Concepts and Technologies of Artificial Intelligence

Assignment 2: Random Search Optimisation and Meta Learning

Vadim Stasiev - 1900872

2021

Table of Contents

[Introduction 2](#_Toc72111309)

[What’s covered? 2](#_Toc72111310)

[Designing a Solution 2](#_Toc72111311)

[Unexpected Issues 5](#_Toc72111312)

[Experiments 7](#_Toc72111313)

[Conclusion 10](#_Toc72111314)

[References 11](#_Toc72111315)

[Appendix 12](#_Toc72111316)

# Introduction

The success of a machine learning solution often depends heavily on the choice of good hyper-parameters.

## What’s covered?

This report addresses the objectives of the second assignment which serve as an improvement to the solution developed in the first, more precisely, the brief outlines the need to use one or more strategies to explore the different combinations of hyper parameters, ideally attain one that yields better results. The results that are being sought are ones that surpass the previously developed recognition solution in at least efficiency if not accuracy, given that the accuracy was already high.

# Designing a Solution

As mentioned above, for this assignment it is required the use of at least one of the following suggested strategies:

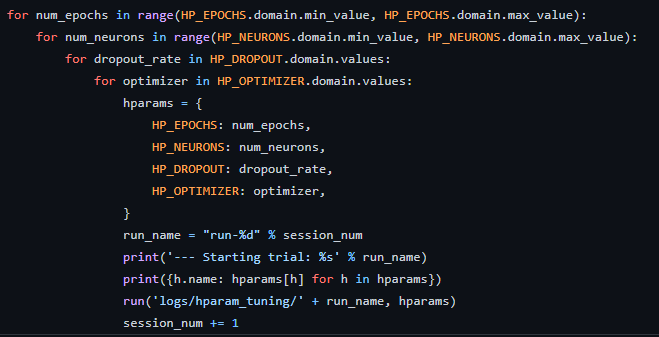
* Random Search
  + Random search is a technique through which random hyper parameter values are tested using a uniform random parameter generator, which ensures that random values are generated uniformly which ensures that different values are tested. (Bengio & Bergstra, 2011)
* Meta Learning
  + In this context, this is referring to the use of an existing model that was trained to optimize the hyper parameters of many different models to optimize the current model. (Hospedales, et al., 2020)
* Adaptive Boosting
  + Also known as “AdaBoost”, it is a machine learning technique that aims to fit models with repeatedly modified versions of the data where the data modifications are made to increase the chances of the model correctly classifying previously incorrect classifications. (sci-kit learn, 2020)
* Cascade Correlation
  + This is a supervised learning algorithm that instead of using a fixed topology, it starts off with a minimal network and automatically trains and adds new hidden layers one by one. This allows the network to learn quickly as well as determine its own size and topology. (Fahlman & Lebiere, 1993)

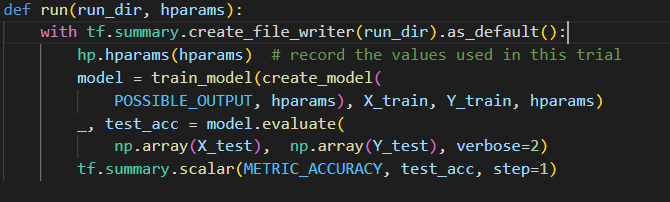
The original solution used an exhaustive search, also known as “grid search”, this was previously used to find the configuration with the best performing number of epochs and neurons, however, for this assignment, it was decided to also add a dropout layer and use its value as another hyper parameter as well as adding the ‘sgd’ and ‘nadam’ as alternative values for the optimizer, which would also be tested individually during the hyper parameter evaluation.

To start with, it was decided to rewrite the grid search to also include the aforementioned added parameters, to recap on what that is and does, grid search is the process through which it is attempted every single combination of the different hyper parameters. It works iteratively through all of the different combinations, making this the ideal, yet unpractical, solution for finding the absolute best hyper parameter combination.

This time around the implementation also revolved on using a tool provided by “TensorFlow” named “TensorBoard”. (TensorFlow, 2021) This tool provides a dashboard for viewing all the results of all of the different runs for easy and fast comparison, however the code had to also be adapted to generate compatible data that can actually be interpreted by this tool.

Bellow it is shown the reimplementation of the grid search to generate the mentioned compatible data for interpretation by “TensorBoard”.

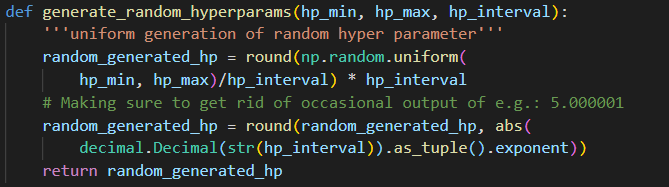




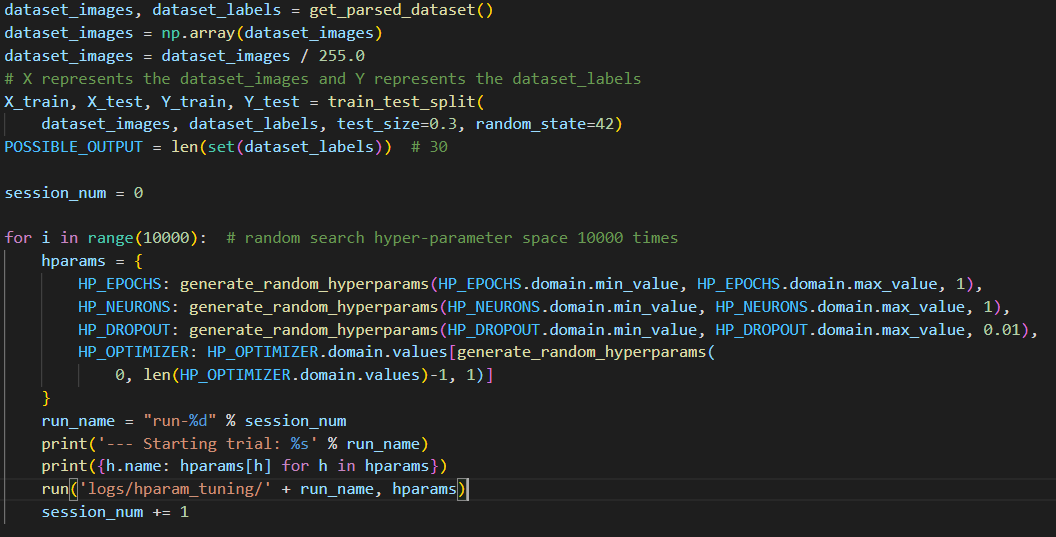
Above it can be seen that all the parameters are being generated and ran iteratively, the function “run” creates the model with the respective parameters and records all the relevant information to the disk.

As already mentioned, the grid search technique is unpractical and this time around due to the addition of the previously mentioned hyper parameters the run took too long so it was eventually cancelled, this also proved the point as to why this technique has hardly any real-world application since any modern machine learning solution, that serves any kind of purpose, would have at least this level of complexity as far as hyper parameters go.

Moving on to using at least one of the suggested strategies, for this assignment, it was decided to use “Random Search” since the implementation of it seemed fairly straight-forward and promising, as long as the parameter values would be uniformly generated such as to ensure no single value was tested more than the others. (scikit learn, 2021) In order to do this, the following function was created:



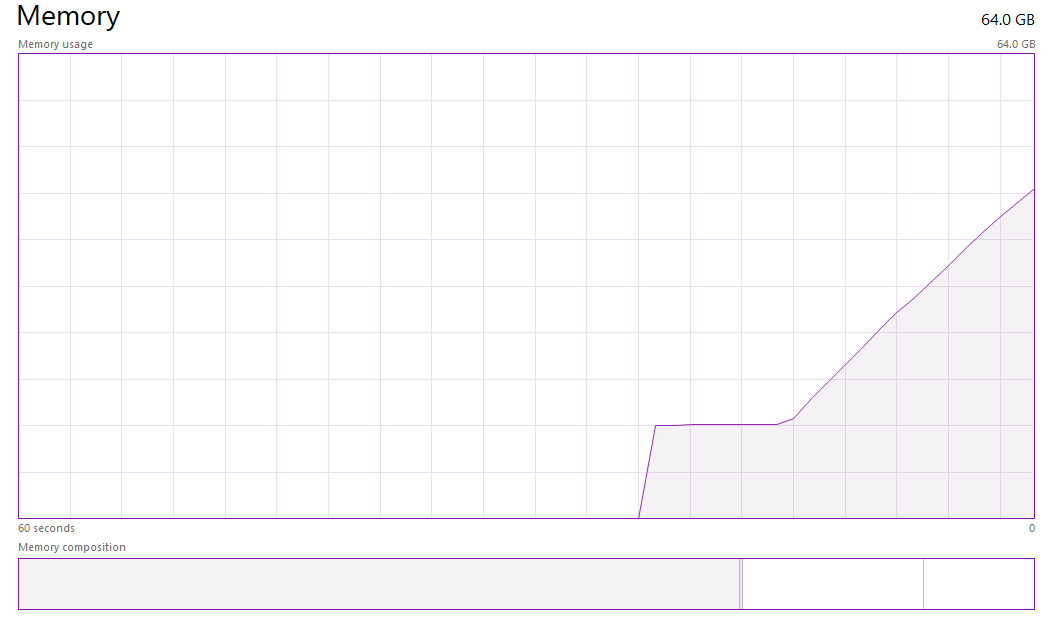
The function was then implemented in the following way:



As it can be seen above, there is a for loop that runs 10,000 times and over each run, different random hyper-parameters are generated uniformly, the run results are then also saved for later analysis.

## Unexpected Issues

Despite looking like a promising tool, TensorBoard appears to have a bug that causes a memory leak which according to the community has been present in the software for at least a few years (TensorFlow GitHub, 2021), and has not been fixed yet, this seems to be triggered whenever TensorBoard has to load a lot of logs, the number of logs tested in here were roughly 10,000 and it seemed to cause the program to fill up the entire system memory (RAM) in the space of a few seconds, naturally if not stopped, the memory leak would then overflow onto to windows paging file. This issue might be related to the windows NTFS file system as it is a known file system to have many issues when dealing with multiple directories nested within a single one but that was not pinpointed, it was instead found a work-around, to work around the issue the “TensorBoard” software needs to be started on an empty directory and then after the fact, move the logs into the working directory.



# Experiments

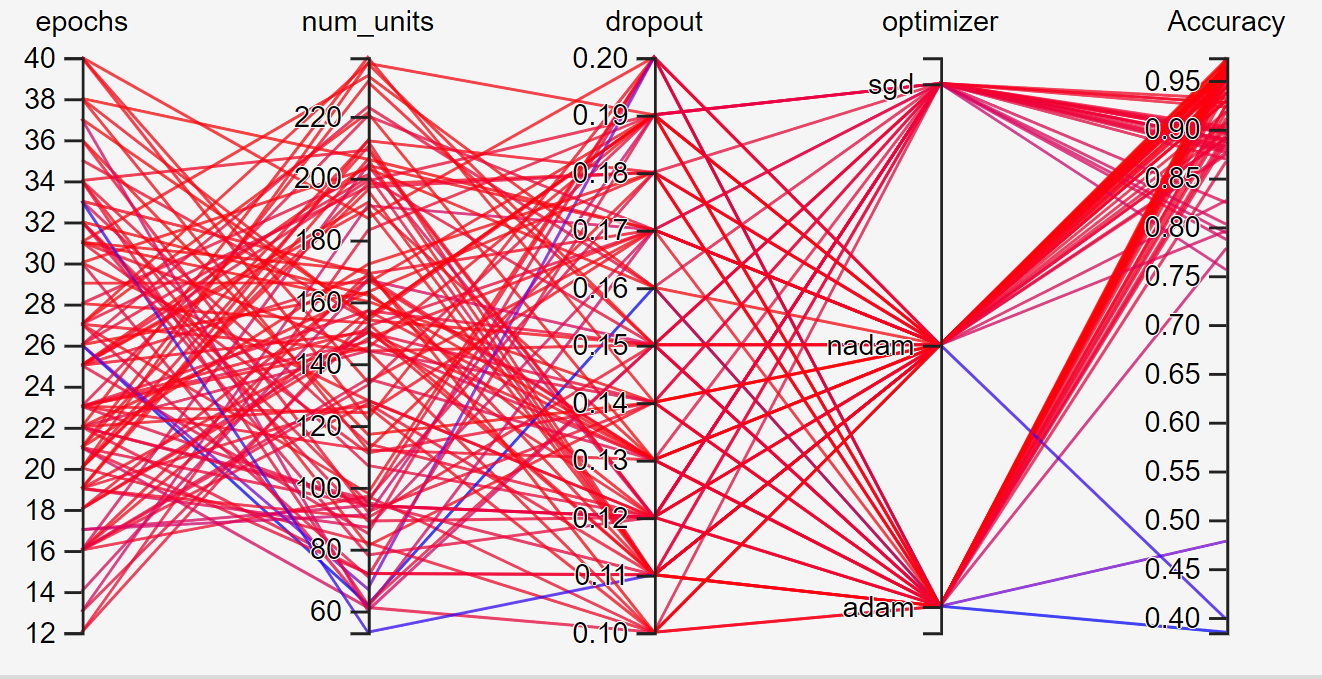
As previously mentioned, the experiments conducted using grid search were cancelled so they will not be included.

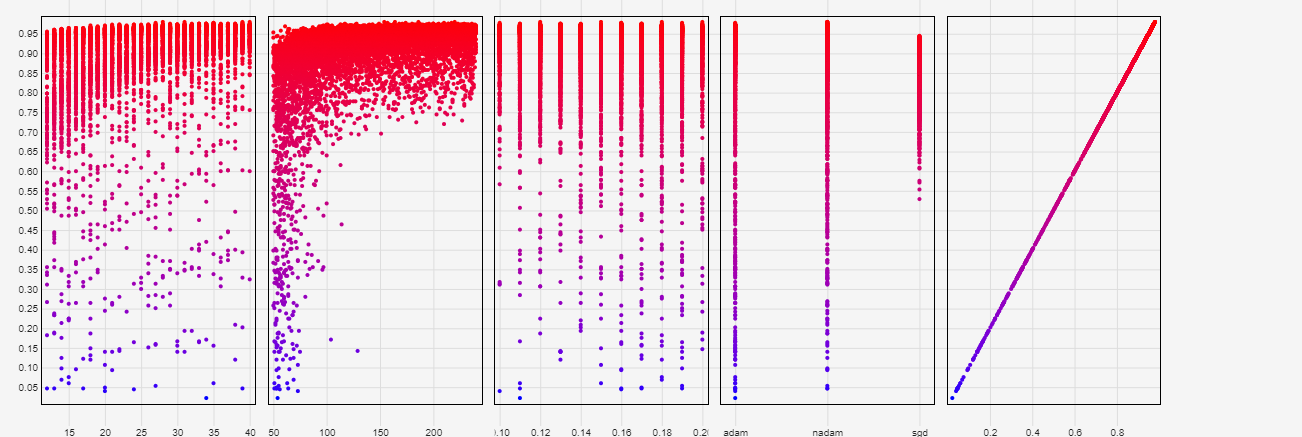
As for the random search experiment, this was concluded with 10,000 runs with the following tested values per parameter:

|  |  |
| --- | --- |
| Epochs | From 12 to 40 |
| Neurons | From 50 to 240 |
| Dropout Rate | 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.17, 0.18, 0.19, 0.2 |
| Optimizer | 'adam', 'sgd', 'nadam' |

As these same values were used in the original grid search, math would suggest that it would have resulted in a staggering 238,260‬ runs, so while 10,000 runs may seem much, it actually equates to roughly only 4% of the total number of combination possibilities.

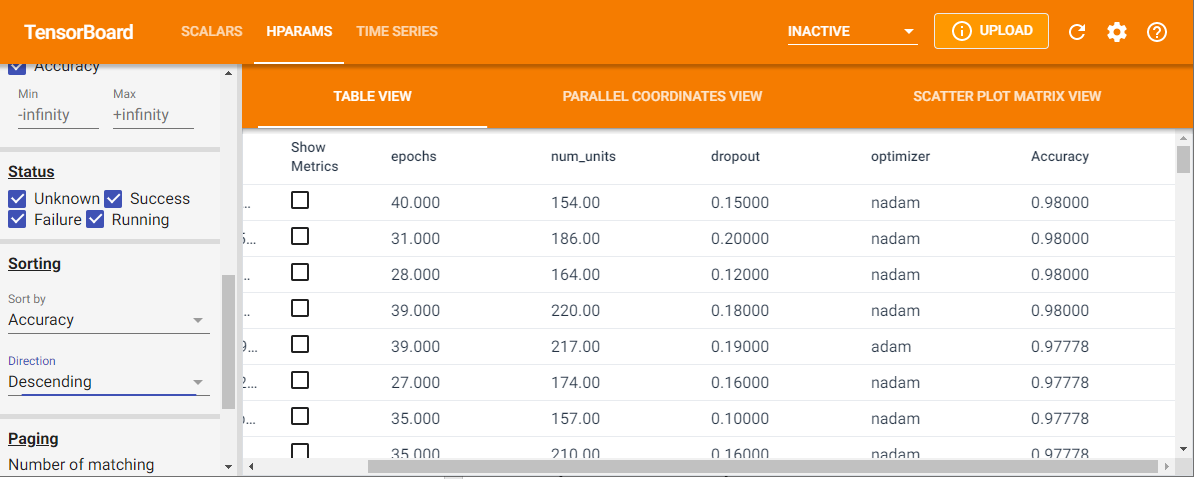
The resulting graph of all the runs can be seen bellow:





While the graphs above can give a rough idea of how the different parameters affected the model’s accuracy, it looks fairly dense and therefore it is hard to drawn conclusions from it, to do that we must instead look at the top results and compare them to each other.

On the TensorBoard, with all the runs selected, under the HPARAMS tab, there is some functionality that allows the data to be sorted, in order to see what the best results were, this data needs to be sorted by the accuracy in a descending direction.



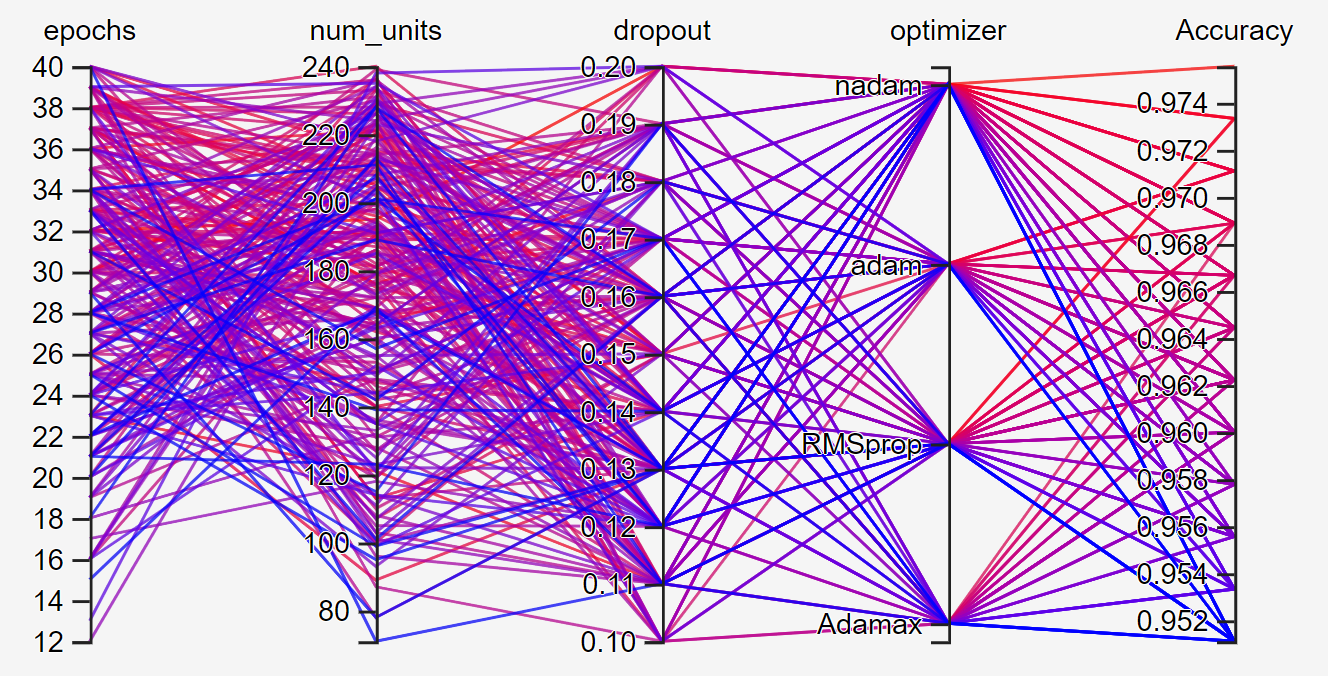
On the resulting table it can be seen the best results of all of the 10,000 experimental runs and while it may not always be the case, in this case, the top result seems to not only have the same accuracy as the few before it, but it also does so by being the most efficient one as it has the least number of hidden neurons, hitting an accuracy of 98% with only 154 neurons.

Another experiment was conducted using more optimizers but only with 1,000 runs, and while due to this small amount of runs, generation a small amount of different combinations, the findings of these results cannot be taken with certainty.

More specifically, the parameters tested were:

|  |  |
| --- | --- |
| Epochs | From 12 to 40 |
| Neurons | From 50 to 240 |
| Dropout Rate | 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.17, 0.18, 0.19, 0.2 |
| Optimizer | 'adam', 'nadam', “RMSprop”, “Adamax” |

The resulting accuracy graph was as follows:



This time, the accuracy was filtered to only show results with accuracies above 95% and as it can be seen from the small sample, nadam continues to reach the highest accuracy from the run samples, however it seems that the newly tested “RMSprop” optimizer produced a run with an accuracy higher than any of the “adam” combinations, which in the previous testing seemed to be highly competitive with “nadam”, this might mean that there might be a combination where the “RMSprop” optimizer might actually surpass the current highest accuracy which uses “nadam”.

# Conclusion

After many failed efforts and strange computing occurrences the requested code was finally produced and likewise the experiments ran, therefore fulfilling the requirements of the task.

With the ending of the experiments it concludes the assignment task, it also concludes the fact that using strategies that don’t require brute force to find answers are far more effective, not to mention, recommended, similarly, other untried strategies could mean leaps in the progress of improving the solution.

To improve on this work there are further steps to take, that include: meta-learning, adaptive-boosting and other machine learning techniques, not to mention further tweaking to the model in an effort to finding the ideal architecture that produces an accuracy as close to the truth as possible, or perhaps even more plausibly, a high accuracy but efficient model that is targeted to run on portable low-consumption hardware.

# References

Bengio, Y. & Bergstra, J., 2011. *Random Search for Hyper-Parameter Optimization,* p. 25.

Fahlman, S. E. & Lebiere, C., 1993. *The Cascade-Correlation Learning Architecture,* p. 10.

Hospedales, T., Antoniou, A., Micaelli, P. & Storkey, A., 2020. *Meta-Learning in Neural Networks: A Survey,* p. 20.

sci-kit learn, 2020. *AdaBoost.* [Online]   
Available at: https://scikit-learn.org/stable/modules/ensemble.html#adaboost  
[Accessed 08 05 2021].

scikit learn, 2021. [Online]   
Available at: https://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.RandomizedSearchCV.html  
[Accessed 10 05 2021].

TensorFlow GitHub, 2021. [Online]   
Available at: https://github.com/tensorflow/tensorboard/issues/766  
[Accessed 10 05 2021].

TensorFlow, n.d. [Online]   
Available at: https://www.tensorflow.org/tensorboard  
[Accessed 08 05 2021].

# Appendix

import decimal

import sys

from os import listdir

from matplotlib import image

from matplotlib import pyplot

import numpy as np

from numpy import asarray

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

import tensorflow as tf

from tensorflow import keras

import cv2

import matplotlib.pyplot as plt

from tensorboard.plugins.hparams import api as hp

tf.get\_logger().setLevel('INFO')

PATH = "."

DNAME = "/dataset"

FILES = listdir(PATH + DNAME)

N\_FILES = len(FILES)

IM1 = image.imread(PATH + DNAME + '/' + FILES[0])

N\_PIXELS = IM1.shape[0]\*IM1.shape[1]

# HYPER PARAMETERS

HP\_EPOCHS = hp.HParam('epochs', hp.IntInterval(12, 40))

HP\_NEURONS = hp.HParam('num\_units', hp.IntInterval(50, 240))

HP\_DROPOUT = hp.HParam('dropout', hp.RealInterval(0.1, 0.2))

HP\_OPTIMIZER = hp.HParam('optimizer', hp.Discrete(['adam', 'sgd', 'nadam']))

METRIC\_ACCURACY = 'accuracy'

with tf.summary.create\_file\_writer('logs/hparam\_tuning').as\_default():

    hp.hparams\_config(

        hparams=[HP\_EPOCHS, HP\_NEURONS, HP\_DROPOUT, HP\_OPTIMIZER],

        metrics=[hp.Metric(METRIC\_ACCURACY, display\_name='Accuracy')],

    )

def get\_parsed\_dataset():

    images = []

    labels = []

    for i in range(0, N\_FILES):

        filename = FILES[i]

        img = cv2.imread(PATH + DNAME + '/' + filename, cv2.IMREAD\_GRAYSCALE)

        images.append(img)

        lbl = int(filename[5:7])

        labels.append(lbl)

    temp\_labels = labels.copy()

    for n in range(len(set(labels))-1, -1, -1):

        current\_max = max(temp\_labels)

        for i in range(len(labels)-1, -1, -1):

            if temp\_labels[i] == current\_max:

                temp\_labels[i] = -1

                labels[i] = n

    return (images, labels)

def create\_model(POSSIBLE\_OUTPUT, hparams):

    model = keras.Sequential([

        keras.layers.Flatten(input\_shape=(

            IM1.shape[0], IM1.shape[1])),  # width and height

        keras.layers.Dense(hparams[HP\_NEURONS], activation='relu'),

        keras.layers.Dropout(hparams[HP\_DROPOUT]),

        keras.layers.Dense(POSSIBLE\_OUTPUT)  # (30) Possible Output Options

    ])

    model.compile(optimizer=hparams[HP\_OPTIMIZER],

                  loss=keras.losses.SparseCategoricalCrossentropy(

                      from\_logits=True),

                  metrics=['accuracy'])

    return model

def train\_model(model, train\_data, train\_labels, hparams):

    model.fit(x=np.array(train\_data),

              y=np.array(train\_labels),

              epochs=hparams[HP\_EPOCHS],

              )

    return model

def plot\_image(i, predictions\_array, true\_label, img):

    true\_label, img = true\_label[i], img[i]

    plt.grid(False)

    plt.xticks([])

    plt.yticks([])

    plt.imshow(img, cmap=plt.cm.binary)

    predicted\_label = np.argmax(predictions\_array)

    if predicted\_label == true\_label:

        color = 'blue'

    else:

        color = 'red'

    plt.xlabel("{} {:2.0f}% ({})".format([predicted\_label],

                                         100\*np.max(predictions\_array),

                                         [true\_label]),

               color=color)

def plot\_value\_array(predictions\_array):

    plt.grid(False)

    x\_range = range(len(predictions\_array))

    plt.xticks(x\_range)

    plt.yticks([])

    thisplot = plt.bar(x\_range,

                       predictions\_array, color="#777777")

    plt.ylim([0, 1])

    predicted\_label = np.argmax(predictions\_array)

    thisplot[predicted\_label].set\_color('red')

def generate\_random\_hyperparams(hp\_min, hp\_max, hp\_interval):

    '''uniform generation of random hyper parameter'''

    random\_generated\_hp = round(np.random.uniform(

        hp\_min, hp\_max)/hp\_interval) \* hp\_interval

    # Making sure to get rid of occasional output of e.g.: 5.000001

    random\_generated\_hp = round(random\_generated\_hp, abs(

        decimal.Decimal(str(hp\_interval)).as\_tuple().exponent))

    return random\_generated\_hp

def run(run\_dir, hparams):

    with tf.summary.create\_file\_writer(run\_dir).as\_default():

        hp.hparams(hparams)  # record the values used in this trial

        model = train\_model(create\_model(

            POSSIBLE\_OUTPUT, hparams), X\_train, Y\_train, hparams)

        \_, test\_acc = model.evaluate(

            np.array(X\_test),  np.array(Y\_test), verbose=2)

        tf.summary.scalar(METRIC\_ACCURACY, test\_acc, step=1)

if \_\_name\_\_ == "\_\_main\_\_":

    dataset\_images, dataset\_labels = get\_parsed\_dataset()

    dataset\_images = np.array(dataset\_images)

    dataset\_images = dataset\_images / 255.0

    # X represents the dataset\_images and Y represents the dataset\_labels

    X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(

        dataset\_images, dataset\_labels, test\_size=0.3, random\_state=42)

    POSSIBLE\_OUTPUT = len(set(dataset\_labels))  # 30

    session\_num = 0

    for i in range(10000):  # random search hyper-parameter space 10000 times

        hparams = {

            HP\_EPOCHS: generate\_random\_hyperparams(HP\_EPOCHS.domain.min\_value, HP\_EPOCHS.domain.max\_value, 1),

            HP\_NEURONS: generate\_random\_hyperparams(HP\_NEURONS.domain.min\_value, HP\_NEURONS.domain.max\_value, 1),

            HP\_DROPOUT: generate\_random\_hyperparams(HP\_DROPOUT.domain.min\_value, HP\_DROPOUT.domain.max\_value, 0.01),

            HP\_OPTIMIZER: HP\_OPTIMIZER.domain.values[generate\_random\_hyperparams(

                0, len(HP\_OPTIMIZER.domain.values)-1, 1)]

        }

        run\_name = "run-%d" % session\_num

        print('--- Starting trial: %s' % run\_name)

        print({h.name: hparams[h] for h in hparams})

        run('logs/hparam\_tuning/' + run\_name, hparams)

        session\_num += 1