//

Classify Cifar10

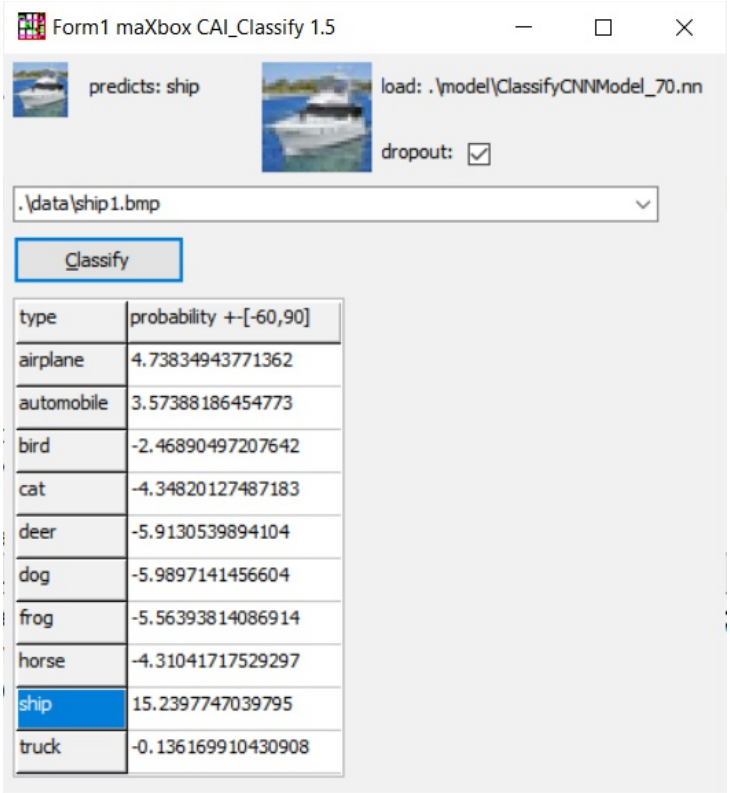
maXbox Starter 105 – CIFAR-10 Image Classifier with loading and testing a pre-trained model.

"Train your models during night – called night train ,-)".

This machine learning tutor explains a classifier based on the so called CIFAR-10 Image Classifier with a pre-trained model.
The pre-trained model is a file: *ClassifyCNNModel_70.nn*

As the name implies, it is a CNN-model. A Convolutional[1] Neural Network (CNN) is a type of deep learning algorithm that is particularly for image recognition and object-detection tasks. It is made up of multiple layers, including convolution layers, pooling layers, and fully connected layers.

Now let's have a look at the app/script below with individual images from Cifar test data. For this, we wrote two useful functions. The first one returns the label associated with predictions made by the model. The second one accepts one image as an argument. Then it will show the image, the prediction the model made and the actual class the image belongs to. Also other probabilities are shown in the multi-classification grid:



Pic:1135_classifiergui2_tutor105.png

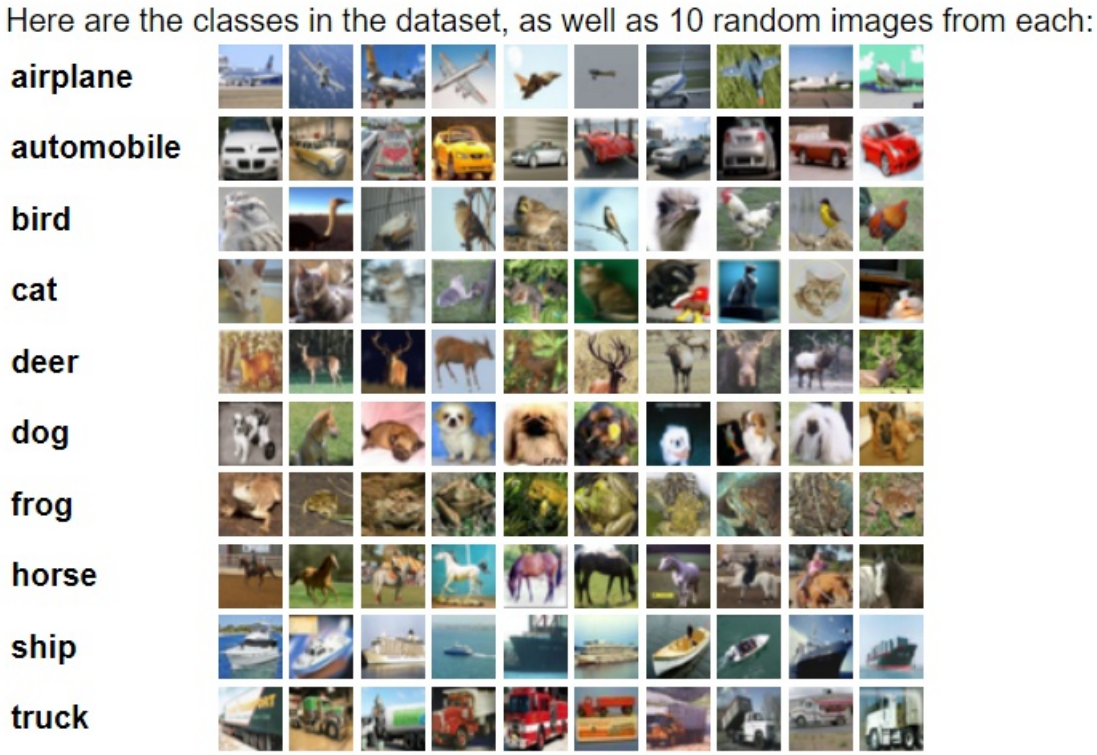
This app allows you to classify pictures from an airplane to a truck or a train. And you see similarities for example a ship (15.2) has some elements of an airplane or automobile (4.7, 3.5) in his feature map.

1 CNNs also known as Shift Invariant or Space Invariant Artificial NN.

Specifically, models are comprised of small linear filters and the result of applying filters called activation maps, or more generally, feature maps. Looking at the following dataset, it will extract features in a constant dot product, even though images has shadows or positioned with various angle. It is important to note that filters acts as feature detectors from the original input image, in our case 32*32 bitmaps.

```
Const PICPATH   = '.\data\';
      TRAINPATH = '.\model\ClassifyCNNModel_70.nn';
```

The proper way to use a CNN doesn't exists. The advice for ugly score is to use a smaller learning rate or larger batch size for the weights that are being fine-tuned and a higher one for the randomly initialized weights (e.g. the ones in the softmax classifier) *TNNetSoftMax*. Pre-trained weights (in *ClassifyCNNModel_70.nn)* are already good, they need to be fine-tuned, not distorted.



Pic:1135_visualcifarset.png

The learning rate is the crucial hyper-parameter used during the training of deep convolution neural networks (DCNN) to improve model accuracy;
By following these ways you can make a CNN model that has a validation set accuracy of more than 95 % but the question is how specific or relevant is this validation.
In our example, values smaller than 0.7 mean false while values bigger than 0.7 mean true. This is called monopolar encoding. CAI also supports bipolar encoding (-1, +1). Let's have a look directly into the source code for the labels and the classify method:

```
var cs10Labels: array[0..9] of string;

procedure setClassifierLabels;
begin
```

```
  cs10Labels[0]:= 'airplane';
  cs10Labels[1]:= 'automobile';
  cs10Labels[2]:= 'bird';
  cs10Labels[3]:= 'cat';
  cs10Labels[4]:= 'deer';
  cs10Labels[5]:= 'dog';
  cs10Labels[6]:= 'frog';
  cs10Labels[7]:= 'horse';
  cs10Labels[8]:= 'ship';
  cs10Labels[9]:= 'truck';
end;
```

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. The dataset is divided into five training batches and one test batch, each with 10000 images.
We now build the Convolution neural network by using 1 Convolution layer, 4 Relu-activation function, dropout- and pooling-layer, 1 fully Connected layer and a SoftMax activation function. Below is the list of all layers which we also define the optimizer and a loss function for the optimizer:

```
Debug TNNet.Struct.LoadFromString ST:
-1)TNNetInput:32;32;3;0;0;0;0;0
0)TNNetConvolutionLinear:64;5;2;1;1;0;0;0
1)TNNetMaxPool:4;4;0;0;0;0;0;0
2)TNNetConvolutionReLU:64;3;1;1;1;0;0;0
3)TNNetConvolutionReLU:64;3;1;1;1;0;0;0
4)TNNetConvolutionReLU:64;3;1;1;1;0;0;0
5)TNNetConvolutionReLU:64;3;1;1;1;0;0;0
6)TNNetDropout:2;0;0;0;0;0;0;0
7)TNNetMaxPool:2;2;0;0;0;0;0;0
8)TNNetFullConnectLinear:10;1;1;0;0;0;0;0
9)TNNetSoftMax:0;0;0;0;0;0;0;0
```

The **main** procedure to classify incoming images loads the model, decides dropout or not (later more) and creates input- and output-volumes with the shape of 32;32;3 or a 32x32x3 volume:

```
begin
  NN:= THistoricalNets.create; //TNNet.Create();
  NN.LoadFromFile(TRAINPATH);
  label2.caption:= 'load: '+TRAINPATH;

  if chkboxdrop.checked then
    NN.EnableDropouts(true) else
       NN.EnableDropouts(false);
  pInput:= TNNetVolume.Create0(32, 32, 3, 1);
  pOutPut:= TNNetVolume.Create0(10, 1, 1, 1);

  LoadPictureIntoVolume(image1.picture, pinput);
  pInput.RgbImgToNeuronalInput(csEncodeRGB);
  NN.Compute65(pInput,0);
  NN.GetOutput(pOutPut);
  writeln('result get class type: '+itoa(pOutPut.GetClass()));
```

Then, we need to add *RgbImgToNeuronalInput* and with the use of SoftMax, we can print output class probabilities to show in the Stringgrid.

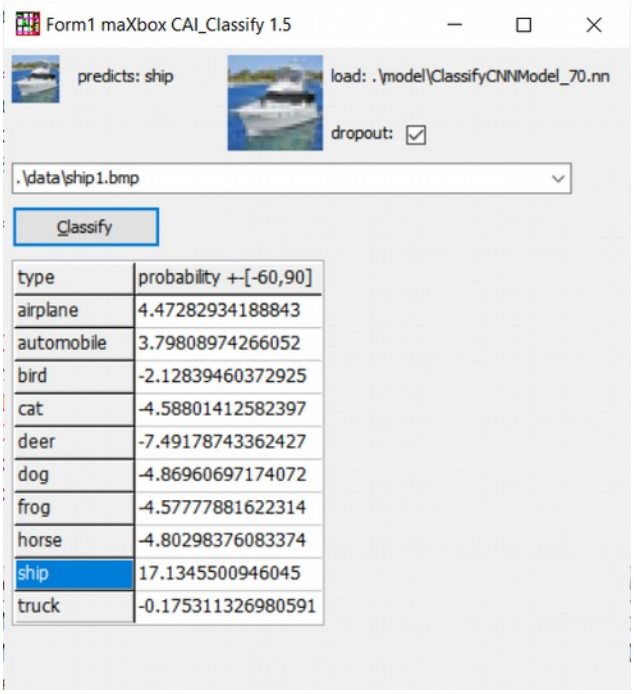
The ***.nn** file in TRAINPATH serves as a pre-trained file (*FAvgWeight*) to classify/predict images we trained on. Also the CIFAR-10 classification examples with *experiments/testcnnalgo/testcnnalgo.lpr* and a number of CIFAR-10 classification examples are available on */experiments*.
Imagine the accuracy goes up and the loss-function (error-rate) goes down. The loss function is the bread & butter of modern machine learning; it takes your algorithm from theoretical to practical and transforms matrix multiplication into deep learning.

Drop out experiments

It's usually very hard to understand neuron by neuron how a neural network dedicated to image classification internally works.
In this technique, an arbitrary neuron is required to activate and then the same back-propagation method used for learning is applied to an input image producing an image that this neuron expects to see.
Adding neurons and neuronal layers is often a possible way to improve artificial neural networks when you have enough hardware and computing time. In the case that you can't afford time, parameters and hardware, you'll look for efficiency with Separable Convolutions (SCNN).
But there's another for me interesting point, the dropout regularization. The dropout layer is a mask that nullifies the contribution of some neurons towards the next layer and leaves unmodified all others. In our model you can see layer 6 as the dropout:

6) **TNNetDropout**:2;0;0;0;0;0;0;0

We can apply a Dropout layer to the input vector, in which case it nullifies some of its features; but we can also apply it to a hidden layer, in which case it nullifies some hidden neurons.

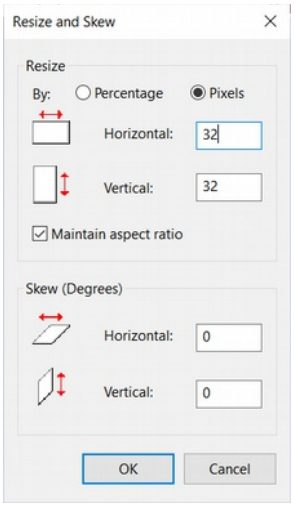


Pic:1135_classifiergui3_tutor105.png

Dropout is a technique where randomly selected neurons are ignored during training. They are "dropped out" randomly. Every time you click on

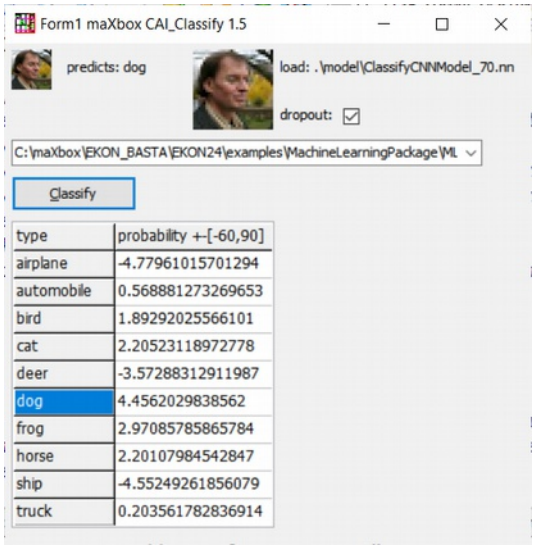
Classify you get another result in small changes. The ship in the first screen is classified with 15.2 now above with 17.1. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass, and any weight updates are not applied to the neuron on the backward pass. If you want a comparable result, deactivate the checkbox. So what's the advantage of dropout? You can imagine that if neurons are randomly dropped out of a network during training, other neurons will have to step in and complement the representation required to make predictions for those missing neurons.

The effect is that a CNN (or whatever deep learning nn) becomes less sensitive to some specific weights of neurons. This in turn, results in a network capable of better generalization and less likely to specialise training data, means you get on the average with new or in training unseen pictures a better result. For example we take a new picture out of the known classification labels. For this we convert the picture at first as a cifar 32*32 24-bit bitmap:



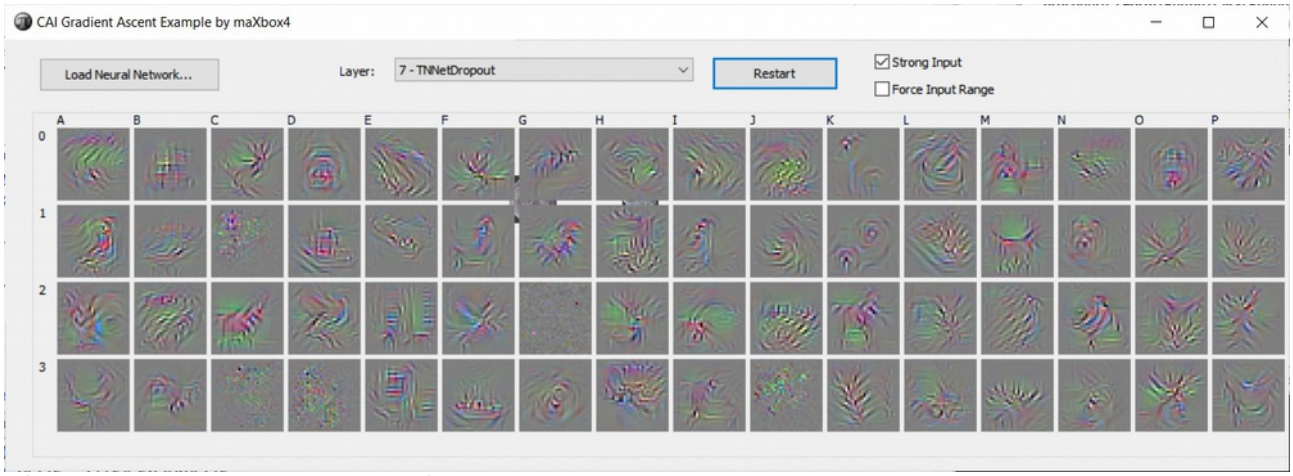
Pic:1135_resizebitmap_tutor105.png

Then we load the picture as *.bmp (just drop in the *./data* directory and try to classify an unknown class with unseen training, but and that's sort of surprising, we get a result:



Pic:1135_dropout_tutor105.png

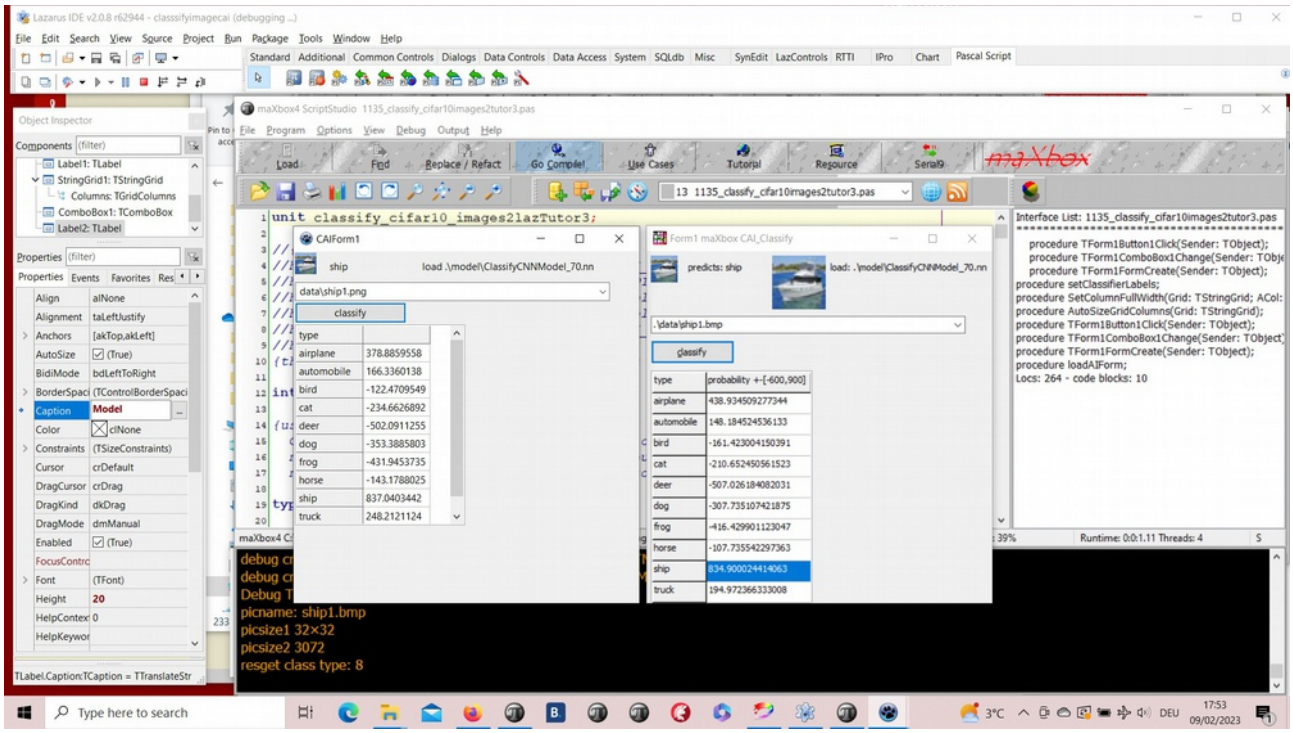
So the result is devastating and amazing too, somewhat between dog and horse is the kind of bionics!
But this can be a baseline for similarities in a recommender system or you can classify the age or sex of a person, enable also in a gender gap research. In our model, a new dropout layer between *TNNetConvolutionReLU* (activation layer) and the hidden layer *TNNetMaxPool* was added. You can also make them visible, but its more art than science or more science than fiction:



Pic:1135_dropout_tutor105.jpg

This visual technique above used to help with the understanding about what individual neurons represent is called Gradient Ascent. You can find more about gradient ascent at http://yosinski.com/deepvis.

In the archive *MachineLearningPackage.zip* you find the script, model and data you need, which works with Lazarus, Jupyter and maXbox too:



Pic:lazarus_maxbox_classifier1.jpg

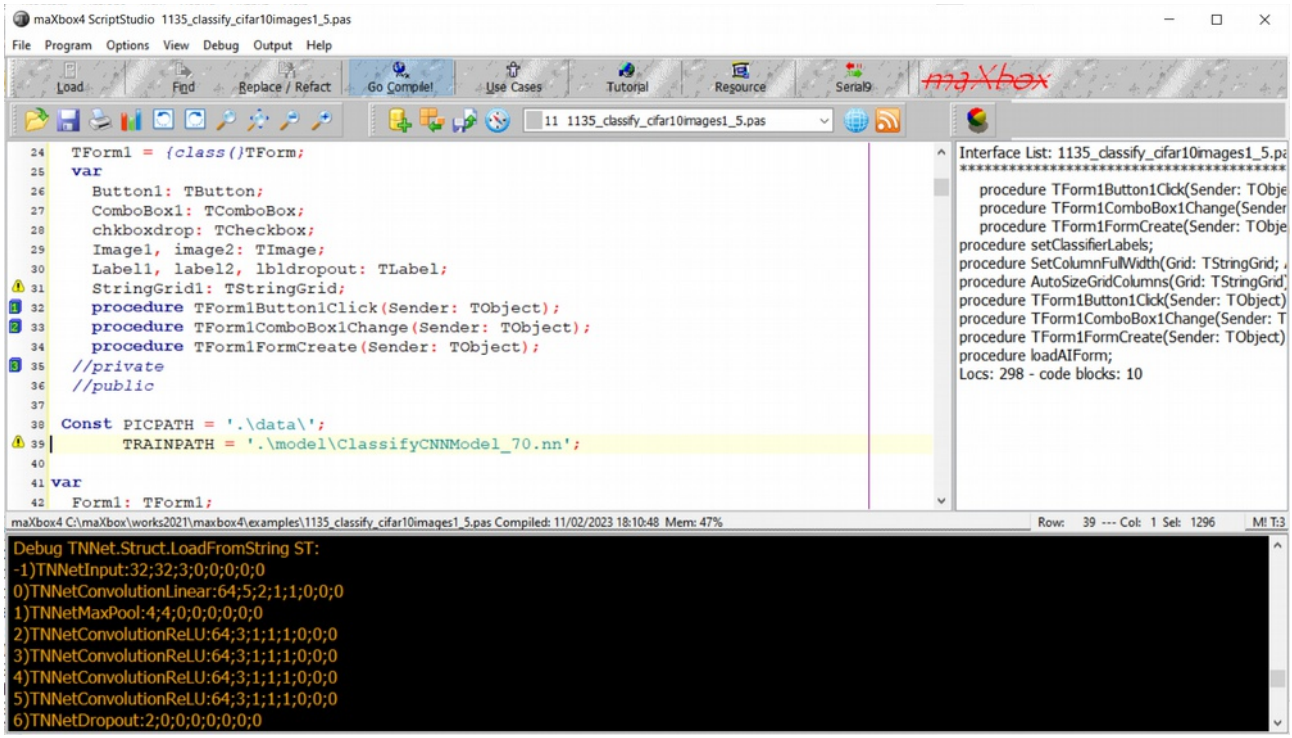
The *FormCreate()* can also be triggered with this few lines of code:

```
procedure TForm1FormCreate(Sender: TObject);
var k,t: integer;
  items: TStringList;
begin
  items:= TStringList.create;
  for k:= 0 to 9 do
    StringGrid1.Cells[0, k+1]:= cs10Labels[k];

  //FindAllFiles(ComboBox1.Items, 'csdata');
  FindFiles(exepath+'data', '*.bmp',items);
  writeln(items.text);
  for t:= 1 to items.count-1 do
     ComboBox1.Items.add(items[t]);
  if ComboBox1.Items.Count > 0 then begin
    ComboBox1.text:= ComboBox1.Items[0];
    if FileExists(ComboBox1.text) then begin
      Image1.Picture.LoadFromFile(ComboBox1.text);
      Image2.Picture.LoadFromFile(ComboBox1.text);
      label1.Caption:= extractfilename(ComboBox1.text);
    end;
  end;
end;
```

FindFiles(exepath+'data','*.bmp',items) is an adoption from Lazarus. In the case that an input image isn't 32x32, you can resize (via copying):

```
TVolume.CopyResizing(Original: TVolume; NewSizeX,NewSizeY: integer);
And Given that you have a trained NN, you could call this:
TNeuralImageFit.ClassifyImage(pNN:TNNet; pImgInput,pOutput:TNNetVolume);
```



Pic:1135_classifier_box_tutor105.png

Conclusion:

The neural-api or CAI API (Conscious Artificial Intelligence) is something like TensorFlow for Pascal and is platform-independent open source library for artificial intelligence or machine learning in the field of speech recognition, image classification, OpenCL, big data, data science and computer vision2.

To be able to run this example, you'll need to load an already trained neural network file and then select the image you intend to classify. CAI stores both architecture and weights into the same *.nn file! Dropout is a simple and powerful regularization technique for neural networks and deep learning models.

Loss functions are different based on a problem statement to which deep learning is being applied. The cost function is another term used interchangeably for the loss function, but it holds a more different meaning. A loss function is for a single training example, while a cost function is an average loss over the complete train dataset.

https://github.com/joaopauloschuler/neural-api

https://sourceforge.net/projects/cai/files/

https://github.com/maxkleiner/neural-api

Reference:

As a Jupyter Notebook:

https://github.com/maxkleiner/maXbox/blob/master/EKON24_SimpleImageClassificationCPU.ipynb

and the same in colab.research:

https://colab.research.google.com/github/maxkleiner/maXbox/blob/master/EKON24_SimpleImageClassificationCPU.ipynb

The whole package with app, script, tutorial, data and model:

https://github.com/maxkleiner/neural-api/blob/master/examples/SimpleImageClassifier/MachineLearningPackage.zip

Doc and Tool: https://maxbox4.wordpress.com

Script Ref: 1135_classify_cifar10images1_5.pas

Max Kleiner 12/02/2023