Supplementary Materials: Convolutional Neural Networks (CNNs) applied to Antimony quantification via reflectance spectroscopy using soils from Northern Portugal: Opportunities and Challenges

Morgana Carvalho 1,2\*, Joana Cardoso-Fernandes 1,2 Alexandre Lima1,2 and Ana C. Teodoro1,2

1 Department of Geosciences, Environment and Spatial Planning, Faculty of Sciences, University of Porto, Rua Campo Alegre, 4169-007 Porto, Portugal; joana.fernandes@fc.up.pt (J.C-F.); allima@fc.up.pt (A.L.); amteodor@fc.up.pt (A.C.T.)

2 ICT (Institute of Earth Sciences) – Porto pole (Portugal), Rua Campo Alegre, 4169-007 Porto, Portugal

\* morgana.carvalho@fc.up.pt

Spectrogram transformation

The script presented in this section consists of the application of the Short-Time Fourier Transform (STFT) to the equal\_length tensor using tf.signal.stft. A new dimension is added to the spectrogram tensor using spectrogram [..., tf.newaxis]. This dimension is added to make the spectrogram suitable as input data with convolution layers in the CNN. The data to be used must be already converted to pickle format and the necessary preprocessing must already be applied. The output is a file in pickle format.

Code S1: Conversion to Spectrogram

 1. import pandas as pd

 2. import tensorflow as tf

 3. import numpy as np

 4. import matplotlib.pyplot as plt

 5. from scipy.io import wavfile

 6. from scipy import signal

 7. import pickle

 8.

 9. # Load the waveform data from a pickle file

10. experiment\_name = "test\_Tr1"

11. input\_file = f'../data/{experiment\_name}/sample\_matched.pickle'

12. output\_file = f'../data/{experiment\_name}/spectogram.pickle'

13.

14. with open(input\_file, 'rb') as handle:

15. data\_dict = pickle.load(handle)

16.

17. # Function to normalize a spectrogram using min-max scaling

18. def normalize\_spectrogram(spec):

19. spec\_min = np.min(spec)

20. spec\_max = np.max(spec)

21. spec\_norm = (spec - spec\_min) / (spec\_max - spec\_min)

22. return spec\_norm

23.

24. # Function to generate a spectrogram from a waveform

25. def get\_spectrogram(waveform):

26. input\_len = 2151

27. waveform = waveform[:input\_len]

28. zero\_padding = tf.zeros([2151] - tf.shape(waveform), dtype=tf.float32)

29. waveform = tf.cast(waveform, dtype=tf.float32)

30. equal\_length = tf.concat([waveform, zero\_padding], 0)

31. spectrogram = tf.signal.stft(equal\_length, frame\_length=128, frame\_step=64)

32. spectrogram = tf.abs(spectrogram)

33. spectrogram = spectrogram[..., tf.newaxis]

34. return spectrogram

35.

36. # Function to plot a spectrogram

37. def plot\_spectrogram(spectrogram, ax):

38. if len(spectrogram.shape) > 2:

39. assert len(spectrogram.shape) == 3

40. spectrogram = np.squeeze(spectrogram, axis=-1)

41. log\_spec = np.log(spectrogram.T + np.finfo(float).eps)

42. height = log\_spec.shape[0]

43. width = log\_spec.shape[1]

44. X = np.linspace(0, np.size(spectrogram), num=width, dtype=int)

45. Y = range(height)

46. ax.pcolormesh(X, Y, log\_spec)

47.

48. # Function to visualize waveform and corresponding spectrogram

49. def draw\_spec(waveform):

50. spectrogram = get\_spectrogram(waveform)

51. fig, axes = plt.subplots(2, figsize=(12, 8))

52. timescale = np.arange(waveform.shape[0])

53. axes[0].plot(timescale, waveform)

54. axes[0].set\_title('Waveform')

55. axes[0].set\_xlim([0, 2151])

56.

57. spec = normalize\_spectrogram(spectrogram.numpy())

58. plot\_spectrogram(spec, axes[1])

59. axes[1].set\_title('Spectrogram')

60. plt.show()

61.

62. # Function to map waveforms to spectrograms for multiple samples

63. def map\_dict\_wave\_spec(data):

64. o\_aux = {}

65. for sample, data in data.items():

66. aux = []

67. for c in data.columns:

68. freq, time, spec = signal.spectrogram(data[c])

69. aux.append(normalize\_spectrogram(spec))

70. o\_aux[sample] = np.array(aux)

71. return o\_aux

72.

73. # Function to map waveforms to spectrograms for a single sample

74. def map\_wave\_spec(data):

75. o\_aux = []

76. for d in data:

77. aux = []

78. for l in d:

79. freq, time, spec = signal.spectrogram(l)

80. aux.append(spec)

81. o\_aux.append(aux)

82. return np.array(o\_aux)

83.

84. # Generate spectrograms for all samples in the data dictionary

85. specs = map\_dict\_wave\_spec(data\_dict)

86.

87. # Save the spectrograms to a pickle file

88. with open(output\_file, 'wb') as handle:

89. pickle.dump(specs, handle, protocol=pickle.HIGHEST\_PROTOCOL)

Convolutional neural networks

MobileNets are a class of highly efficient CNN models, built upon a streamlined architecture that leverages depth wise separable convolutions, being a deep neural network with significantly reduced computational demand. The spectral input data is in pikcle format, and the input labels must be a csv. The data given as input and labels much match and “experiment\_name” must be an already existent file.

Code S2: MobiletNet model

 1. import pandas as pd

 2. import tensorflow as tf

 3. import numpy as np

 4. import os

 5. import datetime

 6. import pickle

 7.

 8. from sklearn.model\_selection import train\_test\_split

 9.

 10. from tensorflow.keras import datasets, layers, models

 11. import matplotlib.pyplot as plt

 12. %matplotlib inline

 13.

 14. # Experiment name and input paths

 15. experiment\_name = "experiment\_1"

 16. input\_data = f'../data/{experiment\_name}/spectogram.pickle' # Receives spectrograms

 17. input\_label = f"../data/{experiment\_name}/labels.csv" # receive labels (targets)

 18.

 19. # Load data from pickle files

 20. with open(input\_data, 'rb') as handle:

 21. specs\_dict = pickle.load(handle)

 22.

 23. labels = pd.read\_csv(input\_label, delimiter=";")

 24. GERA LABELS (GENERATE LABELS)

 25. print(specs\_dict.keys())

 26.

 27. # Prepare data and labels

 28. \_labels = labels[["sample", "Sb\_ppm"]]

 29. specs = []

 30. target = []

 31.

 32. # Loop through labels to create data and target arrays

 33. for r in \_labels.iterrows():

 34. label = [r[1][1]] # Choose what to use, see example

 35. spectograms = specs\_dict[r[1][0]]

 36. respec = tf.squeeze(tf.transpose(spectograms, [1, 2, 0]))

 37. target.append(label)

 38. specs.append(respec)

 39.

 40. specs = np.array(specs)

 41. target = np.array(target)

 42.

 43. # Split data into train and test sets

 44. train\_examples, test\_examples, train\_labels, test\_labels = train\_test\_split(specs, target, test\_size=0.33, random\_state=42)

 45.

 46. # Transform data into TensorFlow datasets

 47. train\_dataset = tf.data.Dataset.from\_tensor\_slices((train\_examples, train\_labels))

 48. test\_dataset = tf.data.Dataset.from\_tensor\_slices((test\_examples, test\_labels))

 49.

 50. # Set AUTOTUNE for performance optimization

 51. AUTOTUNE = tf.data.AUTOTUNE

 52. train\_dataset = train\_dataset.prefetch(buffer\_size=AUTOTUNE)

 53.

 54. # Define batch size and shuffle buffer size

 55. BATCH\_SIZE = 64

 56. SHUFFLE\_BUFFER\_SIZE = 100

 57.

 58. train\_dataset = train\_dataset.batch(BATCH\_SIZE)

 59.

 60. # Model definition (MobileNetV2)

 61. def \_conv\_block(inputs, filters, kernel, strides):

 62. """Convolution Block

 63. This function defines a 2D convolution operation with BN and ReLU6 activation.

 64. """

 65.

 66. x = tf.keras.layers.Conv2D(filters, kernel, padding='same', strides=strides)(inputs)

 67. x = tf.keras.layers.BatchNormalization()(x)

 68. return tf.nn.relu6(x)

 69.

 70. def \_bottleneck(inputs, filters, kernel, t, s, r=False):

 71. """Bottleneck

 72. This function defines a basic bottleneck structure.

 73. """

 74.

 75. tchannel = inputs.shape[-1] \* t

 76.

 77. x = \_conv\_block(inputs, tchannel, (1, 1), (1, 1))

 78.

 79. x = tf.keras.layers.DepthwiseConv2D(kernel, strides=(s, s), depth\_multiplier=1, padding='same')(x)

 80. x = tf.keras.layers.BatchNormalization()(x)

 81. x = tf.nn.relu6(x)

 82.

 83. x = tf.keras.layers.Conv2D(filters, (1, 1), strides=(1, 1), padding='same')(x)

 84. x = tf.keras.layers.BatchNormalization()(x)

 85.

 86. if r:

 87. x = tf.keras.layers.add([x, inputs])

 88. return x

 89.

 90. def \_inverted\_residual\_block(inputs, filters, kernel, t, strides, n):

 91. """Inverted Residual Block

 92. This function defines a sequence of 1 or more identical layers.

 93. """

 94.

 95. x = \_bottleneck(inputs, filters, kernel, t, strides)

 96.

 97. for i in range(1, n):

 98. x = \_bottleneck(x, filters, kernel, t, 1, True)

 99.

100. return x

101.

102. def simple\_cnn\_3\_blocks(input\_shape, n\_output):

103. """Simple CNN Model with 3 Blocks

104. """

105.

106. inputs = tf.keras.layers.Input(input\_shape)

107.

108. x = \_conv\_block(inputs, 32, (3, 3), strides=(2, 2))

109. x = \_inverted\_residual\_block(x, 16, (3, 3), t=1, strides=1, n=1)

110. x = \_inverted\_residual\_block(x, 24, (3, 3), t=6, strides=2, n=2)

111. x = \_inverted\_residual\_block(x, 32, (3, 3), t=6, strides=2, n=3)

112.

113. x = tf.keras.layers.Flatten()(x)

114. x = tf.keras.layers.Dense(64, activation='relu')(x)

115. outputs = tf.keras.layers.Dense(n\_output)(x)

116.

117. return tf.keras.Model(inputs, outputs)

118.

119. # Define input shape and number of outputs

120. input\_shape = (train\_examples.shape[1], train\_examples.shape[2], train\_examples.shape[3])

121. n\_output = \_labels.shape[1] - 1

122.

123. # Create and compile the model

124. model = simple\_cnn\_3\_blocks(input\_shape, n\_output)

125. optimizer = tf.keras.optimizers.Adam(0.01)

126. model.compile(loss='mse', optimizer="adam", metrics=[tf.keras.metrics.RootMeanSquaredError()])

127.

128. # Define callbacks for saving checkpoints and early stopping

129. now = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")

130. name\_experiment = f"{experiment\_name}-{now}"

131. logdir = os.path.join("../logs1", name\_experiment)

132. tensorboard\_callback = tf.keras.callbacks.TensorBoard(logdir)

133. checkpoint\_callback = tf.keras.callbacks.ModelCheckpoint("../checkpoints/custom\_cnn\_simple", save\_freq=500, monitor="val\_root\_mean\_squared\_error")

134.

135. checkpoint\_callback = tf.keras.callbacks.ModelCheckpoint(

136. "../checkpoints/custom\_cnn\_simple",

137. save\_freq=500,

138. monitor="val\_root\_mean\_squared\_error",

139. save\_best\_only=True,

140. mode="min"

141. )

142.

143. early\_stopping\_callback = tf.keras.callbacks.EarlyStopping(

144. monitor='val\_loss',

145. patience=1000,

146. restore\_best\_weights=True

147. )

148.

149. callbacks\_list = [checkpoint\_callback, early\_stopping\_callback]

150.

151. # Train the model

152. history = model.fit(

153. train\_dataset,

154. epochs=5000,

155. validation\_data=(test\_examples, test\_labels),

156. callbacks=callbacks\_list

157. )

158.

159. # Save the trained model

160. model.save(f"../saved\_model/{name\_experiment}\_As\_Sb")

161.

162. # Plot training history

163. def plot\_history(history):

164. hist = pd.DataFrame(history.history)

165. hist['epoch'] = history.epoch

166.

167. plt.figure()

168. plt.xlabel('Epoch')

169. plt.ylabel('Mean Square Error [$MPG^2$]')

170. plt.plot(hist['epoch'], hist['root\_mean\_squared\_error'], label='Train Error')

171. plt.plot(hist['epoch'], hist['val\_root\_mean\_squared\_error'], label='Val Error')

172.

173. plt.legend()

174. plt.show()

175.

176. plot\_history(history)

177.