LABORATÓRIO DE INSTRUMENTAÇÃO
E FÍSICA EXPERIMENTAL DE PARTÍCULAS
partículas e tecnologia

Data challenge

Data Science School 2022, Coimbra

Magda Duarte

Outline/Approach

1. Data analysis

2. Data treatment

3. Basic model assembly
    - 3.1. Layers
    - 3.2. Initial tests

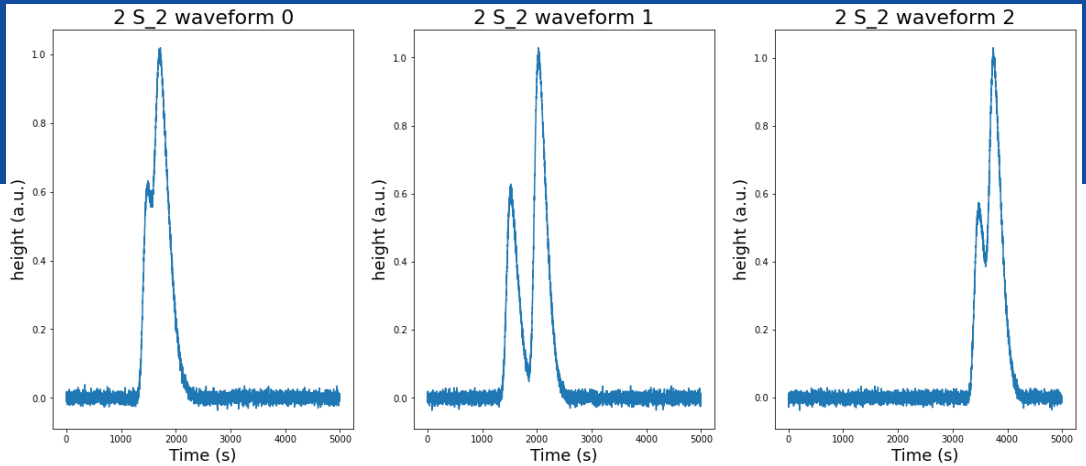
4. Optimizing the model

5. Testing on1S2 waveforms

6. 1S2 and 2S2 distinction

1. Data analysis

Check for data format and verify the waveform expected




```
Input 2S2 waveforms:  (20000, 5000)
Output 2S2 d:  (20000,)
# waveforms:  20000
# points/waveform:  5000
```

2. Data treatment

Divide into train and validation/'test' batches

```
X_train, X_test, y_train, y_test = train_test_split(input2S2_data, output2S2_data, test_size=0.2, random_state=0)
```

```
Number of waveforms for training:  16000
Number of points/waveform for training:  5000
Number of waveforms for test:  4000
Number of points/waveform for test:  5000
```

2. Data treatment

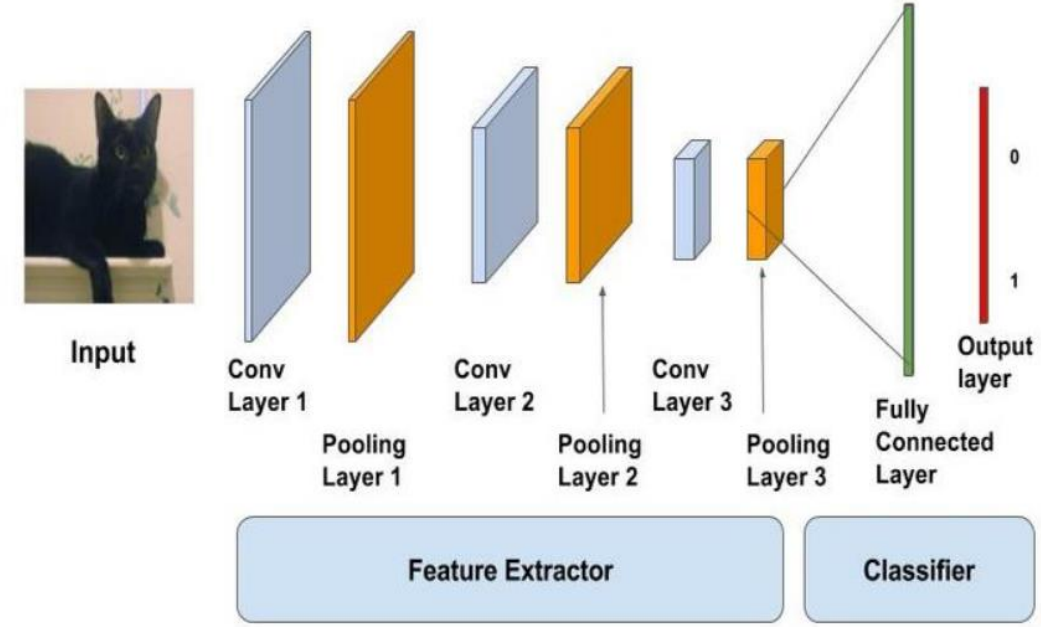
Normalized input data X

$$X = \frac{X - \min(X)}{\max(X) - \min(X)}$$

3. Basic model assembly

Stacking of convolutional, pooling, FCL and output layers

Convolutional neural network



Convolutional neural network

1D Convolution layer

- Receives input with shape (# of waveforms, # timesteps, # ‘features’)

```
X_train3 = X_train.reshape(-1, n_points, 1)
print('X_train after reformating: ', np.shape(X_train3))
X_test3 = X_test.reshape(-1, n_points, 1)
print('X_test after reformating: ', np.shape(X_test3))
```

```
X_train after reformating:  (16000, 5000, 1)
X_test after reformating:  (4000, 5000, 1)
```

Convolutional neural network

1D Convolution layer

- Characterized by number of filters it must use to extrapolate features, kernel size and stride. For smaller feature ‘maps’:
  -> Greater kernel size
  -> Greater stride
  -> More convolutional layers

```
#adding convolutional layer
model.add(Conv1D(32,3,1,activation='relu',input_shape = [n_train_points, 1]))
```

- Started with ‘standard’ values

Convolutional neural network

Pooling layer

```
model.add(MaxPool1D(16))
```

Fully connected layer

```
model.add(Flatten())
model.add(Dense(100,activation='relu'))
```

Output layer

```
model.add(Dense(1,activation='sigmoid'))
```

Regression model evaluation

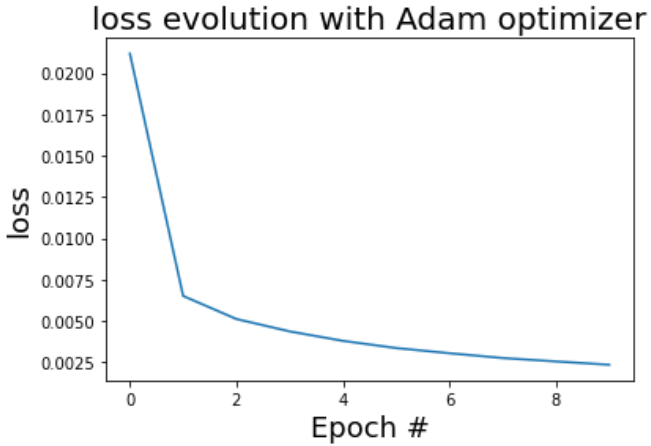
```
import keras.backend as K

def my_acc(y_true, y_pred):
    return K.mean(K.equal(K.round(y_true*10), K.round(y_pred*10)))
```

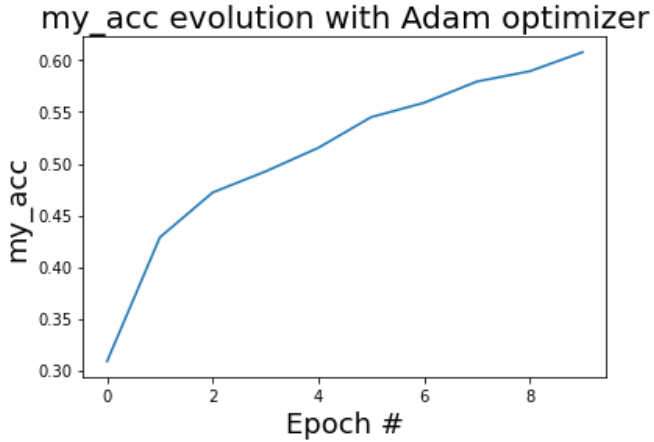
```
optimizer = RMSprop(learning_rate=0.001)

model.compile(optimizer,loss='mean_squared_error', metrics = [my_acc])
```

Initial runs

```
Epoch 1/10
125/125 [==============================] - 38s 294ms/step - loss: 0.0025 - my_acc: 0.6159 - val_loss: 0.0034 - val_my_acc: 0.5439
Epoch 2/10
125/125 [==============================] - 36s 288ms/step - loss: 0.0022 - my_acc: 0.6259 - val_loss: 0.0022 - val_my_acc: 0.6267
Epoch 3/10
125/125 [==============================] - 37s 296ms/step - loss: 0.0020 - my_acc: 0.6489 - val_loss: 0.0022 - val_my_acc: 0.5928
Epoch 4/10
125/125 [==============================] - 37s 299ms/step - loss: 0.0019 - my_acc: 0.6491 - val_loss: 0.0031 - val_my_acc: 0.5640
Epoch 5/10
125/125 [==============================] - 37s 298ms/step - loss: 0.0018 - my_acc: 0.6544 - val_loss: 0.0019 - val_my_acc: 0.6609
Epoch 6/10
125/125 [==============================] - 35s 284ms/step - loss: 0.0017 - my_acc: 0.6731 - val_loss: 0.0011 - val_my_acc: 0.7324
Epoch 7/10
125/125 [==============================] - 36s 289ms/step - loss: 0.0016 - my_acc: 0.6734 - val_loss: 0.0012 - val_my_acc: 0.7212
Epoch 8/10
125/125 [==============================] - 37s 296ms/step - loss: 0.0016 - my_acc: 0.6826 - val_loss: 0.0010 - val_my_acc: 0.7393
Epoch 9/10
125/125 [==============================] - 35s 277ms/step - loss: 0.0015 - my_acc: 0.6835 - val_loss: 0.0014 - val_my_acc: 0.6816
Epoch 10/10
125/125 [==============================] - 35s 283ms/step - loss: 0.0014 - my_acc: 0.6886 - val_loss: 9.8781e-04 - val_my_acc: 0.7361
```

Initial runs





Initial runs

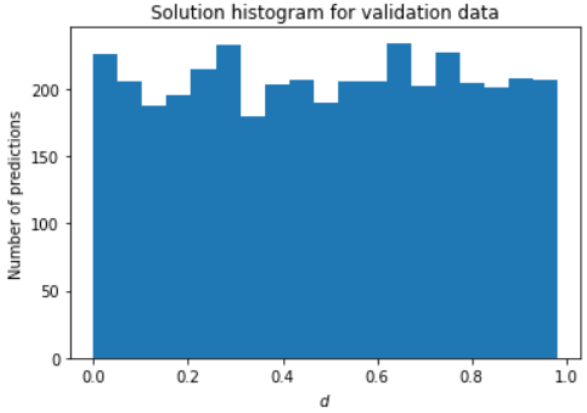
```
# ------ Check the performance of the model

score = model.evaluate(X_test3, y_test, verbose=1)
print('Test loss:', score[0])
```

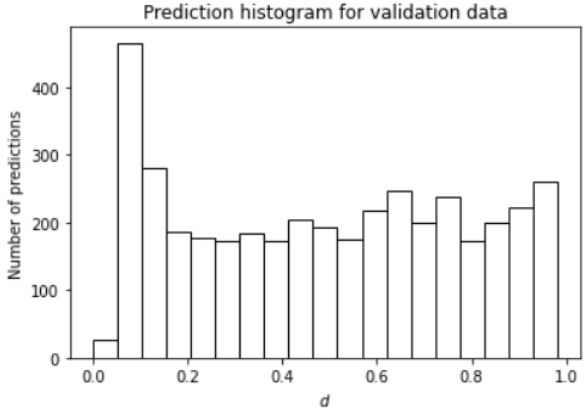
```
125/125 [==============================] - 4s 31ms/step - loss: 9.8781e-04 - my_acc: 0.7387
Test loss: 0.000987810781225562
```

Initial runs

```
[[0.9295618 ]
 [0.13522542]
 [0.80563235]
 ...
 [0.72264767]
 [0.06762588]
 [0.76344526]]
```

```
[[0.97   ]
 [0.2225 ]
 [0.80875]
 ...
 [0.71625]
 [0.0475 ]
 [0.7525 ]]
```



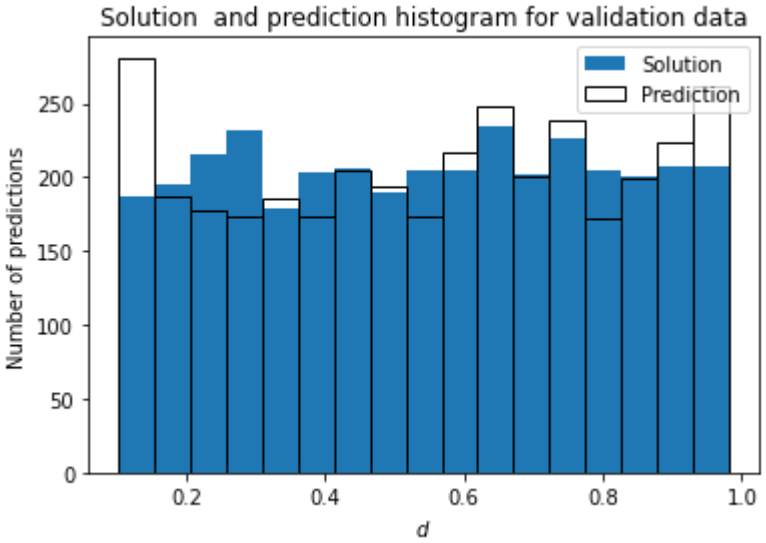


Initial runs

```
[[0.9295618 ]
 [0.13522542]
 [0.80563235]
 ...
 [0.72264767]
 [0.06762588]
 [0.76344526]]
```

```
[[0.97   ]
 [0.2225 ]
 [0.80875]
 ...
 [0.71625]
 [0.0475 ]
 [0.7525 ]]
```

Zooming in



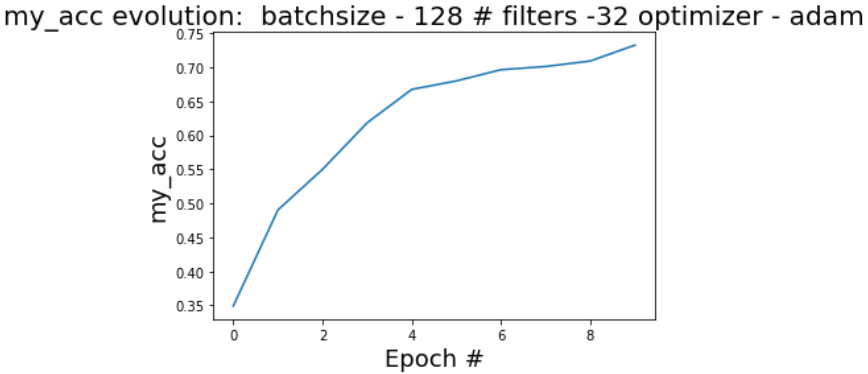
4. Optimizing the model

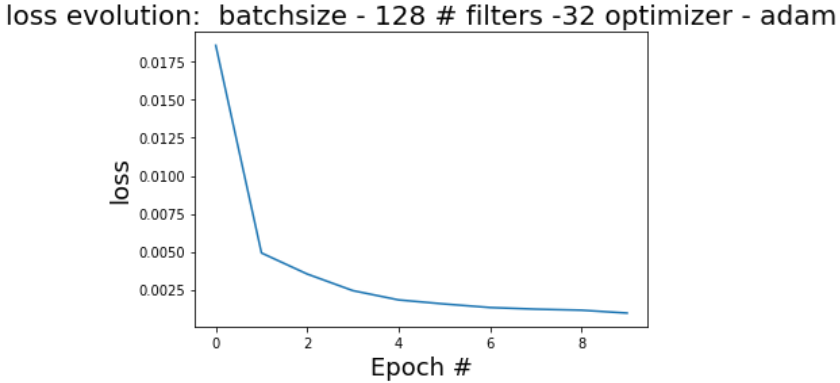
2+D Grid search

- Experimented different:
  - Number of filters on convolution layer
  - Optimizers

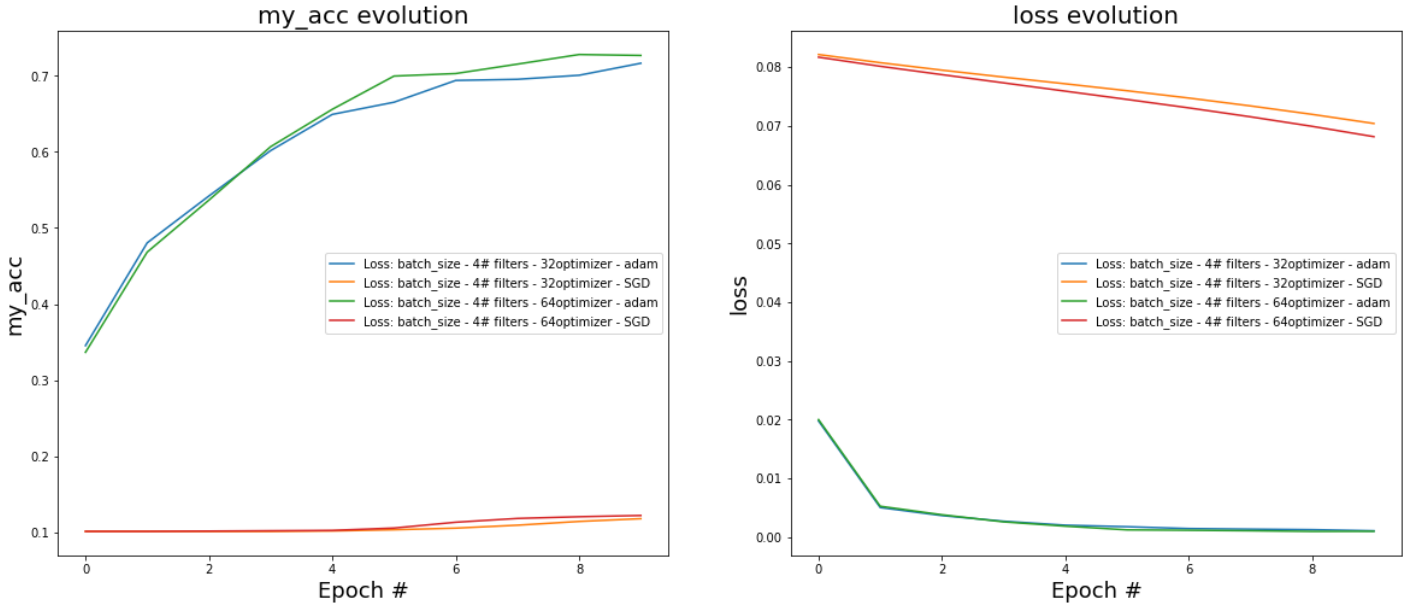
```
def Optimize_param(X_train, y_train, X_test, y_test, n_train_points):
  batch_sizes = [4, 32, 128]
  n_filters = [16, 32, 64]
  n_ConvFilters = [16, 32]
  kernel_sizes = [3]
  optimizers = ['adam', 'SGD', 'RMSprop']
  histories = []

  TrialNumber = 0

  global best_model
  best_model = [0,0,0,'optimizer']
  global best_metrics
  best_metrics = [10**5, 0.0]
  global best_trial
  best_trial = 0

  for bs1 in batch_sizes:
    for nf in n_filters:
      for ks in kernel_sizes:
        for os in optimizers:
```

2+D Grid search - results



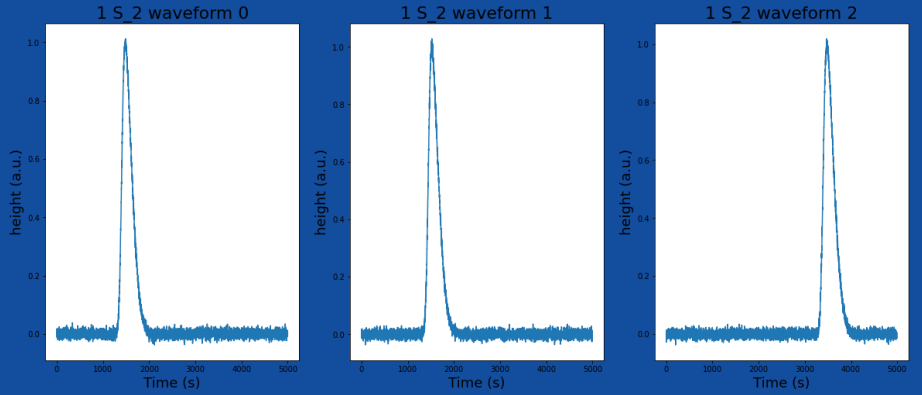


2+D Grid search – best model

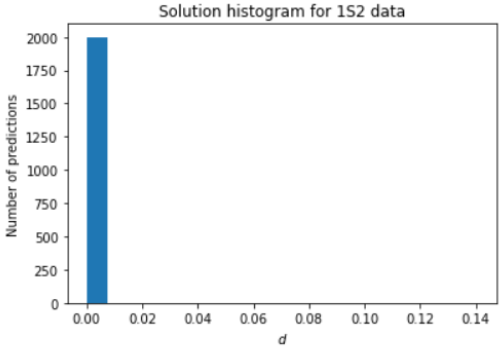


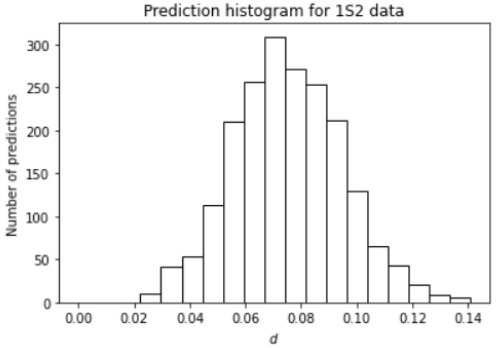
5. Testing on 1S2 waveforms

Checked waveforms and normalized them



Preliminary results – not the optimal parameters



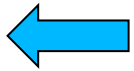


6. 1S2 and 2S2 distinction

Creation of new labels and reformating

```
#equal number of 2S2 and 1S2 curves with d = 0

#retrieve 2S2 waveforms with d = 0
X2S2_d0 = X_train3[y_train2[:,0]<0.01]
#create labels
N = 2000
y2S2 = np.zeros((N,1))
y2S2[:,0] = 2
y1S2 = np.zeros((N,1))
y1S2[:,0] = 1
X2S2_d0 = X2S2_d0[:N]

max_ = np.shape(X2S2_d0)[0]

#concatenate
y_final = np.concatenate((y2S2[:max_,:], y1S2[:max_,:]), axis = 0)
X_final = np.concatenate((X2S2_d0[0:max_,:,:], X_test1S2[0:max_,:,:]), axis = 0)

#shuffle
y_final_ = y_final.reshape(2*max_, 1, 1)
together = np.concatenate((X_final, y_final_), axis = 1)
np.random.shuffle(together)

#retrieve
X_Final = together[:,:5000,:]
y_Final = together[:,5000,:]
print(np.shape(y_Final))
print(np.shape(X_Final))
#print(y_Final)

#split
X_final_train, X_final_test, y_final_train, y_final_test = train_test_split(X_Final, y_Final, test_size=0.2, random_state=0)
```

```
y_FFinal_train = keras.utils.to_categorical(y_final_train.values, 2)
y_FFinal_test = keras.utils.to_categorical(y_final_test.values, 2)
```



Going back to categorical classification

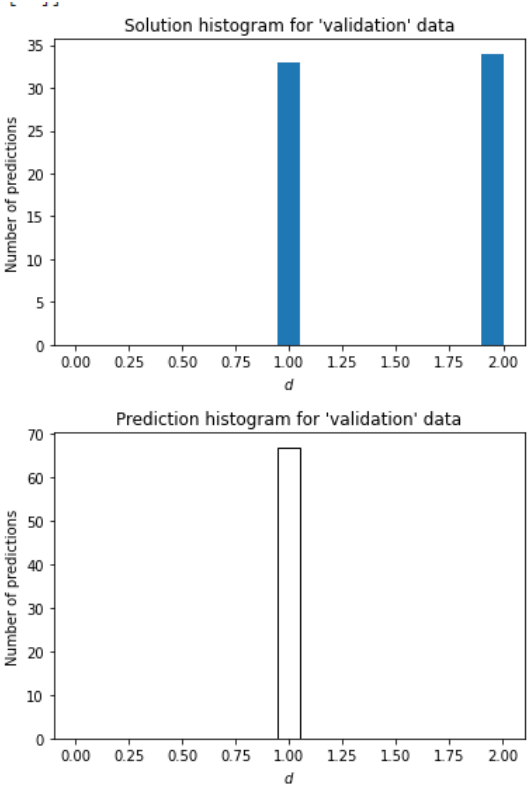
Again, creating and training the model

```
early_stopping_monitor = EarlyStopping(monitor='val_loss', min_delta=0.001, patience=5, verbose=1, mode='auto')
```

```
optimizer = Adam(learning_rate=0.001)

model.compile(optimizer,loss='mean_squared_error', metrics = 'categorical_accuracy')
```

```
# ------ training the model (fitting the data)
history = model.fit(X_final_train, y_FFinal_train,
                    batch_size=4,
                    epochs=10,
                    verbose=1,
                    callbacks=[early_stopping_monitor],
                    validation_data=(X_final_test, y_FFinal_test))
```

Results...


Solution histogram for 'validation' data
Number of predictions
0
5
10
15
20
25
30
35
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
d
Prediction histogram for 'validation' data
Number of predictions
0
10
20
30
40
50
60
70
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
d

Thanks!