Criterion C: Development

Libraries imported:

`import os` the operating system library used for directory operations
`import time` used for time-related functions; here: only the sleep function
`import subprocess` typically used for running shell commands; here: used only to execute the tensorboard command
`import webbrowser` for opening webbrowsers from python
`import numpy as np` handling arrays and operations on them
`import tkinter as tk` Tk GUI toolkit interface commands
`import cv2` OpenCV library used for image processing
`import random` generating pseudo-random numbers
`import matplotlib.pyplot as plt` used for plotting graphs
`import pickle` here: used for saving and loading features and labels sets data
`import tensorflow as tf` a set of standard tools and commands used for neural-network programming
`import matplotlib.patches as mpatches` for adding shapes on pyplot figures

Dataset handling

The user would begin by downloading or creating the training dataset, which will be used to train the different models of neural networks. The dataset needs to be formatted so that groups of square images are put into different folders, which are named as the categories of the groups. Here, the dataset used to build and test the program is the Hand-drawn Electric Circuit Schematic Components dataset downloaded from Kaggle:

https://www.kaggle.com/datasets/moodrammer/handdrawn-circuit-schematic-components.

First, the user needs to input the directory path to the dataset folder.

```
base_path = GUItk.run_input_box("Input the path to directory containing your
dataset folder: ")
```

Hence, the run_input_box function

run_input_box – creates a user input box using a friendly GUI Tkinker

was defined using the GUI Tkinker library.

```python
def run_input_box(prompt):
    def on_submit(event=None):

        user_input[0] = input_box.get()
        root.destroy() #Closes the input box after the user clicks the submit button or the Enter key on the keyboard

    root = tk.Tk()#Initializing the box
    root.title("Input Box")
    label = tk.Label(root, text=prompt)
    label.pack()

    input_box = tk.Entry(root, width=50)#Creates the box
    input_box.pack()
    input_box.focus()  #Focus the input box

    root.bind("<Return>", on_submit)
    input_box.bind("<Return>", on_submit)
    submit_button = tk.Button(root, text="Enter", command=on_submit)#submit button
    submit_button.pack()

    user_input = [None]#a list stores the input
    root.mainloop()
```

The main loop begins using this function to change the working directory to the input base_path.

```python
def main():

    os.chdir(prep.base_path)

    # preprocess and pack the dataset to X.picle and y.pickle

    prep.Create_images_array()
    training_data = prep.Create_training_data()
    prep.Pack_data(training_data)

    x_train, y_train = prep.Load_data()
    x_train = prep.Normalize_data(x_train=x_train)
```

Preprocessing data

Then, the functions from the preprocessing_data.py file are called consequently. The explained functionalities of each defined function will be shown color-coded in the rest of this work, as references to flowcharts from Criterion B: Design documentation:

Create_images_array – converts each image into a grayscale array and re-sizes them to the size input by the user

Create_training_data – prescribes each category an index number, groups the image arrays into these categories, and saves the results of categorized images to a training_data array

Pack_data – packs the training_data into two sets: a features set (containing the image arrays) and a labels set (containing the prescribed categories)

Load_data – loads the features and labels sets

Normalize_data – divides all values in the features set by 255.0 to make the grayscale image arrays contain only values between 0 and 1 – this step is needed to enable easy use of activation functions such as Relu or Softmax

The described functions are standard and commonly used in neural network programming. I modeled them as described in the sourced and researched material https://pythonprogramming.net/introduction-deep-learning-python-tensorflow-keras/.

The first function needs to use the following, pre-defined structures:

```
dataset_folder_name = GUItk.run_input_box("Input the name of your dataset folder: ")

dataset_dir = os.path.join(base_path, dataset_folder_name)

IMG_SIZE = int(GUItk.run_input_box("Input the target image size: "))

CATEGORIES = []

for category_name in os.listdir(dataset_dir):
    CATEGORIES.append(category_name)
```

where IMG_SIZE is input by the user as an integer, for example, 50, so the square images can be re-sized to, for this example, images of size 50x50 pixels.

Training models

Next, the different models have to be trained. The user needs to define the rangers for their desired CNN parameters: the number of dense layers, number of neurons per layer, number of convolutional layers, and kernel sizes. The total number of models to be trained is the product of multiplying the number of elements in each of these array ranges.

```
dense_layers_nums, neurons_per_layer, conv_layers_nums, kernel_sizes = GUItk.run_input_box_with_parameters()

number_of_models = len(dense_layers_nums) * len(neurons_per_layer) * len(conv_layers_nums) * len(kernel_sizes)
model_num = 1
```

For the user to be able to input the desired parameters, a new GUI input box had to be defined.

run_input_box_with_parameters – creates a unique input box using specifically for inputting the model's parameters

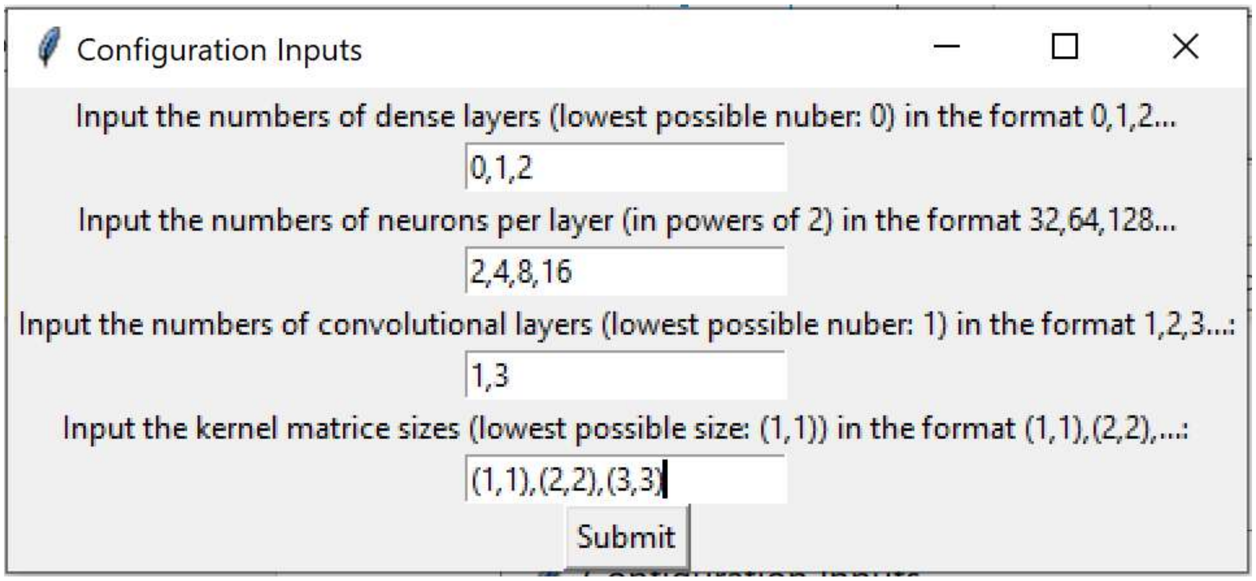
It is designed similarly to the standard run_input_box function, with the exception of differently processing the user inputs.

```python
def run_input_box_with_parameters():
    def on_submit():

        try:

            results['dense_layers_nums'] = [int(x) for x in dense_layers_input.get().split(',')]
            results['neurons_per_layer'] = [int(x) for x in neurons_input.get().split(',')]
            results['conv_layers_nums'] = [int(x) for x in conv_layers_input.get().split(',')]

            input_string = kernel_sizes_input.get().replace(" ", "")
            tuple_strings = input_string[1:-1].split("),(") if input_string.startswith("(") else input_string.split("),(")
            results['kernel_sizes'] = [tuple(map(int, x.split(','))) for x in tuple_strings]

        finally:
            root.destroy()
```

The first three inputs are arrays of integers split by commas and defined as follows:

```python
    #Dense layers input
    tk.Label(root, text="Input the numbers of dense layers (lowest possible nuber: 0) in the format 0,1,2...").pack()
    dense_layers_input = tk.Entry(root)
    dense_layers_input.pack()
```

The kernel_sizes input had to be handled differently – as an array of tuples. The resulting input box, with examples of inputs, in shown below.



Configuration Inputs

Input the numbers of dense layers (lowest possible nuber: 0) in the format 0,1,2...

0,1,2

Input the numbers of neurons per layer (in powers of 2) in the format 32,64,128...

2,4,8,16

Input the numbers of convolutional layers (lowest possible nuber: 1) in the format 1,2,3...:

1,3

Input the kernel matrice sizes (lowest possible size: (1,1)) in the format (1,1),(2,2),...:

(1,1),(2,2),(3,3)

Now, the program needs to iterate through each set of parameters to train a different model for each set. For this, a multiple-level nested loop was used:

```
    for dense_layers_num in train.dense_layers_nums:
        for neurons_num in train.neurons_per_layer:
            for conv_layers_num in train.conv_layers_nums:
                for kernel_size in train.kernel_sizes:

                    NAME = train.Set_NAME(neurons_num, dense_layers_num,
conv_layers_num, kernel_size)
```

The name of each new model is created using the input parameters.

Set_NAME – sets the name of the model to consist of its input parameters

Then, if the given model does not yet exist, the program has to create a new MyModel object.

```
                    if os.path.exists("models/{}.model".format(NAME)) ==
False:
                        MyModel = train.CNN_Model(x_train,
len(prep.CATEGORIES), neurons_num, dense_layers_num, conv_layers_num,
kernel_size)
```

This automatically initializes the training of the new model, which consists of calling the functions of the CNN_Model class defined in the models_training_and_optimization.py file.

__init__ – initializes self.model by calling Define_model_architecture

```
class CNN_Model:

    def __init__(self, x_train, CATEGORIES_num, neurons_num,
dense_layers_num, conv_layers_num, kernel_size):
        self.model = self.Define_model_architecture(x_train,
CATEGORIES_num, neurons_num, dense_layers_num, conv_layers_num,
kernel_size)
```

Define_model_architecture – defines the architecture of the model based on the input parameters: the number of convolutional layers, the number of dense layers, the number of neurons per layer, and the kernel sizes

First, the Sequential function is called, which will allow for the addition of different layers to the architecture of the model.

```
model = tf.keras.models.Sequential()
```

Each of the used layers was taken from the open-source Keras library documentation accessed at https://keras.io/api/layers/.

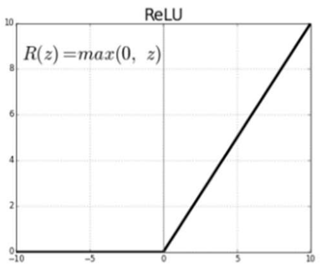
The added layers are in sequence:

- **Convolutional 2d** – applies filters to the image to transform it and create a features map; the size of each "window" of the filter is given by kernel_size, for example, a 3x3 pixel window.

```
        model.add(tf.keras.layers.Conv2D(neurons_num, kernel_size,
input_shape = x_train.shape[1:])) # 3x3 pixels window, shape dynamically
defined of X, here img_size = 50x50
```

- **Activation("relu")** – applies the Rectified Linear Unit (ReLU) function to the previous layer. It outputs the input value if it's positive; otherwise, it outputs zero.

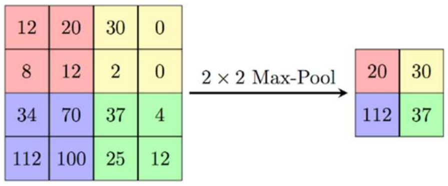
```
model.add(tf.keras.layers.Activation("relu"))
```



(refer to [7.] in Bibliography)

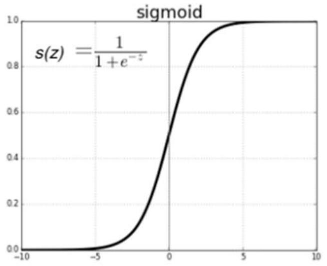
- **MaxPooling2D(pool_size=(2,2))** - performs max pooling with a 2x2 window, reducing the dimensions of the input feature maps by taking the maximum value of the window's area.

```
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2,2)))
```



(refer to [8.] in Bibliography)

- This process is repeated using a for loop for the given range of the number of convolutional layers
- **Flatten** – prepares the data to be input to the Dense layer by flattening it – transforming it into a one-dimensional input
```
model.add(tf.keras.layers.Flatten())
```
- **Dense(64)** – the standard deeply connected layer consisting of a set of neurons, each connecting to every neuron in the previous layer by a weight - for each input neuron, it calculates the weighted sum of all inputs indicating the importance of this input to the neuron – this is the true process behind a neural network learning by weighing the importance of each input; this layer is repeated, along with an Activation("relu") layer, using a for loop in range of dense_layers_num
```
model.add(tf.keras.layers.Dense(64))
```
- **Dense(len(prep.CATEGORIES))** – the final dense layer representing the level of confidence of the model for each category
```
        model.add(tf.keras.layers.Dense(len(prep.CATEGORIES))) #for each
category
```
- **Activation("softmax")** – the final activation layer, this time, using the "softmax" function
```
model.add(tf.keras.layers.Activation("softmax"))
```



(refer to [7.] in Bibliography)

Finally, the model is compiled using:

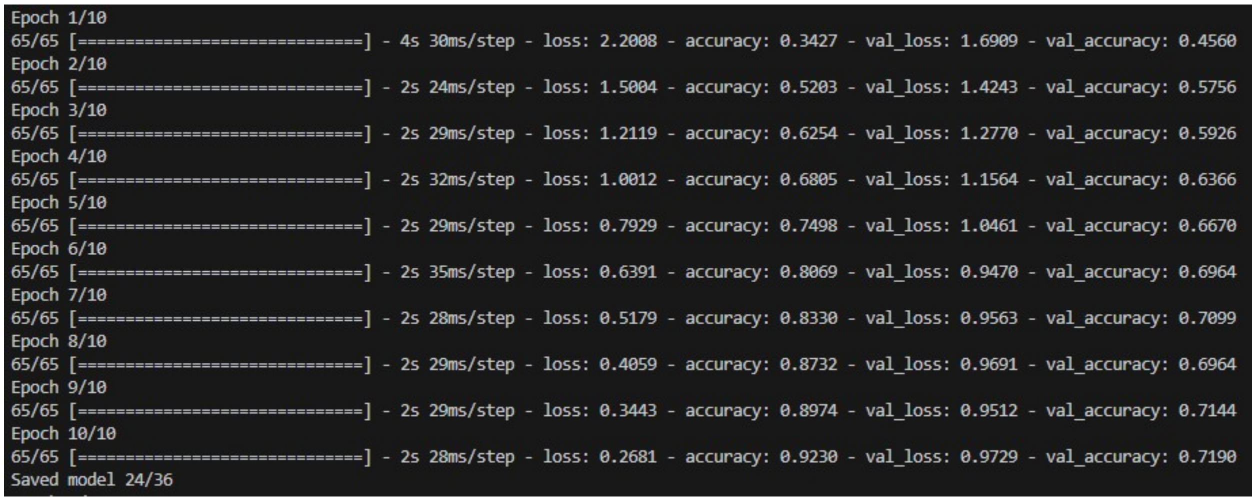
```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy']) # what to track
```

-	the **‘adam’ optimizer** (which specifies the optimization algorithm used to minimize the loss function during training – it adapts the learning rate for each weight of the model),
-	the **'sparse_categorical_crossentropy' loss** (the loss function that the model will try to minimize, here: sparse categorical crossentropy is used because the targets are integers and have multiple categories; it measures the difference between the predicted probabilities and the actual distribution of the labels),
-	the **‘accuracy’ metrics** (the metrics to be evaluated by the model during training and testing; here: 'accuracy' is used because it calculates how often predictions match labels for classification problems).

Train_model – trains the models accordingly to the defined model architecture and saves the history

```
    def Train_model(self, x_train, y_train, tensorboard):
        history = self.model.fit(x_train, y_train, epochs=10,
batch_size=32, validation_split=0.3, callbacks=[tensorboard])
        return history
```

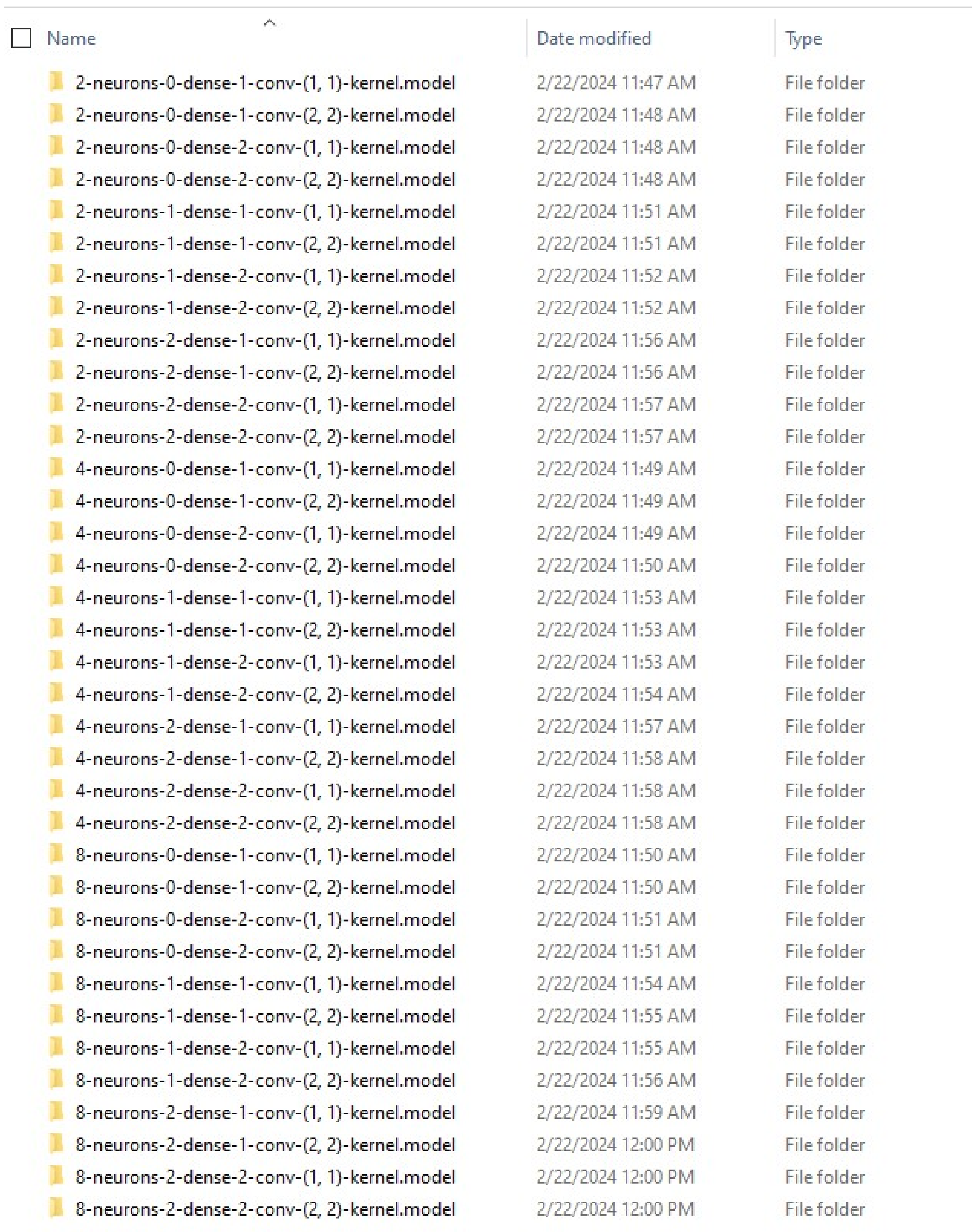
Here, the number of 10 epochs is the number of how many times the training dataset is passed through the network (an example of the run of 10 epochs is shown below),



```
Epoch 1/10
65/65 [==============================] - 4s 30ms/step - loss: 2.2008 - accuracy: 0.3427 - val_loss: 1.6909 - val_accuracy: 0.4560
Epoch 2/10
65/65 [==============================] - 2s 24ms/step - loss: 1.5004 - accuracy: 0.5203 - val_loss: 1.4243 - val_accuracy: 0.5756
Epoch 3/10
65/65 [==============================] - 2s 29ms/step - loss: 1.2119 - accuracy: 0.6254 - val_loss: 1.2770 - val_accuracy: 0.5926
Epoch 4/10
65/65 [==============================] - 2s 32ms/step - loss: 1.0012 - accuracy: 0.6805 - val_loss: 1.1564 - val_accuracy: 0.6366
Epoch 5/10
65/65 [==============================] - 2s 29ms/step - loss: 0.7929 - accuracy: 0.7498 - val_loss: 1.0461 - val_accuracy: 0.6670
Epoch 6/10
65/65 [==============================] - 2s 35ms/step - loss: 0.6391 - accuracy: 0.8069 - val_loss: 0.9470 - val_accuracy: 0.6964
Epoch 7/10
65/65 [==============================] - 2s 28ms/step - loss: 0.5179 - accuracy: 0.8330 - val_loss: 0.9563 - val_accuracy: 0.7099
Epoch 8/10
65/65 [==============================] - 2s 29ms/step - loss: 0.4059 - accuracy: 0.8732 - val_loss: 0.9691 - val_accuracy: 0.6964
Epoch 9/10
65/65 [==============================] - 2s 29ms/step - loss: 0.3443 - accuracy: 0.8974 - val_loss: 0.9512 - val_accuracy: 0.7144
Epoch 10/10
65/65 [==============================] - 2s 28ms/step - loss: 0.2681 - accuracy: 0.9230 - val_loss: 0.9729 - val_accuracy: 0.7190
Saved model 24/36
```

the batch size of 32 means the model will take 32 samples at a time, perform training, and only then update the parameters, the validation split reserves 30% of the training data for validation (evaluating the model's performance after each epoch, which helps prevent overfitting), and the callbacks log events for TensorBoard.

Save_model – saves the models accordingly with their name, as shown below



Name	Date modified	Type
2-neurons-0-dense-1-conv-(1, 1)-kernel.model	2/22/2024 11:47 AM	File folder
2-neurons-0-dense-1-conv-(2, 2)-kernel.model	2/22/2024 11:48 AM	File folder
2-neurons-0-dense-2-conv-(1, 1)-kernel.model	2/22/2024 11:48 AM	File folder
2-neurons-0-dense-2-conv-(2, 2)-kernel.model	2/22/2024 11:48 AM	File folder
2-neurons-1-dense-1-conv-(1, 1)-kernel.model	2/22/2024 11:51 AM	File folder
2-neurons-1-dense-1-conv-(2, 2)-kernel.model	2/22/2024 11:51 AM	File folder
2-neurons-1-dense-2-conv-(1, 1)-kernel.model	2/22/2024 11:52 AM	File folder
2-neurons-1-dense-2-conv-(2, 2)-kernel.model	2/22/2024 11:52 AM	File folder
2-neurons-2-dense-1-conv-(1, 1)-kernel.model	2/22/2024 11:56 AM	File folder
2-neurons-2-dense-1-conv-(2, 2)-kernel.model	2/22/2024 11:56 AM	File folder
2-neurons-2-dense-2-conv-(1, 1)-kernel.model	2/22/2024 11:57 AM	File folder
2-neurons-2-dense-2-conv-(2, 2)-kernel.model	2/22/2024 11:57 AM	File folder
4-neurons-0-dense-1-conv-(1, 1)-kernel.model	2/22/2024 11:49 AM	File folder
4-neurons-0-dense-1-conv-(2, 2)-kernel.model	2/22/2024 11:49 AM	File folder
4-neurons-0-dense-2-conv-(1, 1)-kernel.model	2/22/2024 11:49 AM	File folder
4-neurons-0-dense-2-conv-(2, 2)-kernel.model	2/22/2024 11:50 AM	File folder
4-neurons-1-dense-1-conv-(1, 1)-kernel.model	2/22/2024 11:53 AM	File folder
4-neurons-1-dense-1-conv-(2, 2)-kernel.model	2/22/2024 11:53 AM	File folder
4-neurons-1-dense-2-conv-(1, 1)-kernel.model	2/22/2024 11:53 AM	File folder
4-neurons-1-dense-2-conv-(2, 2)-kernel.model	2/22/2024 11:54 AM	File folder
4-neurons-2-dense-1-conv-(1, 1)-kernel.model	2/22/2024 11:57 AM	File folder
4-neurons-2-dense-1-conv-(2, 2)-kernel.model	2/22/2024 11:58 AM	File folder
4-neurons-2-dense-2-conv-(1, 1)-kernel.model	2/22/2024 11:58 AM	File folder
4-neurons-2-dense-2-conv-(2, 2)-kernel.model	2/22/2024 11:58 AM	File folder
8-neurons-0-dense-1-conv-(1, 1)-kernel.model	2/22/2024 11:50 AM	File folder
8-neurons-0-dense-1-conv-(2, 2)-kernel.model	2/22/2024 11:50 AM	File folder
8-neurons-0-dense-2-conv-(1, 1)-kernel.model	2/22/2024 11:51 AM	File folder
8-neurons-0-dense-2-conv-(2, 2)-kernel.model	2/22/2024 11:51 AM	File folder
8-neurons-1-dense-1-conv-(1, 1)-kernel.model	2/22/2024 11:54 AM	File folder
8-neurons-1-dense-1-conv-(2, 2)-kernel.model	2/22/2024 11:55 AM	File folder
8-neurons-1-dense-2-conv-(1, 1)-kernel.model	2/22/2024 11:55 AM	File folder
8-neurons-1-dense-2-conv-(2, 2)-kernel.model	2/22/2024 11:56 AM	File folder
8-neurons-2-dense-1-conv-(1, 1)-kernel.model	2/22/2024 11:59 AM	File folder
8-neurons-2-dense-1-conv-(2, 2)-kernel.model	2/22/2024 12:00 PM	File folder
8-neurons-2-dense-2-conv-(1, 1)-kernel.model	2/22/2024 12:00 PM	File folder
8-neurons-2-dense-2-conv-(2, 2)-kernel.model	2/22/2024 12:00 PM	File folder

Save_histories_and_colors – saves the histories and colors generated for each model, needed to plot the learning curves with pyplot

Check_and_create_dir – checks if a directory exists and if not – creates it

This causes an error output box to appear

run_output_box – creates a user output box using a friendly GUI Tkinker

if a model with the same name already exists.

```
                else:
                    GUItk.run_output_box("An old model in this folder with this name already exists - it will not be trained again. If you want to train a new one, please delete the old models first.")
                    pass
```

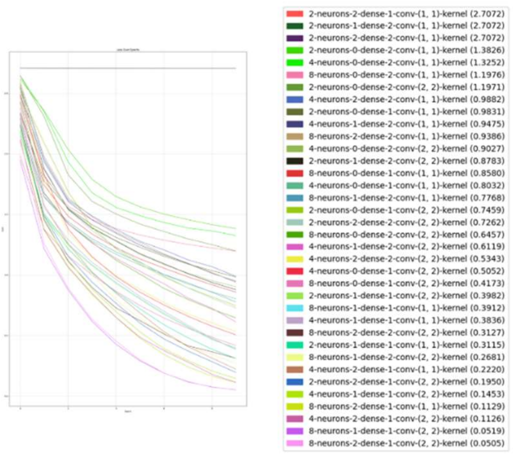
Save_TensorBoard_logs – saves the logs needed to later compile the tensorboard graphs

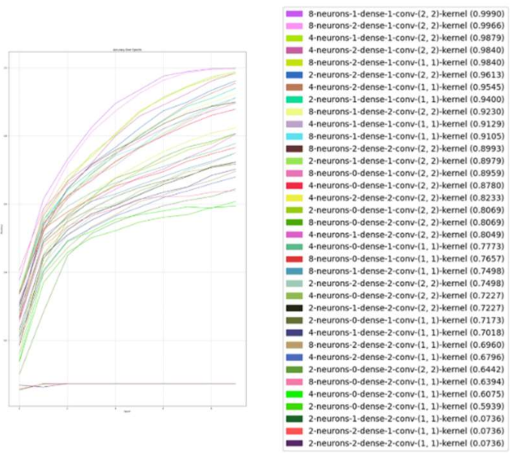
Plotting models' performances

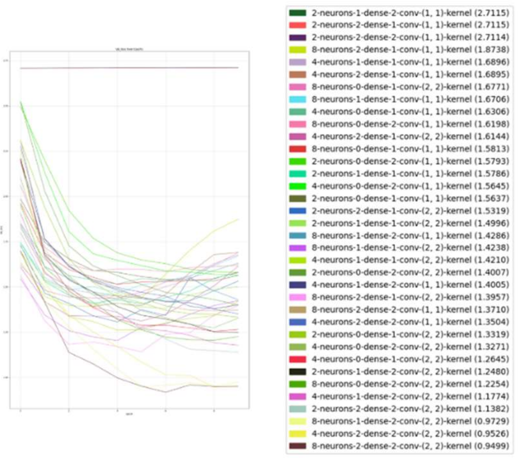
The models are sorted by their final validation accuracy. Then, their learning curves are plotted using pyplot and the following function:

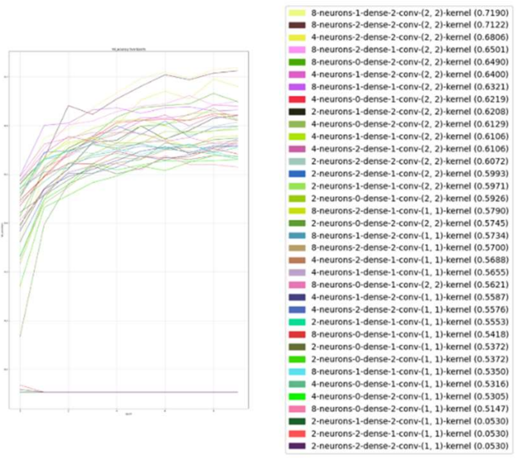
plot_learning_curves – creates the training and validation graphs of accuracy and loss, along with a color-coded legend sorted by the models' final validation accuracy, as well as finds the best model based on validation accuracy and graphs its learning curves separately

Examples of obtained performance plots are shown below:

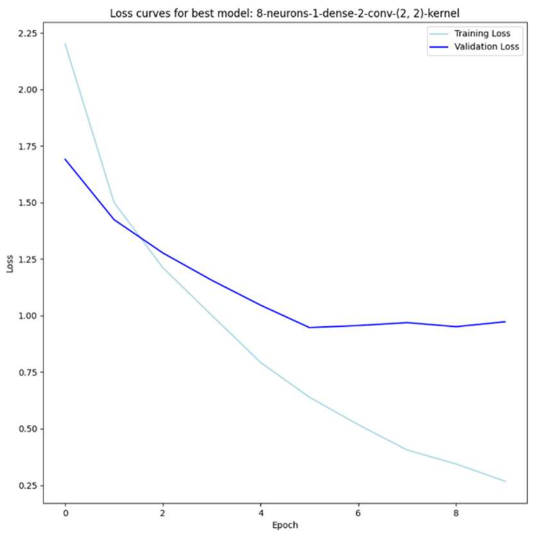


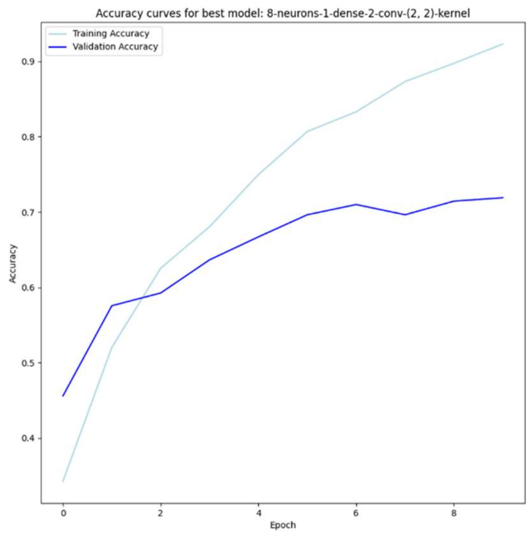






The best model is found by the highest value of the final validation accuracy, and two additional graphs are created, comparing its training and validation losses and accuracies.

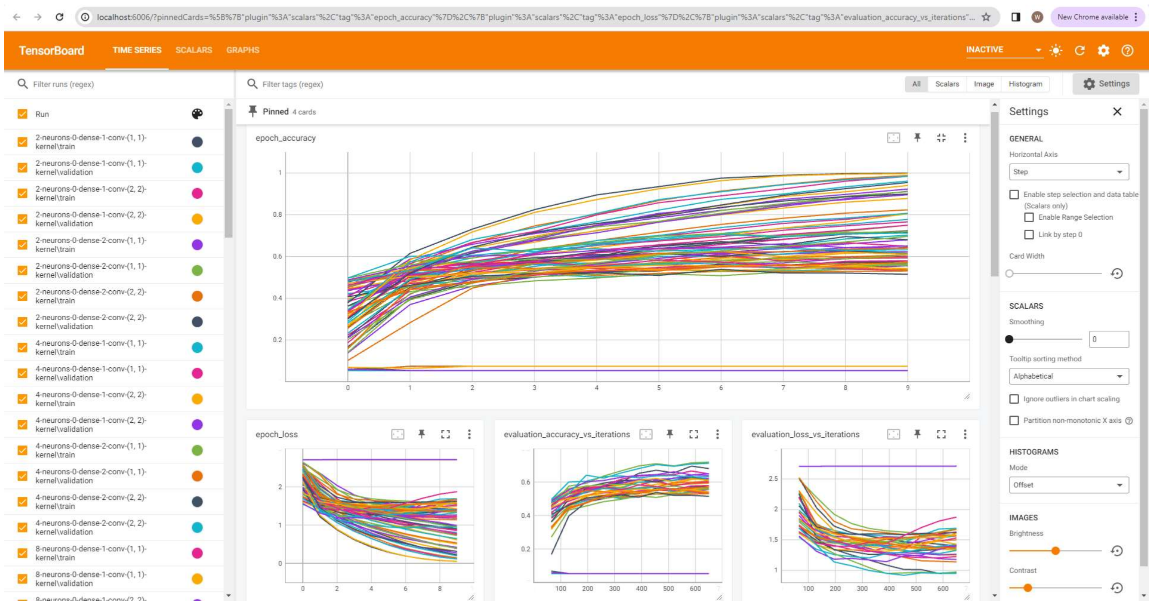




Next, the program opens the online, interactive TensorBoard graphs using the tensorboard_command, waiting 60 seconds to load the logs and 10 seconds to open the localhost website.

```
    # Run the tensorboard command and open logs online

    tensorboard_command = 'tensorboard --logdir logs'
    subprocess.Popen(tensorboard_command, shell=True)
    time.sleep(60)
    #print('opening website')
    os.system('start chrome http://localhost:6006/')
    time.sleep(10)
    GUItk.run_output_box("Your models are trained and their performances ready to evaluate.")
```

An example of the plotted TensorBoard graphs is shown below:



localhost:6006/?pinnedCards=%5B%7B"plugin"%3A"scalars"%2C"tag"%3A"epoch_accuracy"%7D%2C%7B"plugin"%3A"scalars"%2C"tag"%3A"epoch_loss"%7D%2C%7B"plugin"%3A"scalars"%2C"tag"%3A"evaluation_accuracy_vs_iterations"...

Run

- 2-neurons-0-dense-1-conv-(1, 1)-kernel\train
- 2-neurons-0-dense-1-conv-(1, 1)-kernel\validation
- 2-neurons-0-dense-1-conv-(2, 2)-kernel\train
- 2-neurons-0-dense-1-conv-(2, 2)-kernel\validation
- 2-neurons-0-dense-2-conv-(1, 1)-kernel\train
- 2-neurons-0-dense-2-conv-(1, 1)-kernel\validation
- 2-neurons-0-dense-2-conv-(2, 2)-kernel\train
- 2-neurons-0-dense-2-conv-(2, 2)-kernel\validation
- 4-neurons-0-dense-1-conv-(1, 1)-kernel\train
- 4-neurons-0-dense-1-conv-(1, 1)-kernel\validation
- 4-neurons-0-dense-1-conv-(2, 2)-kernel\train
- 4-neurons-0-dense-1-conv-(2, 2)-kernel\validation
- 4-neurons-0-dense-2-conv-(1, 1)-kernel\train
- 4-neurons-0-dense-2-conv-(1, 1)-kernel\validation
- 4-neurons-0-dense-2-conv-(2, 2)-kernel\train
- 4-neurons-0-dense-2-conv-(2, 2)-kernel\validation
- 8-neurons-0-dense-1-conv-(1, 1)-kernel\train
- 8-neurons-0-dense-1-conv-(1, 1)-kernel\validation

Settings

GENERAL

Horizontal Axis: Step

☐ Enable step selection and data table (Scalars only)
☐ Enable Range Selection
☐ Link by step 0

Card Width

SCALARS

Smoothing: 0

Tooltip sorting method: Alphabetical

☐ Ignore outliers in chart scaling
☐ Partition non-monotonic X axis

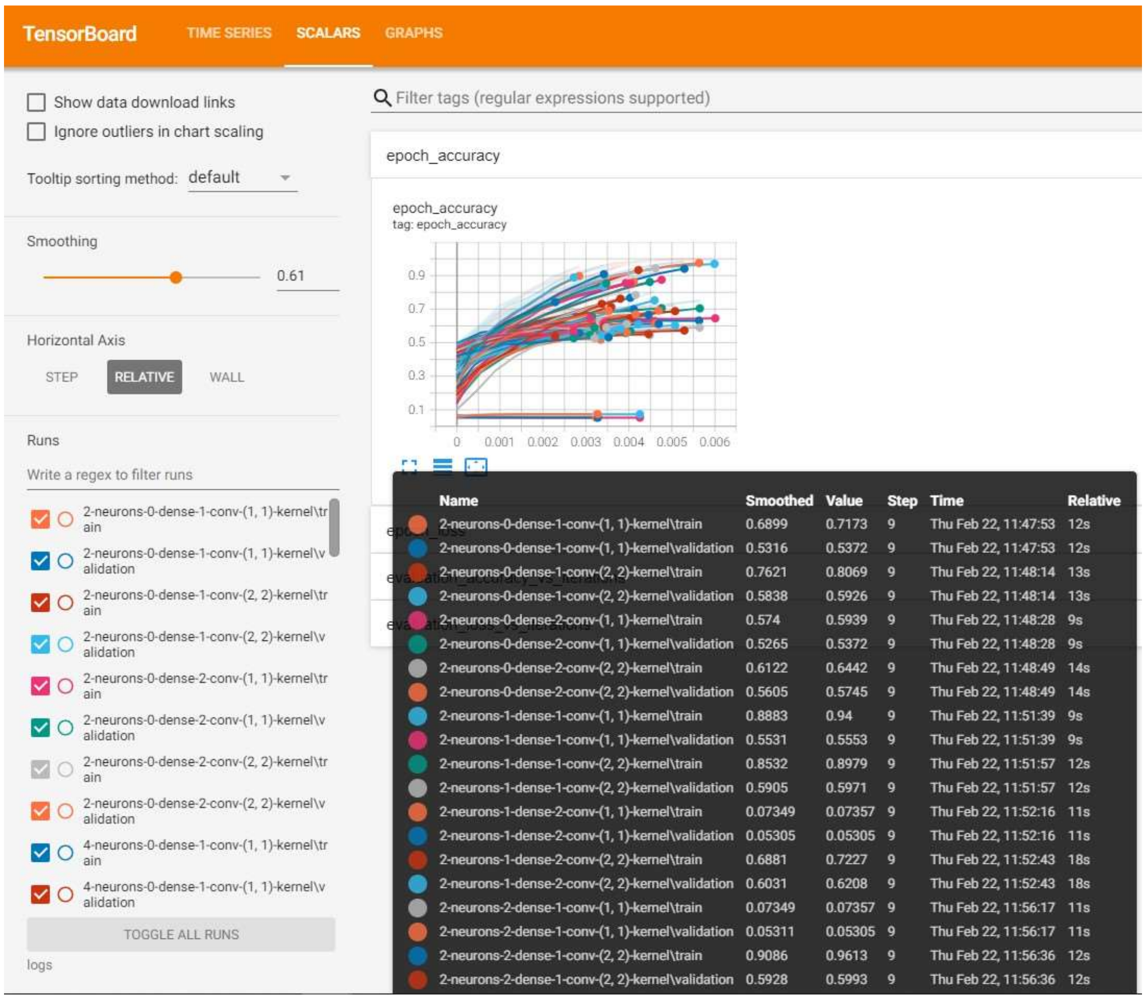
HISTOGRAMS

Mode: Offset

IMAGES

Brightness

Contrast



☐ Show data download links
☐ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.61

Horizontal Axis: STEP [✓ RELATIVE] WALL

Runs

Write a regex to filter runs

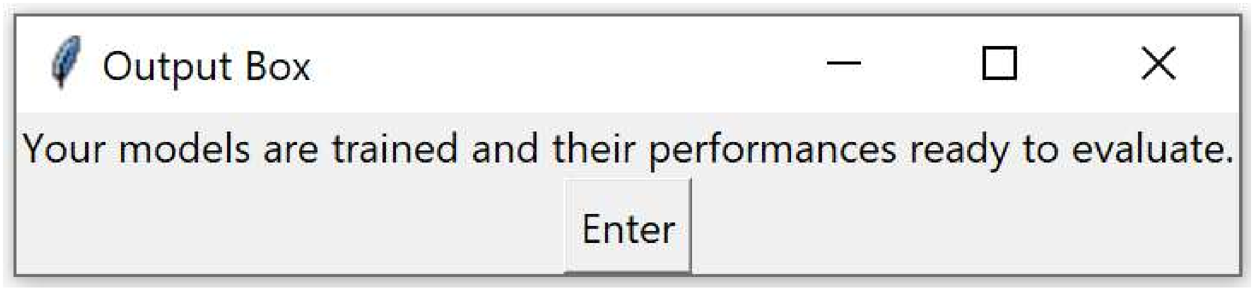
- ✓ 2-neurons-0-dense-1-conv-(1, 1)-kernel\train
- ✓ 2-neurons-0-dense-1-conv-(1, 1)-kernel\validation
- ✓ 2-neurons-0-dense-1-conv-(2, 2)-kernel\train
- ✓ 2-neurons-0-dense-1-conv-(2, 2)-kernel\validation
- ✓ 2-neurons-0-dense-2-conv-(1, 1)-kernel\train
- ✓ 2-neurons-0-dense-2-conv-(1, 1)-kernel\validation
- ✓ 2-neurons-0-dense-2-conv-(2, 2)-kernel\train
- ✓ 2-neurons-0-dense-2-conv-(2, 2)-kernel\validation
- ✓ 4-neurons-0-dense-1-conv-(1, 1)-kernel\train
- ✓ 4-neurons-0-dense-1-conv-(1, 1)-kernel\validation

TOGGLE ALL RUNS

logs

	Name	Smoothed	Value	Step	Time	Relative
●	2-neurons-0-dense-1-conv-(1, 1)-kernel\train	0.6899	0.7173	9	Thu Feb 22, 11:47:53	12s
●	2-neurons-0-dense-1-conv-(1, 1)-kernel\validation	0.5316	0.5372	9	Thu Feb 22, 11:47:53	12s
●	2-neurons-0-dense-1-conv-(2, 2)-kernel\train	0.7621	0.8069	9	Thu Feb 22, 11:48:14	13s
●	2-neurons-0-dense-1-conv-(2, 2)-kernel\validation	0.5838	0.5926	9	Thu Feb 22, 11:48:14	13s
●	2-neurons-0-dense-2-conv-(1, 1)-kernel\train	0.574	0.5939	9	Thu Feb 22, 11:48:28	9s
●	2-neurons-0-dense-2-conv-(1, 1)-kernel\validation	0.5265	0.5372	9	Thu Feb 22, 11:48:28	9s
●	2-neurons-0-dense-2-conv-(2, 2)-kernel\train	0.6122	0.6442	9	Thu Feb 22, 11:48:49	14s
●	2-neurons-0-dense-2-conv-(2, 2)-kernel\validation	0.5605	0.5745	9	Thu Feb 22, 11:48:49	14s
●	2-neurons-1-dense-1-conv-(1, 1)-kernel\train	0.8883	0.94	9	Thu Feb 22, 11:51:39	9s
●	2-neurons-1-dense-1-conv-(1, 1)-kernel\validation	0.5531	0.5553	9	Thu Feb 22, 11:51:39	9s
●	2-neurons-1-dense-1-conv-(2, 2)-kernel\train	0.8532	0.8979	9	Thu Feb 22, 11:51:57	12s
●	2-neurons-1-dense-1-conv-(2, 2)-kernel\validation	0.5905	0.5971	9	Thu Feb 22, 11:51:57	12s
●	2-neurons-1-dense-2-conv-(1, 1)-kernel\train	0.07349	0.07357	9	Thu Feb 22, 11:52:16	11s
●	2-neurons-1-dense-2-conv-(1, 1)-kernel\validation	0.05305	0.05305	9	Thu Feb 22, 11:52:16	11s
●	2-neurons-1-dense-2-conv-(2, 2)-kernel\train	0.6881	0.7227	9	Thu Feb 22, 11:52:43	18s
●	2-neurons-1-dense-2-conv-(2, 2)-kernel\validation	0.6031	0.6208	9	Thu Feb 22, 11:52:43	18s
●	2-neurons-2-dense-1-conv-(1, 1)-kernel\train	0.07349	0.07357	9	Thu Feb 22, 11:56:17	11s
●	2-neurons-2-dense-1-conv-(1, 1)-kernel\validation	0.05311	0.05305	9	Thu Feb 22, 11:56:17	11s
●	2-neurons-2-dense-1-conv-(2, 2)-kernel\train	0.9086	0.9613	9	Thu Feb 22, 11:56:36	12s
●	2-neurons-2-dense-1-conv-(2, 2)-kernel\validation	0.5928	0.5993	9	Thu Feb 22, 11:56:36	12s

Finally, an output box is shown:


Output Box

Your models are trained and their performances ready to evaluate.

Word count : 1200