Model Gradient-Prediction Scatter

November 23, 2021

[ ]:
```
import pickle
import numpy as np
import torch
import matplotlib.pyplot as plt
import glob
import seaborn as sns
from torch import tensor
```

[2]:
```
import seaborn as sns
cmap_green = sns.light_palette("seagreen", as_cmap=True)
cmap_green
```

[2]:

[3]:
```
cmap_red = sns.light_palette('salmon', as_cmap=True)
cmap_red
```

[3]:

[4]:
```
cmap_both = sns.diverging_palette(10, 150, s=75, l=50, center='light',␣
↪as_cmap=True)
cmap_both
```

[4]:

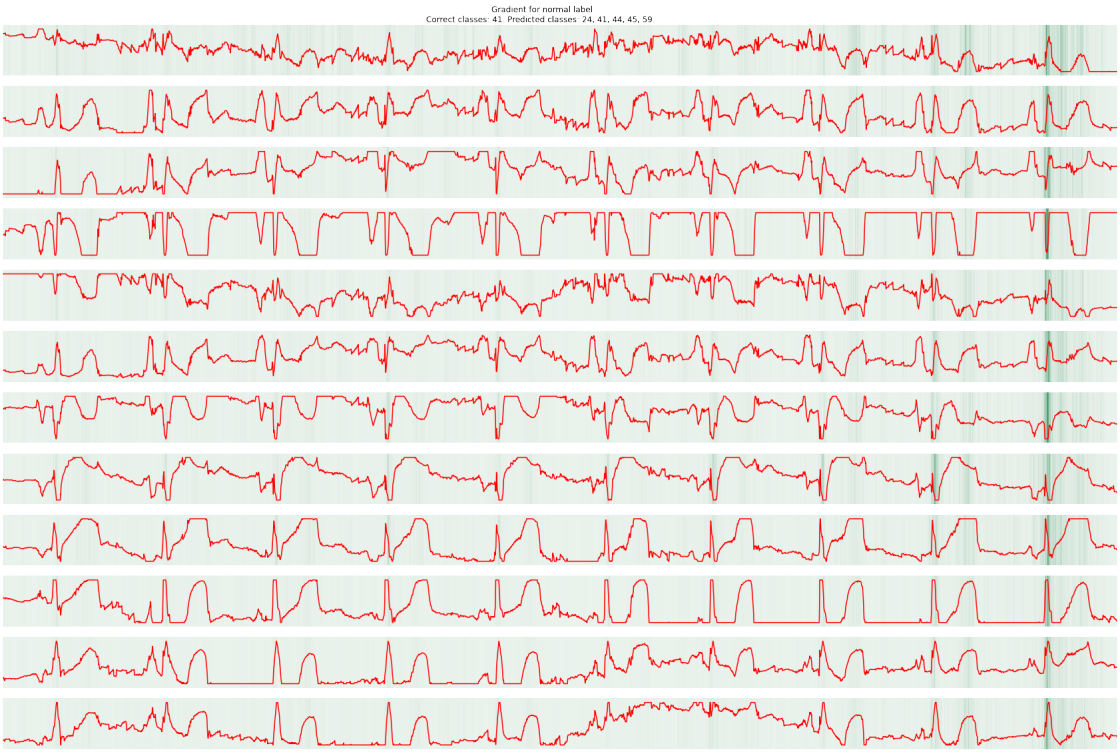
0.1 Normal

[33]:
```
def timeseries_to_image_with_gradient(data: torch.Tensor, labels:torch.Tensor,␣
↪gradient: torch.Tensor, pred:torch.Tensor=None, model_treshholds=None,␣
↪title=None, filenames :str=None, show=False, save=True):
    batches, width, height = data.shape
    for batch in range(batches):
        cmap_green = sns.light_palette("seagreen", as_cmap=True)
        cmap_both = sns.diverging_palette(10, 150, s=75, l=50, center='light',␣
↪as_cmap=True)
        fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
        plt.xlim((0, width))
        plt.ylim((0, 1))
        gradient = gradient[batch].abs()
        grad_norm = (gradient-gradient.min())/(gradient.max()-gradient.min())
        title = title or ""
        if not labels is None:
            title += "Correct classes: " + ", ".join(map(str, np.
↪nonzero(labels[batch].numpy())[0])) + ". "
        if not (pred is None or model_treshholds is None):
            binary_pred = pred[batch] >= model_treshholds
            title += "Predicted classes: " + ", ".join(map(str, np.
↪nonzero(binary_pred.numpy())[0])) + ". "
        fig.suptitle(title, y=0.9)

        for i, ax in enumerate(axs):
            ax.set_xlim((0, width))
            ax.set_ylim((-0.1, 1.1))
            ax.axis('off')
            d = data[batch, :, i]
            g = grad_norm[:, i:i+1].T
            ax.plot(range(width), d, color='red')
            ax.autoscale(False)
            ax.imshow(g, extent=[0, width, -0.1, 1.1], aspect='auto',␣
↪cmap=cmap_green)
        if save:
            plt.savefig(filenames[batch], dpi=fig.dpi)
        if show:
            plt.show()
        plt.close()
```

[7]:
```
```

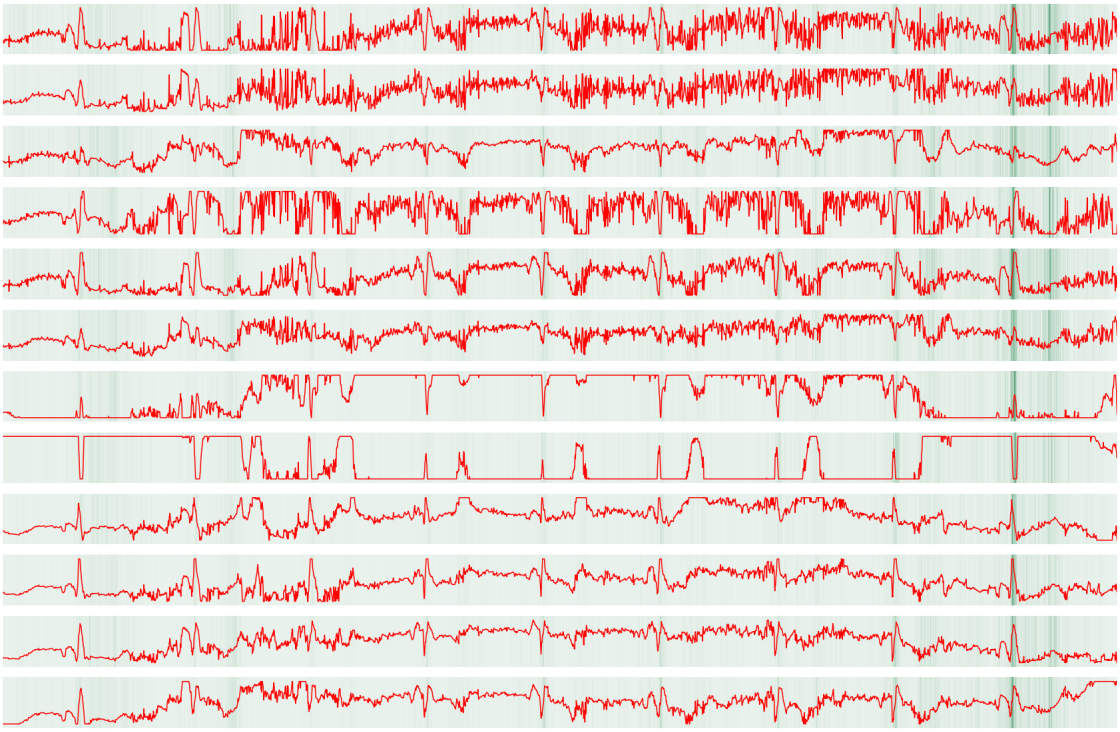
```
thresh_dict = {0: 0.0052883076, 1: 0.004021993, 2: 0.045075733, 3: 0.039263155,␣
↪4: 0.087553956, 5: 0.03217059, 6: 0.07084644, 7: 0.06982235, 8: 0.08900607,␣
↪9: 0.009527704, 10: 0.04808709, 11: 0.023643928, 12: 0.04898496, 13: 0.
↪014053739, 14: 0.11855617, 15: 0.0057171667, 16: 0.06450285, 17: 0.
↪014764133, 18: 0.0027846538, 19: 0.0014224607, 20: 0.0042695478, 21: 0.
↪0020088167, 22: 0.0056370394, 23: 0.023299428, 24: 0.00043616697, 25: 0.
↪0043225554, 26: 0.0006829281, 27: 0.003797319, 28: 0.0023218843, 29: 0.
↪0025232348, 30: 0.08151142, 31: 0.0057842294, 32: 0.055861708, 33: 0.
↪09779097, 34: 0.0037202945, 35: 0.013829965, 36: 0.025294341, 37: 0.
↪072333194, 38: 0.006157023, 39: 0.0070163156, 40: 0.0024529276, 41: 0.
↪396414, 42: 0.054770038, 43: 0.0033453542, 44: 0.027208127, 45: 0.
↪0036930004, 46: 0.095755436, 47: 0.05874352, 48: 0.037359316, 49: 0.
↪0045002145, 50: 0.0026231722, 51: 0.0012562033, 52: 0.0064664735, 53: 0.
↪0096522, 54: 0.011148318, 55: 0.07452798, 56: 0.028948307, 57: 0.005614491,␣
↪58: 0.006150292, 59: 0.0003258857, 60: 0.03152733, 61: 0.015745273, 62: 0.
↪0013439627, 63: 0.02767622, 64: 0.012850131, 65: 0.0044486015, 66: 0.
↪003912842}
thresholds = torch.Tensor(list(thresh_dict.values()))
```

[65]:
```
with open('/home/julian/Downloads/Github/contrastive-predictive-coding/
↪temp-grad.pickle', 'rb') as f:
    saved = pickle.load(f)
data, labels, gradient, pred = saved
filename = 'gradient-normal(correct:41).png'
timeseries_to_image_with_gradient(data, labels, gradient, pred,␣
↪model_treshholds=thresholds, title='Gradient for normal label\n',␣
↪filenames=[filename], save=True, show=True)
```

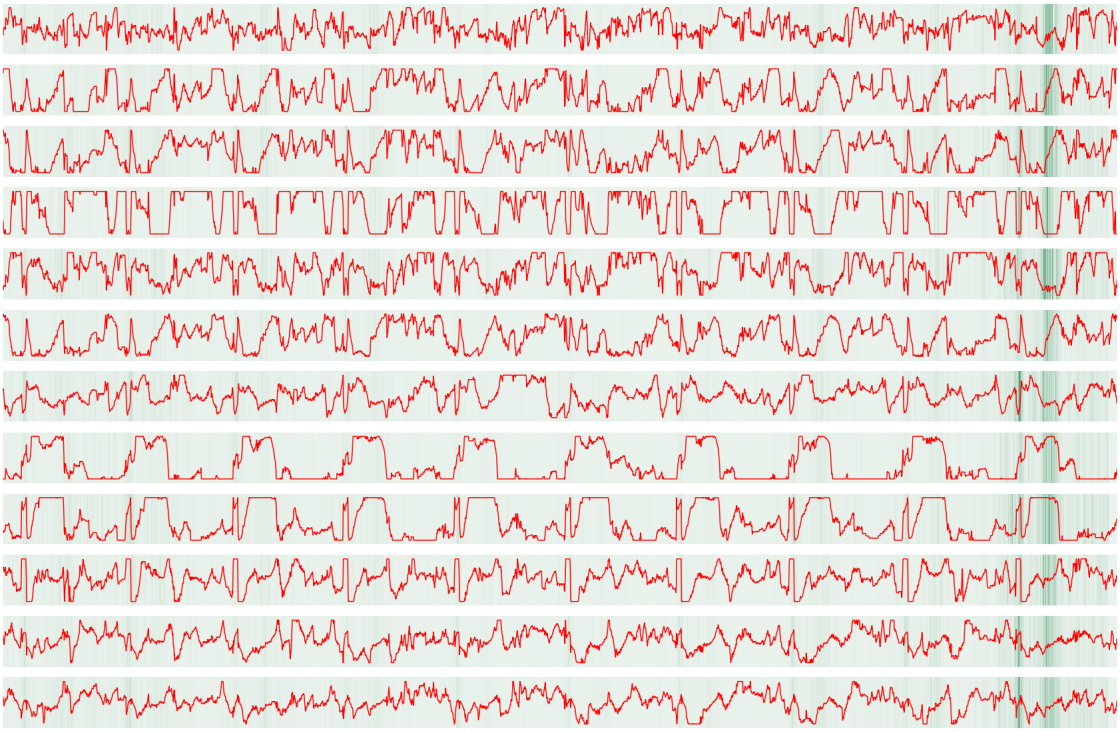


0.2 Inverse

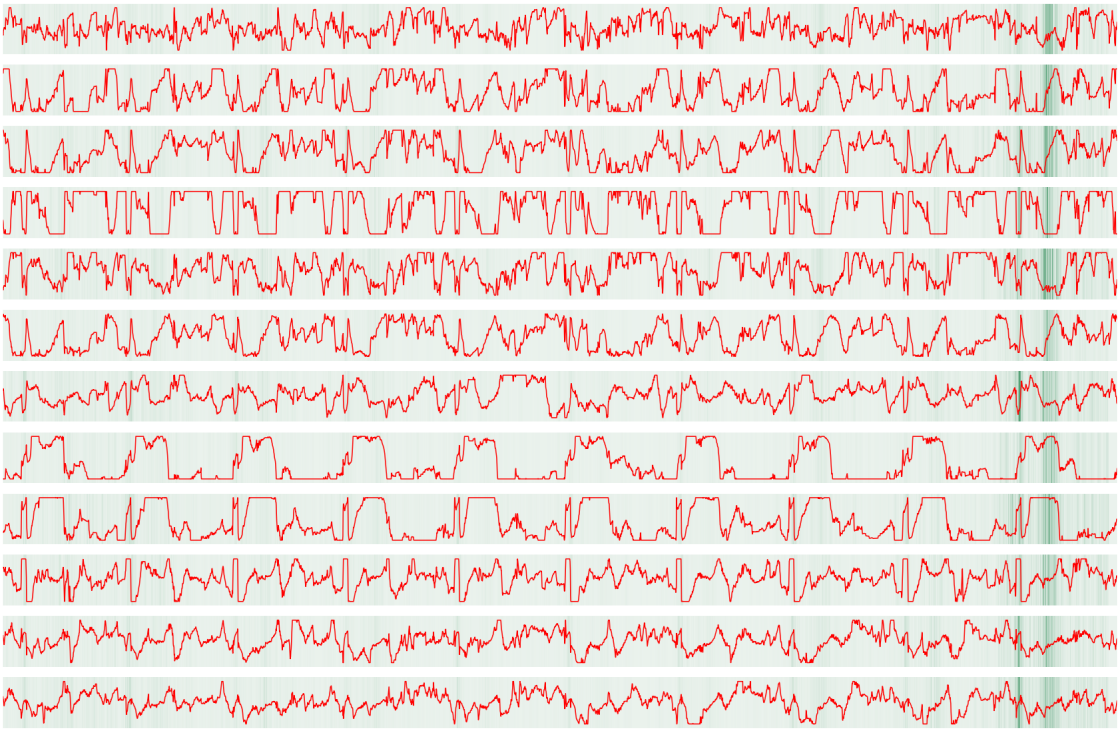
[29]:
```
with open('/home/julian/Downloads/Github/contrastive-predictive-coding/
 ↪temp-grad-inverse41.pickle', 'rb') as f:
    saved = pickle.load(f)
data, labels, gradient, pred = saved
_, width, height = data.shape
cmap_green = sns.light_palette("seagreen", as_cmap=True)
fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
plt.xlim((0, width))
plt.ylim((0, 1))
gradient = gradient.abs()
grad_norm = (gradient-gradient.min())/(gradient.max()-gradient.min())
for i, ax in enumerate(axs):
    ax.set_xlim((0, width))
    ax.set_ylim((-0.1, 1.1))
    ax.axis('off')
    d = data[0, :, i]
    g = grad_norm[0, :, i:i+1].T
    ax.plot(range(width), d, color='red')
    ax.autoscale(False)
    ax.imshow(g, extent=[0, width, -0.1, 1.1], aspect='auto', cmap=cmap_green)
```



[14]:
```
with open('/home/julian/Downloads/Github/contrastive-predictive-coding/
 ↪temp-grad14.pickle', 'rb') as f:
    saved = pickle.load(f)
data, labels, gradient, pred = saved
_, width, height = data.shape
cmap_green = sns.light_palette("seagreen", as_cmap=True)
fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
plt.xlim((0, width))
plt.ylim((0, 1))
gradient = gradient.abs()
grad_norm = (gradient-gradient.min())/(gradient.max()-gradient.min())
for i, ax in enumerate(axs):
    ax.set_xlim((0, width))
    ax.set_ylim((-0.1, 1.1))
    ax.axis('off')
    d = data[0, :, i]
    g = grad_norm[0, :, i:i+1].T
    ax.plot(range(width), d, color='red')
    ax.autoscale(False)
    ax.imshow(g, extent=[0, width, -0.1, 1.1], aspect='auto', cmap=cmap_green)
```

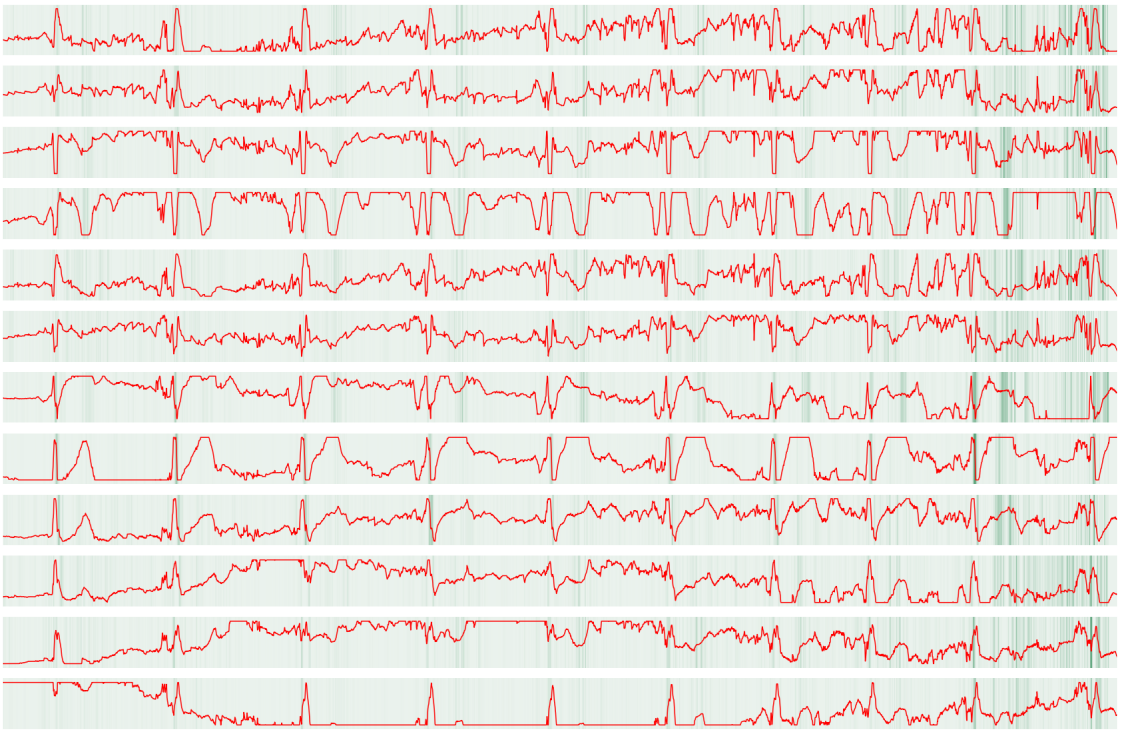


[15]:
```
with open('/home/julian/Downloads/Github/contrastive-predictive-coding/
 ↪temp-grad-inverse14.pickle', 'rb') as f:
    saved = pickle.load(f)
data, labels, gradient, pred = saved
_, width, height = data.shape
cmap_green = sns.light_palette("seagreen", as_cmap=True)
fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
plt.xlim((0, width))
plt.ylim((0, 1))
gradient = gradient.abs()
grad_norm = (gradient-gradient.min())/(gradient.max()-gradient.min())
for i, ax in enumerate(axs):
    ax.set_xlim((0, width))
    ax.set_ylim((-0.1, 1.1))
    ax.axis('off')
    d = data[0, :, i]
    g = grad_norm[0, :, i:i+1].T
    ax.plot(range(width), d, color='red')
    ax.autoscale(False)
    ax.imshow(g, extent=[0, width, -0.1, 1.1], aspect='auto', cmap=cmap_green)
```



[92]:
```
with open('/home/julian/Downloads/Github/contrastive-predictive-coding/
 ↪temp-grad41.pickle', 'rb') as f:
    saved = pickle.load(f)
data, labels, gradient, pred = saved
_, width, height = data.shape
cmap_green = sns.light_palette("seagreen", as_cmap=True)
fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
plt.xlim((0, width))
plt.ylim((0, 1))
gradient = np.clip(gradient, a_min=0, a_max=None)
grad_norm = (gradient-gradient.min())/(gradient.max()-gradient.min())
#fig.tight_layout()
for i, ax in enumerate(axs):
    ax.set_xlim((0, width))
    ax.set_ylim((-0.1, 1.1))
    ax.axis('off')
    d = data[0, :, i]
    g = grad_norm[0, :, i:i+1].T
    ax.plot(range(width), d, color='red')
    ax.autoscale(False)
    ax.imshow(g, extent=[0, width, -0.1, 1.1], aspect='auto', cmap=cmap_green)
```

Test



[120]:
```
for figi in range(67):
    with open(f'/home/julian/Downloads/Github/contrastive-predictive-coding/
↪temp-grad{figi}.pickle', 'rb') as f:
        saved = pickle.load(f)
    data, labels, gradient, pred = saved
    _, width, height = data.shape
    cmap_green = sns.light_palette("seagreen", as_cmap=True)
    fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
    plt.xlim((0, width))
    plt.ylim((0, 1))
    gradient = gradient.abs()
    grad_norm = (gradient-gradient.min())/(gradient.max()-gradient.min())
    for i, ax in enumerate(axs):
        ax.set_xlim((0, width))
        ax.set_ylim((-0.1, 1.1))
        ax.axis('off')
        d = data[0, :, i]
        g = grad_norm[0, :, i:i+1].T
        ax.plot(range(width), d, color='red')
        ax.autoscale(False)
```

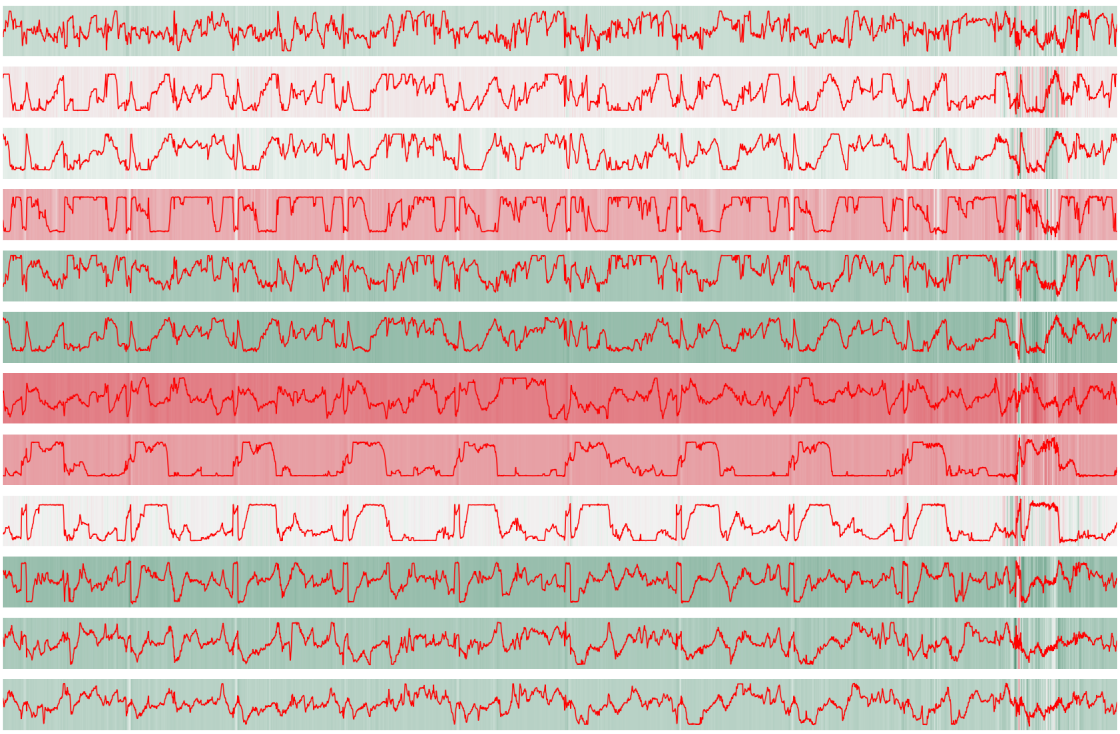
```
        ax.imshow(g, extent=[0, width, -0.1, 1.1], aspect='auto',␣
↪cmap=cmap_green)
    plt.savefig(f'gradient_visual-{figi}.png', dpi=fig.dpi)
    plt.close()
```

[119]:
```
for figi in range(67):
    with open(f'/home/julian/Downloads/Github/contrastive-predictive-coding/
↪temp-grad{figi}.pickle', 'rb') as f:
        saved = pickle.load(f)
    data, labels, gradient, pred = saved
    _, width, height = data.shape
    cmap_green = sns.light_palette("seagreen", as_cmap=True)
    fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
    plt.xlim((0, width))
    plt.ylim((0, 1))
    gradient = gradient
    grad_norm = (gradient-gradient.min())/(gradient.max()-gradient.min())
    for i, ax in enumerate(axs):
        ax.set_xlim((0, width))
        ax.set_ylim((-0.1, 1.1))
        ax.axis('off')
        d = data[0, :, i]
        g = grad_norm[0, :, i:i+1].T
        ax.plot(range(width), d, color='red')
        ax.autoscale(False)
        ax.imshow(g, extent=[0, width, -0.1, 1.1], aspect='auto',␣
↪cmap=cmap_both)
    plt.savefig(f'gradient_visual-noabs-{figi}.png', dpi=fig.dpi)
    plt.close()
```

0.3 Denoising data with gradient

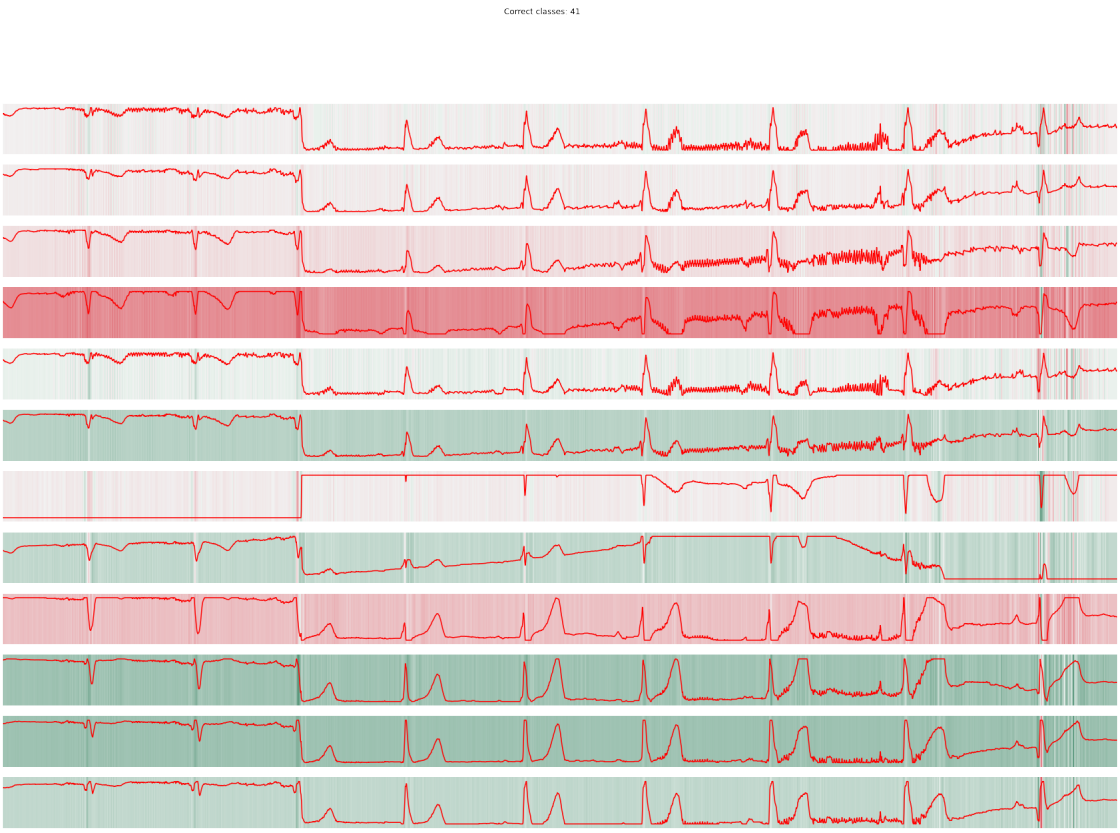
[27]:
```
with open(f'/home/julian/Downloads/Github/contrastive-predictive-coding/
↪temp-grad.pickle', 'rb') as f:
    saved = pickle.load(f)
data, labels, gradient, pred = saved
_, width, height = data.shape
cmap_green = sns.light_palette("seagreen", as_cmap=True)
fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
gradient = gradient
grad_norm = (gradient-gradient.min())/(gradient.max()-gradient.min())
for i, ax in enumerate(axs):
    ax.axis('off')
    d = data[0, :, i]
    g = grad_norm[0, :, i]
    da = d+g
```

```
    ax.set_xlim((0, width))
    ax.set_ylim((da.min()-0.1, da.max()+0.1))
    ax.plot(range(width), da, color='red')

    ax.imshow(grad_norm[0, :, i:i+1].T, extent=[0, width, da.min()-0.1, da.
 ↪max()+0.1], aspect='auto', cmap=cmap_both)
plt.show()
```



[57]:
```
with open(f'/home/julian/Downloads/Github/contrastive-predictive-coding/
 ↪temp-grad.pickle', 'rb') as f:
    saved = pickle.load(f)
data, labels, gradient, pred = saved
_, width, height = data.shape
cmap_green = sns.light_palette("seagreen", as_cmap=True)
fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
grad_abs = gradient.abs()
grad_norm = (grad_abs-grad_abs.min())/(grad_abs.max()-grad_abs.min())*torch.
 ↪sign(gradient)+0.5
fig.suptitle("Correct classes: " + ", ".join(map(str, np.nonzero(labels)[:, 1].
 ↪numpy())))
for i, ax in enumerate(axs):
    ax.axis('off')
    ax.set_xlim((0, width))
```

```
    ax.set_ylim((-0.1, 1.1))
    d = data[0, :, i]
    g = grad_norm[0, :, i]
    ax.plot(range(width), d, color='red')

    ax.imshow(grad_norm[0, :, i:i+1].T, extent=[0, width, -0.1, 1.1],␣
 ↪aspect='auto', cmap=cmap_both)
plt.show()
```



[109]:
```
str([s for s in np.nonzero(labels)[:, 1].numpy()])
```

[109]:
```
'[33, 41, 61]'
```

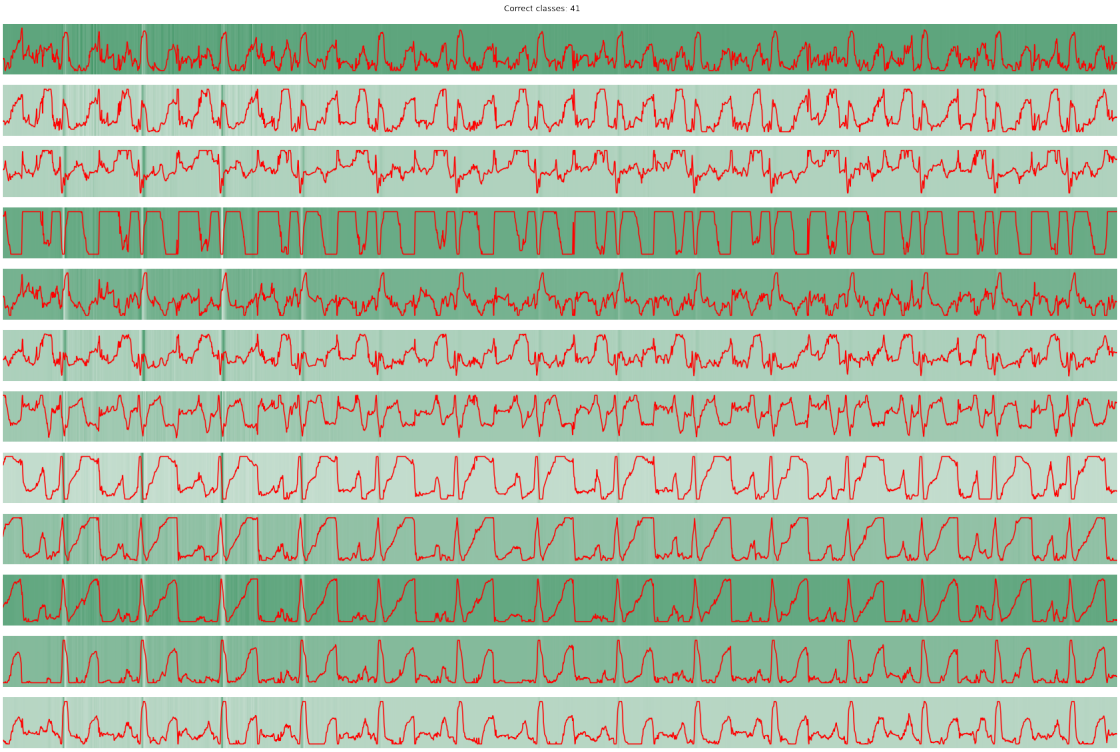
[117]:
```
"Correct classes: " + ", ".join(map(str, np.nonzero(labels)[:, 1].numpy()))
```

[117]:
```
'Correct classes: 33, 41, 61'
```

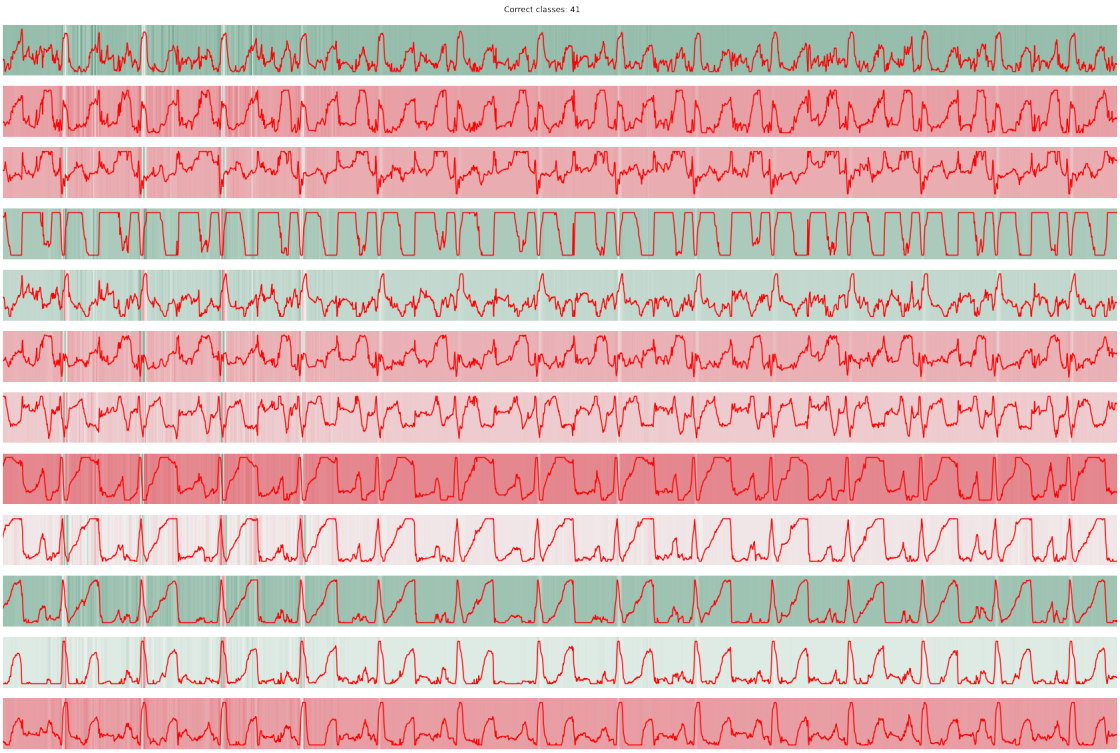
[35]:
```

```

[11]:
```
with open('/home/julian/Downloads/Github/contrastive-predictive-coding/
 ↪temp-grad41.pickle', 'rb') as f:
```

```
    saved = pickle.load(f)
data, labels, gradient, pred = saved
_, width, height = data.shape
cmap_green = sns.light_palette("seagreen", as_cmap=True)
fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
plt.xlim((0, width))
plt.ylim((0, 1))
gradient = gradient.abs()
grad_norm = (gradient-gradient.min())/(gradient.max()-gradient.min())
fig.suptitle("Correct classes: " + ", ".join(map(str, np.nonzero(labels)[:, 1].
 ↪numpy())), y=0.9)
for i, ax in enumerate(axs):
    ax.set_xlim((0, width))
    ax.set_ylim((-0.1, 1.1))
    ax.axis('off')
    d = data[0, :, i]
    g = grad_norm[0, :, i:i+1].T
    ax.plot(range(width), d, color='red')
    ax.autoscale(False)
    ax.imshow(g, extent=[0, width, -0.1, 1.1], aspect='auto', cmap=cmap_green)
plt.savefig('gradient-vis.png', dpi=fig.dpi)
```



[62]:
```
with open('/home/julian/Downloads/Github/contrastive-predictive-coding/
 ↪temp-grad41.pickle', 'rb') as f:
    saved = pickle.load(f)
data, labels, gradient, pred = saved
_, width, height = data.shape
cmap_green = sns.light_palette("seagreen", as_cmap=True)
fig, axs = plt.subplots(data.shape[-1], 1, figsize=(30, 20))
plt.xlim((0, width))
plt.ylim((0, 1))
sign = torch.sign(gradient)
#gradient = gradient.abs()
grad_norm = ((gradient-gradient.min())/(gradient.max()-gradient.min()))
fig.suptitle("Correct classes: " + ", ".join(map(str, np.nonzero(labels)[:, 1].
 ↪numpy())), y=0.9)
for i, ax in enumerate(axs):
    ax.set_xlim((0, width))
    ax.set_ylim((-0.1, 1.1))
    ax.axis('off')
    d = data[0, :, i]
    g = grad_norm[0, :, i:i+1].T
    ax.plot(range(width), d, color='red')
    ax.autoscale(False)
    ax.imshow(g, extent=[0, width, -0.1, 1.1], aspect='auto', cmap=cmap_both)
plt.savefig('gradient-vis.png', dpi=fig.dpi)
```



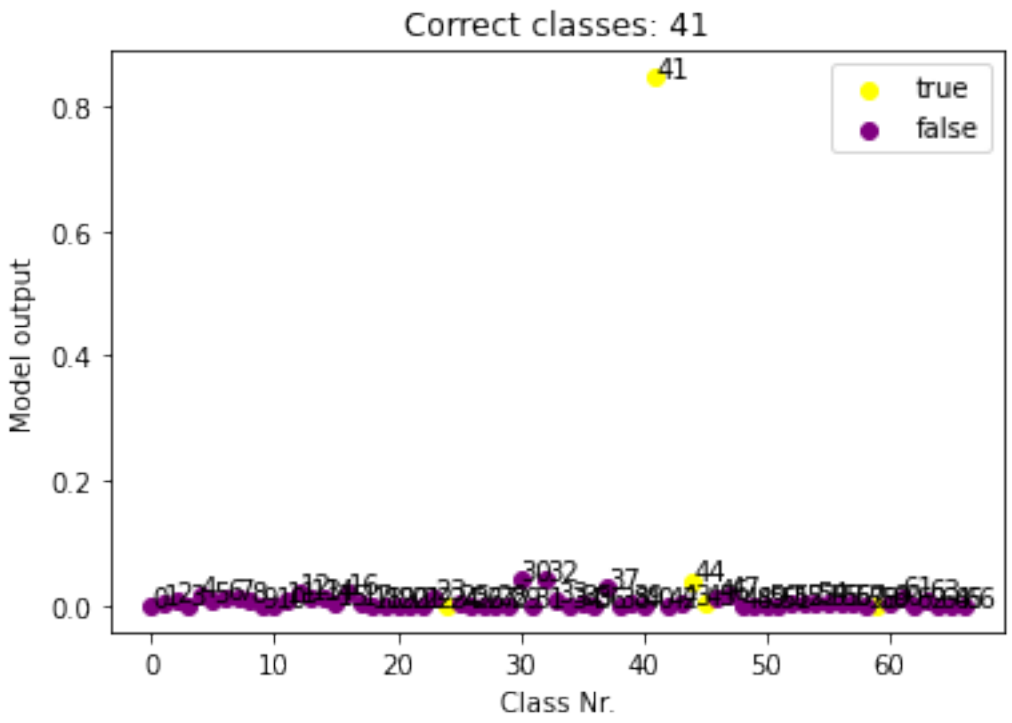
1 Prediction Visualization

[55]:
```
with open('/home/julian/Downloads/Github/contrastive-predictive-coding/
 ↪temp-grad.pickle', 'rb') as f:
    saved = pickle.load(f)
data, labels, gradient, pred = saved
```

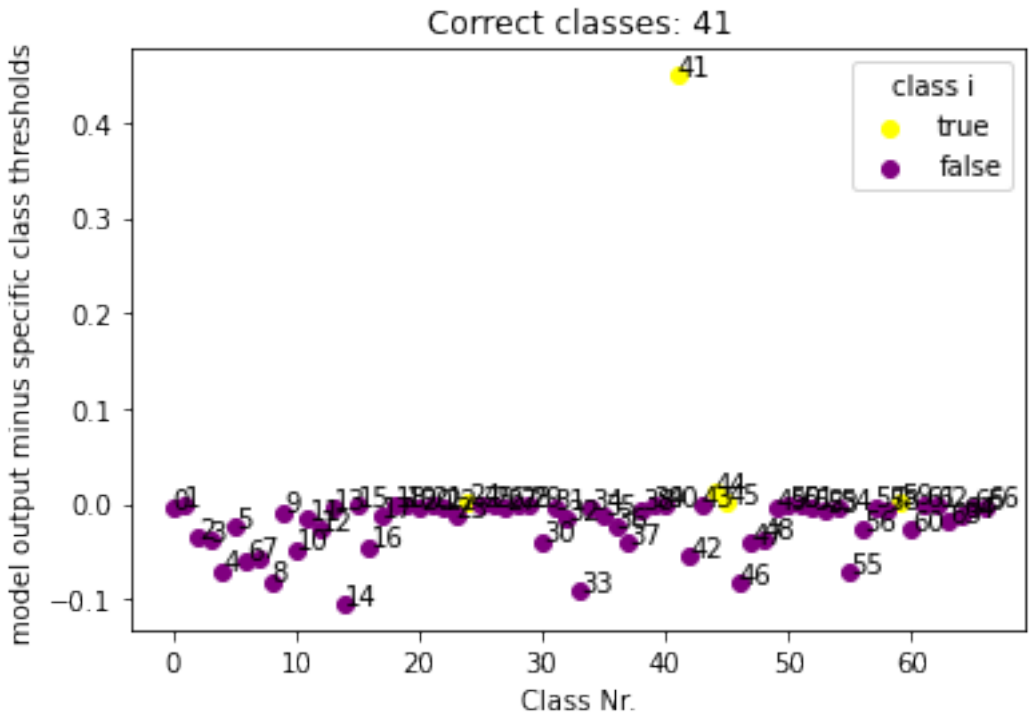
[56]:
```
threshs = {0: 0.0052883076, 1: 0.004021993, 2: 0.045075733, 3: 0.039263155, 4:␣
 ↪0.087553956, 5: 0.03217059, 6: 0.07084644, 7: 0.06982235, 8: 0.08900607, 9:␣
 ↪0.009527704, 10: 0.04808709, 11: 0.023643928, 12: 0.04898496, 13: 0.
 ↪014053739, 14: 0.11855617, 15: 0.0057171667, 16: 0.06450285, 17: 0.
 ↪014764133, 18: 0.0027846538, 19: 0.0014224607, 20: 0.0042695478, 21: 0.
 ↪0020088167, 22: 0.0056370394, 23: 0.023299428, 24: 0.00043616697, 25: 0.
 ↪0043225554, 26: 0.0006829281, 27: 0.003797319, 28: 0.0023218843, 29: 0.
 ↪0025232348, 30: 0.08151142, 31: 0.0057842294, 32: 0.055861708, 33: 0.
 ↪09779097, 34: 0.0037202945, 35: 0.013829965, 36: 0.025294341, 37: 0.
 ↪072333194, 38: 0.006157023, 39: 0.0070163156, 40: 0.0024529276, 41: 0.
 ↪396414, 42: 0.054770038, 43: 0.0033453542, 44: 0.027208127, 45: 0.
 ↪0036930004, 46: 0.095755436, 47: 0.05874352, 48: 0.037359316, 49: 0.
 ↪0045002145, 50: 0.0026231722, 51: 0.0012562033, 52: 0.0064664735, 53: 0.
 ↪0096522, 54: 0.011148318, 55: 0.07452798, 56: 0.028948307, 57: 0.005614491,␣
 ↪58: 0.006150292, 59: 0.0003258857, 60: 0.03152733, 61: 0.015745273, 62: 0.
 ↪0013439627, 63: 0.02767622, 64: 0.012850131, 65: 0.0044486015, 66: 0.
 ↪003912842}
binary_pred = pred[0]>torch.Tensor(list(threshs.values()))
```

[57]:
```
threshold_tensor = torch.Tensor(list(threshs.values()))
diff = (pred[0]-threshold_tensor).numpy()
diff
x = np.arange(len(threshold_tensor))
```

[58]:
```
plt.title("Correct classes: " + ", ".join(map(str, np.nonzero(labels)[:, 1].
 ↪numpy())))
plt.scatter(x[binary_pred], pred[0][binary_pred], c='yellow', label='true')
plt.scatter(x[~binary_pred], pred[0][~binary_pred], c='purple', label='false')

plt.xlabel('Class Nr.')
plt.ylabel('Model output')
for class_i in x:
    plt.annotate(str(class_i), (class_i, pred[0][class_i]))
plt.legend()
plt.savefig('scatterplot-prediction.png')
plt.show()
```

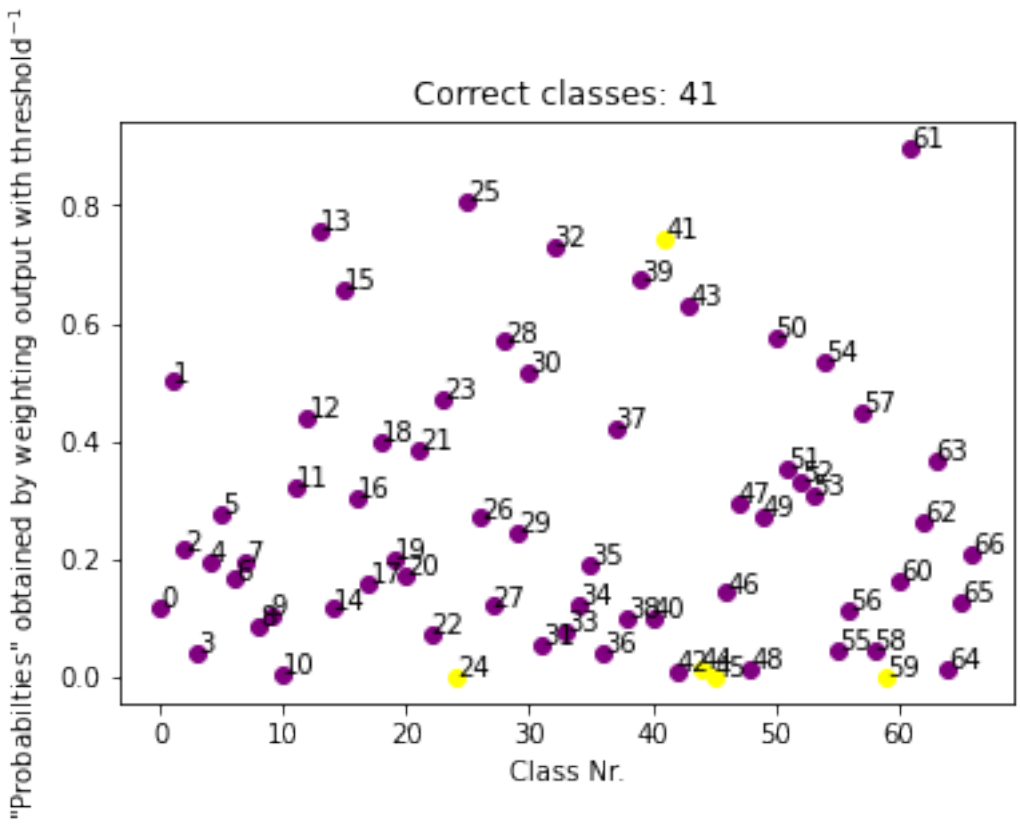


[59]:
```
plt.title("Correct classes: " + ", ".join(map(str, np.nonzero(labels)[:, 1].
 ↪numpy())))
plt.scatter(x[binary_pred], diff[binary_pred], c='yellow', label='true')
plt.scatter(x[~binary_pred], diff[~binary_pred], c='purple', label='false')

plt.xlabel('Class Nr.')
plt.ylabel('model output minus specific class thresholds')
for class_i in x:
    plt.annotate(str(class_i), (class_i, diff[class_i]))
plt.legend(title='class i')
plt.savefig('scatterplot-difference.png')
plt.show()
```



[64]:
```
normed_pos = diff/(1-threshold_tensor)
normed_neg = diff/threshold_tensor
probs = np.where(diff>=0, normed_pos, normed_neg)
```

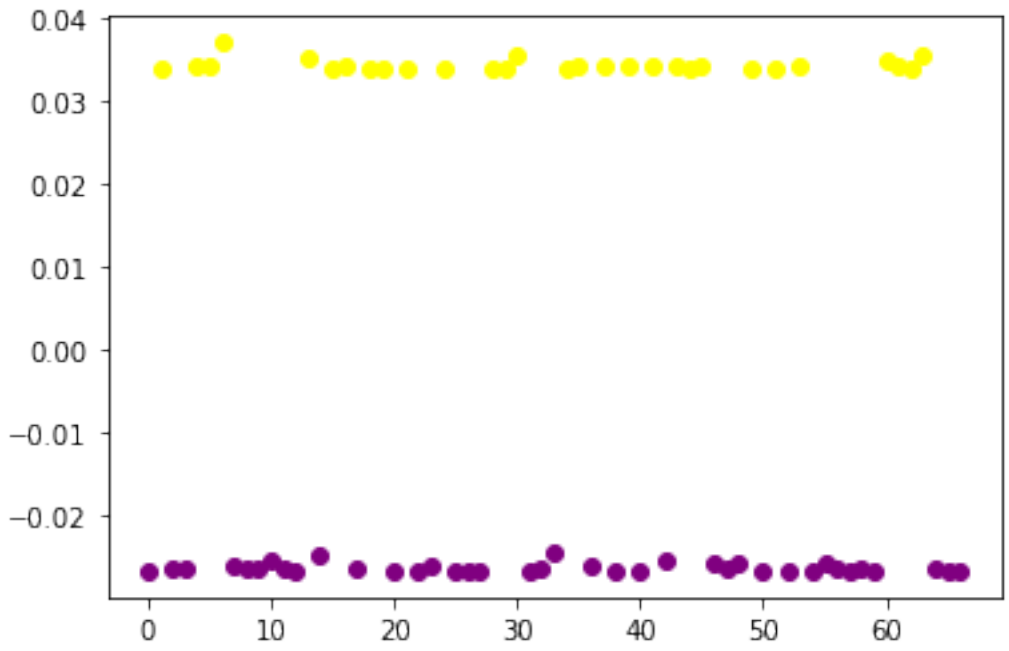
[65]:
```
plt.title("Correct classes: " + ", ".join(map(str, np.nonzero(labels)[:, 1].
 ↪numpy())))
plt.scatter(x[binary_pred], probs[binary_pred], c='yellow', label='true')
plt.scatter(x[~binary_pred], probs[~binary_pred], c='purple', label='false')

plt.xlabel('Class Nr.')
plt.ylabel(r'"Probabilties" obtained by weighting output with threshold$^{-1}$')
for class_i in x:
    plt.annotate(str(class_i), (class_i, probs[class_i]))
plt.savefig('scatterplot-probabilities.png')
plt.show()
```



[220]:
```
softp = torch.nn.Softmax(dim=0)(torch.Tensor(diff[binary_pred]))
softn = torch.nn.Softmax(dim=0)(torch.Tensor(diff[~binary_pred]))
```

[221]:
```
plt.scatter(x[binary_pred], softp, c='yellow')
plt.scatter(x[~binary_pred], -softn, c='purple')
```

[221]: <matplotlib.collections.PathCollection at 0x7fb2e3cd66d0>



[8]:
```
def plot_prediction_scatterplots(labels, pred, model_thresholds, filename=None):
    binary_pred = pred[0] >= model_thresholds
    diff = (pred[0] - model_thresholds).numpy()
    x = np.arange(len(model_thresholds))
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))
    title = "Correct classes: " + ", ".join(map(str, np.nonzero(labels)[:, 1].
↪numpy()))
    title += "\n Predicted classes (binary): " + ", ".join(map(str, np.
↪nonzero(binary_pred.numpy())[0]))
    fig.suptitle(title)
    ##First plot
    ax1.set_aspect='equal'
    ax1.scatter(x[binary_pred], pred[0][binary_pred], c='yellow', label='true')
    ax1.scatter(x[~binary_pred], pred[0][~binary_pred], c='purple',␣
↪label='false')
    ax1.set_xlabel('Class Nr.')

    ax1.set_ylabel('Model output')
    for class_i in x:
        ax1.annotate(str(class_i), (class_i, pred[0][class_i]))
    ax1.legend()
    ##Second Plot
    ax2.set_aspect='equal'
    ax2.scatter(x[binary_pred], diff[binary_pred], c='yellow', label='true')
    ax2.scatter(x[~binary_pred], diff[~binary_pred], c='purple', label='false')
```

```
    ax2.set_xlabel('Class Nr.')
    ax2.set_ylabel('model output minus specific class thresholds')
    for class_i in x:
        ax2.annotate(str(class_i), (class_i, diff[class_i]))
    ax2.legend()
    ##Third Plot
    normed_pos = diff/(1-model_thresholds)
    normed_neg = -pred/model_thresholds
    probs = np.where(diff>=0, normed_pos, normed_neg)
    ax3.scatter(x[binary_pred], probs[binary_pred], c='yellow', label='true')
    ax3.scatter(x[~binary_pred], probs[~binary_pred], c='purple', label='false')

    ax3.set_aspect='equal'
    ax3.set_xlabel('Class Nr.')
    ax3.set_ylabel(r'"Probabilties" obtained by weighting output with␣
↪threshold$^{-1}$')
    for class_i in x:
        ax3.annotate(str(class_i), (class_i, probs[class_i]))
    ax3.legend()
    filename = filename or ""
    fig.tight_layout()
    plt.savefig(filename + 'scatterplot-predictions(combined).png', dpi=fig.dpi)
    plt.show()
```

[9]:
```
plot_prediction_scatterplots(labels, pred, thresholds, filename=None)
```

```
      ␣
↪---------------------------------------------------------------------------

       NameError                                 Traceback (most recent call␣
↪last)

       <ipython-input-9-2b9425ff0e2b> in <module>
   ----> 1 plot_prediction_scatterplots(labels, pred, thresholds, filename=None)


       NameError: name 'labels' is not defined
```

2 Average prediction probs

[5]:
```
import glob
import os
import pandas as pd
def read_label_csv_from_model_folder(model_folder, data_loader_index=0):
```

```
    label_path = glob.glob(os.path.join(model_folder,␣
↪f"labels-dataloader-{data_loader_index}.csv"))
    if len(label_path) == 0:
        raise FileNotFoundError
    df1 = pd.read_csv(label_path[0])
    labels = df1.values[:, 1:].astype(float)
    return labels, df1.columns[1:].values

def read_binary_label_csv_from_model_folder(model_folder, data_loader_index=0):
    l, c = read_label_csv_from_model_folder(model_folder, data_loader_index)
    l = (l > 0).astype(int)
    return l, c

def read_output_csv_from_model_folder(model_folder, data_loader_index=0):
    pred_path = glob.glob(os.path.join(model_folder,␣
↪f"model-*-dataloader-{data_loader_index}-output.csv"))
    if len(pred_path) == 0:
        raise FileNotFoundError
    dfp = pd.read_csv(pred_path[0])
    return dfp.values[:, 1:].astype(float), dfp.columns[1:].values #1 is file
```

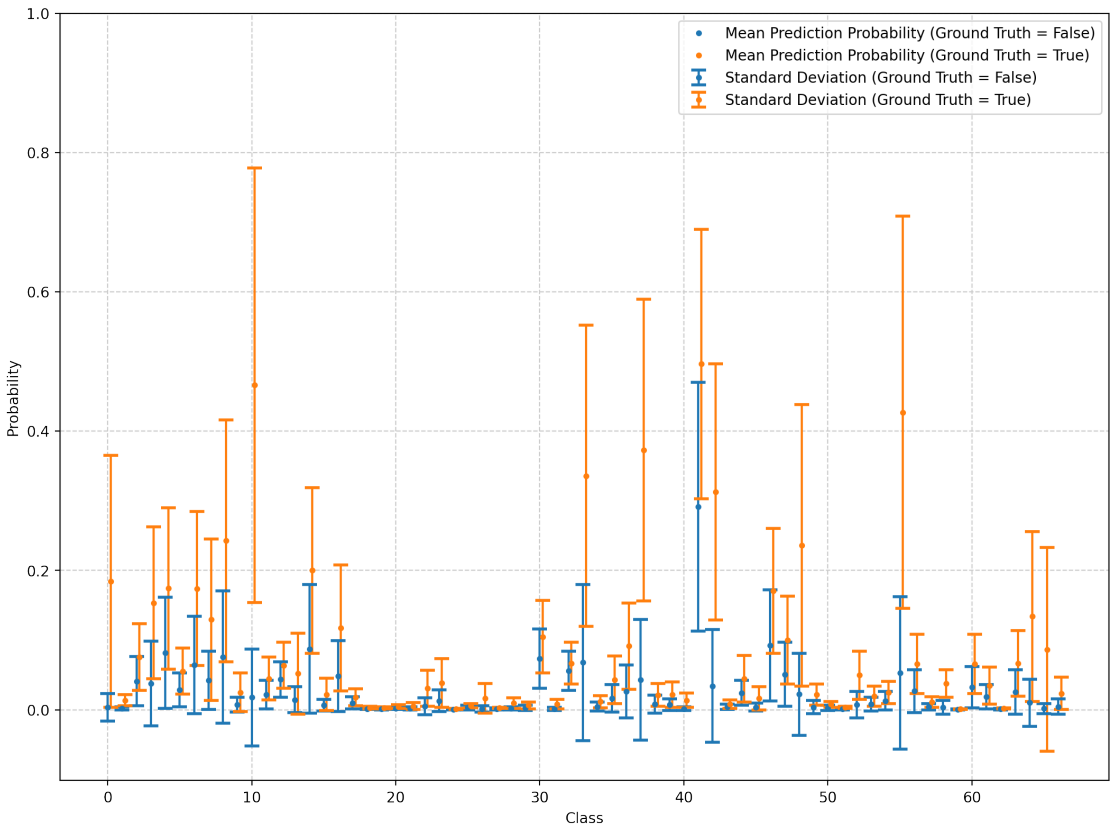
[7]:
```
def plot_errorbar_minmax(meanx, minx, maxx):
    plt.figure(figsize=(20, 15), dpi=200)
    x = np.arange(0, len(meanx))
    (dots, caps, _) = plt.errorbar(x, meanx, yerr=[meanx-minx, maxx-meanx],␣
↪fmt='k.', capsize=5)
    for cap in caps:
        cap.set_color('red')
        cap.set_markeredgewidth(2)
    plt.show()

def plot_errorbar_std(meanxi, stdxi, idx_order, group_names=None, title='',␣
↪save_path=None):
    if not (type(meanxi) == list and type(stdxi) == list):
        meanxi, stdxi = [meanxi], [stdxi]
    fig, ax = plt.subplots(dpi=200, figsize=[6.4*2, 4.8*2])
    ax.set_xlabel('Class')
    ax.set_ylabel('Probability')
    if not save_path:
        ax.set_title(title)
    first = True
    for i in range(len(meanxi)):
        line_label = "Standard Deviation"
        if group_names:
            line_label += f" ({group_names[i]})"
        meanx, stdx = meanxi[i], stdxi[i]
        if idx_order is None:
```

```
            idx_order = np.arange(len(meanx))
        if type(meanx) == list:
            meanx = [m[idx_order] for m in meanx]
        else:
            meanx = meanx[idx_order]
        if type(stdx) == list:
            stdx = [s[idx_order] for s in stdx]
        else:
            stdx = stdx[idx_order]
        x = np.arange(len(meanx))+0.2*i
        (dots, caps, error) = ax.errorbar(x, meanx, yerr=stdx, fmt='.',␣
↪capsize=5, label=line_label)
        dots.set_label(f"Mean Prediction Probability ({group_names[i] if␣
↪type(group_names)==list else group_names})")
        for cap in caps:
            cap.set_markeredgewidth(2)
        #dots.set_markersize(10)
    ax.set_ylim(top=1.)
    if not all(idx_order==np.arange(len(idx_order))):
        ax.set_xticks(np.arange(len(idx_order)))
        ax.set_xticklabels(idx_order, rotation='vertical')
    ax.legend(loc='upper right')
    ax.grid(which='major', color='#CCCCCC', linestyle='--')
    ax.grid(which='minor', color='#CCCCCC', linestyle=':')
    if save_path:
        fig.savefig(save_path, dpi=fig.dpi)
def calculate_plot_with_thresholds(labels, preds, thresholds, model_name,␣
↪title='Prediction Probabilities for each class with Mean and Standard␣
↪Deviation', save_path=None, sort_fn=None, std_two_directional=False):
    diff = (preds - thresholds)
    transformed_probs = (np.where(diff<0, diff/thresholds, diff/
↪(1-thresholds))+1.)/2.
    calculate_plot(labels, transformed_probs, model_name, title, save_path,␣
↪sort_fn, std_two_directional)

def calculate_plot(labels, preds, model_name, title='Prediction Probabilities␣
↪for each class with Mean and Standard Deviation', save_path=None,␣
↪sort_fn=None, std_two_directional=False):
    title = model_name + "\n" + title
    sel0 = np.where(labels==0, preds, np.nan)
    mi0 = np.nanmin(sel0, axis=0)
    ma0 = np.nanmax(sel0, axis=0)
    mean0 = np.nanmean(sel0, axis=0)
    std0 = np.nanstd(sel0, axis=0)

    sel1 = np.where(labels!=0, preds, np.nan)
```

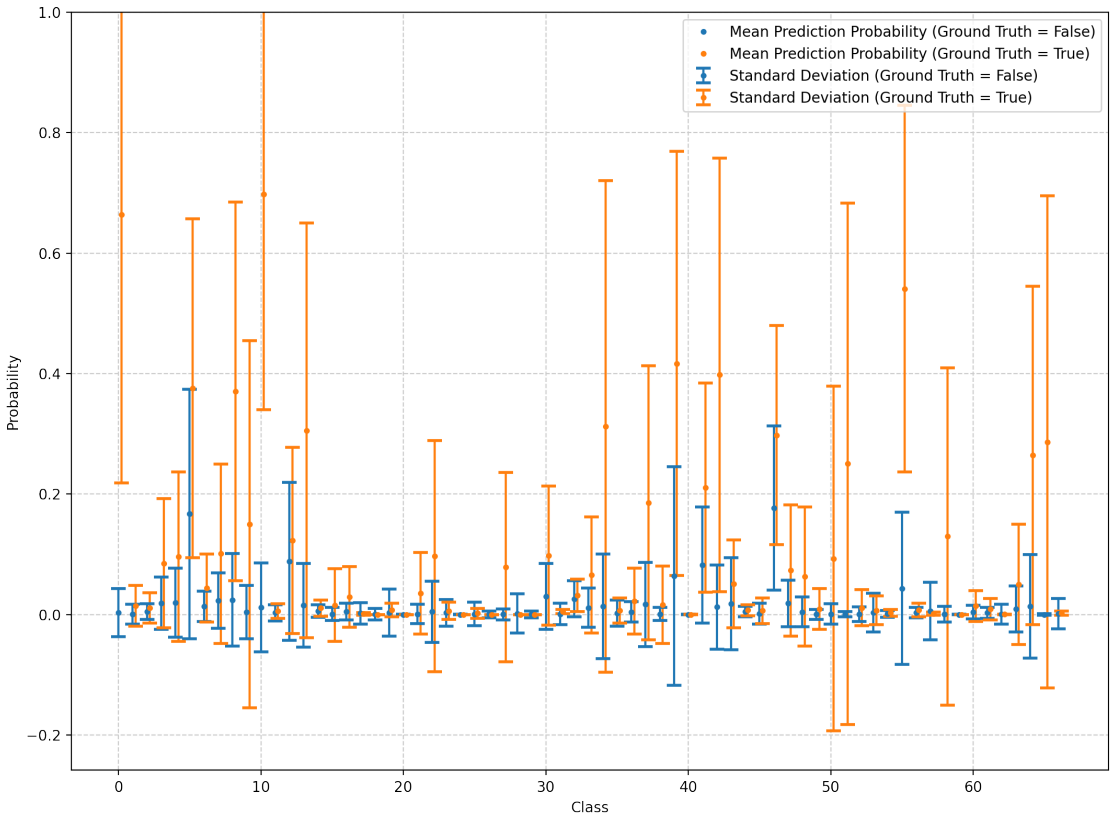
```
    mi1 = np.nanmin(sel1, axis=0)
    ma1 = np.nanmax(sel1, axis=0)
    mean1 = np.nanmean(sel1, axis=0)
    std1 = np.nanstd(sel1, axis=0)
    if std_two_directional:
        diff0 = sel0-mean0
        std0 = np.sqrt(np.nanmean(np.where(diff0>=0, diff0, np.nan)**2,␣
↪axis=0))#np.nanstd(np.where(diff0>=0, sel, np.nan), axis=0)
        stdn0 = np.sqrt(np.nanmean(np.where(diff0<=0, diff0, np.nan)**2,␣
↪axis=0))#np.nanstd(np.where(diff0<=0, sel, np.nan), axis=0)
        diff1 = sel1-mean1
        stdp1 = np.sqrt(np.nanmean(np.where(diff1>=0, diff1, np.nan)**2,␣
↪axis=0))#np.nanstd(np.where(diff1>=0, sel, np.nan), axis=0)
        std1 = np.sqrt(np.nanmean(np.where(diff1<=0, diff1, np.nan)**2,␣
↪axis=0))#np.nanstd(np.where(diff1<=0, sel, np.nan), axis=0)

    idx_order = np.arange(labels.shape[1])
    if not sort_fn is None:
        idx_order = sort_fn(sel0, mi0, ma0, mean0, std0, sel1, mi1, ma1, mean1,␣
↪std1)
        print(idx_order)
    if std_two_directional:
        plot_errorbar_std([mean0, mean1], [[stdn0, std0], [std1, stdp1]],␣
↪group_names=['Ground Truth = False', 'Ground Truth = True'], title=title,␣
↪save_path=save_path, idx_order=idx_order)
    else:
        plot_errorbar_std([mean0, mean1], [std0, std1], group_names=['Ground␣
↪Truth = False', 'Ground Truth = True'], title=title, save_path=save_path,␣
↪idx_order=idx_order)

def sort_by_std_gap(sel0, mi0, ma0, mean0, std0, sel1, mi1, ma1, mean1, std1):
    return np.argsort((mean0+std0) - (mean1-std1))

def sort_by_mean_gap(sel0, mi0, ma0, mean0, std0, sel1, mi1, ma1, mean1, std1):
    return np.argsort(mean0 - mean1)

def sort_by_group_count(sel0, mi0, ma0, mean0, std0, sel1, mi1, ma1, mean1,␣
↪std1):
    return np.argsort(np.count_nonzero(~np.isnan(sel0), axis=0)+np.
↪count_nonzero(~np.isnan(sel1), axis=0))
```

2.1 (no model threshold reweighting)

[8]:
```
model_test_folder = '/home/julian/Downloads/Github/
 ↪contrastive-predictive-coding/models/16_08_21-10-16-test|(40x)cpc/
 ↪14_08_21-15_36-train|(4x)cpc/architectures_cpc.cpc_combined.
 ↪CPCCombined0|train-test-splits-fewer-labels60|use_weights|frozen|C|m:
 ↪all|cpc_downstream_cnn'
#'/home/julian/Downloads/Github/contrastive-predictive-coding/models/
 ↪26_06_21-15-test|(2x)bl_MLP+bl_FCN+bl_TCN_block+bl_TCN_down+bl_TCN_flatten+bl_TCN_last+bl_a
 ↪21_05_21-11-train|bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+bl_cnn_v14+bl_cnn
 ↪architectures_baseline_challenge.baseline_cnn_v14.BaselineNet12|dte:120'
test_labels, classes =␣
 ↪read_binary_label_csv_from_model_folder(model_test_folder, 0)
test_pred, _ = read_output_csv_from_model_folder(model_test_folder, 0)
val_labels, classes =␣
 ↪read_binary_label_csv_from_model_folder(model_test_folder, 1)
val_pred, _ = read_output_csv_from_model_folder(model_test_folder, 1)
save_path='/home/julian/Documents/projekt-master/bilder/
 ↪errorplot-prediction-cpc.png'
calculate_plot(test_labels, test_pred, model_name='CPC', save_path=save_path)
```



[9]:
```
# model_test_folder = '/home/julian/Downloads/Github/
 ↪contrastive-predictive-coding/models/16_08_21-10-16-test|(40x)cpc/
 ↪14_08_21-15_36-train|(4x)cpc/architectures_cpc.cpc_combined.
 ↪CPCCombined0|train-test-splits-fewer-labels60|use_weights|frozen|C|m:
 ↪all|cpc_downstream_cnn'
model_test_folder = '/home/julian/Downloads/Github/
 ↪contrastive-predictive-coding/models/
 ↪11_08_21-15-58-test|(10x)bl_TCN_down+(10x)bl_cnn_v1+(10x)bl_cnn_v14+(10x)bl_cnn_v15+(10x)bl
 ↪10_08_21-14_42-train|bl_TCN_down+bl_cnn_v1+bl_cnn_v14+bl_cnn_v15+bl_cnn_v8/
 ↪architectures_baseline_challenge.baseline_cnn_v15.
 ↪BaselineNet4|train-test-splits-fewer-labels60|use_weights|ConvLyrs:
 ↪5|MaxPool|Linear|BatchNorm|stride_sum:14|dilation_sum:96|padding_sum:
 ↪0|krnls_sum:26'
test_labels, classes =␣
 ↪read_binary_label_csv_from_model_folder(model_test_folder, 0)
test_pred, _ = read_output_csv_from_model_folder(model_test_folder, 0)
val_labels, classes =␣
 ↪read_binary_label_csv_from_model_folder(model_test_folder, 1)
val_pred, _ = read_output_csv_from_model_folder(model_test_folder, 1)
save_path='/home/julian/Documents/projekt-master/bilder/
 ↪errorplot-prediction-blv15.png'
calculate_plot(test_labels, test_pred, model_name='BL_v15', save_path=save_path)
```

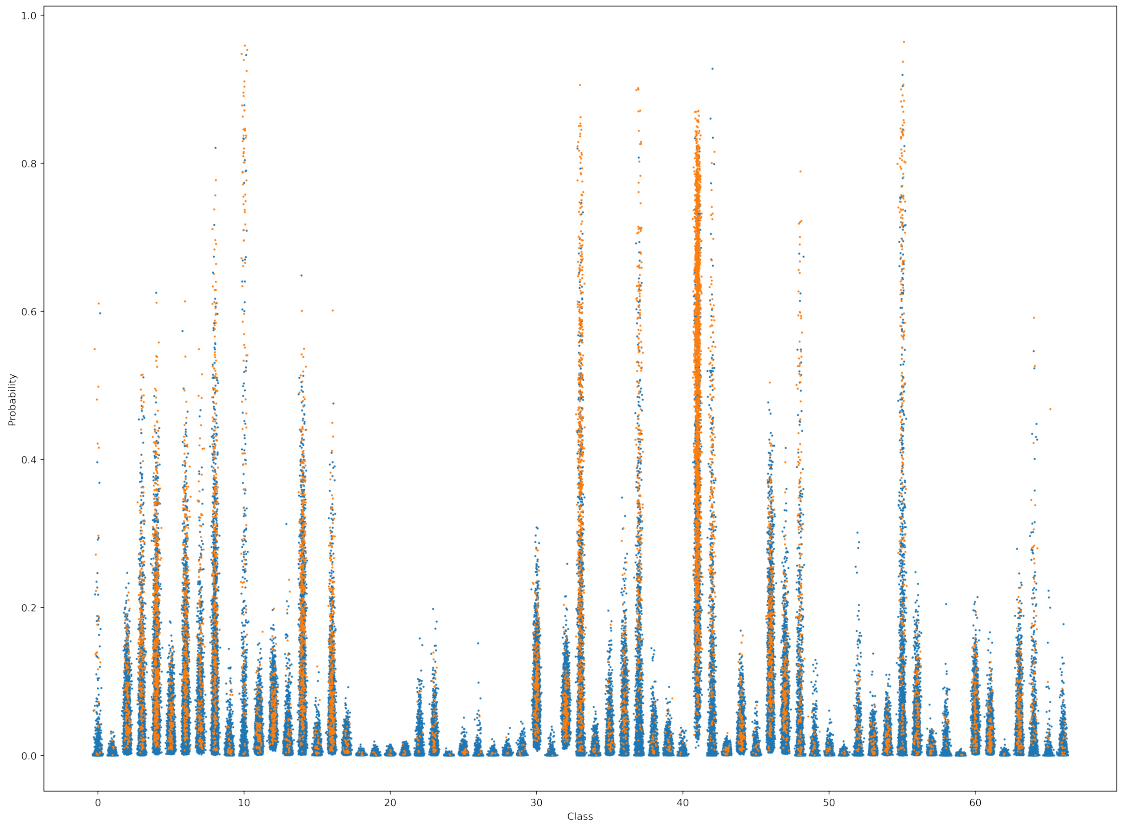


2.2 with threshold reweighting

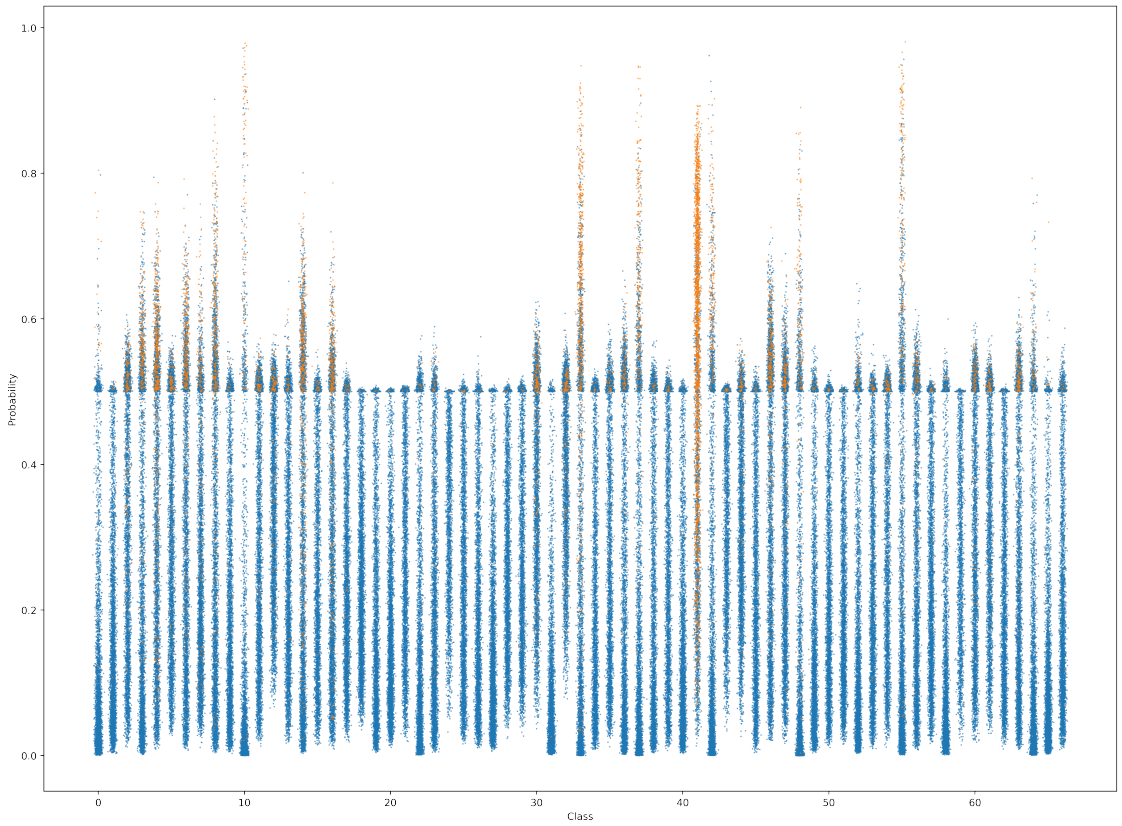
2.2.1 CPC

[10]:
```
model_test_folder = '/home/julian/Downloads/Github/
 ↪contrastive-predictive-coding/models/16_08_21-10-16-test|(40x)cpc/
 ↪14_08_21-15_36-train|(4x)cpc/architectures_cpc.cpc_combined.
 ↪CPCCombined0|train-test-splits-fewer-labels60|use_weights|frozen|C|m:
 ↪all|cpc_downstream_cnn'
#model_test_folder = '/home/julian/Downloads/Github/
 ↪contrastive-predictive-coding/models/
 ↪26_06_21-15-test|(2x)bl_MLP+bl_FCN+bl_TCN_block+bl_TCN_down+bl_TCN_flatten+bl_TCN_last+bl_a
 ↪21_05_21-11-train|bl_cnn_v0+bl_cnn_v0_1+bl_cnn_v0_2+bl_cnn_v0_3+bl_cnn_v1+bl_cnn_v14+bl_cnn
 ↪architectures_baseline_challenge.baseline_cnn_v14.BaselineNet12|dte:120'
test_labels, classes =␣
 ↪read_binary_label_csv_from_model_folder(model_test_folder, 0)
test_pred, _ = read_output_csv_from_model_folder(model_test_folder, 0)
val_labels, classes =␣
 ↪read_binary_label_csv_from_model_folder(model_test_folder, 1)
val_pred, _ = read_output_csv_from_model_folder(model_test_folder, 1)
#for now use ready file:
#threshold_file = '/home/julian/Downloads/Github/contrastive-predictive-coding/
 ↪images/explain/BaselineNet12/thresholds.txt'
threshold_file = '/home/julian/Downloads/Github/contrastive-predictive-coding/
 ↪images/explain/CPCCombined0/thresholds.txt'
with open(threshold_file, 'r') as f:
    data = f.read()
thresholds = np.array(list(map(lambda x: x.item(), map(eval, data.
 ↪split(',')))))  #'slighlty' dangerous
```

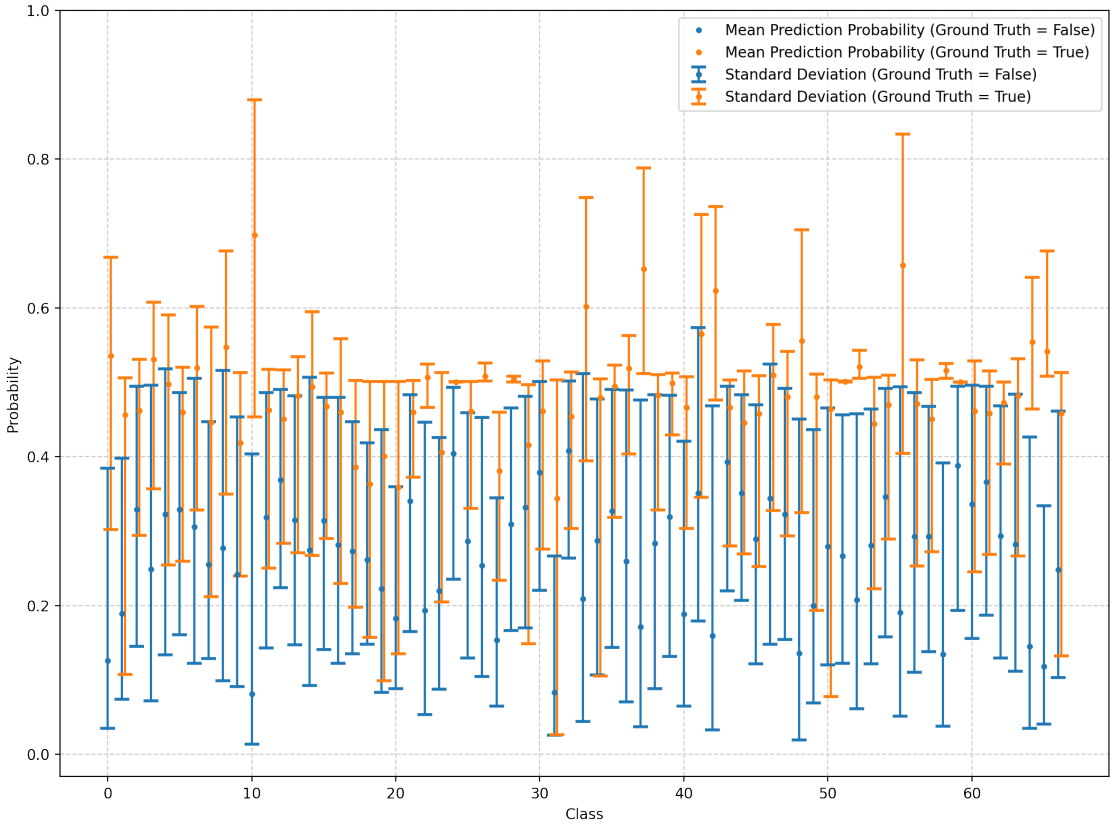
[11]:
```
pos = np.where(test_labels==0, test_pred, np.nan)
neg = np.where(test_labels!=0, test_pred, np.nan)
x1 = np.random.normal(scale=0.1, size=pos.shape)+np.arange(67)[np.newaxis, :]
plt.figure(figsize=[6.4*3, 4.8*3], dpi=400)
plt.scatter(x1, pos, s=1)
x2 = np.random.normal(scale=0.1, size=neg.shape)+np.arange(67)[np.newaxis, :]
plt.scatter(x2, neg, s=1)
plt.xlabel('Class')
plt.ylabel("Probability")
plt.savefig('/home/julian/Documents/projekt-master/bilder/
 ↪errorplot-scattered-cpc.png', dpi=400)

plt.show()
```



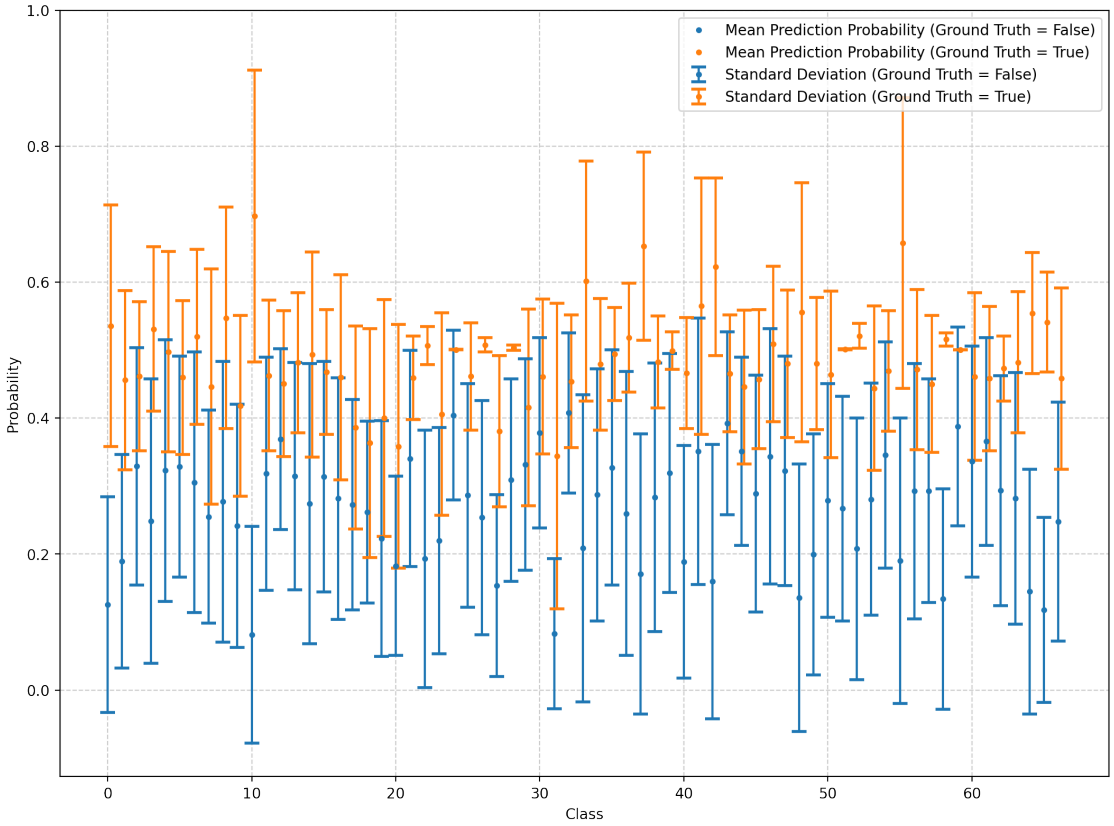
[12]:
```
diff = (test_pred - thresholds)
transformed_probs = (np.where(diff<0, diff/thresholds, diff/(1-thresholds))+1.)/
 ↪2.
pos = np.where(test_labels==0, transformed_probs, np.nan)
neg = np.where(test_labels!=0, transformed_probs, np.nan)
x1 = np.random.normal(scale=0.1, size=pos.shape)+np.arange(67)[np.newaxis, :]
plt.figure(figsize=[6.4*3, 4.8*3], dpi=400)
plt.scatter(x1, pos, s=0.1)
x2 = np.random.normal(scale=0.1, size=neg.shape)+np.arange(67)[np.newaxis, :]
plt.scatter(x2, neg, s=0.1)
plt.xlabel('Class')
plt.ylabel("Probability")
plt.savefig('/home/julian/Documents/projekt-master/bilder/
 ↪errorplot-scattered-thresholds-cpc.png', dpi=400)
plt.show()
```



[13]:
```
calculate_plot_with_thresholds(test_labels, test_pred, thresholds,␣
↪model_name='CPC',
                               save_path='/home/julian/Documents/projekt-master/
↪bilder/errorplot-prediction-threshold-cpc-twodirectional.png',
                               sort_fn=None, std_two_directional=True)
```

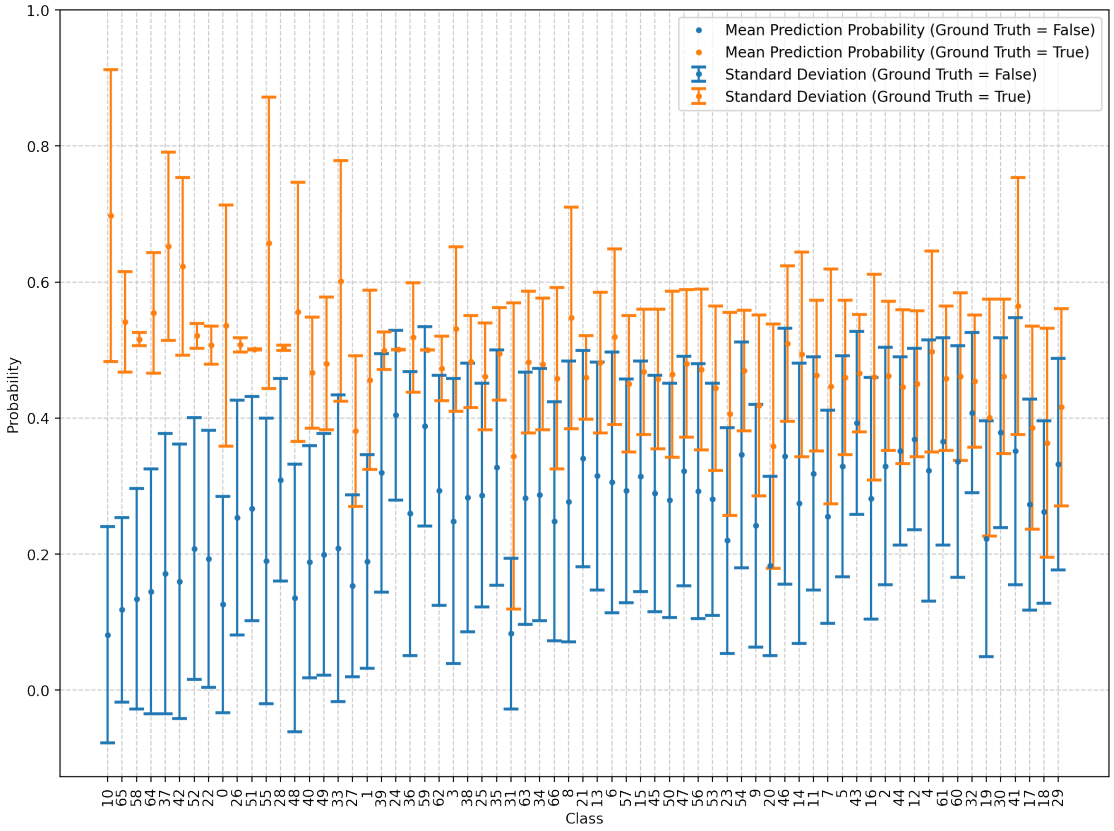


[14]:
```
calculate_plot_with_thresholds(test_labels, test_pred, thresholds,␣
↪model_name='CPC',
                               save_path='/home/julian/Documents/projekt-master/
↪bilder/errorplot-prediction-threshold-cpc.png',
                               sort_fn=None, std_two_directional=False)
```



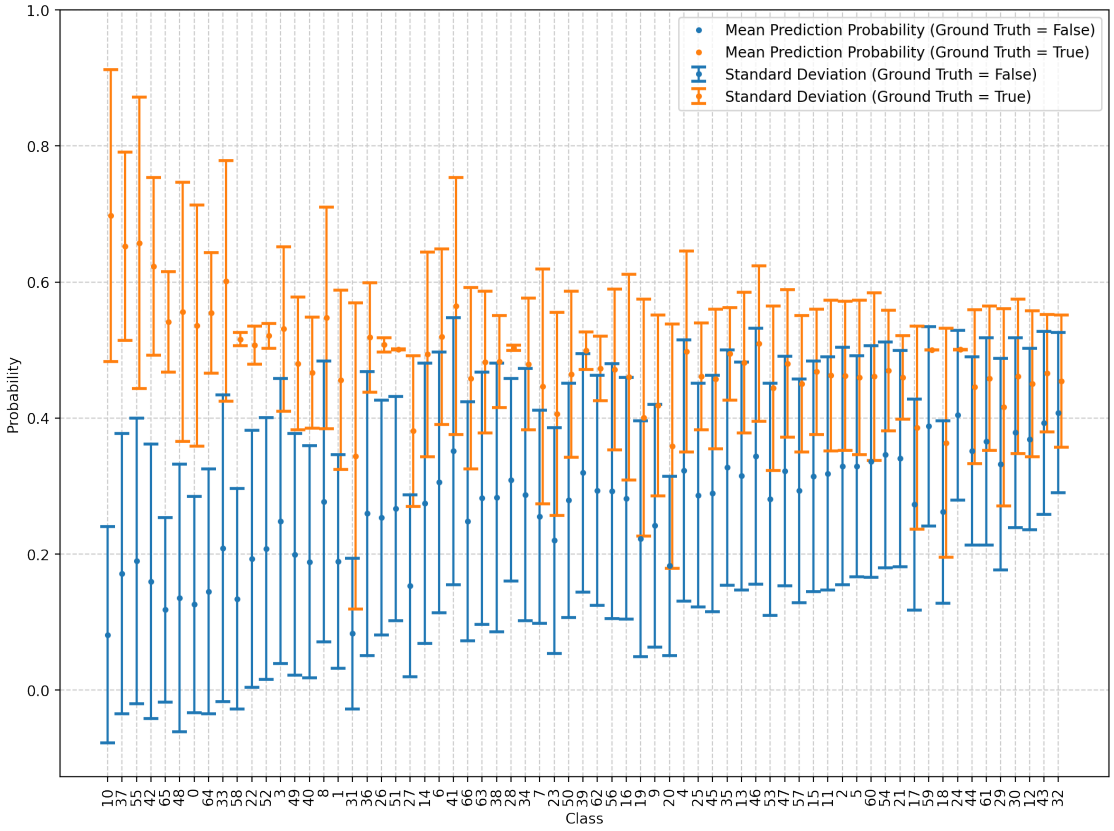
[15]:
```
calculate_plot_with_thresholds(test_labels, test_pred, thresholds,␣
↪model_name='CPC',
                               save_path='/home/julian/Documents/projekt-master/
↪bilder/errorplot-prediction-threshold-cpc-stdgap.png',
                               sort_fn=sort_by_std_gap,␣
↪std_two_directional=False)
```

```
[10 65 58 64 37 42 52 22  0 26 51 55 28 48 40 49 33 27  1 39 24 36 59 62
  3 38 25 35 31 63 34 66  8 21 13  6 57 15 45 50 47 56 53 23 54  9 20 46
 14 11  7  5 43 16  2 44 12  4 61 60 32 19 30 41 17 18 29]
```



[16]:
```
calculate_plot_with_thresholds(test_labels, test_pred, thresholds,␣
↪model_name='CPC',
                               save_path='/home/julian/Documents/projekt-master/
↪bilder/errorplot-prediction-threshold-cpc-meangap.png',
                               sort_fn=sort_by_mean_gap,␣
↪std_two_directional=False)
```

```
[10 37 55 42 65 48  0 64 33 58 22 52  3 49 40  8  1 31 36 26 51 27 14  6
 41 66 63 38 28 34  7 23 50 39 62 56 16 19  9 20  4 25 45 35 13 46 53 47
 57 15 11  2  5 60 54 21 17 59 18 24 44 61 29 30 12 43 32]
```

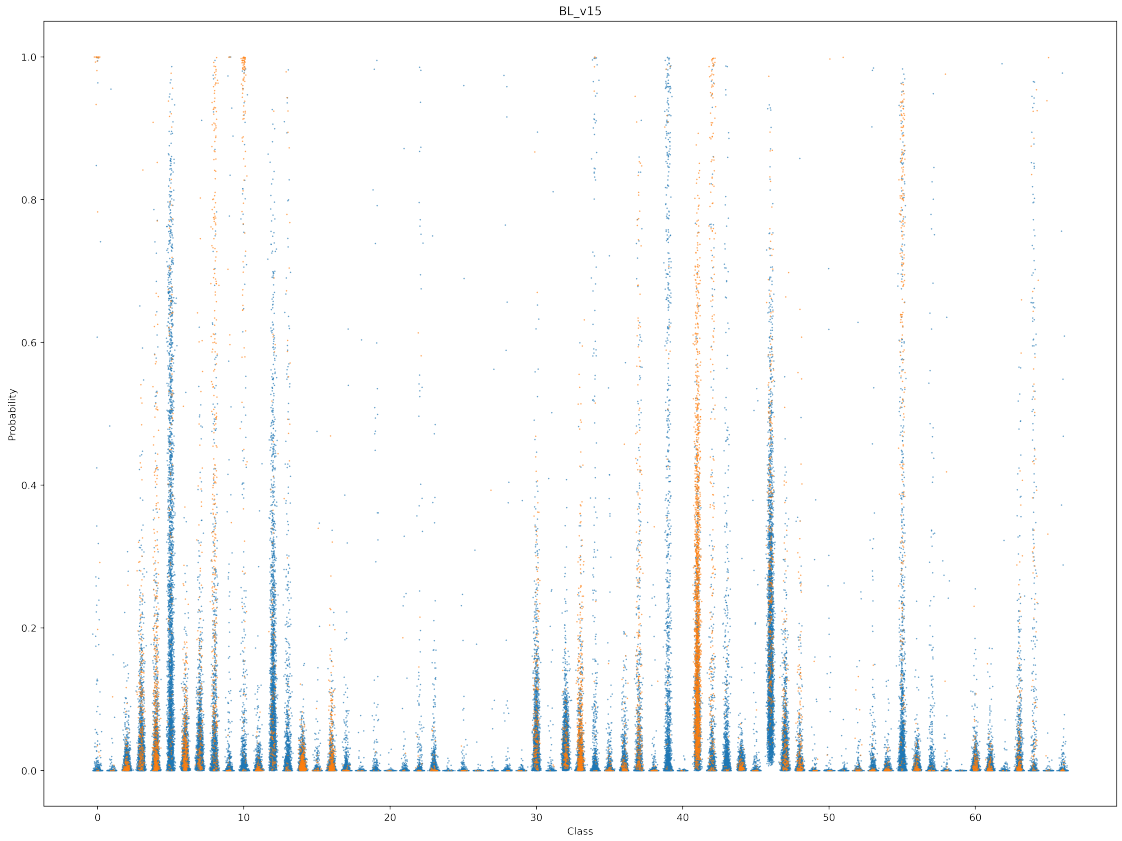


2.2.2 BL

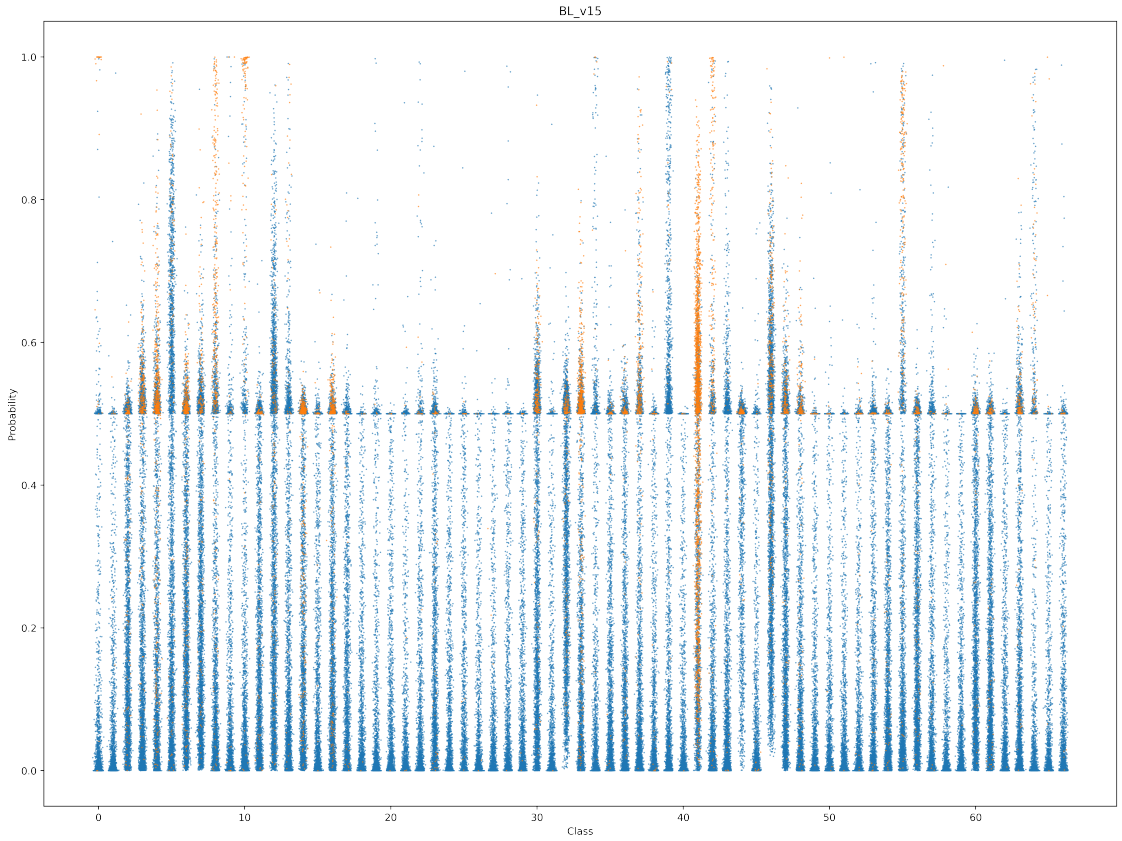
[17]:
```
# model_test_folder = '/home/julian/Downloads/Github/
 ↪contrastive-predictive-coding/models/16_08_21-10-16-test|(40x)cpc/
 ↪14_08_21-15_36-train|(4x)cpc/architectures_cpc.cpc_combined.
 ↪CPCCombined0|train-test-splits-fewer-labels60|use_weights|frozen|C|m:
 ↪all|cpc_downstream_cnn'
model_test_folder = '/home/julian/Downloads/Github/
 ↪contrastive-predictive-coding/models/
 ↪11_08_21-15-58-test|(10x)bl_TCN_down+(10x)bl_cnn_v1+(10x)bl_cnn_v14+(10x)bl_cnn_v15+(10x)bl
 ↪10_08_21-14_42-train|bl_TCN_down+bl_cnn_v1+bl_cnn_v14+bl_cnn_v15+bl_cnn_v8/
 ↪architectures_baseline_challenge.baseline_cnn_v15.
 ↪BaselineNet4|train-test-splits-fewer-labels60|use_weights|ConvLyrs:
 ↪5|MaxPool|Linear|BatchNorm|stride_sum:14|dilation_sum:96|padding_sum:
 ↪0|krnls_sum:26'
test_labels, classes =␣
 ↪read_binary_label_csv_from_model_folder(model_test_folder, 0)
test_pred, _ = read_output_csv_from_model_folder(model_test_folder, 0)
val_labels, classes =␣
 ↪read_binary_label_csv_from_model_folder(model_test_folder, 1)
```

```
val_pred, _ = read_output_csv_from_model_folder(model_test_folder, 1)
#for now use ready file:

# threshold_file = '/home/julian/Downloads/Github/contrastive-predictive-coding/
 ↪images/explain/CPCCombined0/thresholds.txt'
threshold_file = '/home/julian/Downloads/Github/contrastive-predictive-coding/
 ↪images/explain/BaselineNet4/thresholds.txt'
with open(threshold_file, 'r') as f:
    data = f.read()
thresholds = np.array(list(map(lambda x: x.item(), map(eval, data.
 ↪split(',')))))  #'slighlty' dangerous
```

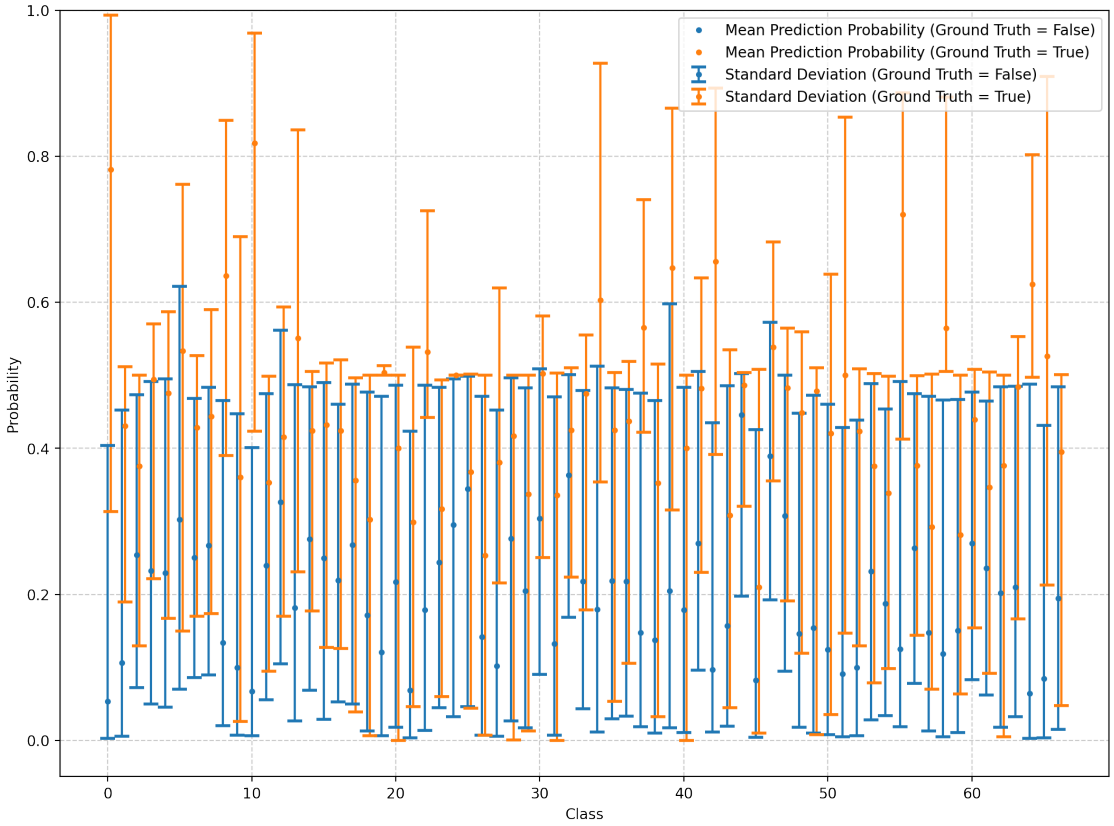
[18]:
```
pos = np.where(test_labels==0, test_pred, np.nan)
neg = np.where(test_labels!=0, test_pred, np.nan)
x1 = np.random.normal(scale=0.1, size=pos.shape)+np.arange(67)[np.newaxis, :]
plt.figure(figsize=[6.4*3, 4.8*3], dpi=400)
plt.scatter(x1, pos, s=0.1)
x2 = np.random.normal(scale=0.1, size=neg.shape)+np.arange(67)[np.newaxis, :]
plt.scatter(x2, neg, s=0.1)
plt.title("BL_v15")
plt.xlabel('Class')
plt.ylabel("Probability")
plt.savefig('/home/julian/Documents/projekt-master/bilder/
 ↪errorplot-scattered-bl.png', dpi=400)
plt.show()
```



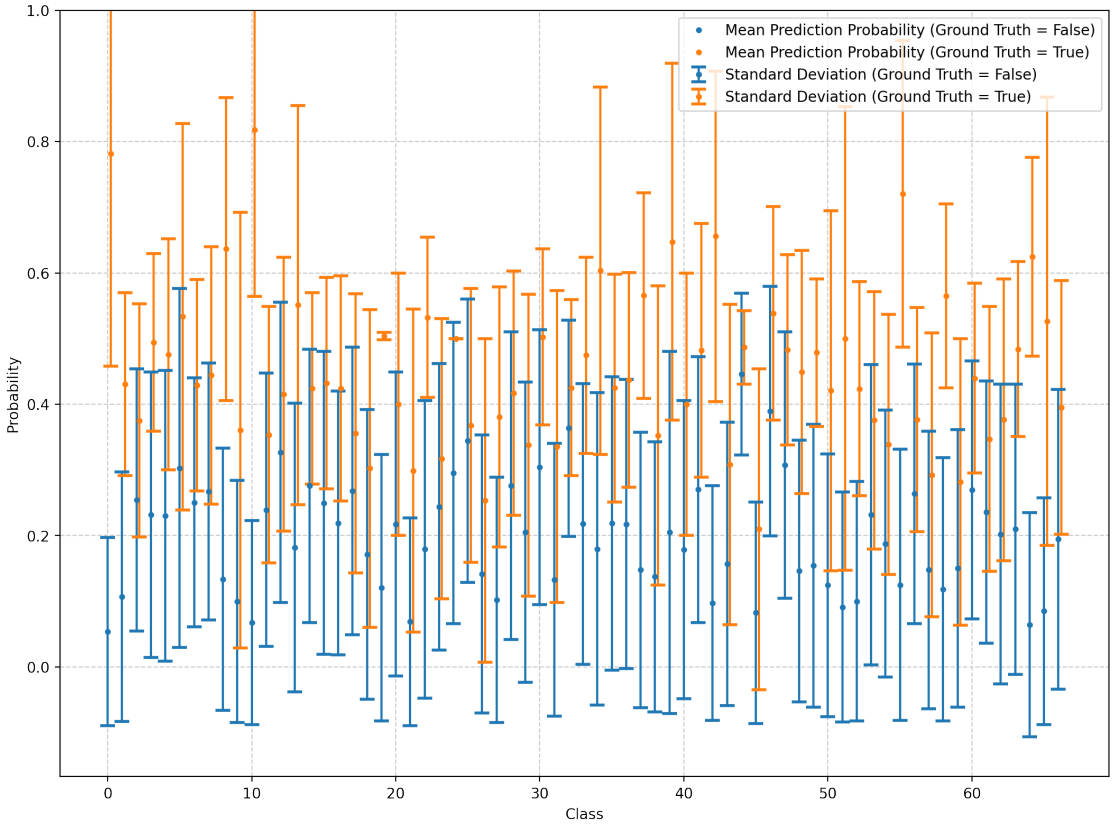
[19]:
```
diff = (test_pred - thresholds)
transformed_probs = (np.where(diff<0, diff/thresholds, diff/(1-thresholds))+1.)/
 ↪2.
pos = np.where(test_labels==0, transformed_probs, np.nan)
neg = np.where(test_labels!=0, transformed_probs, np.nan)
x1 = np.random.normal(scale=0.1, size=pos.shape)+np.arange(67)[np.newaxis, :]
plt.figure(figsize=[6.4*3, 4.8*3], dpi=400)
plt.scatter(x1, pos, s=0.1)
x2 = np.random.normal(scale=0.1, size=neg.shape)+np.arange(67)[np.newaxis, :]
plt.scatter(x2, neg, s=0.1)
plt.title('BL_v15')
plt.xlabel('Class')
plt.ylabel("Probability")
plt.savefig('/home/julian/Documents/projekt-master/bilder/
 ↪errorplot-scattered-thresholds-bl.png', dpi=400)
plt.show()
```



[20]:
```
calculate_plot_with_thresholds(test_labels, test_pred, thresholds,␣
↪model_name='BL_v15',
                               save_path='/home/julian/Documents/projekt-master/
↪bilder/errorplot-prediction-threshold-blv15-twodirectional.png',
                               sort_fn=None, std_two_directional=True)#
```

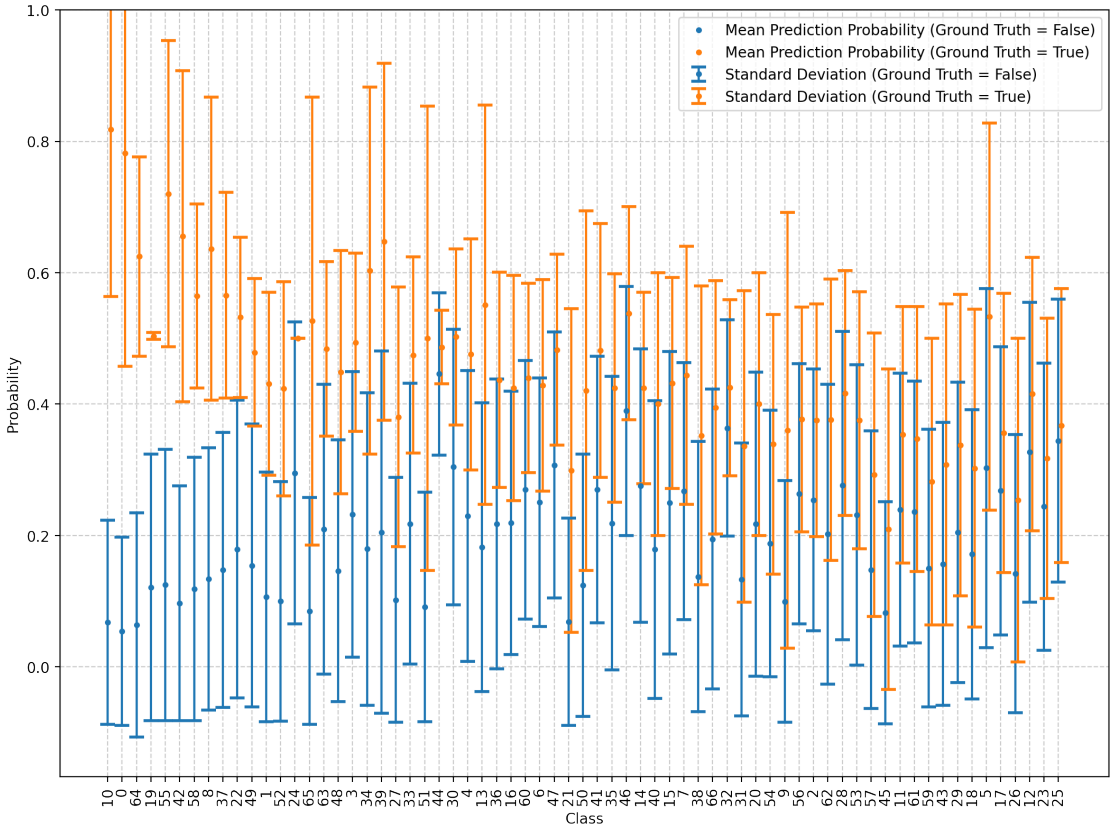


[21]:
```
calculate_plot_with_thresholds(test_labels, test_pred, thresholds,␣
↪model_name='BL_v15',
                               save_path='/home/julian/Documents/projekt-master/
↪bilder/errorplot-prediction-threshold-blv15.png',
                               sort_fn=None, std_two_directional=False)
```



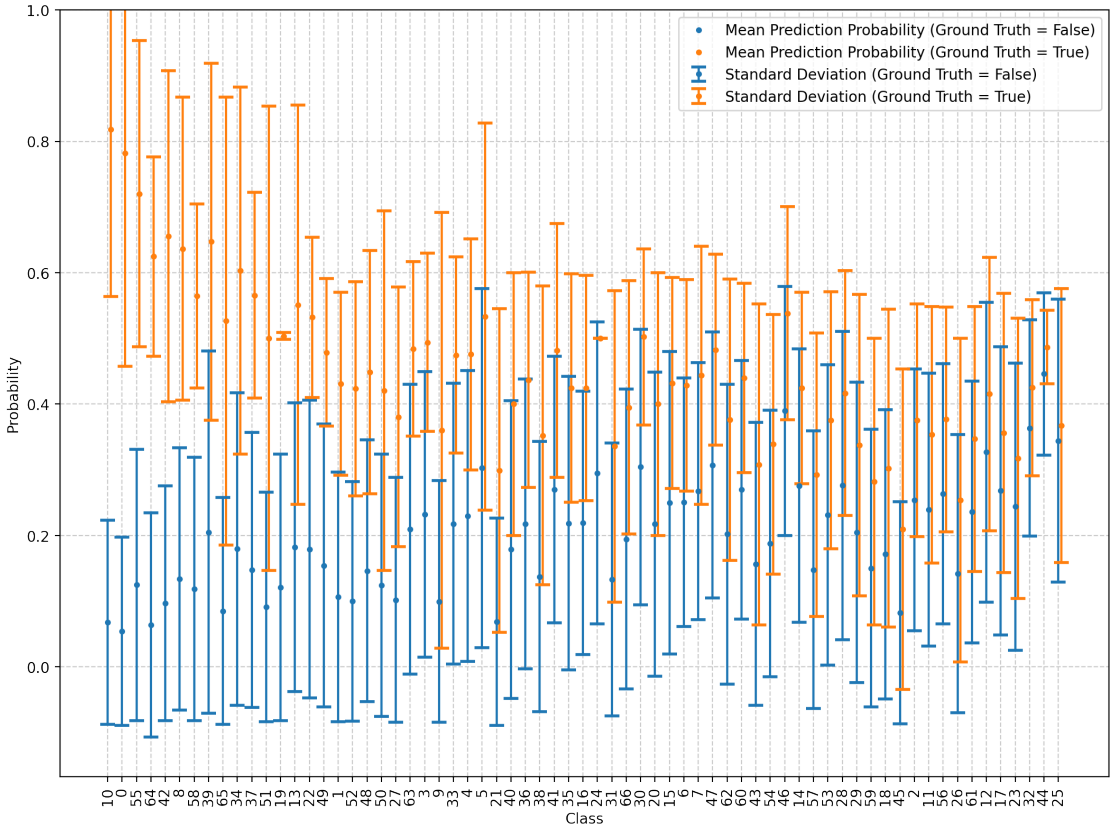
[22]:
```
calculate_plot_with_thresholds(test_labels, test_pred, thresholds,␣
↪model_name='BL_v15',
                               save_path='/home/julian/Documents/projekt-master/
↪bilder/errorplot-prediction-threshold-blv15-stdgap.png',
                               sort_fn=sort_by_std_gap,␣
↪std_two_directional=False)
```

```
[10  0 64 19 55 42 58  8 37 22 49  1 52 24 65 63 48  3 34 39 27 33 51 44
 30  4 13 36 16 60  6 47 21 50 41 35 46 14 40 15  7 38 66 32 31 20 54  9
 56  2 62 28 53 57 45 11 61 59 43 29 18  5 17 26 12 23 25]
```



[23]:
```
calculate_plot_with_thresholds(test_labels, test_pred, thresholds,␣
↪model_name='BL_v15',
                               save_path='/home/julian/Documents/projekt-master/
↪bilder/errorplot-prediction-threshold-blv15-meangap.png',
                               sort_fn=sort_by_mean_gap,␣
↪std_two_directional=False)
```

```
[10  0 55 64 42  8 58 39 65 34 37 51 19 13 22 49  1 52 48 50 27 63  3  9
 33  4  5 21 40 36 38 41 35 16 24 31 66 30 20 15  6  7 47 62 60 43 54 46
 14 57 53 28 29 59 18 45  2 11 56 26 61 12 17 23 32 44 25]
```


Mean Prediction Probability (Ground Truth = False)
Mean Prediction Probability (Ground Truth = True)
Standard Deviation (Ground Truth = False)
Standard Deviation (Ground Truth = True)
Probability
1.0
0.8
0.6
0.4
0.2
0.0
Class
10 0 55 64 42 8 58 39 65 34 37 51 19 13 22 49 1 52 48 50 27 63 3 9 33 4 5 21 40 36 38 41 35 16 24 31 66 30 20 15 6 7 47 62 60 43 54 46 14 57 53 28 29 59 18 45 2 11 56 26 61 12 17 23 32 44 25

[ ]: