```
1: ////////////////////////////////////////////////////
2: CNN Model Validation
3: ____________________________________________________
4: maXbox Starter 89_1 - Train a CNN (Convolutional Neural Network) - Max Kleiner
5:
6: "space your code" "face your code"
7:     — mX4
8:  https://my6.code.blog/2021/09/08/improver-4/
9:
10: This tutor explains training the so called CIFAR-10 Image Classifier.
11: This command line tool runs the CAI Network with CIFAR10 files. We start with a
    CIFAR-10 SELU Classification Example. The SELU activation is self-normalizing the
    neural network; and what does that mean?
12:
13: Well, lets start with, what is normalization? Simply put, we first subtract the
    mean, then divide by the standard deviation. So the components of the network
    (weights, biases and activations) will have a mean of zero and a standard deviation
    of one after normalization. This will be the output value of the SELU activation
    function.
14:
15: A note at the beginning: Its not that easy to define the right model structure
    (layers definition), for example you set the supervised classes wrongfully to 2:
16:
17:   NN:= TNNet.Create();
18:   NumClasses:= 2;
19:   fLearningRate:= 0.009;
20:
21: Then the fit-method as a trainer raise an exception:
22:   Image mX4 Fit Computing starts...
23: ComputeOutputErrorWith should have same sizes.Neurons:0 Output:2 Expected output:10
    Error:2 Error times Deriv:2
24: Error - invalid output value at loss function:0.0000
25: So you train with a 10 classifier and the model expects 10 classes to predict:
26:
27:   NN.AddLayer( TNNetFullConnectLinear.Create28(10,0) );
28:   NN.AddLayer( TNNetSoftMax.Create() );
29:
30: CAI NEURAL API is a pascal based neural network API optimized for AVX, AVX2 and
    AVX512 instruction sets plus OpenCL capable devices including AMD, Intel and NVIDIA
    for GPU capabilities. This API has been tested under Windows and Linux. On January
    this year we got in Delphi support for OpenCL and Threads.
31: This project and API is a sub-project from a bigger and older project called CAI
    and its sister to Keras/TensorFlow based K-CAI NEURAL API.
32:
33: The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000
    images per class. There are 50000 training images and 10000 test images.
34: This example has interesting aspects to look at: Its source code is very small and
    Layers are added sequentially. Then the Training hyper-parameters are defined
    before calling the fit method. You need fit() to train the model! So we start with
    the test-class from the CAI library and we create the neural net layers too, this
    is how a sequential SELU CNN array of layers is added:
35:
36:  procedure TTestCNNAlgoDoRunClassifier89;
37:  //Application.Title:='CIFAR-10 SELU Classification Example';
38:   var
39:     NN: THistoricalNets;
40:     NeuralFit: TNeuralImageFit;
41:     ImgTrainingVolumes, ImgValidationVolumes, ImgTestVolumes: TNNetVolumeList;
42:   begin
43:     if not CheckCIFARFile() then begin
44:       //Terminate;
45:       //exit;
46:       memo2.lines.add('TNNetConvolutionLinear CIFAR-10 Files missing!')
47:     end;
48:     WriteLn('Creating Neural Network...');
49:     NN:= THistoricalNets.Create();
50:     //TestDataParallelism( NN);
```

```
51:
52:     NN.AddLayer(TNNetInput.Create4(32, 32, 3) );
53:     //Function InitSELU( Value : TNeuralFloat) : TNNetLayer');
54:     NN.AddLayer(TNNetConvolutionLinear.Create(64,5,2,1,1)).InitSELU(0).
55:                                                          InitBasicPatterns();
56:     NN.AddLayer( TNNetMaxPool.Create44(4,0,0) );
57:     NN.AddLayer( TNNetSELU.Create() );
58:     NN.AddLayer( TNNetMovingStdNormalization.Create() );
59:     NN.AddLayer( TNNetConvolutionLinear.Create(64,3,1,1,1)).InitSELU(0);
60:     NN.AddLayer( TNNetSELU.Create() );
61:     NN.AddLayer( TNNetConvolutionLinear.Create(64,3,1,1,1)).InitSELU(0);
62:     NN.AddLayer( TNNetSELU.Create() );
63:     NN.AddLayer( TNNetConvolutionLinear.Create(64,3,1,1,1)).InitSELU(0);
64:     NN.AddLayer( TNNetSELU.Create() );
65:     NN.AddLayer( TNNetConvolutionLinear.Create(64,3,1,1,1)).InitSELU(0);
66:     NN.AddLayer( TNNetDropout.Create12(0.5,1) );
67:     NN.AddLayer( TNNetMaxPool.Create44(2,0,0) );
68:     NN.AddLayer( TNNetSELU.Create() );
69:     NN.AddLayer( TNNetFullConnectLinear.Create28(10,0) );
70:     NN.AddLayer( TNNetSoftMax.Create() );
71:     memo2.lines.add('TNNetConvolutionLinear model add')
72:     TestDataParallelism( NN);
73:     memo2.lines.add('TestDataParallelism( NN) passed')
74:     CheckCIFARFile()
75:     try
76:      CreateCifar10Volumes(ImgTrainingVolumes, ImgValidationVolumes,
77:                                              ImgTestVolumes, csEncodeRGB);
78:     except
79:       memo2.lines.add('TNNetConvolutionLinear CIFAR Files missing!')
80:     end;
81:     NeuralFit:= TNeuralImageFit.Create;
82:     NeuralFit.FileNameBase:= 'ImageClassifierSELU_Tutor89_5';
83:     NeuralFit.InitialLearningRate:= 0.0004;
84:          // SELU seems to work better with smaller learning rates.
85:     NeuralFit.LearningRateDecay:= 0.03;
86:     NeuralFit.StaircaseEpochs:= 10;
87:     NeuralFit.Inertia:= 0.9;
88:     NeuralFit.L2Decay:= 0.00001;
89:     NeuralFit.verbose:= true;
90:     try
91:       NN.DebugStructure();
92:     //NN.DebugWeights();
93:       NeuralFit.Fit(NN, ImgTrainingVolumes, ImgValidationVolumes,
94:                     ImgTestVolumes, {NumClasses=}10,{batchsize=}64,{epochs=}5);
95:     finally
96:       NeuralFit.Free;
97:       NN.Free;
98:       ImgTestVolumes.Free;
99:       ImgValidationVolumes.Free;
100:      ImgTrainingVolumes.Free;
101:      //Terminate;
102:      writeln('3 Volumes + NN + NF freed...')
103:    end;
104:  end;
105:
106: The first layer on line 52 sets the 32x32x3 Input Image to define. The second layer
     defines features and feature size with a first Convolution and InitBasicPatterns:
107:
108:  NN.AddLayer(TNNetConvolutionLinear.Create(64,5,2,1,
     1)).InitSELU(0).InitBasicPatterns();
109:
110:  NN.AddLayer(TNNetConvolutionReLU.Create({Features=}64,
111:              {FeatureSize=}3, {Padding=}1, {Stride=}1, {SuppressBias=}1));
112:
113: Then we load CreateCifar10Volumes() ,after we check it, on line 75:
114:
115: Loading 10K images from file "data_batch_1.bin" ...
```

```
116:  GLOBAL MIN MAX -2  1.984375
117:  Done...
118: Loading 10K images from file "data_batch_2.bin" ...
119:  GLOBAL MIN MAX -2  1.984375
120:  Done...
121: Loading 10K images from file "data_batch_3.bin" ...
122:  GLOBAL MIN MAX -2  1.984375
123:  Done...
124: Loading 10K images from file "data_batch_4.bin" ...
125:  GLOBAL MIN MAX -2  1.984375
126:  Done...
127: Loading 10K images from file "data_batch_5.bin" ...
128:  GLOBAL MIN MAX -2  1.984375
129:  Done...
130: Loading 10K images from file "test_batch.bin" ...
131:  GLOBAL MIN MAX -2  1.984375
132:  Done...
133:
134: You can download the 5 files with Python4maXbox (P4M):
135:
136: import os
137: import urllib.request
138:
139: if not os.path.isfile('cifar-10-batches-bin/data_batch_1.bin'):
140:   print("Downloading CIFAR-10 Files")
141:   url = 'https://www.cs.toronto.edu/~kriz/cifar-10-binary.tar.gz'
142:   urllib.request.urlretrieve(url, './file.tar')
143:
144: total 166080
145: -rw-r--r-- 1 root root 170052171 Jul 15 07:15 file.tar
146: drwxr-xr-x 5 root root      4096 Jul 15 07:15 mtprocs/
147: drwxr-xr-x 7 root root      4096 Jul 15 07:15 neural-api/
148: drwxr-xr-x 1 root root      4096 Jul 10 16:29 sample_data/
149:
150: We switch on the NN.DebugStructure(); on line 90:
151:  Layers: 17
152:  Neurons: 331
153:  Weights: 162498
154:  Sum: -3.81792163848877
155: Layer 0 Neurons:0 Weights:0 TNNetInput(32,32,3,0,0) Output:32,32,3 Learning
     Rate:0.0100  Inertia:0.90  Weight Sum:  0.0000
156:  Branches:1
157: Layer 1 Neurons:64 Weights:4800 TNNetConvolutionLinear(64,5,2,1,1) Output:32,32,64
     Learning Rate:0.0100  Inertia:0.90  Weight Sum:  0.0000
158:  Parent:0
159:  Branches:1
160:
161: and so on this you can see with the NN.DebugStructure(); till last layer 16 as the
     softmax layer:
162: Layer 16 Neurons:0 Weights:0 TNNetSoftMax(0,0,0,0,0) Output:10,1,1 Learning
     Rate:0.0100  Inertia:0.90  Weight Sum:  0.0000
163:  Parent:15, Branches:0
164:
165: Now we let run the neural network with the so called fit method:
166:
167: Debug Data.LoadFromString: 2
168: Learning rate:0.000400 L2 decay:0.000010 Inertia:0.900000 Batch size:64 Step
     size:64 Staircase epochs mX4:10
169: Training images: 40000
170: Validation images: 10000
171: Test images: 10000
172: Image mX4 Fit Computing starts...
173: 0 -1 0
174: 0 -1 0
175: 0 -1 0
176: 0 -1 0
177: 0 -1 0
```

```
178: 640 Examples seen. Accuracy: 0.1642 Error: 1.79993 Loss: 2.30224 Threads: 1 Forward
     time: 0.11s Backward time: 0.39s Step time: 2.51s
179: 1280 Examples seen. Accuracy: 0.1558 Error: 1.80014 Loss: 2.30331 Threads: 1
     Forward time: 0.06s Backward time: 0.41s Step time: 2.46s
180: 1920 Examples seen. Accuracy: 0.1497 Error: 1.80016 Loss: 2.30340 Threads: 1
     Forward time: 0.03s Backward time: 0.46s Step time: 2.44s
181: 40000 of samples have been processed.
182: Starting Validation.
183: Epochs: 5 Examples seen:200000 Validation Accuracy: 0.0980 Validation Error: 1.8000
     Validation Loss: 2.3028 Total time: 14.31min
184: Image mX4 FThreadNN[0].DebugWeights(); skipped...
185: Epoch time: 2.0000 minutes. 5 epochs: 0.1600 hours.
186: Epochs: 5. Working time: 0.24 hours.
187: CAI maXbox Neural Fit Finished.
188: 3 Volumes + NN + NF freed...
189: ▯▯▯ mX4 executed: 04/11/2021 16:07:45  Runtime: 0:14:25.509  Memload: 43% use
190: PascalScript maXbox4 - RemObjects & SynEdit
191: C:\maXbox\works2021\maxbox4\examples\1065__CAI_2_SiImageClassifier21_Tutor_89_test2.txt
     File stored
192: 1065__CAI_2_SiImageClassifier21_Tutor_89_test2.txt in maxboxdef.ini stored: 16:14:36
193:
194: So after only 5 epochs (line 93) we get only 9 % accuracy and thats nonsense or too
     low. But with more epochs [50], validation and therefore more time we can get:
195: Starting Testing.
196: Epochs: 50 Examples seen:2000000 Test Accuracy: 0.8383 Test Error: 0.4463 Test
     Loss: 0.4969 Total time: 162.32min
197: Epoch time: 2.7 minutes. 100 epochs: 4.5 hours.
198: Epochs: 50. Working time: 2.71 hours.
199: https://colab.research.google.com/drive/1A5oayNDOeRVzcSy4LHCXsHCjbbed_RjM
200: https://github.com/joaopauloschuler/neural-
     api/blob/master/examples/ImageClassifierSELU/ImageClassifierSELU.lpr
201:
202: Then we get 2 output files, definied in line 81:
203: NeuralFit.FileNameBase:= 'ImageClassifierSELU_Tutor89_5';
204: • ImageClassifierSELU_Tutor89_5.csv
205: • ImageClassifierSELU_Tutor89_5.nn
206:
207: epoch training accuracy  training loss  training error
208: 1     0.0991             2.3048         1.8004
209: 2     0.0943             2.3024         1.7999
210: 3     0.1014             2.3031         1.8001
211: 4     0.0934             2.3011         1.7997
212: 5     0.0964             2.3034         1.8001
213:
214: So what is the context of the 2 files. The csv is just the log of
215: the training with the hyperparameters such as learning rate:
216: epoch training accuracy training loss training error validation accuracy
217: validation loss validation error learning rate time test accuracy test loss test
     error
218: The *.nn file serves as a pretrained file (FAvgWeight) to classify
219: or predict images we trained on. Also the CIFAR-10 classification
220: examples with experiments/testcnnalgo/testcnnalgo.lpr and a number
221: of CIFAR-10 classification examples are available on /experiments.
222:
223: procedure TTestCifar10Algo;
224: var
225:   NN: TNNet;
226:   NeuralFit: TNeuralImageFit;
227:   ImgTrainingVolumes, ImgValidationVolumes, ImgTestVolumes: TNNetVolumeList;
228:   NumClasses: integer;
229:   fLearningRate, fInertia: single;
230:  begin
231:  //This is how a sequential CNN array of layers is added:
232:    NN := TNNet.Create();
233:    NumClasses:= 10;
234:    fLearningRate := 0.001;
235:    fInertia := 0.9;
```

```
236:    NN.AddLayer(TNNetInput.Create(32, 32, 3)); //32x32x3 Input Image
237:    NN.AddLayer(TNNetConvolutionReLU.Create({Features=}16,
238:    {FeatureSize=}5, {Padding=}0, {Stride=}1, {SuppressBias=}0));
239:    NN.AddLayer(TNNetMaxPool.Create({Size=}2));
240:    NN.AddLayer(TNNetConvolutionReLU.Create({Features=}32,
241:    {FeatureSize=}5, {Padding=}0, {Stride=}1, {SuppressBias=}0));
242:    NN.AddLayer(TNNetMaxPool.Create({Size=}2));
243:    NN.AddLayer(TNNetConvolutionReLU.Create({Features=}32,
244:    {FeatureSize=}5, {Padding=}0, {Stride=}1, {SuppressBias=}0));
245:    NN.AddLayer(TNNetLayerFullConnectReLU.Create({Neurons=}32));
246:    NN.AddLayer(TNNetFullConnectLinear.Create(NumClasses));
247:    NN.AddLayer(TNNetSoftMax.Create());
248:    writeln(NN.SaveDataToString);
249:   //readln();
250:  end;
251:
252: Convolutional neural networks are distinguished from other neural networks by their
     superior performance with image, speech, text or audio signal inputs. They have
     three main types of layers, which are:
253:
254: •  Convolutional layer
255: •  Pooling layer
256: •  Fully-connected (FC) layer
257:
258: The convolutional layer is the first layer of a convolutional network. While
     convolutional layers can be followed by additional convolutional layers or pooling
     layers, the fully-connected layer is the final layer. With each layer, CNN
     increases in its complexity, identifying greater portions of the image. Earlier
     layers focus on simple features, such as colors and edges. As the image data
     progresses through layers of the CNN, it starts to recognize larger elements as
     feature maps or shapes of object until it finally identifies intended object.
259:
260: https://www.ibm.com/cloud/learn/convolutional-neural-networks
261:
262: After each convolution operation, a CNN applies a Rectified Linear Unit (ReLU)
     transformation to a feature map, introducing nonlinearity to the model.
263: NN.AddLayer(TNNetConvolutionReLU.Create({Features=}32,
264:    {FeatureSize=}5, {Padding=}0, {Stride=}1, {SuppressBias=}0));
265:
266: Convolutional Layer Tuning
267: Performance is a crucial thing with many aspects. Other Pascal or Python
     implementations (or older or still-under development versions of CPython or maXbox
     bytecode ) may have slightly different performance characteristics. However, it is
     generally safe to assume that they are not slower by more than a factor of O(log
     n).
268:
269: {$Conclusion}:
270: Convolutional neural networks power image recognition and computer vision tasks.
     Computer vision is a field of artificial intelligence (AI) that enables computers
     and systems to derive meaningful information from digital images, videos and other
     visual inputs, and based on those inputs, it can take action.
271: With most algorithms that handle image processing, the filters are typically
     created by an engineer based on heuristics. CNNs can learn what characteristics in
     the filters are the most important. That saves a lot of time and trial and error
     work since we dont need as many parameters.
272: Neural network models predicting data from a probability distribution that is
     multinomial over an n values discrete variable, use a Softmax activation function
     for the output layer activation function. Softmax is typically used as the
     activation function when 2 or more class labels are present in the class membership
     in the classification of multi-class problems.
273:
274: Script Ref: 1065__CAI_2_SiImageClassifier21_Tutor_89_test2.txt
275:
276: http://www.softwareschule.ch/examples/1065__CAI_2_SiImageClassifier21_Tutor_89.txt
277: https://entwickler-konferenz.de/blog/machine-learning-mit-cai/
278: https://www.freecodecamp.org/news/convolutional-neural-network-tutorial-for-
     beginners/
```