Codes

April 18, 2024

Import libraries

[112]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F

from torch.utils.data import TensorDataset
import torch.optim as optim
from torch.optim.lr_scheduler import StepLR, ReduceLROnPlateau
import pandas as pd
import os
import numpy as np
import datetime
import scipy
import seaborn as sns
import sys
from collections import Counter
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

from sklearn.utils import shuffle
from imblearn.over_sampling import SMOTE
from imblearn.combine import SMOTETomek, SMOTEENN
from sklearn.utils import resample
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder,␣
 ↪OneHotEncoder, OrdinalEncoder

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import multilabel_confusion_matrix
from sklearn.metrics import f1_score, accuracy_score

from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

```python
import joblib
from sklearn.utils.class_weight import compute_class_weight
from skorch import NeuralNetClassifier
from skorch.callbacks import EarlyStopping, LRScheduler, Checkpoint
from skorch.helper import predefined_split
from skorch.dataset import Dataset
from skorch.callbacks import EpochScoring
from sklearn.metrics import RocCurveDisplay
from itertools import cycle
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import auc, roc_curve

%load_ext autotime
```

```
time: 16 ms (started: 2024-04-03 10:53:05 +01:00)
```

[ ]:
```python
# Set random seed
np.random.seed(42)
torch.manual_seed(42)
```

[ ]:
```python
if torch.cuda.is_available():
    device = torch.device("cuda")

else:
    device = torch.device("cpu")
print(device)
```

Functions and Tools

[ ]:
```python
def get_timestamp():
    """
        Get current timestamp
    """

    return datetime.datetime.now().strftime("%Y%m%dT%H%M%S")
```

[116]:
```python
def _import_data(path, validation_size=None):
    """
        Import source data
    """

    # Read source files
    df = pd.read_csv(f'source/mitbih_{path}.csv', header=None)

    # Extract data, and labels
    X = df.iloc[:, :-1].values
    y = df.iloc[:, -1].values.astype('int64')
```

```
    # Split into validation set, if needed
    if validation_size:
        X1, X2, y1, y2 = train_test_split(X, y, test_size=validation_size,␣
↪random_state=42)

        return X1, y1, X2, y2

    else:
        return X, y
```

time: 0 ns (started: 2024-04-03 10:53:05 +01:00)

[117]:
```
def _gaussian_noise(X_train):
    """
        Add noise to dataset
    """

    noise = np.random.normal(loc=0, scale=0.03, size=X_train.shape)

    return X_train + noise
```

time: 16 ms (started: 2024-04-03 10:53:06 +01:00)

[118]:
```
def _balancing(X, y, num_sample):
    """
        Balancing data with specific number of records
    """

    # Get records count
    label, count = np.unique(y, return_counts=True)

    X_balanced = []
    y_balanced = []

    for lbl, cnt in zip(label, count):
        X_filter = X[y==lbl]
        y_filter = y[y==lbl]

        # Downsampling if data exceeds desire number
        if cnt > num_sample:
            X_filter, y_filter = resample(X_filter, y_filter,
                                          replace=False,
                                          n_samples=num_sample,
                                          random_state=42)

        # Otherwise, upsampling with bootstrap
        elif cnt < num_sample:
            X_filter, y_filter = resample(X_filter, y_filter,
```

```
                                                replace=True,
                                                n_samples=num_sample,
                                                random_state=42)
            X_balanced.append(X_filter)
            y_balanced.append(y_filter)

        X_balanced = np.concatenate(X_balanced, axis=0)
        y_balanced = np.concatenate(y_balanced, axis=0)

        return X_balanced, y_balanced
```

time: 0 ns (started: 2024-04-03 10:53:06 +01:00)

[119]:
```
def _get_report(y_true, y_pred):
    """
        Generate classification report
    """

    report = classification_report(y_true, y_pred)

    print(report)
```

time: 0 ns (started: 2024-04-03 10:53:06 +01:00)

[2]:
```
def _roc_curve(y_true, y_pred):
    """
    Generate ROC curve
    Code for generating ROC curve obtained from documentation :
    https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
    """

    # Convert true labels to one-hot encoding
    y_true = LabelBinarizer().fit_transform(y_true)

    # Define the number of classes
    n_classes = 5
    class_labels = {0: 'N', 1: 'S', 2: 'V', 3: 'F', 4: 'Q'}

    # Define colors for each class
    colors = cycle(["aqua", "darkorange", "cornflowerblue", "olive", "maroon"])

    # Initialize dictionaries to store fpr, tpr, and roc_auc for each class
    fpr, tpr, roc_auc = dict(), dict(), dict()

    # Compute micro-average ROC
    fpr["micro"], tpr["micro"], _ = roc_curve(y_true.ravel(), y_pred.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
```

```python
# Compute macro-average ROC
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_true[:, i], y_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

fpr_grid = np.linspace(0.0, 1.0, 1000)

# Interpolate all ROC curves at these points
mean_tpr = np.zeros_like(fpr_grid)
for i in range(n_classes):
    mean_tpr += np.interp(fpr_grid, fpr[i], tpr[i])  # linear interpolation

# Average interpolated TPRs and compute macro AUC
mean_tpr /= n_classes

fpr["macro"] = fpr_grid
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

fig, ax = plt.subplots()

# Set the figure size
fig.set_size_inches(8, 6)

# Plot micro-average ROC curve
plt.plot(
    fpr["micro"],
    tpr["micro"],
    label=f"micro-average ROC curve (AUC = {roc_auc['micro']:.2f})",
    color="deeppink",
    linestyle=":",
    linewidth=4,
)

# Plot macro-average ROC curve
plt.plot(
    fpr["macro"],
    tpr["macro"],
    label=f"macro-average ROC curve (AUC = {roc_auc['macro']:.2f})",
    color="navy",
    linestyle=":",
    linewidth=4,
)

# Plot individual ROC curves for each class
for class_id, color in zip(range(n_classes), colors):
    RocCurveDisplay.from_predictions(
```

```
            y_true[:, class_id],
            y_pred[:, class_id],
            name=f"ROC curve for Class {class_labels[class_id]}",
            color=color,
            ax=ax,
        )

    # Set plot labels and title
    ax.set(
        xlabel="False Positive Rate",
        ylabel="True Positive Rate",
        title=" Receiver Operation Curve",
    )

    plt.legend()
    plt.show()
```

[ ]:
```
def _get_confusion_matrix(y_true, y_pred, title=None):
    """
        Generate confusion matrix
    """

    cm = confusion_matrix(y_true, y_pred)

    class_labels = ['N', 'S', 'V', 'F', 'Q']

    # Calculate counts for each class
    class_totals = cm.sum(axis=1)

    # Calculate percentage for each class
    cm_percent = (cm.T / class_totals).T * 100

    plt.figure(figsize=(6, 6))

    # Plot confusion matrix with heatmap
    sns.heatmap(cm_percent, annot=False, cmap="Blues", fmt='d',␣
↪xticklabels=class_labels, yticklabels=class_labels, cbar=False,␣
↪linewidths=1, linecolor='white')

    # Annotate with total predictions
    for i in range(len(class_labels)):
        for j in range(len(class_labels)):
            # Annotations for count
            plt.text(j + 0.5, i + 0.6, f'{cm[i, j]}', ha='center', va='center',␣
↪color='black', fontsize=8)
            # Annotations for percentage
```

```
            plt.text(j + 0.5, i + 0.4, f'{cm_percent[i, j]:.2f}%', ha='center',␣
↪va='center', color='black', fontsize=8)

    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.title(title)
    plt.grid(False)
    plt.show()
```

Models and Pipelines

CNN

[131]:
```
def _convert_to_tensor(X, y):
    """
        Convert data to tensor dataset
    """

    X = X.reshape(-1, 1, X.shape[-1])

    X = torch.from_numpy(X).float()
    y = torch.from_numpy(y).long()

    return X, y
```

```
time: 0 ns (started: 2024-04-03 10:53:30 +01:00)
```

[133]:
```
def _load_cnn_model(cnn, fn):
    """
        Load model for evaluation
    """

    # Load checkpoint
    checkpoint = torch.load(f"models/cnn/{fn}.pth", map_location=device)

    # Initialize model
    model = cnn()
    model_dict = model.state_dict()

    # Update model parameters with checkpoint values
    for key in checkpoint.keys():
        if key in model_dict:
            model_dict[key] = checkpoint[key]

    # Load updated parameters into the model
    model.load_state_dict(model_dict)

    # Move model to cuda (if available)
```

```
    model.to(device)

    # Set model to evaluation mode
    model.eval()

    return model
```

time: 0 ns (started: 2024-04-03 10:53:30 +01:00)

[134]:
```
def _evaluate_cnn(cnn, fn, subset='test', roc_curve=True):
    """
        Evaluate CNN model

        Params:
        ␣
    ↪-------------------------------------------------------------------------------
            cnn - Model Class (CNN or ResCNN)
            fn - Filename of model to be evaluated
            subset - Subset of source data to be evaluated (train, validation,␣
    ↪test)
            roc_curve - Whether to show ROC curve

    """

    # Load model
    model = _load_cnn_model(cnn, fn)


    if subset == 'test':
        # Load data
        X_test, y_test = _import_data('test')

        # Preprocess data
        X, y = _preprocess(X_test, y_test)

    else:
        # Load data
        X_train, y_train, X_val, y_val = _import_data('train',␣
    ↪validation_size=0.2)

        if subset == 'train':
            X, y = _preprocess(X_train, y_train)

        else:
            X, y = _preprocess(X_val, y_val)

    start = datetime.datetime.now()
```

```
    # Evaluate, and predict with probability
    with torch.no_grad():
        outputs = model(X)

    end = datetime.datetime.now()
    print(f"Predicting time: {end-start}")

    # Get predicted labels with highest probability
    _, y_pred = torch.max(outputs, 1)

    # Get classification report
    _get_report(y.cpu(), y_pred.cpu())

    # Generate confusion matric
    _get_confusion_matrix(y.cpu(), y_pred.cpu())

    if roc_curve:
        # Plot ROC curve
        _roc_curve(y.detach().cpu().numpy(), outputs.detach().cpu().numpy())
```

time: 0 ns (started: 2024-04-03 10:53:30 +01:00)

[136]:
```
def _initilise_cnn(model, **kwargs):
    """
        Initialise CNN model using skorch, NeuralNetClassifier
    """

    # Define batchsize, epoch, and loss function
    return NeuralNetClassifier(
                            model,
                            criterion=nn.CrossEntropyLoss,
                            device=device,
                            verbose=True,
                            max_epochs=100,
                            batch_size=128,
                            **kwargs
                        )
```

time: 0 ns (started: 2024-04-03 10:53:30 +01:00)

[137]:
```
def _callbacks(earlystop_patience=10, lr_scheduler=None, checkpoint=True):
    """
        Define callbacks for model training
    """

    # Earlystopping to prevent overfitting, by stop training when validation␣
 ↪loss does not improve more than threshold
```

```
    early_stop = EarlyStopping(monitor='valid_loss',␣
↪patience=earlystop_patience)

    # Model checkpoint to continuosly save best model, with the focus on best␣
↪validation loss
    model_path = f'models/cnn/{get_timestamp()}.pth'
    checkpoint = Checkpoint(
        f_params=model_path,
        monitor='valid_loss_best',
        f_optimizer=None,
        f_history=None,
        f_criterion=None
    )

    # Define callback to compute and log training accuracy
    train_acc = EpochScoring(scoring='accuracy', name='train_acc',␣
↪on_train=True)

    if checkpoint:
        return [early_stop, lr_scheduler, checkpoint, train_acc]

    else:
        return [early_stop, lr_scheduler, train_acc]
```

time: 0 ns (started: 2024-04-03 10:53:30 +01:00)

[1]:
```
def _gridsearchcv(X_train, y_train, model, param_grid, cv=5,␣
↪scoring='f1_macro'):
    """
        Perform paremeter tuning with stratify k-fold cross validation
    """

    skf = StratifiedKFold(n_splits=cv, shuffle=True, random_state=42)
    grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=skf,␣
↪scoring=scoring, n_jobs=-1, verbose=0)
    grid_result = grid.fit(X_train.cpu(), y_train.cpu())

    return grid_result
```

[141]:
```
def _plot_history(history):
    """
        Plot training and validation loss/accuracy over epochs
    """

    fig, axs = plt.subplots(1, 2, figsize=(10, 4))

    axs[0].plot(history[:, 'train_loss'], label='Training')
```

```
    axs[0].plot(history[:, 'valid_loss'], label='Validation')
    axs[0].set_xlabel('Epoch')
    axs[0].set_ylabel('Loss')
    axs[0].legend()

    axs[1].plot(history[:, 'train_acc'], label='Training')
    axs[1].plot(history[:, 'valid_acc'], label='Validation')
    axs[1].set_xlabel('Epoch')
    axs[1].set_ylabel('Accuracy')
    axs[1].legend()

    plt.tight_layout()
    plt.show()
```

time: 0 ns (started: 2024-04-03 10:53:30 +01:00)

[142]:
```
def _preprocess(X, y, balance=False, noise=False):
    """
    Preprocess data with optional balancing and augmentation,
    then convert to tensor dataset and move to CUDA (if available)
    """

    # Balance data if specified
    if balance:
        X, y = _balancing(X, y, balance)

    # Add noise for augmentation if specified
    if noise:
        X = _gaussian_noise(X)

    # Convert data to tensor dataset
    X, y = _convert_to_tensor(X, y)

    # Move data to CUDA device if available
    X, y = X.to(device), y.to(device)

    return X, y
```

time: 0 ns (started: 2024-04-03 10:53:30 +01:00)

[139]:
```
def _cnn_pipeline_with_gridsearch(cnn, param_grid, earlystop_patience=10,
                                  checkpoint=False, balance=False, noise=False,␣
 ↪**kwargs):
    """
        Encapsulated CNN pipeline for importing training data,
        preprocessing, hyperparameter tuning,
        and returning the best parameters

```

```
       Params:
      ␣
↪-------------------------------------------------------------------------------
           cnn - Model Class (CNN or ResCNN)
           param_grid - Dictionary of parameters to be selected through␣
↪gridsearch process
           earlystop_patience - Early stopping threshold
           checkpoint - Whether to enable checkpoint to continuously save best␣
↪model during training
           balance - Whether to balance dataset or not
           noise - Whether to add noise to dataset or not

    """

    # Import training data, and filter out validation set
    X_train, y_train, X_val, y_val = _import_data('train', validation_size=0.2)

    # Preproces train and validation set
    X_train, y_train = _preprocess(X_train, y_train, balance=balance,␣
↪noise=noise)

    # Define scheduler to adjust learning rate during training
    lr_scheduler = LRScheduler(policy='ReduceLROnPlateau', mode='min',␣
↪patience=5, factor=0.5, verbose=True)

    # Define callbacks for earlystopping and learning rate scheduler
    callbacks = _callbacks(earlystop_patience=earlystop_patience,␣
↪lr_scheduler=lr_scheduler)

    # Initialize CNN model
    model = _initilise_cnn(cnn, callbacks=callbacks, optimizer=optim.SGD,␣
↪optimizer__momentum=0.9,
                           optimizer__weight_decay=0.0001, lr=0.05, **kwargs)
    model.initialize()

    # Move to cuda (if available)
    model.module_.to(device)

    # Perform gridsearch cross-validation
    grid_result = _gridsearchcv(X_train, y_train, model, param_grid, cv=5)

    # Get best params and scores
    best_params = grid_result.best_params_
    best_score = grid_result.best_score_
    print("Best score: %f with %s" % (best_score, best_params))
```

```
    return best_params
```

time: 0 ns (started: 2024-04-03 10:53:30 +01:00)

[140]:
```
def _cnn_pipeline_with_best_param(cnn, params, earlystop_patience=10,
                                  checkpoint=True, class_weight=False,␣
 ↪balance=False,
                                  noise=False, fn=None, optimizer=None,␣
 ↪lr_scheduler=None):
    """
        Encapsulated CNN pipeline for training model with best parameters

        Params:
       ␣
 ↪------------------------------------------------------------------------------
            cnn - Model Class (CNN or ResCNN)
            params - Dictionary of best parameters obtained from parameter␣
 ↪selection process
            earlystop_patience - Early stopping threshold
            checkpoint - Whether to enable checkpoint to continuously save best␣
 ↪model during training
            balance - Whether to balance dataset or not
            noise - Whether to add noise to dataset or not
            fn - Customized filename of final model will be saved to
            optimizer - Whether to use default optimizer settings, or manually␣
 ↪passes
            lr_scheduler - Learning rate scheduler

    """

    # Import train and validation set
    X_train, y_train, X_val, y_val = _import_data('train', validation_size=0.2)

    # Compute class weight with class frequency to handle class imbalanced
    weights = torch.tensor(compute_class_weight('balanced',
                                        classes=np.unique(y_train),
                                        y=y_train.flatten()), dtype=torch.float)

    # Preprocess train and validation set
    X_train, y_train = _preprocess(X_train, y_train, balance=balance,␣
 ↪noise=noise)
    X_val, y_val = _preprocess(X_val, y_val, balance=balance, noise=noise)


    # Define scheduler to adjust learning rate during training
    if lr_scheduler is None:
```

```
        lr_scheduler = LRScheduler(policy='ReduceLROnPlateau', mode='min',␣
↪patience=5, factor=0.5, verbose=True)


    # Define callbacks
    callbacks = _callbacks(earlystop_patience=earlystop_patience,␣
↪lr_scheduler=lr_scheduler, checkpoint=checkpoint)

    # Initialize CNN model
    if optimizer is None:
        if class_weight:
            model = _initilise_cnn(cnn, callbacks=callbacks,␣
↪train_split=predefined_split(Dataset(X_val, y_val)),
                                   criterion__weight=weights, optimizer=optim.
↪SGD, optimizer__momentum=0.9, optimizer__weight_decay=0.0001, lr=0.
↪05,**params)

        else:
            model = _initilise_cnn(cnn, callbacks=callbacks,␣
↪train_split=predefined_split(Dataset(X_val, y_val)),
                                   optimizer=optim.SGD, optimizer__momentum= 0.
↪9, optimizer__weight_decay=0.0001, lr=0.05,**params)

    # To train model with replicate structure from reference paper
    else:
        model = _initilise_cnn(cnn, callbacks=callbacks,␣
↪criterion__weight=weights, train_split=predefined_split(Dataset(X_val,␣
↪y_val)), **params)

    model.initialize()

    # Move model to cuda if available
    model.module_.to(device)

    # Train model
    model.fit(X_train, y_train)

    # Get model prediction on train set
    y_pred = model.predict(X_train)

    # Generate classification report for train set
    _get_report(y_train.cpu().numpy(), y_pred)

    # Get model prediction on validation set
    y_pred = model.predict(X_val)
```

```python
    # Generate classification report for validation set
    _get_report(y_val.cpu().numpy(), y_pred)

    # Plot learning graph through epochs, with accuracy and loss of train and␣
↪validation set
    _plot_history(model.history)

    # Save final models
    if not fn:
        fn = get_timestamp()

    fp = f'models/cnn/{fn}.pth'
    torch.save(model.module_.state_dict(), fp)

    print(f"Best model saved to {fp}")

    return model
```

time: 0 ns (started: 2024-04-03 10:53:30 +01:00)

[ ]:
```python
class ConvBlock(nn.Module):
    """
        Convolutional block for a layer of convolution followed by batch␣
↪normalization, activation, and max pooling
    """

    def __init__(self, inputs, outputs, activation=nn.GELU, kernel_size=3,
                 padding='same', pool_kernel=3, pool_stride=2):

        super().__init__()

        # Define convolutional layer
        self.conv = nn.Conv1d(inputs, outputs, kernel_size=kernel_size,␣
↪padding=padding)

        # Batch normalization
        self.bn = nn.BatchNorm1d(outputs)

        # Activation function
        self.activation = activation()

        # Max pooling
        self.maxpool = nn.MaxPool1d(kernel_size=pool_kernel, stride=pool_stride)

    def forward(self, x):
```

```
        # Forward through convolution, batch normalization, activation, and max pooling
        x = self.activation(self.bn(self.conv(x)))
        x = self.maxpool(x)
        return x


class CNN(nn.Module):
    """
        Convolutional Neural Network model with multiple ConvBlocks followed by fully connected layers
    """

    def __init__(self, neurons=128, activation=nn.GELU, dropout=0.3):
        super().__init__()

        # Convolutional layers
        self.conv1 = ConvBlock(1, 32, activation=activation)
        self.conv2 = ConvBlock(32, 64, activation=activation)
        self.conv3 = ConvBlock(64, 128, activation=activation)
        self.conv4 = ConvBlock(128, 256, activation=activation)
        self.conv5 = ConvBlock(256, 512, activation=activation)

        # Adaptive max pooling
        self.pool = nn.AdaptiveMaxPool1d(1)

        # Activation function
        self.activation = activation()

        # Fully connected layer
        self.fc1 = nn.Linear(512, neurons)

        # Batch normalization
        self.bn = nn.BatchNorm1d(neurons)

        # Dropout layer
        self.dropout = nn.Dropout(dropout)

        # Output layer with 5 classes
        self.fc2 = nn.Linear(neurons, 5)

        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):

        # Forward through convolutional layers
        x = self.conv1(x)
```

```
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.conv5(x)

        # Adaptive max pooling
        x = self.pool(x)

        # Flatten before passing to fully connected layers
        x = torch.flatten(x, 1)

        # Fully connected layers
        x = self.activation(self.bn(self.fc1(x)))
        x = self.dropout(x)
        x = self.fc2(x)
        x = self.softmax(x)

        return x
```

[ ]:
```
class ResidualBlock(nn.Module):
    """
        Residual block for a layer of convolution followed by activation and␣
 ↪max pooling.
        This structure is aimed to replicate the models done by M. Kachuee et␣
 ↪al.
        Further details will be described in glossary.
    """

    def __init__(self, in_channels, out_channels, kernel_size=5, stride=1,␣
 ↪padding='same', activation=nn.ReLU):
        super().__init__()

        # Define convolutional layers
        self.conv1 = nn.Conv1d(in_channels, out_channels, kernel_size, stride,␣
 ↪padding)
        self.conv2 = nn.Conv1d(out_channels, out_channels, kernel_size, stride,␣
 ↪padding)

        # Activation function
        self.activation = activation()

        # Max pooling layer
        self.maxpool = nn.MaxPool1d(kernel_size=5, stride=2)

    def forward(self, x):

        # Store the residual
```

```
        residual = x

        # Forward through the first convolutional layer and activation
        out = self.conv1(x)
        out = self.activation(out)

        # Forward through the second convolutional layer
        out = self.conv2(out)

        # Add residual to output
        out += residual

        # Apply activation to output
        out = self.activation(out)

        # Apply max pooling
        out = self.maxpool(out)

        return out

class ResCNN(nn.Module):
    """
        Residual Convolutional Neural Network model with multiple␣
 ↪ResidualBlocks followed by fully connected layers
    """

    def __init__(self, activation=nn.ReLU):
        super().__init__()

        # Define the initial convolutional layer
        self.conv1 = nn.Conv1d(1, 32, kernel_size=5, stride=1)

        # Define the sequence of residual blocks
        self.res_blocks = nn.Sequential(
            ResidualBlock(32, 32, activation=activation),
            ResidualBlock(32, 32, activation=activation),
            ResidualBlock(32, 32, activation=activation),
            ResidualBlock(32, 32, activation=activation),
            ResidualBlock(32, 32, activation=activation)
        )

        # Flatten layer
        self.flatten = nn.Flatten()

        # Activation function
        self.activation = activation()
```

```
        # Fully connected layers
        self.fc1 = nn.Linear(64, 32)
        self.fc2 = nn.Linear(32, 32)
        self.fc3 = nn.Linear(32, 5)

    def forward(self, x):

        # Forward through the initial convolutional layer
        x = self.conv1(x)

        # Forward through the sequence of residual blocks
        x = self.res_blocks(x)

        # Flatten the output
        x = self.flatten(x)

        # Fully connected layers
        x = self.fc1(x)
        x = self.activation(x)
        x = self.fc2(x)

        # Softmax activation for multiclass classification
        x = F.softmax(self.fc3(x), dim=1)

        return x
```

SVM

[17]:
```
def _svm_pipeline(balanced_sample=None, dimredc=None,
                      n_components=None, n_folds=5, class_weight=None,
                      decision_function_shape='ovr', model_fn=None,
                      max_iter=-1):
    """
        Encapsulated SVM pipeline to import data, preproces,
        hyper paramater tuning, and training model

        Params:
        ␣
 ↪------------------------------------------------------------------------------
            balanced_sample - Number of records after balancing
            dimredc - Feature reduction type (PCA, LDA)
            n_components - Number of components will be retained after␣
 ↪transformation
            n_folds - Number of subsets that the dataset will be divided for␣
 ↪cross-validation
            class_weight - Whether to apply class weights or not
            decision_function_shape - Shape of the decision function (ovo, ovr)
            model_fn - Filename of final model
```

```
            max_iter - Maximum number of iteration

    """

    # Import train data
    X_train, y_train = _import_data('train')

    # Balance data if specific
    if balanced_sample:
        X_train, y_train = _balancing(X_train, y_train, balanced_sample)
    print(np.unique(y_train, return_counts=True))

    # Feature reduction if specific
    steps = []
    if dimredc == 'pca':
        steps.append(('pca', PCA(n_components=n_components)))
    elif dimredc == 'lda':
        steps.append(('lda',␣
↪LinearDiscriminantAnalysis(n_components=n_components)))


    # Initialize SVM model
    steps.append(('svm', SVC(decision_function_shape=decision_function_shape,
                             max_iter=max_iter,
                             verbose=1,
                             class_weight=class_weight)))
    pipeline = Pipeline(steps)

    # Perform stratified gridsearch cross validation
    grid_search = GridSearchCV(pipeline, params,␣
↪cv=StratifiedKFold(n_splits=n_folds), n_jobs=-1, scoring='f1_macro')
    grid_search.fit(X_train, y_train)

    # Get best parameters
    print("Best Parameters:", grid_search.best_params_)

    # Define final model with best paramaters
    best_model = grid_search.best_estimator_

    # Save best model for further evaluation
    if model_fn:
        model_fp = f'models/svm/{model_fn}.pkl'
    else:
        model_fp = f'models/svm/{get_timestamp()}.pkl'

    joblib.dump(best_model, model_fp)
    print(f"Model saved to {model_fp}")
```

```python
    # Predict on train set, with classification report
    y_pred = best_model.predict(X_train)
    _get_report(y_train, y_pred)
```

time: 0 ns (started: 2024-03-27 13:59:12 +00:00)

```python
[ ]: def _evaluate_svm(fn, subset='test', roc_curve=True):
    """
        Evaluate SVM model

        Params:
       ␣
  ↪------------------------------------------------------------------------------
            fn - Filename of model to be evaluated
            subset - Subset of source data to be evaluated (train, validation,␣
  ↪test)
            roc_curve - Whether to show ROC curve

    """

    if subset == 'test':
        # Load data
        X, y = _import_data('test')

    else:
        X, y = _import_data('train')

    # Load best model
    model = joblib.load(f"models/svm/{fn}.pkl")

    start = datetime.datetime.now()

    # Evaluate, and predict
    y_pred = model.predict(X)

    end = datetime.datetime.now()
    print(f"Predicting time: {end-start}")

    # Get classification report
    _get_report(y, y_pred)

    # Generate confusion matric
    _get_confusion_matrix(y, y_pred)

    if roc_curve:
```

```
        # Evaluate, and predict with probability
        y_prob_test = model.predict_proba(X)

        # Plot ROC curve
        _roc_curve(y, y_prob_test)
```

1 Set up

[ ]:
```
%run "tools.ipynb"
```

```
cuda
time: 109 ms (started: 2024-04-16 12:11:18 +01:00)
```

2 EDA

[ ]:
```
train_df = pd.read_csv('source/mitbih_train.csv', header=None)
test_df = pd.read_csv('source/mitbih_test.csv', header=None)

if (train_df.isnull().sum().sum()) == 0 :
    print('No missing value in training set!')

if (test_df.isnull().sum().sum()) == 0 :
    print('No missing value in test set!')
```

```
No missing value in training set!
No missing value in test set!
time: 6.3 s (started: 2024-04-12 20:31:13 +01:00)
```

[ ]:
```
train_df.describe()
```

	0	1	2	3	4 \
count	87554.000000	87554.000000	87554.000000	87554.000000	87554.000000
mean	0.890360	0.758160	0.423972	0.219104	0.201127
std	0.240909	0.221813	0.227305	0.206878	0.177058
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.921922	0.682486	0.250969	0.048458	0.082329
50%	0.991342	0.826013	0.429472	0.166000	0.147878
75%	1.000000	0.910506	0.578767	0.341727	0.258993
max	1.000000	1.000000	1.000000	1.000000	1.000000

	5	6	7	8	9 \
count	87554.000000	87554.000000	87554.000000	87554.000000	87554.000000
mean	0.210399	0.205808	0.201773	0.198691	0.196757
std	0.171909	0.178481	0.177240	0.171778	0.168357
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.088416	0.073333	0.066116	0.065000	0.068639
50%	0.158798	0.145324	0.144424	0.150000	0.148734

```
75%        0.287628      0.298237      0.295391      0.290832      0.283636
max        1.000000      1.000000      1.000000      1.000000      1.000000

       …           178           179           180           181  \
count  …  87554.000000  87554.000000  87554.000000  87554.000000
mean   …      0.005025      0.004628      0.004291      0.003945
std    …      0.044154      0.042089      0.040525      0.038651
min    …      0.000000      0.000000      0.000000      0.000000
25%    …      0.000000      0.000000      0.000000      0.000000
50%    …      0.000000      0.000000      0.000000      0.000000
75%    …      0.000000      0.000000      0.000000      0.000000
max    …      1.000000      1.000000      1.000000      1.000000

                182           183           184           185           186  \
count  87554.000000  87554.000000  87554.000000  87554.000000  87554.000000
mean       0.003681      0.003471      0.003221      0.002945      0.002807
std        0.037193      0.036255      0.034789      0.032865      0.031924
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%        0.000000      0.000000      0.000000      0.000000      0.000000
50%        0.000000      0.000000      0.000000      0.000000      0.000000
75%        0.000000      0.000000      0.000000      0.000000      0.000000
max        1.000000      1.000000      1.000000      1.000000      1.000000

                187
count  87554.000000
mean       0.473376
std        1.143184
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        4.000000

[8 rows x 188 columns]

time: 1.16 s (started: 2024-04-12 20:31:19 +01:00)
```

[ ]:
```
X_train, y_train = _import_data('train')
X_test, y_test = _import_data('test')
```

```
time: 5.89 s (started: 2024-04-16 12:11:18 +01:00)
```

[ ]:
```
y_train.shape
```

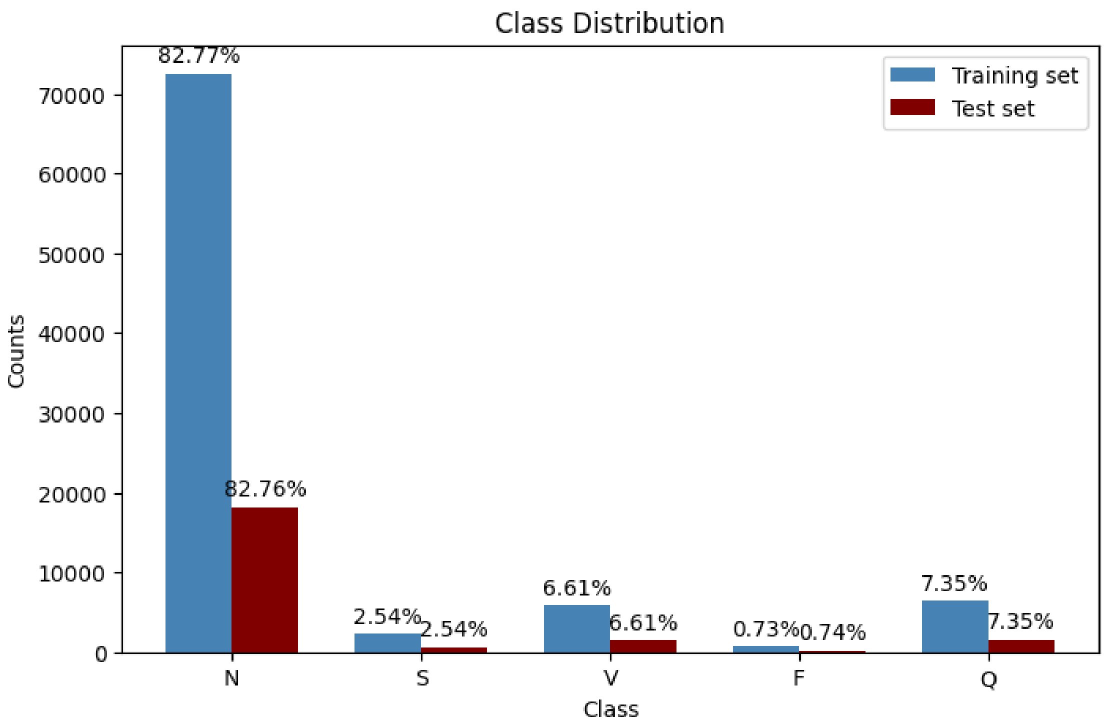
```
(87554,)

time: 16 ms (started: 2024-04-16 12:11:24 +01:00)
```

[ ]:
```
y_test.shape
```

```
(21892,)

time: 15 ms (started: 2024-04-16 12:11:29 +01:00)
```

[ ]:

```
# Create subplot
fig, ax = plt.subplots(figsize=(8, 5))

# Get class counts for training and test sets
classes, train_counts = np.unique(y_train, return_counts=True)
test_counts = [np.sum(y_test == cls) for cls in classes]

# Calculate total counts for each class
total_counts = train_counts + test_counts

# Set the width of the bars
bar_width = 0.35

# Plot the training set counts
train_bars = ax.bar(classes - bar_width/2, train_counts, width=bar_width,␣
 ↪color='steelblue', label='Training set')

# Plot the test set counts
test_bars = ax.bar(classes + bar_width/2, test_counts, width=bar_width,␣
 ↪color='maroon', label='Test set')

# Set labels and title
ax.set_xlabel('Class')
ax.set_ylabel('Counts')
ax.set_title('Class Distribution')
label_mapping = {'N': 0, 'S': 1, 'V': 2, 'F': 3, 'Q': 4}
ax.set_xticks(list(label_mapping.values()), list(label_mapping.keys()))
ax.legend()

# Annotate the bars with percentage
for bars, counts in zip([train_bars, test_bars], [train_counts, test_counts]):
    for bar, count in zip(bars, counts):
        height = bar.get_height()
        percentage = (count / len(y_train)) * 100 if bars == train_bars else␣
 ↪(count / len(y_test)) * 100
        ax.annotate(f'{percentage:.2f}%', xy=(bar.get_x() + bar.get_width() /␣
 ↪2, height), xytext=(0, 3),
                    textcoords="offset points", ha='center', va='bottom')

plt.show()
```

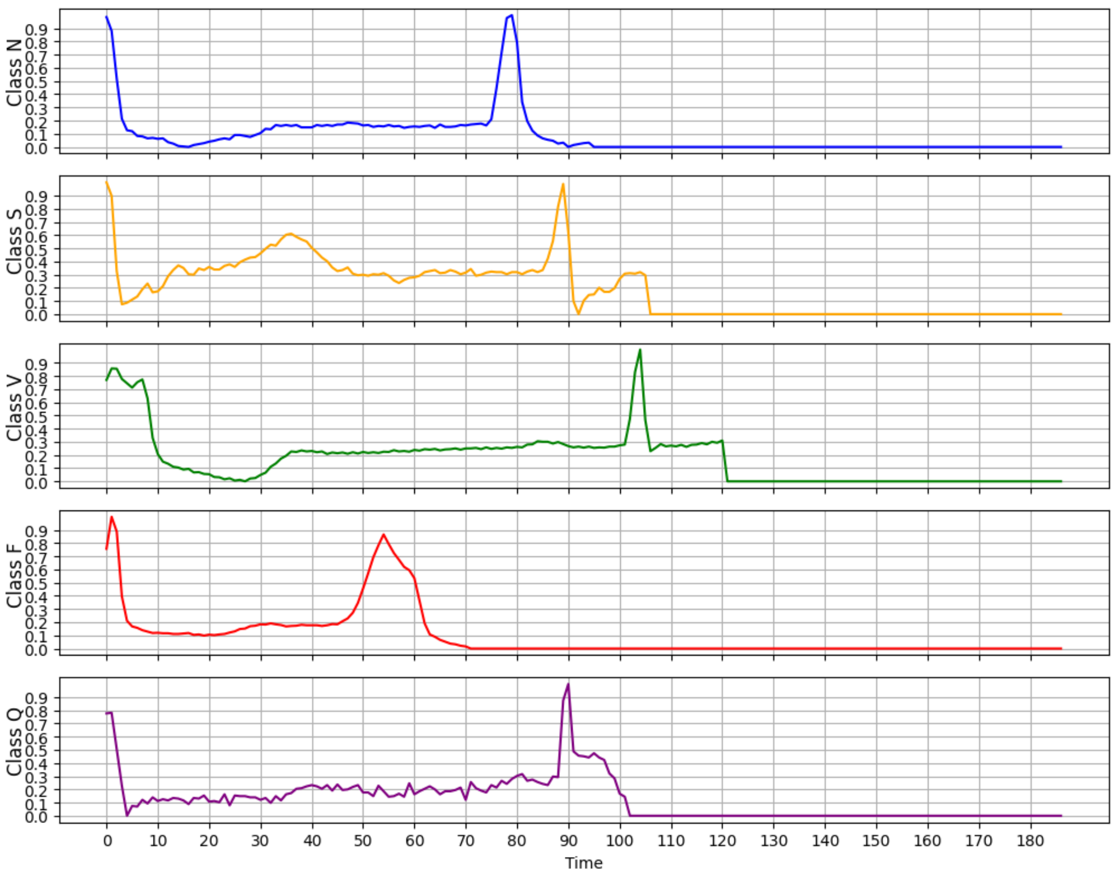



```
time: 312 ms (started: 2024-04-15 22:52:44 +01:00)
```

Dataset contains highly imbalanced class, with over 82% belong to majority class N, and less than 10% of other classes each. Also, train and test set have the same class distribution.

```
[ ]: # Create subplots
fig, axs = plt.subplots(5, 1, figsize=(10, 8), sharex=True)
axs = axs.flatten()

# Define colors for each class
label_colors = ['blue', 'orange', 'green', 'red', 'purple']

# Define label mapping
label_mapping = {0: "N", 1: "S", 2: "V", 3: "F", 4: "Q"}

# Store 1 sample per class
samples_per_class = []
for label in np.unique(y_train):
    X_lbl = X_train[y_train == label][10:]
    samples_per_class.append(X_lbl)

# Plot ECG signal for each class
for i, ax in enumerate(axs):
    sample = samples_per_class[i][0]
```

```
    ax.plot(sample, color=label_colors[i])
    ax.set_title(f"Class {label_mapping[i]}", rotation='vertical', x=-0.04, y=0.
↪3)
    ax.set_xticks(np.arange(0, len(sample), 10))
    ax.set_yticks(np.arange(0, 1, 0.1))
    ax.grid(True)

plt.xlabel('Time')
plt.tight_layout()
plt.show()
```

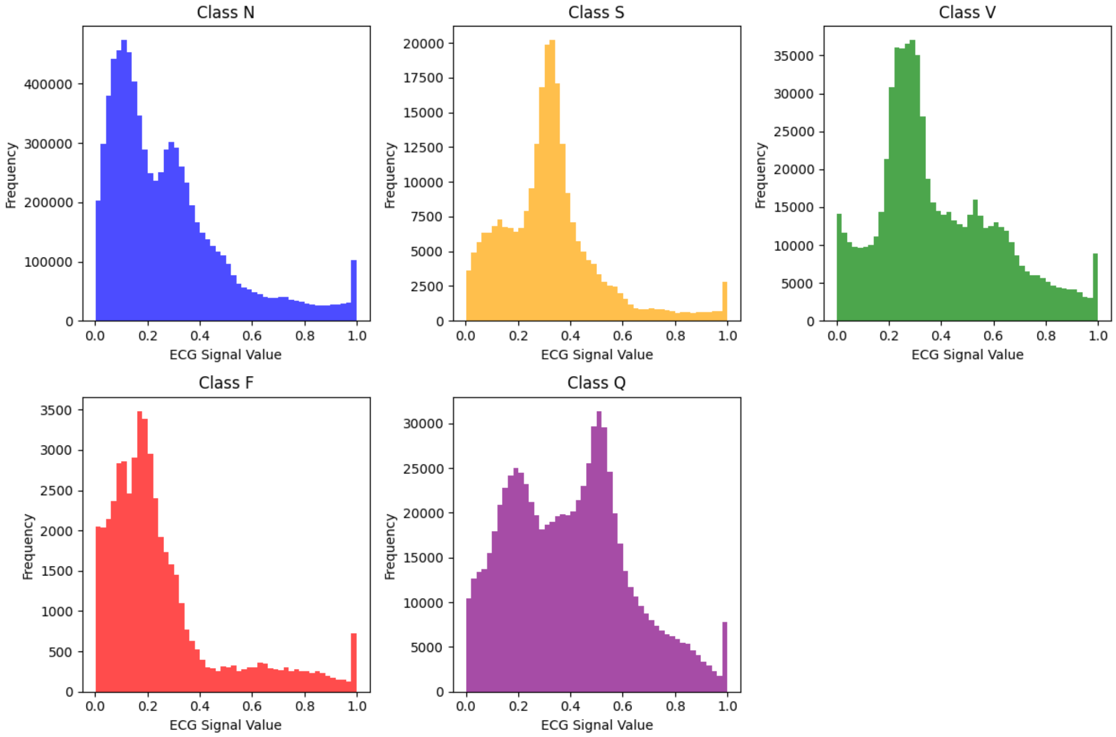



```
time: 1.03 s (started: 2024-04-15 22:55:52 +01:00)
```

Time series plot shows the different pattern in ECG signal among 5 distinct classes.

[ ]:
```
# Extract each class
X_train_0 = X_train[y_train == 0]
X_train_1 = X_train[y_train == 1]
X_train_2 = X_train[y_train == 2]
X_train_3 = X_train[y_train == 3]
```

```
X_train_4 = X_train[y_train == 4]

# Filter out zero-padding
X_train_0 = X_train_0[X_train_0 > 0]
X_train_1 = X_train_1[X_train_1 > 0]
X_train_2 = X_train_2[X_train_2 > 0]
X_train_3 = X_train_3[X_train_3 > 0]
X_train_4 = X_train_4[X_train_4 > 0]

plt.figure(figsize=(12, 8))

# Plot histogram for each class to see data characteristics
plt.subplot(2, 3, 1)
plt.hist(X_train_0.flatten(), bins=50, color='blue', alpha=0.7)
plt.title('Class N')
plt.xlabel('ECG Signal Value')
plt.ylabel('Frequency')

plt.subplot(2, 3, 2)
plt.hist(X_train_1.flatten(), bins=50, color='orange', alpha=0.7)
plt.title('Class S')
plt.xlabel('ECG Signal Value')
plt.ylabel('Frequency')

plt.subplot(2, 3, 3)
plt.hist(X_train_2.flatten(), bins=50, color='green', alpha=0.7)
plt.title('Class V')
plt.xlabel('ECG Signal Value')
plt.ylabel('Frequency')

plt.subplot(2, 3, 4)
plt.hist(X_train_3.flatten(), bins=50, color='red', alpha=0.7)
plt.title('Class F')
plt.xlabel('ECG Signal Value')
plt.ylabel('Frequency')

plt.subplot(2, 3, 5)
plt.hist(X_train_4.flatten(), bins=50, color='purple', alpha=0.7)
plt.title('Class Q')
plt.xlabel('ECG Signal Value')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```

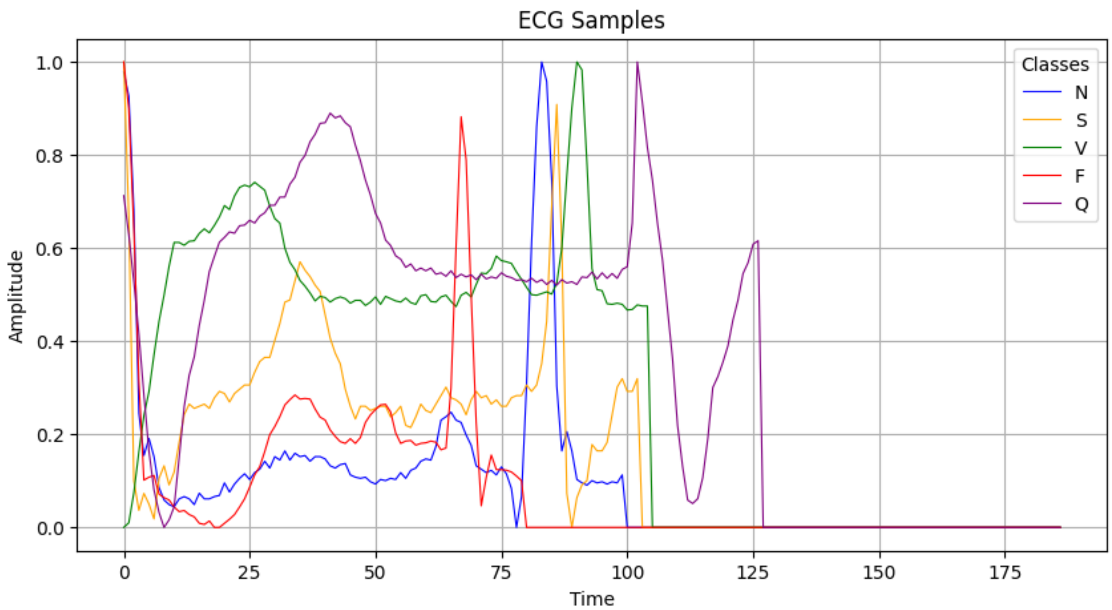


time: 2.49 s (started: 2024-04-15 23:11:09 +01:00)

Histograms show the differences in data distribution among 5 classes

[ ]:
```python
# Create subplots
fig, ax = plt.subplots(figsize=(10, 5))

# Define label mapping
label_mapping = {0: "N", 1: "S", 2: "V", 3: "F", 4: "Q"}

# Define color for each class
colors = ["blue", "orange", "green", "red", "purple"]

# Plot signal of each class
for label in np.unique(y_train):
    # Filter by each class
    X_lbl = X_train[y_train == label][:1]
    for sample in X_lbl:
        ax.plot(sample, color=colors[label], label=label_mapping[label], lw=0.8)

# Define title and label
ax.legend(title="Classes")
ax.set_title("ECG Samples")
ax.set_xlabel("Time")
```

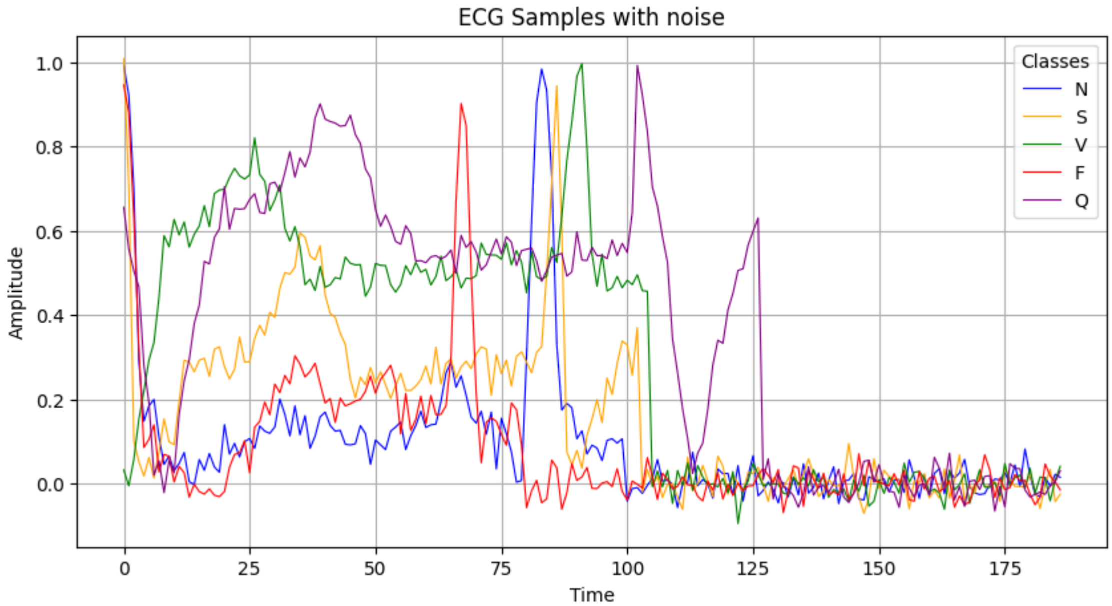
```
ax.set_ylabel("Amplitude")
ax.grid(True)

plt.show()
```




```
time: 438 ms (started: 2024-04-15 23:14:09 +01:00)
```

[ ]:
```
# Create subplots
fig, ax = plt.subplots(figsize=(10, 5))

# Define label mapping
label_mapping = {0: "N", 1: "S", 2: "V", 3: "F", 4: "Q"}

# Define color for each class
colors = ["blue", "orange", "green", "red", "purple"]

# Plot signal of each class
for label in np.unique(y_train):
    # Filter by each class
    X_lbl = _gaussian_noise(X_train)[y_train == label][:1]
    for sample in X_lbl:
        ax.plot(sample, color=colors[label], label=label_mapping[label], lw=0.8)

# Define title and label
ax.legend(title="Classes")
ax.set_title("ECG Samples with noise")
ax.set_xlabel("Time")
```

```
ax.set_ylabel("Amplitude")
ax.grid(True)

plt.show()
```




```
time: 5.11 s (started: 2024-04-15 23:15:22 +01:00)
```

ECG signals before and after applied guassian noise for augmentation to make model more generalized to unseen data

3 CNN

3.0.1 Custom CNN

Parameters to be tuned through Gridserch on 5 folds cross validation

- Dropout rate
- Number of hidden neurons

[ ]:
```
param_grid = {
    'module__dropout': [0.1, 0.2, 0.3, 0.4, 0.5],
    'module__neurons': [64, 128, 256]
}
```

```
time: 0 ns (started: 2024-04-15 23:27:54 +01:00)
```

[ ]:
```
# Perform parameters tuning
best_params = _cnn_pipeline_with_gridsearch(CNN, param_grid,
                                            checkpoint=False,
```

```
                                        balance=False, noise=False)
```

```
  epoch    train_acc    train_loss    valid_acc    valid_loss    cp      lr
dur
-------  -----------  ------------  -----------  ------------  ----  ------
-------
      1       0.8539        1.1332       0.9439
0.9902     +  0.0100  10.7654
      2       0.9495        0.9751       0.9569
0.9620     +  0.0100  9.0220
      3       0.9601        0.9586       0.9622
0.9516     +  0.0100  9.0153
      4       0.9648        0.9504       0.9655
0.9459     +  0.0100  8.7229
      5       0.9674        0.9453       0.9676
0.9424     +  0.0100  8.8085
      6       0.9693        0.9420       0.9694
0.9401     +  0.0100  8.5856
      7       0.9713        0.9393       0.9709
0.9381     +  0.0100  9.1406
      8       0.9731        0.9370       0.9720
0.9364     +  0.0100  8.8585
      9       0.9747        0.9349       0.9737
0.9349     +  0.0100  8.4018
     10       0.9761        0.9331       0.9742
0.9336     +  0.0100  9.4343
     11       0.9773        0.9315       0.9746
0.9335     +  0.0030  9.0732
     12       0.9779        0.9310       0.9749
0.9332     +  0.0030  9.2522
     13       0.9783        0.9306       0.9752
0.9329     +  0.0030  8.8847
     14       0.9786        0.9303       0.9754
0.9326     +  0.0030  8.6738
     15       0.9789        0.9299       0.9758
0.9324     +  0.0030  8.9361
     16       0.9793        0.9296       0.9758        0.9320
+  0.0030  9.3220
     17       0.9795        0.9292       0.9760
0.9318     +  0.0030  9.5151
     18       0.9797        0.9289       0.9764
0.9316     +  0.0030  9.5317
     19       0.9803        0.9285       0.9768
0.9314     +  0.0030  9.7765
     20       0.9805        0.9282       0.9772
0.9311     +  0.0030  9.3587
     21       0.9808        0.9278       0.9768        0.9311
+  0.0009  8.9586
```

22	0.9809	0.9278	0.9769	0.9310	+	0.0009	9.3320
23	0.9811	0.9277	0.9770	0.9310	+	0.0009	9.2943
24	0.9812	0.9276	0.9771	0.9309	+	0.0009	8.8890
25	0.9813	0.9274	0.9770	0.9308	+	0.0009	8.9451
26	0.9813	0.9273	0.9772	0.9308	+	0.0009	8.7363
27	0.9814	0.9273	0.9772	0.9307	+	0.0009	8.9048
28	0.9815	0.9272	0.9772	0.9306	+	0.0009	9.0423
29	0.9816	0.9271	0.9774	0.9305	+	0.0009	9.2815
30	0.9816	0.9271	0.9775	0.9305	+	0.0009	9.2394
31	0.9819	0.9269	0.9774	0.9304	+	0.0003	9.6172
32	0.9819	0.9268	0.9775	0.9304	+	0.0003	9.4173
33	0.9819	0.9268	0.9777	0.9304	+	0.0003	7.4128
34	0.9820	0.9268	0.9777	0.9304	+	0.0003	7.2125
35	0.9821	0.9268	0.9777	0.9304	+	0.0003	7.1140
36	0.9822	0.9267	0.9778	0.9304	+	0.0003	7.3040
37	0.9820	0.9267	0.9775	0.9303	+	0.0003	7.2956
38	0.9821	0.9267	0.9777	0.9303	+	0.0003	7.7008
39	0.9820	0.9267	0.9777	0.9303	+	0.0003	7.0570
40	0.9822	0.9266	0.9777	0.9303	+	0.0003	7.3041
41	0.9822	0.9265	0.9778	0.9303	+	0.0001	7.1863
42	0.9822	0.9266	0.9778	0.9303	+	0.0001	7.3058
43	0.9822	0.9266	0.9779	0.9303	+	0.0001	7.2984
44	0.9822	0.9266	0.9778	0.9303		0.0001	7.2452
45	0.9824	0.9265	0.9779	0.9302	+	0.0001	7.2556

```
     46        0.9823        0.9265        0.9779        0.9302
+  0.0001  7.3535
     47        0.9822        0.9265        0.9779
0.9302     +  0.0001  7.5370
     48        0.9822        0.9265        0.9780        0.9302
+  0.0001  7.2751
     49        0.9821        0.9265        0.9779        0.9302     +
0.0001  7.4848
     50        0.9823        0.9266        0.9780        0.9302     +
0.0001  7.0033
     51        0.9823        0.9265        0.9781        0.9302
+  0.0000  7.0686
     52        0.9822        0.9265        0.9781        0.9302
+  0.0000  7.2410
     53        0.9823        0.9265        0.9781        0.9302     +
0.0000  7.1883
     54        0.9824        0.9265        0.9781        0.9302     +
0.0000  7.3135
     55        0.9824        0.9265        0.9781        0.9302
0.0000  7.2605
     56        0.9822        0.9265        0.9781        0.9302     +
0.0000  7.4943
Stopping since valid_loss has not improved in the last 10 epochs.
Best score: 0.977442 with {'module__dropout': 0.1, 'module__neurons': 128}
time: 1d 9h 28min 15s (started: 2024-04-13 17:14:17 +01:00)
```

[ ]:
```python
print(best_params)
```

```
{'module__dropout': 0.1, 'module__neurons': 128}
time: 0 ns (started: 2024-04-15 10:35:15 +01:00)
```

Train model

[ ]:
```python
# Create final model with best parameter gained from grid search
model = _cnn_pipeline_with_best_param(CNN, best_params,
                                      earlystop_patience=15, checkpoint=True,
                                      class_weight=False, balance=False,␣
 ↪noise=False)
```

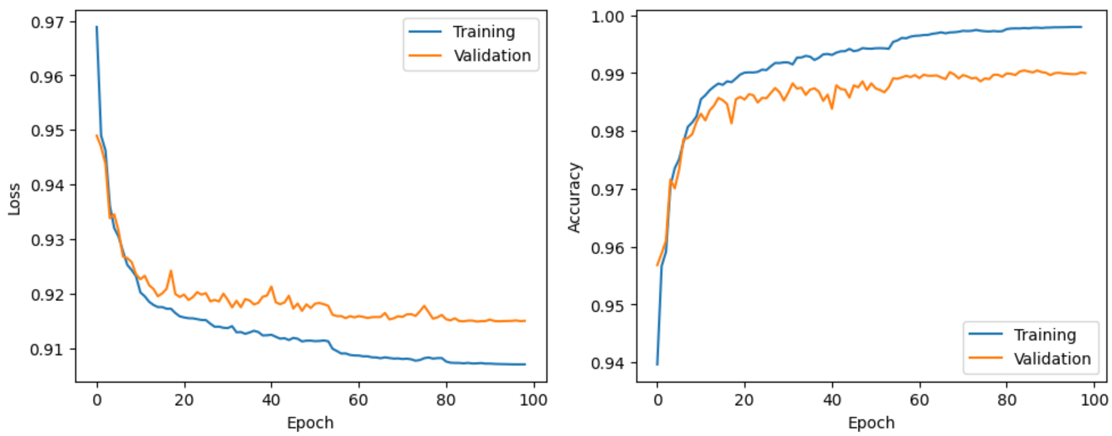
```
Re-initializing module because the following parameters were re-set: dropout,
neurons.
Re-initializing criterion.
Re-initializing optimizer.

C:\Users\kornk\anaconda3\envs\nn\lib\site-
packages\torch\optim\lr_scheduler.py:28: UserWarning: The verbose parameter is
deprecated. Please use get_last_lr() to access the learning rate.
  warnings.warn("The verbose parameter is deprecated. Please use get_last_lr() "

  epoch    train_acc    train_loss    valid_acc    valid_loss    cp     dur
-------  -----------  ------------  -----------  ------------  ----  ------
```

1	0.9397	0.9689	0.9568	0.9489	+	9.4336
2	0.9567	0.9489	0.9588	0.9469	+	9.1213
3	0.9591	0.9462	0.9611	0.9438	+	9.0853
4	0.9705	0.9361	0.9716	0.9338	+	9.3393
5	0.9736	0.9320	0.9701	0.9345		9.3094
6	0.9751	0.9304	0.9733	0.9315	+	10.3064
7	0.9781	0.9278	0.9785	0.9268	+	7.9249
8	0.9807	0.9253	0.9788	0.9265	+	9.5960
9	0.9815	0.9243	0.9794	0.9258	+	9.2702
10	0.9826	0.9232	0.9816	0.9235	+	9.0372
11	0.9855	0.9202	0.9830	0.9226	+	9.2424
12	0.9862	0.9195	0.9818	0.9233		9.4364
13	0.9871	0.9185	0.9836	0.9215	+	9.2527
14	0.9877	0.9179	0.9844	0.9208	+	9.5733
15	0.9882	0.9175	0.9857	0.9195	+	9.7071
16	0.9880	0.9175	0.9853	0.9200		9.0425
17	0.9886	0.9172	0.9846	0.9209		9.1401
18	0.9884	0.9172	0.9813	0.9242		9.2926
19	0.9891	0.9164	0.9855	0.9199		8.7236
20	0.9898	0.9158	0.9859	0.9194	+	9.3346
21	0.9901	0.9156	0.9854	0.9198		9.5157
22	0.9901	0.9155	0.9864	0.9188	+	9.3885
23	0.9901	0.9155	0.9862	0.9193		9.5768
24	0.9902	0.9153	0.9849	0.9203		9.2868
25	0.9906	0.9151	0.9857	0.9198		

9.3491
26 0.9905 0.9151 0.9857 0.9201
9.2264
27 0.9912 0.9145 0.9866
0.9186 + 9.5265
28 0.9918 0.9139 0.9874 0.9188
9.5518
29 0.9917 0.9139 0.9867 0.9186 +
9.3035
30 0.9919 0.9137 0.9853 0.9200
9.2138
31 0.9918 0.9137 0.9866 0.9188
9.4149
32 0.9915 0.9140 0.9882 0.9175
+ 9.2845
33 0.9927 0.9129 0.9873 0.9187
9.1830
34 0.9927 0.9129 0.9875 0.9175 9.0731
35 0.9930 0.9126 0.9862 0.9190
9.1072
36 0.9928 0.9129 0.9872 0.9187 9.0756
37 0.9923 0.9132 0.9874 0.9180 9.3046
38 0.9927 0.9129 0.9868 0.9183 9.3763
39 0.9932 0.9123 0.9852 0.9194
9.6227
40 0.9933 0.9124 0.9863 0.9196 9.6892
41 0.9931 0.9124 0.9838 0.9213 9.4488
42 0.9935 0.9121 0.9879 0.9184
9.3823
43 0.9938 0.9117 0.9873 0.9181
9.5229
44 0.9938 0.9118 0.9872 0.9184 9.4323
45 0.9942 0.9115 0.9858 0.9196
9.2577
46 0.9938 0.9119 0.9879 0.9173 +
9.4048
47 0.9939 0.9117 0.9876 0.9182 9.6746
48 0.9943 0.9112 0.9886
0.9168 + 9.7949
49 0.9942 0.9114 0.9871 0.9180 9.3420
50 0.9942 0.9114 0.9882 0.9173 9.3905
51 0.9943 0.9113 0.9874 0.9181 9.2120
52 0.9943 0.9113 0.9871 0.9183 9.0525
53 0.9943 0.9114 0.9867 0.9180 9.0384
54 0.9942 0.9113 0.9876 0.9178 9.2766
55 0.9954 0.9099 0.9891
0.9161 + 9.4243
56 0.9957 0.9095 0.9890 0.9159

```
+  9.5320
     57        0.9961        0.9090        0.9893
0.9159     +  9.1651
     58        0.9960        0.9090        0.9895        0.9155
+  9.5528
     59        0.9963        0.9087        0.9893        0.9159
9.4884
     60        0.9964        0.9087        0.9897        0.9156
9.5678
     61        0.9965        0.9086        0.9891        0.9159
9.3629
     62        0.9966        0.9085        0.9897        0.9157
9.4445
     63        0.9966        0.9085        0.9895        0.9155        9.3120
     64        0.9968        0.9083        0.9895        0.9157
9.5840
     65        0.9969        0.9083        0.9896        0.9157
9.5502
     66        0.9971        0.9081        0.9893        0.9157
9.1607
     67        0.9969        0.9083        0.9890        0.9164        9.5691
     68        0.9970        0.9082        0.9902        0.9153
+  9.7085
     69        0.9971        0.9081        0.9898        0.9155
9.4443
     70        0.9972        0.9081        0.9891        0.9158        9.4413
     71        0.9973        0.9080        0.9897        0.9157
9.5149
     72        0.9973        0.9081        0.9894        0.9162        9.3699
     73        0.9973        0.9079        0.9891        0.9162
9.7471
     74        0.9975        0.9077        0.9892        0.9159
9.6346
     75        0.9973        0.9078        0.9886        0.9167        8.9931
     76        0.9972        0.9082        0.9891        0.9178        8.4420
     77        0.9972        0.9083        0.9890        0.9166        9.3346
     78        0.9973        0.9081        0.9897        0.9154        9.1190
     79        0.9972        0.9082        0.9897        0.9156        9.2462
     80        0.9973        0.9082        0.9894        0.9161        9.2093
     81        0.9976        0.9075        0.9899        0.9153
9.1218
     82        0.9977        0.9073        0.9899        0.9151
+  9.1884
     83        0.9977        0.9073        0.9897        0.9155
9.2774
     84        0.9977        0.9073        0.9903
0.9150     +  8.4976
     85        0.9978        0.9072        0.9905
```

```
0.9149      +  8.8975
     86        0.9977        0.9073        0.9903        0.9150        9.7051
     87        0.9978        0.9072        0.9901        0.9150
9.1389
     88        0.9979        0.9072        0.9905        0.9149      +
9.2376
     89        0.9978        0.9073        0.9902        0.9149        8.9907
     90        0.9979        0.9072        0.9901        0.9150
9.6113
     91        0.9979        0.9072        0.9897        0.9152
9.1120
     92        0.9979        0.9071        0.9900        0.9150
9.2032
     93        0.9979        0.9071        0.9901        0.9149
9.0313
     94        0.9979        0.9071        0.9899        0.9149
9.2120
     95        0.9980        0.9070        0.9899        0.9150
9.6548
     96        0.9980        0.9070        0.9898        0.9150
9.0758
     97        0.9980        0.9070        0.9898        0.9151
9.3722
     98        0.9980        0.9070        0.9901        0.9149        9.0986
Stopping since valid_loss has not improved in the last 15 epochs.
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     57892
           1       1.00      0.96      0.98      1797
           2       1.00      1.00      1.00      4676
           3       0.99      0.93      0.96       496
           4       1.00      1.00      1.00      5182

    accuracy                           1.00     70043
   macro avg       1.00      0.98      0.99     70043
weighted avg       1.00      1.00      1.00     70043

              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14579
           1       0.94      0.84      0.89       426
           2       0.98      0.97      0.97      1112
           3       0.89      0.81      0.84       145
           4       1.00      0.99      1.00      1249

    accuracy                           0.99     17511
   macro avg       0.96      0.92      0.94     17511
weighted avg       0.99      0.99      0.99     17511
```




```
Best model saved to models/cnn/20240415T112137.pth
time: 15min 51s (started: 2024-04-15 11:05:46 +01:00)
```

Additional approaches for comparison

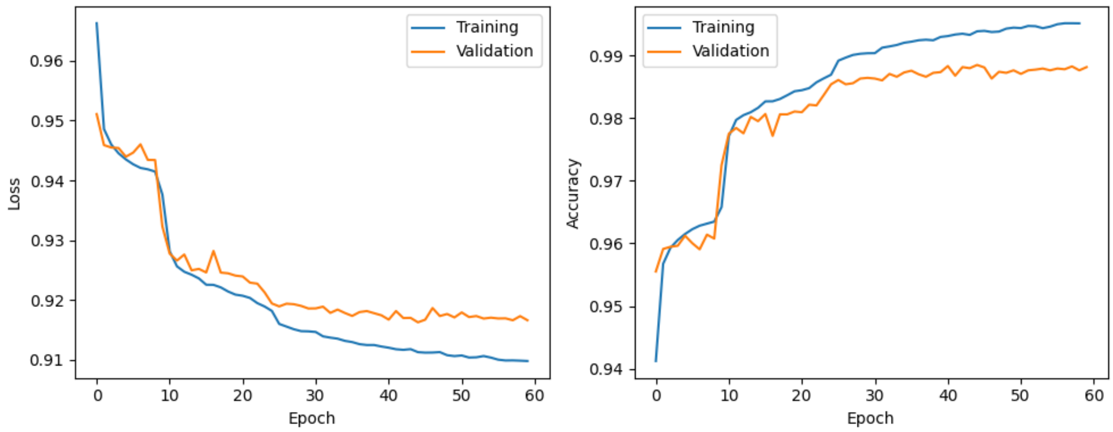
[ ]:
```python
# ReLU

best_params = {'module__dropout': 0.1,
               'module__activation': nn.ReLU,
               'module__neurons': 128}

model = _cnn_pipeline_with_best_param(CNN, best_params,
                                      earlystop_patience=15, checkpoint=True,
                                      class_weight=False, balance=False,␣
 ↪noise=False)
```

```
The autotime extension is already loaded. To reload it, use:
  %reload_ext autotime
Re-initializing module because the following parameters were re-set: activation,
dropout, neurons.
Re-initializing criterion.
Re-initializing optimizer.

C:\Users\kornk\anaconda3\envs\nn\lib\site-
packages\torch\optim\lr_scheduler.py:28: UserWarning: The verbose parameter is
deprecated. Please use get_last_lr() to access the learning rate.
  warnings.warn("The verbose parameter is deprecated. Please use get_last_lr() "

  epoch    train_acc    train_loss    valid_acc    valid_loss    cp     dur
-------  -----------  ------------  -----------  ------------  ----  ------
      1       0.9412        0.9663       0.9555
0.9511     +  9.4769
      2       0.9567        0.9486       0.9591
```

0.9459 + 9.0507
3 0.9593 0.9460 0.9595
0.9455 + 8.9376
4 0.9605 0.9445 0.9596
0.9454 + 9.0370
5 0.9615 0.9435 0.9612
0.9440 + 9.2897
6 0.9623 0.9427 0.9600 0.9446
8.9918
7 0.9628 0.9421 0.9591 0.9460
8.9469
8 0.9632 0.9418 0.9614
0.9434 + 9.2820
9 0.9635 0.9415 0.9608 0.9434
+ 8.8828
10 0.9658 0.9376 0.9725
0.9323 + 9.1782
11 0.9771 0.9281 0.9775
0.9277 + 9.2035
12 0.9797 0.9256 0.9784
0.9266 + 9.0920
13 0.9805 0.9247 0.9776 0.9276
9.1793
14 0.9809 0.9242 0.9802
0.9249 + 9.2926
15 0.9816 0.9236 0.9795 0.9252
9.1915
16 0.9827 0.9225 0.9806
0.9246 + 9.0351
17 0.9827 0.9225 0.9772 0.9282
9.1965
18 0.9830 0.9221 0.9806 0.9246
+ 9.1886
19 0.9836 0.9214 0.9806 0.9244
+ 9.1207
20 0.9843 0.9209 0.9810
0.9241 + 9.1066
21 0.9844 0.9207 0.9809 0.9239
+ 9.1089
22 0.9848 0.9203 0.9821
0.9229 + 9.3463
23 0.9857 0.9194 0.9820 0.9227
+ 9.3201
24 0.9863 0.9189 0.9837
0.9212 + 9.1017
25 0.9869 0.9181 0.9854
0.9194 + 9.0263
26 0.9891 0.9160 0.9861

0.9189 + 8.9709
27 0.9896 0.9155 0.9854 0.9194
9.0150
28 0.9901 0.9151 0.9856 0.9193
9.1316
29 0.9902 0.9148 0.9863 0.9190
8.9114
30 0.9903 0.9147 0.9864
0.9185 + 9.1689
31 0.9903 0.9146 0.9863 0.9186
9.0649
32 0.9912 0.9139 0.9860 0.9189
9.1915
33 0.9914 0.9137 0.9870
0.9178 + 9.0381
34 0.9916 0.9135 0.9866 0.9184
9.0842
35 0.9920 0.9131 0.9873
0.9178 + 9.0380
36 0.9922 0.9129 0.9876
0.9173 + 9.0866
37 0.9924 0.9126 0.9870 0.9179
9.0543
38 0.9925 0.9124 0.9866 0.9181
9.1183
39 0.9924 0.9124 0.9872 0.9178 8.9015
40 0.9929 0.9122 0.9873 0.9174
9.0708
41 0.9931 0.9120 0.9883
0.9167 + 8.8861
42 0.9933 0.9117 0.9868 0.9181
9.0306
43 0.9934 0.9116 0.9881 0.9170
9.2468
44 0.9933 0.9117 0.9880 0.9170 8.9612
45 0.9938 0.9112 0.9885
0.9162 + 9.0153
46 0.9939 0.9112 0.9881 0.9167
9.1338
47 0.9937 0.9112 0.9863 0.9186 8.9543
48 0.9938 0.9112 0.9874 0.9173 9.0585
49 0.9943 0.9107 0.9872 0.9176
9.0486
50 0.9944 0.9106 0.9876 0.9171
9.1488
51 0.9943 0.9107 0.9870 0.9179 9.0201
52 0.9947 0.9103 0.9876 0.9171
9.2848

```
     53       0.9946        0.9104       0.9877        0.9173        9.0446
     54       0.9943        0.9106       0.9879        0.9169        9.1593
     55       0.9946        0.9103       0.9876        0.9170
8.9580
     56       0.9950        0.9100       0.9879        0.9169
9.1170
     57       0.9951        0.9098       0.9878        0.9169
9.0373
     58       0.9951        0.9099       0.9882        0.9166        9.2081
     59       0.9951        0.9098       0.9876        0.9173
9.0646
Stopping since valid_loss has not improved in the last 15 epochs.
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     57892
           1       0.99      0.90      0.94      1797
           2       0.99      0.99      0.99      4676
           3       0.95      0.85      0.89       496
           4       1.00      1.00      1.00      5182

    accuracy                           1.00     70043
   macro avg       0.99      0.95      0.96     70043
weighted avg       1.00      1.00      1.00     70043

              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14579
           1       0.93      0.81      0.87       426
           2       0.98      0.96      0.97      1112
           3       0.90      0.79      0.84       145
           4       1.00      0.99      0.99      1249

    accuracy                           0.99     17511
   macro avg       0.96      0.91      0.93     17511
weighted avg       0.99      0.99      0.99     17511
```




```
Best model saved to models/cnn/20240413T150503.pth
time: 9min 28s (started: 2024-04-13 14:55:34 +01:00)
```

[ ]:
```
# With LeakyReLU

best_params = {'module__dropout': 0.1,
               'module__activation': nn.LeakyReLU,
               'module__neurons': 128}

%run "Tools.ipynb"
model = _cnn_pipeline_with_best_param(CNN, best_params,
                                      earlystop_patience=15, checkpoint=True,
                                      class_weight=None, balance=False,␣
 ↪noise=False)
```

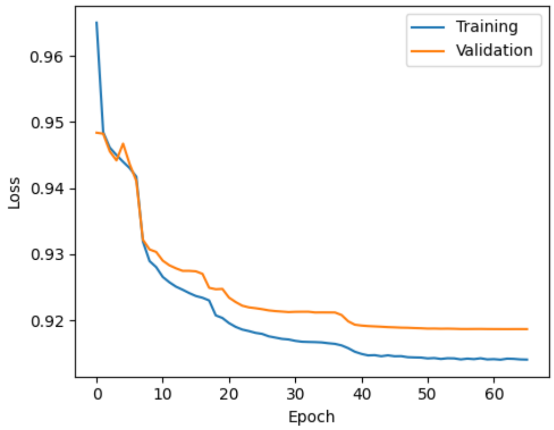
```
The autotime extension is already loaded. To reload it, use:
  %reload_ext autotime
Re-initializing module because the following parameters were re-set: activation,
dropout, neurons.
Re-initializing criterion.
Re-initializing optimizer.
  epoch    train_acc    train_loss    valid_acc    valid_loss    cp      lr
dur
-------  -----------  ------------  -----------  ------------  ----  ------
------
      1       0.9425        0.9650       0.9575
0.9484     +  0.0500  6.3081
      2       0.9567        0.9485       0.9573        0.9482
+  0.0500  6.3936
      3       0.9593        0.9461       0.9593
0.9456     +  0.0500  6.0357
      4       0.9601        0.9450       0.9605
0.9442     +  0.0500  6.2029
```

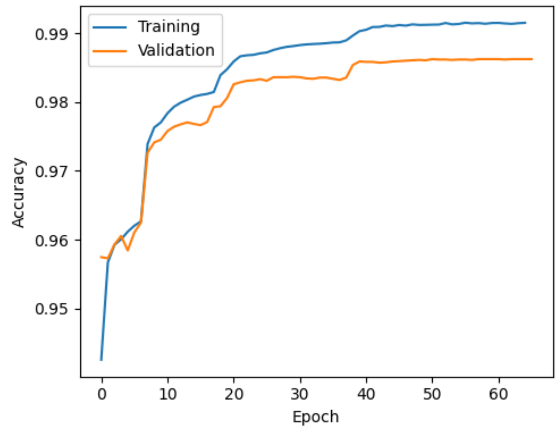
5 0.9612 0.9440 0.9584 0.9467
0.0500 6.3879
6 0.9620 0.9430 0.9611
0.9436 + 0.0500 6.2798
7 0.9627 0.9417 0.9625
0.9411 + 0.0500 6.2113
8 0.9739 0.9319 0.9726
0.9322 + 0.0500 6.2545
9 0.9763 0.9290 0.9741
0.9307 + 0.0500 6.2490
10 0.9770 0.9280 0.9745
0.9303 + 0.0500 6.5303
11 0.9784 0.9266 0.9758
0.9290 + 0.0150 6.3700
12 0.9793 0.9257 0.9764
0.9283 + 0.0150 6.2050
13 0.9799 0.9251 0.9768
0.9279 + 0.0150 6.4005
14 0.9803 0.9246 0.9770
0.9275 + 0.0150 6.6155
15 0.9808 0.9241 0.9768 0.9275
0.0150 6.2211
16 0.9810 0.9237 0.9766 0.9274
+ 0.0150 6.0781
17 0.9812 0.9234 0.9771
0.9270 + 0.0150 6.1254
18 0.9815 0.9230 0.9793
0.9249 + 0.0150 6.2195
19 0.9839 0.9207 0.9794
0.9247 + 0.0150 6.6632
20 0.9848 0.9203 0.9806 0.9247
0.0150 6.7984
21 0.9859 0.9196 0.9826
0.9234 + 0.0045 6.3415
22 0.9867 0.9190 0.9829
0.9228 + 0.0045 6.5141
23 0.9868 0.9186 0.9831
0.9222 + 0.0045 6.6634
24 0.9869 0.9184 0.9832
0.9220 + 0.0045 6.2202
25 0.9871 0.9181 0.9833
0.9218 + 0.0045 6.0442
26 0.9872 0.9179 0.9831 0.9217
+ 0.0045 6.2188
27 0.9876 0.9176 0.9836
0.9215 + 0.0045 6.3302
28 0.9878 0.9174 0.9836 0.9214
+ 0.0045 6.2968

29	0.9880	0.9172	0.9836	0.9213	+	0.0045	6.2819
30	0.9881	0.9171	0.9837	0.9212	+	0.0045	6.1148
31	0.9883	0.9169	0.9836	0.9213		0.0013	6.2650
32	0.9884	0.9168	0.9834	0.9213		0.0013	6.4477
33	0.9884	0.9167	0.9834	0.9213		0.0013	5.9978
34	0.9885	0.9167	0.9836	0.9212	+	0.0013	6.1424
35	0.9885	0.9166	0.9836	0.9212		0.0013	6.2621
36	0.9886	0.9165	0.9834	0.9212		0.0013	6.3442
37	0.9887	0.9164	0.9832	0.9212	+	0.0013	6.3761
38	0.9890	0.9162	0.9836	0.9208	+	0.0013	6.4961
39	0.9897	0.9158	0.9854	0.9198	+	0.0013	6.4411
40	0.9903	0.9152	0.9859	0.9193	+	0.0013	6.4088
41	0.9905	0.9149	0.9858	0.9192	+	0.0004	6.4701
42	0.9909	0.9147	0.9858	0.9191	+	0.0004	6.5533
43	0.9909	0.9147	0.9857	0.9191	+	0.0004	6.4603
44	0.9911	0.9146	0.9858	0.9190	+	0.0004	6.4932
45	0.9910	0.9147	0.9859	0.9190	+	0.0004	6.3853
46	0.9912	0.9146	0.9860	0.9189	+	0.0004	6.4278
47	0.9911	0.9146	0.9860	0.9189	+	0.0004	6.4227
48	0.9913	0.9144	0.9861	0.9189	+	0.0004	6.4535
49	0.9912	0.9144	0.9861	0.9188	+	0.0004	6.6075
50	0.9912	0.9144	0.9861	0.9188	+	0.0004	6.3223
51	0.9912	0.9142	0.9862	0.9187	+	0.0001	6.3359
52	0.9913	0.9143	0.9862	0.9188		0.0001	6.6168

```
     53        0.9915        0.9141        0.9862        0.9187
+  0.0001  6.3620
     54        0.9913        0.9143        0.9861        0.9187        0.0001
6.4619
     55        0.9913        0.9142        0.9862        0.9187     +
0.0001  6.4021
     56        0.9915        0.9141        0.9862        0.9187
+  0.0001  6.5470
     57        0.9914        0.9142        0.9861        0.9187     +
0.0001  6.2971
     58        0.9915        0.9141        0.9862        0.9187        0.0001
6.2987
     59        0.9914        0.9142        0.9862        0.9187        0.0001
6.2600
     60        0.9915        0.9141        0.9862        0.9187     +
0.0001  6.3117
     61        0.9915        0.9141        0.9862        0.9187     +
0.0000  6.3127
     62        0.9914        0.9140        0.9862        0.9187
+  0.0000  6.1781
     63        0.9914        0.9142        0.9862        0.9187     +
0.0000  6.2346
     64        0.9915        0.9142        0.9862        0.9187        0.0000
6.4051
     65        0.9915        0.9141        0.9862        0.9187        0.0000
6.1205
Stopping since valid_loss has not improved in the last 15 epochs.
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     57892
           1       0.99      0.80      0.89      1797
           2       0.99      0.99      0.99      4676
           3       0.96      0.73      0.83       496
           4       1.00      1.00      1.00      5182

    accuracy                           0.99     70043
   macro avg       0.99      0.90      0.94     70043
weighted avg       0.99      0.99      0.99     70043

              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14579
           1       0.93      0.74      0.82       426
           2       0.97      0.96      0.97      1112
           3       0.92      0.73      0.82       145
           4       0.99      0.99      0.99      1249

    accuracy                           0.99     17511
```

```
   macro avg       0.96      0.88      0.92     17511
weighted avg       0.99      0.99      0.99     17511
```






```
Best model saved to models/cnn/20240413T004512.pth
time: 7min 16s (started: 2024-04-13 00:37:55 +01:00)
```

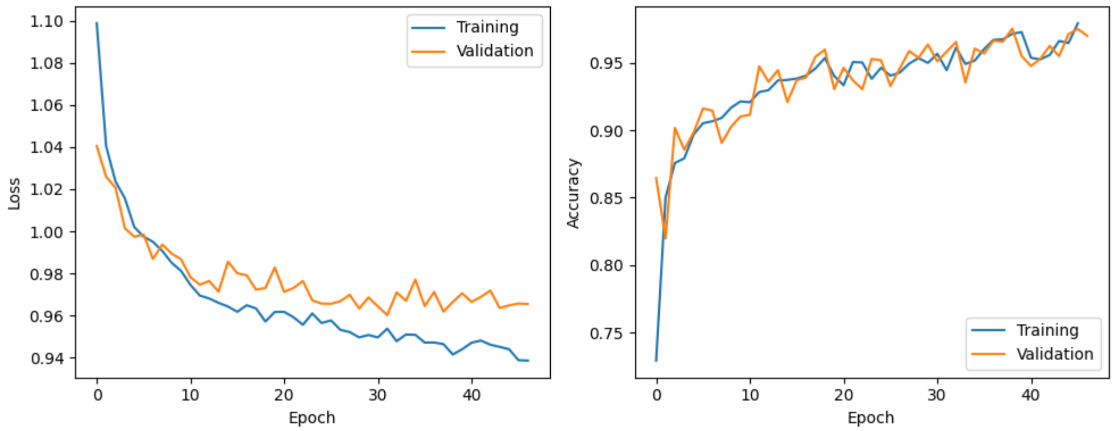
[ ]:
```
# With class weights

best_params = {'module__dropout': 0.1,
               'module__neurons': 128}

model = _cnn_pipeline_with_best_param(CNN, best_params,
                                      earlystop_patience=15, checkpoint=True,
                                      class_weight=True, balance=False,␣
 ↪noise=False)
```

```
The autotime extension is already loaded. To reload it, use:
  %reload_ext autotime
Re-initializing module because the following parameters were re-set: activation,
dropout, neurons.
Re-initializing criterion because the following parameters were re-set: weight.
Re-initializing optimizer.

C:\Users\kornk\anaconda3\envs\nn\lib\site-
packages\torch\optim\lr_scheduler.py:28: UserWarning: The verbose parameter is
deprecated. Please use get_last_lr() to access the learning rate.
  warnings.warn("The verbose parameter is deprecated. Please use get_last_lr() "

  epoch    train_acc    train_loss    valid_acc    valid_loss    cp     dur
-------  -----------  ------------  -----------  ------------  ----  ------
      1       0.7290        1.0989       0.8646
1.0405     +  9.3654
      2       0.8500        1.0405       0.8198        1.0259
```

					+	9.1883
3	0.8758	1.0237	0.9018	1.0208	+	9.1671
4	0.8792	1.0157	0.8857	1.0015	+	9.1999
5	0.8969	1.0021	0.8985	0.9975	+	9.0275
6	0.9053	0.9975	0.9162	0.9984		9.1147
7	0.9067	0.9950	0.9148	0.9869	+	9.0160
8	0.9093	0.9906	0.8907	0.9937		9.1138
9	0.9169	0.9850	0.9027	0.9894		9.5245
10	0.9215	0.9812	0.9103	0.9867	+	9.4169
11	0.9209	0.9745	0.9116	0.9781	+	9.7799
12	0.9284	0.9695	0.9476	0.9746	+	9.4232
13	0.9299	0.9681	0.9361	0.9764		9.3713
14	0.9371	0.9660	0.9447	0.9713	+	9.3350
15	0.9375	0.9642	0.9211	0.9856		9.1023
16	0.9385	0.9617	0.9372	0.9801		9.3777
17	0.9407	0.9649	0.9393	0.9791		9.3136
18	0.9460	0.9633	0.9547	0.9723		9.2096
19	0.9537	0.9571	0.9600	0.9731		9.1114
20	0.9404	0.9617	0.9307	0.9828		9.2274
21	0.9335	0.9617	0.9464	0.9712	+	9.0888
22	0.9508	0.9592	0.9372	0.9730		9.1070
23	0.9504	0.9556	0.9305	0.9764		9.1922
24	0.9383	0.9610	0.9531	0.9672	+	9.1775
25	0.9465	0.9564	0.9521	0.9656	+	9.0899
26	0.9406	0.9577	0.9329	0.9655	+	9.4965
27	0.9430	0.9532	0.9461	0.9668		9.3117

```
     28        0.9496        0.9521       0.9589        0.9698
9.0634
     29        0.9537        0.9496       0.9539        0.9633
+  9.3468
     30        0.9501        0.9507       0.9638        0.9686
9.1610
     31        0.9569        0.9496       0.9512        0.9643
9.5245
     32        0.9447        0.9537       0.9584        0.9602     +
9.0295
     33        0.9615        0.9477       0.9657        0.9710
9.1164
     34        0.9493        0.9510       0.9356        0.9669        9.1024
     35        0.9518        0.9508       0.9608        0.9771        9.0464
     36        0.9604        0.9471       0.9571        0.9644
9.0611
     37        0.9672        0.9472       0.9666        0.9712
9.3670
     38        0.9677        0.9464       0.9659        0.9618
9.1247
     39        0.9717        0.9415       0.9756        0.9664
9.3451
     40        0.9730        0.9440       0.9552        0.9705        9.1744
     41        0.9539        0.9471       0.9478        0.9664        9.1196
     42        0.9527        0.9481       0.9532        0.9689        9.2367
     43        0.9560        0.9461       0.9627        0.9719        9.2175
     44        0.9664        0.9451       0.9551        0.9635        9.3446
     45        0.9648        0.9440       0.9715        0.9648        9.3201
     46        0.9796        0.9388       0.9753        0.9657
9.3749
Stopping since valid_loss has not improved in the last 15 epochs.
              precision    recall  f1-score   support

           0       1.00      0.97      0.98     57892
           1       0.84      0.94      0.89      1797
           2       0.95      0.98      0.97      4676
           3       0.29      0.93      0.44       496
           4       0.99      1.00      0.99      5182

    accuracy                           0.97     70043
   macro avg       0.81      0.96      0.85     70043
weighted avg       0.99      0.97      0.98     70043

              precision    recall  f1-score   support

           0       0.99      0.97      0.98     14579
           1       0.80      0.86      0.83       426
           2       0.94      0.96      0.95      1112
```

```
           3       0.33      0.91      0.49       145
           4       0.99      0.99      0.99      1249

    accuracy                           0.97     17511
   macro avg       0.81      0.94      0.85     17511
weighted avg       0.98      0.97      0.97     17511
```




```
Best model saved to models/cnn/20240413T125808.pth
time: 7min 34s (started: 2024-04-13 12:50:33 +01:00)
```

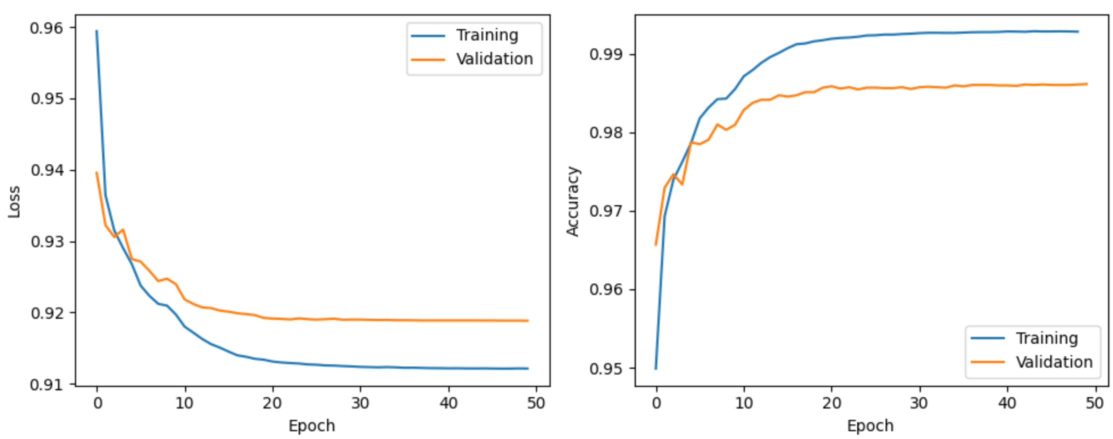
[ ]:
```python
# With noise

best_params = {'module__dropout': 0.1,
               'module__neurons': 128}

model = _cnn_pipeline_with_best_param(CNN, best_params,
                                      earlystop_patience=15, checkpoint=True,
                                      class_weight=False, balance=False,␣
 ↪noise=True)
```

```
The autotime extension is already loaded. To reload it, use:
  %reload_ext autotime
Re-initializing module because the following parameters were re-set: activation,
dropout, neurons.
Re-initializing criterion.
Re-initializing optimizer.
  epoch    train_acc    train_loss    valid_acc    valid_loss    cp      lr
dur
-------  -----------  ------------  -----------  ------------  ----  ------
------
      1       0.9499        0.9594       0.9657
0.9395     +  0.0500  7.5531
```

2	0.9693	0.9364	0.9729	0.9322	+	0.0500	7.4975
3	0.9739	0.9315	0.9746	0.9306	+	0.0500	7.5624
4	0.9762	0.9290	0.9733	0.9316		0.0500	7.5365
5	0.9786	0.9268	0.9787	0.9275	+	0.0500	7.6423
6	0.9818	0.9238	0.9785	0.9271	+	0.0500	7.7520
7	0.9831	0.9223	0.9790	0.9258	+	0.0500	7.4463
8	0.9842	0.9212	0.9810	0.9244	+	0.0500	7.6275
9	0.9843	0.9209	0.9803	0.9247		0.0500	7.4080
10	0.9855	0.9197	0.9809	0.9240	+	0.0500	7.4302
11	0.9871	0.9180	0.9828	0.9218	+	0.0150	7.6209
12	0.9879	0.9172	0.9837	0.9212	+	0.0150	7.7226
13	0.9888	0.9163	0.9841	0.9207	+	0.0150	7.3813
14	0.9895	0.9156	0.9841	0.9206	+	0.0150	7.4572
15	0.9901	0.9151	0.9847	0.9203	+	0.0150	7.4060
16	0.9907	0.9145	0.9845	0.9201	+	0.0150	7.3525
17	0.9912	0.9140	0.9847	0.9199	+	0.0150	7.6057
18	0.9913	0.9138	0.9851	0.9198	+	0.0150	7.6922
19	0.9916	0.9135	0.9851	0.9196	+	0.0150	7.3478
20	0.9917	0.9134	0.9857	0.9192	+	0.0150	7.4078
21	0.9919	0.9131	0.9858	0.9191	+	0.0045	7.4719
22	0.9920	0.9130	0.9856	0.9191	+	0.0045	7.4126
23	0.9920	0.9129	0.9857	0.9190	+	0.0045	7.6099
24	0.9921	0.9128	0.9854	0.9191		0.0045	7.5896
25	0.9923	0.9127	0.9857	0.9191		0.0045	7.4342

26	0.9923	0.9127	0.9857	0.9190	+	0.0045	7.4891
27	0.9924	0.9126	0.9856	0.9190		0.0045	7.4701
28	0.9924	0.9125	0.9856	0.9191		0.0045	7.4539
29	0.9925	0.9125	0.9857	0.9190	+	0.0045	7.4661
30	0.9925	0.9124	0.9855	0.9190		0.0045	7.6559
31	0.9926	0.9124	0.9857	0.9190		0.0013	7.4272
32	0.9926	0.9123	0.9858	0.9190	+	0.0013	7.6714
33	0.9926	0.9123	0.9857	0.9189	+	0.0013	7.5241
34	0.9926	0.9123	0.9857	0.9189		0.0013	7.5424
35	0.9926	0.9123	0.9860	0.9189	+	0.0013	7.5592
36	0.9927	0.9122	0.9858	0.9189		0.0013	7.5898
37	0.9927	0.9122	0.9860	0.9189	+	0.0013	7.5598
38	0.9927	0.9122	0.9860	0.9189	+	0.0013	7.4851
39	0.9927	0.9122	0.9860	0.9189		0.0013	7.6400
40	0.9928	0.9122	0.9860	0.9189		0.0013	7.5404
41	0.9928	0.9122	0.9860	0.9189		0.0004	7.6700
42	0.9928	0.9122	0.9859	0.9189		0.0004	7.7998
43	0.9928	0.9121	0.9861	0.9189		0.0004	7.5054
44	0.9928	0.9121	0.9860	0.9189		0.0004	7.3961
45	0.9928	0.9121	0.9861	0.9189	+	0.0004	7.4161
46	0.9928	0.9121	0.9860	0.9189	+	0.0004	7.4564
47	0.9928	0.9121	0.9860	0.9188	+	0.0004	7.3999
48	0.9928	0.9121	0.9860	0.9188	+	0.0004	7.5602
49	0.9928	0.9121	0.9861	0.9188		0.0004	7.9389

```
Stopping since valid_loss has not improved in the last 15 epochs.
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     57892
           1       0.99      0.82      0.90      1797
           2       0.99      0.99      0.99      4676
           3       0.99      0.82      0.90       496
           4       1.00      1.00      1.00      5182

    accuracy                           0.99     70043
   macro avg       0.99      0.92      0.96     70043
weighted avg       0.99      0.99      0.99     70043

              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14579
           1       0.94      0.75      0.84       426
           2       0.96      0.95      0.96      1112
           3       0.90      0.73      0.81       145
           4       1.00      0.99      0.99      1249

    accuracy                           0.99     17511
   macro avg       0.96      0.89      0.92     17511
weighted avg       0.99      0.99      0.99     17511
```



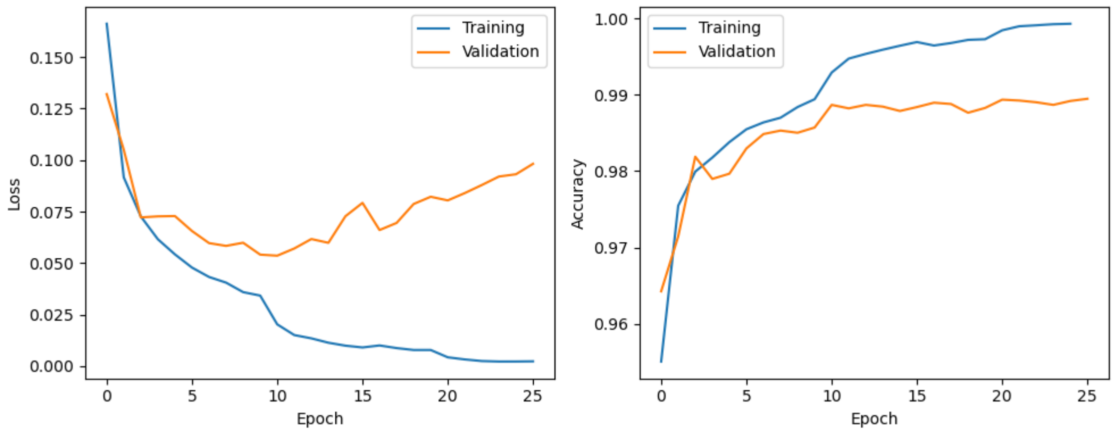

```
Best model saved to models/cnn/20240413T110602.pth
time: 6min 34s (started: 2024-04-13 10:59:28 +01:00)
```

[ ]:
```
# With Step learning rate scheduler

lr_scheduler = LRScheduler(policy='StepLR', step_size=10, gamma=0.3)
```

```python
best_params = {'module__dropout': 0.1,
               'module__neurons': 128}

model = _cnn_pipeline_with_best_param(CNN, best_params, lrschedule=lr_scheduler,
                                      earlystop_patience=15, checkpoint=True,
                                      class_weight=None, balance=False,␣
 ↪noise=False)
```

```
Re-initializing module because the following parameters were re-set: activation,
dropout, neurons.
Re-initializing criterion.
Re-initializing optimizer.
  epoch    train_acc    train_loss    valid_acc    valid_loss    cp      lr
dur
-------  -----------  ------------  -----------  ------------  ----  ------
------
      1       0.9550        0.1662       0.9643
0.1320     +  0.0100  7.5059
      2       0.9755        0.0916       0.9714
0.1050     +  0.0100  6.5200
      3       0.9799        0.0726       0.9819
0.0722     +  0.0100  6.4677
      4       0.9818        0.0616       0.9790        0.0727
0.0100  6.3224
      5       0.9838        0.0542       0.9797        0.0728
0.0100  6.2651
      6       0.9855        0.0478       0.9830
0.0656     +  0.0100  6.3778
      7       0.9864        0.0433       0.9849
0.0597     +  0.0100  6.3579
      8       0.9870        0.0405       0.9853
0.0583     +  0.0100  6.4560
      9       0.9884        0.0359       0.9850        0.0599
0.0100  6.4587
     10       0.9894        0.0342       0.9857
0.0541     +  0.0100  6.3375
     11       0.9929        0.0202       0.9887
0.0536     +  0.0030  6.4054
     12       0.9948        0.0150       0.9882        0.0571
0.0030  6.4361
     13       0.9954        0.0134       0.9887        0.0617
0.0030  6.3275
     14       0.9959        0.0113       0.9885        0.0598
0.0030  6.4277
     15       0.9964        0.0099       0.9879        0.0727
0.0030  6.4376
     16       0.9969        0.0090       0.9884        0.0792
0.0030  7.2261
```

```
     17        0.9965        0.0100        0.9890        0.0660
0.0030  6.4022
     18        0.9968        0.0087        0.9888        0.0695
0.0030  6.3624
     19        0.9972        0.0078        0.9877        0.0787
0.0030  7.1420
     20        0.9973        0.0078        0.9883        0.0822
0.0030  9.3075
     21        0.9985        0.0042        0.9894        0.0804
0.0009  7.3965
     22        0.9990        0.0032        0.9893        0.0840
0.0009  8.2413
     23        0.9991        0.0024        0.9890        0.0879
0.0009  7.9417
     24        0.9993        0.0022        0.9887        0.0920
0.0009  8.9985
     25        0.9993        0.0022        0.9892        0.0931        0.0009
6.9576
Stopping since valid_loss has not improved in the last 15 epochs.
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     57892
           1       1.00      0.99      1.00      1797
           2       1.00      1.00      1.00      4676
           3       0.96      0.95      0.95       496
           4       1.00      1.00      1.00      5182

    accuracy                           1.00     70043
   macro avg       0.99      0.99      0.99     70043
weighted avg       1.00      1.00      1.00     70043

              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14579
           1       0.92      0.85      0.88       426
           2       0.98      0.97      0.97      1112
           3       0.90      0.79      0.84       145
           4       1.00      0.99      0.99      1249

    accuracy                           0.99     17511
   macro avg       0.96      0.92      0.94     17511
weighted avg       0.99      0.99      0.99     17511
```




```
Best model saved to models/cnn/20240403T184404.pth
time: 3min 15s (started: 2024-04-03 18:40:48 +01:00)
```

[ ]:
```python
# With data balancing

best_params = {'module__dropout': 0.1,
               'module__neurons': 128}

model = _cnn_pipeline_with_best_param(CNN, best_params,
                                      earlystop_patience=15, checkpoint=True,
                                      class_weight=False, balance=20000,␣
 ↪noise=False)
```

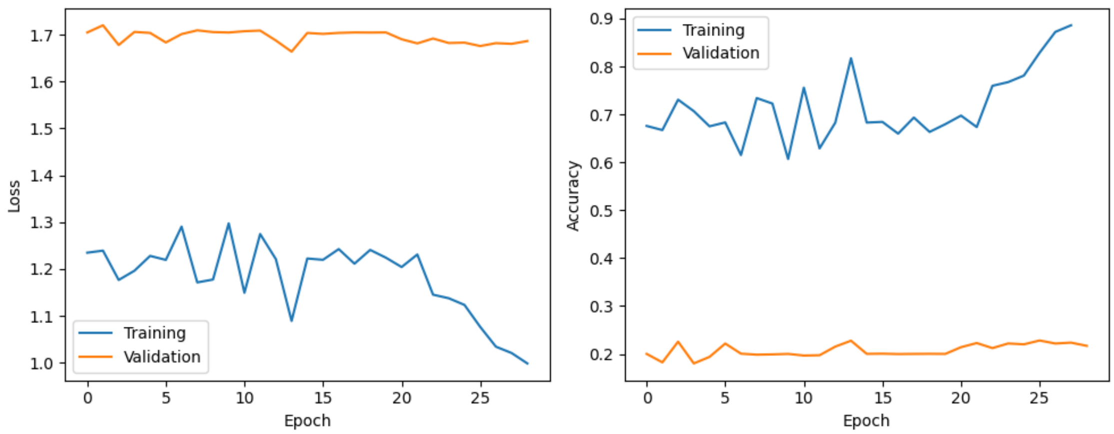
```
The autotime extension is already loaded. To reload it, use:
  %reload_ext autotime
Re-initializing module because the following parameters were re-set: activation,
dropout, neurons.
Re-initializing criterion.
Re-initializing optimizer.
```

```
/usr/local/lib/python3.10/dist-packages/torch/optim/lr_scheduler.py:28:
UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to
access the learning rate.
  warnings.warn("The verbose parameter is deprecated. Please use get_last_lr() "
```

```
  epoch    train_acc    train_loss    valid_acc    valid_loss    cp     dur
-------  -----------  ------------  -----------  ------------  ----  ------
      1       0.6758        1.2347       0.2000
1.7047     +  8.8092
      2       0.6672        1.2391       0.1824        1.7201
8.0163
      3       0.7307        1.1765       0.2256
1.6780     +  8.5924
      4       0.7067        1.1961       0.1801        1.7059          8.8382
```

```
      5      0.6751       1.2279      0.1938       1.7038       7.9770
      6      0.6832       1.2193      0.2217       1.6834       8.5457
      7      0.6150       1.2903      0.2005       1.7013
8.7747
      8      0.7340       1.1712      0.1984       1.7093
7.9737
      9      0.7226       1.1774      0.1990       1.7056       8.4941
     10      0.6068       1.2973      0.2000       1.7047
8.8832
     11      0.7558       1.1492      0.1966       1.7074
8.0631
     12      0.6289       1.2745      0.1972       1.7088       8.5041
     13      0.6826       1.2209      0.2152       1.6876       8.8821
     14      0.8171       1.0890      0.2276
1.6638     +  8.0340
     15      0.6829       1.2223      0.2001       1.7037       8.6226
     16      0.6843       1.2195      0.2005       1.7017       8.8141
     17      0.6598       1.2424      0.1997       1.7038       8.0918
     18      0.6934       1.2112      0.2000       1.7048       8.5498
     19      0.6634       1.2406      0.2002       1.7046       8.6406
     20      0.6795       1.2240      0.2000       1.7048       8.1519
     21      0.6974       1.2041      0.2140       1.6903       8.5743
     22      0.6735       1.2308      0.2227       1.6814       8.5315
     23      0.7596       1.1451      0.2121       1.6918       8.3079
     24      0.7673       1.1374      0.2218       1.6822       8.5838
     25      0.7809       1.1232      0.2200       1.6831       8.2319
     26      0.8286       1.0764      0.2280       1.6758
8.5398
     27      0.8721       1.0341      0.2217       1.6820
8.7515
     28      0.8859       1.0205      0.2235       1.6805
8.0470
Stopping since valid_loss has not improved in the last 15 epochs.
             precision    recall  f1-score   support

           0       0.40      0.02      0.04     20000
           1       0.00      0.00      0.00     20000
           2       0.71      0.06      0.11     20000
           3       0.00      0.00      0.00     20000
           4       0.21      1.00      0.34     20000

    accuracy                           0.22    100000
   macro avg       0.26      0.22      0.10    100000
weighted avg       0.26      0.22      0.10    100000

             precision    recall  f1-score   support

           0       0.46      0.02      0.04     20000
```

```
            1       0.00      0.00      0.00     20000
            2       0.82      0.06      0.12     20000
            3       0.00      0.00      0.00     20000
            4       0.21      1.00      0.34     20000

     accuracy                           0.22    100000
    macro avg       0.30      0.22      0.10    100000
 weighted avg       0.30      0.22      0.10    100000
```




```
Best model saved to drive/MyDrive/Colab
Notebooks/Project/models/cnn/20240415T100929.pth
time: 4min 22s (started: 2024-04-15 10:05:07 +00:00)
```

3.0.2 Residual neural network (based on M. Kachuee et al.)

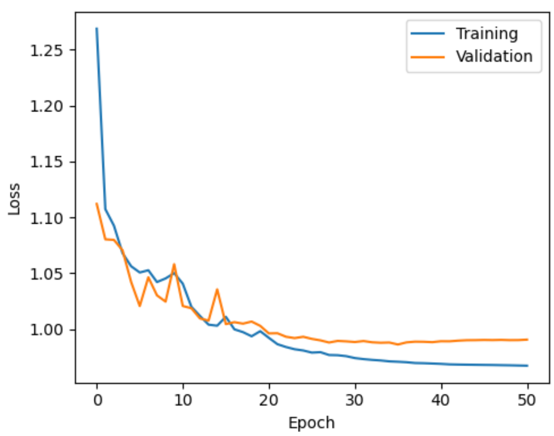
[ ]:
```
params = {'module__activation': nn.ReLU,
          'optimizer': optim.Adam,
          'lr': 0.001}

scheduler = LRScheduler(policy='ExponentialLR', gamma=0.75)

model = _cnn_pipeline_with_best_param(ResCNN, params, lr_scheduler=scheduler,
                                      earlystop_patience=15, checkpoint=True,
                                      optimizer=True)
```

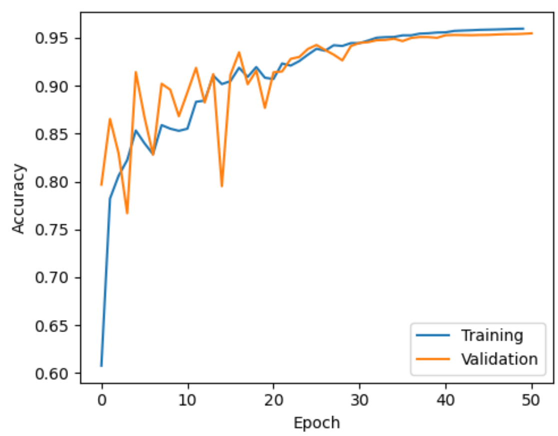
```
Re-initializing module because the following parameters were re-set: activation.
Re-initializing criterion because the following parameters were re-set: weight.
Re-initializing optimizer.
  epoch    train_acc    train_loss    valid_acc    valid_loss    cp      lr
dur
-------  -----------  ------------  -----------  ------------  ----  ------
-------
```

1 0.6076 1.2688 0.7967
1.1120 + 0.0010 45.0406
2 0.7822 1.1074 0.8654
1.0803 + 0.0010 22.6163
3 0.8060 1.0926 0.8294 1.0798
+ 0.0010 22.2166
4 0.8224 1.0680 0.7668 1.0705
+ 0.0010 22.8190
5 0.8533 1.0562 0.9143
1.0421 + 0.0010 22.6351
6 0.8401 1.0507 0.8675 1.0206
+ 0.0010 22.0015
7 0.8285 1.0527 0.8280 1.0464 0.0010
22.0334
8 0.8589 1.0421 0.9021 1.0302
0.0010 22.3899
9 0.8552 1.0453 0.8959 1.0246 0.0010
22.9173
10 0.8529 1.0501 0.8682 1.0581 0.0010
23.6895
11 0.8551 1.0406 0.8933 1.0207
0.0003 23.4467
12 0.8833 1.0200 0.9187
1.0187 + 0.0003 23.2027
13 0.8845 1.0118 0.8825 1.0095
+ 0.0003 23.2078
14 0.9107 1.0040 0.9122 1.0076
+ 0.0003 22.7886
15 0.9018 1.0030 0.7951 1.0356
0.0003 22.4892
16 0.9049 1.0111 0.9114 1.0045 +
0.0003 21.9644
17 0.9188 0.9998 0.9350 1.0062
0.0003 21.9745
18 0.9092 0.9973 0.9016 1.0049
0.0003 20.1796
19 0.9196 0.9936 0.9157 1.0068
0.0003 19.8418
20 0.9084 0.9982 0.8769 1.0029 +
0.0003 19.3045
21 0.9070 0.9922 0.9141 0.9962
+ 0.0001 18.4385
22 0.9233 0.9865 0.9150 0.9963
0.0001 17.5530
23 0.9211 0.9840 0.9281 0.9932
+ 0.0001 16.9179
24 0.9258 0.9820 0.9301 0.9920
+ 0.0001 17.1739

25	0.9325	0.9809	0.9384	0.9931		0.0001	16.9192
26	0.9386	0.9790	0.9426	0.9913	+	0.0001	17.2744
27	0.9366	0.9794	0.9374	0.9899	+	0.0001	16.6498
28	0.9423	0.9768	0.9325	0.9880	+	0.0001	16.7674
29	0.9416	0.9767	0.9265	0.9894		0.0001	16.7618
30	0.9446	0.9758	0.9415	0.9890		0.0001	14.3881
31	0.9445	0.9741	0.9448	0.9884		0.0000	15.6936
32	0.9472	0.9732	0.9456	0.9893		0.0000	15.0261
33	0.9502	0.9725	0.9476	0.9882		0.0000	14.2160
34	0.9508	0.9719	0.9479	0.9878	+	0.0000	14.1285
35	0.9510	0.9711	0.9491	0.9880		0.0000	12.4536
36	0.9527	0.9708	0.9467	0.9862	+	0.0000	13.0646
37	0.9527	0.9704	0.9499	0.9881		0.0000	12.3366
38	0.9545	0.9697	0.9509	0.9888		0.0000	12.0045
39	0.9548	0.9695	0.9508	0.9887		0.0000	11.4171
40	0.9556	0.9692	0.9501	0.9883		0.0000	11.7081
41	0.9558	0.9689	0.9528	0.9891		0.0000	11.3872
42	0.9573	0.9685	0.9531	0.9891		0.0000	11.4429
43	0.9577	0.9683	0.9529	0.9897		0.0000	9.0214
44	0.9580	0.9682	0.9528	0.9901		0.0000	9.6019
45	0.9584	0.9681	0.9531	0.9901		0.0000	9.5109
46	0.9586	0.9680	0.9532	0.9903		0.0000	8.8917
47	0.9589	0.9679	0.9536	0.9902		0.0000	8.4810
48	0.9591	0.9677	0.9539	0.9904		0.0000	8.7160

```
     49        0.9595        0.9676        0.9539        0.9901
0.0000  8.4946
     50        0.9596        0.9674        0.9543        0.9902
0.0000  8.7672
Stopping since valid_loss has not improved in the last 15 epochs.
              precision    recall  f1-score   support

           0       0.99      0.96      0.98     57892
           1       0.64      0.88      0.74      1797
           2       0.88      0.96      0.92      4676
           3       0.39      0.90      0.54       496
           4       0.97      0.98      0.97      5182

    accuracy                           0.96     70043
   macro avg       0.77      0.94      0.83     70043
weighted avg       0.97      0.96      0.96     70043

              precision    recall  f1-score   support

           0       0.99      0.96      0.97     14579
           1       0.59      0.81      0.68       426
           2       0.85      0.92      0.88      1112
           3       0.41      0.90      0.57       145
           4       0.97      0.97      0.97      1249

    accuracy                           0.95     17511
   macro avg       0.76      0.91      0.82     17511
weighted avg       0.96      0.95      0.96     17511
```






```
Best model saved to models/cnn/20240403T190703.pth
time: 15min 21s (started: 2024-04-03 18:51:41 +01:00)
```

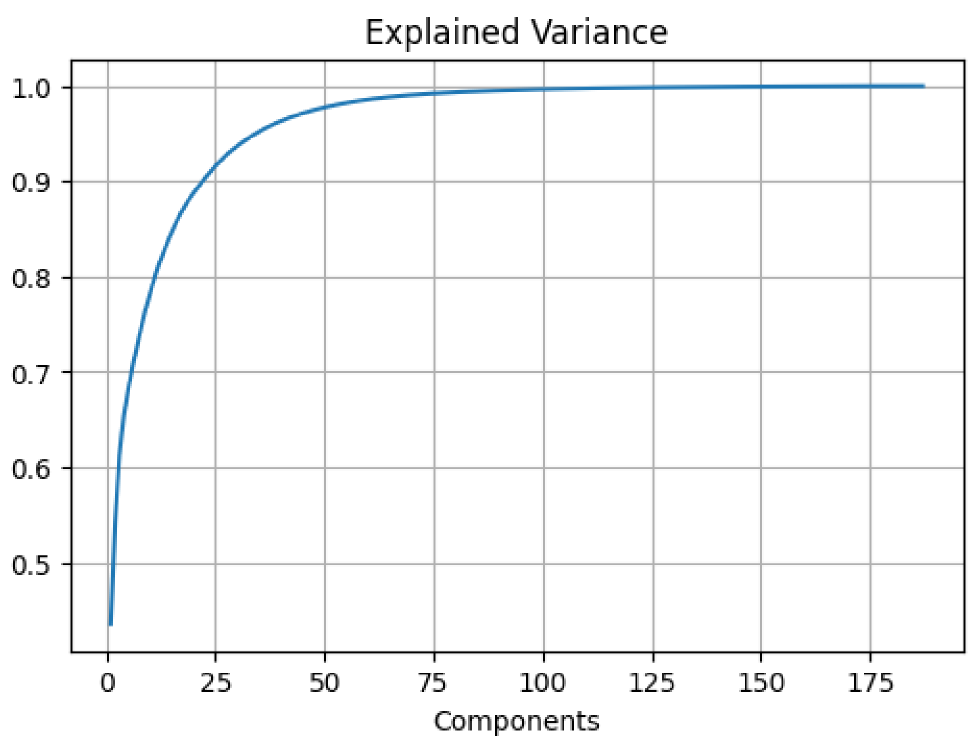
4 SVM

Evaluate number of components for PCA and LDA by examining explained variance plot

[ ]:
```
X_train, y_train = _import_data('train')
```

time: 5.12 s (started: 2024-04-03 19:09:46 +01:00)

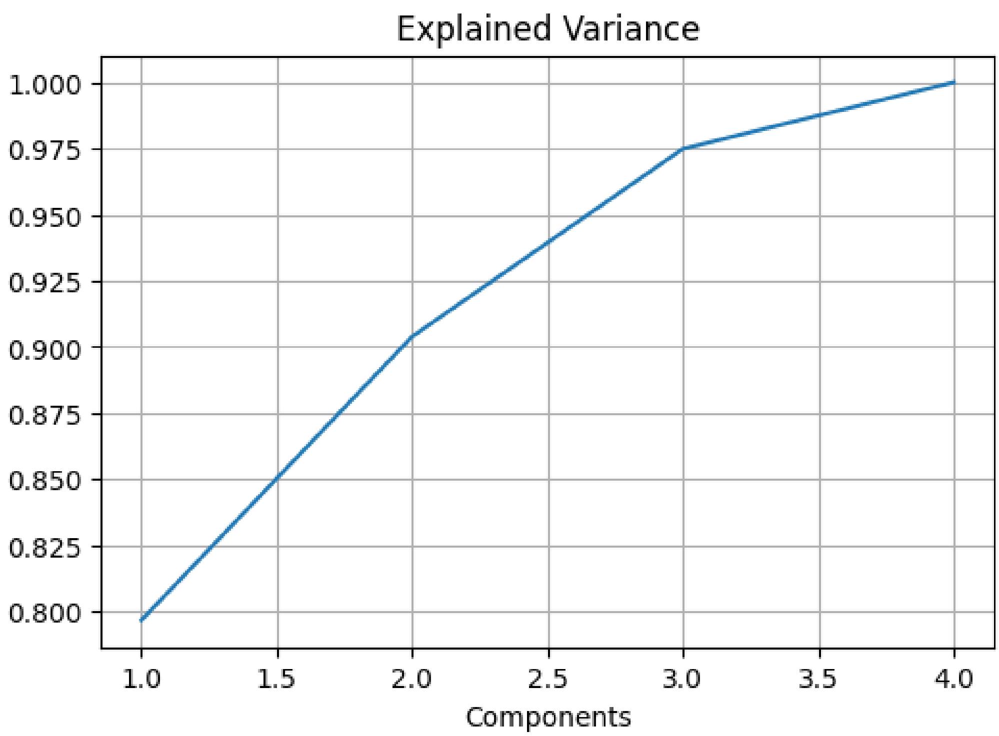
[ ]:
```
# Perform PCA on training set
pca = PCA()
pca.fit(X_train)

# Calculate the explained variance ratio
ev_pca = pca.explained_variance_ratio_.cumsum()

# Plot the explained variance
plt.figure(figsize=(6, 4))
plt.plot(range(1, len(ev_pca) + 1), ev_pca)
plt.title('Explained Variance')
plt.xlabel('Components')
plt.grid(True)
plt.show()
```



time: 2.36 s (started: 2024-04-16 00:09:09 +01:00)

From the explained variance plot, 100 components will be used as PCA captures nearly 100% of the variance

[ ]:
```
# Perform LDA on training set
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)

# Calculate the explained variance ratio
ev_lda = lda.explained_variance_ratio_.cumsum()

# Plot the explained variance
plt.figure(figsize=(6, 4))
plt.plot(range(1, len(ev_lda) + 1), ev_lda)
plt.title('Explained Variance')
plt.xlabel('Components')
plt.grid(True)
plt.show()
```




```
time: 2.48 s (started: 2024-04-16 00:09:14 +01:00)
```

From the explained variance plot, 4 components will be used as LDA captures 100% of the variance

Parameter Tuning and Model Training Parameters to be tuned through Gridserch on 5 folds cross validation - C (Regularization paramater) - Gamma - Kernel

[ ]:
```
params = {
    'svm__C': [0.1, 1, 10, 100, 1000],
    'svm__gamma': [0.001, 0.01, 0.1, 1],
    'svm__kernel': ['linear', 'rbf'],
}
```

```
time: 0 ns (started: 2024-04-03 19:09:20 +01:00)
```

[ ]:
```
# With class weight
_svm_pipeline(balanced_sample=None, dimredc=None,
              n_components=None, n_folds=5, class_weight='balanced',␣
 ↪model_fn=None,
              max_iter=1500)
```

```
(array([0, 1, 2, 3, 4], dtype=int64), array([72471,  2223,  5788,   641,  6431],
dtype=int64))
[LibSVM]
```

```
C:\Users\kornk\anaconda3\lib\site-packages\sklearn\svm\_base.py:297:
ConvergenceWarning: Solver terminated early (max_iter=1500).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
```

```
Best Parameters: {'svm__C': 100, 'svm__gamma': 1, 'svm__kernel': 'rbf'}
Model saved to models/svm/20240327T164911.pkl
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     72471
           1       1.00      1.00      1.00      2223
           2       1.00      1.00      1.00      5788
           3       0.98      1.00      0.99       641
           4       1.00      1.00      1.00      6431

    accuracy                           1.00     87554
   macro avg       1.00      1.00      1.00     87554
weighted avg       1.00      1.00      1.00     87554

              precision    recall  f1-score   support

           0       0.98      0.99      0.99     18118
           1       0.84      0.75      0.79       556
           2       0.96      0.93      0.94      1448
           3       0.85      0.75      0.80       162
           4       1.00      0.97      0.98      1608

    accuracy                           0.98     21892
   macro avg       0.93      0.88      0.90     21892
weighted avg       0.98      0.98      0.98     21892
```

```
time: 1h 22min 13s (started: 2024-03-27 15:31:30 +00:00)
```

Additional approaches for comparison

```
[ ]: # Downsampling majority to 20000, and upsampling using bootstraping method to
     ↪all minority classes to 20000 each
     _svm_pipeline(balanced_sample=20000, dimredc=None,
                   n_components=None, n_folds=5, model_fn=None,
                   max_iter=1500)
```

```
(array([0, 1, 2, 3, 4], dtype=int64), array([20000, 20000, 20000, 20000, 20000],
dtype=int64))
[LibSVM]

C:\Users\kornk\anaconda3\lib\site-packages\sklearn\svm\_base.py:297:
ConvergenceWarning: Solver terminated early (max_iter=1500).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(

Best Parameters: {'svm__C': 1000, 'svm__gamma': 1, 'svm__kernel': 'rbf'}
Model saved to models/svm/20240327T195247.pkl
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     72471
           1       0.83      1.00      0.91      2223
           2       0.96      1.00      0.98      5788
           3       0.87      1.00      0.93       641
           4       0.99      1.00      0.99      6431

    accuracy                           0.99     87554
   macro avg       0.93      1.00      0.96     87554
weighted avg       0.99      0.99      0.99     87554

              precision    recall  f1-score   support

           0       0.99      0.98      0.99     18118
           1       0.72      0.78      0.75       556
           2       0.92      0.94      0.93      1448
           3       0.74      0.79      0.76       162
           4       0.99      0.97      0.98      1608

    accuracy                           0.97     21892
   macro avg       0.87      0.89      0.88     21892
weighted avg       0.97      0.97      0.97     21892

time: 1h 50min 40s (started: 2024-03-27 18:06:42 +00:00)
```

```
[ ]: # With LDA, feature reduction
     _svm_pipeline(balanced_sample=None, scaler=None, dimredc='lda',
                   n_components=4, n_folds=5, class_weight='balanced', model_fn=None,
```

```
                 max_iter=1500)
```

```
(array([0, 1, 2, 3, 4], dtype=int64), array([72471,  2223,  5788,   641,  6431],
dtype=int64))
[LibSVM]

C:\Users\kornk\anaconda3\lib\site-packages\sklearn\svm\_base.py:297:
ConvergenceWarning: Solver terminated early (max_iter=1500).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(

Best Parameters: {'svm__C': 1000, 'svm__gamma': 1, 'svm__kernel': 'rbf'}
Model saved to models/svm/20240403T194254.pkl
              precision    recall  f1-score   support

           0       0.90      0.49      0.64     72471
           1       0.01      0.10      0.02      2223
           2       0.15      0.73      0.25      5788
           3       0.27      0.26      0.26       641
           4       0.13      0.03      0.05      6431

    accuracy                           0.46     87554
   macro avg       0.29      0.32      0.24     87554
weighted avg       0.77      0.46      0.55     87554

time: 15min 5s (started: 2024-04-03 19:29:23 +01:00)
```

[ ]:
```
# With PCA, feature reduction
_svm_pipeline(balanced_sample=None, scaler=None, dimredc='pca',
                 n_components=100, n_folds=5, class_weight='balanced',
 ↪model_fn=None,
                 max_iter=1500)
```

```
(array([0, 1, 2, 3, 4], dtype=int64), array([72471,  2223,  5788,   641,  6431],
dtype=int64))
[LibSVM]

C:\Users\kornk\anaconda3\lib\site-packages\sklearn\svm\_base.py:297:
ConvergenceWarning: Solver terminated early (max_iter=1500).  Consider pre-
processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(

Best Parameters: {'svm__C': 100, 'svm__gamma': 1, 'svm__kernel': 'rbf'}
Model saved to models/svm/20240327T180008.pkl
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     72471
           1       1.00      1.00      1.00      2223
           2       1.00      1.00      1.00      5788
           3       0.98      1.00      0.99       641
```

```
           4       1.00      1.00      1.00      6431

    accuracy                           1.00     87554
   macro avg       0.99      1.00      1.00     87554
weighted avg       1.00      1.00      1.00     87554

              precision    recall  f1-score   support

           0       0.98      0.99      0.99     18118
           1       0.83      0.74      0.79       556
           2       0.96      0.93      0.94      1448
           3       0.83      0.74      0.78       162
           4       1.00      0.97      0.98      1608

    accuracy                           0.98     21892
   macro avg       0.92      0.88      0.90     21892
weighted avg       0.98      0.98      0.98     21892

time: 47min 58s (started: 2024-03-27 17:15:18 +00:00)
```

5 Set up

[ ]:
```
%run "tools.ipynb"
```

```
The autotime extension is already loaded. To reload it, use:
  %reload_ext autotime
cpu
time: 94 ms (started: 2024-04-17 22:34:31 +01:00)
```

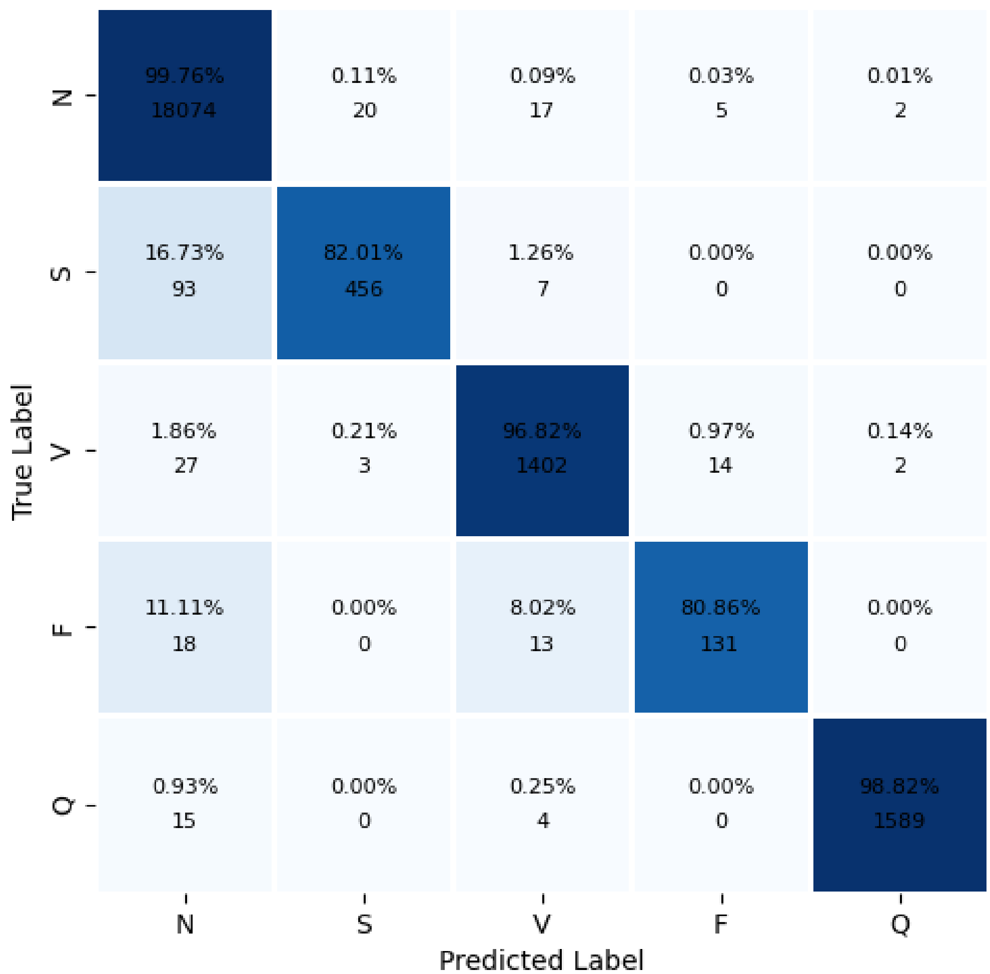
6 CNN

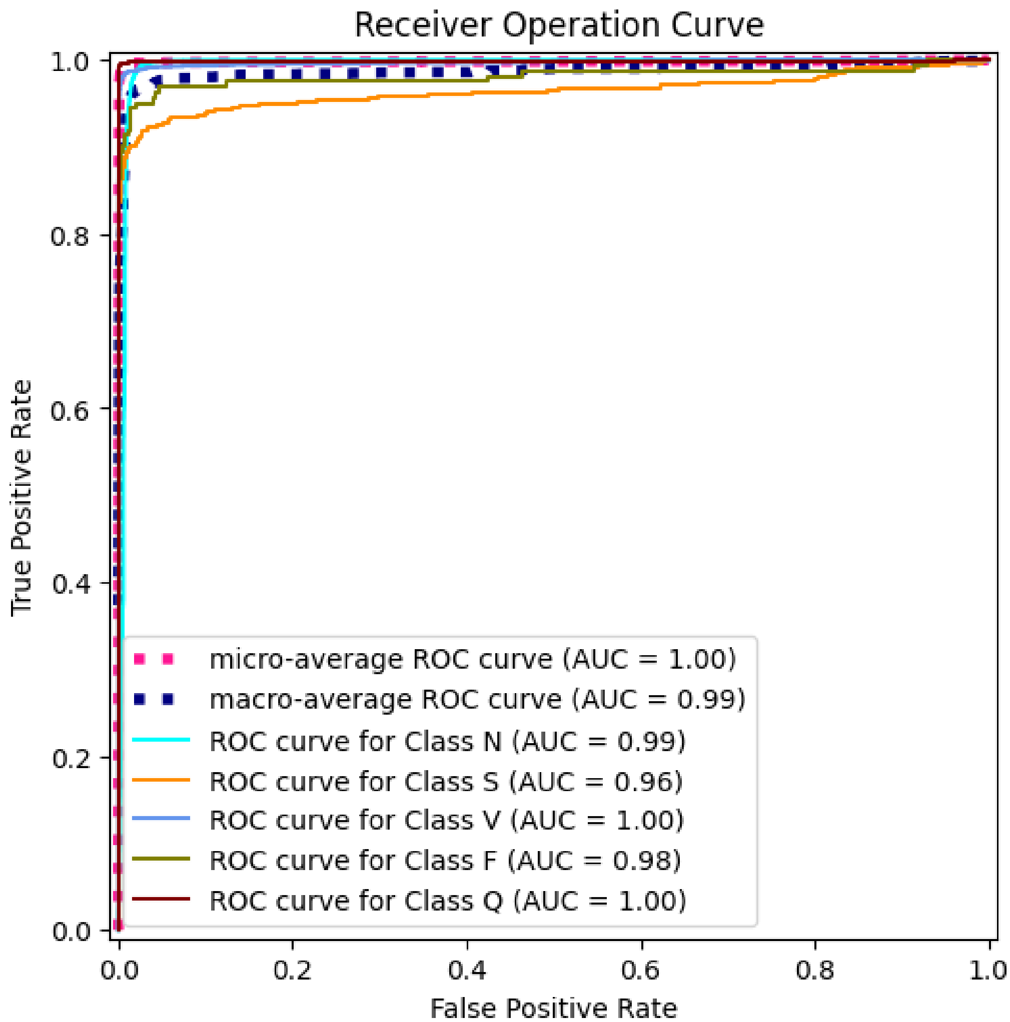
6.0.1 CNN

[ ]:
```
# Evaluate on Test set

_evaluate_cnn(CNN, '20240415T112137', subset='test')
```

```
Predicting time: 0:00:04.909453
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     18118
           1       0.95      0.82      0.88       556
           2       0.97      0.97      0.97      1448
           3       0.87      0.81      0.84       162
           4       1.00      0.99      0.99      1608

    accuracy                           0.99     21892
   macro avg       0.96      0.92      0.94     21892
```

```
weighted avg       0.99      0.99      0.99     21892
```





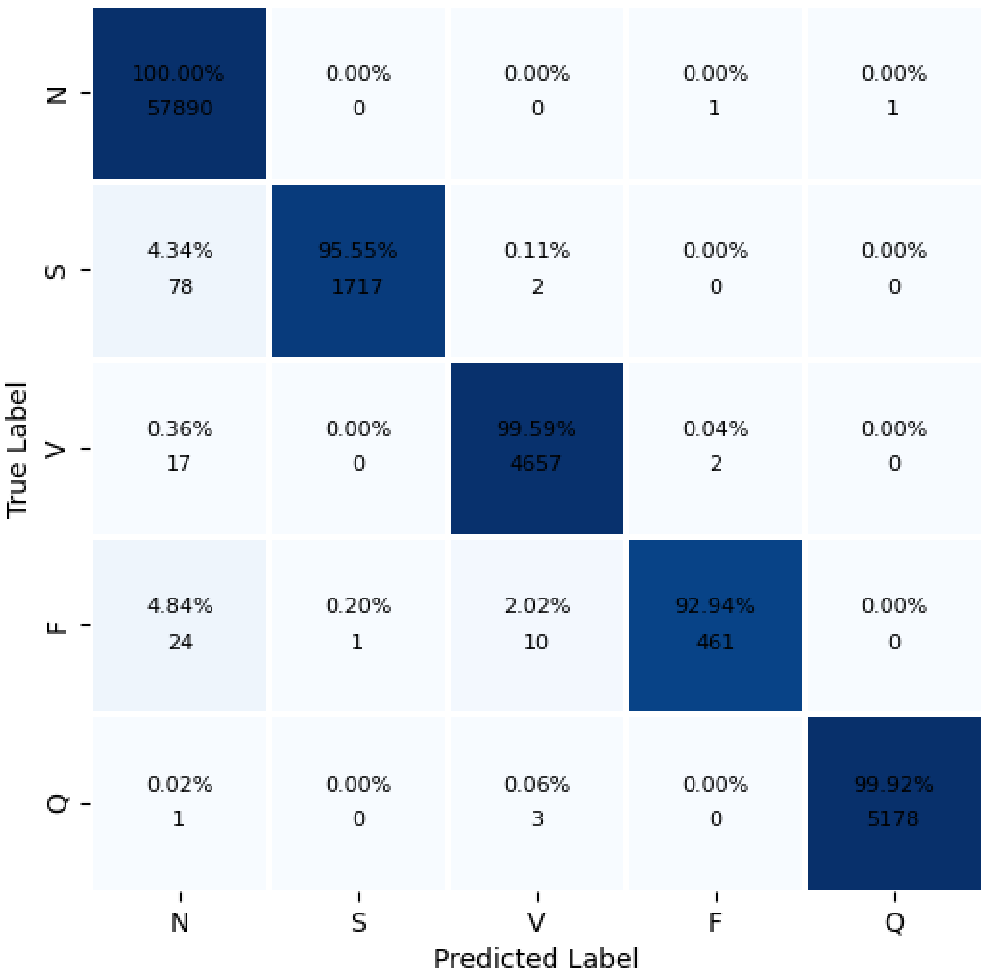

```
time: 6.72 s (started: 2024-04-17 23:31:04 +01:00)
```

Results on Train and Validation Set

[ ]:
```
# Evaluate on Train set

# _evaluate_cnn(CNN, '20240415T112137', subset='train', roc_curve=False)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     57892
           1       1.00      0.96      0.98      1797
           2       1.00      1.00      1.00      4676
           3       0.99      0.93      0.96       496
           4       1.00      1.00      1.00      5182

    accuracy                           1.00     70043
```

```
   macro avg       1.00      0.98      0.99     70043
weighted avg       1.00      1.00      1.00     70043
```

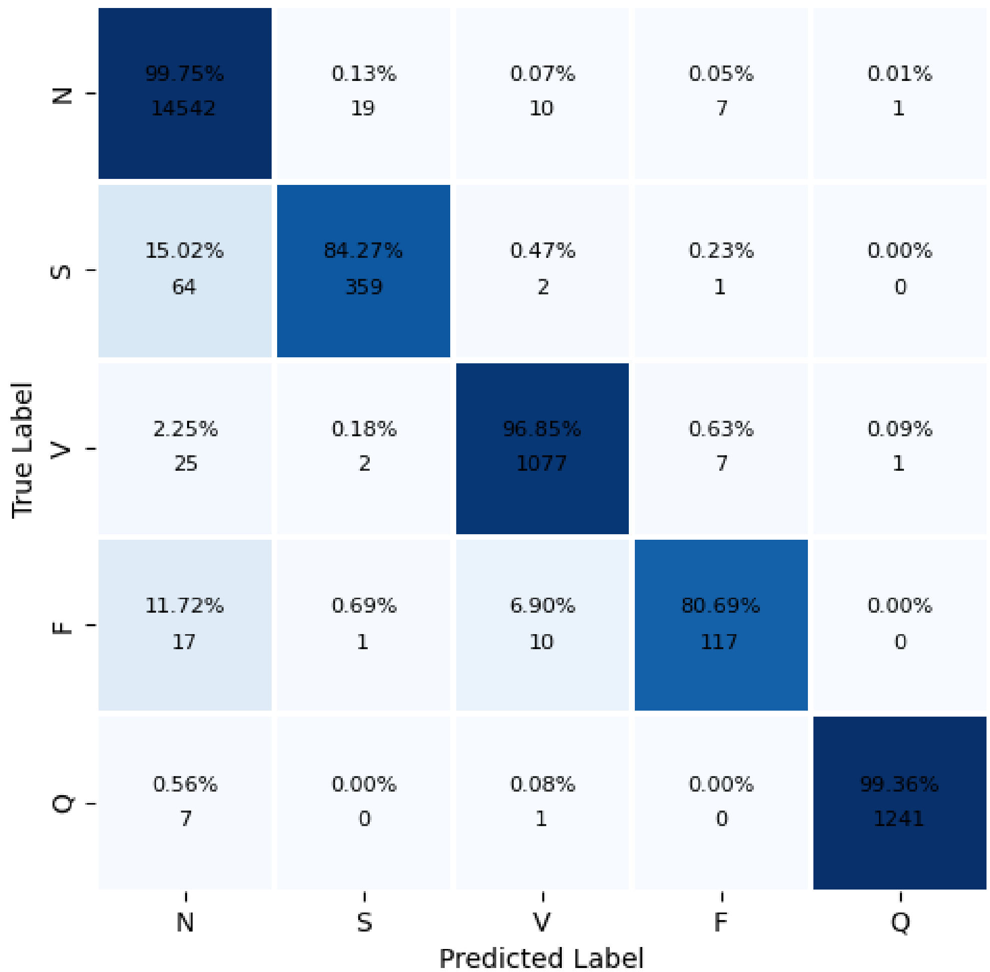



```
time: 23.5 s (started: 2024-04-16 00:38:20 +01:00)
```

[ ]:
```python
# Evaluate on Validation set

# _evaluate_cnn(CNN, '20240415T112137', subset='validation', roc_curve=False)
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14579
           1       0.94      0.84      0.89       426
           2       0.98      0.97      0.97      1112
           3       0.89      0.81      0.84       145
           4       1.00      0.99      1.00      1249
```

```
    accuracy                           0.99     17511
   macro avg       0.96      0.92      0.94     17511
weighted avg       0.99      0.99      0.99     17511
```




```
time: 9.42 s (started: 2024-04-16 00:39:08 +01:00)
```

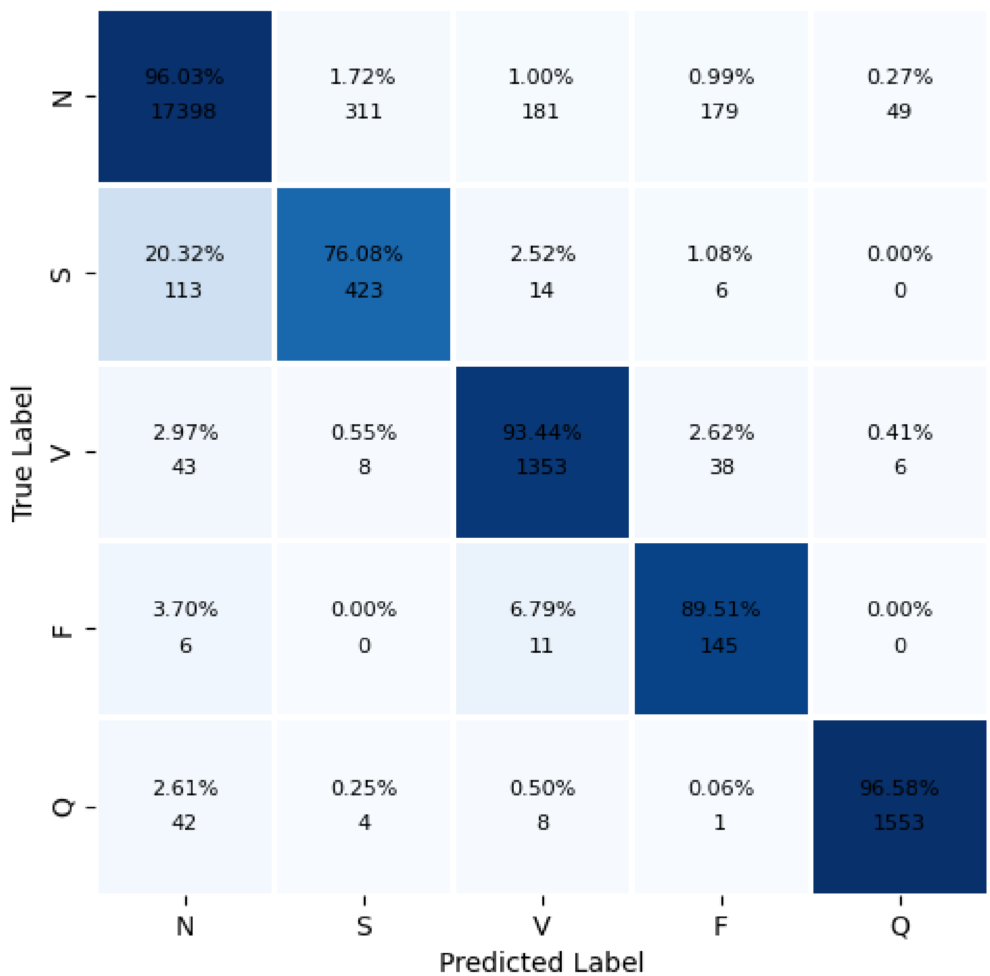
6.0.2 Residual neural network (replicate structure and parameters obtained from M. Kachuee et al.)

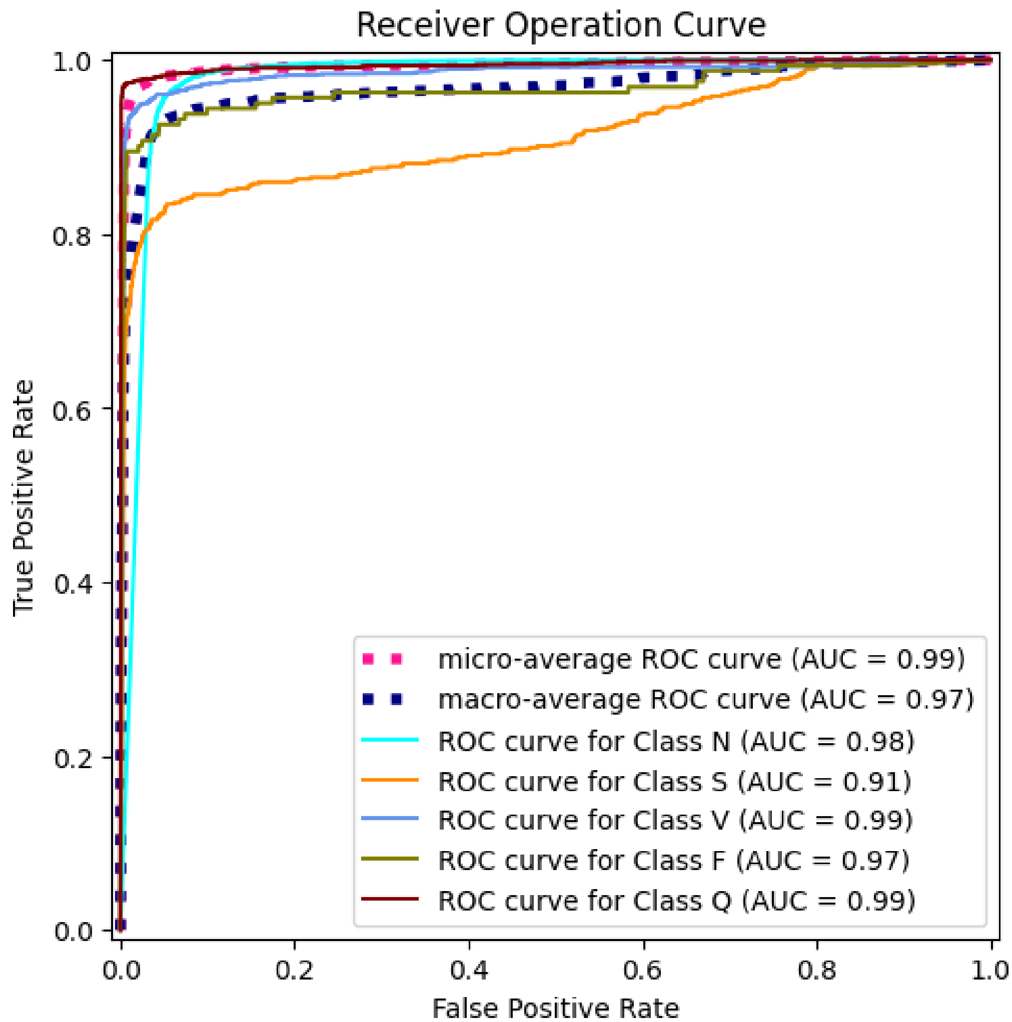
M. Kachuee, S. Fazeli and M. Sarrafzadeh, "ECG Heartbeat Classification: A Deep Transferable Representation," 2018 IEEE International Conference on Healthcare Informatics (ICHI), New York, NY, USA, 2018, pp. 443-444, doi: 10.1109/ICHI.2018.0009

[ ]:
```
# Evaluate on Test set
```

```
_evaluate_cnn(ResCNN, '20240403T190703', subset='test')
```

```
Predicting time: 0:00:03.349093
              precision    recall  f1-score   support

           0       0.99      0.96      0.97     18118
           1       0.57      0.76      0.65       556
           2       0.86      0.93      0.90      1448
           3       0.39      0.90      0.55       162
           4       0.97      0.97      0.97      1608

    accuracy                           0.95     21892
   macro avg       0.76      0.90      0.81     21892
weighted avg       0.96      0.95      0.96     21892
```





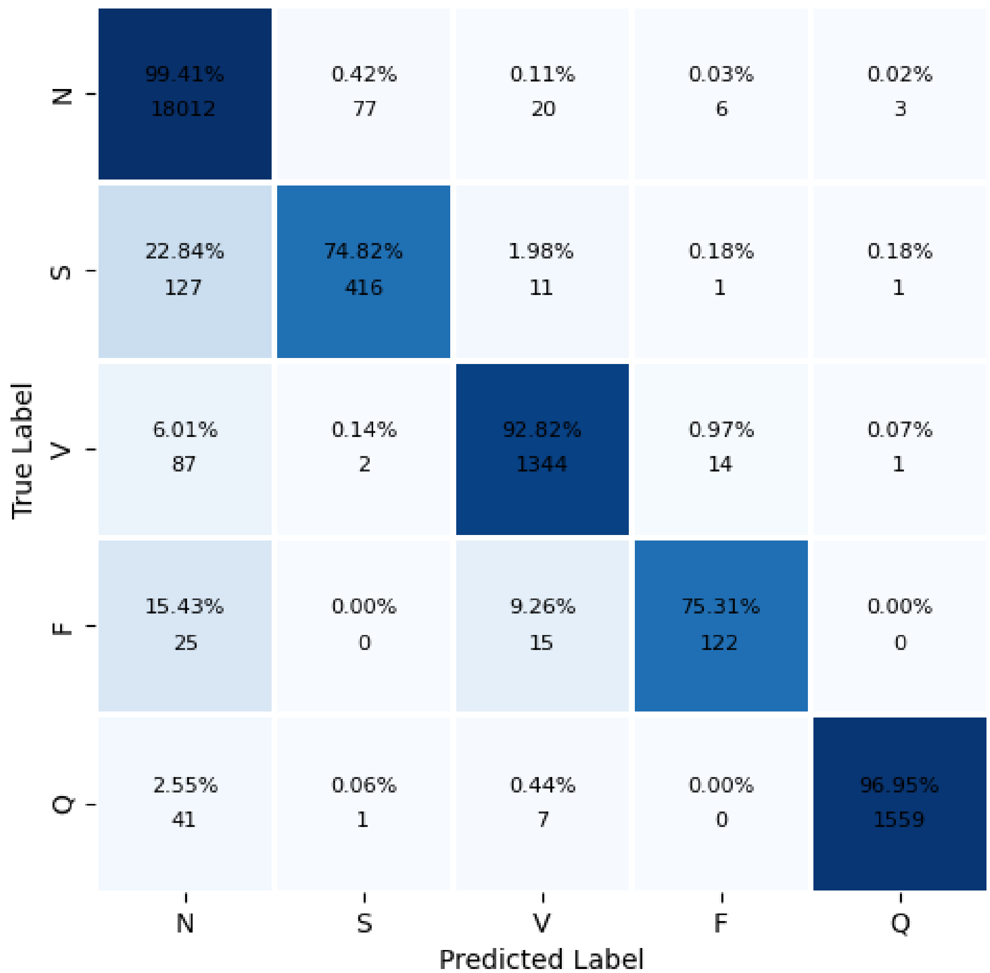

```
time: 5.5 s (started: 2024-04-17 20:46:02 +01:00)
```

7 SVM

[ ]:
```
# Evaluate on Test set without ROC curve
# Takes around 1.30 mins for loading model, and predicting labels

_evaluate_svm('20240415T231637', subset='test', roc_curve=False)
```

```
Predicting time: 0:00:57.763997
              precision    recall  f1-score   support

           0       0.98      0.99      0.99     18118
           1       0.84      0.75      0.79       556
           2       0.96      0.93      0.94      1448
```

```
           3       0.85      0.75      0.80       162
           4       1.00      0.97      0.98      1608

    accuracy                           0.98     21892
   macro avg       0.93      0.88      0.90     21892
weighted avg       0.98      0.98      0.98     21892
```

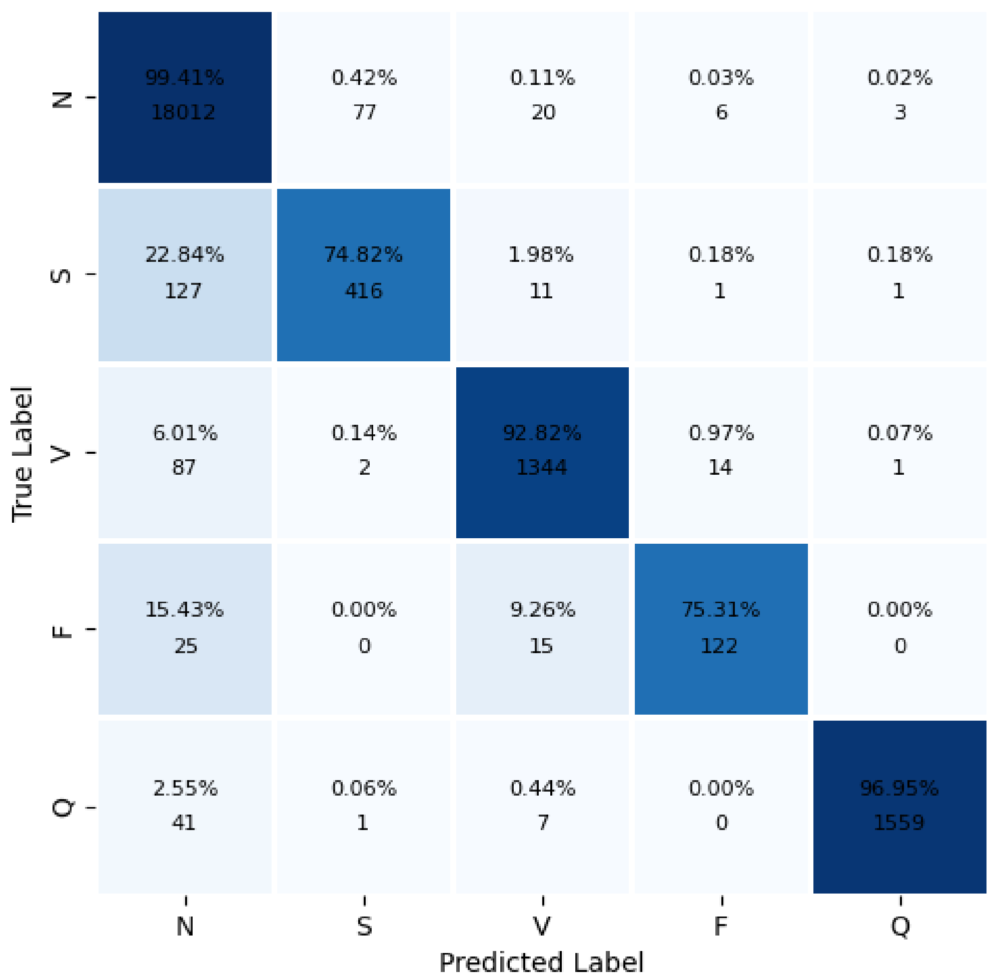


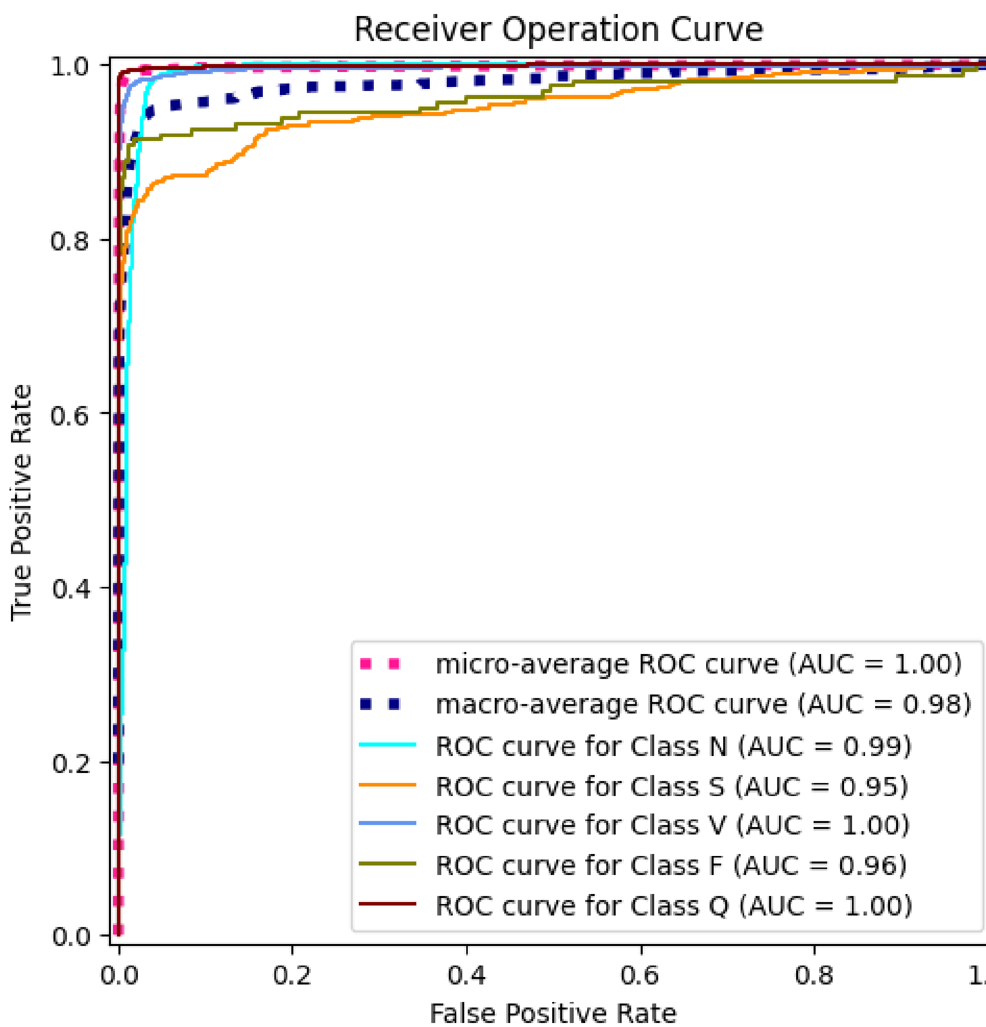

```
time: 58.7 s (started: 2024-04-17 22:32:04 +01:00)
```

[ ]:
```
# Evaluate on Test set with ROC curve
# Takes around 4 mins for loading model, predicting labels, and generate ROC
↪curve

_evaluate_svm('20240415T231637', subset='test')
```

```
Predicting time: 0:00:59.283368
```

	precision	recall	f1-score	support
0	0.98	0.99	0.99	18118
1	0.84	0.75	0.79	556
2	0.96	0.93	0.94	1448
3	0.85	0.75	0.80	162
4	1.00	0.97	0.98	1608
accuracy			0.98	21892
macro avg	0.93	0.88	0.90	21892
weighted avg	0.98	0.98	0.98	21892





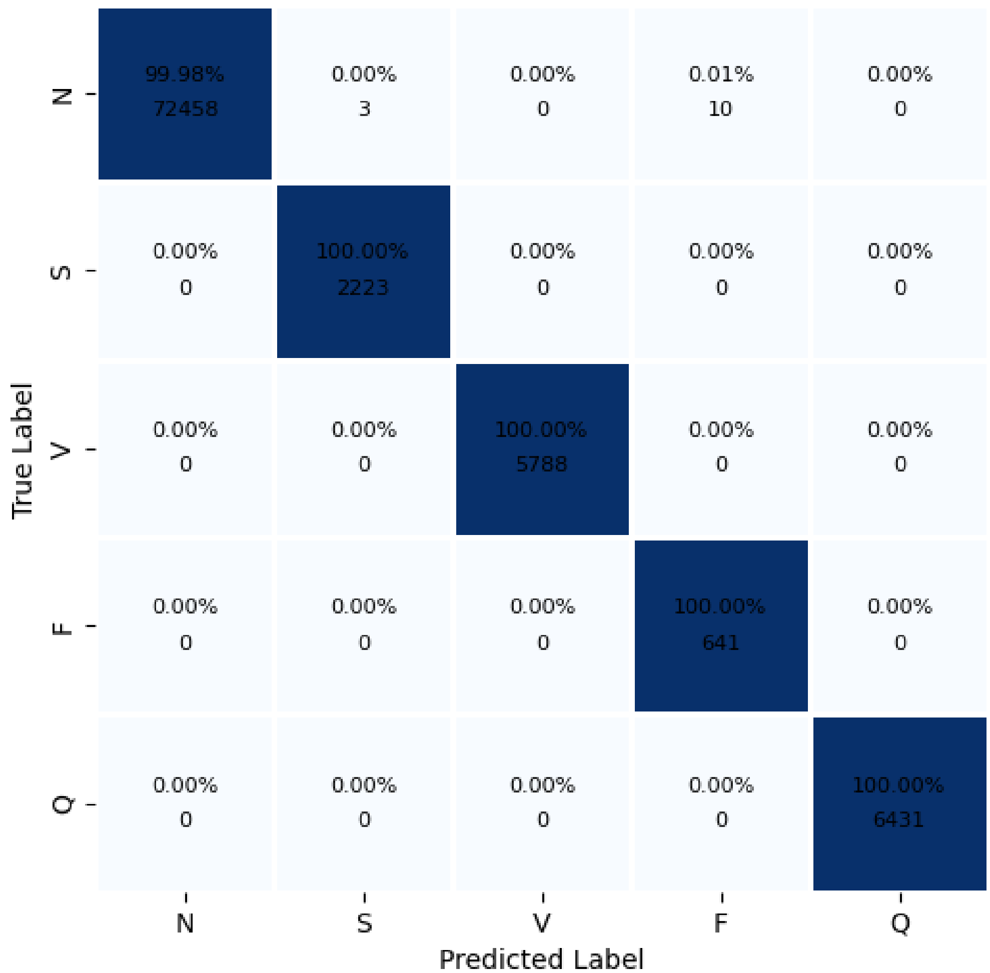

```
time: 2min (started: 2024-04-17 22:34:35 +01:00)
```

Results on Train Set

[ ]:
```
# Evaluate on Train set with ROC curve
# Takes around 6 mins for loading model, predicting labels

# _evaluate_svm('20240415T231637', subset='train', roc_curve=False)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     72471
           1       1.00      1.00      1.00      2223
           2       1.00      1.00      1.00      5788
           3       0.98      1.00      0.99       641
           4       1.00      1.00      1.00      6431
```

accuracy			1.00	87554
macro avg	1.00	1.00	1.00	87554
weighted avg	1.00	1.00	1.00	87554




```
time: 5min 58s (started: 2024-04-16 00:45:33 +01:00)
```