UMU_54907 DeepLearning

2019VT

Lab1 v0.0

Mnist classification with CNN

sygr0003 ==Sylvie Grafeuille

sylgrafe@gmail.com

UMEÅ UNIVERSITET

Applied Physics and Electronics

Umeå universitet

Postadress: 901 87 Umeå

Content

Figures

1. Intro

1.1. About this document

This document is a laboration rapport . The aim is to prove several simple convolution neural network (CNN) to solve the fashion_Mnist problem.

I wrote this document in order to not mix up code with comments

The notebooks have almost no comment and and can be more easily edited when testing the models

This document contains the necessary information to understand the models

The document written in English except for the "uppgift" given by the teacher written in Swedish.

V 0.0 , 20190331

Intended audience.

The reader is expected to be a beginner in machine learning with some basic knowledge about python , jupyter , colab and machine learning .

Structure of the document

Chapter 1 give information about this document and the lab

Chapter 2 present the content on gitHUB

Chapter 3 presents the notebooks and the results of the test

Chapter 4 contains technical background and information related to the models

implemented in the notebooks

About the references

The explanation comes mostly from the book "deep_learning with python" Francois Cholet

Information given without references comes from docString in python

1.2. Uppgift Laboration 1 – CNN

Labbspec, Deep Learning , Kalle Prorok

Din uppgift är att koda ihop ett Convolutional Neural Net (CNN) som analyserar MNIST-data (du får välja mellan Fashion eller Handskrivna siffror) och ska använda en metod med bättre resultat än det alla-till-alla nätet som presenterades i början på Tensorflow-avsnittet (i Colab).

Se lärplattformen för mer information hur detta kan gå till och/eller läs lämpliga böcker, googla

etc, glöm inte att ange referenser i din dokumentation.

Genomförande

Du får använda Python Keras (helst) alternativt Matlab. Du ska ha skrivit all kod själv. Du kan använda och inspireras av exempel du hittar men skriv in allt själv, ändra variabelnamn dokumentera etc, så du lär dig och förstår vad som sker på djupet så du kan lösa andra problem på egen hand efter kursen.
Eftersom det kan vara tidsödande att provköra ditt program är det viktigt att du dokumenterar/presenterar del- och slutresultat tydligt samt även berättar lite kort vad varje kodrad gör samt analyserar/jämför slutresultatet och hur/varför det fungerar som det gör.

Dokumentation och Lagring av kod

För att klasskamraterna och lärare ska kunna nå, och vid behov provköra, koden behöver den lagras på något åtkomligt sätt, t ex på GitHub eller i Google Drive, så att du kan ange lämplig länk till hur och var man kommer åt den.

Redovisning – Peer-review

I uppgiften ingår att kommentera andras lösningar. Tanken med det är att båda parter ska lära sig lite extra, jag hoppas det kan vara kul och lärorikt att se hur andra löst det och på så vis förbättra sig själv.

Uppdatering baserat på feedback
När du erhållt feedback från din(a) kamrat(er)får du justera kod och dokumentation. Ev kan vi behöva köra en andra runda med återkoppling från en uppdaterad version. När genomgångarna är klara tittar läraren igenom kommentarer och resultat och ger båda parter betyg.

Betygskriterier

Utföraren
Grundnivå G: Ett väldokumenterat och förståeligt program som använder sig av CNN. Minst två olika inställningar på nån av parametrarna (t ex strides, fönsterstorlekar eller padding) ska köras och jämföras och slutresultatet kommenteras. Man ska nå minst 98% rätt på testdata i fallet med handskrivna siffor och 90% för Fashion-Mnist.

Mellannivå VG: Användande av metod (t ex Drop-out) för regularisering och även testa Data augmentation. Det är inte säkert att resultatet blir bättre men du ska visa att du känner till och kan

beskriva hur metoderna används och vad de kan göra för nytta.

Hög nivå MVG: Filtrens inlärda detektion ska visualiseras på lämpligt sätt. Man ska få en ide om hur nätverket analyserar indata allteftersom i de olika lagren. Efter det använda transfer learning, dvs basera nätet på ett i förväg tränat stort nät och justera till så att det fungerar på dessa nya data.

Om redovisaren använt/utfört delar av MVG-nivån och delar av VG ges VG.

En försenad inlämning kan bara ge G.

Bedömaren

Grundnivå G: kommenterat kort och noterat felaktigheter

Mellannivå VG: gett förbättringsförslag, personliga tips och konstruktiva idéer.

Update from the teacher : must use fashion_mnist

Jag fick en fråga om MNIST; det finns minst (mnist ;) ?) 2 varianter; först en med handskrivna siffror som är vanlig. Den är kul men har nu blivit så enkel att lösa med 98% så vi ska kika på den lite svårare med modekläder, handväskor, skor etc och en första lösning som ni ska jobba vidare på så den blir bättre finns t ex här:

https://www.tensorflow.org/tutorials/keras/basic_classification

Skriv egen kod så ni vet/lär vad ni håller på med.

About the DATA for lab1

https://www.tensorflow.org/tutorials/keras/basic_classification

Fashion MNIST is a slightly more challenging problem than regular MNIST. Both datasets are relatively small and are used to verify that an algorithm works as expected. They're good starting points to test and debug code.

We will use 60,000 images to train the network and 10,000 images to evaluate how accurately the network learned to classify images. You can access the Fashion MNIST directly from TensorFlow, just import and load the data:

1.3. My environment for programming and test

The environment

a desktop (no GPU) running Ubuntu 18.04

the code is written in python . This lab involves python pgm run on localhost and colab , jupyter notebooks run on localhost and colab

Keras and tensor flow as backend where used. The versions used in local host seems to be the same as the one in colab.

```
syl1>  python3  --version
Python 3.6.7
syl1>  jupyter --version
4.4.0
syl1>  python3 localW0.py
Using TensorFlow backend.
keras.__version__:  2.2.4
```

and in colab:

```
! python –version
! jupyter --version
import keras
print (keras.__version__)
Python 3.6.7
4.4.0
Using TensorFlow backend.
2.2.4
```

2. About the files on gitHub

As Colab is in a virtual environment without easy procedure to read and write files I decided to put all information about this lab on Github

2.1. The Repositories

All files related to this lab are put from 2 public repositories on github

https://github.com/SylGrafe/lab1Repo

https://github.com/SylGrafe/lab1Lib

https://github.com/SylGrafe/lab1Lib

contains lab1Utils.py a python file imported by all notebooks

(a copy of the file was also used for the test on local host)

lab1Utils.py contains:

routines to dump information about the models and the test as json string

routines to access any file containing this information and display the content.

https://github.com/SylGrafe/lab1Repo

In this repo you find :

colab_ReadJSON a notebook presented later on.

colab_xx notebooks implementing differents models

this file : lab1sygr0003.pdf

README

2.2. about the notebooks

ColabReadJson is a notebook used to access the file containing test , results dumped a json string , the other notebooks contains implementations of CNN models.

colabReadJSON

The notebook ColabReadJson access theDumpFileName , the file containing dumped results.

Via a simple interface the user may choose and print :

a summary of all results

a more details presentation for any result

to plot the fit history for any result

by default theDumpFileName is lab1Repo/results.json

Notice that by changing the parameter theDumpfilename in the notebook you may access another file created in the virtual environment

the commands for user interface

In this extract from output for readJson pgm (run on local host) you get the list of user commands

to get this list of command press retrun

```
syl1>  python3  localReadJson.py
 using dump file ../repo/results.json
indStr [moreStr] >
     explore the content of the dumpfile xxx
     if indStr== s     summary : print list of all records  in the file
```

```
if indStr== o     ordered summary , best results first
if indStr ==e     exit the Pgm
if indStr== valid record indice       print part of the record
if indStr==validindice and moreStr==a   print the record
if indStr==validindice and moreStr==p   print part of the record and plot the history
```

example list of records

In this example (also run on local host)the user asked to get an ordered list of results .

Because the user used the command ‘o’ the list is ordered, the results with highest values for test_acc are presented first.

```
syl1>  python3 localReadJson.py
 using dump file ../repo/results.json

indStr [moreStr] >  o
 indice <codeRef> timeStample, modelStruct, info
test[loss,acc] <--> max (val_acc) at i/nb epochs:
  0 <colab_CNN02> 2503_1352,  wind55, testing window sizes
 [0.0261, 0.9928] <--> 0.991 at 4/5:
  1 <colab_CNN01> 2403_1656,  s1_same, testing strides and padding
 [0.0236, 0.9925] <--> 0.992 at 4/6:
  2 <colab_CNN01> 2303_1652,  s1_valid, testing srides and padding
 [0.0294, 0.9917] <--> 0.9928 at 4/6:
  3 <colab_CNN02> 2503_1350,  wind33, testing window sizes
 [0.0325, 0.9904] <--> 0.9907 at 2/5:
...
```

example history plot from colabReadJSON

In this example (once more run on local host) , the user ask to see the history plot

for the first result of the list

```
syl1>  python3 localReadJson.py
indStr [moreStr] >  0 p
colab_CNN02, wind55, rmsprop, categorical_crossentropy, 2503_1352
test Results [0.0261, 0.9928]
```

```
best val accuracy: at epochs  4 /5  value 0.991


history Params: {'batch_size': 128, 'epochs': 5, 'steps': None, 'samples': 54000, 'verbose': 0,
'do_validation': True, 'metrics': ['loss', 'acc', 'val_loss', 'val_acc']}
```

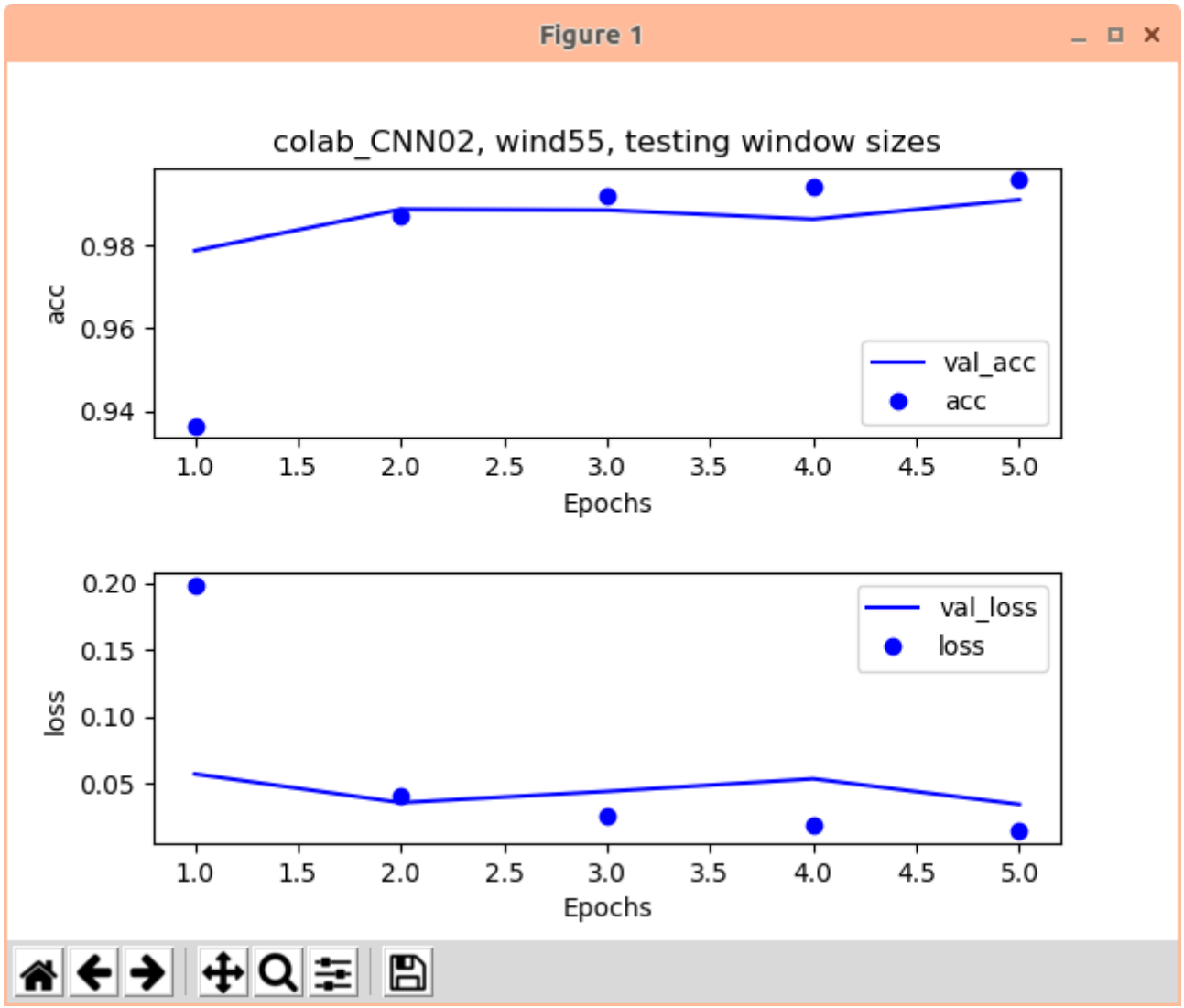

Illustration 1: plot history for colabCNN02 wind55

The Notebooks with NN models

The other notebooks contains several implementations of neural networks ,

colab_D01 contains only densely connected layers

the other notebooks implements CNN models

After each test the results and some more information are dumped in a json file

The user should put reasonable values in the **'modelStruct'** and' **infoStr**' in order to be able to identify the results later among all dumped results.

Notice that all the results issued from one notebook are put into the a single json file

2.3. about results.json

results.json is a file containing of the results produces by the several model implementations

To create results.json I did:

run pgm with model implementation (this produces a json file related to the notebook).

If the pgm was run in colab download the json file on localhost

When all test were done some interesting json files were then concatenated into a single file: results.json who was hen push back into

https://github.com/SylGrafe/lab1Repo

example of information dumped for one test

Here is given the kind of information dumped for one test.

{"modelStruct": "s1_valid", "compInfo": "rmsprop, categorical_crossentropy", "histDict": {"val_loss": [0.0712, 0.054, 0.0492, 0.0373, 0.0257, 0.0259], "val_acc": [0.9795, 0.9842, 0.9843, 0.9882, 0.9928, 0.9918], "loss": [0.2617, 0.0634, 0.0443, 0.0325, 0.0267, 0.0206], "acc": [0.9188, 0.9805, 0.9864, 0.9897, 0.9914, 0.9935]}, "histParams": {"batch_size": 128, "epochs": 6, "steps": null, "samples": 54000, "verbose": 0, "do_validation": true, "metrics": ["loss", "acc", "val_loss", "val_acc"]}, "timeStamp": "2303_1652", "info": "testing srides and padding", "h5": "", "testRes": [0.0294, 0.9917], "codeRef": "colab_CNN01"}

explanations

codeRef : the reference to the note book from which the results where produced

"codeRef": "colab_CNN01"

modelStruct a reference to a if block in the code related to the structure of the model

{"modelStruct": "s1_valid"

timestamp : a time stamp at which the pgm was run

"timeStamp": "2303_1652"

info : some more nformation that the user wanted to dump

"info": "testing srides and padding"

the other parts are now listed without more explanations

"compInfo": "rmsprop, categorical_crossentropy", "

histDict": {"val_loss": [0.0712, 0.054, 0.0492, 0.0373, 0.0257, 0.0259], "val_acc": [0.9795, 0.9842, 0.9843, 0.9882, 0.9928, 0.9918], "loss": [0.2617, 0.0634, 0.0443, 0.0325, 0.0267, 0.0206], "acc": [0.9188, 0.9805, 0.9864, 0.9897, 0.9914, 0.9935]},

"histParams": {"batch_size": 128, "epochs": 6, "steps": null, "samples": 54000, "verbose": 0, "do_validation": true, "metrics": ["loss", "acc", "val_loss", "val_acc"]},

3. Data , Notebooks and Results

Before presenting the model and results lets have a look at the data

3.1. about images in fashion_mnist

The notebook colab_D01 contains code to visualize part of the fashion_Mnist dataset .

The labels are given by

```
# for visualisation purpose save the train images and  train labels before reshaping
  class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',        'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

the code for plotting the images is

```
if (doPrintImages):
# visualize the training data .
  offset=500
  plt.figure(figsize=(10,10))
  for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images_orig[offset+i], cmap=plt.cm.binary)
```

```
  plt.xlabel(class_names[train_labels_orig[offset+i]])
plt.show()
```

((this text is a format flag for the document))

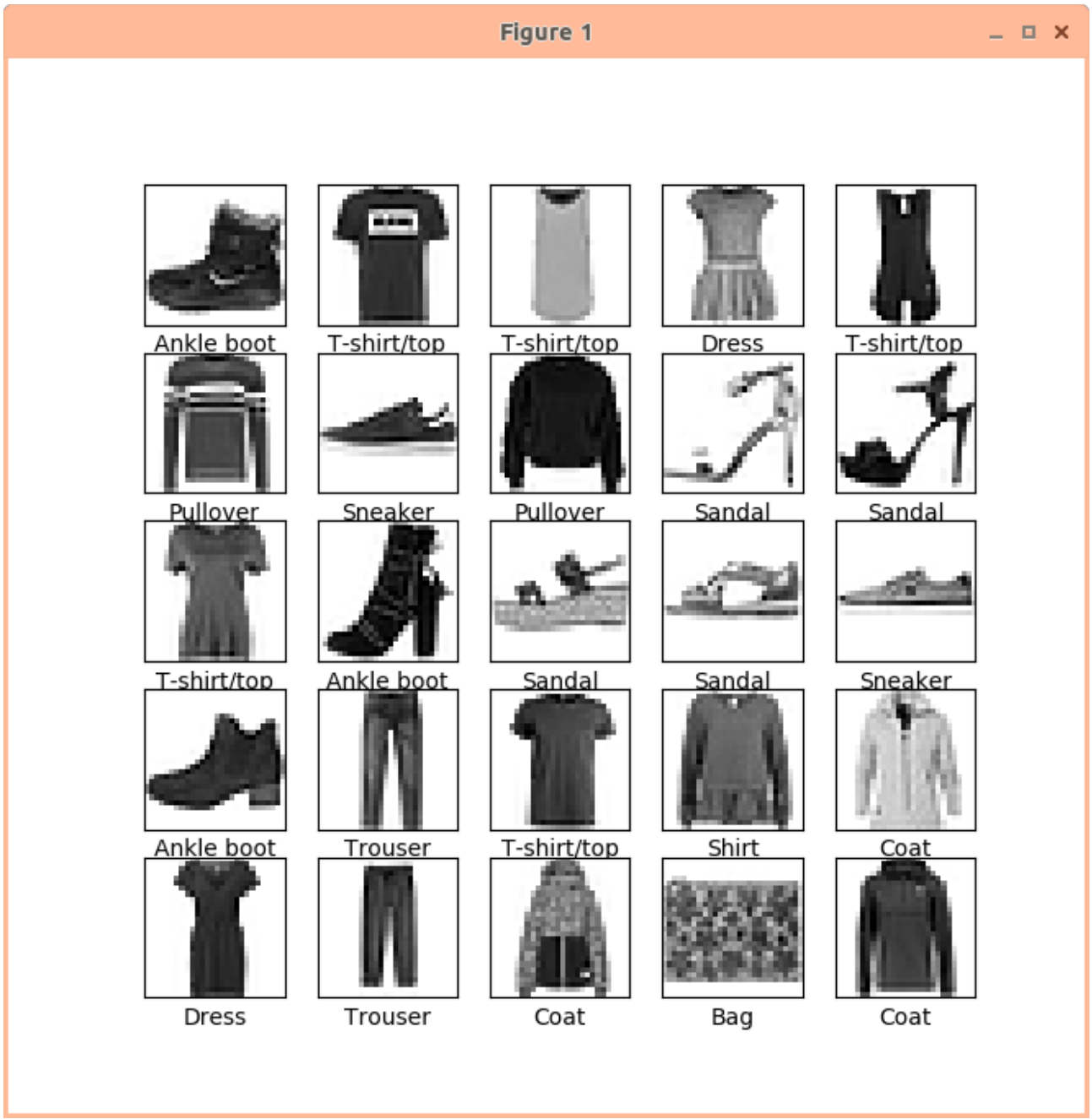

Illustration 2: Some of the fashion pictures to classify

characteristics of the images

Poor resolution

I did react immediately to the fact that the images to classify have really a poor resolution . I wonder if that did not make the classification more difficult .

other characteristics

The images seem to fill up the frame as much as possible be centered and as vertical as possible,

open questions

I wonder how a model trained on pictures that fill up the frame , that are straights would give some test_res on pictures that are rotated or that do not fill the frame. But I did not try to quest the answer of this question

How does the gray coloring influence the performances of the model (how would it bw with only black or white pixels)?

3.2. The Notebooks

The aim of this lab it to beat the performances a simple model . This simple model is presented first

The notebooks created for this lab are presented afterwards.

To beat Test accuracy: 0.8778

https://www.tensorflow.org/tutorials/keras/basic_classification

the structure of the model to beat is given here:

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5)
```

this gives the results Test accuracy: 0.8778

notebook colab_CNN0

colab_CNN00 is a notebook in which the influence of number of layer was tested

relevant code extract from the notebook :

```
if (modelStruct == "3Conv" or modelStruct=="2Conv" or modelStruct=="1Conv"):
  theModel.add(layers.Conv2D(32, (3, 3), activation='relu',
                         input_shape=(28, 28, 1)))
elif (modelStruct == "3Conv" or modelStruct=="2Conv"):
  theModel.add(layers.MaxPooling2D((2, 2)))
  theModel.add(layers.Conv2D(64, (3, 3), activation='relu'))
elif (modelStruct == "3Conv"):
  theModel.add(layers.MaxPooling2D((2, 2)))
  theModel.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

notebook colab_CNN01

In this notebook the influence of strides and padding and window size where tested by changing the values of the parameters myStrides and myPadding and myWind

Relevant code extract from the notebook:

```
elif (modelStruct == "s1_same" ):
  myStrides=1
  myPadding='same'
  theModel.add(layers.Conv2D(32, myWind,     strides=myStrides , padding=myPadding ,
                         activation='relu', input_shape=(28, 28, 1)))
  theModel.add(layers.MaxPooling2D((2, 2)))
  theModel.add(layers.Conv2D(64, myWind, activation='relu',      strides=myStrides , padding=myPadding))
  theModel.add(layers.MaxPooling2D((2, 2)))
  theModel.add(layers.Conv2D(64, myWind, activation='relu'))
elif (modelStruct == "s2_same" ):
  myStrides=2
  myPadding='same'
  # when  strides=2 do not use  MaxPooling layers
  theModel.add(layers.Conv2D(32, myWind,     strides=myStrides , padding=myPadding ,
                         activation='relu', input_shape=(28, 28, 1)))
  theModel.add(layers.Conv2D(64, myWind, activation='relu',      strides=myStrides , padding=myPadding))
  theModel.add(layers.Conv2D(64, myWind, activation='relu'))
```

notebook colab_CNN02

In this notebook the influence of drop out and data augmentation was tested

dropout

The relevant code to test the dropout is

```
theModel.add(layers.Conv2D(32, myWind,   padding=myPadding,    activation='relu', input_shape=(28, 28, 1)))
theModel.add(layers.MaxPooling2D((2, 2)))
theModel.add(layers.Conv2D(64, myWind, padding=myPadding ,  activation='relu'))
theModel.add(layers.MaxPooling2D((2, 2)))
theModel.add(layers.Conv2D(64, myWind, padding=myPadding ,      activation='relu'))
if (doDropout):
  theModel.add(Dropout(0.5))
theModel.add(layers.Flatten())
theModel.add(layers.Dense(64, activation='relu'))
theModel.add(layers.Dense(10, activation='softmax'))
```

data augmentation

The relevant code to test data augmentation is:

```
if (doDataAug):
  myZoom=0.03
  myRot=7
  modelStruct +=" ,z=%.2f ,rot=%d " % (myZoom , myRot)
  train_datagen = ImageDataGenerator(
      featurewise_center=False,
      featurewise_std_normalization=False,
      rotation_range=myRot,
      zoom_range = myZoom,
      width_shift_range = 0,
      height_shift_range=0,
      shear_range=0,
      horizontal_flip=False,
      fill_mode='nearest')
```

colab_D01

In this notebook densely connected only models were implemented. The nb of layers and nb of neurons per layers are the parameters. These models were implemented as a reference to be compared with the models implementation s using CNN

The notebook contains also code to visualize the data and code for plotting confusion matrix or the images that were falsely labeled .

model structure

The relevant code is:

```
if (modelStruct == "oneLayer"):
  L1NeuronsNb = 512
  infoStr="1 dense layer %d" % (L1NeuronsNb)
  theModel.add(layers.Dense(L1NeuronsNb,activation='relu' , input_shape=(28*28,)))
elif (modelStruct == "twoLayers"):
  L2NeuronsNb = 512
  L1NeuronsNb = 512
  infoStr="2 dense layer %d , %d"   % (L1NeuronsNb , L2NeuronsNb)
  theModel.add(layers.Dense(L1NeuronsNb,activation='relu' , input_shape=(28*28,)))
  theModel.add(layers.Dense(L1NeuronsNb,activation='relu' , input_shape=(28*28,)))
```

Plot wrong prediction

On this notebook there is also code also for plotting the images that were falsely labeled :

```
# get predictions print confusion matrics
pred = theModel.predict( test_images)
from sklearn.metrics import confusion_matrix
import numpy as np
predicted_labels=np.argmax(pred,axis=1)
true_labels = np.argmax(test_labels, axis=1)


cm2=confusion_matrix(true_labels , predicted_labels)
print (cm2)


if doPrintImages :
  #  visualize some wrong predictions
```

```
ind0=250
for ind in range( ind0 , len(test_labels)):
  l_orig = test_labels_orig[ind]
  l_pred=  predicted_labels[ind]
  if (l_orig != l_pred) :
    print ( "ind=%d %s -->%s "  % (
    ind , class_names [l_orig] ,class_names [l_pred] ))
    plt.imshow(test_images_orig[ind])
    break
```

example : plot one wrong prediction

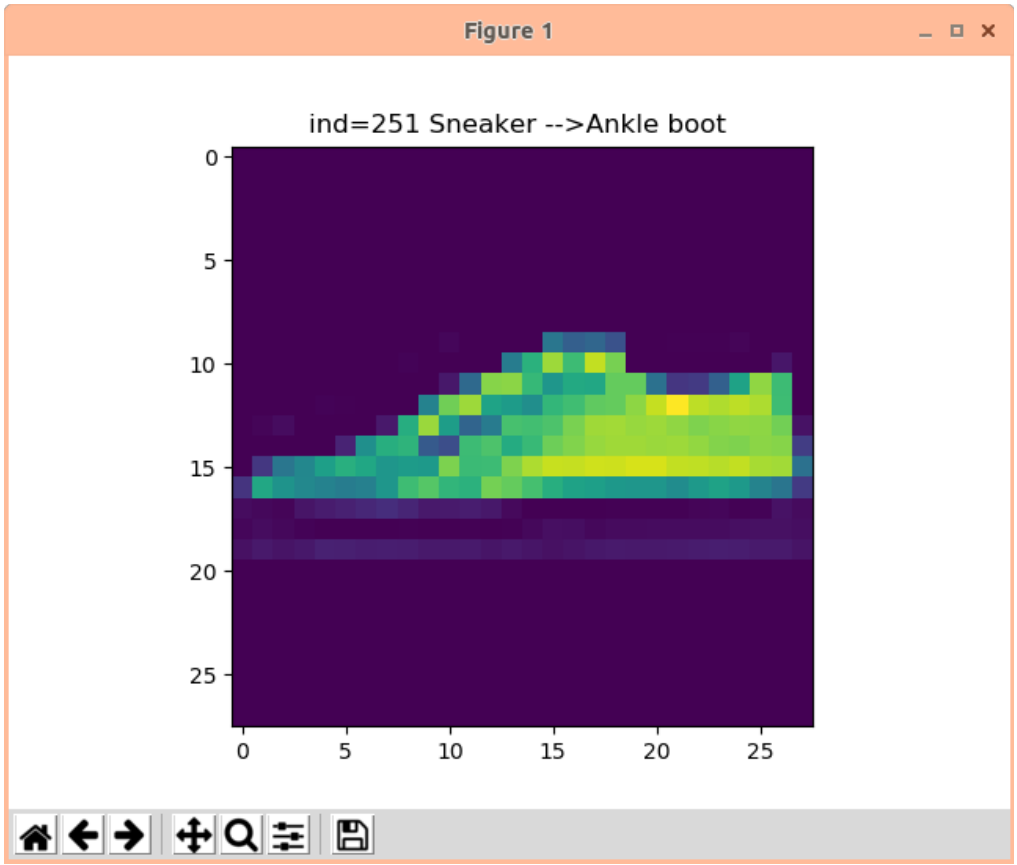

Illustration 3: D01 plot one wrong prediction

3.3. Results

Overfitting

The general rule is the the models over-fit rapidly .

On example with data augmentation is presented to illustrate over fitting problem

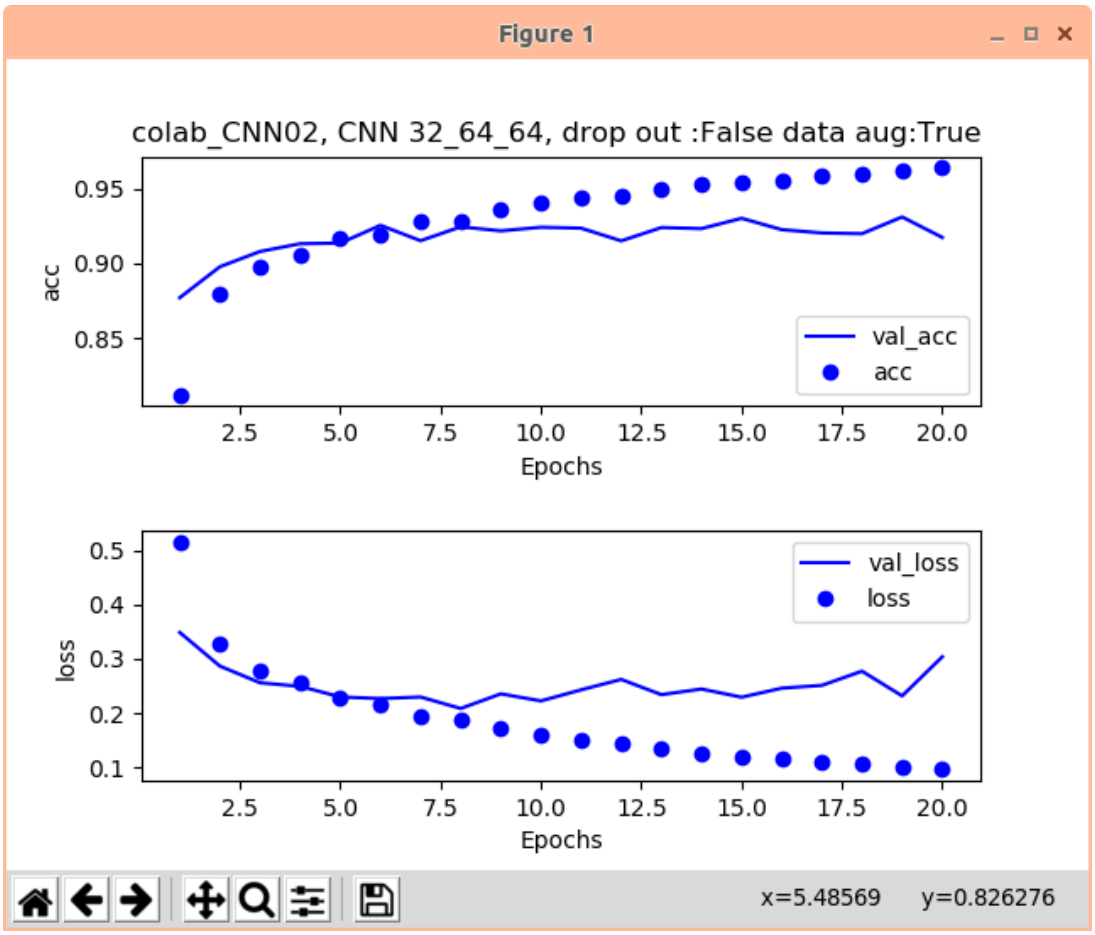

Illustration 4: CNN02 overfit 20190328

The results of the test are presented now , referred by the name of the notebook that produced them

colab_CNN00

The results of the notebook colab_CNN00 ,the test differs by

changing : the nb of layers , the bash size , the optimizer , the size of the CNN filters

```
syl1> python3 localReadJson.py ../repo/results_CN00.json
indStr [moreStr] > so

 indice <codeRef> timeStample, modelStruct, info
test[loss,acc] <--> max (val_acc) at i/nb epochs:

 0 <colab_CNN00> 2703_1340,  2Conv, CNN filters: 32,  64
[0.2491, 0.9185] <--> 0.924 at 6/8:
 1 <colab_CNN00> 2703_1302,  2Conv, CNN filters: 32,  64
[0.2722, 0.9138] <--> 0.9267 at 6/8:
 2 <colab_CNN00> 2703_1301,  2Conv, CNN filters: 32,  64
[0.2511, 0.9118] <--> 0.9218 at 5/8:
 3 <colab_CNN00> 2703_1239,  3Conv, CNN filters: 32, 64, 64
[0.2677, 0.91] <--> 0.9145 at 9/10:
 4 <colab_CNN00> 2703_1304,  3Conv, CNN filters: 32, 64, 64
[0.2841, 0.9067] <--> 0.9103 at 7/8:
 5 <colab_CNN00> 2703_1241,  3Conv, CNN filters: 32, 64, 64
[0.321, 0.9057] <--> 0.915 at 13/15:
 6 <colab_CNN00> 2703_1243,  3Conv, CNN filters: 64, 128, 128
[0.3852, 0.9052] <--> 0.9188 at 11/15:
 7 <colab_CNN00> 2703_1254,  3Conv, CNN filters: 64, 128, 128
[0.2817, 0.9041] <--> 0.9097 at 7/8:
 8 <colab_CNN00> 2703_1300,  1Conv, CNN filter: 32
[0.3175, 0.9039] <--> 0.9147 at 6/8:
 9 <colab_CNN00> 2703_1556,  3Conv, CNN filters: 32, 64, 64
[0.2741, 0.902] <--> 0.9075 at 6/8:
```

Result for 2 layers

The best results came from the solution with 2 layers , (filter 32 , 64) . adam optimizer and batch size 64 as seen in this output from readJson.py

```
indStr [moreStr] >  0 p
colab_CNN00, 2Conv, adam, categorical_crossentropy, 2703_1340
 strides ,padding for wind: wind55
```

test Results [0.2491, 0.9185]

best val accuracy: at epochs 6 /8 value 0.924

history Params: {'batch_size': 64, 'epochs': 8, 'steps': None, 'samples': 54000, 'verbose': 0, 'do_validation': True, 'metrics': ['loss', 'acc', 'val_loss', 'val_acc']}

plot history for best results with 2 layers

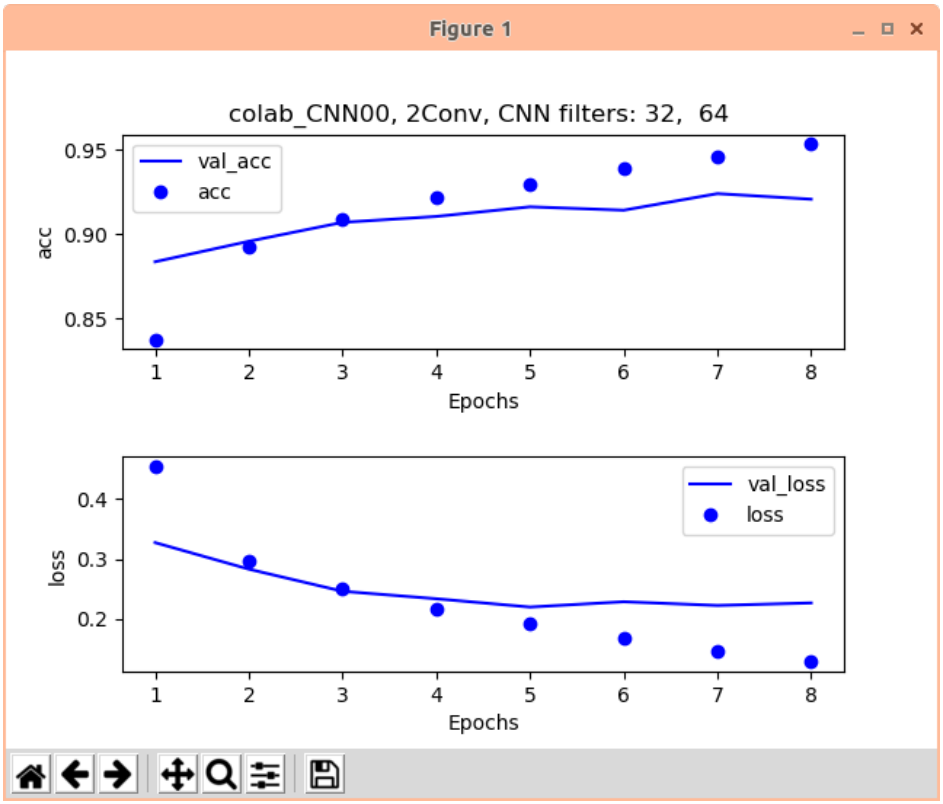

Illustration 5: CNN00 best 2 layers

20190327

the curves are not so beautiful an therefore I present the results for 3 layers

results 3 layers

plot history for best results with 3 layers.

colab_CNN00, 3Conv, rmsprop, categorical_crossentropy, 2703_1239

CNN filters: 32, 64, 64

test Results [0.2677, 0.91]

best val accuracy: at epochs 9 /10 value 0.9145

history Params: {'batch_size': 128, 'epochs': 10, 'steps': None, 'samples': 54000, 'verbose': 0, 'do_validation': True, 'metrics': ['loss', 'acc', 'val_loss', 'val_acc']}

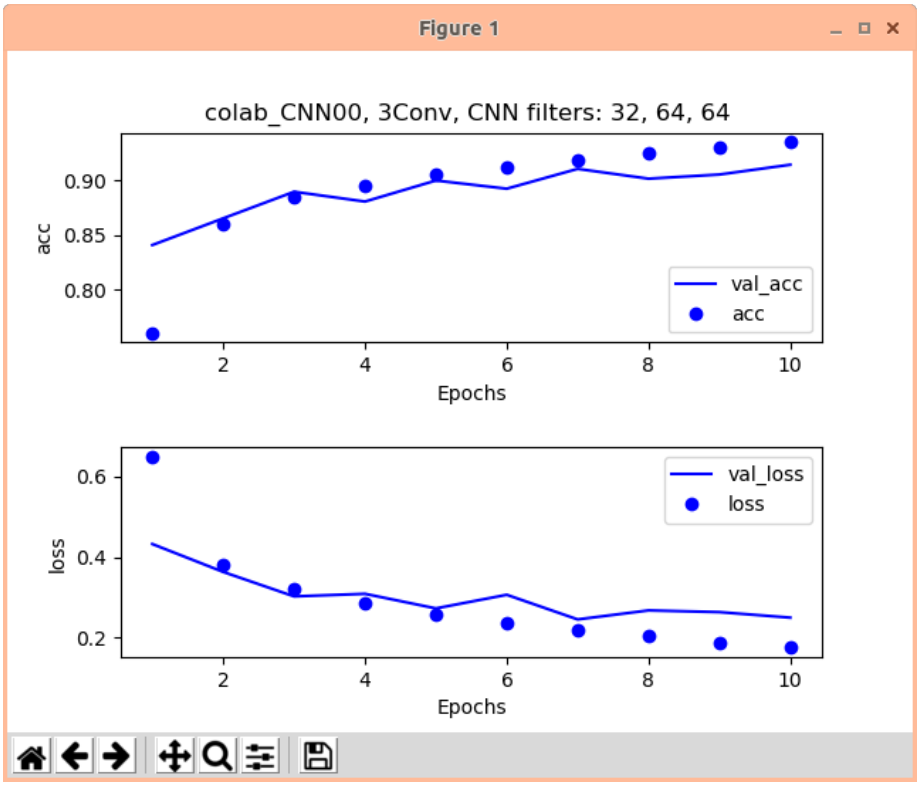

Illustration 6: CN00 best 3 layers 20190327

The curves are more beautiful

worst results

The worst results comes from solution with only 1 layer or with 3 layers

indice <codeRef> timeStample, modelStruct, info

test[loss,acc] <--> max (val_acc) at i/nb epochs:

8 <colab_CNN00> 2703_1300, 1Conv, CNN filter: 32

[0.3175, 0.9039] <--> 0.9147 at 6/8:

9 <colab_CNN00> 2703_1556, 3Conv, CNN filters: 32, 64, 64

[0.2741, 0.902] <--> 0.9075 at 6/8:

conclusion , best with 3 layers

Even if the best results were for a model with 2 layers I could think that the val_loss curve follow

better the loss curve in a model with 3 CNN layers . Therefore in the next notebooks three layers were used.

The difference for test accuracy between the best and worst results is about 1% an that does not seem to be such a big difference

colab_CNN01

Testing the influence of strides padding and the window size using a model with 3 CNN layers

list of results

The output from some test is presented here

```
syl1>  python3 localReadJson.py  colab_CNN01_2803.json
indStr [moreStr] > so
 indice <codeRef> timeStample, modelStruct, info
test[loss,acc] <--> max (val_acc) at i/nb epochs:
 0 <colab_CNN01> 2803_1204,  s1_same,  strides ,padding for wind: wind55
[0.2503, 0.91] <--> 0.9105 at 4/6:
 1 <colab_CNN01> 2803_1135,  s2_same,  strides ,padding for wind: wind33
[0.2578, 0.9088] <--> 0.9093 at 5/6:
 2 <colab_CNN01> 2803_1133,  s1_same,  strides ,padding for wind: wind33
[0.2438, 0.9075] <--> 0.9187 at 5/6:
 3 <colab_CNN01> 2803_1134,  s1_valid,  strides ,padding for wind: wind33
[0.2699, 0.9011] <--> 0.9042 at 5/6:
 4 <colab_CNN01> 2803_1206,  s2_same,  strides ,padding for wind: wind55
[0.2878, 0.8974] <--> 0.9078 at 4/6:
 5 <colab_CNN01> 2803_1202,  s2_valid,  strides ,padding for wind: wind33
[0.2829, 0.8973] <--> 0.9083 at 5/6:
```

best result

best result was obtained with a window size of 55 stride (1,1) and same padding

```
colab_CNN01, s1_same, adam, categorical_crossentropy, 2803_1204
 strides ,padding for wind: wind55
test Results [0.2503, 0.91]
best val accuracy: at epochs  4 /6  value 0.9105
history Params: {'batch_size': 64, 'epochs': 6, 'steps': None, 'samples': 54000, 'verbose': 0,
'do_validation': True, 'metrics': ['loss', 'acc', 'val_loss', 'val_acc']}
```

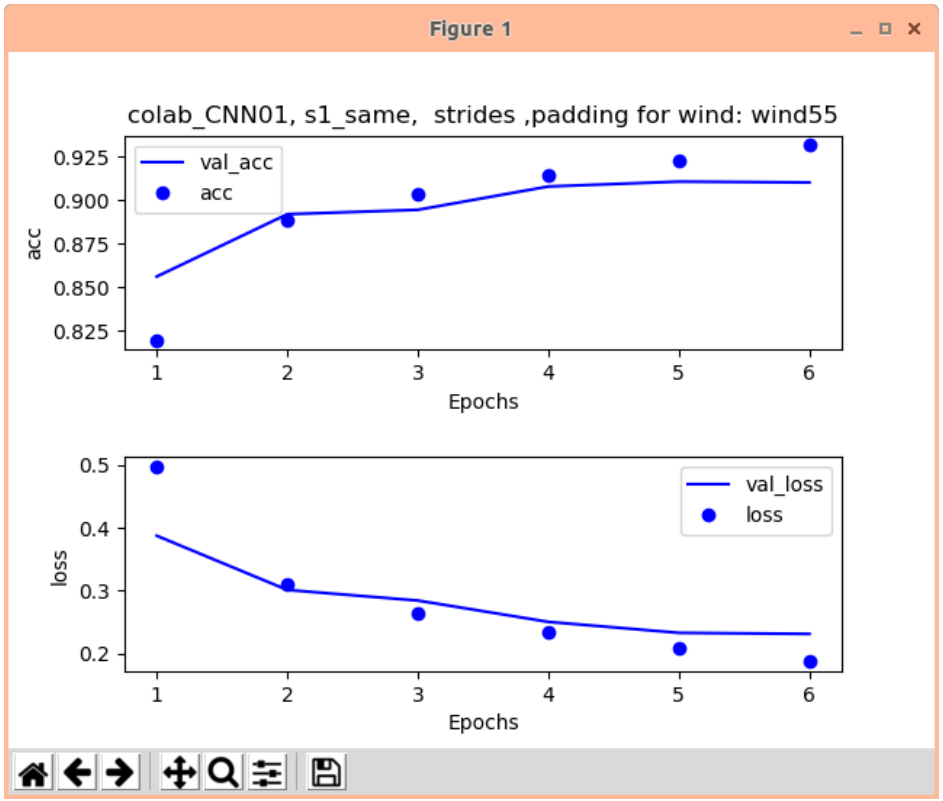

Illustration 7: CNN01 , best results 20190328

conclusion :best with padding='same'

Seems that padding =='same' and strides = (1,1) and wind(5,5) gives the best results

There is not much differences in the results between wind = (3,3) and wind=(5,5)

In the next notebooks the test will be made with wind=(3,3) and wind=(5,5)

colab_CNN02

Testing the influence of drop out by inserting a drop out just before the first dense layers and also testing data augmentation

To try to leaan more about the parameters for data augmentation I did look at :

Image Augmentation for Deep Learning With Keras

https://machinelearningmastery.com/image-augmentation-deep-learning-keras/

list of results at 20190328

In this test the data augmentation was defined by

```
if (doDataAug):
  train_datagen = ImageDataGenerator(
      featurewise_center=False,      featurewise_std_normalization=False,      rotation_range=7,
      zoom_range = 0.03,      width_shift_range = 0,      height_shift_range=0,
      shear_range=0,      horizontal_flip=False,      fill_mode='nearest')
```

The results for theses test is presented here

```
using dump file colab_CNN02_2803.json
indStr [moreStr] >  o
 indice <codeRef> timeStample, modelStruct, info
test[loss,acc] <--> max (val_acc) at i/nb epochs:
 0 <colab_CNN02> 2803_1531,  CNN 32_64_64, drop out :False data aug:True wind:(5, 5)
[0.2602, 0.9192] <--> 0.9268 at 5/8:
 1 <colab_CNN02> 2803_1518,  CNN 32_64_64, drop out :False data aug:False wind:(3, 3)
[0.2538, 0.9175] <--> 0.923 at 7/8:
 2 <colab_CNN02> 2803_1513,  CNN 32_64_64, drop out :False data aug:True wind:(3, 3)
[0.2413, 0.916] <--> 0.9231 at 7/8:
 3 <colab_CNN02> 2803_1500,  CNN 32_64_64, drop out :False data aug:True
[0.3243, 0.9152] <--> 0.9309 at 18/20:
 4 <colab_CNN02> 2803_1526,  CNN 32_64_64, drop out :True data aug:True wind:(5, 5)
[0.2591, 0.9072] <--> 0.9152 at 5/8:
```

details for best result

a more detailed view of the best result is :

colab_CNN02, CNN 32_64_64, adam, categorical_crossentropy, 2803_1531

drop out :False data aug:True wind:(5, 5)

test Results [0.2602, 0.9192]

best val accuracy: at epochs 5 /8 value 0.9268

history Params: {'epochs': 8, 'steps': 937.5, 'verbose': 0, 'do_validation': True, 'metrics': ['loss', 'acc', 'val_loss', 'val_acc']}

and here is the history plot

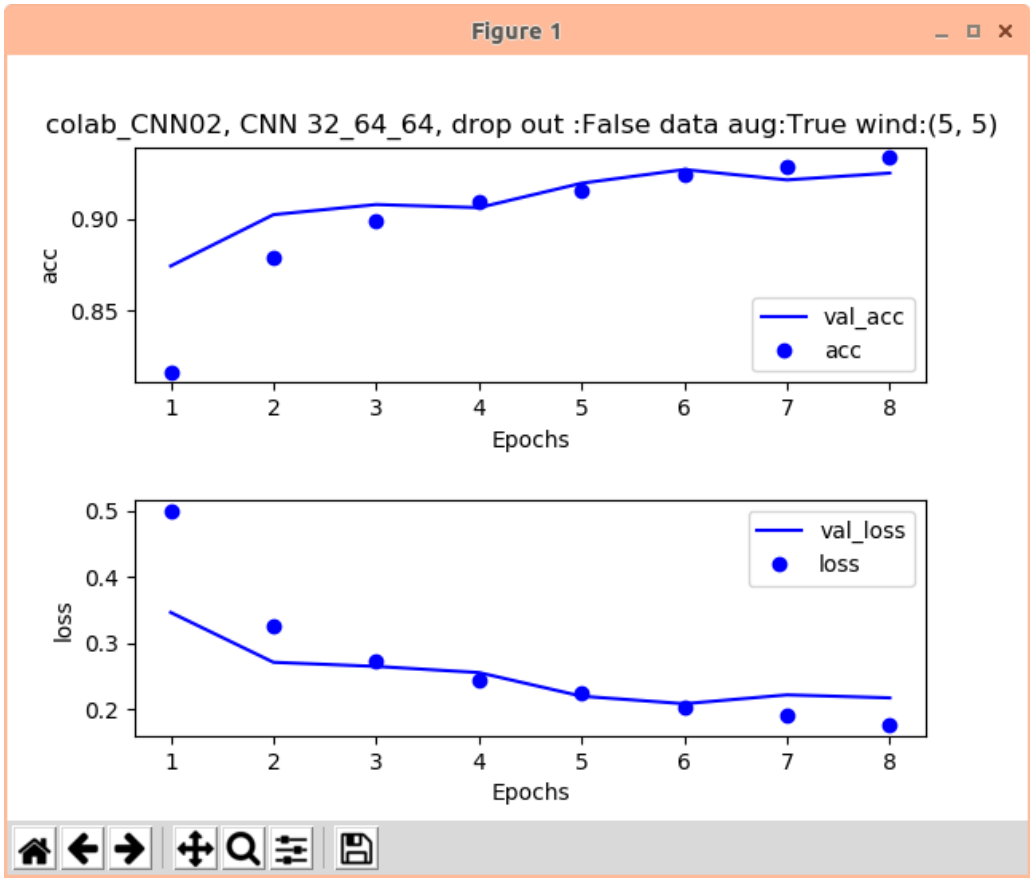

Illustration 8: CNN02 data augmentation 20190328

results at 20190330

In this test , the zoom and rotation parameters for data augmentation were tested.

Then I also tried to use dropout , drop out use with an increased the nb of epochs gave the best result.

the relevant for code for testing the parameters is:

```
# try several zoom and rot values
if (doDataAug):
  myZoom=0.03
```

```
myRot=15
modelStruct +=" ,z=%.2f ,rot=%d " % (myZoom , myRot)
train_datagen = ImageDataGenerator(      featurewise_center=False,      featurewise_std_normalization=False,
    rotation_range=myRot,      zoom_range = myZoom,      width_shift_range = 0,      height_shift_range=0,
    shear_range=0,      horizontal_flip=False,      fill_mode='nearest')
```

list of results

```
syl1>  python3 localReadJson.py  colab_CNN02_3003.json
 using dump file colab_CNN02_3003.json
indStr [moreStr] >  o
...o...
 indice <codeRef> timeStample, modelStruct, info
test[loss,acc] <--> max (val_acc) at i/nb epochs:
   0 <colab_CNN02> 3003_1230,  32_64_64 ,z=0.01 ,rot=7 , drop out :True data aug:True wind:(5, 5)
[0.2259, 0.9229] <--> 0.9322 at 9/12:
   1 <colab_CNN02> 3003_1127,  32_64_64 ,z=0.01 ,rot=7 , drop out :False data aug:True wind:(5, 5)
[0.2507, 0.9185] <--> 0.9256 at 5/8:
   2 <colab_CNN02> 3003_1133,  32_64_64 ,z=0.01 ,rot=0 , drop out :False data aug:True wind:(5, 5)
[0.28, 0.9167] <--> 0.9256 at 6/8:
   3 <colab_CNN02> 3003_1215,  32_64_64 ,z=0.01 ,rot=2 , drop out :False data aug:True wind:(5, 5)
[0.2681, 0.9143] <--> 0.9196 at 3/8:
   4 <colab_CNN02> 3003_1143,  32_64_64 ,z=0.00 ,rot=0 , drop out :False data aug:True wind:(5, 5)
[0.2754, 0.9131] <--> 0.9278 at 5/8:
   5 <colab_CNN02> 3003_1113,  32_64_64 ,z=0.08 ,rot=7 , drop out :False data aug:True wind:(5, 5)
[0.2585, 0.9128] <--> 0.9215 at 7/8:
   6 <colab_CNN02> 3003_1209,  32_64_64 ,z=0.03 ,rot=15 , drop out :False data aug:True wind:(5, 5)
[0.2953, 0.9126] <--> 0.9237 at 11/12:
   7 <colab_CNN02> 3003_1110,  32_64_64 ,z=0.03 ,rot=7 , drop out :False data aug:True wind:(5, 5)
[0.2675, 0.9118] <--> 0.919 at 7/8:
   8 <colab_CNN02> 3003_1149,  32_64_64 ,z=0.01 ,rot=15 , drop out :False data aug:True wind:(5, 5)
[0.2556, 0.91] <--> 0.9209 at 7/8:
   9 <colab_CNN02> 3003_1205,  32_64_64 ,z=0.01 ,rot=15 , drop out :False data aug:True wind:(5, 5)
[0.2709, 0.9087] <--> 0.9222 at 8/12:
  10 <colab_CNN02> 3003_1119,  32_64_64 ,z=2.00 ,rot=7 , drop out :False data aug:True wind:(5, 5)
[0.6106, 0.7773] <--> 0.8103 at 6/8:
```

Notice the bad result with the unrealistic zoom =2.

Otherwise the best result gives a test_acc of .9229 and the worst 0.9087

so results difference between the best and word result is about 1.4%

details for the best result

colab_CNN02, 32_64_64 ,z=0.01 ,rot=7 , adam, categorical_crossentropy, 3003_1230

drop out :True data aug:True wind:(5, 5)

test Results [0.2259, 0.9229]

best val accuracy: at epochs 9 /12 value 0.9322

history Params: {'epochs': 12, 'steps': 937.5, 'verbose': 0, 'do_validation': True, 'metrics': ['loss', 'acc', 'val_loss', 'val_acc']}

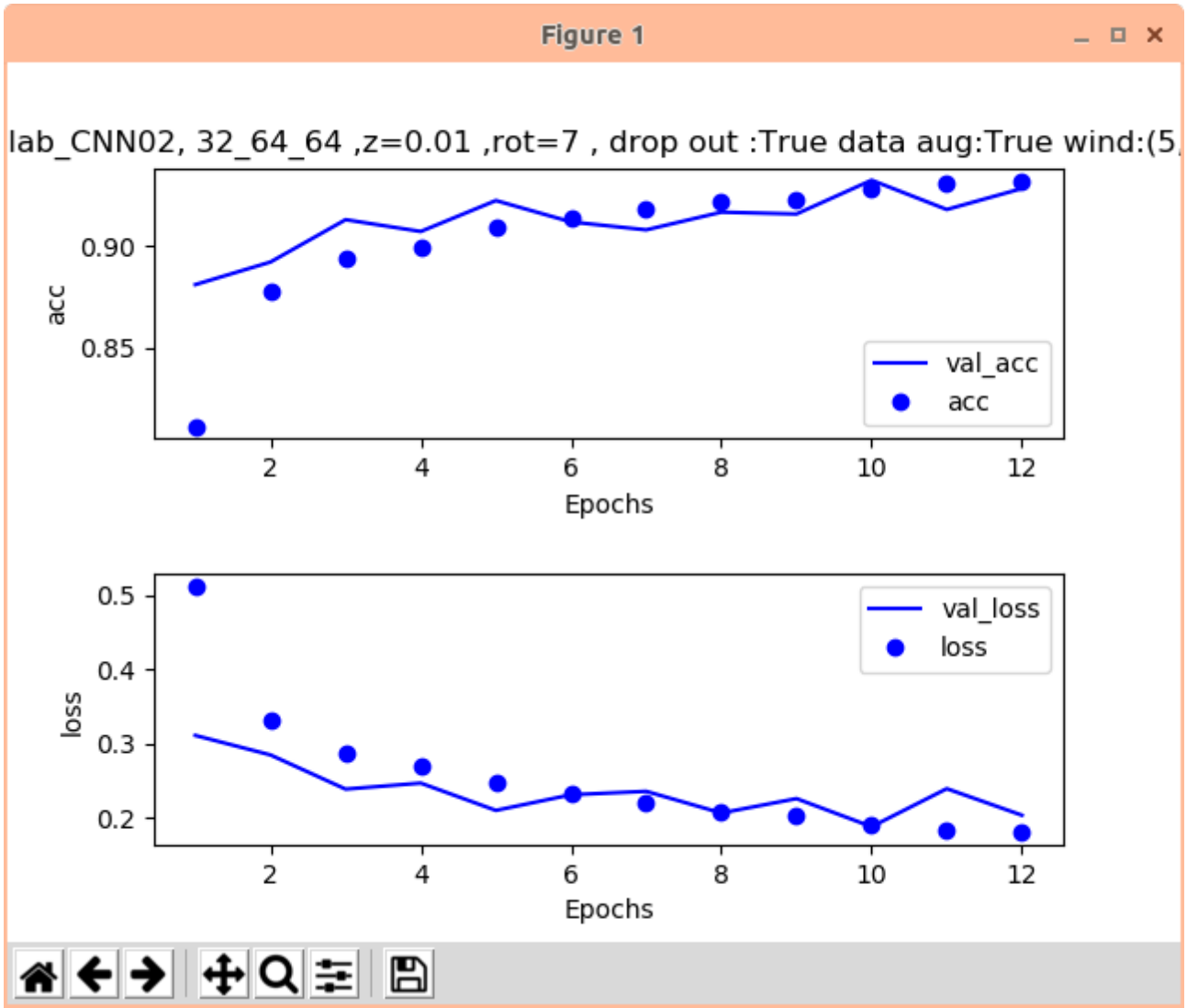

Illustration 9: CNN02 best results 3003

conclusion : best with data aug and dropout

according to the test made 20190330 Data augmentation made a difference of almost 1% for test_acc . The best parameters were zoom= 0.01 and rot = 7 and using drop out (0.5)

Notice the test ind==3 in the list it has (rot=0 and zoom=0) and gives test_Acc 0.9113.
I my eyes this test does not do data augmentation because all parameters are "neutral""

It is important to remember to increase the number of epochs when using dropout.

colab_D01

Some results with only densely connected layers , one or 2 layers are given now as reference.
Note that test_acc for the two layers solution was worst than test_acc of model to beat

10 <colab_D01> 2703_1235, twoLayers, 2 dense layer 512 , 512
[0.3785, 0.8783] <--> 0.892 at 6/10:
11 <colab_D01> 2703_1115, oneLayer, 1 dense layer 1024
[0.358, 0.8729] <--> 0.8825 at 4/5:
12 <colab_D01> 2703_1115, twoLayers, 2 dense layer 1024 , 1024
[0.4306, 0.8586] <--> 0.8698 at 3/5:

4. References and cheat sheet

In this chapter some crucial notions related to the models are presented

4.1. Neural Networks

loading and preparing the data

the dataset (Cholet ch 2.1)

The fashion_MNIST dataset, is a set of 60,000 training images, plus 10,000 test images.

```
     7      (train_images, train_labels), (test_images, test_labels) =
fashion_mnist.load_data()
```

preprocessing the data (Cholet ch2.1)

Before training, the data is processed by reshaping it into the shape the network expects and scaling it so that all values are in the [0, 1] interval.

Previously, our training images, for instance, were stored in an array of shape (60000, 28, 28) of type uint8 with values in the [0,255] interval.

We transform it into a float32 array of shape (60000, 28 * 28) with values between 0 and 1.

```
     8      train_images = train_images.reshape((60000, 28, 28, 1))
     9      train_images = train_images.astype('float32') / 255
    10      test_images = test_images.reshape((10000, 28, 28, 1))
    11      test_images = test_images.astype('float32') / 255
```

preparing the data (Cholet ch3.5.2)

To vectorize the labels, there are two possibilities: you can cast the label list as an integer tensor, or you can use one-hot encoding. One-hot encoding is a widely used format for categorical data, also called categorical encoding. In this case, one-hot encoding of the labels consists of embedding each label as an all-zero vector with a 1 in the place of the label index.

```
    12      train_labels = to_categorical(train_labels)
    13      test_labels = to_categorical(test_labels)
```

training a model (Cholet Ch 3.1)

Training a neural network revolves around the following objects:

Layers, which are combined into a network (or model)

The input data and corresponding targets

The loss function, which defines the feedback signal used for learning

The optimizer, which determines how learning proceeds

To train a model call the fit function

```
network.fit (train_images , train_labels , epochs=5 , verbose=2,
batch_size=128)
```

Adding dropout

Cholet 4.4.3.

Dropout is one of the most effective and most commonly used regularization techniques for neural networks, ... Dropout, applied to a layer, consists of randomly dropping out (setting to zero) a number of output features of the layer during training.

...

The dropout rate is the fraction of the features that are zeroed out; it's usually set between 0.2 and 0.5. At test time, no units are dropped out; instead, the layer's output values are scaled down by a factor equal to the dropout rate, to balance for the fact that more units are active than at training time.

Illustration of hyperparameters

Extracted from Practical guide to hyperparameters search for deep learning models

https://blog.floydhub.com/guide-to-hyperparameters-search-for-deep-learning-models/

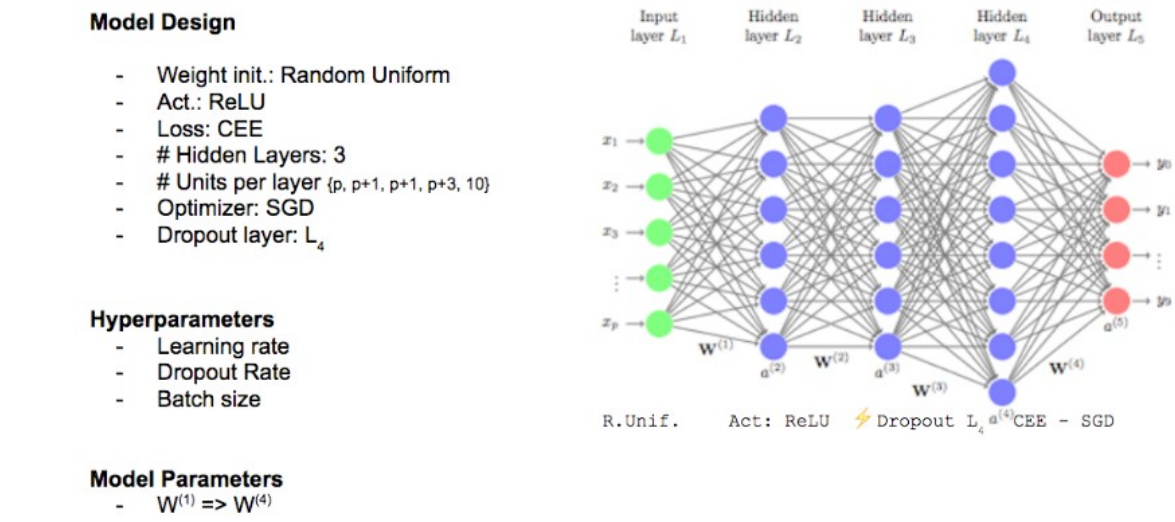

Illustration 10: model design , hyper parameters and parameters

4.2. Convolutional neural network

Short definition

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

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery.

...

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

About convolution

The convolution operation

The fundamental difference between a densely connected layer and a convolution layer is this:

Dense layers learn global patterns in their input feature space (for example, for a MNIST digit, patterns involving all pixels), whereas convolution layers learn local patterns : in the case of

images, patterns found in small 2D windows of the inputs. In this example, these windows are all 3 × 3.

nb of filters and conv window size

Cholet 5.1.1 intro to convnet

Convolutions are defined by two key parameters:

Size of the patches extracted from the inputs— These are typically 3 × 3 or 5 × 5. In the example, they were 3 × 3, which is a common choice.

Depth of the output feature map— The number of filters computed by the convolution. The example started with a depth of 32 and ended with a depth of 64.

In Keras Conv2D layers, these parameters are the first arguments passed to the layer: Conv2D(output_depth, (window_height, window_width)).

A convolution works by sliding these windows of size 3 × 3 or 5 × 5 over the 3D input feature map, stopping at every possible location, and extracting the 3D patch of surrounding features (shape (window_height, window_width, input_depth)). Each such 3D patch is then transformed (via a tensor product with the same learned weight matrix, called the convolution kernel) into a 1D vector of shape (output_depth,). All of these vectors are then spatially reassembled into a 3D output of shape (height, width,vision output_depth).

Every spatial location in the output feature map corresponds to the same location in the input feature map (for example, the lower-right corner of the output contains information about the lower-right corner of the input). For instance, with 3 × 3 windows, the vector output[i, j, :] comes from the 3D patch input[i-1:i+1, j-1:j+1, :].

Structure of a basic convnet

a basic convnet is a stack of Conv2D and MaxPooling2D layers

This is a pattern you'll see in almost all convnets.

Feature map

Convolutions operate over 3D tensors, called feature maps, with two spatial axes (height and width) as well as a depth axis (also called the channels axis). For an RGB image, the dimension of the depth axis is 3, because the image has three color channels: red, green, and blue. For a black-and-white picture, like the MNIST digits, the depth is 1 (levels of gray). **The convolution operation extracts patches from its input feature map** and applies the same transformation to all of these patches, producing an output feature map. This output feature map is still a 3D tensor: it has a width and a height. Its depth can be arbitrary, because the output depth is a parameter of the

layer, and the different channels in that depth axis no longer stand for specific colors as in RGB input; rather, they stand for filters. Filters encode specific aspects of the input data: at a high level, a single filter could encode the concept “presence of a face in the input,” for instance.

about features map depth and size I Cholet 5.2.3

The depth of the feature maps progressively increases in the network (from 32 to 64), whereas the size of the feature maps decreases (from 28x28 to 3x3) , see the output from model.summary() :

padding and strides

Cholet 5.1.1

Note that the output width and height may differ from the input width and height. They may differ for two reasons:

1) Border effects, which can be countered by padding the input feature map
2) The use of strides, which I’ll define in a second Let’s take a deeper look at these notions.

strides

The other factor that can influence output size is the notion of strides. The description of convolution so far has assumed that the center tiles of the convolution windows are all contiguous. But the distance between two successive windows is a parameter of the convolution, called its stride, which defaults to 1. It’s possible to have strided convolutions: convolutions with a stride higher than 1.

padding

Padding consists of adding an appropriate number of rows and columns on each side of the input feature map so as to make it possible to fit center convolution windows around every input tile. For a 3 × 3 window, you add one column on the right, one column on the left, one row at the top, and one row at the bottom.

In Conv2D layers, padding is configurable via the padding argument, which takes two values: "valid", which means no padding (only valid window locations will be used); and "same", which means “pad in such a way as to have an output with the same width and height as the input.”

The padding argument defaults to "valid".

The max-pooling operation

(Cholet 5.1.2)

The role of max pooling is to **aggressively downsample feature maps**. Max pooling consists of

extracting windows from the input feature maps and outputting the max value of each channel. It's conceptually similar to convolution, except that instead of transforming local patches via a

learned linear transformation (the convolution kernel), they're transformed via a hardcoded max tensor operation. A big difference from convolution is that max pooling is usually done with 2 × 2 windows and stride 2, in order to downsample the feature maps by a factor of 2. On the

other hand, convolution is typically done with 3 × 3 windows and no stride (stride 1).

Image preprocessing and data augmenation

Cholet 5.2.5

Data augmentation, is a powerful technique for mitigating overfitting in computer vision.

Overfitting is caused by having too few samples to learn from, rendering you unable to train a model that can generalize to to new data.

Given infinite data, your model would be exposed to every possible aspect of the data distribution at hand: you would never overfit. Data augmentation takes the approach of **generating more training data** from existing training samples, by augmenting the samples via a number of random transformations that yield believable-looking images. The goal is that at training time, your model will never see the exact same picture twice. This helps expose the model to more aspects of the data and generalize better. In Keras, this can be done by configuring a number of random transformations to be performed on the images read by the **ImageDataGenerator instance.**

4.3. CNN and image processing in keras

The information presented here related directly to the programs tested in this lab .

keras.layers.Conv2D

The parameters that were tested in the pgm are filter , kernel_size , padding and strides . There definition and relation to the cons2d layer is given now

The conv2d layer in keras

https://keras.io/layers/convolutional/#conv2d

```
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1),
padding='valid', data_format=None,
dilation_rate=(1, 1), activation=None,
use_bias=True, kernel_initializer='glorot_uniform',
```

bias_initializer='zeros', kernel_regularizer=None,

bias_regularizer=None, activity_regularizer=None,

kernel_constraint=None, bias_constraint=None)

filter

filters: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).

kernel_size

kernel_size: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.

strides

strides: An integer or tuple/list of 2 integers, specifying the strides of the convolution along

the height and width. Can be a single integer to specify the same value for all spatial dimensions.

Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.

padding

padding: one of "valid" or "same" (case-insensitive). Note that "same" is slightly inconsistent

across backends with strides != 1, as described here

max pooling

about model.add(layers.MaxPooling2D((2, 2)))

Max pooling operation for spatial data.

Init signature: layers.MaxPooling2D(pool_size=(2, 2), ... , **kwargs)

Arguments pool_size: integer or tuple of 2 integers, factors by which to downscale (vertical, horizontal). (2, 2) will halve the input in both spatial dimension.

Example of code

```
    16    model.add(layers.MaxPooling2D((2, 2)))
```

Image preprocessing and data augmentation

https://keras.io/preprocessing/image/

Keras has a module with image-processing helper tools, located at keras.preprocessing.image. In particular, it contains the class ImageDataGenerator, which lets you quickly set up Python generators that can automatically turn image files on disk into batches of preprocessed tensors.

ImageDataGenerator class

```
keras.preprocessing.image.ImageDataGenerator(
featurewise_center=False, samplewise_center=False, featurewise_std_normalization=False,
samplewise_std_normalization=False, zca_whitening=False, zca_epsilon=1e-06,
rotation_range=0,width_shift_range=0.0, height_shift_range=0.0, brightness_range=None,
shear_range=0.0, zoom_range=0.0, channel_shift_range=0.0, fill_mode='nearest',
cval=0.0, horizontal_flip=False, vertical_flip=False, rescale=None,
preprocessing_function=None,data_format=None, validation_split=0.0,dtype=None)
```

Arguments used in colab_CNN02

featurewise_center: Boolean. Set input mean to 0 over the dataset, feature-wise.

featurewise_std_normalization: Boolean. Divide inputs by std of the dataset, feature-wise.

rotation_range: Int. Degree range for random rotations.

width_shift_range: Float, 1-D array-like or int

float: fraction of total width, if < 1, or pixels if >= 1.

1-D array-like: random elements from the array.

int: integer number of pixels from interval (-width_shift_range, +width_shift_range)

With width_shift_range=2 possible values are integers [-1, 0, +1], same as with width_shift_range=[-1, 0, +1], while with width_shift_range=1.0 possible values are floats in the half-open interval [-1.0, +1.0[.

height_shift_range: Float, 1-D array-like or int

float: fraction of total height, if < 1, or pixels if >= 1.

1-D array-like: random elements from the array.

int: integer number of pixels from interval (-height_shift_range, +height_shift_range)

With height_shift_range=2 possible values are integers [-1, 0, +1], same as with

height_shift_range=[-1, 0, +1], while with height_shift_range=1.0 possible values are floats in the half-open interval [-1.0, +1.0[.

zoom_range: Float or [lower, upper]. Range for random zoom. If a float, [lower, upper] = [1-zoom_range, 1+zoom_range].

shear_range: Float. Shear Intensity (Shear angle in counter-clockwise direction in degrees)

fill_mode: One of {"constant", "nearest", "reflect" or "wrap"}. Default is 'nearest'. Points outside the boundaries of the input are filled according to the given mode:

'constant': kkkkkkkk|abcd|kkkkkkkk (cval=k)

'nearest': aaaaaaaa|abcd|dddddddd

'reflect': abcddcba|abcd|dcbaabcd

'wrap': abcdabcd|abcd|abcdabcd

flow method

```
flow(x, y=None, batch_size=32, shuffle=True, sample_weight=None, seed=None,
save_to_dir=None, save_prefix='', save_format='png', subset=None)
```

Takes data & label arrays, generates batches of augmented data.

Some Arguments

x: Input data. Numpy array of rank 4 or a tuple.

If tuple, the first element should contain the images

and the second element another numpy array or a list of numpy arrays that gets passed to the output without any modifications.

Can be used to feed the model miscellaneous data along with the images. In case of grayscale data, the channels axis of the image array should have value 1, in case of RGB data, it should have value 3, and in case of RGBA data, it should have value 4.

y: Labels.

batch_size: Int (default: 32).

shuffle: Boolean (default: True).

Returns

An Iterator yielding tuples of (x, y) where x is a numpy array of image data (in the case of a single image input) or a list of numpy arrays (in the case with additional inputs) and y is a numpy array of corresponding labels. If 'sample_weight' is not None, the yielded tuples are of the form (x, y, sample_weight). If y is None, only the numpy array x is returned.

Fit method

Sometime you see datagen.fit() and sometime not. Here is the reason .

```
fit(x, augment=False, rounds=1, seed=None)
```

Fits the data generator to some sample data.

This computes the internal data stats related to the data-dependent transformations, based on an array of sample data.

Only required if featurewise_center or featurewise_std_normalization or zca_whitening are set to True.

Arguments

x: Sample data. Should have rank 4. In case of grayscale data, the channels axis should have value 1, in case of RGB data, it should have value 3, and in case of RGBA data, it should have value 4.

augment: Boolean (default: False). Whether to fit on randomly augmented samples.

rounds: Int (default: 1). If using data augmentation (augment=True), this is how many augmentation passes over the data to use.

seed: Int (default: None). Random see

Fit_generator method

https://keras.io/models/model/

```
fit_generator(generator, steps_per_epoch=None, epochs =1, verbose=1, callbacks=None,
validation_data=None, validation_steps=None, validation_freq=1, class_weight=None,
max_queue_size=10, workers=1, use_multiprocessing=False,
 shuffle=True, initial_epoch=0)
```

Trains the model on data generated batch-by-batch by a Python generator (or an instance of Sequence). The generator is run in parallel to the model, for efficiency. For instance, this allows you to do real-time data augmentation on images on CPU in parallel to training your model on GPU.

The use of keras.utils.Sequence guarantees the ordering and guarantees the single use of every input per epoch when using use_multiprocessing=True.

Some Arguments used in the notebook

generator: A generator or an instance of Sequence (keras.utils.Sequence) object in order to avoid duplicate data when using multiprocessing. The output of the generator must be either
a tuple (inputs, targets)
a tuple (inputs, targets, sample_weights).
This tuple (a single output of the generator) makes a single batch. Therefore, all arrays in this tuple must have the same length (equal to the size of this batch). Different batches may have different sizes. For example, the last batch of the epoch is commonly smaller than the others, if the size of the dataset is not divisible by the batch size. The generator is expected to loop over its data indefinitely. An epoch finishes when steps_per_epoch batches have been seen by the model.

steps_per_epoch: Integer. Total number of steps (batches of samples) to yield from generator before declaring one epoch finished and starting the next epoch. It should typically be equal to ceil(num_samples / batch_size) Optional for Sequence: if unspecified, will use the len(generator) as a number of steps.

epochs: Integer. Number of epochs to train the model. An epoch is an iteration over the entire data provided, as defined by steps_per_epoch. Note that in conjunction with initial_epoch, epochs is to be understood as "final epoch". The model is not trained for a number of iterations given by epochs, but merely until the epoch of index epochs is reached.
verbose: Integer. 0, 1, or 2. Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per epoch.
callbacks: List of keras.callbacks.Callback instances. List of callbacks to apply during training. See callbacks.
validation_data: This can be either

a generator or a Sequence object for the validation data
tuple (x_val, y_val)
tuple (x_val, y_val, val_sample_weights)
on which to evaluate the loss and any model metrics at the end of each epoch. The model will not be
trained on this data.

validation_steps: Only relevant if validation_data is a generator. Total number of steps (batches of samples) to yield from validation_data generator before stopping at the end of every epoch. It should typically be equal to the number of samples of your validation dataset divided by the batch size. Optional for Sequence: if unspecified, will use the len(validation_data) as a number of steps.

Returns
A History object. Its History.history attribute is a record of training loss values and metrics values at successive epochs, as well as validation loss values and validation metrics values (if applicable).

4.4. about dense layers at the end of convolutional neural networks

https://www.quora.com/Why-are-the-often-dense-layers-at-the-end-of-convolutional-neural-networks

So when we talk about convolution and max-pooling layers, the network is learning the relevant features. For example if you have a task to identify whether the image contains a face or not. CNN automatically learns the features in the form of filters from these images. In the final layers, you will have filters which resembles an eye, nose, ears, etc.

...

Now you have all the features required as an input of the neural network. Normal ANN's work is required, adding dense layers and output nodes at the final layer.

xxxxxxxxxxxxx end of document