    i_test = indices[-1]

    assert np.intersect1d(i_train, i_val).size == 0
    assert np.intersect1d(i_train, i_test).size == 0
    assert np.intersect1d(i_val, i_test).size == 0
    assert np.all(np.sort(np.hstack((i_train, i_val, i_test))) == np.arange(n))

    return i_train, i_val, i_test


i_train, i_val, i_test = split_data(labels)
```

```
def build_summary(labels, indices):
    classes = np.unique(labels)
    data = []
    for subset, i in indices:
        y = labels[i]
        data.append({'subset': subset, 'total_count': y.size})
        for cls in classes:
            data[-1][f'{cls}_count'] = np.sum(y == cls)
    return pd.DataFrame(data)


IPython.display.display(build_summary(labels, (('train', i_train), ('val', i_val
print(f'Subjects in training set   : {np.unique(subjects[i_train])}')
print(f'Subjects in validation set : {np.unique(subjects[i_val])}')
print(f'Subjects in test set       : {np.unique(subjects[i_test])}')
```

| | subset | total_count | old_count | parkinson_count | young_count |
|---|---|---|---|---|---|
| **0** | train | 330 | 150 | 42 | 138 |
| **1** | val | 110 | 50 | 13 | 47 |
| **2** | test | 110 | 49 | 14 | 47 |

```
Subjects in training set   : ['o1' 'o2' 'o3' 'o4' 'o5' 'pd1' 'pd2' 'pd3' 'pd4' 'p
d5' 'y1' 'y2' 'y3'
 'y4' 'y5']
Subjects in validation set : ['o1' 'o2' 'o3' 'o4' 'o5' 'pd1' 'pd2' 'pd3' 'pd4' 'p
d5' 'y1' 'y2' 'y3'
 'y4' 'y5']
Subjects in test set       : ['o1' 'o2' 'o3' 'o4' 'o5' 'pd1' 'pd2' 'pd3' 'pd4' 'p
d5' 'y1' 'y2' 'y3'
 'y4' 'y5']
```

### Question 2

Comment on the method used to split data into training, validation, and test sets. Is is appropriate? What would be another approach?

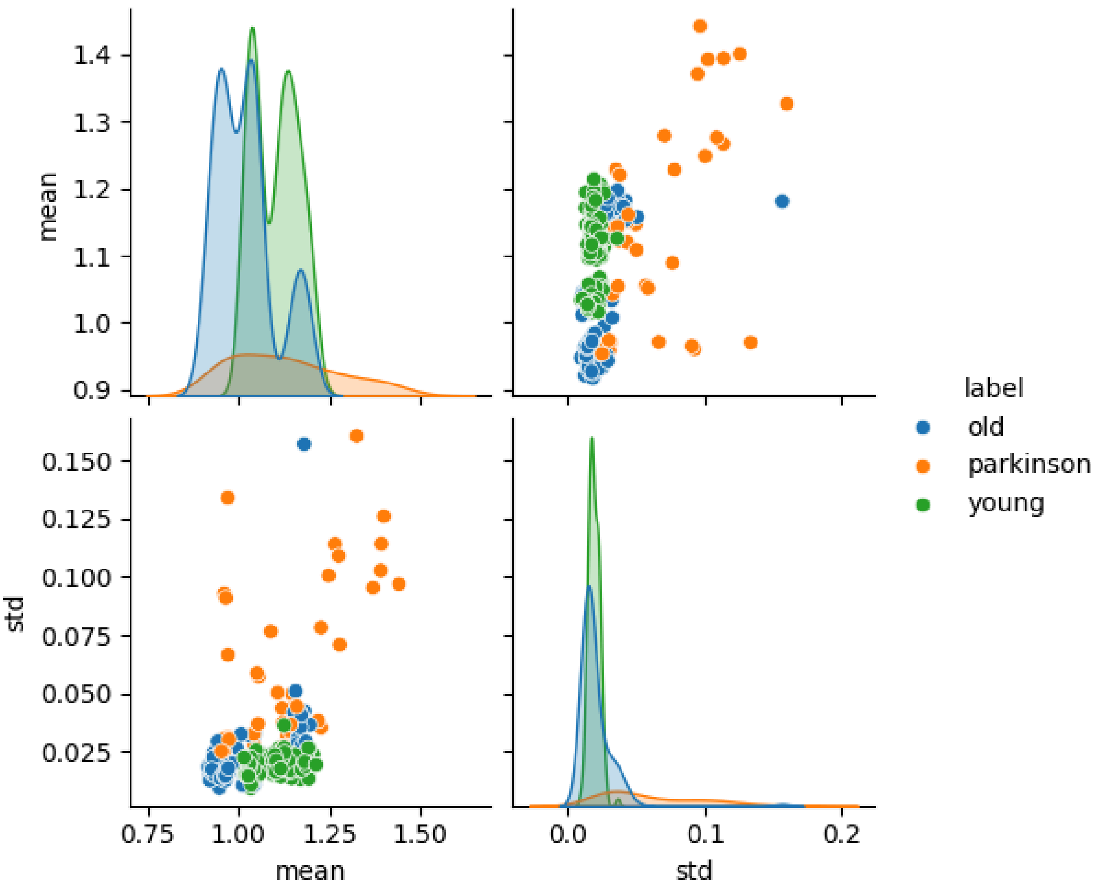
This method preserves the class distribution of the sample set. If this distribution is not uniform, then the classifier will be biased towards the most probable class. Indeed, there are much less Parkinson subjects than other classes. To avoid this, one could resample the dataset to obtain training, validation and test sets with uniform class distributions.

To better understand the data, we visualize the mean and standard deviation of each window of stride intervals in the training set.

In [5]:
```
def plot_stats(intervals, labels):
    data = pd.DataFrame({
        'label': labels,
        'mean': np.mean(intervals, axis=1),
        'std': np.std(intervals, axis=1),
    })
    sns.pairplot(data, hue='label')
```

```
plot_stats(intervals[i_train], labels[i_train])
```

Figure




To ensure the training procedure is stable, we center and scale the windows of stride intervals such that they have approximately zero mean and unit variance.

In [6]:
```
def scale_intervals(intervals, i_train):
    intervals = intervals - np.mean(intervals[i_train])
    intervals = intervals / np.std(intervals[i_train])
    return intervals


def print_stats(intervals, i_train, i_val, i_test):
    subsets = (
        ('Training set', i_train),
        ('Validation set', i_val),
        ('Test set', i_test),
    )
    for name, i in subsets:
        mu = np.mean(intervals[i])
        sigma = np.std(intervals[i])
        print(f'{name:14} : {mu:+.3f} ± {sigma:.3f}')


print('Before scaling')
print_stats(intervals, i_train, i_val, i_test)

scaled_intervals = scale_intervals(intervals, i_train)
```

```
print('\nAfter scaling')
print_stats(scaled_intervals, i_train, i_val, i_test)
```

```
Before scaling
Training set   : +1.067 ± 0.100
Validation set : +1.071 ± 0.100
Test set       : +1.074 ± 0.100

After scaling
Training set   : +0.000 ± 1.000
Validation set : +0.042 ± 0.991
Test set       : +0.074 ± 0.996
```

We also encode the labels with one-hot encoding.

In [7]:
```
def encode_labels(labels):
    categories = [np.unique(labels)]
    encoder = sklearn.preprocessing.OneHotEncoder(categories=categories, sparse_
    return encoder.fit_transform(labels[:, None])


encoded_labels = encode_labels(labels)


def print_encoded_labels(labels, encoded_labels, n=10):
    df = pd.DataFrame(encoded_labels, columns=np.unique(labels))
    df.insert(0, 'class', labels)
    df = df.iloc[np.random.permutation(len(df)), :]
    IPython.display.display(df.head(n))


print_encoded_labels(labels, encoded_labels)
```

| | class | old | parkinson | young |
|---|---|---|---|---|
| **423** | young | 0.0 | 0.0 | 1.0 |
| **548** | young | 0.0 | 0.0 | 1.0 |
| **437** | young | 0.0 | 0.0 | 1.0 |
| **324** | young | 0.0 | 0.0 | 1.0 |
| **122** | old | 1.0 | 0.0 | 0.0 |
| **446** | young | 0.0 | 0.0 | 1.0 |
| **128** | old | 1.0 | 0.0 | 0.0 |
| **520** | young | 0.0 | 0.0 | 1.0 |
| **15** | old | 1.0 | 0.0 | 0.0 |
| **170** | old | 1.0 | 0.0 | 0.0 |

We implement two models to classify the stride intervals: a multi-layer perceptron (MLP) and a convolutional neural network (CNN).

In [8]:
```
class MlpModel(torch.nn.Module):

    def __init__(self,
```

```
                 input_size,
                 output_size,
                 n_hidden_layers=1,
                 n_units=128,
                 dropout=0.0):
        super().__init__()
        self.input_size = input_size
        self.output_size = output_size
        self.n_hidden_layers = n_hidden_layers
        self.n_units = n_units
        self.dropout = dropout
        self.layers = self._build_layers()

    def _build_layers(self):
        sizes = [self.input_size]
        sizes.extend(itertools.repeat(self.n_units, self.n_hidden_layers))
        layers = []
        for i in range(self.n_hidden_layers):
            layers.append(torch.nn.Linear(sizes[i], sizes[i + 1]))
            layers.append(torch.nn.ReLU())
            if 0.0 < self.dropout < 1.0:
                layers.append(torch.nn.Dropout(self.dropout))
        layers.append(torch.nn.Linear(sizes[-1], self.output_size))
        return torch.nn.Sequential(*layers)

    def forward(self, x):
        return self.layers(x)


class CnnModel(torch.nn.Module):

    def __init__(self,
                 input_size,
                 output_size,
                 n_layers=3,
                 n_initial_channels=8,
                 kernel_size=3):
        super().__init__()
        self.input_size = input_size
        self.output_size = output_size
        self.n_layers = n_layers
        self.n_initial_channels = n_initial_channels
        self.kernel_size = kernel_size
        self.layers = self._build_layers()

    def _build_layers(self):
        layers = []

        n_output_channels = 1
        for i in range(self.n_layers):
            n_input_channels = n_output_channels
            n_output_channels = self.n_initial_channels * 2 ** i
            layers.append(torch.nn.Conv1d(
                in_channels=n_input_channels,
                out_channels=n_output_channels,
                kernel_size=self.kernel_size,
                padding='same',
            ))
            layers.append(torch.nn.ReLU())
            if i < self.n_layers - 1:
```

```
                layers.append(torch.nn.MaxPool1d(kernel_size=2))
            else:
                layers.append(torch.nn.AdaptiveAvgPool1d(1))
        layers.append(torch.nn.Flatten())

        layers.append(torch.nn.Linear(
            in_features=self.n_initial_channels * 2 ** (self.n_layers - 1),
            out_features=self.output_size,
        ))

        return torch.nn.Sequential(*layers)

    def forward(self, x):
        x = x.view(x.shape[0], 1, x.shape[1])
        return self.layers(x)
```

We also define a classifier class and utility functions to make it easier to train and evaluate models.

In [9]:
```
class Classifier(pl.LightningModule):

    def __init__(self, config):
        super().__init__()
        self.save_hyperparameters()
        self.config = config
        self.model = self._build_model()
        self.example_input_array = torch.zeros((1, self.model.input_size))

    def _build_model(self):
        name = self.config['model']['name']
        config = self.config['model'].get('config', {})
        if name == 'mlp':
            return MlpModel(**config)
        elif name == 'cnn':
            return CnnModel(**config)
        else:
            raise ValueError(f'unknown model: {name!r}')

    def configure_optimizers(self):
        name = self.config['optimizer']['name']
        config = self.config['optimizer'].get('config', {})
        if name == 'sgd':
            return torch.optim.SGD(self.parameters(), **config)
        elif name == 'adam':
            return torch.optim.Adam(self.parameters(), **config)
        else:
            raise ValueError(f'unknown optimizer: {name!r}')

    def forward(self, x):
        return self.model(x)

    def training_step(self, batch, batch_idx):
        return self._run_step(batch, 'train')

    def validation_step(self, batch, batch_idx):
        self._run_step(batch, 'val')

    def test_step(self, batch, batch_idx):
        self._run_step(batch, 'test')
```

```
    def predict_step(self, batch, batch_idx, dataloader_idx=0):
        x, y = batch
        return self.model(x)

    def _run_step(self, batch, subset):
        x, y = batch
        logits = self.model(x)
        loss = torch.nn.functional.cross_entropy(logits, y)
        acc = (torch.argmax(y, 1) == torch.argmax(logits, 1)).float().mean()
        self.log_dict({
            f'{subset}_loss': loss,
            f'{subset}_acc': acc,
        }, on_step=False, on_epoch=True, prog_bar=True)
        return loss


def build_loader(*tensors, batch_size=50, shuffle=False, n_workers=0):
    dataset = torch.utils.data.TensorDataset(*map(torch.Tensor, tensors))
    return torch.utils.data.DataLoader(
        dataset=dataset,
        batch_size=batch_size,
        shuffle=shuffle,
        num_workers=n_workers,
    )


def train_model(name, config, x, y, i_train, i_val, batch_size=50, n_epochs=10):
    train_loader = build_loader(x[i_train], y[i_train], batch_size=batch_size, s
    val_loader = build_loader(x[i_val], y[i_val], batch_size=batch_size)
    classifier = Classifier(config)
    print(pl.utilities.model_summary.ModelSummary(classifier, max_depth=-1))

    with warnings.catch_warnings():
        warnings.simplefilter('ignore')
        trainer = pl.Trainer(
            default_root_dir=LOG_DIRECTORY,
            logger=pl.loggers.TensorBoardLogger(LOG_DIRECTORY, name),
            enable_model_summary=False,
            max_epochs=n_epochs,
        )
        trainer.fit(classifier, train_loader, val_loader)

    return classifier


def evaluate_model(model, x, y, i_train, i_val, i_test, batch_size=50):
    loader = build_loader(x, y, batch_size=batch_size)

    with warnings.catch_warnings():
        warnings.simplefilter('ignore')
        trainer = pl.Trainer(
            default_root_dir=LOG_DIRECTORY,
            logger=False,
            enable_progress_bar=False,
            enable_model_summary=False,
        )
        z = trainer.predict(model, loader)
    z = np.vstack([u.numpy() for u in z])
```

```
    references = np.argmax(y, axis=1)
    predictions = np.argmax(z, axis=1)
    matrices = {}
    for subset, indices in (('train', i_train), ('val', i_val), ('test', i_test)
        matrices[subset] = sklearn.metrics.confusion_matrix(
            references[indices],
            predictions[indices],
        )

    return matrices
```

Finally, we define functions to plot confusion matrices.

In [10]:
```
def plot_confusion_matrix(c, labels=None, title=None):
    c = np.asarray(c)

    fig = plt.figure(figsize=(5, 4), constrained_layout=True)
    image = plt.imshow(c, cmap='Blues', interpolation='nearest')

    threshold = (c.min() + c.max()) / 2
    for i, j in itertools.product(range(c.shape[0]), repeat=2):
        if c[i, j] < threshold:
            color = image.cmap(image.cmap.N)
        else:
            color = image.cmap(0)
        text = format(c[i, j], '.2g')
        if c.dtype.kind != 'f':
            integer_text = format(c[i, j], 'd')
            if len(integer_text) < len(text):
                text = integer_text
        plt.text(j, i, text, color=color, ha='center', va='center')

    if labels is not None:
        plt.xticks(np.arange(c.shape[-1]), labels, rotation=45, ha='right')
        plt.yticks(np.arange(c.shape[-1]), labels)
    plt.xlabel('Predictions')
    plt.ylabel('References')
    if title is not None:
        plt.title(title)



def plot_confusion_matrices(matrices, labels):
    for subset in ('train', 'val', 'test'):
        c = matrices[subset]
        accuracy = np.trace(c) / c.sum()
        c = c / np.sum(c, axis=1, keepdims=True)
        title = f'{subset.capitalize()} set (accuracy = {accuracy:.3f})'
        plot_confusion_matrix(c, labels=labels, title=title)
```

We use TensorBoard to visualize performance metrics during training.

If you prefer to view TensorBoard in a separate window, you can open http://localhost:6006/ in your web browser.

In [11]:
```
%reload_ext tensorboard
%tensorboard --logdir ../logs/gait_classification --port 6007
```

# Index de C:\

| Nom | Taille | Date de modification |
|---|---|---|
| $Recycle.Bin/ | | 23/11/2023 20:06:09 |
| AppData/ | | 09/12/2023 12:20:38 |
| Config.Msi/ | | 22/11/2024 19:32:34 |
| DATA/ | | 23/11/2023 21:31:22 |
| Dokumente und Einstellungen/ | | 23/11/2023 17:25:38 |
| gurobi1003/ | | 24/05/2024 19:45:05 |
| hp/ | | 02/09/2023 15:07:32 |
| Kairos/ | | 09/12/2023 12:21:05 |
| OneDriveTemp/ | | 13/05/2024 09:21:25 |
| PerfLogs/ | | 07/05/2022 07:24:50 |
| Program Files/ | | 12/12/2024 09:39:35 |
| Program Files (x86)/ | | 03/10/2024 15:05:28 |
| ProgramData/ | | 22/11/2024 19:32:39 |
| Programme/ | | 23/11/2023 17:25:38 |
| Recovery/ | | 23/11/2023 19:49:01 |
| System Volume Information/ | | 09/12/2024 19:44:33 |
| System.sav/ | | 03/09/2023 02:01:15 |
| Users/ | | 23/11/2023 20:57:48 |
| Windows/ | | 13/11/2024 03:36:39 |
| DumpStack.log.tmp | 12.0 kB | 22/11/2024 19:32:37 |
| hiberfil.sys | 6.3 GB | 12/12/2024 09:38:15 |
| install.log | 1015 kB | 09/12/2023 12:21:15 |
| pagefile.sys | 31.0 GB | 09/12/2024 20:19:31 |
| Rapport-batterie.html | 83.5 kB | 02/12/2024 11:32:23 |
| swapfile.sys | 16.0 MB | 22/11/2024 19:32:37 |

We define a configuration for an MLP model with the following layers:

- Dense layer (output size = 128)
- ReLU activation
- Dense layer (output size = 128)
- ReLU activation
- Dense layer (output size = 3)

We then train the MLP model with stochastic gradient descent (SGD) and evaluate it.

In [12]:
```
batch_size = 10
n_epochs = 200

mlp_config = {
    'model': {
        'name': 'mlp',
        'config': {
            'input_size': scaled_intervals.shape[1],
            'output_size': encoded_labels.shape[1],
            'n_hidden_layers': 2,
            'n_units': 128,
        },
    },
    'optimizer': {
        'name': 'sgd',
        'config': {
            'lr': 0.0001,
        },
    },
}

mlp_model = train_model(
    name='mlp',
    config=mlp_config,
    x=scaled_intervals,
    y=encoded_labels,
    i_train=i_train,
    i_val=i_val,
    batch_size=batch_size,
    n_epochs=n_epochs,
)

mlp_matrices = evaluate_model(
    model=mlp_model,
    x=scaled_intervals,
    y=encoded_labels,
    i_train=i_train,
    i_val=i_val,
    i_test=i_test,
)

plot_confusion_matrices(mlp_matrices, np.unique(labels))
```

```
  | Name           | Type       | Params | Mode  | In sizes | Out sizes
----------------------------------------------------------------------------
0 | model          | MlpModel   | 21.1 K | train | [1, 32]  | [1, 3]
1 | model.layers   | Sequential | 21.1 K | train | [1, 32]  | [1, 3]
2 | model.layers.0 | Linear     | 4.2 K  | train | [1, 32]  | [1, 128]
3 | model.layers.1 | ReLU       | 0      | train | [1, 128] | [1, 128]
4 | model.layers.2 | Linear     | 16.5 K | train | [1, 128] | [1, 128]
5 | model.layers.3 | ReLU       | 0      | train | [1, 128] | [1, 128]
6 | model.layers.4 | Linear     | 387    | train | [1, 128] | [1, 3]
----------------------------------------------------------------------------
21.1 K    Trainable params
0         Non-trainable params
21.1 K    Total params
0.084     Total estimated model params size (MB)
7         Modules in train mode
0         Modules in eval mode
Sanity Checking: |          | 0/? [00:00<?, ?it/s]
```

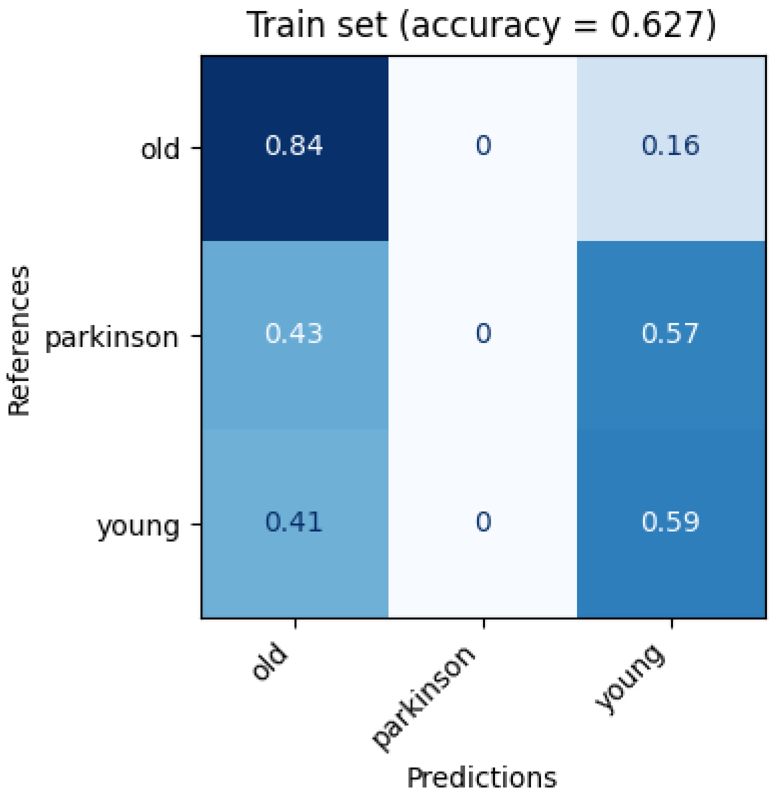
```
Training: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

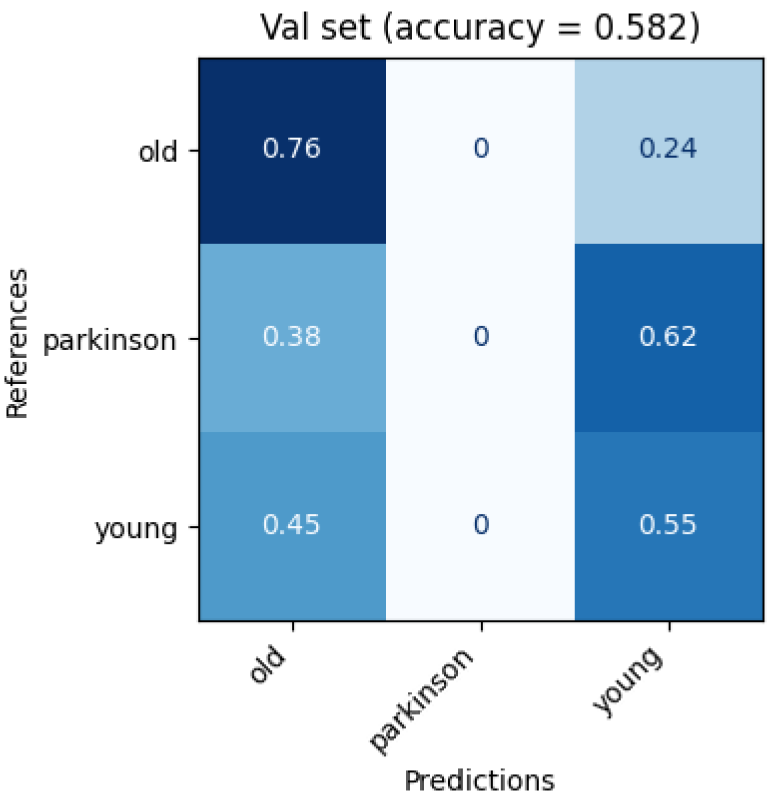
```
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

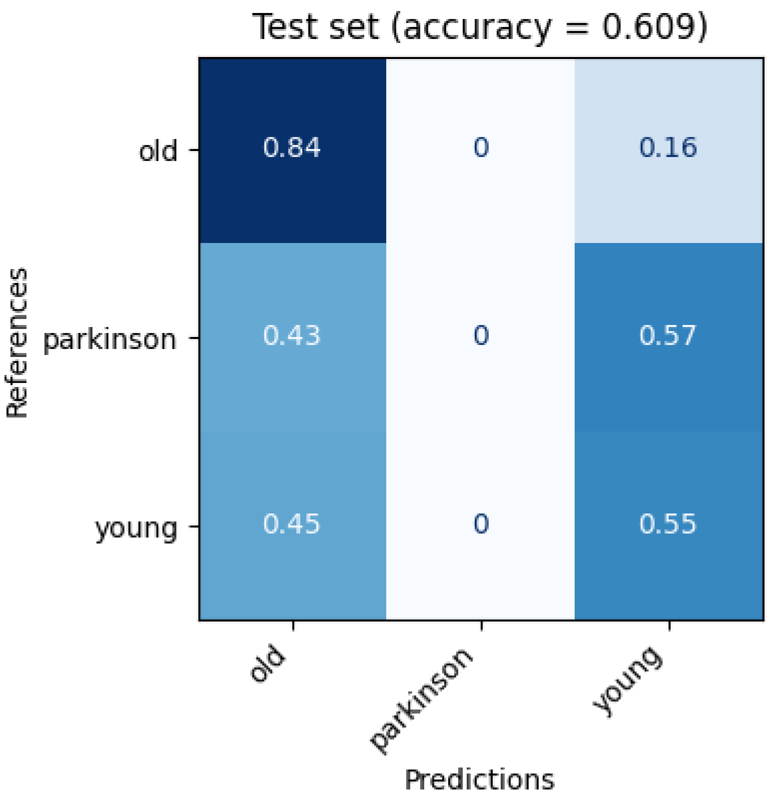
Figure



Figure




Figure




**Question 3**

Based on the metrics shown in TensorBoard, comment on the training procedure. Does the model overfit?

Based on the confusion matrices, what is the main issue of the model? What is a probable explanation for these results?

**Answer:**

The model seems to get stuck in a local minimum, as the training accuracy stops increasing past a number of epochs. A overfitting model would see its training accuracy increase while its validation accuracy stays constant.

Most importantly, the model never classifies any sample as `parkinson`. This could be due to the lower proportion of parkinson subjects in the dataset.

Next, we train and evaluate the same MLP model architecture except that this time we use SGD with momentum for optimizing the parameters.

In [13]:
```
mlp_momentum_config = {
    'model': {
        'name': 'mlp',
        'config': {
            'input_size': scaled_intervals.shape[1],
            'output_size': encoded_labels.shape[1],
            'n_hidden_layers': 2,
            'n_units': 128,
        },
    },
    'optimizer': {
        'name': 'sgd',
        'config': {
            'lr': 0.0001,
            'momentum': 0.9,
        },
    },
}

mlp_momentum_model = train_model(
    name='mlp_momentum',
    config=mlp_momentum_config,
    x=scaled_intervals,
    y=encoded_labels,
    i_train=i_train,
    i_val=i_val,
    batch_size=batch_size,
    n_epochs=n_epochs,
)

mlp_momentum_matrices = evaluate_model(
    model=mlp_momentum_model,
    x=scaled_intervals,
    y=encoded_labels,
    i_train=i_train,
    i_val=i_val,
    i_test=i_test,
)

plot_confusion_matrices(mlp_momentum_matrices, np.unique(labels))
```

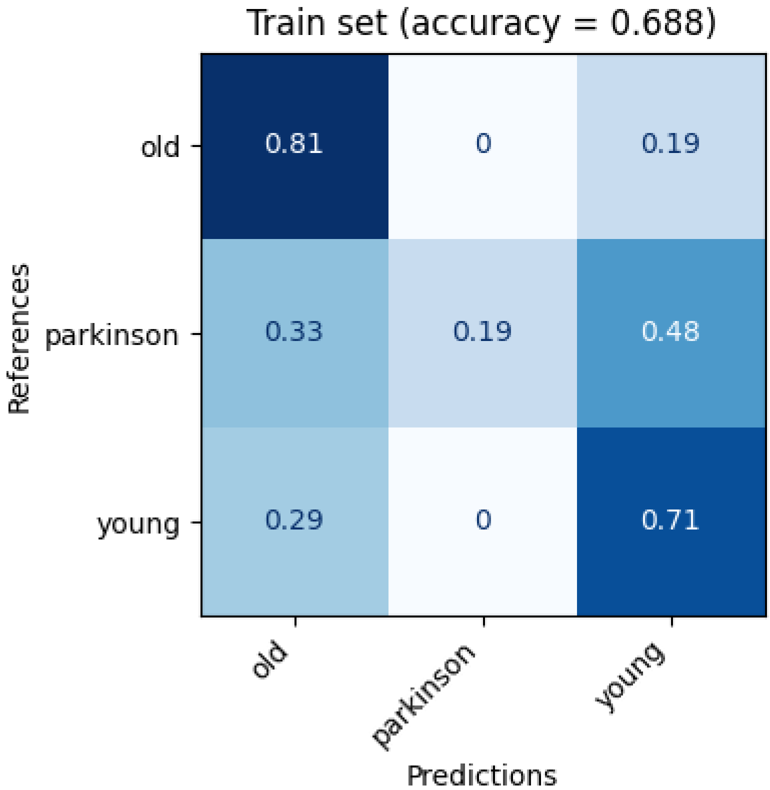
```
  | Name           | Type       | Params | Mode  | In sizes | Out sizes
-----------------------------------------------------------------------------
0 | model          | MlpModel   | 21.1 K | train | [1, 32]  | [1, 3]
1 | model.layers   | Sequential | 21.1 K | train | [1, 32]  | [1, 3]
2 | model.layers.0 | Linear     | 4.2 K  | train | [1, 32]  | [1, 128]
3 | model.layers.1 | ReLU       | 0      | train | [1, 128] | [1, 128]
4 | model.layers.2 | Linear     | 16.5 K | train | [1, 128] | [1, 128]
5 | model.layers.3 | ReLU       | 0      | train | [1, 128] | [1, 128]
6 | model.layers.4 | Linear     | 387    | train | [1, 128] | [1, 3]
-----------------------------------------------------------------------------
21.1 K    Trainable params
0         Non-trainable params
21.1 K    Total params
0.084     Total estimated model params size (MB)
7         Modules in train mode
0         Modules in eval mode
Sanity Checking: |          | 0/? [00:00<?, ?it/s]
Training: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

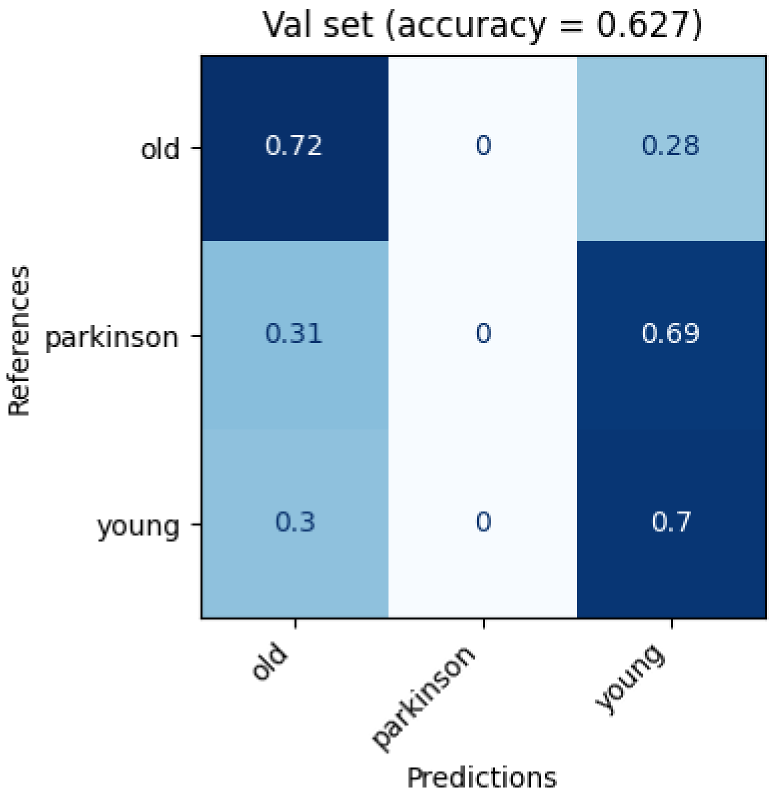
```
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

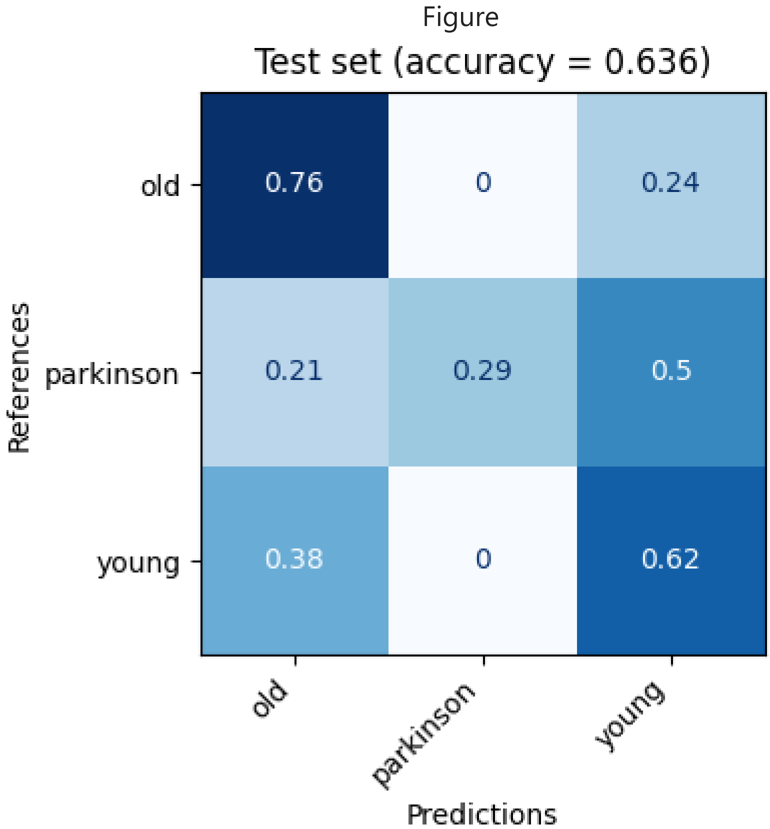
```
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

Figure




Figure






**Question 4**

What is the main difference of using momentum for training? Does the model overfit? Does momentum help to significantly improve the model accuracy?

**Answer:**

The momentum allows escaping local minima of the loss optimization landscape. This helps the model to learn to recognize parkinson subjects, eventually increasing the end accuracy. Furthermore, the convergence is faster.

Now, we use the same MLP architecture again, but we use the Adam optimizer for training instead of SGD.

In [14]:
```
mlp_adam_config = {
    'model': {
        'name': 'mlp',
        'config': {
            'input_size': scaled_intervals.shape[1],
            'output_size': encoded_labels.shape[1],
            'n_hidden_layers': 2,
            'n_units': 128,
        },
    },
    'optimizer': {
        'name': 'adam',
        'config': {
            'lr': 0.0001,
        },
    },
}
```

```python
mlp_adam_model = train_model(
    name='mlp_adam',
    config=mlp_adam_config,
    x=scaled_intervals,
    y=encoded_labels,
    i_train=i_train,
    i_val=i_val,
    batch_size=batch_size,
    n_epochs=n_epochs,
)

mlp_adam_matrices = evaluate_model(
    model=mlp_adam_model,
    x=scaled_intervals,
    y=encoded_labels,
    i_train=i_train,
    i_val=i_val,
    i_test=i_test,
)

plot_confusion_matrices(mlp_adam_matrices, np.unique(labels))
```

```
  | Name           | Type       | Params | Mode  | In sizes | Out sizes
----------------------------------------------------------------------------
0 | model          | MlpModel   | 21.1 K | train | [1, 32]  | [1, 3]
1 | model.layers   | Sequential | 21.1 K | train | [1, 32]  | [1, 3]
2 | model.layers.0 | Linear     | 4.2 K  | train | [1, 32]  | [1, 128]
3 | model.layers.1 | ReLU       | 0      | train | [1, 128] | [1, 128]
4 | model.layers.2 | Linear     | 16.5 K | train | [1, 128] | [1, 128]
5 | model.layers.3 | ReLU       | 0      | train | [1, 128] | [1, 128]
6 | model.layers.4 | Linear     | 387    | train | [1, 128] | [1, 3]
----------------------------------------------------------------------------
21.1 K    Trainable params
0         Non-trainable params
21.1 K    Total params
0.084     Total estimated model params size (MB)
7         Modules in train mode
0         Modules in eval mode
Sanity Checking: |          | 0/? [00:00<?, ?it/s]
Training: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

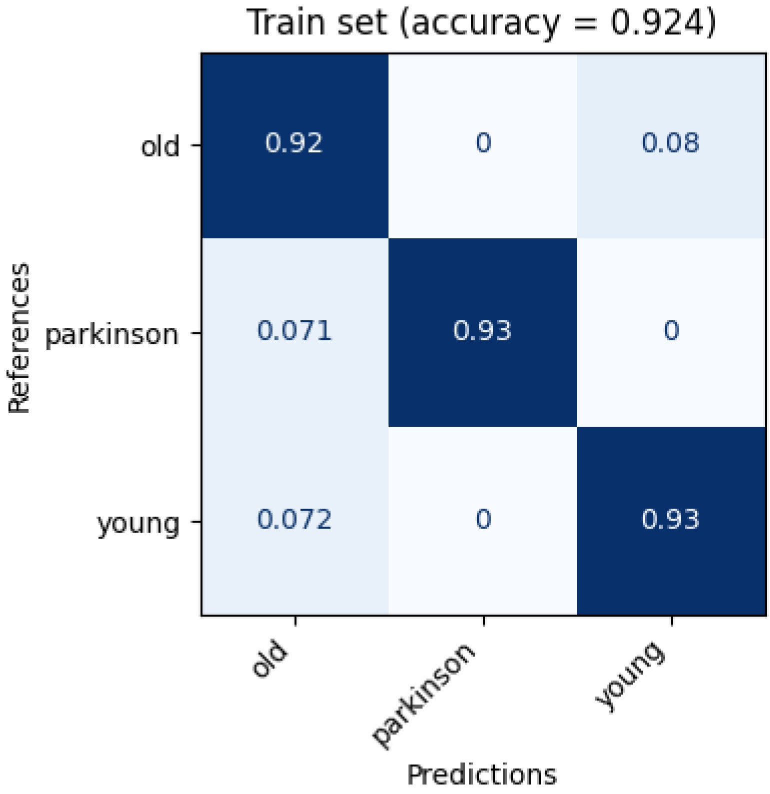
```
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

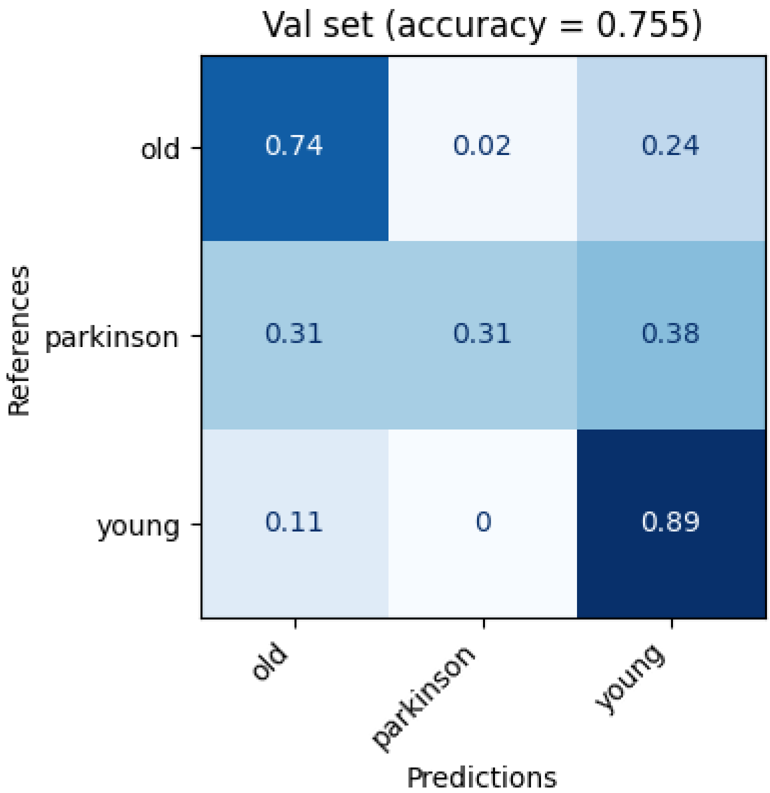
```
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

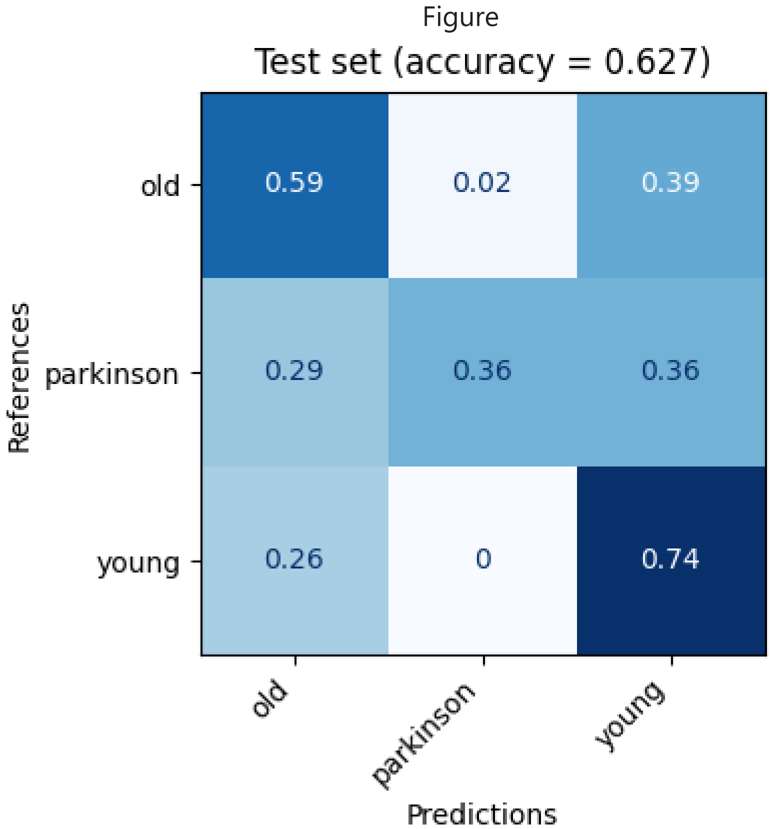
```
Validation: |          | 0/? [00:00<?, ?it/s]
```

Figure




Figure






**Question 5**

What are the main differences of the MLP model trained with the Adam optimized compared to the models trained with SGD (with and without momentum)? Does the model overfit? Does the Adam optimizer help to achieve better overall performance (as shown in the confusion matrices)?

**Answer:** Using the Adam optimizer, the model training converges faster and reaches higher accuracy. However, it overfits faster than the other algorithms. Regarding the classification, the model has a worse overall accuracy. It is better at identifying (true) parkinson and young subjects but performs worse on old subjects.

Finally, we train a CNN model with the Adam optimizer. This model includes the following layers:

- Convolutional layer (kernel size = 3, output size = 32, output channels = 8)
- ReLU activation
- Max pooling (output size = 16, output channels = 8)
- Convolutional layer (kernel size = 3, output size = 16, output channels = 16)
- ReLU activation
- Max pooling (output size = 8, output channels = 16)
- Convolutional layer (kernel size = 3, output size = 8, output channels = 32)
- ReLU activation
- Global average pooling (output size = 32)
- Dense layer (output size = 3)

In [15]:
```
cnn_config = {
    'model': {
        'name': 'cnn',
        'config': {
            'input_size': scaled_intervals.shape[1],
            'output_size': encoded_labels.shape[1],
            'n_layers': 3,
            'n_initial_channels': 8,
            'kernel_size': 3,
        },
    },
    'optimizer': {
        'name': 'adam',
        'config': {
            'lr': 0.0001,
        },
    },
}

cnn_model = train_model(
    name='cnn',
    config=cnn_config,
    x=scaled_intervals,
    y=encoded_labels,
    i_train=i_train,
    i_val=i_val,
    batch_size=batch_size,
    n_epochs=n_epochs,
)

cnn_matrices = evaluate_model(
    model=cnn_model,
    x=scaled_intervals,
    y=encoded_labels,
    i_train=i_train,
    i_val=i_val,
    i_test=i_test,
)

plot_confusion_matrices(cnn_matrices, np.unique(labels))
```

```
   | Name             | Type              | Params | Mode  | In sizes     | Out siz
es
-------------------------------------------------------------------------------------
----------
0  | model            | CnnModel          | 2.1 K  | train | [1, 32]      | [1, 3]
1  | model.layers     | Sequential        | 2.1 K  | train | [1, 1, 32]   | [1, 3]
2  | model.layers.0   | Conv1d            | 32     | train | [1, 1, 32]   | [1, 8,
32]
3  | model.layers.1   | ReLU              | 0      | train | [1, 8, 32]   | [1, 8,
32]
4  | model.layers.2   | MaxPool1d         | 0      | train | [1, 8, 32]   | [1, 8,
16]
5  | model.layers.3   | Conv1d            | 400    | train | [1, 8, 16]   | [1, 16,
16]
6  | model.layers.4   | ReLU              | 0      | train | [1, 16, 16] | [1, 16,
16]
7  | model.layers.5   | MaxPool1d         | 0      | train | [1, 16, 16] | [1, 16,
8]
8  | model.layers.6   | Conv1d            | 1.6 K  | train | [1, 16, 8]   | [1, 32,
8]
9  | model.layers.7   | ReLU              | 0      | train | [1, 32, 8]   | [1, 32,
8]
10 | model.layers.8   | AdaptiveAvgPool1d | 0      | train | [1, 32, 8]   | [1, 32,
1]
11 | model.layers.9   | Flatten           | 0      | train | [1, 32, 1]   | [1, 32]
12 | model.layers.10  | Linear            | 99     | train | [1, 32]      | [1, 3]
-------------------------------------------------------------------------------------
----------
2.1 K     Trainable params
0         Non-trainable params
2.1 K     Total params
0.008     Total estimated model params size (MB)
13        Modules in train mode
0         Modules in eval mode
Sanity Checking: |          | 0/? [00:00<?, ?it/s]
Training: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

```
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
Validation: |          | 0/? [00:00<?, ?it/s]
```

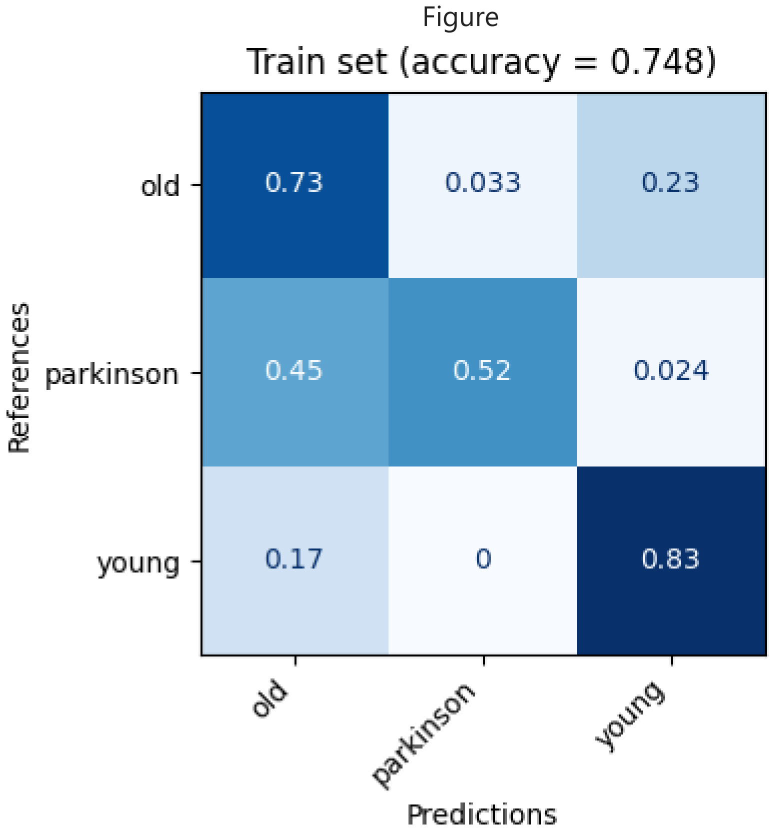


Figure
Train set (accuracy = 0.748)
old
parkinson
young
References
0.73
0.033
0.23
0.45
0.52
0.024
0.17
0
0.83
old
parkinson
young
Predictions

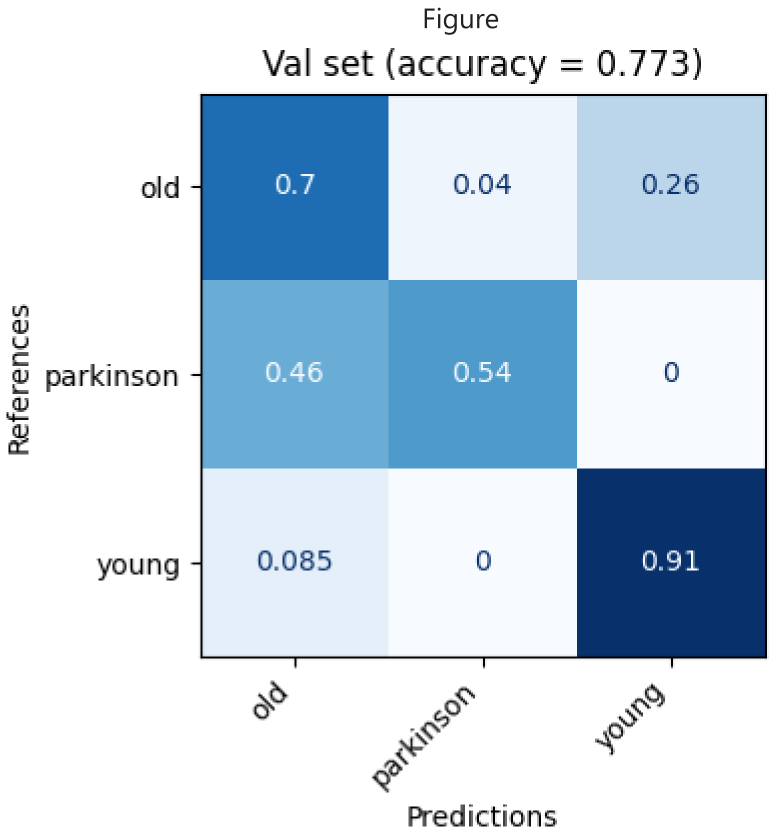
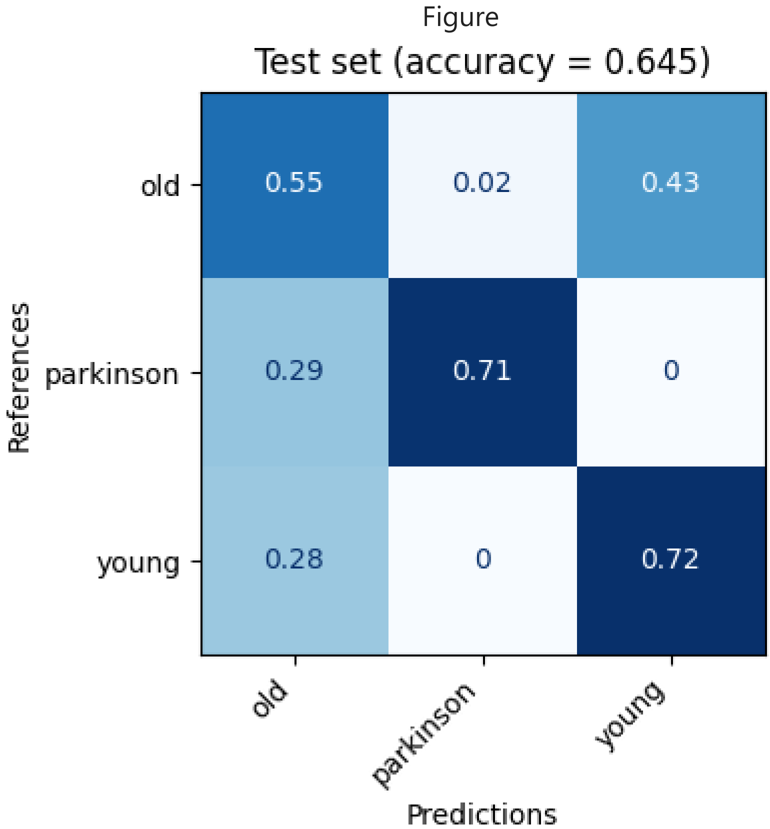


Figure
Val set (accuracy = 0.773)
old
parkinson
young
References
0.7
0.04
0.26
0.46
0.54
0
0.085
0
0.91
old
parkinson
young
Predictions




**Question 6**

What are the main differences of the CNN model with respect the MLP models trained with different optimizers? Does the CNN model overfit?

Can you think of a few reasons to explain why the MLP model trained with the Adam optimizer and the CNN model behave differently in terms of overfitting?

Overall, what is the main limitation of this dataset of stride intervals for training neural network models?

**Answer:** The CNN model has less parameters and is therefore less prone to overfitting.

As the dataset is limited in size, it is insufficient to train a large model that can generalize well.