Victor Lindholm vicli815
Jonathan Törnquist jonto406
TBMI26

Second Revise, Questions and results laboration 2

Q1. The shape of X_train and X_test has 4 values. What do each of these represent?

Changes are marked in red

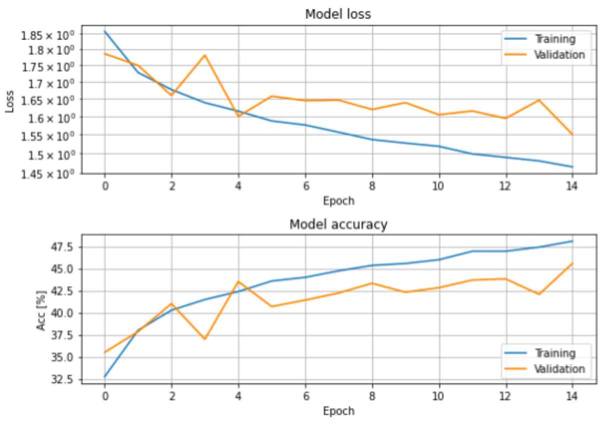
Shape of training data:
(50000, 32, 32, 3)
Shape of test data:
(10000, 32, 32, 3)

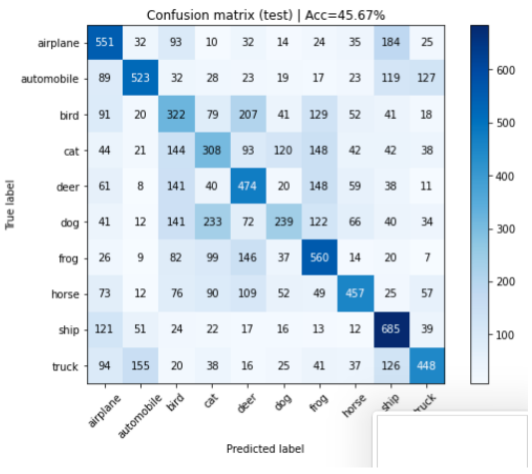
The datasets are of 60 000 images each with a size of 32x32 pixels, divided into 50 000 training images and 10 000 test images. The last number 3, describes the label as a number representation as RGB channel. The number 3 represents red, green, and blue.
https://www.cs.toronto.edu/~kriz/cifar.html

Q2. Train a Fully Connected model that achieves above 45% accuracy on the test data. Provide a short description of your model and show the evaluation image.

We use 150 hidden layers, a ReLu activation function and 10 output nodes with a softmax function (probability).

```
x = Dense(150, activation='relu')(x)
x = Dense(10, activation='softmax')(x) #10 output classes
```





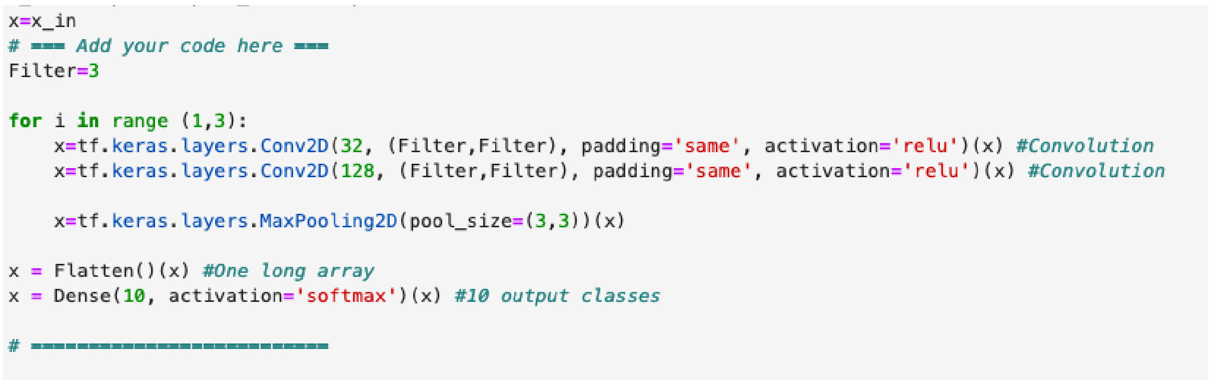
	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	551	32	93	10	32	14	24	35	184	25
automobile	89	523	32	28	23	19	17	23	119	127
bird	91	20	322	79	207	41	129	52	41	18
cat	44	21	144	308	93	120	148	42	42	38
deer	61	8	141	40	474	20	148	59	38	11
dog	41	12	141	233	72	239	122	66	40	34
frog	26	9	82	99	146	37	560	14	20	7
horse	73	12	76	90	109	52	49	457	25	57
ship	121	51	24	22	17	16	13	12	685	39
truck	94	155	20	38	16	25	41	37	126	448

Q3. Compare the model from Q2 to the one you used for the MNIST dataset in the first assignment, in terms of size and test accuracy. Why do you think this dataset is much harder to classify than the MNIST handwritten digits?

This dataset contained 50 000 training samples with a size of 32x32 compared to the first laboration which contained 2770 samples and in a 8x8 matrix. In other words the size of the data to be classified was much bigger and more complex in this assignment. Leading to more training time and a need for an effective model. When classifying a dog for example you need to withcount for locality and shift

Victor Lindholm vicli815
Jonathan Törnquist jonto406
TBMI26
invariance. A dog can be viewed from several angles, it can be of different colours and size. It is easier to classify numbers because the picture in our example is just black and white and there is a more standardised way of writing numbers.

Q4. Train a CNN model that achieves at least 62% test accuracy. Provide a short description of your model and show the evaluation image.

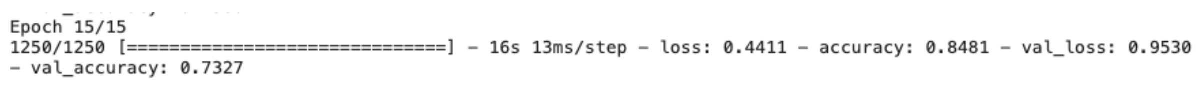


```
x=x_in
# === Add your code here ===
Filter=3

for i in range (1,3):
    x=tf.keras.layers.Conv2D(32, (Filter,Filter), padding='same', activation='relu')(x) #Convolution
    x=tf.keras.layers.Conv2D(128, (Filter,Filter), padding='same', activation='relu')(x) #Convolution

    x=tf.keras.layers.MaxPooling2D(pool_size=(3,3))(x)

x = Flatten()(x) #One long array
x = Dense(10, activation='softmax')(x) #10 output classes

# ==========================
```

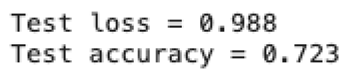
In this model the first step is to initiate a loop which repeats the code 2 times. We uses two blocks. The loop starts with convolution where 32 filters were added where the kernel size was set to (3, 3). These filters are independent of the input nodes and control the number of filters and dimensionality of the output space in a convolutional network. Padding was added as same (evenly padding to all directions so the output has the same dimensions as the input). An activations function (ReLu) is then added. Then we add another convolution layer with 128 filters, same kernel size and same padding as well as relu activation. The next step is pooling = Grouping together. In our example a Max pooling was used, which down-samples the input by reducing its dimensionality. This makes the model more effective, since it doesn't have to learn as many parameters, as well as reducing the risk of overfitting. In our model, the down-sample is done by applying a max filter to subregions of the input. We apply a 9x9 matrix, divided into 9 3x3 sub-matrixes. From each sub-matrix, we take the max value and map it into a 3x3 matrix, placing each selected value at its corresponding place in the 3x3 matrix.

Then we did the same as in Q2 and flattened x into one array and added 10 output nodes with an softmax activation function. This block was then repeated through a loop.

This gave us an accuracy of 72.3 %.

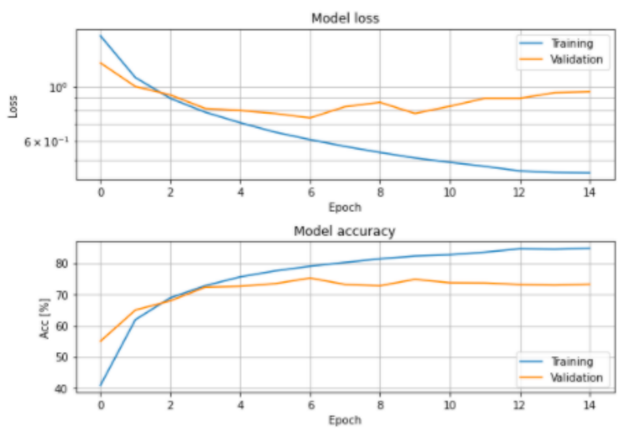


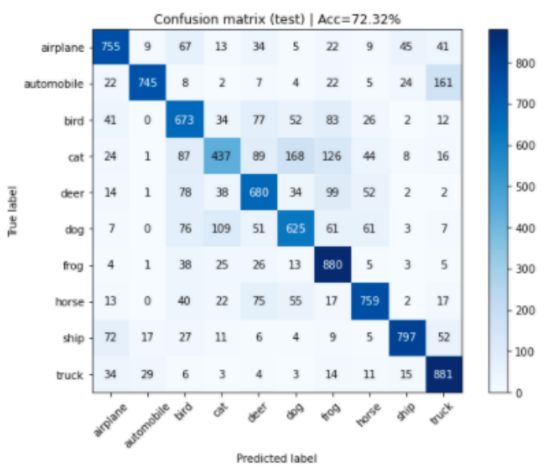
```
Epoch 15/15
1250/1250 [==============================] - 16s 13ms/step - loss: 0.4411 - accuracy: 0.8481 - val_loss: 0.9530
- val_accuracy: 0.7327
```



```
Test loss = 0.988
Test accuracy = 0.723
```

Victor Lindholm vicli815
Jonathan Törnquist jonto406
TBMI26



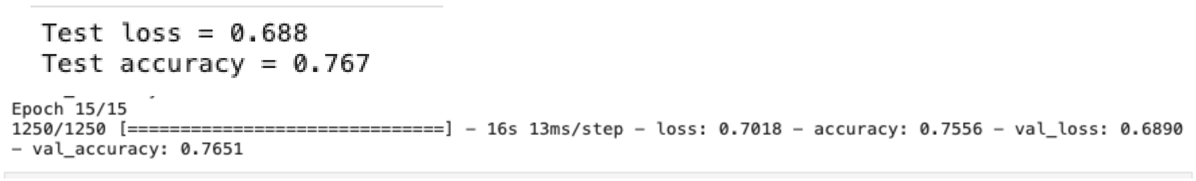


Q5. Compare the CNN model with the previous Fully Connected model. You should find that the CNN is much more efficient, i.e. achieves higher accuracy with fewer parameters. Explain in your own words how this is possible.

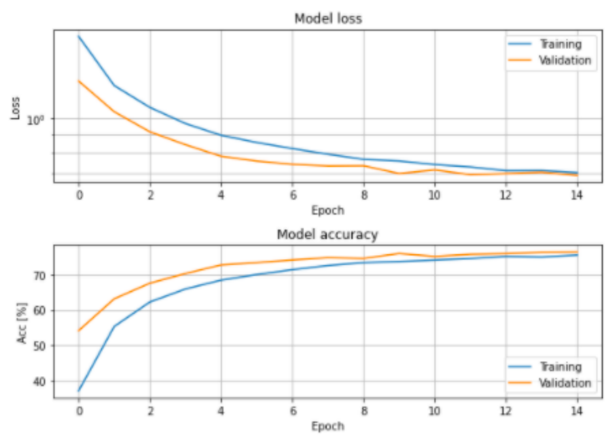
The CNN is better than the earlier model because it uses a kernel which is spatially local and shift invariant. Also the pooling makes the classification more efficient. We use fewer parameters, but reach a higher accuracy.

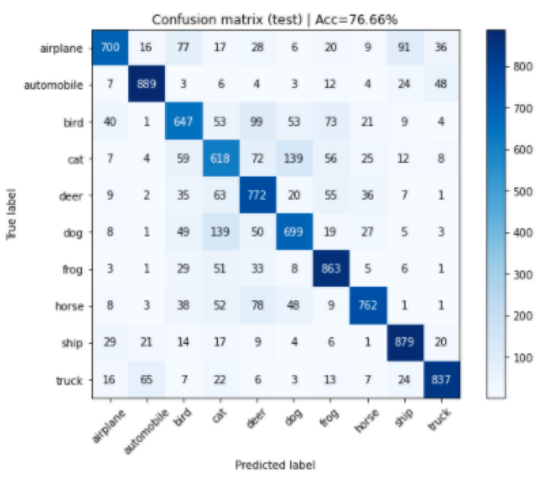
Q6. Train the CNN-model with added Dropout layers. Describe your changes and show the evaluation image.

The dropout layer is added after the pooling, using 0.35 as dropout rate. This gives us an increase of 4.4 %-units (and variates a little by each new run).



```
Test loss = 0.688
Test accuracy = 0.767

Epoch 15/15
1250/1250 [==============================] - 16s 13ms/step - loss: 0.7018 - accuracy: 0.7556 - val_loss: 0.6890
- val_accuracy: 0.7651
```





Victor Lindholm vicli815
Jonathan Törnquist jonto406
TBMI26

Q7. Compare the models from Q4 and Q6 in terms of the training accuracy, validation accuracy, and test accuracy. Explain the similarities and differences (remember that the only difference between the models should be the addition of Dropout layers). Hint: what does the dropout layer do at test time?

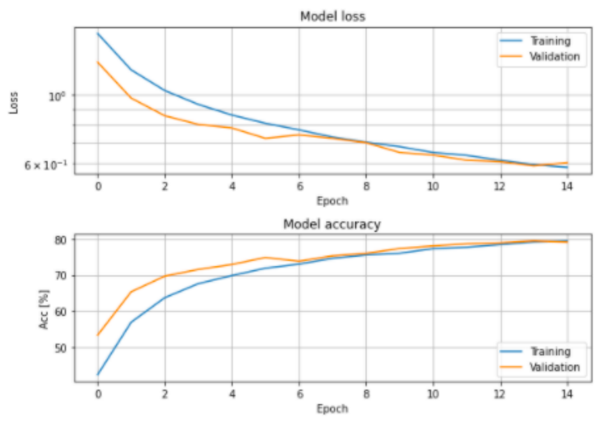
When we add the dropout layer we reduce the overfitting by randomly setting nodes to zero. This is forcing the network to use different sets of hidden features each time. This leads to a lower accuracy in training data, but better validation and test accuracy compared to Q4.

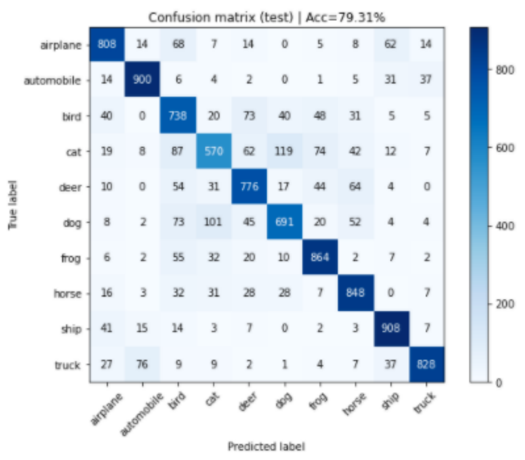
Victor Lindholm vicli815
Jonathan Törnquist jonto406
TBMI26

Q8. Train the CNN model with added BatchNorm layers and show the evaluation image.

```
Test loss = 0.625
Test accuracy = 0.793
```

```
Epoch 15/15
1250/1250 [==============================] - 19s 15ms/step - loss: 0.5839 - accuracy: 0.7947 - val_loss: 0.6047
- val_accuracy: 0.7918
```





	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	808	14	68	7	14	0	5	8	62	14
automobile	14	900	6	4	2	0	1	5	31	37
bird	40	0	738	20	73	40	48	31	5	5
cat	19	8	87	570	62	119	74	42	12	7
deer	10	0	54	31	776	17	44	64	4	0
dog	8	2	73	101	45	691	20	52	4	4
frog	6	2	55	32	20	10	864	2	7	2
horse	16	3	32	31	28	28	7	848	0	7
ship	41	15	14	3	7	0	2	3	908	7
truck	27	76	9	9	2	1	4	7	37	828

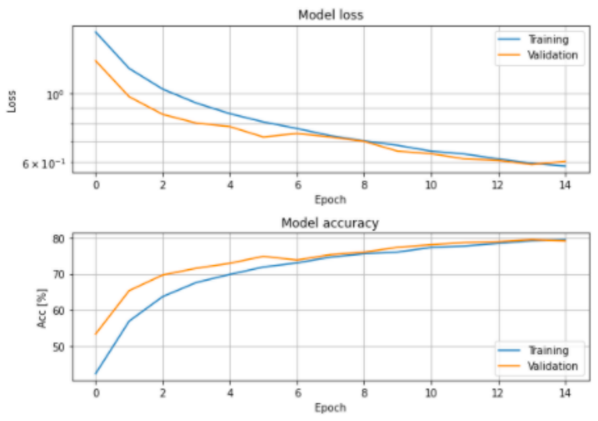
Q9. When using BatchNorm one must take care to select a good minibatch size. Describe what problems might arise if the wrong minibatch size is used. You can reason about this given the description of BatchNorm in the Notebook, or you can search for the information in other sources. Do not forget to provide links to the sources if you do!
The minibatch size is the amount of data that is being used to update weights in each training epoch. Say that we have training data = 50 000 and minibatch size = 500. Then 50 000 / 500 = 100 updates are done per epoch. A small batch size leads to a faster training time, but could also lead to oscillatory behavior and could obstruct convergence[1]. A too-large batch size slows down the training time and could lead to overfitting and poor generalization.

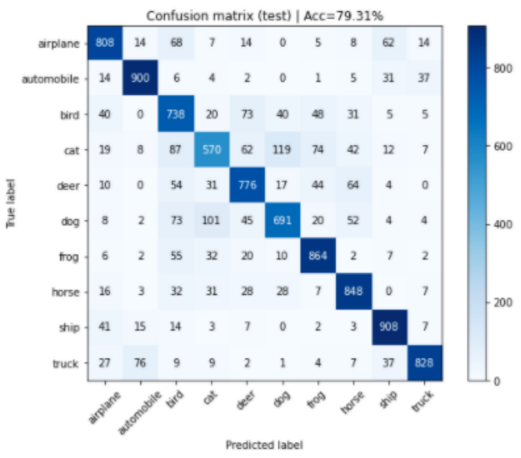
[1] K., Shen (2018) - *Effect of batch size on training dynamics*
https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e

Victor Lindholm vicli815
Jonathan Törnquist jonto406
TBMI26

Q10. Design and train a model that achieves at least 75% test accuracy in at most 25 epochs. Explain your model and motivate the design choices you have made and show the evaluation image.

We continued to build up the CNN model by blocks and by adding dropout layer and batch norm layer. The dropout layer was added to reduce the overfitting from the first model and the batch norm layer was partially added to reduce training time, regulariate the network and ease the initial weight setting. Then we added another dense layer with a relu activation function and tried different values. For example by changing the dropout rate. The requested test accuracy was reached within 15 epochs. See attached figures and code.





Final model, accuracy 79.3 %

```
Test loss = 0.625
Test accuracy = 0.793
```

```
from tensorflow.keras.utils import plot_model

x_in = Input(shape=X_train.shape[1:])
x=x_in

# === Add your code here ===
Filter=3

for i in range (1,3):
    x=tf.keras.layers.Conv2D(32, (Filter,Filter), padding='same', activation='relu')(x) #Convolution
    x=tf.keras.layers.Conv2D(128, (Filter,Filter), padding='same')(x) #Convolution
    x = tf.keras.layers.BatchNormalization(
        axis=-1, momentum=0.75, epsilon=0.002, center=True, scale=True,
        beta_initializer='zeros', gamma_initializer='ones', #Initial weigth
        moving_mean_initializer='zeros', #my=0
        moving_variance_initializer='ones', #sigma=1
        renorm=False, renorm_clipping=None, renorm_momentum=0.99) (x)

    x=Activation('relu')(x) #Add activation function after Batch.
    x=tf.keras.layers.MaxPooling2D(pool_size=(3,3))(x)
    x=tf.keras.layers.Dropout(
        rate=0.35, noise_shape=None, seed=None)(x)


x = Flatten()(x) #One long array
x = Dense(100, activation='relu')(x)
x = Dense(10, activation='softmax')(x) #10 output classes
```