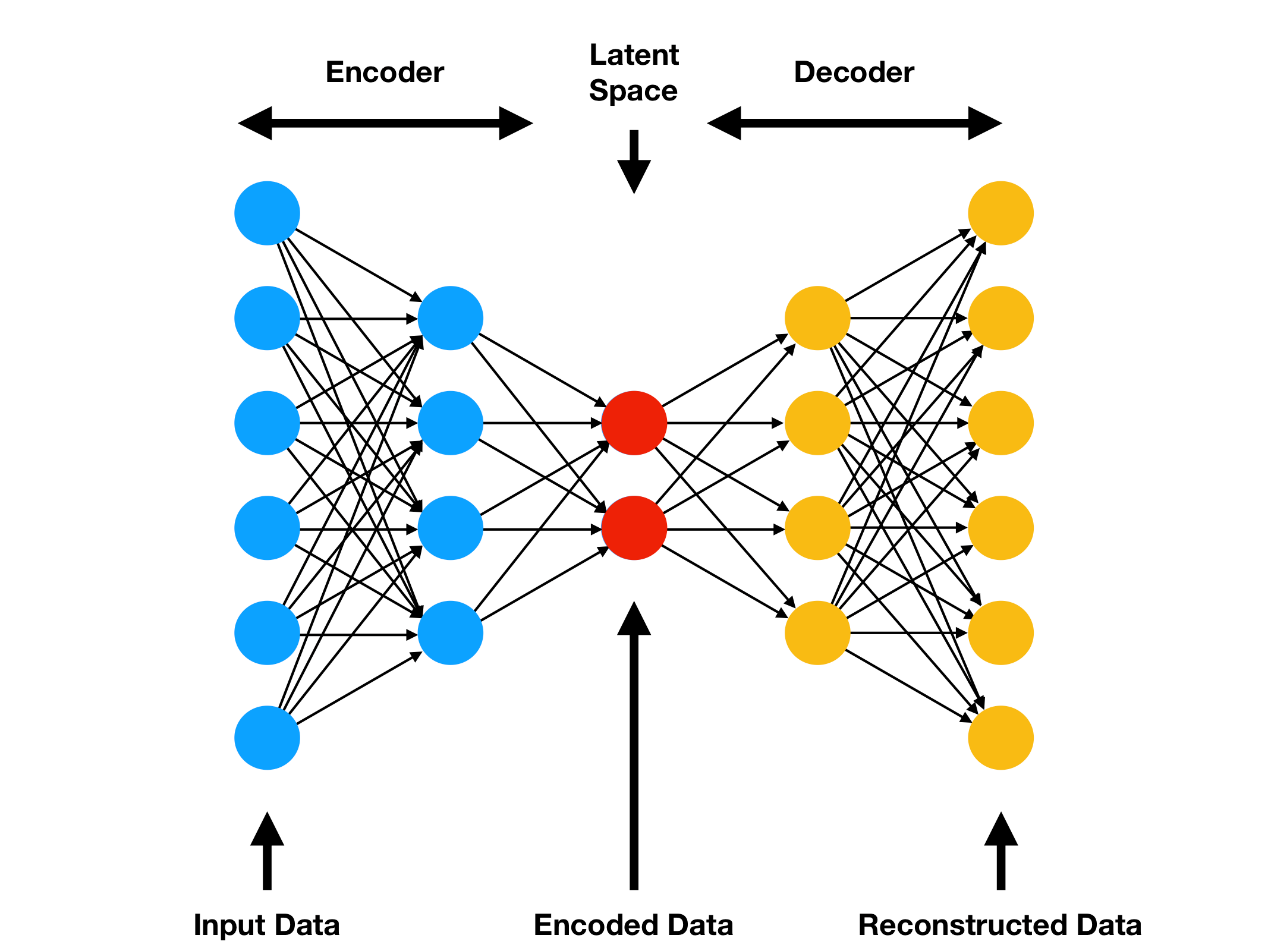
# Creation and training of the autoencoder

In this report, we will create a transceiver using the python programming language, which is based on the theory of autoencoders.

The autoencoder is a neural network which consists of 2 basic structures the transmitter, which accepts as input symbols which have a specific length of digits (vectors) and after processing them in a specific way, transmits them to the receiver, who receives the message and tries to rebuild it.



The figure above shows an example of an auto-encoder structure. we will create a different structure which can represent the transmitter and the receiver in a telecommunication channel. Then we will carry out the training of the system, without the mediation of a channel between the transmitter and the receiver (end to end learning). But let's start building our system.

The first thing we need to do is import the libraries we will use.

from keras.models import Sequential

from keras.layers import Dense

from keras.utils.vis\_utils import plot\_model

import numpy as np

import tensorflow as tf

from matplotlib import pyplot

import pandas as pd

Next we have to enter in the system the data we collected in files from the OptiSystem 17.1 software.

# import data from files

X\_OOK= np.array(pd.read\_excel(r'C:\Users\User\Desktop\Πτυχιακή\OOK-in.xlsx', nrows=32763))

y1\_OOK= np.array(pd.read\_excel(r'C:\Users\User\Desktop\Πτυχιακή\OOK-out-500m.xlsx', nrows=32763))

y2\_OOK= np.array(pd.read\_excel(r'C:\Users\User\Desktop\Πτυχιακή\OOK-out-1km.xlsx', nrows=32763))

y3\_OOK= np.array(pd.read\_excel(r'C:\Users\User\Desktop\Πτυχιακή\OOK-out-1.5km.xlsx', nrows=32763))

y4\_OOK= np.array(pd.read\_excel(r'C:\Users\User\Desktop\Πτυχιακή\OOK-out-2km.xlsx', nrows=32763))

X\_PPM= np.array(pd.read\_excel(r'C:\Users\User\Desktop\Πτυχιακή\PPM-in.xlsx', nrows=32763))

y1\_PPM= np.array(pd.read\_excel(r'C:\Users\User\Desktop\Πτυχιακή\PPM-out-500m.xlsx', nrows=32763))

y2\_PPM= np.array(pd.read\_excel(r'C:\Users\User\Desktop\Πτυχιακή\PPM-out-1km.xlsx', nrows=32763))

y3\_PPM= np.array(pd.read\_excel(r'C:\Users\User\Desktop\Πτυχιακή\PPM-out-1.5km.xlsx', nrows=32763))

y4\_PPM= np.array(pd.read\_excel(r'C:\Users\User\Desktop\Πτυχιακή\PPM-out-2km.xlsx', nrows=32763))

Before inputs and outputs are entered into the system, they must be converted to one-hot vectors in order to obtain a binary form. This is implemented with the following code.

# convert data to one-hot vectors

X\_OOK= tf.one\_hot(X\_OOK, depth=64, dtype= tf.int32)

y1\_OOK= tf.one\_hot(y1\_OOK, depth=64, dtype= tf.int32)

y2\_OOK= tf.one\_hot(y2\_OOK, depth=64, dtype= tf.int32)

y3\_OOK= tf.one\_hot(y3\_OOK, depth=64, dtype= tf.int32)

y4\_OOK= tf.one\_hot(y4\_OOK, depth=64, dtype= tf.int32)

X\_PPM= tf.one\_hot(X\_PPM, depth=64, dtype= tf.int32)

y1\_PPM= tf.one\_hot(y1\_PPM, depth=64, dtype= tf.int32)

y2\_PPM= tf.one\_hot(y2\_PPM, depth=64, dtype= tf.int32)

y3\_PPM= tf.one\_hot(y3\_PPM, depth=64, dtype= tf.int32)

y4\_PPM= tf.one\_hot(y4\_PPM, depth=64, dtype= tf.int32)

After completing the data preparation for the system, we can now create our neural network. It will consist of an encoder as a transmitter and a decoder as a receiver that will receive the encoded message and try to recompose it.

The encoder consists of 4 layers one with 64 neurons 2 with 256 neurons and one with 48 neurons all fully connected.

Respectively, the decoder consists of 4 layers also 2 to 256 neurons and 2 to 64 neurons and here fully connected. The following is the code that implements the encoder and the decoder and then connects them to each other.

#define the encoder as transmitter part

def get\_encoder(n\_inputs= 64):

model= Sequential()

model.add(Dense(64, activation= None, input\_shape=(None, n\_inputs)))

model.add(Dense(256, activation= 'relu'))

model.add(Dense(256, activation= 'relu'))

model.add(Dense(48, activation= 'sigmoid'))

model.compile(loss= 'mse' , optimizer= 'adam')

print(model.summary())

plot\_model(model, to\_file='encoder.png', show\_shapes=True, show\_layer\_names=True)

return model

#define the decoder as receiver part

def get\_decoder():

model= Sequential()

model.add(Dense(256, activation= 'relu', input\_shape=(None, 48)))

model.add(Dense(256, activation= 'relu'))

model.add(Dense(64, activation= None))

model.add(Dense(64, activation= 'softmax'))

model.compile(loss= 'mse' , optimizer= 'adam')

print(model.summary())

plot\_model(model, to\_file='decoder.png', show\_shapes=True, show\_layer\_names=True)

return model

#define the autoencoder

def get\_autoencoder(Transmitter, Receiver):

model= Sequential()

model.add(Transmitter)

model.add(Receiver)

model.compile(loss= 'mse' , optimizer= 'adam' , metrics=[ 'accuracy' ])

plot\_model(model, to\_file='autoencoder.png', show\_shapes=True, show\_layer\_names=True)

return model

Transmitter= get\_encoder()

Receiver= get\_decoder()

The structure of the encoder, the decoder and final system can be seen in the pictures below.

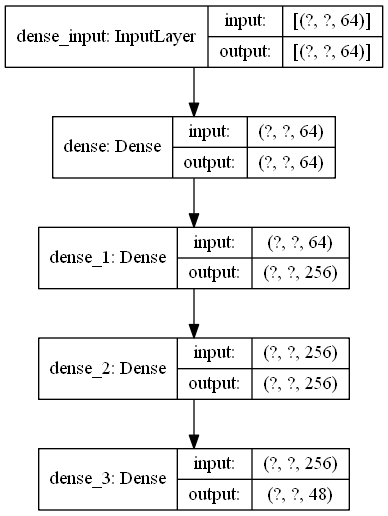
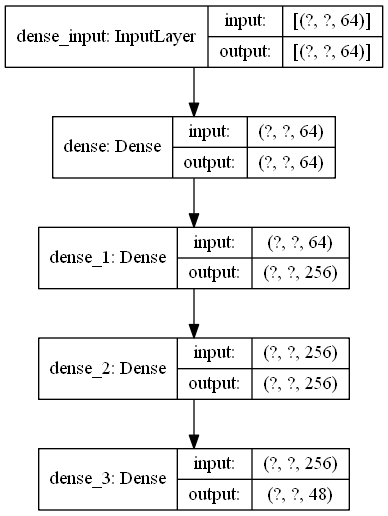


Figure 1 Encoder

Figure 2 Decoder

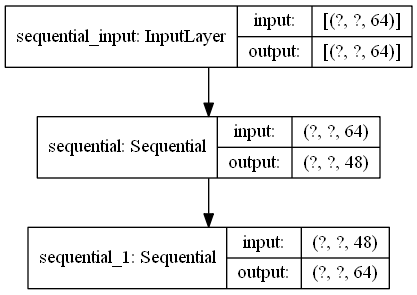


Figure 3 Autoencoder

Once we have created our system, it is time to carry out its training. The training will be done with the data mentioned above. The data were obtained for an optical wireless system with OOK and PPM configuration for distances between transmitter and receiver from 500 meters to 2 kilometers with measurements per 500 meters. We create functions that train the system for each input-output pair first for OOK and then for PPM.

#train the model

def train\_model\_OOK1(model):

hist= model.fit(X\_OOK, y1\_OOK, epochs=100, batch\_size=250)

loss= hist.history['loss']

return loss

def train\_model\_OOK2(model):

hist= model.fit(X\_OOK, y2\_OOK, epochs=100, batch\_size=250)

loss= hist.history['loss']

return loss

def train\_model\_OOK3(model):

hist= model.fit(X\_OOK, y3\_OOK, epochs=100, batch\_size=250)

loss= hist.history['loss']

return loss

def train\_model\_OOK4(model):

hist= model.fit(X\_OOK, y4\_OOK, epochs=100, batch\_size=250)

loss= hist.history['loss']

return loss

def train\_model\_PPM1(model):

hist= model.fit(X\_PPM, y1\_PPM, epochs=100, batch\_size=250)

loss= hist.history['loss']

return loss

def train\_model\_PPM2(model):

hist= model.fit(X\_PPM, y2\_PPM, epochs=100, batch\_size=250)

loss= hist.history['loss']

return loss

def train\_model\_PPM3(model):

hist= model.fit(X\_PPM, y3\_PPM, epochs=100, batch\_size=250)

loss= hist.history['loss']

return loss

def train\_model\_PPM4(model):

hist= model.fit(X\_PPM, y4\_PPM, epochs=100, batch\_size=250)

loss= hist.history['loss']

return loss

After the training we create diagrams that illustrate the course of the loss of the system during the training. The diagrams show in pairs the course of training for OOK and PPM together for the same distance of transmitter and receiver.

model= get\_autoencoder(Transmitter, Receiver)

pyplot.plot(train\_model\_OOK1(model), label='Autoencoder-OOK')

model= get\_autoencoder(Transmitter, Receiver)

pyplot.plot(train\_model\_PPM1(model), label='Autoencoder-PPM')

pyplot.xlabel('Epochs', fontsize=18)

pyplot.ylabel('Loss', fontsize=18)

pyplot.title('Loss of Autoencoder after OOK vs after PPM for 500m', fontsize=18)

pyplot.grid()

pyplot.legend()

pyplot.show()

model= get\_autoencoder(Transmitter, Receiver)

pyplot.plot(train\_model\_OOK2(model), label='Autoencoder-OOK')

model= get\_autoencoder(Transmitter, Receiver)

pyplot.plot(train\_model\_PPM2(model), label='Autoencoder-PPM')

pyplot.xlabel('Epochs', fontsize=18)

pyplot.ylabel('Loss', fontsize=18)

pyplot.title('Loss of Autoencoder after OOK vs after PPM for 1km', fontsize=18)

pyplot.grid()

pyplot.legend()

pyplot.show()

model= get\_autoencoder(Transmitter, Receiver)

pyplot.plot(train\_model\_OOK3(model), label='Autoencoder-OOK')

model= get\_autoencoder(Transmitter, Receiver)

pyplot.plot(train\_model\_PPM3(model), label='Autoencoder-PPM')

pyplot.xlabel('Epochs', fontsize=18)

pyplot.ylabel('Loss', fontsize=18)

pyplot.title('Loss of Autoencoder after OOK vs after PPM for 1.5km', fontsize=18)

pyplot.grid()

pyplot.legend()

pyplot.show()

model= get\_autoencoder(Transmitter, Receiver)

pyplot.plot(train\_model\_OOK4(model), label='Autoencoder-OOK')

model= get\_autoencoder(Transmitter, Receiver)

pyplot.plot(train\_model\_PPM4(model), label='Autoencoder-PPM')

pyplot.xlabel('Epochs', fontsize=18)

pyplot.ylabel('Loss', fontsize=18)

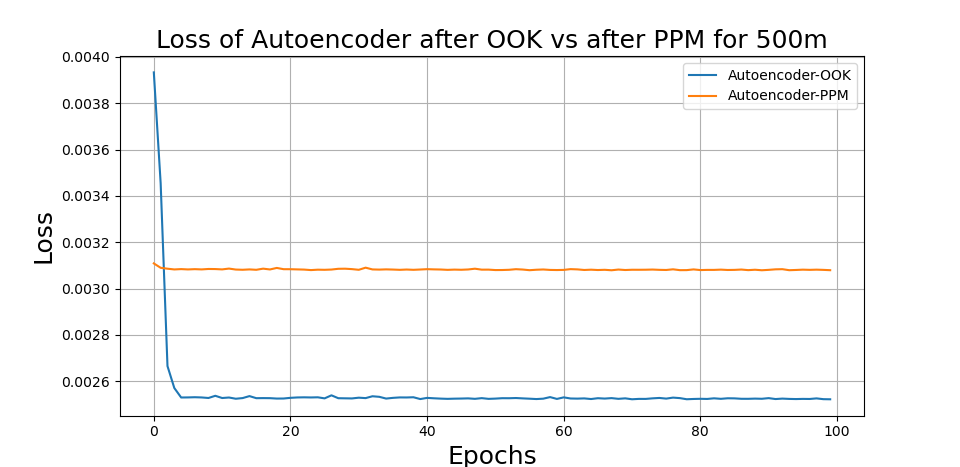
pyplot.title('Loss of Autoencoder after OOK vs after PPM for 2km', fontsize=18)

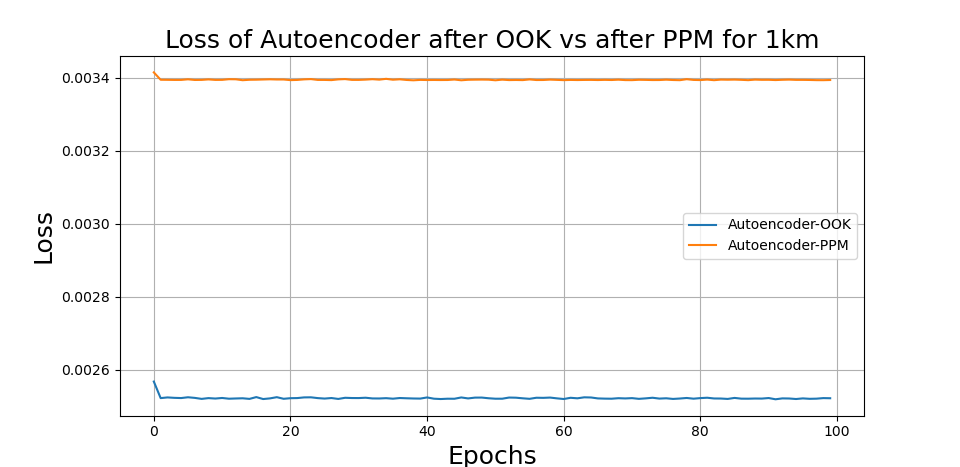
pyplot.grid()

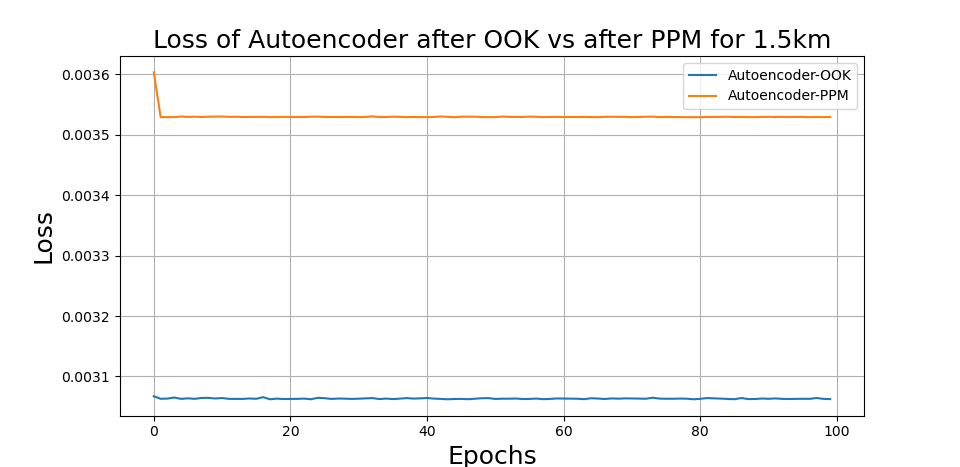
pyplot.legend()

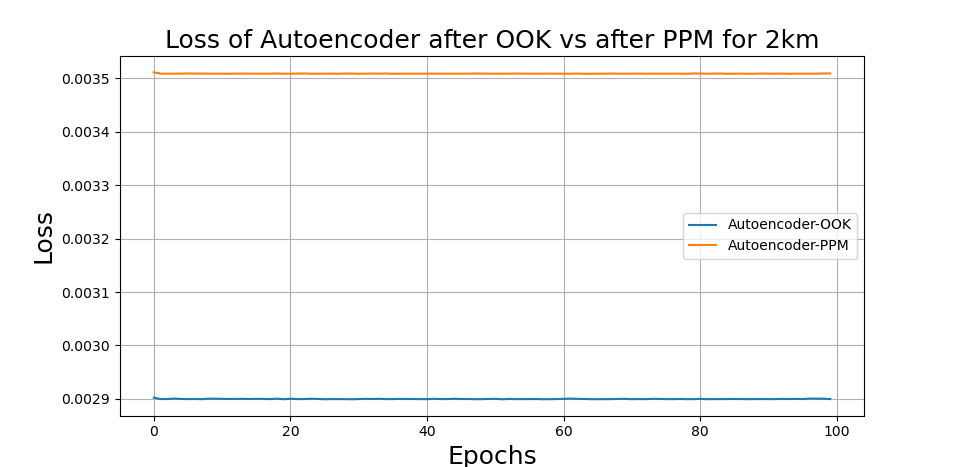
pyplot.show()

The resulting diagrams are as follows.









We observe that the losses of the system almost immediately stabilize in terms of their prices and only in the first chart they start from higher percentages and then decrease. This is because the system almost immediately learns to handle the data it receives so the losses are stabilized to the extent that the system performs in each case, so they can not be reduced anymore.

With the following code we can determine the final value of accuracy and losses for each case of training.

#evaluate the model for OOK & PPM

loss, acc = model.evaluate(X\_OOK, y1\_OOK, verbose=1)

print('OOK: Loss: %f, Accuracy: %f' % (loss, acc\*100)) #the accuracy (x100%)

loss, acc = model.evaluate(X\_OOK, y2\_OOK, verbose=1)

print('OOK: Loss: %f, Accuracy: %f' % (loss, acc\*100)) #the accuracy (x100%)

loss, acc = model.evaluate(X\_OOK, y3\_OOK, verbose=1)

print('OOK: Loss: %f, Accuracy: %f' % (loss, acc\*100)) #the accuracy (x100%)

loss, acc = model.evaluate(X\_OOK, y4\_OOK, verbose=1)

print('OOK: Loss: %f, Accuracy: %f' % (loss, acc\*100)) #the accuracy (x100%)

loss, acc = model.evaluate(X\_PPM, y1\_PPM, verbose=1)

print('PPM: Loss: %f, Accuracy: %f' % (loss, acc\*100)) #the accuracy (x100%)

loss, acc = model.evaluate(X\_PPM, y2\_PPM, verbose=1)

print('PPM: Loss: %f, Accuracy: %f' % (loss, acc\*100)) #the accuracy (x100%)

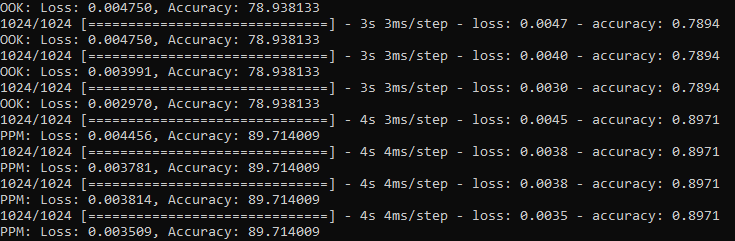
loss, acc = model.evaluate(X\_PPM, y3\_OOK, verbose=1)

print('PPM: Loss: %f, Accuracy: %f' % (loss, acc\*100)) #the accuracy (x100%)

loss, acc = model.evaluate(X\_PPM, y4\_OOK, verbose=1)

print('PPM: Loss: %f, Accuracy: %f' % (loss, acc\*100)) #the accuracy (x100%)

The results are shown in the image below.



We can see that the accuracy in PPM is higher, 89.7% compared to OOK, 78.9%.

Finally we are ready to make predictions for the exit in any case. This is done with the following code.

#make predictions for the probability of output y

yhatOOK1= model.predict(y1\_OOK, verbose=0)

yhatOOK2= model.predict(y2\_OOK, verbose=0)

yhatOOK3= model.predict(y3\_OOK, verbose=0)

yhatOOK4= model.predict(y4\_OOK, verbose=0)

yhatPPM1= model.predict(y1\_PPM, verbose=0)

yhatPPM2= model.predict(y2\_PPM, verbose=0)

yhatPPM3= model.predict(y3\_PPM, verbose=0)

yhatPPM4= model.predict(y4\_PPM, verbose=0)

The exit after the predictions will be a list of probabilities. To return the data in binary format we convert it with the following criterion. In each message the highest probability is converted to 1 and the rest to 0.

# convert predicted data in zeros and ones and print them

maxOOK1= np.amax(yhatOOK1, axis=2)

maxOOK2= np.amax(yhatOOK2, axis=2)

maxOOK3= np.amax(yhatOOK3, axis=2)

maxOOK4= np.amax(yhatOOK4, axis=2)

maxPPM1= np.amax(yhatPPM1, axis=2)

maxPPM2= np.amax(yhatPPM2, axis=2)

maxPPM3= np.amax(yhatPPM3, axis=2)

maxPPM4= np.amax(yhatPPM4, axis=2)

for i in range(len(yhatOOK1[:,0,0])):

for j in range(len(yhatOOK1[0,:,0])):

for k in range(len(yhatOOK1[0,0,:])):

if yhatOOK1[i,j,k]==maxOOK1[i,j]:

yhatOOK1[i,j,k]=1

else: yhatOOK1[i,j,k]=0

print(yhatOOK1)

for i in range(len(yhatOOK2[:,0,0])):

for j in range(len(yhatOOK2[0,:,0])):

for k in range(len(yhatOOK2[0,0,:])):

if yhatOOK2[i,j,k]==maxOOK2[i,j]:

yhatOOK2[i,j,k]=1

else: yhatOOK2[i,j,k]=0

print(yhatOOK2)

for i in range(len(yhatOOK3[:,0,0])):

for j in range(len(yhatOOK3[0,:,0])):

for k in range(len(yhatOOK3[0,0,:])):

if yhatOOK3[i,j,k]==maxOOK3[i,j]:

yhatOOK3[i,j,k]=1

else: yhatOOK3[i,j,k]=0

print(yhatOOK3)

for i in range(len(yhatOOK4[:,0,0])):

for j in range(len(yhatOOK4[0,:,0])):

for k in range(len(yhatOOK4[0,0,:])):

if yhatOOK4[i,j,k]==maxOOK4[i,j]:

yhatOOK4[i,j,k]=1

else: yhatOOK4[i,j,k]=0

print(yhatOOK4)

for i in range(len(yhatPPM1[:,0,0])):

for j in range(len(yhatPPM1[0,:,0])):

for k in range(len(yhatPPM1[0,0,:])):

if yhatPPM1[i,j,k]==maxPPM1[i,j]:

yhatPPM1[i,j,k]=1

else: yhatPPM1[i,j,k]=0

print(yhatPPM1)

for i in range(len(yhatPPM2[:,0,0])):

for j in range(len(yhatPPM2[0,:,0])):

for k in range(len(yhatPPM2[0,0,:])):

if yhatPPM2[i,j,k]==maxPPM2[i,j]:

yhatPPM2[i,j,k]=1

else: yhatPPM2[i,j,k]=0

print(yhatPPM2)

for i in range(len(yhatPPM3[:,0,0])):

for j in range(len(yhatPPM3[0,:,0])):

for k in range(len(yhatPPM3[0,0,:])):

if yhatPPM3[i,j,k]==maxPPM3[i,j]:

yhatPPM3[i,j,k]=1

else: yhatPPM3[i,j,k]=0

print(yhatPPM3)

for i in range(len(yhatPPM4[:,0,0])):

for j in range(len(yhatPPM4[0,:,0])):

for k in range(len(yhatPPM4[0,0,:])):

if yhatPPM4[i,j,k]==maxPPM4[i,j]:

yhatPPM4[i,j,k]=1

else: yhatPPM4[i,j,k]=0

print(yhatPPM4)