FACE RECOGNITION

CONVOLUTINAL NEURAL NETWORK

Bratucu Ana Maria 1231A

Artificial intelligence has known an impressive growth in the last few years and it has become the go to technology that is used in many fields.

We used for our project a convolutional neural network, a Deep learning Algorithm that classifies a collection of images into more categories based on input pictures [1].

The picture data set we used is the keras implemented data set, called Cifar100. As an observation before we start talking about the code, it was inspired from the Antonio Gulli, Sujit Pal - Deep Learning with Keras (2017, Packt) book [6].

*from keras.datasets import cifar100*

Keras is an neural networks API, in python that uses TensorFlow or Theano [2]. Going back to Cifar100, it contains 60k pictures, with the resolution of 32x32 pixels that run in an RGB color [3]. There are 20 superclasses that represents the main categories in the data set and each of this category has 100 subclasses. For example, the “human” superclass has subclass of “girl” or “boy”. The data set is split into 50k pictures of training and 10k of testing [4].

For the libraries, we use the usual matplotlib and numpy for plotting and respectively, for mathematical computations. We load the data into train and test parts

*(X\_train, y\_train), (X\_test, y\_test) = cifar100.load\_data()*

We normalize the data, so all the values are within the range of 0 and 1 [5]. It takes the data and normalize it as an array because this is the way to use the pictures afterwards. Also we convert it to float.

*X\_train = X\_train.astype('float32')*

*X\_test = X\_test.astype('float32')*

*X\_train = X\_train / 255.0*

*X\_test = X\_test / 255.0*

After, we need to convert the labels into categorical matrix structure, using the function:

*y\_train = np\_utils.to\_categorical(y\_train)*

Then we start to build the neural network model:

*model = Sequential()*

We have 3 convolutional network layers. We use theono kernel that uses Numpy, therefore the input shape needs to have the number of channels at the end like this:

*input\_shape=(32, 32, 3)*

After, we train the model and predict the testing data

*history = model.fit(X\_train, y\_train, batch\_size=BATCH\_SIZE,*

*epochs=NB\_EPOCH, validation\_split=VALIDATION\_SPLIT,*

*verbose=VERBOSE)*

*y\_pred=model.predict(X\_test)*

Training the network found itself to be a tricky task. We used 12 epochs and it took about 3 hours. We save the generated weights because we don’t want to train the model for 3 hours just so we can predict a picture. Using these weights, we will be able to test the model more easily.

*model\_json = model.to\_json()*

*open('cifar10\_architecture.json', 'w').write(model\_json)*

*model.save\_weights('cifar100\_weights.h5', overwrite=True)*

We have selected ReLu function for the activation, to add nonlinearity to the network [7].

*model.add(Activation('relu'))*

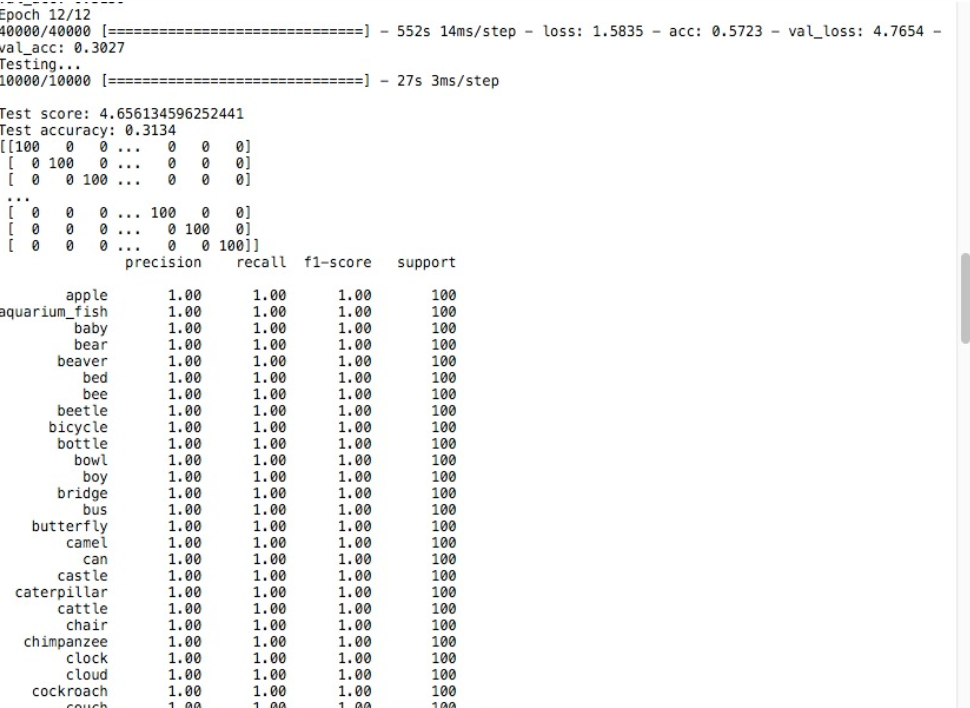
We selected max pooling layers that are 2x2 because we the pictures are represented by a 2D matrix [8].

*model.add(MaxPooling2D(pool\_size=(2, 2), dim\_ordering="th"))*

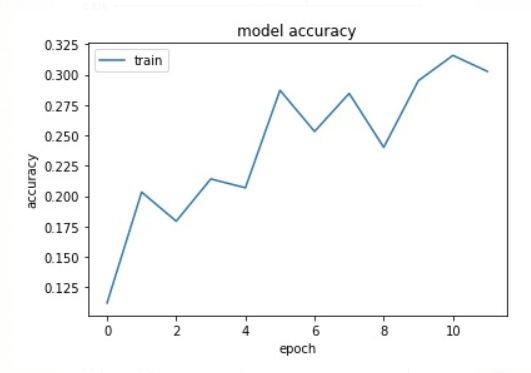
Ouput layer uses a softmax activation to calculate the loss. To test how the model trains with respect to epochs, we train the model on 12 epochs.

*model.add(Dense(100, activation='softmax'))*

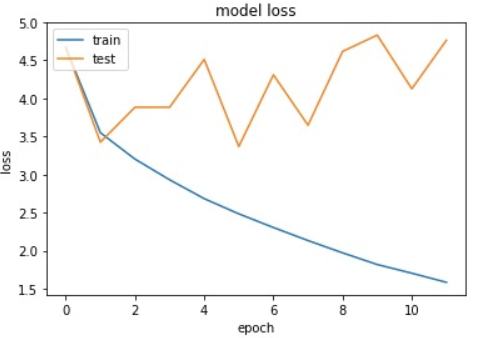
Taking our model into account, we can see that the convolution neural networks models performs:



As we can see, the accuracy is fairly low: only 31%. Therefore, we expect bad predictions when we will test the network. A solution to this problem is to train the model more, on more epochs.



Although the accuracy is lower than it should be, we can see an improvement in time, as we go further in epochs.



For testing our neural network, we use the same model as before, when we trained it and we load the obtained weights into our trained model:

*model=modelTrain('cifar100\_weights.h5')*

Then, we load an image we want to predict the output to, having the resolution of 32 pixel, so that we can compare it to the trained pictures which have the 32 resolution as a standard:

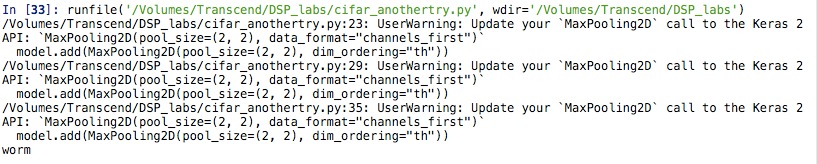
*image = load\_img('face.jpeg', target\_size=(32, 32))*

We predict the image and print the label with respect to the obtained output. This way, we classify our picture into a category:

*classs=model.predict(image)*

*print(labels[a[0]])*

For recognizing a face, we add a picture containing a person and the model should recognize it as a face, not any other object or animal. Therefore, the result is:



Our result is “worm” which is wrong and that is due to the low accuracy we achieved on our model. A solution for improving it would be to train it more time, on more epochs.

In conclusion, our neural network doesn’t have a high performance because we didn’t train it enough, but that could be avoided with more epochs trained.

**CODE FOR TRAINING THE MODEL:**

from keras.datasets import cifar100

from keras.utils import np\_utils

from keras.models import Sequential

from keras.layers.core import Dense, Dropout, Activation, Flatten

from keras.layers.convolutional import Conv2D, MaxPooling2D

from keras.optimizers import SGD, Adam, RMSprop

from keras.layers import MaxoutDense, BatchNormalization

from keras.constraints import maxnorm

from sklearn.metrics import classification\_report, confusion\_matrix

import numpy as np

import matplotlib.pyplot as plt

#from quiver\_engine import server

# CIFAR\_100 is a set of 60K images 32x32 pixels on 3 channels

IMG\_CHANNELS = 3

IMG\_ROWS = 32

IMG\_COLS = 32

#constant

BATCH\_SIZE = 128

NB\_EPOCH = 20

NB\_CLASSES = 10

VERBOSE = 1

VALIDATION\_SPLIT = 0.2

OPTIM = RMSprop()

#load dataset

(X\_train, y\_train), (X\_test, y\_test) = cifar100.load\_data()

print(X\_train[0])

# normalize inputs from 0-255 to 0.0-1.0

X\_train = X\_train.astype('float32')

X\_test = X\_test.astype('float32')

X\_train = X\_train / 255.0

X\_test = X\_test / 255.0

# one hot encode outputs

y\_train = np\_utils.to\_categorical(y\_train)

y\_test = np\_utils.to\_categorical(y\_test)

num\_classes = y\_test.shape[1]

model = Sequential()

model.add(Conv2D(32, (3, 3), input\_shape=(32, 32, 3), padding='same', kernel\_constraint=maxnorm(3)))

model.add(Activation('relu'))

model.add(BatchNormalization())

model.add(Dropout(0.2))

model.add(MaxPooling2D(pool\_size=(2, 2), dim\_ordering="th"))

model.add(Conv2D(64, (3, 3), padding='same', kernel\_constraint=maxnorm(3)))

model.add(Activation('relu'))

model.add(BatchNormalization())

model.add(Dropout(0.2))

model.add(MaxPooling2D(pool\_size=(2, 2), dim\_ordering="th"))

model.add(Conv2D(128, (3, 3), padding='same', kernel\_constraint=maxnorm(3)))

model.add(Activation('relu'))

model.add(BatchNormalization())

model.add(Dropout(0.2))

model.add(MaxPooling2D(pool\_size=(2, 2), dim\_ordering="th"))

model.add(Flatten())

model.add(Dense(1024, activation='relu', kernel\_constraint=maxnorm(3)))

model.add(Dropout(0.2))

model.add(Dense(512, activation='relu', kernel\_constraint=maxnorm(3)))

model.add(Dropout(0.2))

model.add(Dense(100, activation='softmax'))

# train

#optim = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)

model.compile(loss='categorical\_crossentropy', optimizer=OPTIM,metrics=['accuracy'])

history = model.fit(X\_train, y\_train, batch\_size=BATCH\_SIZE,

epochs=NB\_EPOCH, validation\_split=VALIDATION\_SPLIT,

verbose=VERBOSE)

print('Testing...')

score = model.evaluate(X\_test, y\_test,batch\_size=BATCH\_SIZE, verbose=VERBOSE)

print("\nTest score:", score[0])

print('Test accuracy:', score[1])

y\_pred=model.predict(X\_test)

y\_pred=np.round(y\_test)

#t\_test=np.round(t\_test)

#save model

model\_json = model.to\_json()

open('cifar10\_architecture.json', 'w').write(model\_json)

model.save\_weights('cifar100\_weights.h5', overwrite=True)

matrix = confusion\_matrix(y\_test.argmax(axis=1), y\_pred.argmax(axis=1))

print(matrix)

labels = [

'apple', 'aquarium\_fish', 'baby', 'bear', 'beaver', 'bed', 'bee', 'beetle',

'bicycle', 'bottle', 'bowl', 'boy', 'bridge', 'bus', 'butterfly', 'camel',

'can', 'castle', 'caterpillar', 'cattle', 'chair', 'chimpanzee', 'clock',

'cloud', 'cockroach', 'couch', 'crab', 'crocodile', 'cup', 'dinosaur',

'dolphin', 'elephant', 'flatfish', 'forest', 'fox', 'girl', 'hamster',

'house', 'kangaroo', 'keyboard', 'lamp', 'lawn\_mower', 'leopard', 'lion',

'lizard', 'lobster', 'man', 'maple\_tree', 'motorcycle', 'mountain', 'mouse',

'mushroom', 'oak\_tree', 'orange', 'orchid', 'otter', 'palm\_tree', 'pear',

'pickup\_truck', 'pine\_tree', 'plain', 'plate', 'poppy', 'porcupine',

'possum', 'rabbit', 'raccoon', 'ray', 'road', 'rocket', 'rose',

'sea', 'seal', 'shark', 'shrew', 'skunk', 'skyscraper', 'snail', 'snake',

'spider', 'squirrel', 'streetcar', 'sunflower', 'sweet\_pepper', 'table',

'tank', 'telephone', 'television', 'tiger', 'tractor', 'train', 'trout',

'tulip', 'turtle', 'wardrobe', 'whale', 'willow\_tree', 'wolf', 'woman',

'worm'

]

print(classification\_report(y\_test, y\_pred, target\_names=labels))

# list all data in history

print(history.history.keys())

# summarize history for accuracy

#plt.plot(mo)

plt.plot(history.history['val\_acc'])

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

# summarize history for loss

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

**CODE FOR TESTING THE MODEL:**

import numpy as np

from keras.models import Sequential, model\_from\_json

from keras.layers import Dense, Activation

from keras.layers import Dropout

from keras.layers import Flatten

from keras.constraints import maxnorm

from keras.optimizers import SGD

from keras.layers.convolutional import Conv2D

from keras.layers.convolutional import MaxPooling2D

from keras.utils import np\_utils

from keras import backend as K

from keras.layers import MaxoutDense, BatchNormalization

from keras.preprocessing.image import load\_img

from keras.preprocessing.image import img\_to\_array

def modelTrain(weights\_path=None):

model = Sequential()

model.add(Conv2D(32, (3, 3), input\_shape=(32, 32, 3), padding='same', kernel\_constraint=maxnorm(3)))

model.add(Activation('relu'))

model.add(BatchNormalization())

model.add(Dropout(0.2))

model.add(MaxPooling2D(pool\_size=(2, 2), dim\_ordering="th"))

model.add(Conv2D(64, (3, 3), padding='same', kernel\_constraint=maxnorm(3)))

model.add(Activation('relu'))

model.add(BatchNormalization())

model.add(Dropout(0.2))

model.add(MaxPooling2D(pool\_size=(2, 2), dim\_ordering="th"))

model.add(Conv2D(128, (3, 3), padding='same', kernel\_constraint=maxnorm(3)))

model.add(Activation('relu'))

model.add(BatchNormalization())

model.add(Dropout(0.2))

model.add(MaxPooling2D(pool\_size=(2, 2), dim\_ordering="th"))

model.add(Flatten())

model.add(Dense(1024, activation='relu', kernel\_constraint=maxnorm(3)))

model.add(Dropout(0.2))

model.add(Dense(512, activation='relu', kernel\_constraint=maxnorm(3)))

model.add(Dropout(0.2))

model.add(Dense(100, activation='softmax'))

if weights\_path:

model.load\_weights(weights\_path)

return model

model=modelTrain('cifar100\_weights.h5')

image = load\_img('face.jpeg', target\_size=(32, 32))

# convert the image pixels to a numpy array

image = img\_to\_array(image)

# reshape data for the model

image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))

labels = [

'apple', 'aquarium\_fish', 'face', 'bear', 'beaver', 'bed', 'bee', 'beetle',

'bicycle', 'bottle', 'bowl', 'face', 'bridge', 'bus', 'butterfly', 'camel',

'can', 'castle', 'caterpillar', 'cattle', 'chair', 'chimpanzee', 'clock',

'cloud', 'cockroach', 'couch', 'crab', 'crocodile', 'cup', 'dinosaur',

'dolphin', 'elephant', 'flatfish', 'forest', 'fox', 'face', 'hamster',

'house', 'kangaroo', 'keyboard', 'lamp', 'lawn\_mower', 'leopard', 'lion',

'lizard', 'lobster', 'man', 'maple\_tree', 'motorcycle', 'mountain', 'mouse',

'mushroom', 'oak\_tree', 'orange', 'orchid', 'otter', 'palm\_tree', 'pear',

'pickup\_truck', 'pine\_tree', 'plain', 'plate', 'poppy', 'porcupine',

'possum', 'rabbit', 'raccoon', 'ray', 'road', 'rocket', 'rose',

'sea', 'seal', 'shark', 'shrew', 'skunk', 'skyscraper', 'snail', 'snake',

'spider', 'squirrel', 'streetcar', 'sunflower', 'sweet\_pepper', 'table',

'tank', 'telephone', 'television', 'tiger', 'tractor', 'train', 'trout',

'tulip', 'turtle', 'wardrobe', 'whale', 'willow\_tree', 'wolf', 'face',

'worm'

]

classs=model.predict(image)

tclass=classs.max()

pos=tclass==classs

a=np.where(pos)[1]

#print(a[0])

print(labels[a[0]])

**References**

[1]: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[2]: https://keras.io/

[3]: https://en.wikipedia.org/wiki/CIFAR-10

[4]: https://www.kaggle.com/c/ml2016-7-cifar-100

[5]: https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/

[6]: Antonio Gulli, Sujit Pal - Deep Learning with Keras (2017, Packt)

[7]: https://medium.com/@bian0628/image-classification-cifar-10-dc1c23db46d5

[8]: https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks