PROJECT REPORT

ON

"Classification of SEM using machine learning "

Submitted in partial fulfilment of Degree of

Master of Science

(Physics)

Submitted by

Mr. Darshan Vijay Satone

(M.Sc 2nd year Physics)

Under the guidance of

Dr. Prashant .M. Gade

Professor,

Department Of Physics

Post Graduate Teaching Department of Physics

Rashtrasant Tukadoji Maharaj Nagpur University

,Nagpur

[2018-19]

CERTIFICATE

It is certify that the project entitled **"Classification of SEM images using machine learning"** which is being submitted **Mr. Darshan Vijay Satone** in the partial fulfilment towards attaining the Degree of **" MASTER OF SCIENCE "** in **Physics** (faculty of science) . Post Graduate Teaching Department of the Physics , Rashtrashant Tukadoji Maharaj Nagpur University , Nagpur . It is the record of her work carried out by her under my guidance . I found that the work is comprehensive and completed in all respect and ready for the submission to the University.

25/04/19

Head Deptt. of Physics
Rashtrasant Tukadoji Maharaj
Nagpur University, Nagpur

Approved By

Dr. Prashant .M.Gade
Professor & Head ,
Department Of Physics ,
Rashtrasant Tukadoji Maharaj Nagpur University,
Nagpur – 440033,

DECLARATION

I hereby, declare that the piece of work presented in this project report entitled, **"Classification of SEM images using machine learning"** has been carried out by me under the guidance of
Prof .**P.M.Gade.**
The work has been carried out at the **Post Graduate Teaching Department of the Physics , Rashtrashant Tukadoji Maharaj Nagpur University , Nagpur** it has been submitted as the project work in Degree of Master of Science in Physics . The result and conculsion given in the project work has been arrived as a consequence of the research work carried out by me.

Place : Nagpur
Date : 25/04/19

(Mr. Darshan Vijay Satone)

CERTIFIED BY :
Dr. Prashant .M.Gade
Professor & Head ,
Department Of Physics ,
Rashtrasant Tukadoji Maharaj Nagpur University,Nagpur .

INDEX

Project Work

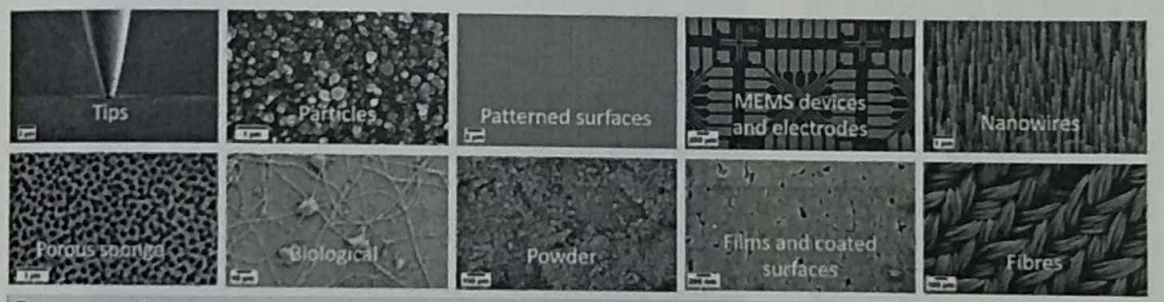
Goals

Since I'm a newcomer to this issue, I took a ready-made model from Keras blog. My goals were to understand how the model works, describe it; customise the model and teach it to recognise photos Scanning Electron Microscope of different materials.

Techniques and tools

I used Python syntax for this project. As a framework I used Keras, which is a high-level neural network API written in Python. But Keras can't work by itself, it needs a backend for low-level operations. Thus I installed a dedicated software library — Google's TensorFlow.

As a development environment I used the Anaconda Environment (Spyder). I used Matplotlib for visualisation.



Tag	Category	N images
L0	Porous_Sponge	171
L1	Patterned_surface	3310
L2	Particles	3412
L3	Films_Coated_Surface	308
L4	Powder	895
L5	Tips	1561
L6	Nanowires	3656
L7	Biological	953
L8	MEMS_devices_and_electrodes	4158
L9	Fibres	153
	TOTAL	18577

NFFA-EUROPE - SEM Dataset

by Aversa, Rossella; Modarres, Mohammad Hadi; Cozzini, Stefano; Ciancio, Regina;

Feb 19, 2018

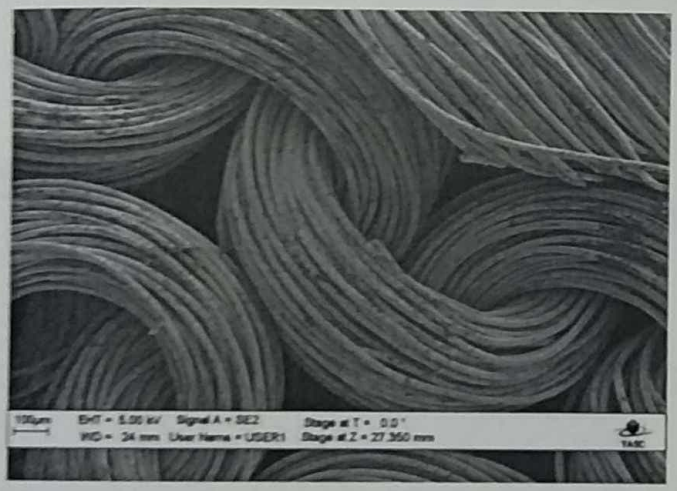
Last updated at Apr 16, 2018

For network training and tasting I used a dataset of photos Scanning Electron Microscope

Convolutional neural networks and image classification :

Convolutional neural networks (CNN) is a special architecture of arti>cial neural networks, proposed by Yann LeCun in 1988. CNN uses some features of the visual cortex. One of the most popular uses of this architecture is image classification. For example Facebook uses CNN for automatic tagging algorithms, Amazon — for generating product recommendations and Google — for search through among users' photos.

Let us consider the use of CNN for image classification in more detail. The main task of image classification is acceptance of the input image and the following definition of its class. This is a skill that people learn from their birth and are able to easily determine that the image in the picture is an elephant. But the computer sees the pictures quite differently:



01010100 01101000 01101001 0111001
00100000 01101001 01110011 0010000
01110100 01101000 01100101 0010000
01110100 01110101 01110100 0110111
01110010 01101001 01100001 0110110
00100000 01110100 01101111 0010000
01101100 01100101 01100001 0111001
01101110 00100000 01100010 0110100
01101110 01100001 01110010 0111100
00101110 00100000 01001001 0010000
01101000 01101111 01110000 0110010
00100000 01111001 01101111 0111010
00100000 01100101 01101110 0110101
01101111 01111001 00100000 0110100
01110100 00100001

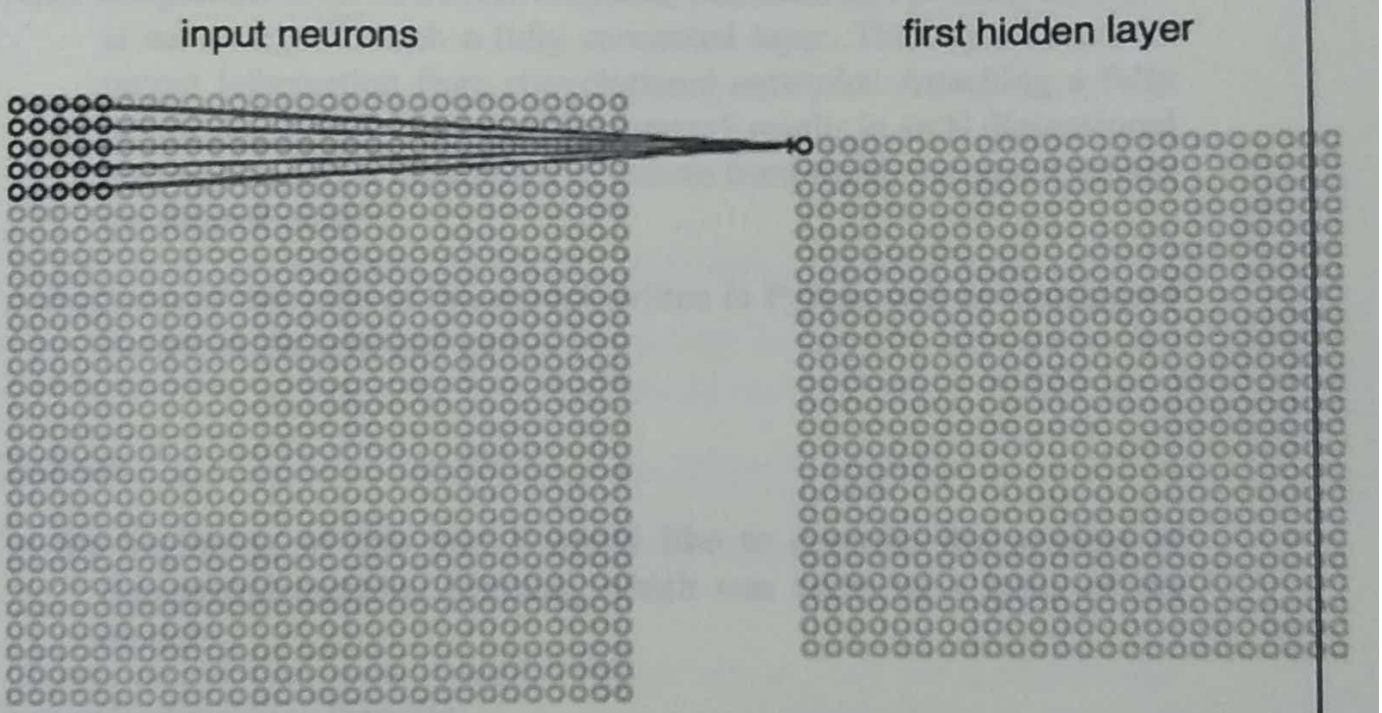
Instead of the image, the computer sees an array of pixels. For example, if image size is 300 x 300. In this case, the size of the array will be 300x300x3. Where 300 is width, next 300 is height and 3 is RGB channel values and 300x300x1 if it is black and white image. The computer is assigned a value from 0 to 255 to each of these numbers. This value describes the intensity of the pixel at each point.

To solve this problem the computer looks for the characteristics of the base level. In human understanding such characteristics are for example

the trunk or large ears. For the computer, these characteristics are boundaries or curvatures. And then through the groups of convolutional layers the computer constructs more abstract concepts.

In more detail: the image is passed through a series of convolutional, nonlinear, pooling layers and fully connected layers, and then generates the output.

The Convolution layer is always the first. The image (matrix with pixel values) is entered into it. Imagine that the reading of the input matrix begins at the top left of image. Next the software selects a smaller matrix there, which is called a @lter (or neuron, or core). Then the alter produces convolution, i.e. moves along the input image. The alter task is to multiply its values by the original pixel values. All these multiplications are summed up. One number is obtained in the end. Since the alter has read the image only in the upper left corner, it moves further and further right by 1 unit performing a similar operation. After passing the alter across all positions, a matrix is obtained, but smaller then a input matrix.



This operation, from a human perspective, is analogous to identifying boundaries and simple colours on the image. But in order to recognise the properties of a higher level such as the trunk or large ears the whole network is needed.

The network will consist of several convolutional networks mixed with nonlinear and pooling layers. When the image passes through one convolution layer, the output of the first layer becomes the input for the second layer. And this happens with every further convolutional layer.

The nonlinear layer is added after each convolution operation. It has an activation function, which brings nonlinear property. Without this property a network would not be sufficiently intense and will not be able to model the response variable (as a class label).

The pooling layer follows the nonlinear layer. It works with width and height of the image and performs a downsampling operation on them. As a result the image volume is reduced. This means that if some features (as for example boundaries) have already been identified in the previous convolution operation, then a detailed image is no longer needed for further processing, and it is compressed to less detailed pictures.

After completion of series of convolutional, nonlinear and pooling layers, it is necessary to attach a fully connected layer. This layer takes the output information from convolutional networks. Attaching a fully connected layer to the end of the network results in an N dimensional vector, where N is the amount of classes from which the model selects the desired class.

A fragment of the code of this model written in Python will be considered further in the practical part.

Process

In the beginning of this part I would like to describe the process of Supervised machine learning, which was taken as a basis of the model.

Supervised machine learning

It is one of the ways of machine learning where the model is trained by input data and expected output data.

To create such model, it is necessary to go through the following phases:

Model Construction

Model Training

Model Testing

Model Evaluation

Model construction depends on machine learning algorithms. In this projects case, it was neural networks.

Such an algorithm looks like:

1. begin with its object: model = Sequential()

2. then consist of layers with their types: model.add(type_of_layer())

3. after adding a sufficient number of layers the model is compiled. At this moment Keras communicates with TensorFlow for construction of the model. During model compilation it is important to write a loss function and an optimizer algorithm. It looks like: model.comile(loss= 'name_of_loss_function', optimizer= 'name_of_opimazer_alg') The loss function shows the accuracy of each prediction made by the model.

Before model training it is important to scale data for their further use.

After model construction it is time for model training. In this phase, the model is trained using training data and expected output for this data.

It's look this way: model.>t(training_data, expected_output).

Progress is visible on the console when the script runs. At the end it will report the final accuracy of the model.

Once the model has been trained it is possible to carry out model testing. During this phase a second set of data is loaded. This data set has never been seen by the model and therefore it's true accuracy will be veri>ed.

After the model training is complete, and it is understood that the model shows the right result, it can be saved by: model.save("name_of_>le.h5").

Finally, the saved model can be used in the real world. The name of this phase is model evaluation. This means that the model can be used to evaluate new data.

Classification Model

For my project I used only four classes i.e. fibres, biological, porous, coated .

Here I would like to describe the code that was taken as the basis of this project. It is considered that a deep learning model needs a large amount of data. But the model given in this script is excellent for training with a small amount of data. Because of that I took only available photos per class for training and divided photos of each class into 80% for training and 20% for testing .

Using little data is possible when the image is preprocessing with Keras ImageDataGenerator class. This class can create a number of random transformations, which helps to increase the number of images when it is needed.

CODE

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Mar 13 13:25:55 2019

@author: darshansatone
"""

import os,cv2

import numpy as np

import matplotlib.pyplot as plt
```

```
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split


from keras import backend as K
K.set_image_dim_ordering('th')


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 Convolution2D, MaxPooling2D
from keras.optimizers import SGD,RMSprop,adam


#path of dataset
PATH = os.getcwd()
# Define data path
data_path = PATH + '/data'
data_dir_list = os.listdir(data_path)

img_rows=128
img_cols=128
```

```
num_channel=1
num_epoch=20
# Define the number of classes
num_classes = 4

img_data_list=[]

for dataset in data_dir_list:
img_list=os.listdir(data_path+'/'+ dataset)
print ('Loaded the images of dataset-'+'{}\n'.format(dataset))
for img in img_list:
input_img=cv2.imread(data_path + '/'+ dataset + '/'+ img )
input_img=cv2.cvtColor(input_img, cv2.COLOR_BGR2GRAY)
input_img_resize=cv2.resize(input_img,(128,128))
img_data_list.append(input_img_resize)



img_data = np.array(img_data_list)
img_data = img_data.astype('float32')
img_data /= 255
print (img_data.shape)


if num_channel==1:
```

```
if K.image_dim_ordering()=='th':
img_data= np.expand_dims(img_data, axis=1)
print (img_data.shape)
else:
img_data= np.expand_dims(img_data, axis=4)
print (img_data.shape)
else:
if K.image_dim_ordering()=='th':
img_data=np.rollaxis(img_data,3,1)
print (img_data.shape)
# Define the number of classes

num_classes = 4

num_of_samples = img_data.shape[0]
labels = np.ones((num_of_samples,),dtype='int64')



labels[0:202]=0
labels[202:404]=1
labels[404:606]=2
labels[606:]=3
```

```
names = ['cats','dogs','horses','humans']


# convert class labels to on-hot encoding

Y = np_utils.to_categorical(labels, num_classes)


#Shuffle the dataset

x,y = shuffle(img_data,Y, random_state=2)

# Split the dataset

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=2)


# Defining the model

input_shape=img_data[0].shape


model = Sequential()


model.add(Convolution2D(32,
3,3,border_mode='same',input_shape=input_shape))

model.add(Activation('relu'))

model.add(Convolution2D(32, 3, 3))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.5))
```

```
model.add(Convolution2D(64, 3, 3))

model.add(Activation('relu'))

#model.add(Convolution2D(64, 3, 3))

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

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.5))


model.add(Flatten())

model.add(Dense(64))

model.add(Activation('relu'))

model.add(Dropout(0.5))

model.add(Dense(num_classes))

model.add(Activation('softmax')
```

Let us look at the first convolution layer Conv 2D. The number 32 shows the amount of output alter in the convolution. Numbers 3, 3 correspond to the kernel size, which determinate the width and height of the 2D convolution window. An important component of the first convolution layer is an input shape, which is the input array of pixels. Further convolution layers are constructed in the same way, but do not include the input shape.

The activation function of this model is Relu. This function setts the zero threshold and looks like: $f(x) = \max(0,x)$. If $x > 0$ — the volume of the array of pixels remains the same, and if $x < 0$ — it cuts oZ unnecessary details in the channel.

Max Pooling 2D layer is pooling operation for spatial data. Numbers 2, 2 denote the pool size, which halves the input in both spatial dimension.

After three groups of layers there are two fully connected layers. Flatten performs the input role. Next is Dense — densely connected layer with the value of the output space (64) and Relu activation function. It follows Dropout, which is preventing overfitting. Overfitting is the phenomenon when the constructed model recognises the examples from the training sample, but works relatively poorly on the examples of the test sample. Dropout takes value between 0 and 1. The last fully connected layer has 1 output and Sigmoid activation function.

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

#model.compile(loss='categorical_crossentropy',
optimizer=sgd,metrics=["accuracy"])

model.compile(loss='categorical_crossentropy',
optimizer='rmsprop',metrics=["accuracy"])
```

Next step is model compiling. It has a binary cross entropy loss function, which will show the sum of all individual losses. The optimiser algorithm is RMS drop, which is good for recurrent neural networks. The accuracy metrics shows the performance of the model.

#Viewing model_configuration

```
model.summary()

model.get_config()

model.layers[0].get_config()

model.layers[0].input_shape

model.layers[0].output_shape

model.layers[0].get_weights()
np.shape(model.layers[0].get_weights()[0])

model.layers[0].trainable
```

```
Out[21]: True

In [22]: model.summary()
    ...:
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 128, 128)	320
activation_1 (Activation)	(None, 32, 128, 128)	0
conv2d_2 (Conv2D)	(None, 32, 126, 126)	9248
activation_2 (Activation)	(None, 32, 126, 126)	0
max_pooling2d_1 (MaxPooling2	(None, 32, 63, 63)	0
dropout_1 (Dropout)	(None, 32, 63, 63)	0
conv2d_3 (Conv2D)	(None, 64, 61, 61)	18496
activation_3 (Activation)	(None, 64, 61, 61)	0
max_pooling2d_2 (MaxPooling2	(None, 64, 30, 30)	0
dropout_2 (Dropout)	(None, 64, 30, 30)	0
flatten_1 (Flatten)	(None, 57600)	0
dense_1 (Dense)	(None, 64)	3686464
activation_4 (Activation)	(None, 64)	0
dropout_3 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 4)	260
activation_5 (Activation)	(None, 4)	0

```
Total params: 3,714,788
Trainable params: 3,714,788
Non-trainable params: 0
```

Training

```
hist = model.fit(X_train, y_train,
batch_size=16,
nb_epoch=num_epoch,
verbose=1,
validation_data=(X_test, y_test))
```

#hist = model.fit(X_train, y_train, batch_size=32, nb_epoch=20,verbose=1, validation_split=0.2)

Batch size the number of training examples in one forward/backward pass (or for 1 epoch, which is expected).

Then the already described Image Data Generator is added for training and tasting samples. But it has a new transformation, which is called rescale. It multiplies the data by the given value.

The How_from_directory(directory) method is added for training and testing data. First, the path to the folders is speci>ed. Further, the target size follows. It shows width and height to which images will be resized. Next, the batch size is added. Finally binary class mode is set.

```
# Training with callbacks


from keras import callbacks


filename='model_train_new.csv'

csv_log=callbacks.CSVLogger(filename, separator=',', append=False)



early_stopping=callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=0, verbose=0, mode='min')


filepath="Best-weights-my_model-{epoch:03d}-{loss:.4f}-{acc:.4f}.hdf5"



checkpoint = callbacks.ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')


callbacks_list = [csv_log,early_stopping,checkpoint]


hist = model.fit(X_train, y_train, batch_size=16, nb_epoch=num_epoch, verbose=1, validation_data=(X_test, y_test),callbacks=callbacks_list)
```

Training is possible with the help of the @t_generator. Here it is important to indicate a number of epochs, which defines for how many times the training will repeat. 1 epoch is 1 forward pass and 1 backward pass over all the training examples.

Also, in this section steps_per_epoch and validation_steps are set. Steps_per_epoch (or number of iterations) shows total number of steps,

which is used to declare one epoch finished and begin the next. Typically this number is equal to the number of samples for training divided by the batch size (16). It means that the number of iterations: 1257/ 16 = 25. Validation_steps is total number of steps (batches of samples) to validate before stopping.

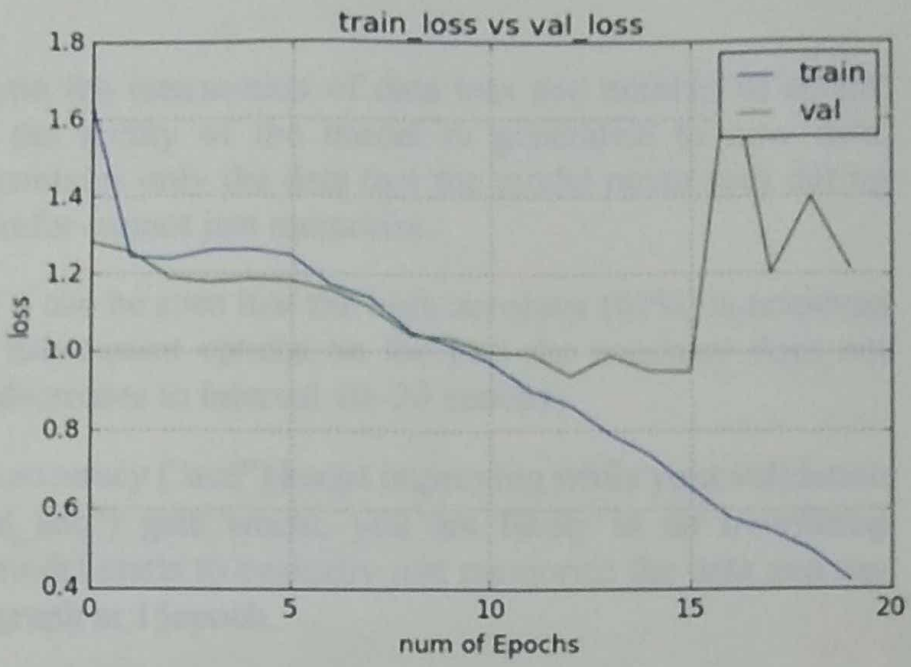
When the model is trained it should be saved with save_weights.

Best-weights-my_model-002-1.0665-0.6086.hdf5
Best-weights-my_model-001-1.0675-0.6086.hdf5
Best-weights-my_model-002-1.0885-0.6086.hdf5
Best-weights-my_model-001-1.0664-0.6086.hdf5
Best-weights-my_model-001-1.0791-0.6086.hdf5
Best-weights-my_model-001-1.0937-0.6086.hdf5

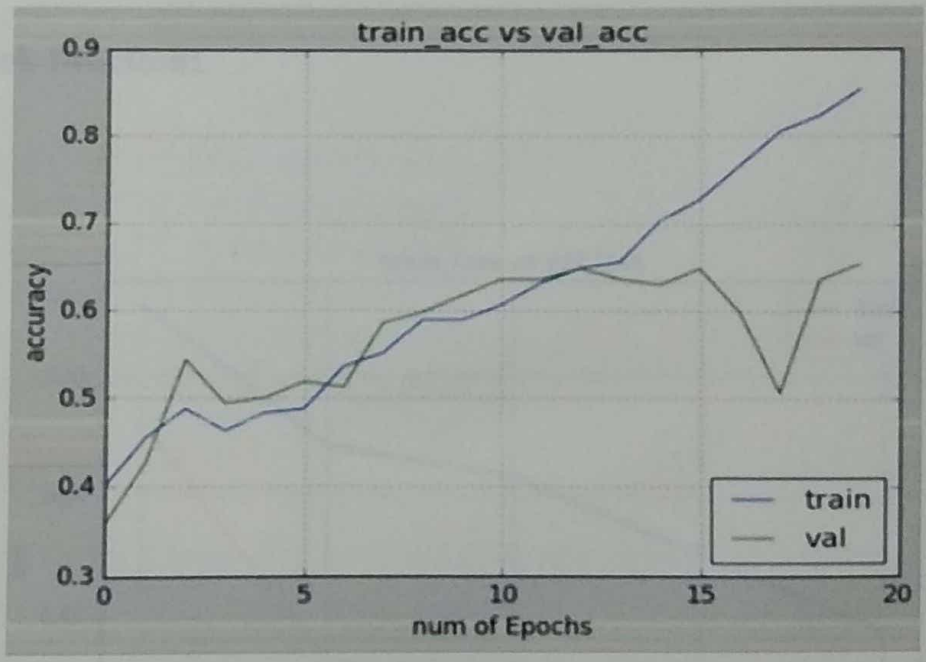
```
# visualising losses and accuracy

train_loss=hist.history['loss']

val_loss=hist.history['val_loss']

train_acc=hist.history['acc']

val_acc=hist.history['val_acc']

xc=range(3)


plt.figure(1,figsize=(7,5))

plt.plot(xc,train_loss)

plt.plot(xc,val_loss)

plt.xlabel('num of Epochs')

plt.ylabel('loss')

plt.title('train_loss vs val_loss')

plt.grid(True)
```

```
plt.legend(['train','val'])
#print plt.style.available # use bmh, classic,ggplot for big pictures
plt.style.use(['classic'])

plt.figure(2,figsize=(7,5))
plt.plot(xc,train_acc)
plt.plot(xc,val_acc)
plt.xlabel('num of Epochs')
plt.ylabel('accuracy')
plt.title('train_acc vs val_acc')
plt.grid(True)
plt.legend(['train','val'],loc=4)
#print plt.style.available # use bmh, classic,ggplot for big pictures
plt.style.use(['classic'])
```

Results



As a result of testing the model, I got good accuracy: 67% of correct classification samples after 20 epochs. The only drawback was that I had to wait about 50 minutes until 20 epochs come to the end (looking at the fact that I had a very small number of photos for training and at low system specifications).



Since we altered the parameters in our convolutional layer by reducing the size to its half there is always a chance of data loss.

Over 20 epochs the data loss was found optimum for both training and validating session.

For this, I decided to build two plots. The first shows the dependence of the data loss on the number of epochs. The data loss was calculated using

additional dataset of 1543 pictures. The second plot shows the dependence of accuracy and validation accuracy on the number of epochs during the testing.

The first graph shows the intersection of data loss and number of epoch. Data loss shows the ability of the model to generalise to new data. Validation dataset contains only the data that the model never sees during the training and therefor cannot just memorise.

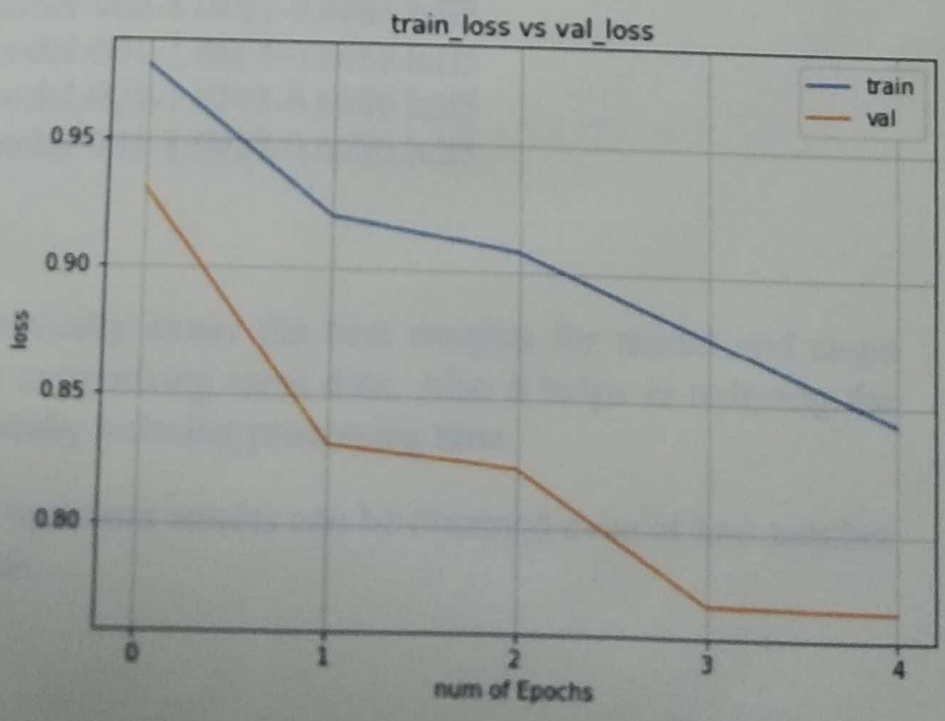
On the Second plot it can be seen that the high accuracy (67%) is achieved after 10 epoch. In subsequent epochs on the plot the accuracy does not improve (and even decreases in interval 10–20 epochs).

If your training data accuracy ("acc") keeps improving while your validation data accuracy ("val_acc") gets worse, you are likely in an overfitting situation, i.e. your model starts to basically just memorise the data and can be seen in our first graph at 15epoch.

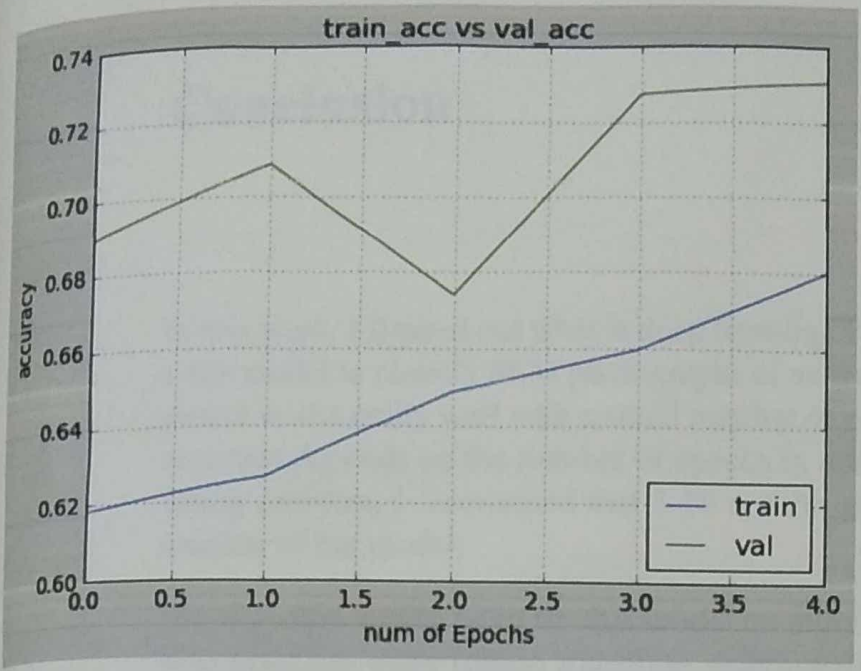
This means that after the 10th epoch the model can show the same result, but it will not be better. Consequently, this model is be sufficient to train on 10 epochs.

Using Keras callback function:

By using Keras callback function problem of overfitting and data loss can be overcome easily.



In this graph we can see the loss of data went on decreasing and model stopped at 5epochs.



The data loss is less while validating model than training session .

In this graph we can see accuracy is increased to 73% . The validation accuracy gave good results than training session.

Keras callback function stores best weights and optimise neural network in real time.

Best-weights-my_model-002-1.0665-0.6086.hdf5
Best-weights-my_model-001-1.0675-0.6086.hdf5
Best-weights-my_model-002-1.0885-0.6086.hdf5
Best-weights-my_model-001-1.0664-0.6086.hdf5
Best-weights-my_model-001-1.0791-0.6086.hdf5
Best-weights-my_model-001-1.0937-0.6086.hdf5

This function automatically stores the best weights for model and stops neural network from memorising same data. Also it helps in reducing the number of epochs thereby reducing processing time.

Using this technique optimum results can be obtained even at less number of epochs and data loss.

Conclusion

In this work, I figured out what is deep learning. I assembled and trained the CNN model to classify SEM photographs of materials. I have tested that this model works really well with a small number of photos. I measured how the accuracy depends on the number of epochs in order to detect potential over fitting problem. I determined that 5-10 epochs are enough for a successful training of the model.

My next step would be to try this model on more data sets and try to apply it to practical tasks (phase transition determination). I would also like to experiment with the neural network design in order to see how a higher efficiency can be achieved in various problems.

References

https://www.geeksforgeeks.org

https://towardsdatascience.com/the-7-steps-of-machine-learning-2877d7e5548e

http://www.datamind.cz/cz/vam-na-miru/umela-inteligence-a-strojove-uceni-ai-machine-learning

https://en.wikipedia.org/wiki/Artificial_neural_network

https://en.wikipedia.org/wiki/Deep_learning

https://en.wikipedia.org/wiki/Convolutional_neural_network

https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural- Networks/

https://www.lynda.com/Google-TensorFlow-tutorials/Building-Deep-Learning-Applications-Keras-2-0/601801-2.html

https://blog.keras.io/building-powerful-image-classi>cation-models-using-very-little-data.html

NFFA-EUROPE - SEM Dataset

by Aversa, Rossella; Modarres, Mohammad Hadi; Cozzini, Stefano; Ciancio, Regina;

Feb 19, 2018

Last updated at Apr 16, 2018

https://keras.io