CSCE633 Homework 01

Lu Sun

October 2020

Question 1: Maximum likelihood estimate

(a.i) Show that $\hat{\mu} = \frac{\sum\limits_{n=1}^{N} x_n}{N}$ and $\hat{\sigma}^2 = \frac{\sum\limits_{n=1}^{N} (x_n - \hat{\mu})^2}{N}$

Answer:

Assume $\chi = \{x_1, x_2, ..., x_N\}$, function $L(\mu, \sigma^2|\chi)$ is the likelihood function defined as follow:

$$L(\mu, \sigma^2|\chi) = \prod_{n=1}^{N} P(x_n|\mu, \sigma^2) \overset{x_n \sim N(\mu,\sigma^2)}{=} \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(- \frac{(x_n - \mu)^2}{2\sigma^2}\Big)$$

Then the following equivalent formula can be obtained:

$$\max_{\mu,\sigma} L(\mu, \sigma^2|\chi) \Longleftrightarrow \max_{\mu,\sigma} \log\Big(L(\mu, \sigma^2|\chi)\Big) \Longleftrightarrow \max_{\mu,\sigma} \sum_{n=1}^{N} \Big[\log(\frac{1}{\sqrt{2\pi}}) - \log\sigma + \Big(-\frac{(x_n - \mu)^2}{2\sigma^2}\Big)\Big]$$

$$\Longleftrightarrow \max_{\mu,\sigma} N\log(\frac{1}{\sqrt{2\pi}}) - N\log\sigma - \sum_{n=1}^{N} \frac{(x_n - \mu)^2}{2\sigma^2} \Longleftrightarrow \min_{\mu,\sigma} N\log\sigma + \sum_{n=1}^{N} \frac{(x_n - \mu)^2}{2\sigma^2}$$

Suppose $f(\mu, \sigma) = N\log\sigma + \sum\limits_{n=1}^{N} \frac{(x_n-\mu)^2}{2\sigma^2}$, then the following formula can be obtained:

$$\begin{aligned}
\frac{\partial f}{\partial \mu} &= \sum_{n=1}^{N} \frac{\mu - x_n}{\sigma^2} &&= \frac{N}{\sigma^2} \cdot \Big(\mu - \frac{\sum\limits_{n=1}^{N} x_n}{N}\Big) \\
\frac{\partial f}{\partial \sigma} &= \frac{N}{\sigma} - \sum_{n=1}^{N} \frac{(x_n-\mu)^2}{\sigma^3} &&= \frac{N}{\sigma^3} \cdot \Big(\sigma^2 - \frac{\sum\limits_{n=1}^{N} (x_n-\mu)^2}{N}\Big)
\end{aligned}$$

Set $\frac{\partial f}{\partial \mu} = 0$, $\frac{\partial f}{\partial \sigma} = 0$, suppose $\hat{\mu}, \hat{\sigma} \in \arg\max\limits_{\mu,\sigma} L(\mu, \sigma^2|\chi)$,finally, we can get $\hat{\mu} = \frac{\sum\limits_{n=1}^{N} x_n}{N}$ and $\hat{\sigma}^2 = \frac{\sum\limits_{n=1}^{N} (x_n-\hat{\mu})^2}{N}$

(a.ii) Using the data in $Q1\_data.csv$ and above formula, estimate μ, σ^2

Answer:

μ	1.0711573934217282
σ^2	0.08840831592001618

Table 1: Results

The corresponding code is shown as follow:

```
import numpy as np
import xlrd

# Give the location of the file
filename = ("/content/drive/My ...
     Drive/2020_FALL/CSCE633/HomeWork/HW03/Q1_Data.xlsx")
data_file = xlrd.open_workbook(filename)  # Open File "to read"
data = data_file.sheet_by_index(0)
data = np.array(data.col_values(0))
#print(data.shape)
mu = np.mean(data)
sigma_squra = np.mean((data-mu)*(data-mu))
print(mu,sigma_squra,np.std(data)**2)
```

(b) Multinomial distribution: Compute the maximum likelihood estimate of ϕ

Answer:

According to multinomial distribution's definition, $(Y_1, Y_2, Y_3) \sim Multinomia(\theta_1, \theta_2, \theta_3, N)$, where $\theta_1 := (1-\phi)^2, \theta_2 = \phi^2, \theta_3 = 2\phi(1-\phi), N = N_1 + N_2 + N_3$. Suppose $y = (N_1, N_2, N_3)$, then the corresponding likelihood function $L(\phi|N_1, N_2, N_3) := L(\theta_1, \theta_2, \theta_3|y, N)$ can be defined as follow:

$$
\begin{aligned}
L(\phi|N_1, N_2, N_3) &= L(\theta_1, \theta_2, \theta_3|y, N) := p(y|\theta_1, \theta_2, \theta_3, N) \\
&= \tfrac{(N_1+N_2+N_3)!}{N_1!N_2!N_3!}(1-\phi)^{2N_1}\phi^{2N_2}(2\phi(1-\phi))^{N_3} \\
&= \tfrac{2^{N_3}(N_1+N_2+N_3)!}{N_1!N_2!N_3!}(1-\phi)^{2N_1+N_3}\phi^{2N_2+N_3}
\end{aligned}
$$

Then the following equivalent formula can be obtained:

$$
\begin{aligned}
\max_{\phi} L(\phi|N_1, N_2, N_3) &\iff \max_{\phi} \log\Big(L(\phi|N_1, N_2, N_3)\Big) \\
&\iff \max_{\phi} \log \tfrac{2^{N_3}(N_1+N_2+N_3)!}{N_1!N_2!N_3!} + (2N_1 + N_3)\log(1-\phi) + (2N_2 + N_3)\log\phi \\
&\iff \max_{\phi} (2N_1 + N_3)\log(1-\phi) + (2N_2 + N_3)\log\phi
\end{aligned}
$$

Suppose $f(\phi) := (2N_1 + N_3)\log(1-\phi) + (2N_2 + N_3)\log\phi$, then

$$
\begin{aligned}
\frac{\partial f}{\partial \phi} &= -\tfrac{2N_1+N_3}{1-\phi} + \tfrac{2N_2+N_3}{\phi} \\
&= \tfrac{1}{\phi(1-\phi)}\Big(-2N_1\phi - N_3\phi + 2N_2 + N_3 - 2N_2\phi - N_3\phi \Big) \\
&= \tfrac{1}{\phi(1-\phi)}\Big(2N_2 + N_3 - 2(N_1+N_2+N_3)\phi \Big) \\
&= \tfrac{2(N_1+N_2+N_3)}{\phi(1-\phi)}\Big(\tfrac{2N_2+N_3}{2(N_1+N_2+N_3)} - \phi \Big)
\end{aligned}
$$

Set $\frac{\partial f}{\partial \phi} = 0$, we have $\phi = \frac{2N_2+N_3}{2(N_1+N_2+N_3)}$

Question 2: Machine learning for facial recognition

(a) Visualization:

Answer:

The code for plotting is as follow:

Listing 1: Visualization

```
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
import numpy as np

labelMap = {0:"Angry", 1:"Disgust", 2:"Fear", 3:"Happy", ...
    4:"Sad", 5:"Surprise", 6:"Neutral"}

def funcPlot(images_set,labels_set,dataName):
  fig = plt.figure(figsize=[25,25])
  index = 0
  plot_num = 0
  for i in range(len(images_set)):
    if labels_set[i] == index:
      a = fig.add_subplot(4,4,plot_num+1)
      _ = plt.imshow(images_set[i])
      a.set_title("label:{},emotion:{}\n Visualizing a random ...
          image ({}th) from {} dataset".format(
          labels_set[i],labelMap[labels_set[i]],i+1,dataName))
      plot_num +=1
      if plot_num%2 ==0:
        index +=1
    if index ==7:
      break
  plt.show()
funcPlot(train_images,train_labels,'train')
```

The plot is shown as follow:

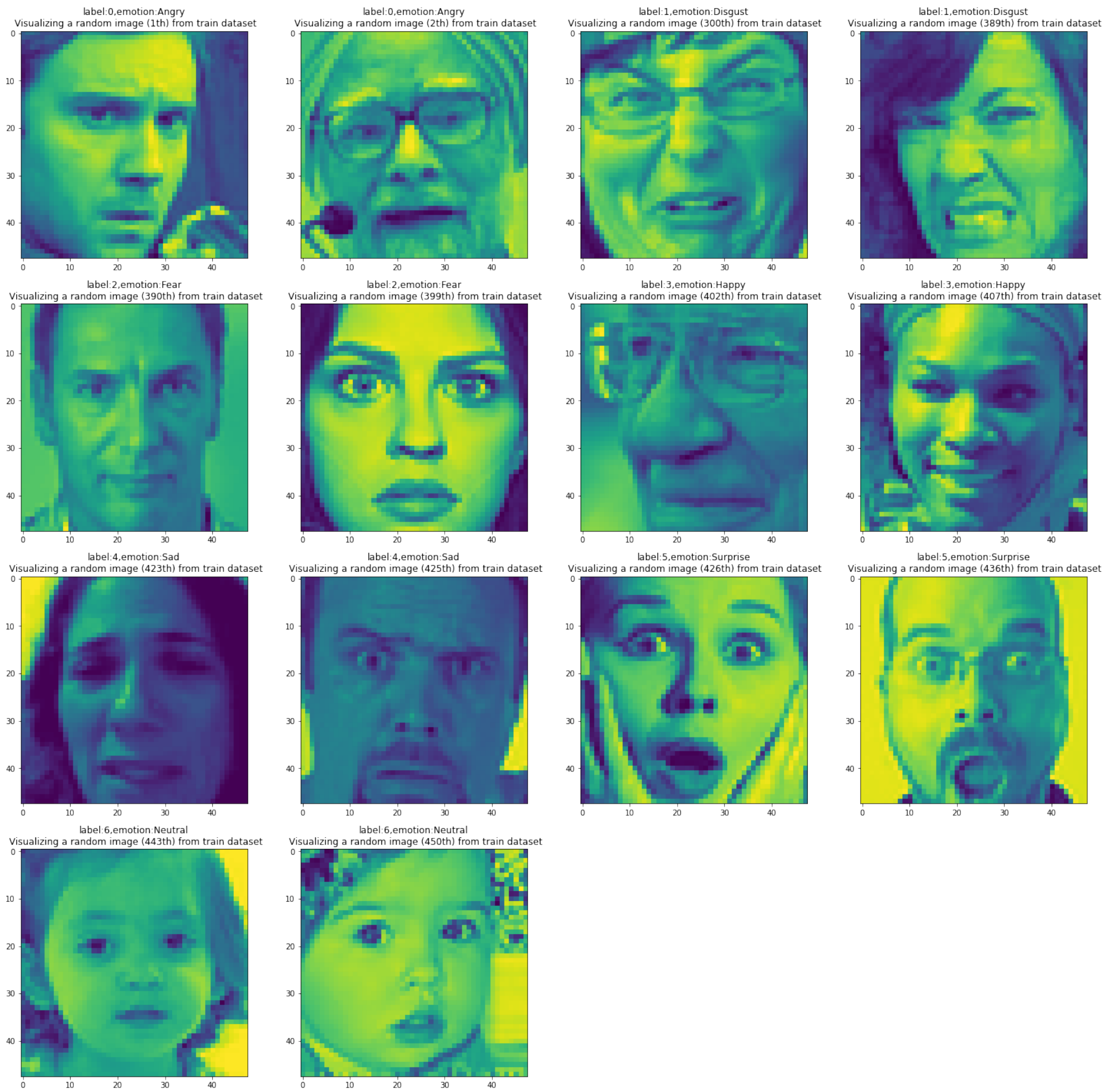


Figure 1: Visualization

(b) Data exploration:

Answer:

The code is as follow:

```
num_emotion = []
for i in range(7):
  num_tep = np.count_nonzero(train_labels==i)
  num_emotion.append(num_tep)
  print(i,labelMap[i],num_tep)
print(len(train_labels),sum(num_emotion))
```

The results are shown as follow:

label	emotion	number
0	Angry	3995
1	Disgust	436
2	Fear	4097
3	Happy	7215
4	Sad	4830
5	Surprise	3171
6	Neutral	4965

Table 2: Count the number of samples per emotion in the training data.

(c) Image classification with FNNS:

(c.i)

- Experiment on #layers,#nodes per layer,activation function,dropout,weight regularization.
- Report classification accuracy on the training and validation sets, running time and #parameters.
- Plot the cross-entropy loss over the number of iterations during training for 2-3 hyper-parameter combinations.

Answer:

Before the construction of FNN or CNN, data preprocessing is needed to improve traning performance. The code is as follow:

Listing 2: Data Preporcessing

```
# tf.image.per_image_standardization
def np_per_image_standardization(ImageData):
  N = 48*48
  N = 1/np.sqrt(N)
  ImageData = np.array(ImageData,dtype=float)
```

```
  for i in range(len(ImageData)):
    mean = np.mean(ImageData[i])
    stddev = np.std(ImageData[i])
    adjusted_stddev = max(stddev,N)
    #image_tep = (ImageData[i] - mean)/adjusted_stddev
    #print(image_tep)
    ImageData[i] = (ImageData[i] - mean)/adjusted_stddev
    #print(ImageData[i])
  return ImageData
#print(np_per_image_standardization(train_images[0:2])[1])
train_image = np_per_image_standardization(train_images)
valid_image = np_per_image_standardization(valid_images)
test_image = np_per_image_standardization(test_images)
```

Here, we need to set following hyper parameters:

- N = # layers $\in \{2, 3, 4, 5, 6\}$
- n = # nodes or initial layers $\in \{48 * 48, 48 * 48/2, 24 * 24\}$
- af = activation function $\in \{'relu', 'sigmoid', 'tanh'\}$
- dp = dropout ration $\in \{0.001, 0.01\}$
- wr = weight regularization $\in \{0.2, 0.3, 0.4\}$

To experiment on the above parameter, the following FNN structure is needed:

```
 ----------------------------------------------------------------
Layer (type)          Input Shape               Output Shape
=================================================================
1 layer (Dense)       (None, 48*48)             (None,n)
 ----------------------------------------------------------------
2 layer (Dense)       (None, n)                 (None,n/2)
 ----------------------------------------------------------------
...                   ...                       ...
 ----------------------------------------------------------------
N-2 layer (Dense)     (None, n/((2)^(N-4)))     (None, n/((2)^(N-3)))
 ----------------------------------------------------------------
dropout (Dropout)     (None, n/((2)^(N-3)))     (None, n/((2)^(N-3)))
 ----------------------------------------------------------------
N layer (Dense)       (None, n/((2)^(N-3)))     (None, 7)
=================================================================
```

Namely, always use only one dropout layer, and the dropout ratio is related to dp. The code for FNN is as follow. In the following code $epoch = 30, batch\_size = 256, lr = 0.01, momentum = 0.9$ and SGD is chosen as the optimal method.

- Input: $num\_layers, num\_nodes, act\_func, dropout\_ratio, regulaize\_ratio$ are related to the above hyper parameters.
- Input: $train\_status = True$ means using training data set for training, validation data set for fitting the model; $train\_status = False$ means using training and validation data set for training model, testing data set for evaluation the model.

- Output: When $train\_status = True$, output 'training accuracy','validation accuracy','running time','number of parameters'; when $train\_status = False$, output 'testing loss','testing accuracy'.

Listing 3: FNN

```
def funcFnn(num_layers,num_nodes,act_func, dropout_ratio, ...
    regulaize_ratio,train_status = True):
  #num_layers = 2,3,4,5,6
  #num_does = 48*48,48*48/2,24*24
  #act_func = 'relu' or 'sigmoid' or 'tanh'
  #dropout_ratio = 0.2,0.3,0.4
  #regulaize_ratio = 0.001,0.01
  #  Define a Feed-Forward Model
  model = Sequential()
  nodes_tep = num_nodes
  for i in range(num_layers-1):
    if i != num_layers-2:
      model.add(Dense(nodes_tep,
                activation=act_func,
                input_shape=(48*48,),
                name="{}_hidden_layer".format(i+1)))
      nodes_tep = nodes_tep//2
    else:
      if i == 0:#no drop out, layer number =2
        model.add(Dense(nodes_tep,
                  activation=act_func,
                  input_shape=(48*48,),
                  name="{}_hidden_layer".format(i+1)))
      else:
        model.add(Dropout(dropout_ratio))
  model.add(Dense(7,
                  activation='softmax',
                  name="{}_hidden_layer".format(num_layers)))

  #  Validate your Model Architecture
  print(model.summary())

  # Compile model
  opt = SGD(lr=0.01, momentum=0.9, decay=regulaize_ratio)
  model.compile(optimizer=opt,
                loss='categorical_crossentropy',
                metrics=['accuracy'],)

  if train_status == True:
    # Train model
    time1 = time.time()
    training = model.fit(flatten_train_images,
                         to_categorical(train_labels),
                         epochs=30,
                         batch_size=256,
                         validation_data=(flatten_valid_images,
                                          to_categorical(valid_labels)))
    time2 = time.time() -time1
    funcPlot_ce(training.history['accuracy'],training.history['val_accuracy'],
                training.history['loss'],training.history['val_loss'],
```

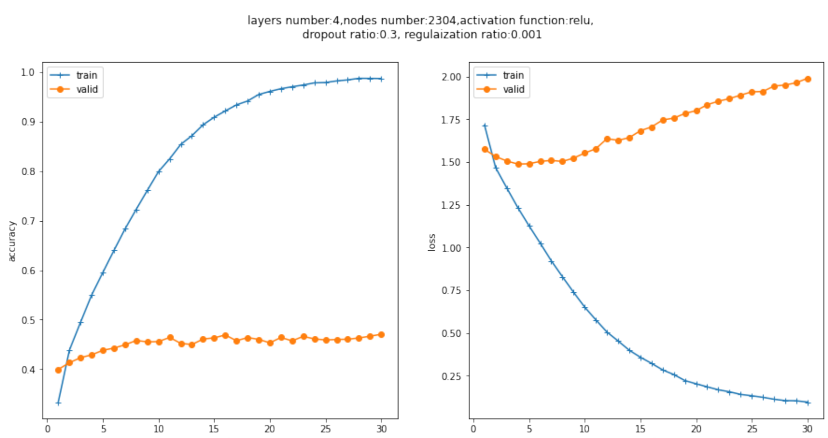
```
                   num_layers,num_nodes,act_func,
                   dropout_ratio,regulaize_ratio)
    return [training.history['accuracy'],
            training.history['val_accuracy']],
            time2,model.count_params()
  else:
    # Test model
    training = model.fit(np.vstack([flatten_train_images,
                                    flatten_valid_images]),
                         to_categorical(np.hstack([train_labels,
                                                   valid_labels])),
                         epochs=30,
                         batch_size=256,
                         )
    performance = model.evaluate(flatten_test_images,
                                 to_categorical(test_labels))
    return performance
```

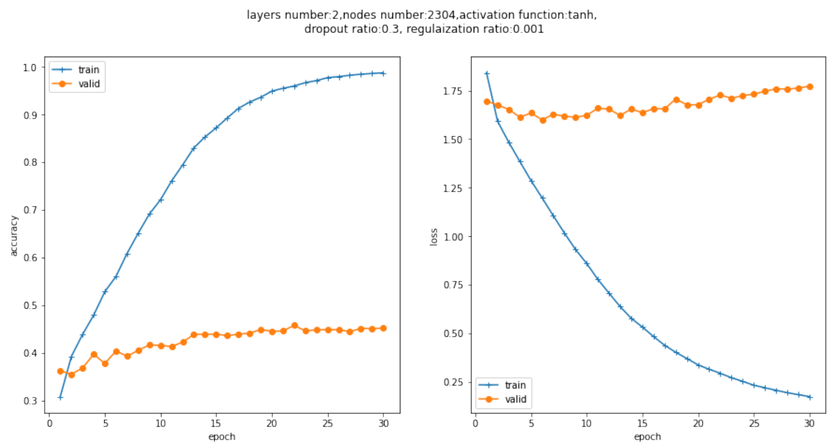
The full results are shown in Appendix-'fnn results'. In this section only the best parameter combination is shown(here, 'best' mean the highest valid accuracy and the lowest time):

#layers	#initial nodes	activate function	dropout ratio	weight ratio
4	2304	'relu'	0.3	0.001
	training accuracy	validation accuracy	running time	#parameters
	0.9864153861999512	0.47060462832450867	21.376885414123535	7974151

Table 3: FNN Best Results

The plots are shown as follow:





(c.ii)Run the best model in (c.i) on the testing set. Report the emotion classification accuracy on the testing set.

Answer: The code is as follow:

```
performance =funcFnn(4,2304,'relu',0.3, 0.001,train_status=False)
#2304 ['relu', 0.001, 0.3, 4]
```

The results are shown as follow:

train loss	train acc	test loss	test acc
0.1054	0.9846	1.8753	0.4831

Table 4: FNN Testing

(d) Image classification with CNNS:

(d.i)

- Experiment on #layers,filter size,stride size,activation function,dropout,weight regularization.
- Report classification accuracy on the training and validation sets, running time and #parameters.
- How do these metrics compare to the FNN.

Answer:

Here, we need to set following hyper parameters:

- N = # layers $\in \{7, 10\}$ for stride 1 and 7 for stride 2
- fs = filter size $\in \{3, 5\}$
- ss = stride size $\in \{1, 2\}$
- af = activation function $\in \{'relu',' sigmoid',' tanh'\}$
- dp = dropout ration $\in \{0.001, 0.01\}$
- wr = weight regularization $\in \{0.2, 0.3, 0.4\}$

To experiment on the above parameter, the following CNN structure for stride 1 (take $N = 10$ as an example)is needed:

```
----------------------------------------------------------------
Layer (type)        Input Shape             Output Shape
================================================================
1 layer (Conv2D)    (None, 48, 48, 1)       (None, 48, 48, 32)
----------------------------------------------------------------
2 layer (Conv2D)    (None, 48, 48, 32)      (None, 48, 48, 32)
----------------------------------------------------------------
max_pooling2d       (None, 48, 48, 32)      (None, 24, 24, 32)
----------------------------------------------------------------
4 layer (Conv2D)    (None, 24, 24, 32)      (None, 24, 24, 64)
----------------------------------------------------------------
5 layer (Conv2D)    (None, 24, 24, 64)      (None, 24, 24, 64)
----------------------------------------------------------------
max_pooling2d       (None, 24, 24, 64)      (None, 12, 12, 64)
----------------------------------------------------------------
flatten (Flatten)   (None, 12, 12, 64)      (None, 12*12*64)
----------------------------------------------------------------
8 layer (Dense)     (None, 12*12*64)        (None, 512)
----------------------------------------------------------------
dropout (Dropout)   (None, 512)             (None, 512)
----------------------------------------------------------------
10 layer(Dense)     (None, 512)             (None, 7)
================================================================
```

the following CNN structure for stride 2 is needed:

```
 ----------------------------------------------------------------
Layer (type)         Input Shape             Output Shape
=================================================================
1 layer (Conv2D)     (None, 48, 48, 1)       (None, 24, 24, 32)
 ----------------------------------------------------------------
2 layer (Conv2D)     (None, 24, 24, 32)      (None, 12, 12, 32)
 ----------------------------------------------------------------
max_pooling2d        (None, 12, 12, 32)      (None, 6, 6, 32)
 ----------------------------------------------------------------
flatten (Flatten)    (None, 6, 6, 32)        (None, 6*6*32)
 ----------------------------------------------------------------
5 layer (Dense)      (None, 6*6*32)          (None, 512)
 ----------------------------------------------------------------
dropout (Dropout)    (None, 512)             (None, 512)
 ----------------------------------------------------------------
7 layer(Dense)      (None, 512)             (None, 7)
=================================================================
```

Namely, always use only one dropout layer, and the dropout ratio is related to *dp*. The code for FNN is as follow. In the following code $epoch = 30, batch\_size = 256, lr = 0.01, momentum = 0.9$ and SGD is chosen as the optimal method.

- Input: $num\_layers, num\_nodes, act\_func, dropout\_ratio, regulaize\_ratio$ are related to the above hyper parameters.
- Input: $train\_status = True$ means using training data set for training, validation data set for fitting the model; $train\_status = False$ means using training and validation data set for training model, testing data set for evaluation the model.
- Output: When $train\_status = True$, output 'training accuracy','validation accuracy','running time','number of parameters'; when $train\_status = False$, output 'testing loss','testing accuracy'.

Listing 4: CNN

```
def funcCnn(num_layers,filter_size,stride_size,act_func, ...
    dropout_ratio, regulaize_ratio,train_status = True):
  #num_layers =  4+ 3,+6, or 4+ 3
  #stride_size = 1        or 2
  #filter_size = 3 or 5
  #act_func = 'relu' or 'sigmoid' or 'tanh'
  #dropout_ratio = 0.2,0.3,0.4
  #regulaize_ratio = 0.001,0.01
  #  Define a Feed-Forward Model
  model = Sequential()
  nodes_tep = 32
  for i in range((num_layers-4)//3):
    if i==0:
      model.add(Conv2D(nodes_tep,
                       kernel_size = filter_size,
                       activation=act_func,
                       input_shape=(48,48,1),
```

```
                        strides=(stride_size,stride_size),
                        padding = 'same',
                        name="{}_hidden_layer".format(i*3+1)))
      model.add(Conv2D(nodes_tep,
                        kernel_size = filter_size,
                        activation=act_func,
                        strides=(stride_size,stride_size),
                        padding='same',
                        name="{}_hidden_layer".format(i*3+2)))
      model.add(MaxPooling2D(pool_size=(2,2)))
      nodes_tep = nodes_tep*2
    else:
      model.add(Conv2D(nodes_tep,
                        kernel_size = filter_size,
                        activation=act_func,
                        strides=(stride_size,stride_size),
                        padding = 'same',
                        name="{}_hidden_layer".format(i*3+1)))
      model.add(Conv2D(nodes_tep,
                        kernel_size = filter_size,
                        activation=act_func,
                        strides=(stride_size,stride_size),
                        padding='same',
                        name="{}_hidden_layer".format(i*3+2)))
      model.add(MaxPooling2D(pool_size=(2,2)))
      nodes_tep = nodes_tep*2
  model.add(Flatten())
  model.add(Dense(512,
            activation=act_func,
            name="{}_hidden_layer".format(num_layers-2)))
  model.add(Dropout(dropout_ratio))
  model.add(Dense(7,
            activation='softmax',
            name="{}_hidden_layer".format(num_layers)))
  #  Validate your Model Architecture
  print(model.summary())

  # Compile model
  opt = SGD(lr=0.01, momentum=0.9, decay=regulaize_ratio)
  model.compile(optimizer=opt,
                loss='categorical_crossentropy',
                metrics=['accuracy'],)

  if train_status == True:
    # Train model
    time1 = time.time()
    training = model.fit(train_images_3d,
                         to_categorical(train_labels),
                         epochs=30,
                         batch_size=256,
                         validation_data=(valid_images_3d,
                                          to_categorical(valid_labels)))
    time2 = time.time() -time1
    funcPlot_ce_3d(training.history['accuracy'],training.history['val_accuracy'],
                training.history['loss'],training.history['val_loss'],
                num_layers,filter_size,stride_size,act_func,
                dropout_ratio,regulaize_ratio)
```

```
    return [training.history['accuracy'],
            training.history['val_accuracy']],
            time2,model.count_params()
  else:
    # Test model
      training = model.fit(np.vstack([train_images_3d,
                                      valid_images_3d]),
                           to_categorical(np.hstack([train_labels,
                                                     valid_labels])),
                           epochs=30,
                           batch_size=256,
                           )
      performance = model.evaluate(test_images_3d ,
                        to_categorical(test_labels))
      return performance
```

The full results are shown in Appendix-'cnn results'. In this section only the best parameter combination is shown(here, 'best' mean the highest valid accuracy and the lowest time):

#layers	stride	filter	activate function	dropout ratio	weight ratio
10	1	5	'relu'	0.3	0.001
		training accuracy	validation accuracy	running time	#parameters
		0.9803197383880615	0.5717470049858093	160.68341517448425	4902887

Table 5: CNN Best Results

From the data set in appendix, the following comparison results with FNN can be obtained:

- Train accuracy of cnn is a little bit larger than that of fnn in total.
- Valid accuracy of cnn is larger than that of fnn. In fnn, the top 10 largest numbers is between [0.45,0.49], while that in cnn is [0.55,0.58]
- Time of cnn is more longer than that of fnn. Running time of fnn is within 50, while cnn's occurs over 100.
- The number of parameter in cnn is usually 2 or 3 times larger than that of fnn.

(d.ii)Run the best model in (d.i) on the testing set. Report the emotion classification accuracy on the testing set. How does this compare to FNN?

Answer: The code is as follow:

```
performance =funcCnn(10,5,1,'relu',0.3, 0.001,train_status=False)
```

The results are shown as follow:

train loss	train acc	test loss	test acc
0.0615	0.9805	2.1902	0.6013

Table 6: CNN Testing

By comparing with FNN, the following results can be obtained:

- The testing accuracy of CNN(over 0.55) is larger than that of FNN(less than 0.5).

(e) Bayesian optimization for hyper-parameter tuning:

- Use publicly available libraries to perform a Bayesian optimization on the hyper-parameter space using the validation set.
- Report the emotion classification accuracy on the testing set

Answer:
The code is as follow:
First the model of CNN is re-defined.

Listing 5: CNN model

```
def  black_box_function(num_layers,filter_size,stride_size, ...
    dropout_ratio, regulaize_ratio,act_func='relu',train_status ...
    = True):
  num_layers = int(num_layers)
  filter_size = int(filter_size)
  stride_size = int(stride_size)
  model = Sequential()
  nodes_tep = 32
  for i in range((num_layers-4)//3):
    if i==0:
      model.add(Conv2D(nodes_tep,
                        kernel_size = filter_size,
                        activation=act_func,
                        input_shape=(48,48,1),
                        strides=(stride_size,stride_size),
                        padding = 'same',
                        name="{}_hidden_layer".format(i*3+1)))
      model.add(Conv2D(nodes_tep,
                        kernel_size = filter_size,
                        activation=act_func,
                        strides=(stride_size,stride_size),
                        padding='same',
                        name="{}_hidden_layer".format(i*3+2)))
      model.add(MaxPooling2D(pool_size=(2,2)))
      nodes_tep = nodes_tep*2
    else:
      model.add(Conv2D(nodes_tep,
                        kernel_size = filter_size,
                        activation=act_func,
                        strides=(stride_size,stride_size),
```

```
                           padding = 'same',
                           name="{}_hidden_layer".format(i*3+1)))
        model.add(Conv2D(nodes_tep,
                           kernel_size = filter_size,
                           activation=act_func,
                           strides=(stride_size,stride_size),
                           padding='same',
                           name="{}_hidden_layer".format(i*3+2)))
        model.add(MaxPooling2D(pool_size=(2,2)))
        nodes_tep = nodes_tep*2
    model.add(Flatten())
    model.add(Dense(512,
              activation=act_func,
              name="{}_hidden_layer".format(num_layers-2)))
    model.add(Dropout(dropout_ratio))
    model.add(Dense(7,
              activation='softmax',
              name="{}_hidden_layer".format(num_layers)))
    #  Validate your Model Architecture
    print(model.summary())

    # Compile model
    opt = SGD(lr=0.01, momentum=0.9, decay=regulaize_ratio)
    model.compile(optimizer=opt,
                  loss='categorical_crossentropy',
                  metrics=['accuracy'],)

    if train_status == True:
      # Train model
      time1 = time.time()
      training = model.fit(train_images_3d,
                             to_categorical(train_labels),
                             epochs=30,
                             batch_size=256,
                             validation_data=(valid_images_3d,
                                              to_categorical(valid_labels)))
      time2 = time.time() -time1
      funcPlot_ce_3d(training.history['accuracy'],training.history['val_accuracy'],
                  training.history['loss'],training.history['val_loss'],
                  num_layers,filter_size,stride_size,act_func,
                  dropout_ratio,regulaize_ratio)

      return training.history['val_accuracy'][-1]
    else:
      # Test model
        training = model.fit(np.vstack([train_images_3d,
                                         valid_images_3d]),
                             to_categorical(np.hstack([train_labels,
                                                       valid_labels])),
                             epochs=30,
                             batch_size=256,
                             )
        performance = model.evaluate(test_images_3d ,
                                  to_categorical(test_labels))
        return performance
```

Next, the Bayesian Optimization is used.

Listing 6: Bayesian Optimization

```
from bayes_opt import BayesianOptimization
###################Hyperparameter Setting######################
pbounds = {'num_layers':(7,10),
            'filter_size':(3,5),
            'stride_size':(1,2),
            'dropout_ratio':(0.2,0.4),
            'regulaize_ratio':(0.001,0.01)}#'act_func':'relu'

optimizer = BayesianOptimization(
    f=black_box_function,
    pbounds=pbounds,
    verbose=2, # verbose = 1 prints only when a maximum is observed
    random_state=1,
)

optimizer.maximize(
    init_points=5,
    n_iter=5,
)
black_box_function(optimizer.max['params']['num_layers'],
                   optimizer.max['params']['filter_size'],
                   optimizer.max['params']['stride_size'],
                   optimizer.max['params']['dropout_ratio'],
                   optimizer.max['params']['regulaize_ratio'],
                   act_func='relu',train_status = False)
```

The results of best parameter are as follow:

#layers	stride	filter	activate function	dropout ratio	weight ratio
10	1	5	'relu'	0.4	0.001

Table 7: New Method CNN Best parameter Results

The results of testing is shown as follow:

train loss	train acc	test loss	test acc
0.0955	0.9684	2.2181	0.5904

Table 8: New method CNN Testing

Appendix

Listing 7: fnn results

```
#nodes, [act, regular, drop, #layer], Training acc, Validation ...
    acc, Length of time, #parameters
2304 ['relu', 0.001, 0.2, 2] 0.9947403073310852 ...
    0.45388686656951904 18.317853212356567 5326855
2304 ['relu', 0.001, 0.2, 3] 0.9230206608772278 ...
    0.4494287967681885 17.613085746765137 5326855
2304 ['relu', 0.001, 0.2, 4] 0.9939043521881104 ...
    0.47088325023651123 21.377570629119873 7974151
2304 ['relu', 0.001, 0.2, 5] 0.9969347715377808 ...
    0.4555586576461792 22.753923654556274 8634247
2304 ['relu', 0.001, 0.2, 6] 0.9973527193069458 ...
    0.4527723491191864 23.634246110916138 8798407
2304 ['relu', 0.001, 0.3, 2] 0.994078516960144 ...
    0.4421844482421875 17.041662454605103 5326855
2304 ['relu', 0.001, 0.3, 3] 0.8790971636772156 ...
    0.45026469230651855 17.583329916000366 5326855
2304 ['relu', 0.001, 0.3, 4] 0.9864153861999512 ...
    0.47060462832450867 21.376885414123535 7974151
2304 ['relu', 0.001, 0.3, 5] 0.9964122772216797 ...
    0.45444414019584656 22.792892932891846 8634247
2304 ['relu', 0.001, 0.3, 6] 0.9969347715377808 ...
    0.45583727955818176 23.252522945404053 8798407
2304 ['relu', 0.001, 0.4, 2] 0.9949841499328613 ...
    0.4508219659328461 17.14392066001892 5326855
2304 ['relu', 0.001, 0.4, 3] 0.8271273970603943 ...
    0.4563945531845093 17.588998794555664 5326855
2304 ['relu', 0.001, 0.4, 4] 0.9753038883209229 ...
    0.4669824540615082 21.348151445388794 7974151
2304 ['relu', 0.001, 0.4, 5] 0.9954021573066711 ...
    0.4527723491191864 23.354179620742798 8634247
2304 ['relu', 0.001, 0.4, 6] 0.9968999028205872 ...
    0.44747841358184814 23.386398315429688 8798407
2304 ['relu', 0.01, 0.2, 2] 0.7382354140281677 ...
    0.4399554133415222 17.782411336898804 5326855
2304 ['relu', 0.01, 0.2, 3] 0.6340172290802002 ...
    0.4366118609905243 17.4662344455719 5326855
2304 ['relu', 0.01, 0.2, 4] 0.7470827698707581 ...
    0.43911951780319214 21.563490629196167 7974151
2304 ['relu', 0.01, 0.2, 5] 0.8092584013938904 ...
    0.4430203437805176 22.881914377212524 8634247
2304 ['relu', 0.01, 0.2, 6] 0.875404953956604 ...
    0.43466147780418396 23.304298162460327 8798407
2304 ['relu', 0.01, 0.3, 2] 0.7416141033172607 ...
    0.4282529950141907 17.152796030044556 5326855
2304 ['relu', 0.01, 0.3, 3] 0.5961893200874329 ...
    0.4329896867275238 17.701224088668823 5326855
2304 ['relu', 0.01, 0.3, 4] 0.7067469954490662 ...
    0.4399554133415222 21.87585973739624 7974151
2304 ['relu', 0.01, 0.3, 5] 0.7554076910018921 ...
    0.4485929310321808 22.736225843429565 8634247
2304 ['relu', 0.01, 0.3, 6] 0.8361141085624695 ...
    0.43466147780418396 23.347081184387207 8798407
```

```
2304 ['relu', 0.01, 0.4, 2] 0.7374690771102905 ...
     0.43828365206718445 17.44435691833496 5326855
2304 ['relu', 0.01, 0.4, 3] 0.5640391707420349 ...
     0.4329896867275238 17.864420175552368 5326855
2304 ['relu', 0.01, 0.4, 4] 0.6685360074043274 ...
     0.45249372720718384 21.58438229560852 7974151
2304 ['relu', 0.01, 0.4, 5] 0.7107875347137451 ...
     0.4407913088798523 22.889854192733765 8634247
2304 ['relu', 0.01, 0.4, 6] 0.7850499749183655 ...
     0.42407354712486267 23.874338388442993 8798407
1152 ['relu', 0.001, 0.2, 2] 0.9886795282363892 ...
     0.44469210505485535 16.810179471969604 2663431
1152 ['relu', 0.001, 0.2, 3] 0.8625866174697876 ...
     0.45583727955818176 16.425300359725952 2663431
1152 ['relu', 0.001, 0.2, 4] 0.979309618473053 ...
     0.45750904083251953 17.40920901298523 3323527
1152 ['relu', 0.001, 0.2, 5] 0.9956808090209961 ...
     0.44469210505485535 17.32829475402832 3487687
1152 ['relu', 0.001, 0.2, 6] 0.996516764163971 ...
     0.44246307015419006 17.732422351837158 3528295
1152 ['relu', 0.001, 0.3, 2] 0.9884704947471619 ...
     0.4441348612308502 16.402238845825195 2663431
1152 ['relu', 0.001, 0.3, 3] 0.8054965138435364 ...
     0.44106993079185486 16.83409023284912 2663431
1152 ['relu', 0.001, 0.3, 4] 0.9586541056632996 ...
     0.4536082446575165 16.89425802230835 3323527
1152 ['relu', 0.001, 0.3, 5] 0.9899682998657227 ...
     0.45110058784484863 17.05216908454895 3487687
1152 ['relu', 0.001, 0.3, 6] 0.99543696641922 0.4349400997161865 ...
     17.575154781341553 3528295
1152 ['relu', 0.001, 0.4, 2] 0.9884008765220642 ...
     0.4335469603538513 16.140185832977295 2663431
1152 ['relu', 0.001, 0.4, 3] 0.7414748072624207 ...
     0.4522151052951813 16.991451740264893 2663431
1152 ['relu', 0.001, 0.4, 4] 0.9219059944152832 ...
     0.4689328372478485 17.02564263343811 3323527
1152 ['relu', 0.001, 0.4, 5] 0.9832108616828918 ...
     0.44469210505485535 16.931578159332275 3487687
1152 ['relu', 0.001, 0.4, 6] 0.9939740300178528 ...
     0.44246307015419006 17.36157727241516 3528295
1152 ['relu', 0.01, 0.2, 2] 0.6815284490585327 ...
     0.42630258202552795 16.256353616714478 2663431
1152 ['relu', 0.01, 0.2, 3] 0.5835103988647461 ...
     0.4288102388381958 16.450093030929565 2663431
1152 ['relu', 0.01, 0.2, 4] 0.6575986742973328 ...
     0.4399554133415222 17.249725341796875 3323527
1152 ['relu', 0.01, 0.2, 5] 0.6799609661102295 ...
     0.43410420417785645 17.620203733444214 3487687
1152 ['relu', 0.01, 0.2, 6] 0.7416489720344543 ...
     0.4338255822658539 17.12582039833069 3528295
1152 ['relu', 0.01, 0.3, 2] 0.6881814002990723 ...
     0.4296461343765259 16.310710668563843 2663431
1152 ['relu', 0.01, 0.3, 3] 0.547249972820282 ...
     0.42630258202552795 16.49140238761902 2663431
1152 ['relu', 0.01, 0.3, 4] 0.6230102181434631 ...
     0.43187516927719116 16.98663592338562 3323527
```

```
1152 ['relu', 0.01, 0.3, 5] 0.6443623900413513 ...
     0.4296461343765259 17.09008550643921 3487687
1152 ['relu', 0.01, 0.3, 6] 0.6795778274536133 ...
     0.42992475628852844 17.347952604293823 3528295
1152 ['relu', 0.01, 0.4, 2] 0.6848026514053345 ...
     0.4335469603538513 16.05794405937195 2663431
1152 ['relu', 0.01, 0.4, 3] 0.5196279883384705 ...
     0.42685985565185547 16.198853731155396 2663431
1152 ['relu', 0.01, 0.4, 4] 0.5895712375640869 ...
     0.4349400997161865 17.09304904937744 3323527
1152 ['relu', 0.01, 0.4, 5] 0.6072311997413635 ...
     0.4310392737388611 16.84554362297058 3487687
1152 ['relu', 0.01, 0.4, 6] 0.6365599632263184 ...
     0.4201727509498596 17.475181341171265 3528295
576 ['relu', 0.001, 0.2, 2] 0.9663519859313965 0.438562273979187 ...
     15.844948053359985 1331719
576 ['relu', 0.001, 0.2, 3] 0.790483832359314 0.4497074484825134 ...
     15.790835857391357 1331719
576 ['relu', 0.001, 0.2, 4] 0.9238218069076538 ...
     0.44106993079185486 16.557732582092285 1495879
576 ['relu', 0.001, 0.2, 5] 0.9704622030258179 ...
     0.4388408958911896 16.735814332962036 1536487
576 ['relu', 0.001, 0.2, 6] 0.987913191318512 ...
     0.42630258202552795 16.945908308029175 1546423
576 ['relu', 0.001, 0.3, 2] 0.9653070569038391 ...
     0.4393981695175171 15.461366176605225 1331719
576 ['relu', 0.001, 0.3, 3] 0.7253822684288025 ...
     0.45305100083351135 16.405776977539062 1331719
576 ['relu', 0.001, 0.3, 4] 0.8726879954338074 ...
     0.45750904083251953 16.53704261779785 1495879
576 ['relu', 0.001, 0.3, 5] 0.9446863532066345 ...
     0.43605461716651917 16.80001449584961 1536487
576 ['relu', 0.001, 0.3, 6] 0.9659688472747803 ...
     0.43911951780319214 16.82990837097168 1546423
576 ['relu', 0.001, 0.4, 2] 0.9657250046730042 ...
     0.43689051270484924 16.100160598754883 1331719
576 ['relu', 0.001, 0.4, 3] 0.6650179624557495 ...
     0.4449707567691803 15.814398288726807 1331719
576 ['relu', 0.001, 0.4, 4] 0.8256644010543823 ...
     0.44664251804351807 16.469887495040894 1495879
576 ['relu', 0.001, 0.4, 5] 0.913197934627533 ...
     0.43466147780418396 16.345154285430908 1536487
576 ['relu', 0.001, 0.4, 6] 0.9607788324356079 ...
     0.4310392737388611 16.32367467880249 1546423
576 ['relu', 0.01, 0.2, 2] 0.6339126825332642 ...
     0.42546671628952026 15.024202346801758 1331719
576 ['relu', 0.01, 0.2, 3] 0.5433835983276367 ...
     0.42713847756385803 15.257815837860107 1331719
576 ['relu', 0.01, 0.2, 4] 0.5838587284088135 0.419058233499527 ...
     16.048134565353394 1495879
576 ['relu', 0.01, 0.2, 5] 0.5912083387374878 0.4257453382015228 ...
     16.306710481643677 1536487
576 ['relu', 0.01, 0.2, 6] 0.5684280395507812 ...
     0.42685985565185547 16.652992248535156 1546423
576 ['relu', 0.01, 0.3, 2] 0.6353408098220825 0.4176650941371918 ...
     15.30307126045227 1331719
```

```
576 ['relu', 0.01, 0.3, 3] 0.5177122354507446 0.419058233499527 ...
    15.714508295059204 1331719
576 ['relu', 0.01, 0.3, 4] 0.5578389763832092 ...
    0.41627195477485657 16.304210424423218 1495879
576 ['relu', 0.01, 0.3, 5] 0.5551220774650574 0.423237681388855 ...
    16.62777304649353 1536487
576 ['relu', 0.01, 0.3, 6] 0.5828137397766113 0.416829198598861 7 ...
    17.256230115890503 1546423
576 ['relu', 0.01, 0.4, 2] 0.637813925743103 0.4176650941371918 ...
    15.545871019363403 1331719
576 ['relu', 0.01, 0.4, 3] 0.49047335982322693 0.419058233499527 ...
    15.693447589874268 1331719
576 ['relu', 0.01, 0.4, 4] 0.5283012390136719 0.4187796115875244 ...
    16.06034755706787 1495879
576 ['relu', 0.01, 0.4, 5] 0.5397958755493164 ...
    0.42128726840019226 17.639638662338257 1536487
576 ['relu', 0.01, 0.4, 6] 0.528963029384613 0.41432154178619385 ...
    17.21776032447815 1546423
2304 ['sigmoid', 0.001, 0.2, 2] 0.48392489552497864 ...
    0.3884090185165405 17.774089336395264 5326855
2304 ['sigmoid', 0.001, 0.2, 3] 0.41342437267303467 ...
    0.37057676911354065 18.137301206588745 5326855
2304 ['sigmoid', 0.001, 0.2, 4] 0.3793583810329437 ...
    0.376706600189209 22.48316740989685 7974151
2304 ['sigmoid', 0.001, 0.2, 5] 0.3513183891773224 ...
    0.372527152299881 23.78548002243042 8634247
2304 ['sigmoid', 0.001, 0.2, 6] 0.2525688707828522 ...
    0.25494566559791565 24.040858268737793 8798407
2304 ['sigmoid', 0.001, 0.3, 2] 0.4791876971721649 ...
    0.3780997395515442 17.6877179145813 5326855
2304 ['sigmoid', 0.001, 0.3, 3] 0.3995959460735321 ...
    0.37698522210121155 18.054802417755127 5326855
2304 ['sigmoid', 0.001, 0.3, 4] 0.37378522753715515 ...
    0.37698522210121155 22.045512914657593 7974151
2304 ['sigmoid', 0.001, 0.3, 5] 0.34658122062683105 ...
    0.3611033856868744 23.638893842697144 8634247
2304 ['sigmoid', 0.001, 0.3, 6] 0.2515239119529724 ...
    0.24937307834625244 23.991887092590332 8798407
2304 ['sigmoid', 0.001, 0.4, 2] 0.4783865809440613 ...
    0.37837839126586914 17.567976713180542 5326855
2304 ['sigmoid', 0.001, 0.4, 3] 0.3875439763069153 ...
    0.3794929087162018 17.974888801574707 5326855
2304 ['sigmoid', 0.001, 0.4, 4] 0.3708244860172272 ...
    0.37614935636520386 22.29586935043335 7974151
2304 ['sigmoid', 0.001, 0.4, 5] 0.33797764778137207 ...
    0.3633323907852173 23.900073766708374 8634247
2304 ['sigmoid', 0.001, 0.4, 6] 0.25124526023864746 ...
    0.24937307834625244 23.929548978805542 8798407
2304 ['sigmoid', 0.01, 0.2, 2] 0.4014768898487091 ...
    0.3736416697502136 17.650300979614258 5326855
2304 ['sigmoid', 0.01, 0.2, 3] 0.3714166283607483 ...
    0.36974087357521057 18.188048362731934 5326855
2304 ['sigmoid', 0.01, 0.2, 4] 0.3502385914325714 ...
    0.3658400774002075 22.16035270690918 7974151
2304 ['sigmoid', 0.01, 0.2, 5] 0.2653871476650238 ...
    0.27835050225257874 23.691832542419434 8634247
```

```
2304 ['sigmoid', 0.01, 0.2, 6] 0.24354732036590576 ...
    0.24937307834625244 23.983513832092285 8798407
2304 ['sigmoid', 0.01, 0.3, 2] 0.4027308523654938 ...
    0.3736416697502136 17.501641035079956 5326855
2304 ['sigmoid', 0.01, 0.3, 3] 0.3650074899196625 ...
    0.3753134608268738 17.995738983154297 5326855
2304 ['sigmoid', 0.01, 0.3, 4] 0.3406597375869751 ...
    0.3647255599498749 21.976600170135498 7974151
2304 ['sigmoid', 0.01, 0.3, 5] 0.260858952999115 ...
    0.25996097922325134 23.478772163391113 8634247
2304 ['sigmoid', 0.01, 0.3, 6] 0.2443832904100418 ...
    0.24937307834625244 23.93365168571472 8798407
2304 ['sigmoid', 0.01, 0.4, 2] 0.40102407336235046 ...
    0.3716912865638733 17.534956693649292 5326855
2304 ['sigmoid', 0.01, 0.4, 3] 0.3599568009376526 ...
    0.37419894337654114 17.839532136917114 5326855
2304 ['sigmoid', 0.01, 0.4, 4] 0.332613468170166 ...
    0.36528280377388 22.492613077163696 7974151
2304 ['sigmoid', 0.01, 0.4, 5] 0.2539621591567993 ...
    0.25076621770858765 23.53408169746399 8634247
2304 ['sigmoid', 0.01, 0.4, 6] 0.2456720918416977 ...
    0.24937307834625244 23.892779111862183 8798407
1152 ['sigmoid', 0.001, 0.2, 2] 0.4927026331424713 ...
    0.3970465362071991 16.460169315338135 2663431
1152 ['sigmoid', 0.001, 0.2, 3] 0.41561880707740784 ...
    0.3911953270435333 16.87459444999695 2663431
1152 ['sigmoid', 0.001, 0.2, 4] 0.3760841488838196 ...
    0.37614935636520386 17.223569869995117 3323527
1152 ['sigmoid', 0.001, 0.2, 5] 0.3502385914325714 ...
    0.36639732122421265 17.29114842414856 3487687
1152 ['sigmoid', 0.001, 0.2, 6] 0.25100141763687134 ...
    0.24937307834625244 17.676671266555786 3528295
1152 ['sigmoid', 0.001, 0.3, 2] 0.49392175674438477 ...
    0.39398160576820374 16.27293038368225 2663431
1152 ['sigmoid', 0.001, 0.3, 3] 0.4019993841648102 ...
    0.3806074261665344 16.618448495864868 2663431
1152 ['sigmoid', 0.001, 0.3, 4] 0.3737155497074127 ...
    0.3750348389148712 17.312512159347534 3323527
1152 ['sigmoid', 0.001, 0.3, 5] 0.34128671884536743 ...
    0.3647255599498749 17.28181743621826 3487687
1152 ['sigmoid', 0.001, 0.3, 6] 0.25145423412323 ...
    0.24937307834625244 17.771810054779053 3528295
1152 ['sigmoid', 0.001, 0.4, 2] 0.4941655993461609 ...
    0.392031192779541 16.489774465560913 2663431
1152 ['sigmoid', 0.001, 0.4, 3] 0.39088788628578186 ...
    0.3822791874408722 16.627821922302246 2663431
1152 ['sigmoid', 0.001, 0.4, 4] 0.36626145243644714 ...
    0.376706600189209 17.45064377784729 3323527
1152 ['sigmoid', 0.001, 0.4, 5] 0.3348427414894104 ...
    0.3591529726982116 7 17.159162759780884 3487687
1152 ['sigmoid', 0.001, 0.4, 6] 0.2513149082660675 ...
    0.24937307834625244 18.532127380371094 3528295
1152 ['sigmoid', 0.01, 0.2, 2] 0.40137240290641785 ...
    0.37698522210121155 16.475163459777832 2663431
1152 ['sigmoid', 0.01, 0.2, 3] 0.37587517499923706 ...
    0.3794929087162018 16.66596269607544 2663431
```

```
1152 ['sigmoid', 0.01, 0.2, 4] 0.3446305990219116 ...
     0.36416828632354736 17.16248345375061 3323527
1152 ['sigmoid', 0.01, 0.2, 5] 0.25626111030578613 ...
     0.25717470049858093 17.584635972976685 3487687
1152 ['sigmoid', 0.01, 0.2, 6] 0.24612490832805634 ...
     0.24937307834625244 17.70481777191162 3528295
1152 ['sigmoid', 0.01, 0.3, 2] 0.40043193101882935 ...
     0.3778211176395416 16.563292503356934 2663431
1152 ['sigmoid', 0.01, 0.3, 3] 0.36629629135131836 ...
     0.3716912865638733 17.514570236206055 2663431
1152 ['sigmoid', 0.01, 0.3, 4] 0.3372461497783661 ...
     0.36416828632354736 17.878440856933594 3323527
1152 ['sigmoid', 0.01, 0.3, 5] 0.25295203924179077 ...
     0.24993033707141876 18.027076482772827 3487687
1152 ['sigmoid', 0.01, 0.3, 6] 0.24372148513793945 ...
     0.24937307834625244 18.029422760009766 3528295
1152 ['sigmoid', 0.01, 0.4, 2] 0.3988644778728485 ...
     0.3758707046508789 17.07562494277954 2663431
1152 ['sigmoid', 0.01, 0.4, 3] 0.35487130284309387 ...
     0.37559208273887634 16.954060077667236 2663431
1152 ['sigmoid', 0.01, 0.4, 4] 0.3271099627017975 ...
     0.35915297269821167 17.497801065444946 3323527
1152 ['sigmoid', 0.01, 0.4, 5] 0.25291720032691956 ...
     0.24937307834625244 17.6130154132843 3487687
1152 ['sigmoid', 0.01, 0.4, 6] 0.24727436900138855 ...
     0.24937307834625244 17.6580913066864 3528295
576 ['sigmoid', 0.001, 0.2, 2] 0.49336445331573486 ...
     0.3948175013065338 15.463596820831299 1331719
576 ['sigmoid', 0.001, 0.2, 3] 0.41729074716567993 ...
     0.39258846640586853 16.34825372695923 1331719
576 ['sigmoid', 0.001, 0.2, 4] 0.3759099841117859 ...
     0.37977153062820435 17.449793577194214 1495879
576 ['sigmoid', 0.001, 0.2, 5] 0.3423316776752472 ...
     0.36639732122421265 17.095805883407593 1536487
576 ['sigmoid', 0.001, 0.2, 6] 0.25124526023864746 ...
     0.24937307834625244 17.700989484786987 1546423
576 ['sigmoid', 0.001, 0.3, 2] 0.49430492520332336 ...
     0.39035943150520325 15.6623854637146 1331719
576 ['sigmoid', 0.001, 0.3, 3] 0.3962520360946655 ...
     0.38757315278053284 15.59645414352417 1331719
576 ['sigmoid', 0.001, 0.3, 4] 0.3711031377315521 ...
     0.37559208273887634 16.55583930015564 1495879
576 ['sigmoid', 0.001, 0.3, 5] 0.33738547563552856 ...
     0.3583170771598816 16.749063968658447 1536487
576 ['sigmoid', 0.001, 0.3, 6] 0.25124526023864746 ...
     0.24937307834625244 16.931774139404297 1546423
576 ['sigmoid', 0.001, 0.4, 2] 0.49277231097221375 ...
     0.40234047174453735 15.35754680633545 1331719
576 ['sigmoid', 0.001, 0.4, 3] 0.38876309990882874 ...
     0.3850654661655426 15.897498846054077 1331719
576 ['sigmoid', 0.001, 0.4, 4] 0.3649030029773712 ...
     0.37057676911354065 16.960141897201538 1495879
576 ['sigmoid', 0.001, 0.4, 5] 0.32630881667137146 ...
     0.349958211183548 17.45371389389038 1536487
576 ['sigmoid', 0.001, 0.4, 6] 0.2513497471809387 ...
     0.24937307834625244 17.580547332763672 1546423
```

```
576 ['sigmoid', 0.01, 0.2, 2] 0.40154656767845154 ...
    0.3764279782772064 15.58283543586731 1331719
576 ['sigmoid', 0.01, 0.2, 3] 0.3744122087955475 ...
    0.3803287744522095 15.697761297225952 1331719
576 ['sigmoid', 0.01, 0.2, 4] 0.33964958786964417 ...
    0.3619392514228821 16.56407594680786 1495879
576 ['sigmoid', 0.01, 0.2, 5] 0.25305652618408203 ...
    0.2532739043235779 17.135308504104614 1536487
576 ['sigmoid', 0.01, 0.2, 6] 0.24595074355602264 ...
    0.24937307834625244 17.371949434280396 1546423
576 ['sigmoid', 0.01, 0.3, 2] 0.40116339921951294 ...
    0.3845082223415375 16.30842900276184 1331719
576 ['sigmoid', 0.01, 0.3, 3] 0.365947961807251 ...
    0.37336304783821106 15.681404113769531 1331719
576 ['sigmoid', 0.01, 0.3, 4] 0.3345988988876343 ...
    0.3661186993122101 16.617504119873047 1495879
576 ['sigmoid', 0.01, 0.3, 5] 0.24821484088897705 ...
    0.24937307834625244 17.203335762023926 1536487
576 ['sigmoid', 0.01, 0.3, 6] 0.2458114176988601 7 ...
    0.24937307834625244 17.642887115478516 1546423
576 ['sigmoid', 0.01, 0.4, 2] 0.4014768898487091 ...
    0.37837839126586914 15.481964588165283 1331719
576 ['sigmoid', 0.01, 0.4, 3] 0.3612804412841797 ...
    0.3722485303878784 15.789100646972656 1331719
576 ['sigmoid', 0.01, 0.4, 4] 0.3187502324581146 ...
    0.3566453158855438 16.602768898010254 1495879
576 ['sigmoid', 0.01, 0.4, 5] 0.24546308815479279 ...
    0.24937307834625244 17.111416578292847 1536487
576 ['sigmoid', 0.01, 0.4, 6] 0.247378870844841 ...
    0.24937307834625244 17.312248945236206 1546423
2304 ['tanh', 0.001, 0.2, 2] 0.9884008765220642 ...
    0.43605461716651917 17.46232271194458 5326855
2304 ['tanh', 0.001, 0.2, 3] 0.7799644470214844 ...
    0.4393981695175171 17.731907606124878 5326855
2304 ['tanh', 0.001, 0.2, 4] 0.9900031089782715 ...
    0.4449707567691803 21.802587032318115 7974151
2304 ['tanh', 0.001, 0.2, 5] 0.9963774681091309 ...
    0.44887155294418335 22.893362760543823 8634247
2304 ['tanh', 0.001, 0.2, 6] 0.9967606067657471 ...
    0.4435775876045227 23.501283884048462 8798407
2304 ['tanh', 0.001, 0.3, 2] 0.9875997304916382 ...
    0.4519364833831787 17.251676082611084 5326855
2304 ['tanh', 0.001, 0.3, 3] 0.6858824491500854 ...
    0.42908889055252075 17.488033294677734 5326855
2304 ['tanh', 0.001, 0.3, 4] 0.9828973412513733 ...
    0.4329896867275238 21.477012157440186 7974151
2304 ['tanh', 0.001, 0.3, 5] 0.9963425993919373 ...
    0.4315965473651886 23.164745092391968 8634247
2304 ['tanh', 0.001, 0.3, 6] 0.9965864419937134 ...
    0.43911951780319214 23.502732515335083 8798407
2304 ['tanh', 0.001, 0.4, 2] 0.9873210787773132 ...
    0.4491501748561859 17.78806734085083 5326855
2304 ['tanh', 0.001, 0.4, 3] 0.6061514019966125 ...
    0.43326830863952637 17.73441195487976 5326855
2304 ['tanh', 0.001, 0.4, 4] 0.9727611541748047 ...
    0.4402340352535248 21.54858422279358 7974151
```

```
2304 ['tanh', 0.001, 0.4, 5] 0.9954021573066711 ...
    0.4282529950141907 22.78629755973816 8634247
2304 ['tanh', 0.001, 0.4, 6] 0.9958549737930298 ...
    0.43967679142951965 23.36026692390442 8798407
2304 ['tanh', 0.01, 0.2, 2] 0.6466264724731445 ...
    0.4017832279205322 17.355450868606567 5326855
2304 ['tanh', 0.01, 0.2, 3] 0.5295900106430054 ...
    0.4054054021835327 17.57131814956665 5326855
2304 ['tanh', 0.01, 0.2, 4] 0.6187258362770081 ...
    0.41794371604919434 21.387348175048828 7974151
2304 ['tanh', 0.01, 0.2, 5] 0.7195652723312378 ...
    0.4201727509498596 22.930729150772095 8634247
2304 ['tanh', 0.01, 0.2, 6] 0.8091539144515991 ...
    0.4154360592365265 23.303861379623413 8798407
2304 ['tanh', 0.01, 0.3, 2] 0.6427949666976929 ...
    0.40651991963386536 17.62941575050354 5326855
2304 ['tanh', 0.01, 0.3, 3] 0.4946880638599396 ...
    0.40150460600852966 17.41468334197998 5326855
2304 ['tanh', 0.01, 0.3, 4] 0.5848340392112732 ...
    0.4081917107105255 21.33003544807434 7974151
2304 ['tanh', 0.01, 0.3, 5] 0.6768609285354614 ...
    0.41432154178619385 22.773396015167236 8634247
2304 ['tanh', 0.01, 0.3, 6] 0.7898916602134705 ...
    0.4257453382015228 23.46743369102478 8798407
2304 ['tanh', 0.01, 0.4, 2] 0.6480197906494141 ...
    0.4040122628211975 17.873716115951538 5326855
2304 ['tanh', 0.01, 0.4, 3] 0.4560939073562622 ...
    0.3934243619441986 17.58400011062622 5326855
2304 ['tanh', 0.01, 0.4, 4] 0.5454387068748474 ...
    0.4054054021835327 21.55454730987549 7974151
2304 ['tanh', 0.01, 0.4, 5] 0.6430736184120178 ...
    0.4132070243358612 22.88928198814392 8634247
2304 ['tanh', 0.01, 0.4, 6] 0.7385837435722351 ...
    0.4104207158088684 23.34817600250244 8798407
1152 ['tanh', 0.001, 0.2, 2] 0.981713056564331 ...
    0.4274170994758606 15.856738090515137 2663431
1152 ['tanh', 0.001, 0.2, 3] 0.7396286725997925 ...
    0.44246307015419006 16.289222717285156 2663431
1152 ['tanh', 0.001, 0.2, 4] 0.9869378805160522 ...
    0.4215658903121948 17.042230367660522 3323527
1152 ['tanh', 0.001, 0.2, 5] 0.9964470863342285 ...
    0.4132070243358612 16.60645318031311 3487687
1152 ['tanh', 0.001, 0.2, 6] 0.9967257380485535 ...
    0.42713847756385803 17.300907611846924 3528295
1152 ['tanh', 0.001, 0.3, 2] 0.9817827343940735 ...
    0.4352187216281891 15.857577562332153 2663431
1152 ['tanh', 0.001, 0.3, 3] 0.6420634388923645 ...
    0.43048202991485596 16.300681829452515 2663431
1152 ['tanh', 0.001, 0.3, 4] 0.9784387946128845 ...
    0.4237949252128601 16.762439489364624 3323527
1152 ['tanh', 0.001, 0.3, 5] 0.99543696641922 0.4204513728618622 ...
    16.71262240409851 3487687
1152 ['tanh', 0.001, 0.3, 6] 0.996238112449646 ...
    0.42351630330085754 17.054337978363037 3528295
1152 ['tanh', 0.001, 0.4, 2] 0.9800062775611877 ...
    0.43410420417785645 16.33008360862732 2663431
```

```
1152 ['tanh', 0.001, 0.4, 3] 0.567139208316803 ...
    0.42685985565185547 17.041271924972534 2663431
1152 ['tanh', 0.001, 0.4, 4] 0.9656205177307129 ...
    0.4146001636981964 17.894006490707397 3323527
1152 ['tanh', 0.001, 0.4, 5] 0.9950538277626038 ...
    0.4182223379611969 17.23530077934265 3487687
1152 ['tanh', 0.001, 0.4, 6] 0.9960987567901611 ...
    0.43271106481552124 16.9537513256073 3528295
1152 ['tanh', 0.01, 0.2, 2] 0.6170538663864136 ...
    0.41153523325920105 15.943588495254517 2663431
1152 ['tanh', 0.01, 0.2, 3] 0.5057647228240967 ...
    0.4056840240955353 16.469724416732788 2663431
1152 ['tanh', 0.01, 0.2, 4] 0.580027163028717 ...
    0.40651991963386536 16.73384690284729 3323527
1152 ['tanh', 0.01, 0.2, 5] 0.6400431990623474 ...
    0.40512678027153015 16.789484977722168 3487687
1152 ['tanh', 0.01, 0.2, 6] 0.7182416915893555 ...
    0.40791305899620056 17.552493572235107 3528295
1152 ['tanh', 0.01, 0.3, 2] 0.619805634021759 0.400390088558197 ...
    16.24989604949951 2663431
1152 ['tanh', 0.01, 0.3, 3] 0.46685707569122314 ...
    0.39732515811920166 16.555344820022583 2663431
1152 ['tanh', 0.01, 0.3, 4] 0.5394127368927002 ...
    0.40373364090919495 17.446431398391724 3323527
1152 ['tanh', 0.01, 0.3, 5] 0.6240900158882141 ...
    0.40317636728286743 17.161571264266968 3487687
1152 ['tanh', 0.01, 0.3, 6] 0.6735169887542725 ...
    0.4042908847332001 17.32123041152954 3528295
1152 ['tanh', 0.01, 0.4, 2] 0.6252394914627075 ...
    0.4062412977218628 16.102884531021118 2663431
1152 ['tanh', 0.01, 0.4, 3] 0.43738898634910583 ...
    0.3900808095932007 16.166160345077515 2663431
1152 ['tanh', 0.01, 0.4, 4] 0.5098052620887756 ...
    0.4056840240955353 16.772926807403564 3323527
1152 ['tanh', 0.01, 0.4, 5] 0.5966073274612427 ...
    0.4090275764465332 16.99749231338501 3487687
1152 ['tanh', 0.01, 0.4, 6] 0.6604200601577759 ...
    0.3976037800312042 16.910732984542847 3528295
576 ['tanh', 0.001, 0.2, 2] 0.9538124203681946 ...
    0.4210086464881897 15.753499746322632 1331719
576 ['tanh', 0.001, 0.2, 3] 0.6723327040672302 ...
    0.42908889055252075 15.732567071914673 1331719
576 ['tanh', 0.001, 0.2, 4] 0.9588630795478821 ...
    0.41432154178619385 16.21043086051941 1495879
576 ['tanh', 0.001, 0.2, 5] 0.9939391613006592 0.411813884973526 ...
    16.907785654067993 1536487
576 ['tanh', 0.001, 0.2, 6] 0.9963077902793884 ...
    0.4146001636981964 16.516563892364502 1546423
576 ['tanh', 0.001, 0.3, 2] 0.954300045967102 0.4140429198741913 ...
    15.348542213439941 1331719
576 ['tanh', 0.001, 0.3, 3] 0.587411642074585 ...
    0.42630258202552795 15.399630069732666 1331719
576 ['tanh', 0.001, 0.3, 4] 0.936849057674408 ...
    0.41348564624786377 16.221810579299927 1495879
576 ['tanh', 0.001, 0.3, 5] 0.9919885993003845 ...
    0.42713847756385803 16.42500948905945 1536487
```

```
576 ['tanh', 0.001, 0.3, 6] 0.9937649965286255 ...
    0.4104207158088684 16.524204969406128 1546423
576 ['tanh', 0.001, 0.4, 2] 0.9519662857055664 ...
    0.42407354712486267 15.268642902374268 1331719
576 ['tanh', 0.001, 0.4, 3] 0.5230067372322083 ...
    0.4132070243358612 15.444881439208984 1331719
576 ['tanh', 0.001, 0.4, 4] 0.9158103466033936 ...
    0.41376426815986633 16.296162128448486 1495879
576 ['tanh', 0.001, 0.4, 5] 0.9828276634216309 ...
    0.41237112879753113 16.601536512374878 1536487
576 ['tanh', 0.001, 0.4, 6] 0.991396427154541 ...
    0.41989412903785706 17.3948974609375 1546423
576 ['tanh', 0.01, 0.2, 2] 0.5893970727920532 ...
    0.39676791429519653 15.202072381973267 1331719
576 ['tanh', 0.01, 0.2, 3] 0.48019784688949585 ...
    0.3956533968448639 15.447781085968018 1331719
576 ['tanh', 0.01, 0.2, 4] 0.5306698083877563 ...
    0.39816105365753174 16.0943763256073 1495879
576 ['tanh', 0.01, 0.2, 5] 0.5803406834602356 0.392031192779541 ...
    16.59665608406067 1536487
576 ['tanh', 0.01, 0.2, 6] 0.6210247874259949 0.4017832279205322 ...
    17.959771633148193 1546423
576 ['tanh', 0.01, 0.3, 2] 0.5898846983909607 0.3872945010662079 ...
    15.486732244491577 1331719
576 ['tanh', 0.01, 0.3, 3] 0.4496499300003052 ...
    0.38813039660453796 15.561345100402832 1331719
576 ['tanh', 0.01, 0.3, 4] 0.5046501159667969 0.3964892625808716 ...
    16.53203511238098 1495879
576 ['tanh', 0.01, 0.3, 5] 0.5572119951248169 0.4062412977218628 ...
    16.19697618484497 1536487
576 ['tanh', 0.01, 0.3, 6] 0.6019715070724487 ...
    0.39816105365753174 17.017328023910522 1546423
576 ['tanh', 0.01, 0.4, 2] 0.5852519869804382 ...
    0.39453887939453125 15.907772779464722 1331719
576 ['tanh', 0.01, 0.4, 3] 0.423804372549057 0.3956533968448639 ...
    15.529592037200928 1331719
576 ['tanh', 0.01, 0.4, 4] 0.4853878617286682 ...
    0.40011143684387207 16.43423080444336 1495879
576 ['tanh', 0.01, 0.4, 5] 0.5354766845703125 ...
    0.39593201875686646 16.408893585205078 1536487
576 ['tanh', 0.01, 0.4, 6] 0.5759518146514893 ...
    0.39370298385620117 17.018637657165527 1546423
```

Listing 8: cnn results

```
stride,filter, [act, regular, drop, #layer], Training acc, ...
    Validation acc, Length of time, #parameters
1 3 ['relu', 0.001, 0.2, 7] 0.9938346743583679 ...
    0.5243800282478333 100.7469801902771 9450855
1 3 ['relu', 0.001, 0.2, 10] 0.9878783822059631 ...
    0.5491780638694763 139.16866326332092 4787687
1 3 ['relu', 0.001, 0.3, 7] 0.9883660078048706 0.530509889125824 ...
    95.47065615653992 9450855
1 3 ['relu', 0.001, 0.3, 10] 0.9709498882293701 ...
    0.553078830242157 138.85558891296387 4787687
1 3 ['relu', 0.001, 0.4, 7] 0.9781601428985596 ...
```

```
        0.5366397500038147 95.0034830570221 9450855
1 3 ['relu', 0.001, 0.4, 10] 0.953777551651001 ...
        0.5689607262611389 138.56160140037537 4787687
1 3 ['relu', 0.01, 0.2, 7] 0.6775575876235962 0.4915018081665039 ...
        95.12564063072205 9450855
1 3 ['relu', 0.01, 0.2, 10] 0.5974084734916687 ...
        0.5012538433074951 137.9055745601654 4787687
1 3 ['relu', 0.01, 0.3, 7] 0.652338981628418 0.4920590817928314 ...
        94.72391152381897 9450855
1 3 ['relu', 0.01, 0.3, 10] 0.5612525939941406 ...
        0.4920590817928314 137.94090056419373 4787687
1 3 ['relu', 0.01, 0.4, 7] 0.598314106464386 0.485093355178833 ...
        94.4593141078949 9450855
1 3 ['relu', 0.01, 0.4, 10] 0.5573861598968506 ...
        0.48620784282684326 138.22271823883057 4787687
1 5 ['relu', 0.001, 0.2, 7] 0.9943919777870178 ...
        0.5288380980491638 111.4976315498352 9467751
1 5 ['relu', 0.001, 0.2, 10] 0.9894806742668152 ...
        0.5625522136688232 161.2128026485443 4902887
1 5 ['relu', 0.001, 0.3, 7] 0.988435685634613 0.5452772378921509 ...
        110.99427890777588 9467751
1 5 ['relu', 0.001, 0.3, 10] 0.9803197383880615 ...
        0.5717470049858093 160.68341517448425 4902887
1 5 ['relu', 0.001, 0.4, 7] 0.9838029742240906 ...
        0.5422123074531555 110.5250141620636 9467751
1 5 ['relu', 0.001, 0.4, 10] 0.966665506362915 ...
        0.5678461790084839 160.08203840255737 4902887
1 5 ['relu', 0.01, 0.2, 7] 0.7624786496162415 0.5254945755004883 ...
        110.39865827560425 9467751
1 5 ['relu', 0.01, 0.2, 10] 0.6627538204193115 0.522150993347168 ...
        160.42731165885925 4902887
1 5 ['relu', 0.01, 0.3, 7] 0.6940680742263794 0.5213151574134827 ...
        110.246661901474 9467751
1 5 ['relu', 0.01, 0.3, 10] 0.6271204352378845 ...
        0.5260518193244934 160.36043524742126 4902887
1 5 ['relu', 0.01, 0.4, 7] 0.6834790706634521 0.5126776099205017 ...
        109.99097466468811 9467751
1 5 ['relu', 0.01, 0.4, 10] 0.5849733352661133 0.522150993347168 ...
        160.0080018043518 4902887
1 3 ['sigmoid', 0.001, 0.2, 7] 0.3575882017612457 ...
        0.3647255599498749 99.58895063400269 9450855
1 3 ['sigmoid', 0.001, 0.2, 10] 0.2513149082660675 ...
        0.24937307834625244 146.14503645896912 4787687
1 3 ['sigmoid', 0.001, 0.3, 7] 0.35647356510162354 ...
        0.35720255970954895 99.25110173225403 9450855
1 3 ['sigmoid', 0.001, 0.3, 10] 0.2513149082660675 ...
        0.24937307834625244 145.68951535224915 4787687
1 3 ['sigmoid', 0.001, 0.4, 7] 0.33491238951683044 ...
        0.338534414768219 98.63535737991333 9450855
1 3 ['sigmoid', 0.001, 0.4, 10] 0.2513149082660675 ...
        0.24937307834625244 145.26306200027466 4787687
1 3 ['sigmoid', 0.01, 0.2, 7] 0.2513149082660675 ...
        0.24937307834625244 99.60439085960388 9450855
1 3 ['sigmoid', 0.01, 0.2, 10] 0.2513497471809387 ...
        0.24937307834625244 145.78223037719727 4787687
1 3 ['sigmoid', 0.01, 0.3, 7] 0.2513149082660675 ...
        0.24937307834625244 99.78467798233032 9450855
```

```
1 3 ['sigmoid', 0.01, 0.3, 10] 0.2513149082660675 ...
    0.24937307834625244 146.21798253059387 4787687
1 3 ['sigmoid', 0.01, 0.4, 7] 0.2513149082660675 ...
    0.24937307834625244 99.53646039962769 9450855
1 3 ['sigmoid', 0.01, 0.4, 10] 0.2513149082660675 ...
    0.24937307834625244 146.3772623538971 4787687
1 5 ['sigmoid', 0.001, 0.2, 7] 0.3722177743911743 ...
    0.36862635612487793 115.91524457931519 9467751
1 5 ['sigmoid', 0.001, 0.2, 10] 0.2513149082660675 ...
    0.24937307834625244 170.36932492256165 4902887
1 5 ['sigmoid', 0.001, 0.3, 7] 0.37225261330604553 ...
    0.36918362975120544 115.76063799858093 9467751
1 5 ['sigmoid', 0.001, 0.3, 10] 0.2513149082660675 ...
    0.24937307834625244 170.18098258972168 4902887
1 5 ['sigmoid', 0.001, 0.4, 7] 0.34954196214675903 ...
    0.3555307984352112 115.76917505264282 9467751
1 5 ['sigmoid', 0.001, 0.4, 10] 0.2513149082660675 ...
    0.24937307834625244 170.72689938545227 4902887
1 5 ['sigmoid', 0.01, 0.2, 7] 0.25309136509895325 ...
    0.2516021132469177 115.80425357818604 9467751
1 5 ['sigmoid', 0.01, 0.2, 10] 0.2513149082660675 ...
    0.24937307834625244 170.04838252067566 4902887
1 5 ['sigmoid', 0.01, 0.3, 7] 0.2513149082660675 ...
    0.24937307834625244 115.51301860809326 9467751
1 5 ['sigmoid', 0.01, 0.3, 10] 0.2513149082660675 ...
    0.24937307834625244 169.5968382358551 4902887
1 5 ['sigmoid', 0.01, 0.4, 7] 0.2513149082660675 ...
    0.24937307834625244 115.72228169441223 9467751
1 5 ['sigmoid', 0.01, 0.4, 10] 0.2513149082660675 ...
    0.24937307834625244 169.6595869064331 4902887
1 3 ['tanh', 0.001, 0.2, 7] 0.972865641117096 0.4717191457748413 ...
    102.10168075561523 9450855
1 3 ['tanh', 0.001, 0.2, 10] 0.9453481435775757 ...
    0.5032042264938354 148.73275923728943 4787687
1 3 ['tanh', 0.001, 0.3, 7] 0.915462076663971 0.4781275987625122 ...
    98.96073389053345 9450855
1 3 ['tanh', 0.001, 0.3, 10] 0.8976975679397583 ...
    0.5165784358978271 148.23435854911804 4787687
1 3 ['tanh', 0.001, 0.4, 7] 0.8558639883995056 ...
    0.46308162808418274 98.87562537193298 9450855
1 3 ['tanh', 0.001, 0.4, 10] 0.8646069169044495 ...
    0.4998606741428375 148.27487707138062 4787687
1 3 ['tanh', 0.01, 0.2, 7] 0.5891880393028259 ...
    0.46308162808418274 98.32232666015625 9450855
1 3 ['tanh', 0.01, 0.2, 10] 0.5672088861465454 ...
    0.5045973658561707 147.9903700351715 4787687
1 3 ['tanh', 0.01, 0.3, 7] 0.5709359645843506 0.4661465585231781 ...
    98.60124158859253 9450855
1 3 ['tanh', 0.01, 0.3, 10] 0.5581873059272766 ...
    0.5029256343841553 147.28257131576538 4787687
1 3 ['tanh', 0.01, 0.4, 7] 0.5614267587661743 0.4661465585231781 ...
    98.22979640960693 9450855
1 3 ['tanh', 0.01, 0.4, 10] 0.5521613359451294 ...
    0.49122318625450134 147.15708827972412 4787687
1 5 ['tanh', 0.001, 0.2, 7] 0.9904211163520813 ...
    0.48342156410217285 118.46507668495178 9467751
1 5 ['tanh', 0.001, 0.2, 10] 0.9956111311912537 ...
```

```
    0.49958205223083496 175.93617820739746 4902887
1 5 ['tanh', 0.001, 0.3, 7] 0.9650980234146118 ...
    0.4775703549385071 118.35507392883301 9467751
1 5 ['tanh', 0.001, 0.3, 10] 0.9914661049842834 ...
    0.4934522211551666 175.57788586616516 4902887
1 5 ['tanh', 0.001, 0.4, 7] 0.9151137471199036 ...
    0.47422680258750916 118.4087302684784 9467751
1 5 ['tanh', 0.001, 0.4, 10] 0.9858232736587524 ...
    0.508776843547821 176.18922019004822 4902887
1 5 ['tanh', 0.01, 0.2, 7] 0.610679566860199 0.47060462832450867 ...
    118.9135205745697 9467751
1 5 ['tanh', 0.01, 0.2, 10] 0.6161831021308899 ...
    0.5135135054588318 175.94586610794067 4902887
1 5 ['tanh', 0.01, 0.3, 7] 0.5893273949623108 ...
    0.47088325023651123 118.99840521812439 9467751
1 5 ['tanh', 0.01, 0.3, 10] 0.6108537316322327 ...
    0.5112844705581665 176.56326174736023 4902887
1 5 ['tanh', 0.01, 0.4, 7] 0.5793305039405823 ...
    0.46642518043518066 118.87477135658264 9467751
1 5 ['tanh', 0.01, 0.4, 10] 0.5959803462028503 ...
    0.5051546096801758 176.1305079460144 4902887
2 3 ['relu', 0.001, 0.2, 7, 3] 0.6676303744316101 ...
    0.5090554356575012 26.252427339553833 603495
2 3 ['relu', 0.001, 0.3, 7, 3] 0.6229753494262695 ...
    0.49122318625450134 26.370532274246216 603495
2 3 ['relu', 0.001, 0.4, 7, 3] 0.5881779193878174 ...
    0.4934522211551666 26.994784355163574 603495
2 3 ['relu', 0.01, 0.2, 7, 3] 0.4322686195373535 ...
    0.4154360592365265 26.264293909072876 603495
2 3 ['relu', 0.01, 0.3, 7, 3] 0.42763593792915344 ...
    0.4176650941371918 26.22248601913452 603495
2 3 ['relu', 0.01, 0.4, 7, 3] 0.41408616304397583 ...
    0.4048481583595276 26.2367205619812 603495
2 5 ['relu', 0.001, 0.2, 7, 5] 0.7439130544662476 ...
    0.514906644821167 29.071922779083252 620391
2 5 ['relu', 0.001, 0.3, 7, 5] 0.6934410929679871 ...
    0.5171356797218323 28.97377586364746 620391
2 5 ['relu', 0.001, 0.4, 7, 5] 0.6409836411476135 ...
    0.5112844705581665 29.079978466033936 620391
2 5 ['relu', 0.01, 0.2, 7, 5] 0.4577658474445343 ...
    0.44106993079185486 28.915831804275513 620391
2 5 ['relu', 0.01, 0.3, 7, 5] 0.44961509108543396 ...
    0.42769575119018555 28.861820697784424 620391
2 5 ['relu', 0.01, 0.4, 7, 5] 0.42523249983787537 ...
    0.4257453382015228 28.90476703643799 620391
2 3 ['sigmoid', 0.001, 0.2, 7, 3] 0.25138458609580994 ...
    0.24937307834625244 26.393722772598267 603495
2 3 ['sigmoid', 0.001, 0.3, 7, 3] 0.2513497471809387 ...
    0.24937307834625244 26.842809915542603 603495
2 3 ['sigmoid', 0.001, 0.4, 7, 3] 0.2513149082660675 ...
    0.24937307834625244 26.443675994873047 603495
2 3 ['sigmoid', 0.01, 0.2, 7, 3] 0.24253718554973602 ...
    0.24937307834625244 26.230082750320435 603495
2 3 ['sigmoid', 0.01, 0.3, 7, 3] 0.24577657878398895 ...
    0.24937307834625244 26.42145872116089 603495
2 3 ['sigmoid', 0.01, 0.4, 7, 3] 0.24894632399082184 ...
    0.24937307834625244 26.35397958755493 603495
```

```
2 5 ['sigmoid', 0.001, 0.2, 7, 5] 0.2935664653778076 ...
    0.30064085125923157 29.36149787902832 620391
2 5 ['sigmoid', 0.001, 0.3, 7, 5] 0.2513149082660675 ...
    0.24937307834625244 29.35851740837097 620391
2 5 ['sigmoid', 0.001, 0.4, 7, 5] 0.2513149082660675 ...
    0.24937307834625244 29.28976345062256 620391
2 5 ['sigmoid', 0.01, 0.2, 7, 5] 0.2475530356168747 ...
    0.24937307834625244 29.2594633102417 620391
2 5 ['sigmoid', 0.01, 0.3, 7, 5] 0.2484935075044632 ...
    0.24937307834625244 29.258183240890503 620391
2 5 ['sigmoid', 0.01, 0.4, 7, 5] 0.2505486011505127 ...
    0.24937307834625244 29.296756744384766 620391
2 3 ['tanh', 0.001, 0.2, 7, 3] 0.5310878157615662 ...
    0.46168848872184753 26.989234447479248 603495
2 3 ['tanh', 0.001, 0.3, 7, 3] 0.5184437036514282 ...
    0.46642518043518066 26.53629183769226 603495
2 3 ['tanh', 0.001, 0.4, 7, 3] 0.5059388875961304 ...
    0.4639175236225128 26.420804023742676 603495
2 3 ['tanh', 0.01, 0.2, 7, 3] 0.45240169763565063 ...
    0.4477570354938507 26.40638303756714 603495
2 3 ['tanh', 0.01, 0.3, 7, 3] 0.44021037220954895 ...
    0.4321537911891937 26.54572367668152 603495
2 3 ['tanh', 0.01, 0.4, 7, 3] 0.4371799826622009 ...
    0.43744775652885437 26.430195093154907 603495
2 5 ['tanh', 0.001, 0.2, 7, 5] 0.5788776874542236 ...
    0.4803566336631775 29.563300848007202 620391
2 5 ['tanh', 0.001, 0.3, 7, 5] 0.5649448037147522 ...
    0.4731122851371765 29.513533115386963 620391
2 5 ['tanh', 0.001, 0.4, 7, 5] 0.5451948642730713 ...
    0.4753413200378418 29.448615789413452 620391
2 5 ['tanh', 0.01, 0.2, 7, 5] 0.462433397769928 ...
    0.44747841358184814 29.524261236190796 620391
2 5 ['tanh', 0.01, 0.3, 7, 5] 0.4637570083141327 ...
    0.45528003573417664 29.447476387023926 620391
2 5 ['tanh', 0.01, 0.4, 7, 5] 0.45710405707359314 ...
    0.4471997916698456 30.300016403198242 620391
```