CNN for CIFAR10 with Tensorflow 2[¶](#gjdgxs)

Import TensorFlow[¶](#30j0zll)

In [1]:

*# TensorFlow and tf.keras*  
**import** tensorflow **as** tf  
*# from tensorflow.keras import datasets, layers, models*  
  
*# Helper libraries*  
**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
  
print(tf**.**\_\_version\_\_)

2.4.1

Download and prepare the CIFAR-10 dataset[¶](#1fob9te)

In [2]:

*# Import and load the CIFAR-10 data directly from tf.keras*  
(train\_images, train\_labels), (test\_images, test\_labels) **=** tf**.**keras**.**datasets**.**cifar10**.**load\_data()

In [3]:

class\_names **=** ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer',  
 'Dog', 'Frog', 'Horse', 'Ship', 'Truck']

Explore the data[¶](#3znysh7)

In [4]:

train\_images**.**shape

Out[4]:

(50000, 32, 32, 3)

In [5]:

len(train\_labels)

Out[5]:

50000

In [6]:

train\_labels

Out[6]:

array([[6],  
 [9],  
 [9],  
 ...,  
 [9],  
 [1],  
 [1]], dtype=uint8)

In [7]:

test\_images**.**shape

Out[7]:

(10000, 32, 32, 3)

In [8]:

len(test\_labels)

Out[8]:

10000

Preprocess the data[¶](#2et92p0)

In [9]:

train\_images, test\_images **=** train\_images **/** 255.0, test\_images **/** 255.0

Build the model[¶](#tyjcwt)

Set up parameters[¶](#3dy6vkm)

In [10]:

param\_epoch\_count **=** 20  
param\_batch\_size **=** 32  
param\_act\_fn **=** 'relu'  
param\_optimizer **=** 'adam'  
param\_loss\_fn **=** tf**.**keras**.**losses**.**SparseCategoricalCrossentropy(from\_logits**=True**)

Set up architecture[¶](#1t3h5sf)

In [17]:

y\_train, y\_test **=** train\_labels**.**flatten(), test\_labels**.**flatten()  
  
*# number of classes*  
K **=** len(set(y\_train))  
print("number of classes:", K)  
  
**from** tensorflow.keras.layers **import** Input, Conv2D, Dense, Flatten, \  
 Dropout, GlobalMaxPooling2D, MaxPooling2D, \  
 BatchNormalization  
  
*#### Create the convolutional base*  
i **=** Input(shape**=**train\_images[0]**.**shape)  
  
x **=** Conv2D(32, (3, 3), activation**=**'relu', padding**=**'same')(i)  
x **=** BatchNormalization()(x)  
x **=** Conv2D(32, (3, 3), activation**=**'relu', padding**=**'same')(x)  
x **=** BatchNormalization()(x)  
x **=** MaxPooling2D((2, 2))(x)  
  
x **=** Conv2D(64, (3, 3), activation**=**'relu', padding**=**'same')(x)  
x **=** BatchNormalization()(x)  
x **=** Conv2D(64, (3, 3), activation**=**'relu', padding**=**'same')(x)  
x **=** BatchNormalization()(x)  
x **=** MaxPooling2D((2, 2))(x)  
  
x **=** Conv2D(128, (3, 3), activation**=**'relu', padding**=**'same')(x)  
x **=** BatchNormalization()(x)  
x **=** Conv2D(128, (3, 3), activation**=**'relu', padding**=**'same')(x)  
x **=** BatchNormalization()(x)  
x **=** MaxPooling2D((2, 2))(x)  
  
*#### Add Dense layers on top*  
x **=** Flatten()(x)  
x **=** Dropout(0.2)(x)  
x **=** Dense(1024, activation**=**'relu')(x)  
x **=** Dropout(0.2)(x)  
x **=** Dense(K, activation**=**'softmax')(x)  
  
model **=** tf**.**keras**.**models**.**Model(i, x)

number of classes: 10

In [18]:

model**.**summary()

Model: "model"  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param #   
=================================================================  
input\_1 (InputLayer) [(None, 32, 32, 3)] 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_3 (Conv2D) (None, 32, 32, 32) 896   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
batch\_normalization (BatchNo (None, 32, 32, 32) 128   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_4 (Conv2D) (None, 32, 32, 32) 9248   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
batch\_normalization\_1 (Batch (None, 32, 32, 32) 128   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d\_2 (MaxPooling2 (None, 16, 16, 32) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_5 (Conv2D) (None, 16, 16, 64) 18496   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
batch\_normalization\_2 (Batch (None, 16, 16, 64) 256   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_6 (Conv2D) (None, 16, 16, 64) 36928   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
batch\_normalization\_3 (Batch (None, 16, 16, 64) 256   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d\_3 (MaxPooling2 (None, 8, 8, 64) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_7 (Conv2D) (None, 8, 8, 128) 73856   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
batch\_normalization\_4 (Batch (None, 8, 8, 128) 512   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
conv2d\_8 (Conv2D) (None, 8, 8, 128) 147584   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
batch\_normalization\_5 (Batch (None, 8, 8, 128) 512   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
max\_pooling2d\_4 (MaxPooling2 (None, 4, 4, 128) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
flatten\_1 (Flatten) (None, 2048) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dropout (Dropout) (None, 2048) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_2 (Dense) (None, 1024) 2098176   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dropout\_1 (Dropout) (None, 1024) 0   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_3 (Dense) (None, 10) 10250   
=================================================================  
Total params: 2,397,226  
Trainable params: 2,396,330  
Non-trainable params: 896  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Compile the model[¶](#4d34og8)

In [19]:

model**.**compile(optimizer**=**param\_optimizer,  
 loss**=**param\_loss\_fn,  
 metrics**=**['accuracy'])

Train the model[¶](#2s8eyo1)

In [20]:

history **=** model**.**fit(train\_images, train\_labels, batch\_size**=**32, epochs**=**param\_epoch\_count, validation\_data**=**(test\_images, test\_labels))

Epoch 1/20  
1563/1563 [==============================] - 13s 8ms/step - loss: 1.6950 - accuracy: 0.4487 - val\_loss: 0.9943 - val\_accuracy: 0.6566  
Epoch 2/20  
1563/1563 [==============================] - 12s 7ms/step - loss: 0.8759 - accuracy: 0.6952 - val\_loss: 0.8773 - val\_accuracy: 0.6991  
Epoch 3/20  
1563/1563 [==============================] - 11s 7ms/step - loss: 0.7077 - accuracy: 0.7541 - val\_loss: 0.6951 - val\_accuracy: 0.7625  
Epoch 4/20  
1563/1563 [==============================] - 11s 7ms/step - loss: 0.5803 - accuracy: 0.7994 - val\_loss: 0.7189 - val\_accuracy: 0.7701  
Epoch 5/20  
1563/1563 [==============================] - 11s 7ms/step - loss: 0.4717 - accuracy: 0.8379 - val\_loss: 0.6566 - val\_accuracy: 0.7831  
Epoch 6/20  
1563/1563 [==============================] - 11s 7ms/step - loss: 0.4088 - accuracy: 0.8591 - val\_loss: 0.6379 - val\_accuracy: 0.7982  
Epoch 7/20  
1563/1563 [==============================] - 11s 7ms/step - loss: 0.3424 - accuracy: 0.8811 - val\_loss: 0.7040 - val\_accuracy: 0.7857  
Epoch 8/20  
1563/1563 [==============================] - 11s 7ms/step - loss: 0.2743 - accuracy: 0.9051 - val\_loss: 0.7340 - val\_accuracy: 0.7834  
Epoch 9/20  
1563/1563 [==============================] - 11s 7ms/step - loss: 0.2342 - accuracy: 0.9177 - val\_loss: 0.6666 - val\_accuracy: 0.8053  
Epoch 10/20  
1563/1563 [==============================] - 11s 7ms/step - loss: 0.2036 - accuracy: 0.9284 - val\_loss: 0.6525 - val\_accuracy: 0.8142  
Epoch 11/20  
1563/1563 [==============================] - 11s 7ms/step - loss: 0.1763 - accuracy: 0.9397 - val\_loss: 0.6226 - val\_accuracy: 0.8315  
Epoch 12/20  
1563/1563 [==============================] - 11s 7ms/step - loss: 0.1497 - accuracy: 0.9485 - val\_loss: 0.6984 - val\_accuracy: 0.8290  
Epoch 13/20  
1563/1563 [==============================] - 11s 7ms/step - loss: 0.1436 - accuracy: 0.9509 - val\_loss: 0.6758 - val\_accuracy: 0.8283  
Epoch 14/20  
1563/1563 [==============================] - 12s 7ms/step - loss: 0.1274 - accuracy: 0.9563 - val\_loss: 0.6991 - val\_accuracy: 0.8250  
Epoch 15/20  
1563/1563 [==============================] - 11s 7ms/step - loss: 0.1190 - accuracy: 0.9590 - val\_loss: 0.6958 - val\_accuracy: 0.8285  
Epoch 16/20  
1563/1563 [==============================] - 12s 7ms/step - loss: 0.1101 - accuracy: 0.9632 - val\_loss: 0.8000 - val\_accuracy: 0.8222  
Epoch 17/20  
1563/1563 [==============================] - 11s 7ms/step - loss: 0.1111 - accuracy: 0.9636 - val\_loss: 0.8107 - val\_accuracy: 0.8251  
Epoch 18/20  
1563/1563 [==============================] - 11s 7ms/step - loss: 0.0950 - accuracy: 0.9683 - val\_loss: 0.7232 - val\_accuracy: 0.8254  
Epoch 19/20  
1563/1563 [==============================] - 12s 7ms/step - loss: 0.0947 - accuracy: 0.9681 - val\_loss: 0.6816 - val\_accuracy: 0.8357  
Epoch 20/20  
1563/1563 [==============================] - 11s 7ms/step - loss: 0.0829 - accuracy: 0.9725 - val\_loss: 0.8145 - val\_accuracy: 0.8420

Evaluate the model[¶](#17dp8vu)

In [21]:

fig, ax **=** plt**.**subplots()  
fig**.**set\_size\_inches(18.5, 10.5)  
  
ax**.**xaxis**.**set\_ticks(np**.**arange(0, param\_epoch\_count**+**1))  
ax**.**yaxis**.**set\_ticks(np**.**arange(0, 1.1, 0.1))  
ax**.**grid()  
  
train\_acc **=** history**.**history['accuracy']  
test\_acc **=** history**.**history['val\_accuracy']  
  
color\_train**=**'#238ef0'  
color\_test**=**'#01b1b3'  
  
plt**.**rcParams**.**update({'font.size': 18})  
  
plt**.**plot(np**.**arange(1,param\_epoch\_count**+**1), train\_acc, label**=**'Mokymo', c**=**color\_train, linewidth**=**4)  
plt**.**plot(np**.**arange(1,param\_epoch\_count**+**1), test\_acc, label**=**'Testavimo', c**=**color\_test, linewidth**=**4)  
  
plt**.**plot(param\_epoch\_count, train\_acc[**-**1],'co', c**=**color\_train, markersize**=**15)  
plt**.**text(param\_epoch\_count, train\_acc[**-**1], "{:.4f}"**.**format(train\_acc[**-**1]), c**=**color\_train)  
  
plt**.**plot(param\_epoch\_count, test\_acc[**-**1], 'co', c**=**color\_test, markersize**=**15)  
plt**.**text(param\_epoch\_count, test\_acc[**-**1], "{:.4f}"**.**format(test\_acc[**-**1]), c**=**color\_test)  
  
plt**.**xlabel('Epochos', fontsize**=**18)  
plt**.**ylabel('Klasifikavimo tikslumas', fontsize**=**18)  
  
plt**.**xlim([1, param\_epoch\_count])  
plt**.**ylim([0.0, 1.0])  
  
plt**.**legend(loc**=**'lower right')

Out[21]:

<matplotlib.legend.Legend at 0x7fad709d84e0>

Classification Report and Confusion Matrix[¶](#3rdcrjn)

In [24]:

**from** sklearn.metrics **import** confusion\_matrix , classification\_report  
**from** sklearn.utils **import** shuffle  
**from** mlxtend.plotting **import** plot\_confusion\_matrix  
  
predicted\_test\_images **=** model**.**predict(test\_images)  
predicted\_test\_image\_classes **=** [np**.**argmax(element) **for** element **in** predicted\_test\_images]  
  
print("\nClassification Report: \n\n",  
 classification\_report(test\_labels, predicted\_test\_image\_classes))  
  
cmat\_multiclass **=** confusion\_matrix(test\_labels, predicted\_test\_image\_classes)  
  
print("\n\nConfusion Matrix:")  
  
fig, ax **=** plot\_confusion\_matrix(conf\_mat**=**cmat\_multiclass,  
 colorbar**=True**,  
 show\_absolute**=True**,  
 show\_normed**=False**,  
 class\_names**=**class\_names,  
 figsize**=**(10,10))  
  
plt**.**show()

Classification Report:   
  
 precision recall f1-score support  
  
 0 0.84 0.89 0.86 1000  
 1 0.90 0.94 0.92 1000  
 2 0.74 0.78 0.76 1000  
 3 0.71 0.69 0.70 1000  
 4 0.79 0.85 0.82 1000  
 5 0.83 0.72 0.77 1000  
 6 0.87 0.89 0.88 1000  
 7 0.89 0.89 0.89 1000  
 8 0.91 0.93 0.92 1000  
 9 0.94 0.85 0.89 1000  
  
 accuracy 0.84 10000  
 macro avg 0.84 0.84 0.84 10000  
weighted avg 0.84 0.84 0.84 10000  
  
  
  
Confusion Matrix:

Classification results of some images[¶](#26in1rg)

In [25]:

*# calculate the softmax of a vector*  
*# in order to convert vector of outputs*  
*# to vector of probabilities*  
**def** softmax(vector):  
 e **=** np**.**exp(vector)  
 **return** e **/** e**.**sum()

In [29]:

*# select how many images to present*  
sample\_img\_count **=** 30  
  
test\_image\_predictions **=** model**.**predict(test\_images[:sample\_img\_count])  
*#print(test\_image\_predictions)*  
  
*# convert vectors of outputs to vectors of probabilities*  
test\_image\_predictions **=** [softmax(x) **for** x **in** test\_image\_predictions]  
  
*# get the classes our CNN predicted*  
predicted\_classes **=** [np**.**argmax(element) **for** element **in** test\_image\_predictions]  
  
*# get the correct classes*  
correct\_classes **=** list(test\_labels[:sample\_img\_count]**.**reshape(**-**1,))  
  
**for** i **in** range(sample\_img\_count):  
 ylabel **=** f'[{i**+**1}]'  
   
 plt**.**figure(figsize **=** (15,2))  
 plt**.**imshow(test\_images[i])  
  
 xlabel **=** (  
 f'Correct class: {class\_names[correct\_classes[i]]} (#{correct\_classes[i]})\n'  
 f'Predicted class: {class\_names[predicted\_classes[i]]}(#{predicted\_classes[i]})\n'  
 )  
 ylabel **=** f'[{i**+**1}] '  
   
 plt**.**xlabel(xlabel, fontsize**=**16)  
 plt**.**ylabel(ylabel, fontsize**=**18, fontweight**=**'bold', color**=**'#01b1b3', rotation**=**0)

/usr/local/lib/python3.6/dist-packages/ipykernel\_launcher.py:19: RuntimeWarning: More than 20 figures have been opened. Figures created through the pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly closed and may consume too much memory. (To control this warning, see the rcParam `figure.max\_open\_warning`).

The classification probabilites for each class from above:[¶](#lnxbz9)

In [30]:

print('\nOutput probabilities:\n ')  
  
**for** i **in** range(sample\_img\_count):  
 print(f'[{i**+**1}]:\n')  
  
 np**.**set\_printoptions(precision**=**4, suppress**=True**)  
 print(test\_image\_predictions[i])  
 np**.**set\_printoptions()  
 print('\n\n')

Output probabilities:  
   
[1]:  
  
[0.0853 0.0853 0.0853 0.232 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853]  
  
  
  
[2]:  
  
[0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.232 0.0853]  
  
  
  
[3]:  
  
[0.0854 0.0855 0.0854 0.0854 0.0854 0.0854 0.0854 0.0854 0.2317 0.0854]  
  
  
  
[4]:  
  
[0.2162 0.0892 0.0866 0.0868 0.0863 0.0863 0.0863 0.0863 0.0874 0.0886]  
  
  
  
[5]:  
  
[0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.232 0.0853 0.0853 0.0853]  
  
  
  
[6]:  
  
[0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.232 0.0853 0.0853 0.0853]  
  
  
  
[7]:  
  
[0.0854 0.2304 0.0854 0.0854 0.0854 0.0854 0.0854 0.0854 0.0854 0.0861]  
  
  
  
[8]:  
  
[0.0856 0.0856 0.087 0.0856 0.0856 0.0856 0.2285 0.0856 0.0856 0.0856]  
  
  
  
[9]:  
  
[0.0853 0.0853 0.0853 0.232 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853]  
  
  
  
[10]:  
  
[0.0853 0.232 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853]  
  
  
  
[11]:  
  
[0.1849 0.0879 0.0888 0.1073 0.0892 0.0883 0.0881 0.0879 0.0894 0.0882]  
  
  
  
[12]:  
  
[0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.232 ]  
  
  
  
[13]:  
  
[0.086 0.086 0.0866 0.0872 0.0888 0.2215 0.086 0.086 0.086 0.086 ]  
  
  
  
[14]:  
  
[0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.232 0.0853 0.0853]  
  
  
  
[15]:  
  
[0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.232 ]  
  
  
  
[16]:  
  
[0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.232 0.0853]  
  
  
  
[17]:  
  
[0.0855 0.0855 0.0855 0.0867 0.0855 0.2292 0.0855 0.0855 0.0855 0.0855]  
  
  
  
[18]:  
  
[0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.232 0.0853 0.0853]  
  
  
  
[19]:  
  
[0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.232 0.0853]  
  
  
  
[20]:  
  
[0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.232 0.0853 0.0853 0.0853]  
  
  
  
[21]:  
  
[0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.232 0.0853 0.0853]  
  
  
  
[22]:  
  
[0.1897 0.0876 0.1098 0.0876 0.0876 0.0876 0.0876 0.0876 0.0876 0.0876]  
  
  
  
[23]:  
  
[0.0886 0.0885 0.1371 0.0885 0.155 0.0885 0.0885 0.0885 0.0885 0.0885]  
  
  
  
[24]:  
  
[0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.232 ]  
  
  
  
[25]:  
  
[0.0853 0.0853 0.0853 0.0853 0.232 0.0853 0.0853 0.0853 0.0853 0.0853]  
  
  
  
[26]:  
  
[0.0858 0.0858 0.2243 0.0858 0.0882 0.0858 0.0868 0.0858 0.0858 0.0858]  
  
  
  
[27]:  
  
[0.0865 0.0865 0.0865 0.0952 0.2125 0.0869 0.0865 0.0865 0.0865 0.0865]  
  
  
  
[28]:  
  
[0.232 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853]  
  
  
  
[29]:  
  
[0.0853 0.0854 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.2319]  
  
  
  
[30]:  
  
[0.0853 0.0853 0.0853 0.0853 0.0853 0.0853 0.232 0.0853 0.0853 0.0853]

In [ ]: