**Deep Learning**

**Graded Assignment 2:**

**CIFAR 100 (with 20 classes)**

**Group 08**

Bhawna Dixit (02014095)

Thomas Kok (02013791)

Emiel Maes (01308305)

**Table of Contents**

[**Problem description**](#_7b990umyo8op) **2**

[**Unregularized model**](#_yihl3qterlym) **4**

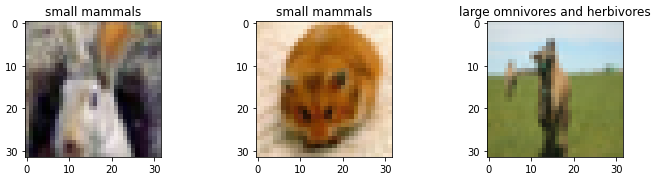
[**Regularization**](#_cmi0jrr46tfs) **11**

[**Augmentation**](#_evmnimhesh7v) **21**

[**Model evaluation and analysis**](#_4twnimjdq8dm) **26**

# Problem description

The goal is to classify the CIFAR 100 dataset which consists of labeled pictures. The dataset has both ‘fine labels’ (100 classes) and ‘coarse labels’ (20 classes). Here, the ‘coarse’ classification task is considered in which each picture has to be classified into 1 out of the 20 classes. Some examples of the dataset are visualized in figure 1. The dataset has a training set of 50,000 32x32 colour images and a test set of 10,000 32x32 colour images. During the model training, the training set of 50.000 samples is split into 44,000 samples for training and 6,000 samples for validation. It must also be noted that each class has the same amount of samples which means that this is a balanced dataset.



*Figure 1: Samples from the data*

The objective of the second assignment is to train a convolutional neural network which is able to classify the images well (validation accuracy ⩾ 72%). However, the focus is to really understand how each part of the architecture (type of layer, number of neurons, batchnorm, batch size, learning rate, regularization technique..) affects the performance of the model. Therefore, the assignment is split into several steps which are defined as follows:

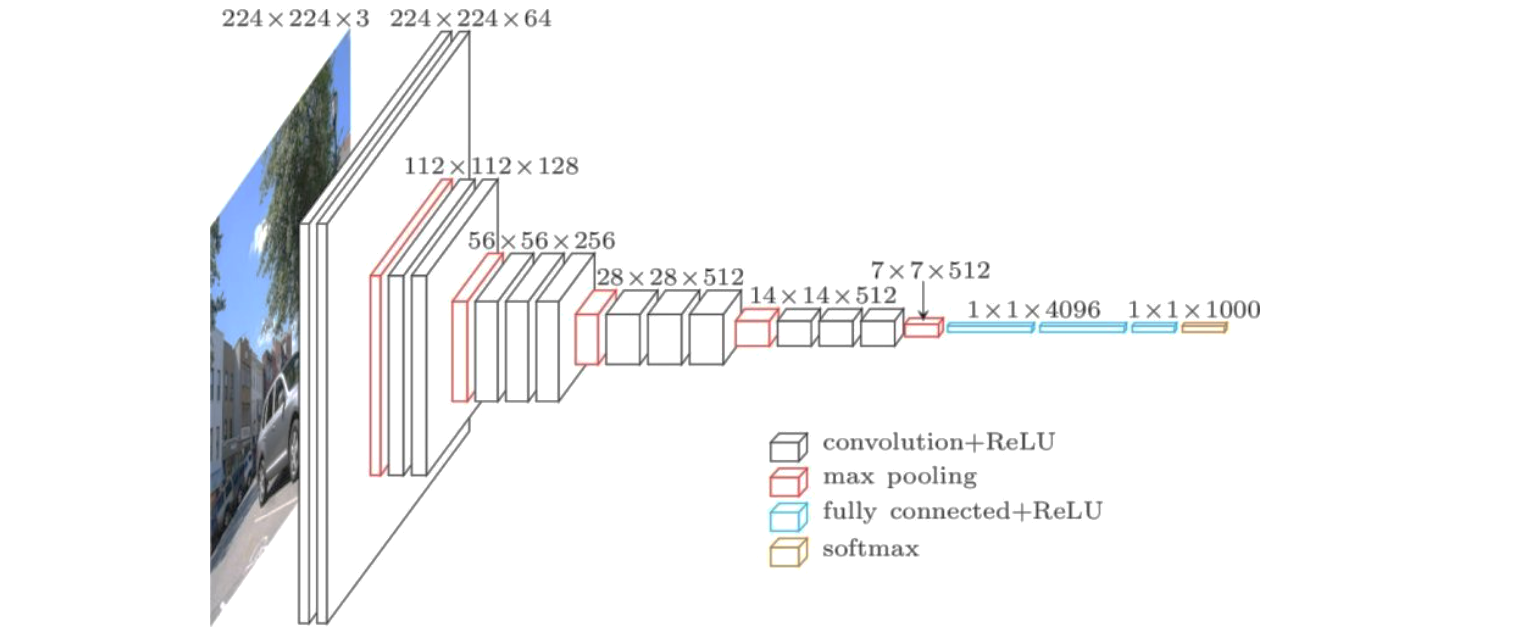
* The first step is to create an initial model which is powerful enough to classify the training samples. Typically, some overfitting will occur and there will be a gap between the training and validation accuracy. In order to find the optimal architecture, each of the hyperparameters are changed individually, while fixing the values for all the other hyperparameters. Once all the different hyperparameters are considered, it is possible to select the best performing model and continue to the second step.
* The second step is to optimize the validation performance and close the gap between the training and validation accuracy. This is typically done by using regularization techniques. In the second step of the assignment, several regularization techniques such as L1, L2, maxnorm and dropout layers are investigated. Here, again each hyperparameter is being tuned individually while keeping the other hyperparameters fixed.
* Once the gap between the training and validation accuracy is smaller than in the first step, it is still possible to evaluate the impact of data augmentation. In this third step, each image is augmented which means that certain modifications (translation, rotation, zoom, shear..) are applied to the image such that the training data size is increased. This should result in a better performance of the model as more data is available.
* The final step is then to train the regularized model on the whole training set (the 50,000 samples), along with the augmentation technique and then check the performance on the unseen test data. The next sections will discuss these steps more in detail and give an overview of the methodologies that were used.

# 

# Unregularized model

In order to not start with a completely random model, some prior research was done to determine an initial model architecture. Based on the VGG16 paper[[1]](#footnote-0). It is possible to note that this assignment deals with a somewhat less complex task than the one described in the VGG16 paper where they have 500.000 training samples and 200 classes while in this assignment, 50.000 training samples are available to be classified into 20 classes. Therefore, it was decided to start with the first 2 “block structure” layers from the VGG16 paper and add a fully connected dense net of 512 neurons, followed by an output layer of 20 neurons as there are 20 classes to be classified.

Based on the initial notebook and several other blogs[[2]](#footnote-1), initial values for the other hyperparameters were chosen. An initial batch size of 128, the number of epochs was set to 30 and the Adam optimizer was used with its initial learning rate set to 0.001 (default value). However, these are just some initial values and will be further tuned in the process.



*Figure 1: VGG16 Network architecture[[3]](#footnote-2)*

The initial architecture that was used looks as follows:

* Block 1: 3x3 Conv2D 64 - Relu - 3x3 Conv2D 64 - Relu - Maxpool 2x2
* Block 2: 3x3 Conv2D 128 - Relu - 3x3 Conv2D 128 - Relu - Maxpool 2x2
* Flatten layer
* Dense layer with 512 neurons
* Dense layer with 20 neurons

The table below describes the tuning process in which the changes to the current best model architecture and hyperparameters are described during each step. Next to that, the optimal architecture that was obtained at the end of each step is briefly described how it relates to the previous step. Next to that, the train and validation accuracies, along with their validation curves are given for the best obtained model when each step is completed (and when the model does differ from the previous step).

*Table 1: Tuning process for unregularized model*

| **Step** | **Model architecture or hyperparameter which is being tuned** | **Optimal architecture at the end of this step** |
| --- | --- | --- |
| 0 | Starting model which is the  initial architecture based on first 2 blocks of the VGG net (nothing is tuned here) | The **initial model architecture** that was described above. This model gives the following performance which already looks quite good for an initial model. |
| 1 | Check performance when adding Batchnorm Layers in the blocks (before or after relu function) | **Same architecture as the one in step 0, but now with batchnorm layers before the relu activation function.** As can be seen, the performance on the training accuracy drops while the validation accuracy increases. This is mainly due to the fact that batchnorm also adds some regularization to the model. It was decided to continue with the batchnorm layers as the power of the model can still be increased later on by adding more channels or more blocks. |
| 2 | Defining the number of blocks | The last block which was defined as “Block 2” in the initial model (the one with 128 channels) is copied. Also a batchnorm layer is added to the block before the relu function (based on the previous step). Such a block was added once and twice to the current architecture. This seems to improve the performance of the model as the training accuracy increases compared to the previous obtained training accuracy. **The optimal number of blocks to add was 1 block with 128 channels which gave the best train/test accuracy of 95%/64%.** When adding 2 blocks, this accuracy decreased a little to 93% and 63%. Removing such blocks from the initial architecture seems to decrease the performance. |
| 3 | Defining the number of layers per block | For this step, we try to see if the number of Conv2D/BatchNorm/ReLu layers in each block is appropriate. The number of such layers was 2 in step 2, we tried 1 and 3 here, as well as additional configurations with increased/decreased number of blocks to compensate for the changed number of layers. **None of the configurations improved upon the 2 layer-block**. This means that the optimal architecture after this step is still the same as the one obtained from the previous step. |
| 4 | Number of channels per block | At the moment, we have three blocks with corresponding number of channels in each block: 64, 128, 128. We will keep the non-decreasing (because this makes most sense with CNNs conceptually) number of channels, but try several different configurations (different number of channels within a block and different number of channels over the blocks). **Empirically, 64 -> 128 -> 512 got the best results and also the most impressive training curves**. This architecture obtained a training accuracy of 98%. |
| 5 | Kernel size | Several different kernel sizes were used in the ‘blocks’ ranging from 2x2 to 5x5. Also a combination of 5x5 in the first block and 3x3 in the following blocks was inspected but none of them seemed to improve the accuracy from the current model that was the best so far. We concluded to **continue with the already existing 3x3 kernel sizes**. |
| 6 | Pooling size and type | Max pooling is better able to extract the most important features like edges while average pooling extracts more smooth features. Therefore, we expect that the MaxPooling2D will result in higher performance. However, we also looked at AveragePooling2D and our initial thoughts got confirmed when inspecting the validation plots. Next to the pooling types, we also considered the pooling sizes (2x2),(3x3) and (5x5). However, the **(3x3) pooling size still showed the best result** and therefore **no changes were made to the current model**. |
| 7 | Number/neurons of dense layers | Here, we tried several different architectures (both wider and deeper) in the dense layers. Some of the architectures we tried look as follows: 1024->512->20;  512->256->128->20;  256->20 but none of them seemed to improve the existing performance. Next to that, also Global Average Pooling (and Global Max Pooling) as alternatives to the fully connected layers were considered but these did not seem to improve the performance any further. Therefore, we chose to **continue with the existing model architecture.** |
| 8 | Initial learning rate and batch size | Several values were used for the initial learning rate, starting from the default rate which is 0.001, we also considered 0.0005, 0.00025 and 0.00001. However, by changing the initial learning rate, still some fluctuations in the training accuracies could be seen on our plots. Therefore, we had to iterate back and forth between adjusting the initial learning rate and the batch size. Batch sizes that were considered are 128, 256, 512 and 1024.The **learning rates 0.0005 and 0.00025 seemed to give similar performance in combination with a batch size of 512**. As they performed similarly, **we chose to continue with the 0.00025 learning rate.** |

The **final unregularized model architecture** looks as follows:

* Block 1:
  + 3x3 Conv2D 64 -> Batchnorm -> Relu
  + 3x3 Conv2D 64 -> Batchnorm -> Relu
  + Maxpool 2x2
* Block 2:
  + 3x3 Conv2D 128 -> Batchnorm -> Relu
  + 3x3 Conv2D 128 -> Batchnorm -> Relu
  + Maxpool 2x2
* Block 3:
  + 3x3 Conv2D 512 -> Batchnorm -> Relu
  + 3x3 Conv2D 512 -> Batchnorm -> Relu
  + Maxpool 2x2
* Flatten layer
* Dense layer with 512 neurons
* Dense layer with 20 neurons
* Softmax

And has the following hyperparameters:

* Batch size of 512
* Initial learning rate of 0.00025
* Number of epochs: 30

The validation curves for the corresponding best performing model are visualized in step 8. As can be seen, the model is highly overfitting on the training data as it reaches 100% accuracy. It also has a very steep curve as can be seen from the training accuracy. Other than that, it also seems that the batch size and learning rate are appropriate as there are no more fluctuations. The next step is now to regularize the model and close the gap between the training and validation accuracy which will be discussed in the next section.

# Regularization

During the regularization phase, we use early stopping with the patience hyperparameter set to 15 and the min\_delta hyperparameter set to 0.0001. When running a certain model, we also make sure that the training is not stopped when the performance is still improving and therefore the number of epochs was increased to 150.

**Conclusions**

The results of the best tuning results are summarized in the table below (Table 2). Further, the tuning steps for these hyperparameters are separately summarized in tables subsequently for reference (Table 3, 4, 5).

*Table 2: Tuning process for regularized model*

| **Step** | **Model architecture or hyperparameter which is being tuned** | **Optimal architecture at the end of this step** |
| --- | --- | --- |
| 1 | Dropout | * Based on previous research, initially the default dropout (p=0.5) was used only after the dense layer (512) in the network, however, the model performed poorly as the training set was overfitting. * To overcome the overfitting from the previous model, 3 additional dropout layers were applied at convolutional layers. The input layer was set at p=0.2 as dropping out more input data can adversely affect the training. The other two dropout layers were added at consecutive convolutional blocks (p=0.3, p=0.4). * The dropout was increased proportionally based on the number of neurons. The validation accuracy significantly improved from 0.56 to 0.69. These results were in line with the findings of Srivastava et al. 2014[[4]](#footnote-3), that application of dropout to convolutional layers aided in generalization in CIFAR-10 dataset. * However, the validation accuracy didn’t improve as intended, therefore other variations of dropout were tried that are listed in the dropout table separately. It was observed that validation accuracy didn’t improve for the variations where the same dropout was applied throughout the network. * The best model with optimal dropouts reached to 71% validation accuracy which is shown below. |
| 2 | MaxNorm | MaxNorm constraint ensures that the weights of the network remain small. This is useful to apply in our model as we need to have more regularization because using only dropout still produces overfitting to some extent. Typical values for the max norm hyperparameter are on orders of 3 or 4[[5]](#footnote-4). However, after some experimentation we found that for our case, this was not the most appropriate value. A final value of 1 for the MaxNorm constraint was chosen with the best results. |
| 3 | L1\_L2 regularization | For additional regularization, we also considered L1 and L2 regularization. This allows for some complex additional regularization on top of MaxNorm. We considered values in the standard range of 1 to the negative powers, with a focus on lower values (as we already have some regularization).  Therefore, in final steps, we applied further penalization for overfitting with L1\_L2 regularization, the best decay values were found to be 1e-6 and 1e-3 respectively. |

*Table 3: Tuning process for regularized model with dropout*

| **Model** | **Amount of dropout that was used** | **Performance of the corresponding model** |
| --- | --- | --- |
| 1 | A single dropout layer (p=0.5 default value) was added to the model at the dense layer. |  |
| 2 | In addition to the above layer, three dropout layers were added at each convolution block randomly starting from low dropout (p=0.2) to increasing it as the number of neurons increased subsequently in the layers.  p=0.2 (Conv2D, 64)  p=0.3 (Conv2D, 128)  p=0.4 (Conv2D, 512)  p=0.5 (Dense, 512) |  |
| 3 | From the above layers the dropout layer at the dense layer was removed and the dropout layers at convolution layers were kept. |  |
| 4 | The dropout layers were kept the same for all the 4 dropout layers (p=0.2). |  |
| 5 | From the above regularization model 2, the value of the last dropout layer was decreased to 0.4.  p=0.2 (Conv2D, 64)  p=0.3 (Conv2D, 128)  p=0.4 (Conv2D, 512)  p=0.4 (Dense, 512) |  |
| 6 | The dropouts were increased from the above model.  p=0.3 (Conv2D, 64)  p=0.4 (Conv2D, 128)  p=0.5 (Conv2D, 512)  p=0.5 (Dense, 512) | Overfitting significantly reduced in the model, however, val accuracy is currently at 71%. |
| 7 | The dropouts were increased from the above model.  p=0.3 (Conv2D, 64)  p=0.4 (Conv2D, 128)  p=0.6 (Conv2D, 512)  p=0.6 (Dense, 512) | The overfitting on training data reduces as the last two dropout layers are increased. This will be the best performing model which will be used for adding the next regularization techniques. |
| 8 | The dropouts were increased from the above model. | Adding even higher values for the dropout results in a slight decrease in performance which could mean that too high values are used for the dropout. |

*Table 4: Tuning process for regularized model with Maxnorm*

| **Model** | **Maxnorm** | **Performance of the corresponding model** |
| --- | --- | --- |
| 1 | Maxnorm of 2 in each Conv2D and Dense layer (together with the dropout layers) |  |
| 2 | Maxnorm of 4 in each of the Conv2D layers and maxnorm of 2 in the Dense layer |  |
| 3 | Maxnorm of 1 in each of the Conv2D and Dense layer | Performance with the best model when using the best model after applying early stopping. |
| 4 | Maxnorm of 3 in each of the Conv2D layers and Maxnorm of 3 in the Dense layer |  |

*Table 5: Tuning process for regularized model with L1 and L2*

| **Model** | **L1 and L2** | **Performance of the corresponding model** |
| --- | --- | --- |
| 1 | l1 = 1e-6  l2 = 1e-4 |  |
| 2 | l1 = 1e-6  l2 = 1e-3 |  |
| 3 | l1 = 1e-5  l2 = 1e-2 |  |
| 4 | l2 = 1e-4 |  |

The **final regularized model architecture** looks as follows:

* Block 1:
  + 3x3 Conv2D 64 (maxnorm=1, L1= 1e-6, L2=1e-3) -> Batchnorm -> Relu
  + 3x3 Conv2D 64 (maxnorm=1, L1= 1e-6, L2=1e-3)-> Batchnorm -> Relu
  + Maxpool 2x2
  + Dropout 0.3
* Block 2:
  + 3x3 Conv2D 128 (maxnorm=1, L1= 1e-6, L2=1e-3)-> Batchnorm -> Relu
  + 3x3 Conv2D 128 (maxnorm=1, L1= 1e-6, L2=1e-3)-> Batchnorm -> Relu
  + Maxpool 2x2
  + Dropout 0.4
* Block 3:
  + 3x3 Conv2D 512 (maxnorm=1, L1= 1e-6, L2=1e-3)-> Batchnorm -> Relu
  + 3x3 Conv2D 512 (maxnorm=1, L1= 1e-6, L2=1e-3)-> Batchnorm -> Relu
  + Maxpool 2x2
  + Dropout 0.6
* Flatten layer
* Dense layer 512 (maxnorm=1, L1= 1e-6, L2=1e-3)
* Dropout 0.6
* Dense layer 20
* Softmax layer

And has the following hyperparameters:

* Batch size of 512
* Initial learning rate of 0.00025
* Number of epochs: 150 (with early stopping)

When reconstructing the best model encountered during training (was achieved after training for 112 epochs), it is possible to use this model to check the final performance of the model on the validation set. When doing so, we obtained a training accuracy of 93.81% and a validation accuracy of 72.92%. The next section will now look if it is possible to even increase this validation accuracy when using more data by synthetically generating samples.

# Augmentation

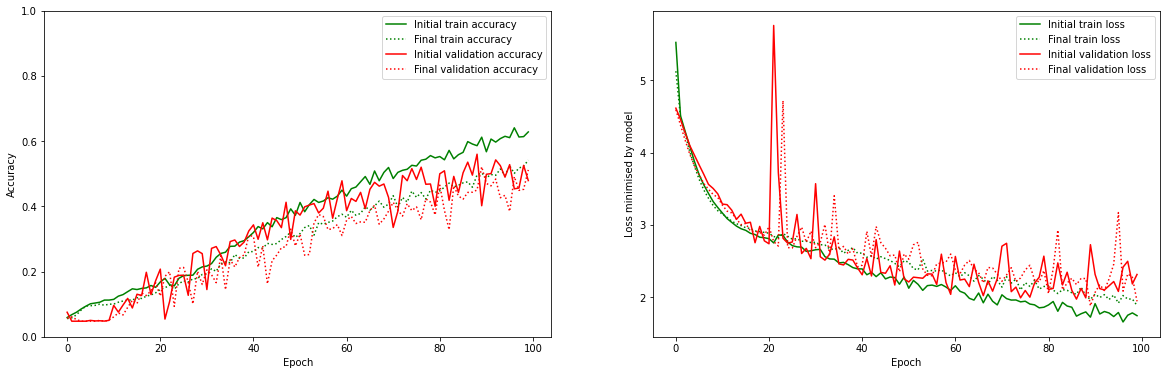
In order to obtain even better results, an approach could be to obtain more data. However, when data is rather limited, another idea is to generate synthetic data which can be done through data augmentation. When using augmentation techniques, new data (synthetic pictures) is generated ‘on the fly’ when the training batches are read. Before the training samples are fed to the model, the sample is augmented by performing several transformations[[6]](#footnote-5). Transformations that were considered for this assignment relate to the following ones:

* **rotation\_range** which randomly rotates images in a predefined range (0 to 180 degrees)
* **width\_shift\_range** randomly shifts the images horizontally based on a fraction of the total picture width
* **height\_shift\_range** randomly shifts the images horizontally based on a fraction of the total picture height
* **horizontal\_flip** randomly flips the image horizontally
* **vertical\_flip** randomly flips the image vertically
* **fill\_mode** tells what happens with ‘new’ pixels that occur due to a certain transformation (for example when the image is shifted to the right, it means that new pixels are created on the left of the image and a technique is needed to fill these pixels)
* **shear\_range** the shear angle which is applied in counter-clockwise direction (in degrees)
* **zoom\_range** defines the range for the random zoom

As mentioned in the notebook, an example image was taken and different values for the augmentation technique were used such that the new created images still looked realistic.

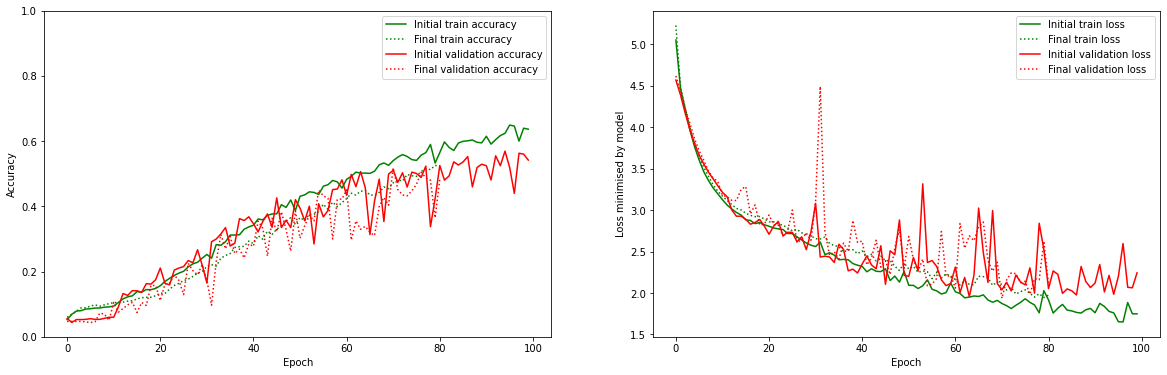
We found the best options for augmentation by evaluating the options in a sequential manner, to check if there is an improvement in the results. For configuring the augmentation, to save time we use the x\_train\_small dataset instead of the full x\_train dataset. In combination with using the generator, this gives some worse results for the model but because we are only comparing relative results we can still use these results to determine the best configuration.

For the first step, we considered the shift transformation, in both width and height. We set a value of 0.2, and compare the results with the non-augmented data.



*The training and validation curves for the unaugmented data (initial accuracy) and shifted data (final accuracy), with shift 0.2.*

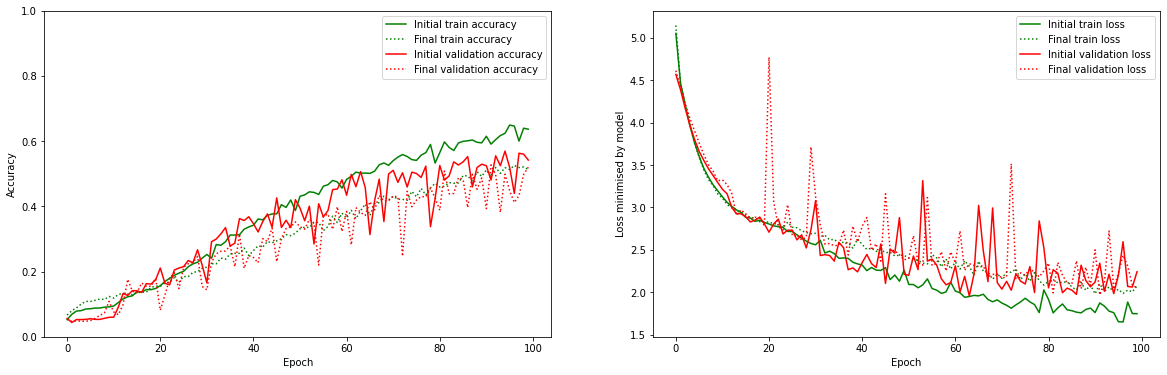
These results don’t show a considerable improvement, so perhaps the shift of 0.2 was too major and results in vague data at the edges of the images, and as such we tried again with a shift of 0.1 both in width and height.



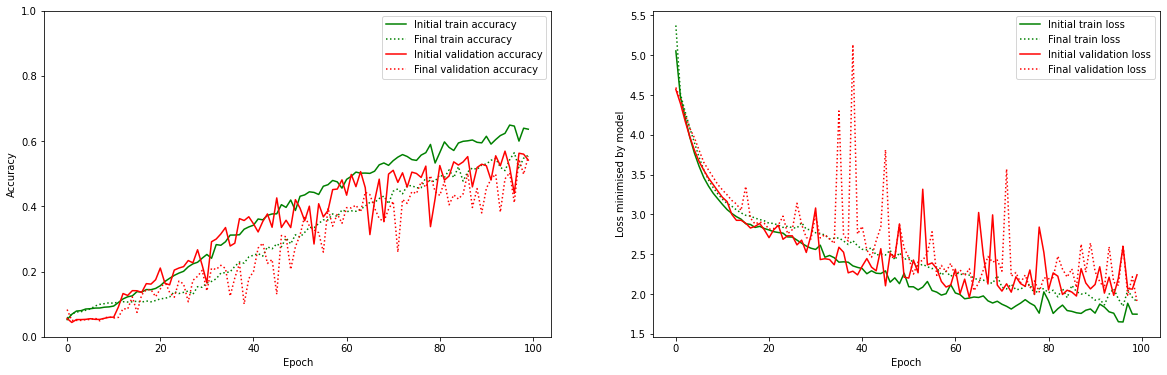
*The training and validation curves for the unaugmented data (initial accuracy) and shifted data (final accuracy), with shift 0.1.*

For a smaller shift, the results seem again similar to the unaugmented results, but we can not see an improvement. Therefore, we will not be using shift augmentation.

The next option we consider is rotation augmentation. With this, the images will be rotated slightly, hopefully generating new interesting images. First, we consider a rotation max degree of 30 degrees. This is based on the fact that visual inspection shows that higher rotation degrees can create nonsensical images, so this value should not be too high. Too low of a value might not provide much information though. We will thus try two different values: 30 degrees and 15 degrees.



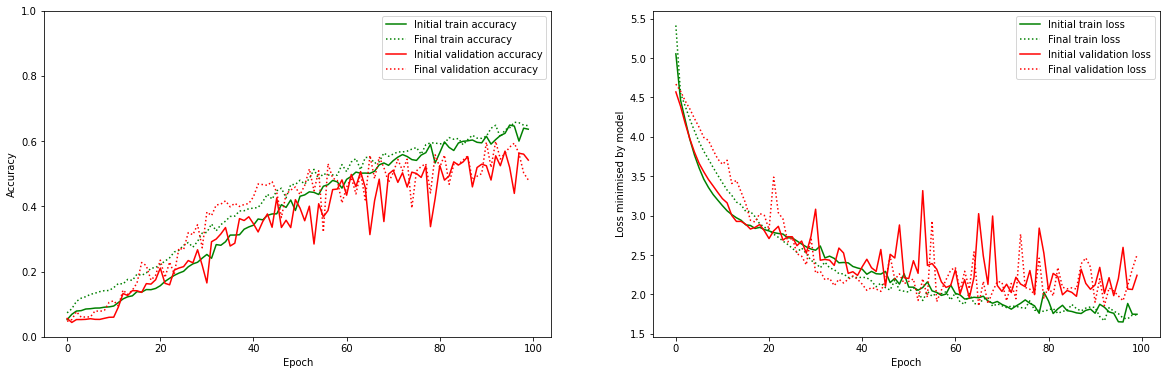
*The training and validation curves for the unaugmented data (initial accuracy) and rotated data (final accuracy), with 30 rotation degrees.*



*The training and validation curves for the unaugmented data (initial accuracy) and rotated data (final accuracy), with 15 rotation degrees.*

The rotation does not seem to improve upon the baseline either, with the validation results ever so slightly worse throughout the validation curve.

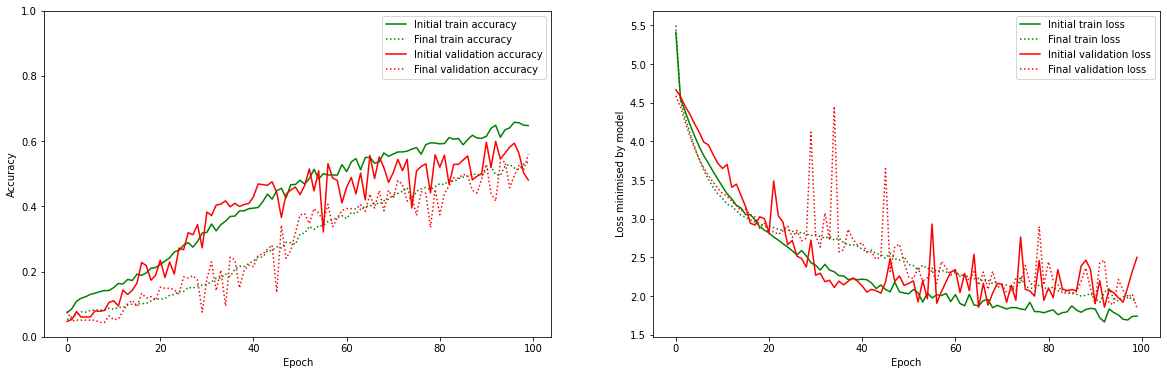
Next, we tried flipping. This seems to be quite a straightforward transformation, flipping the image either horizontally or vertically. With a quick visual explanation, we can see that flipping images vertically generates images that do not make as much sense. As such, we only use horizontal flipping.



*The training and validation curves for the unaugmented data (initial accuracy) and horizontally flipped data (final accuracy).*

The horizontally flipped augmentation actually gives a slight improvement over the original unaugmented data, throughout both the training and the validation curve. For this reason, we will use horizontal flipping as augmentation.

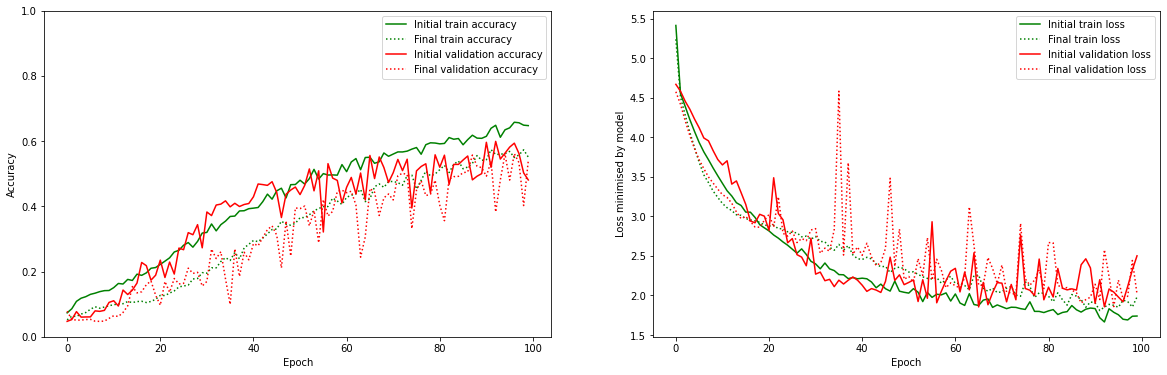
Next, we consider zooming as augmentation. This zooms in on the image, essentially cropping it. We considered 0.2 to be a fair value, providing images that look similar to the original and zoomed in not too much.



*The training and validation curves for the horizontally flipped data (initial accuracy) and horizontally flipped + zoomed in data (final accuracy).*

The zooming in on the data actually gives a noticeable negative difference in performance, it does not seem to help at all. For this reason, we do not use zooming as augmentation. Logically, this might make sense as zooming in on the figure only removes information.

Next, the final augmentation technique we considered is shear transformation. Shear mapping is a linear map that displaces each point in a fixed direction, sort of like a rectangular to parallellogram transformation. Because this is a pretty distorting transformation, from visual inspection, we used a maximum of 15 degrees.

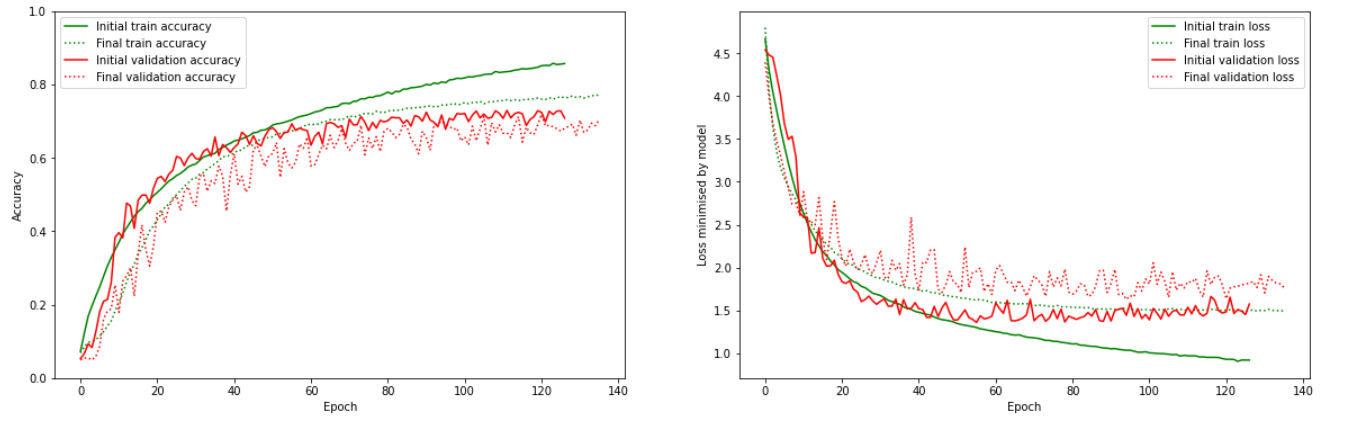


*The training and validation curves for the horizontally flipped data (initial accuracy) and horizontally flipped + shear transformed data (final accuracy).*

This did not improve the results either, and we thus do not use shear transformation for augmentation.

In the end, the only augmentation strategy we used is horizontal flipping. Reflecting on the results, we are not certain that this is the only one that is truly effective. Based on literature and previous experiences, several of the augmentation techniques that failed here work well in general. We consider the possibility that the data we used here is not representative: perhaps the data here is not too limited and the augmentation only introduces distortions. However, we simply do not have enough time to explore this much further, and we will continue with the horizontal flipping augmentation.

Final result : Model trained with x\_train so 44.000 samples and data augmentation. This best performing model obtained a train set accuracy of 78.00% and val set accuracy of 71.85.



# Model evaluation and analysis

To test the performance of our model architecture, we re-trained the final regularized model on the full training data (the 50,000 training samples) and did not split up the data into train and validation sets. Once the model was trained on this data, we then evaluated it on the test set which was never seen before. The regularized model performed very well and achieved an overall accuracy of 74.12% on this test set. We refer to this model as the **regularized model (no data augmentation)**.

The same process as above was repeated again, but now also with data augmentation and its corresponding optimal data augmentation hyperparameter values. The regularized model with data augmentation performed slightly worse and achieved an overall accuracy on the test set of 67.27%. We refer to this model as the **regularized model with data augmentation**.

*Table 6: High level summary about the differences between the two models*

| **Regularized model (no data augmentation)** | **Regularized model with data augmentation** |
| --- | --- |
| 94.94% accuracy on training set | 74.12% accuracy on training set |
| 74.11 % accuracy on test set | 67.27% accuracy on test set |
| 2589 misclassified samples | 3273 Misclassified samples |

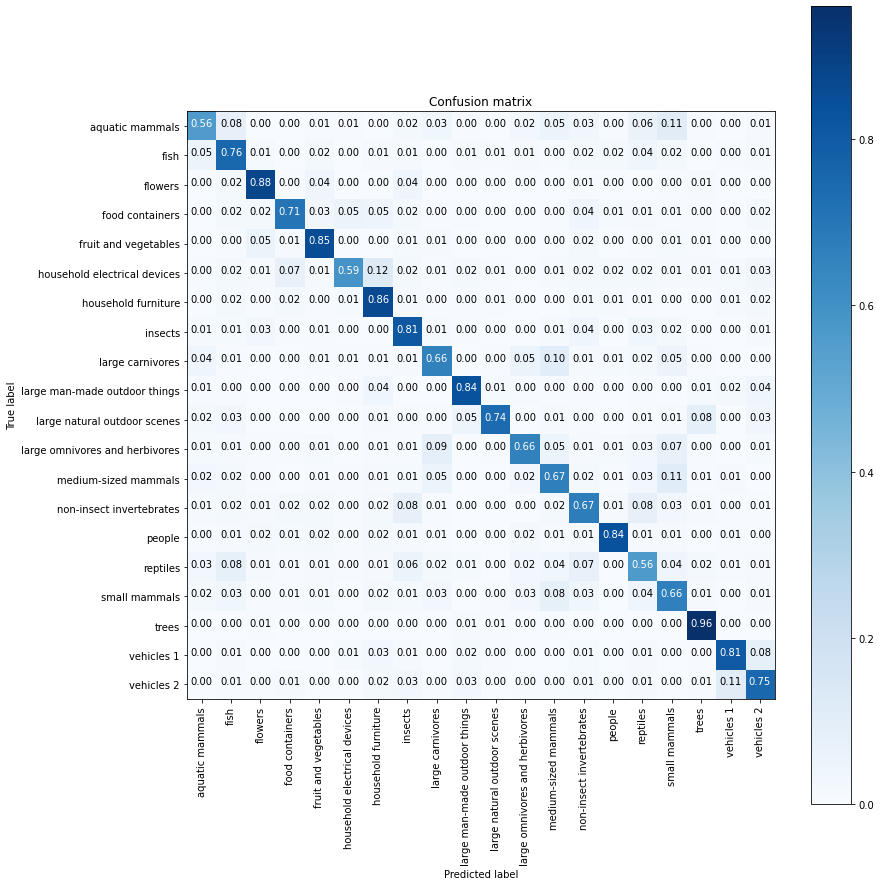
The next step is now to look more into detail on where both models make certain mistakes. First, the confusion matrix is considered which shows us how well each model is at predicting a certain class and where it makes certain mistakes. Once we have an idea where the model makes certain mistakes, we can look at specific misclassified samples and see if we can understand the mistakes by the model.

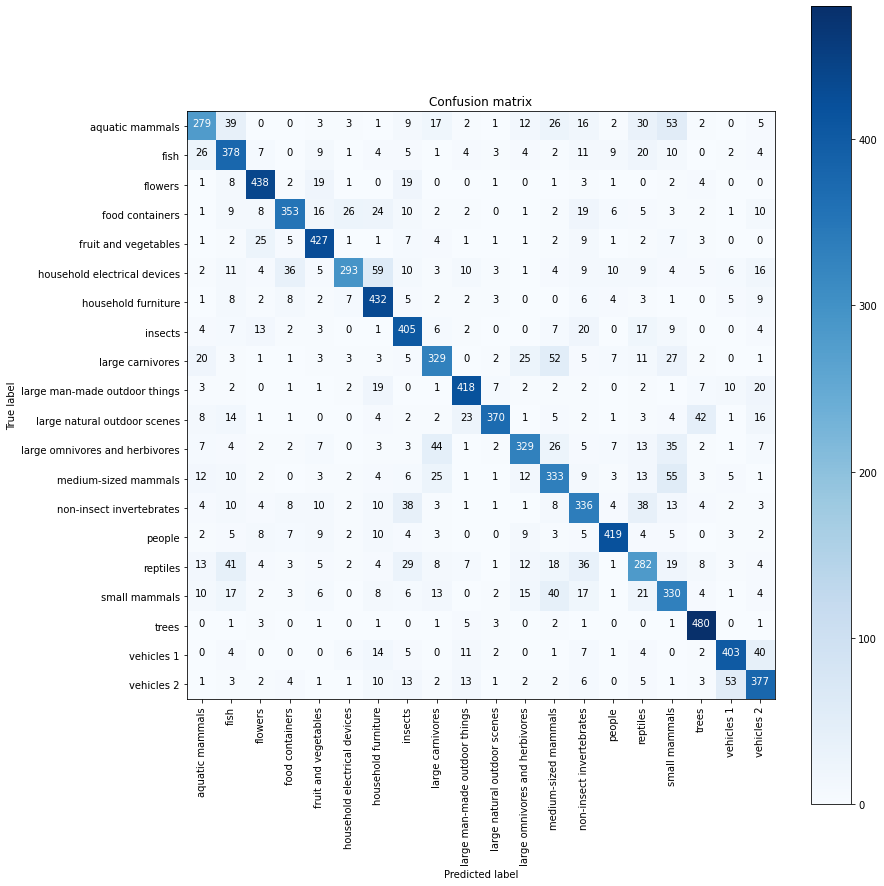
**Analysis for the regularized model (no data augmentation)**

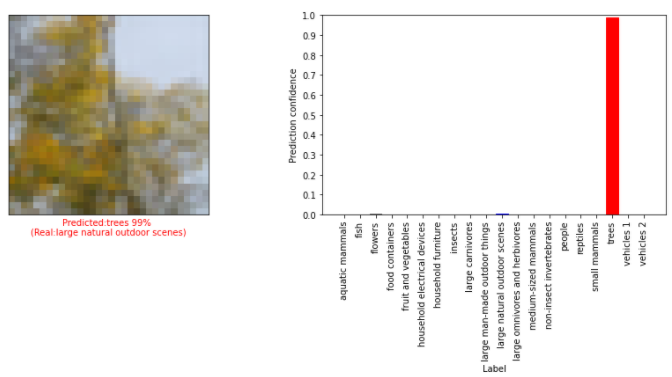
When looking at the confusion matrix for the regularized model without data augmentation, the following conclusions can be drawn:

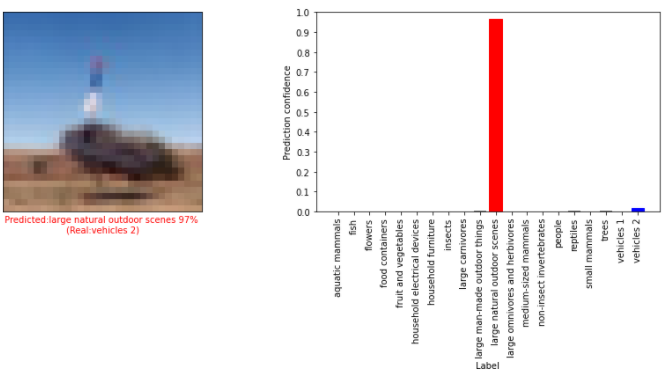
* In general, the model seems to be able to classify some classes very well where it achieves 75% or more. These classes relate to fish, flowers, fruit and vegetables, household furniture, insects, large man-made outdoor things, people, trees, vehicles 1 and vehicles 2.
* The model seems to perform moderate (65% to 75%) on the classes food containers, large carnivores, large natural outdoor scenes, large omnivores and herbivores, medium-sized mammals, non insect invertebrates and small mammals.
* The model seems to be very bad (<65%) at classifying aquatic mammals, household electrical devices and reptiles.

Below, the two (normalized and denormalized) confusion matrices are given on which our conclusions are based

****



When looking more in detail about what examples get misclassified, it is possible to see that some examples are even very hard for us as humans to classify in one of these 20 classes. Two such misclassified samples are visualized below in which the red bar shows the predicted label. As can be seen from the first picture, it is difficult to see that this example belongs to the real-large natural outdoor scenes. The same is valid for the other figure where the model predicts this to be part of the large natural outdoor scenes while it belongs to vehicles 2.

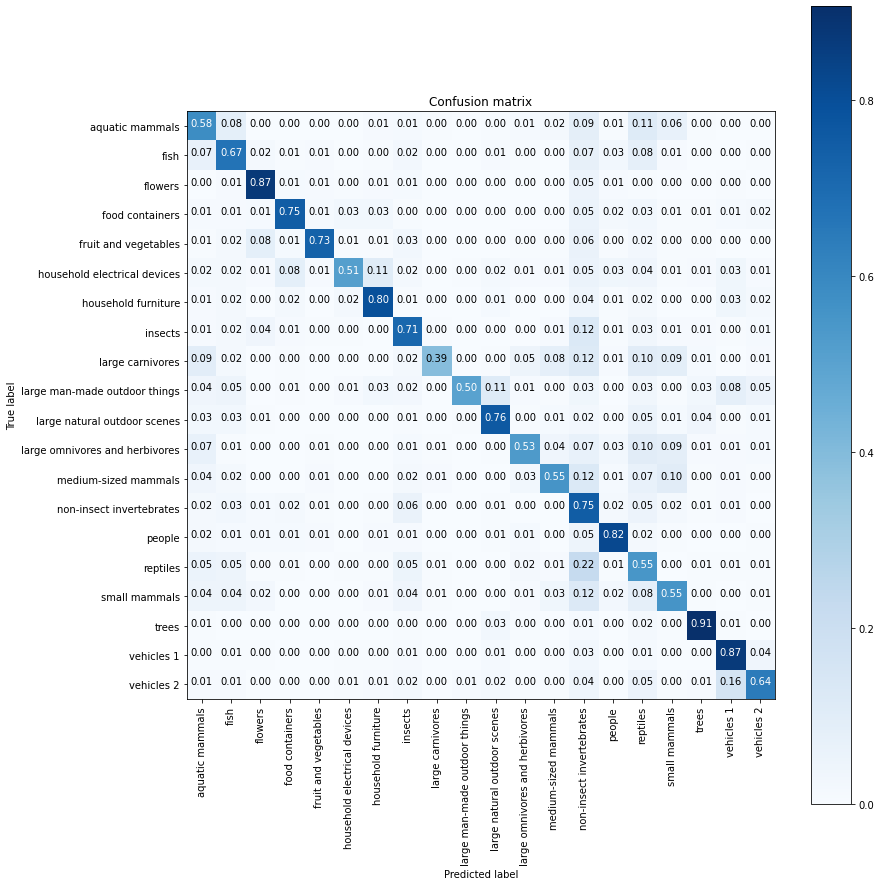


**Analysis for the regularized model with data augmentation**

When looking at the confusion matrix for the regularized model with data augmentation, the following conclusions can be drawn:

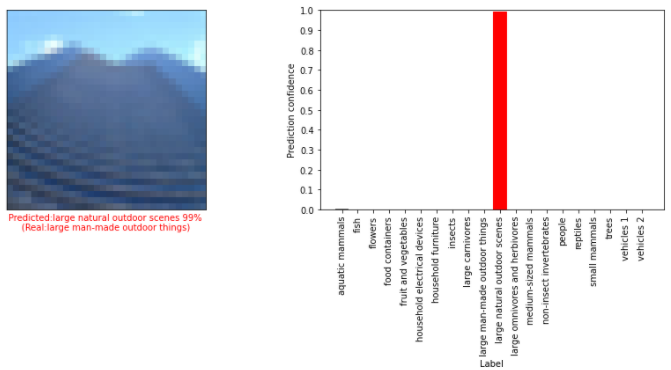
* In general, the model seems to be able to classify some classes very well where it achieves 75% or more. These classes relate to flowers, food containers, household furniture, large natural outdoor scenes, non-insect invertebrates, people, trees and vehicles 1
* The model seems to perform moderate (65% to 75%) on the classes fish, fruit and vegetables, insects and vehicles 2.
* The model seems to be very bad (<65%) at classifying aquatic mammals, household electrical devices and reptiles, large carnivores, large man-made outdoor things, large omnivores and herbivores, medium-sized mammals, reptiles and small mammals.

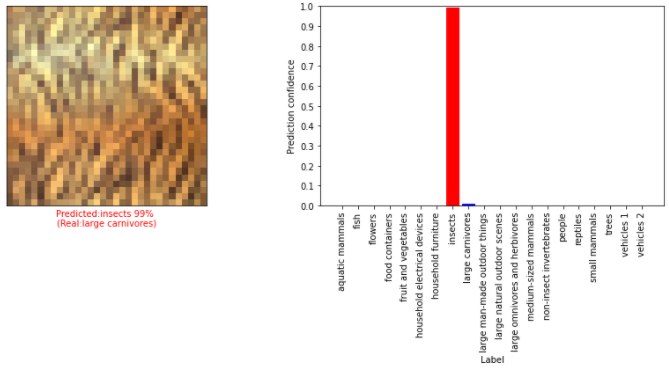
Below, the two (normalized and denormalized) confusion matrices are given on which our conclusions are based.

****



When looking more in detail at some misclassified examples, it is possible to see that quite some examples are again very hard for us as humans to classify in one of these 20 classes. As can be seen from the first picture, it is difficult to see that this example belongs to the real-large man-made outdoor things. This is supposed to be a building but one could also confuse it with a close up picture from an ocean for example. The same is valid for the second figure where the model predicts this to be part of the insects while it belongs to the real-large carnivores.





As some classes got well classified by the model with data augmentation and others got well classified by the model without data augmentation, a final step could be to combine both models in an ensemble in which they can complement one another!

1. <https://arxiv.org/pdf/1409.1556.pdf> [↑](#footnote-ref-0)
2. <https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/>

   <https://www.javatpoint.com/cifar-10-and-cifar-100-dataset-in-tensorflow>

   <https://medium.com/analytics-vidhya/image-classification-using-tensorflow2-0-with-cifar-10-dataset-cc595ceb0082> [↑](#footnote-ref-1)
3. <https://neurohive.io/en/popular-networks/vgg16/> [↑](#footnote-ref-2)
4. [Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929–1958](http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf?utm_content=buffer79b43&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer) [↑](#footnote-ref-3)
5. <https://cs231n.github.io/neural-networks-2/#reg> [↑](#footnote-ref-4)
6. <https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator>

   <https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/> [↑](#footnote-ref-5)