Exercise 4: Object Classification Using Deep Learning

Download the file for exercise 4

Part 1: Implementation (10 points)

In this exercise you will implement a neural network to classify images of the CIFAR10 dataset (https://www.cs.toronto.edu/~kriz/cifar.html). It consists of 60000 images with a resolution of 32x32. The different classes and example images for each classes are shown in Figure 1 below. We will use the library Keras with the TensorFlow backend to create our neural networks. More specifically, we will use the **Functional API** of Keras.

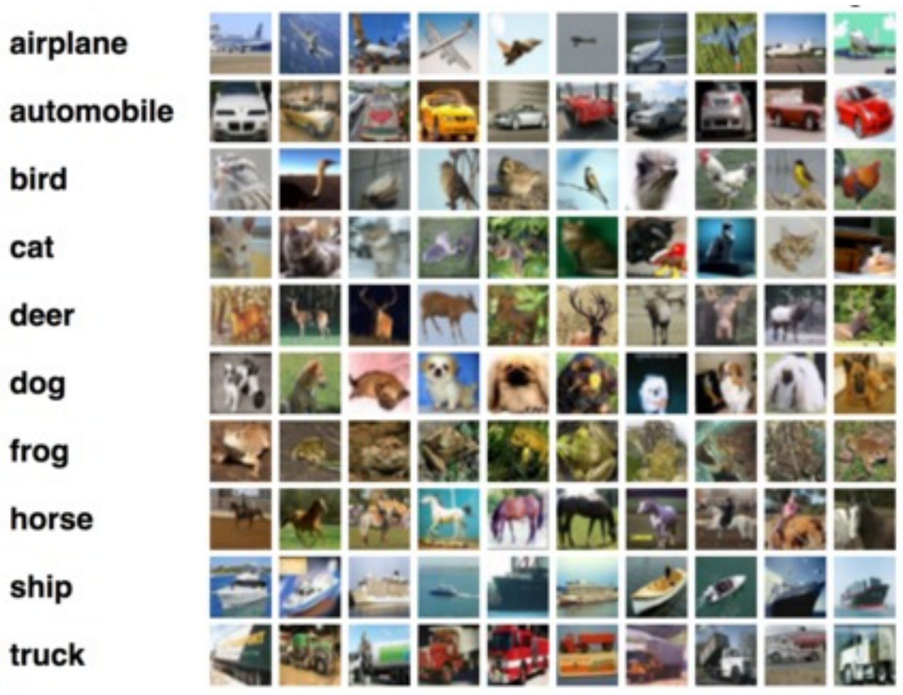

Figure 1: Examples of the different classes of the CIFAR10 dataset.

Deep Learning requires substantial Computing capabilities to be done in a reasonable amount of time. GPUs or purpose-built TPUs excel at the kind of computations that have to be done in Convolutional Neural Networks (CNNs) due to their properties (optimized for parallelized computation, high memory bandwidth). Since we don't expect you to have the required hardware at home, we are going to use Jupyter Notebooks for this exercise which enables executing code on remote machines easily. There are two possibilities to run the code:

- The easiest way for you to implement your solution is to use Google Colab. They provide GPU resources for free which highly reduce training time of neural networks. Additional all libraries you need to use are already installed on their hosted runtime. A Google Colab project is already set up here and you can just create a copy in your Google Drive and start coding in your Browser. To submit your solution, please download the .ipynb file and submit it with your documentation in a .zip file.
- To run the program on your own computer, we provide two conda environments depending on if you have an NVIDIA GPU or not. If you have a NVIDIA GPU you should use this file and without a GPU use this file. You can create your conda environment by calling `conda env create -f mvcr_ex4_gpu.yml` . To activate it call `conda activate mvcr_ex4_gpu` to open your Jupyter Notebook call `jupyter notebook mvcr_exercise4.ipynb` . To see the notebook in your browser navigate to http://localhost:8888. Start coding now!

Normalize Images (1 Point)

Your first task is to create the function `normalize_images` which takes the training images `X_train` and the test images `X_test` as parameters. The outputs should be the two normalized sets with the shape `(dataset_size, 32, 32, 3)` and data type `np.float32` . Normalize in a way, that the returned arrays have approximately zero mean ($\mu = 0$) and unit variance ($\sigma = 1$). You can decide if you want to normalize per color channel or over the whole dataset. Only use the standard numpy functionality to implement this function (e.g. `np.mean()` , `np.var()` , `np.std()` , etc.)

IMPORTANT: Use the same mean and and standard deviation you get from the training set also for the test set, because we want our training and test data to go through the same transformation.

More detailed explanation why we do normalization: Andrew Ng - Normalizing Inputs

Preprocess labels (1 Point)

Keras needs the labels as an one-hot encoded vector for training. The function `preprocess_labels(labels, no_of_classes)` should take a label with shape `(batch_size, 1)` , an integer with the number of classes and create a one-hot encoded numpy array with the shape `(batch_size, number_of_classes)` . Keras has the built-in function `tensorflow.keras.utils.to_categorical` which creates this vector for us. For this exercise, don't use it or similar functions.

Loss Function (1 Point)

We need a loss function to perform the actual training of our model. Implement the cross-entropy loss with the mathematical functions defined in tensorflow.keras.backend. The function `my_crossentropy_loss` should take the input tensor `y_true` with the one-hot-encoded training labels and the predictions of our network `y_pred` which is a tensor of the same dimensions with the softmax prediction values for each class (between 0 and 1). The shape of the input tensors is `(number_of_predictions, number_of_classes)` and the shape of the output tensor should be `(number_of_predictions,)`. You should use the functions provided in the Keras backend e.g. `K.sum()` (URL: Documentation K.sum()). Don't use the built-in crossentropy loss of Keras. For an example check the code snippet below. The cross-entropy loss is defined as

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{k=1}^{K} y_k \log(\hat{y}_k).$$

This video gives a good explanation about the cross-entropy loss:
https://youtu.be/ueO_Ph0Pyqk

Linear Classifier (1 Point)

In a first Step we want to train a simple network with only one layer (no hidden layers). A dense layer takes each input value (each pixel intensity) multiplies it with a weight and adds a bias term for each output neuron. An animation of the process is shown here:
https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#3

To be able to feed the images to the network, we need to reshape them to a vector. The function `vectorize_images(images)` takes as input parameter our images as a `numpy.array` with the shape `(batch_size, height, width, color_channels)` and returns a `numpy.array` with the shape `(batch_size, input_dim)`. You can use the function `np.reshape()`.

To create the model please use the **functional API** of Keras since it is more flexible than the sequential model: https://keras.io/getting-started/functional-api-guide/

We can also have a look at some predictions of our model of the test set. The function `plot_predictions(data, model_name, class_name=None)` plots predictions of of random images of the test set with our model. The input data should be `X_test` preprocessed as the model expects it. By specifying the optional parameter class_name, only correct predictions, false positives and false negatives of this class will be plotted. Feel free to change the parameters below. In the image below you can see an example output of the function with the `class_name='horse'` for a different model.

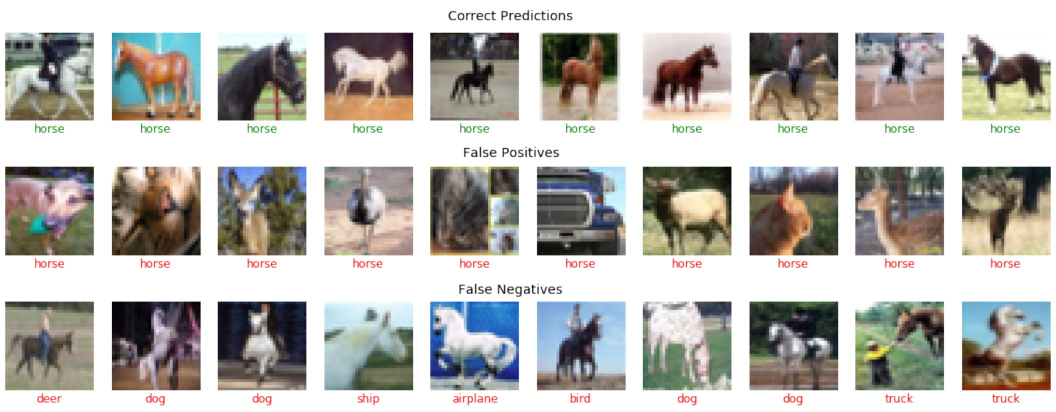

Figure 2: Example prediction results of the linear classifier for the class 'horse'.

Multilayer Perceptron (1 Point)

In our last example with one layer we reached appr. 30% of validation accuracy which is already better than random but not really satisfactory. Now it's time to add the "Deep" to Deep Learning. Create a model with multiple dense layers with ReLu activation and train it again for about 30 Epochs. The exact structure of the model is up to you, a good starting point is a layer with 256 units, another layer with 128 units and the output layer. This model should reach around 50% of accuracy.

MLP with Regularization (1 Point)

To counter the problem of overfitting, you should add measures for regularization. Create a new model based on your MLP from the last point which includes at least one regularization technique. It should improve the validation accuracy slightly.

Convolutional Neural Network (2 Points)

When converting our images to vectors, all spatial information is lost. A convolutional neural network takes matrices (tensors to be more precise) as input and computes features which make use of spatial information. In this exercise implement a neural network as specified in the image below. Add regularization techniques until you reach at least 75% of validation accuracy.

Notes: 3x3 Conv X means Conv2d with kernel 3x3 and 32 units. Maxpool /2 means maxpool with kernel=(2,2) and stride=2.

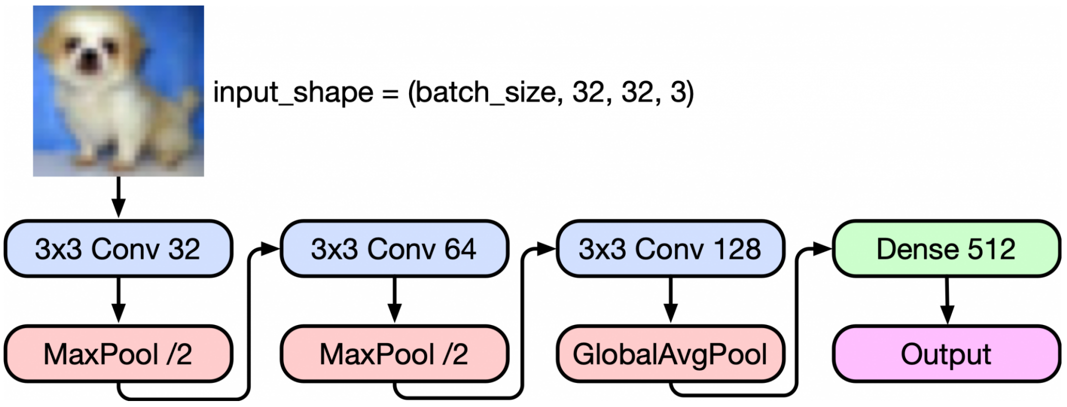

Figure 3: Convolutional neural network Architecture

Finetuning (2 Points)

In the next step we want to use a model which is already pretrained on a large dataset (Imagenet) and finetune it for the CIFAR10 dataset. Keras already provides us with multiple pretrained models (see https://keras.io/applications/). Your task is to take the ResNet50 model and **only** retrain the last layer. You should upsample the images first because ResNet is trained on Images with the resolution of (224, 224). However training with this resolution takes very long with not that much gain. A resolution of (128, 128) is a good compromise between accuracy and training time. You can use the `UpSampling2D` layer of Keras. You should also preprocess the images the same way as ResNet using the function `preprocess_input` from the ResNet50 library. This approach should get you to over 80% of validation accuracy. Training time will take much longer here (~120 seconds per epoch), so we will only train for 5 epochs.

Forbidden functions

Please note that the built-in `tensorflow.keras.utils.to_categorical` function must not be used to solve this exercise.

Remarks

Please only change the code in the dedicated areas. You shouldn't need to change it anywhere else. On the bottom of the file there is a dedicated area, which you can use for experiments for your documentation. You also shouldn't need to use any additional libraries other than the ones already imported. If you need to change the code somewhere else or need to import a different library to get a functioning program, that might be a bug. Please report it in the TUWEL forum or send us a mail to machinevision@acin.tuwien.ac.at

Part 2: Documentation (6 points)

- Answer each numbered question separately, with the answer labelled according to the

experiment and question number. For example, experiment 1 question 2 should be numbered 1.2. You do not need to include the question text.
- Include images to support your answers. These should be referenced in the answers. The document should include information about what parameter settings were used to generate images so that they can be reproduced.
- Your answers should attempt an explanation of why an effect occurs, not just describe what you see in the supporting images.
- Be precise. "It is better" or similarly vague statements without any additional information are not sufficient.

Points may be deducted if your report does not satisfy the above points.

Maximum 1500 words, arbitrary number of pages for result images (referenced in the text). Violating the text limit will lead to point deductions!

Experiment 1 (1 Points)

1. Include all plots of the training history of all models (`linear_classifier` , `mlp_model` , `mlpreg_model` , `cnn_model` , `resnet_model`). Write a short description for all of them with potential problems. (1 Point)

Experiment 2 (2 Points)

1. Retrain the Multilayer Perceptron with Regularization `mlpreg_model` with a stochastic gradient descent optimizer with a training rate of 0.1. To reset the model, load the weights you saved before training as `'mlpreg_model_weights.h5'` . Afterwards compile the model again with the new optimizers. Use `optimizers.SGD()` to create the optimizer. Describe the training process. (0.5 Points)
2. Now train again with a SGD optimizer with a learning_rate of 0.001. Don't forget to reset the weights with `mlpreg_model.load_weights(mlpreg).h5` . How did the training process change? What is the result compared to the higher training rate? (0.5 Points)
3. The Adam optimizer we used for the CNN is an optimizer with an adaptive learning rate. Explain the benefits of a learning rate decay. Did your experiments support this idea? (You don't have to look into the Adam optimizer, just explain the general idea of learning rate decay.) (1 Point)

Experiment 3 (2 Points)

1. Try your different models with different input images. You can use the function `plot_predictions(data, model_name, class_name=None)` to get predictions of random images of the test set with a specific model. The input data

should be `X_test` preprocessed as the model expects it. By specifying the optional parameter class_name, only correct predictions, false positives and false negatives of this class will be plotted. You can also use the function `load_image_link()` or `load_image_colab` to open an image provided by url or uploaded to Google Colab. You need to preprocess the image the way your model expects it and can then get the Top-5 predictions by using `predict_image(nn_model, image)`. Pick 3 example images with predictions of different models and add a short description why you think the prediction is correct or incorrect. (1 point)
2. Pick one class and use `plot_predictions` with the linear_classifier and try to describe why some images are predicted correctly while others are not. Why does the convolutional neural network work better in general. Some tipps can be found here: http://cs231n.github.io/linear-classify/ (1 point)

Experiment 4 (1 Point)

1. Calculate the receptive field of each convolutional layer for the CNN model you implemented. Also put how you derived these numbers in your documentation. (1 Point)

You must specify the parameters used in all of your screenshots.

Assistance

Please keep to the following hierarchy whenever there are questions or problems:

1. Forum: Use the TUWEL forum to discuss your problems.
2. Email for questions or in-person help requests: machinevision@acin.tuwien.ac.at

Upload

Please upload the Jupyter Notebook file **mvcr_ex4.ipynb**, as well as your documentation (pdf-format) **as one ZIP-file** via TUWEL.