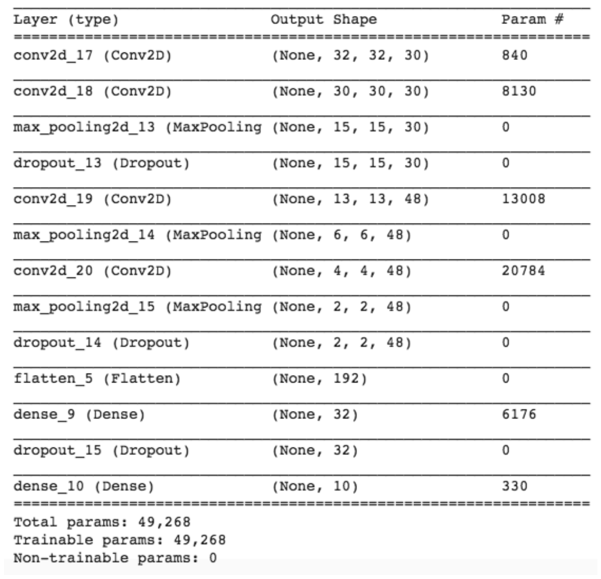
Deep Learning - 236606 - HW3

Ido Glanz - 302568936 Matan Weksler - 302955372

1. **CNN architectures:**

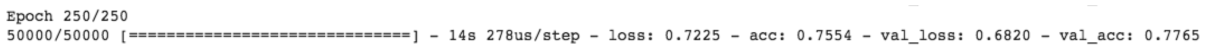
   In this part, we experimented with different CNN architectures for the CIFAR-10 dataset aiming to reach 85% accuracy for the validation set and using the Keras API while maintaining less than 50K trainable parameters. The dataset in general consists of 50K samples of 10 different classes, each one a picture 32-by-32 (RGB).

   We started by reading on different architectures (mostly ones reviews in class) and using first a simple CNN with 13 layers as follows:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_17 (Conv2D)           (None, 32, 32, 30)        840       
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 30, 30, 30)        8130      
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 15, 15, 30)        0         
_________________________________________________________________
dropout_13 (Dropout)         (None, 15, 15, 30)        0         
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 13, 13, 48)        13008     
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 6, 6, 48)          0         
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 4, 4, 48)          20784     
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 2, 2, 48)          0         
_________________________________________________________________
dropout_14 (Dropout)         (None, 2, 2, 48)          0         
_________________________________________________________________
flatten_5 (Flatten)          (None, 192)               0         
_________________________________________________________________
dense_9 (Dense)              (None, 32)                6176      
_________________________________________________________________
dropout_15 (Dropout)         (None, 32)                0         
_________________________________________________________________
dense_10 (Dense)             (None, 10)                330       
=================================================================
Total params: 49,268
Trainable params: 49,268
Non-trainable params: 0
```

   Using the Adam optimizer with learning rate 0.0001, 1e-6 decay and running for 250 epochs we got:



```
Epoch 250/250
50000/50000 [==============================] - 14s 278us/step - loss: 0.7225 - acc: 0.7554 - val_loss: 0.6820 - val_acc: 0.7765
```

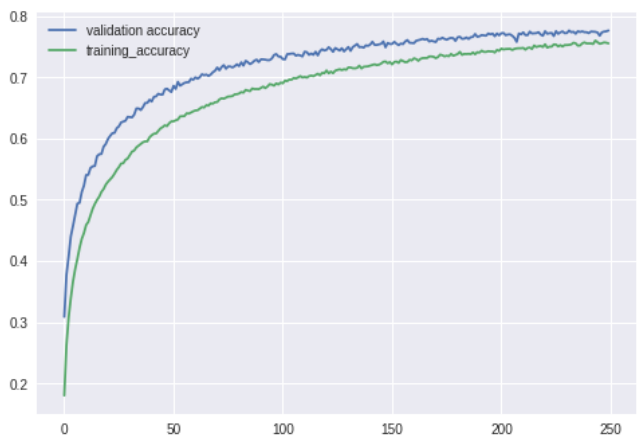


Figure 1 - Accuracy plots for the first architecture. After a certain epoch number the test accuracy was pretty much saturated and didn't seem to improve anymore.

Running for more epochs did not do the job and we had to look for other architectures which could converge to the needed accuracy with the given parameters size

After reading more we started experimenting with the DenseNet architecture (*Huang, Gao, et al. "Densely connected convolutional networks." CVPR. Vol. 1. No. 2. 2017*), which is in a way similar to ResNet, utilizing a building block unit in which each layers' input feature map dimension is merged with previous layers' input as so it grows gradually inside a block. While in ResNet the previous layers' input is added (summed) to the current ones, in DenseNet it is concatenated as so the inputs dimension gradually grows (i.e. if the block input is a of 10 feature maps, and the first layers output is 4, the next layers input will be 14 feature maps and so on). The blocks are then chained together with transition layers in between, reducing the number of feature maps where needed. These types of architectures allow usage of less parameters (as they in a sense "share" layers) and after altering it a bit to reduce the number of parameters even more we ended up with the following network:

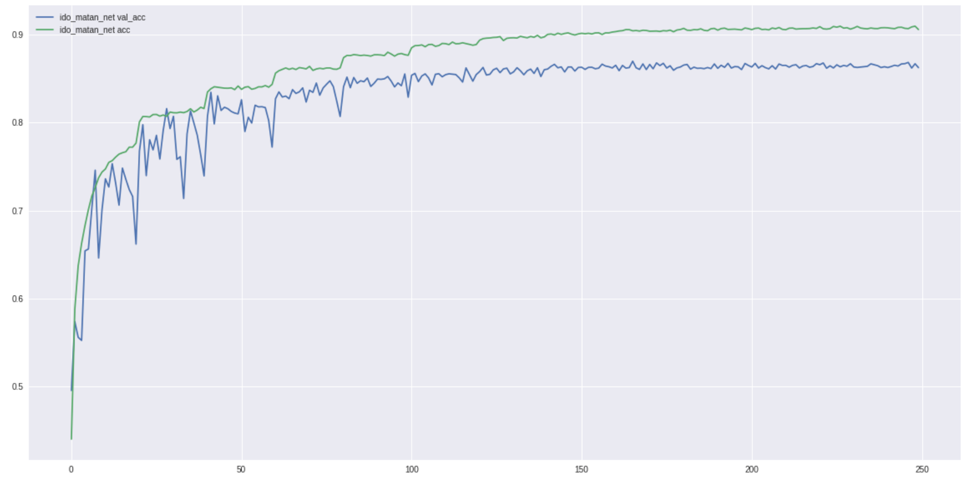
```
Layer (type)                     Output Shape         Param #     Connected to
==========================================================================================
input_1 (InputLayer)             (None, 32, 32, 3)    0
__________________________________________________________________________________________
conv2d_1 (Conv2D)                (None, 32, 32, 24)   648         input_1[0][0]
__________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 32, 32, 24)   96          conv2d_1[0][0]
__________________________________________________________________________________________
activation_1 (Activation)        (None, 32, 32, 24)   0           batch_normalization_1[0][0]
__________________________________________________________________________________________
conv2d_2 (Conv2D)                (None, 32, 32, 48)   1152        activation_1[0][0]
__________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 32, 32, 48)   192         conv2d_2[0][0]
__________________________________________________________________________________________
activation_2 (Activation)        (None, 32, 32, 48)   0           batch_normalization_2[0][0]
__________________________________________________________________________________________
conv2d_3 (Conv2D)                (None, 32, 32, 12)   5184        activation_2[0][0]
__________________________________________________________________________________________
concatenate_1 (Concatenate)      (None, 32, 32, 36)   0           conv2d_3[0][0]
                                                                  conv2d_1[0][0]
__________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 32, 32, 36)   144         concatenate_1[0][0]
__________________________________________________________________________________________
activation_3 (Activation)        (None, 32, 32, 36)   0           batch_normalization_3[0][0]
__________________________________________________________________________________________
```

```
conv2d_4 (Conv2D)               (None, 32, 32, 48)   1728        activation_3[0][0]
________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 32, 32, 48)   192         conv2d_4[0][0]
________________________________________________________________________________________________
activation_4 (Activation)       (None, 32, 32, 48)   0           batch_normalization_4[0][0]
________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 32, 32, 12)   5184        activation_4[0][0]
________________________________________________________________________________________________
concatenate_2 (Concatenate)     (None, 32, 32, 48)   0           conv2d_5[0][0]
                                                                 concatenate_1[0][0]
________________________________________________________________________________________________
batch_normalization_5 (BatchNor (None, 32, 32, 48)   192         concatenate_2[0][0]
________________________________________________________________________________________________
activation_5 (Activation)       (None, 32, 32, 48)   0           batch_normalization_5[0][0]
________________________________________________________________________________________________
conv2d 6 (Conv2D)               (None, 32, 32, 24)   1152        activation 5[0][0]
________________________________________________________________________________________________
average_pooling2d_1 (AveragePoo (None, 16, 16, 24)   0           conv2d_6[0][0]
________________________________________________________________________________________________
batch_normalization_6 (BatchNor (None, 16, 16, 24)   96          average_pooling2d_1[0][0]
________________________________________________________________________________________________
activation_6 (Activation)       (None, 16, 16, 24)   0           batch_normalization_6[0][0]
________________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, 16, 16, 48)   1152        activation_6[0][0]
________________________________________________________________________________________________
batch normalization 7 (BatchNor (None, 16, 16, 48)   192         conv2d 7[0][0]
________________________________________________________________________________________________
activation_7 (Activation)       (None, 16, 16, 48)   0           batch_normalization_7[0][0]
________________________________________________________________________________________________
conv2d_8 (Conv2D)               (None, 16, 16, 12)   5184        activation_7[0][0]
________________________________________________________________________________________________
concatenate_3 (Concatenate)     (None, 16, 16, 36)   0           conv2d_8[0][0]
                                                                 average_pooling2d_1[0][0]
________________________________________________________________________________________________
batch_normalization_8 (BatchNor (None, 16, 16, 36)   144         concatenate_3[0][0]
________________________________________________________________________________________________
activation_8 (Activation)       (None, 16, 16, 36)   0           batch_normalization_8[0][0]
________________________________________________________________________________________________
conv2d_9 (Conv2D)               (None, 16, 16, 48)   1728        activation_8[0][0]
________________________________________________________________________________________________
batch_normalization_9 (BatchNor (None, 16, 16, 48)   192         conv2d_9[0][0]
________________________________________________________________________________________________
activation_9 (Activation)       (None, 16, 16, 48)   0           batch_normalization_9[0][0]
________________________________________________________________________________________________
conv2d_10 (Conv2D)              (None, 16, 16, 12)   5184        activation_9[0][0]
________________________________________________________________________________________________
concatenate_4 (Concatenate)     (None, 16, 16, 48)   0           conv2d_10[0][0]
                                                                 concatenate_3[0][0]
________________________________________________________________________________________________
batch_normalization_10 (BatchNo (None, 16, 16, 48)   192         concatenate_4[0][0]
________________________________________________________________________________________________
activation_10 (Activation)      (None, 16, 16, 48)   0           batch_normalization_10[0][0]
________________________________________________________________________________________________
conv2d_11 (Conv2D)              (None, 16, 16, 24)   1152        activation_10[0][0]
________________________________________________________________________________________________
average pooling2d 2 (AveragePoo (None, 8, 8, 24)     0           conv2d 11[0][0]
________________________________________________________________________________________________
batch_normalization_11 (BatchNo (None, 8, 8, 24)     96          average_pooling2d_2[0][0]
________________________________________________________________________________________________
activation_11 (Activation)      (None, 8, 8, 24)     0           batch_normalization_11[0][0]
________________________________________________________________________________________________
conv2d_12 (Conv2D)              (None, 8, 8, 48)     1152        activation_11[0][0]
________________________________________________________________________________________________
batch_normalization_12 (BatchNo (None, 8, 8, 48)     192         conv2d_12[0][0]
________________________________________________________________________________________________
activation 12 (Activation)      (None, 8, 8, 48)     0           batch normalization 12[0][0]
________________________________________________________________________________________________
conv2d_13 (Conv2D)              (None, 8, 8, 12)     5184        activation_12[0][0]
________________________________________________________________________________________________
concatenate_5 (Concatenate)     (None, 8, 8, 36)     0           conv2d_13[0][0]
                                                                 average_pooling2d_2[0][0]
________________________________________________________________________________________________
batch_normalization_13 (BatchNo (None, 8, 8, 36)     144         concatenate_5[0][0]
________________________________________________________________________________________________
activation_13 (Activation)      (None, 8, 8, 36)     0           batch_normalization_13[0][0]
________________________________________________________________________________________________
conv2d_14 (Conv2D)              (None, 8, 8, 48)     1728        activation_13[0][0]
________________________________________________________________________________________________
batch_normalization_14 (BatchNo (None, 8, 8, 48)     192         conv2d_14[0][0]
________________________________________________________________________________________________
activation_14 (Activation)      (None, 8, 8, 48)     0           batch_normalization_14[0][0]
________________________________________________________________________________________________
conv2d_15 (Conv2D)              (None, 8, 8, 12)     5184        activation_14[0][0]
________________________________________________________________________________________________
concatenate_6 (Concatenate)     (None, 8, 8, 48)     0           conv2d_15[0][0]
                                                                 concatenate 5[0][0]
________________________________________________________________________________________________
batch_normalization_15 (BatchNo (None, 8, 8, 48)     192         concatenate_6[0][0]
________________________________________________________________________________________________
activation_15 (Activation)      (None, 8, 8, 48)     0           batch_normalization_15[0][0]
________________________________________________________________________________________________
global_average_pooling2d_1 (Glo (None, 48)           0           activation_15[0][0]
________________________________________________________________________________________________
dense_1 (Dense)                 (None, 10)           490         global_average_pooling2d_1[0][0]
================================================================================================
Total params: 45,634
Trainable params: 44,410
```

```
Non-trainable params: 1,224
```

We used the SGD optimizer and the datagen to enrich the data, and running for 250 epochs we obtained:

```
Epoch 250/250
782/782 [==============================] - 50s 63ms/step - loss: 0.3406 - acc: 0.9057 - val_loss: 0.5165 - val_acc: 0.8625
```



2. Transfer Learning

In this section, we experimented with 2 different transfer learning techniques using the VGG-16 architecture, trained for CIFAR-100 dataset in aim to generate a classifier for a subsection of the CIFAR-10 data. This relays on the general idea of having the inner layers of the network serve as feature extractors, regardless of the number of the specific number of classes in the given data set, and have the last fully connected layer (or layers) classify the image based on the features extracted and the number of classes.

a. In the first part, we used the pre-trained weights for the VGG-16 on CIFAR-100 dataset that was given in the recitation and replaced the last layer with one augmented for the 10 classes in the CIFAR-10 dataset. We then froze all the weights aside from the new layer we introduced, hence having only 5130 out of 15,001,418 parameters that are trainable. We ran the network using subsets of the CIFAR-10 of sizes 100, 1000 and 10K and the results were as follows:

Train set size	Train set loss	Train set accuracy	Test set loss	Test set accuracy
100	1.7424	0.6389	2.2422	0.45
1000	1.8095	0.5399	1.7672	0.5530
10K	1.8868	0.5252	1.7202	0.5820

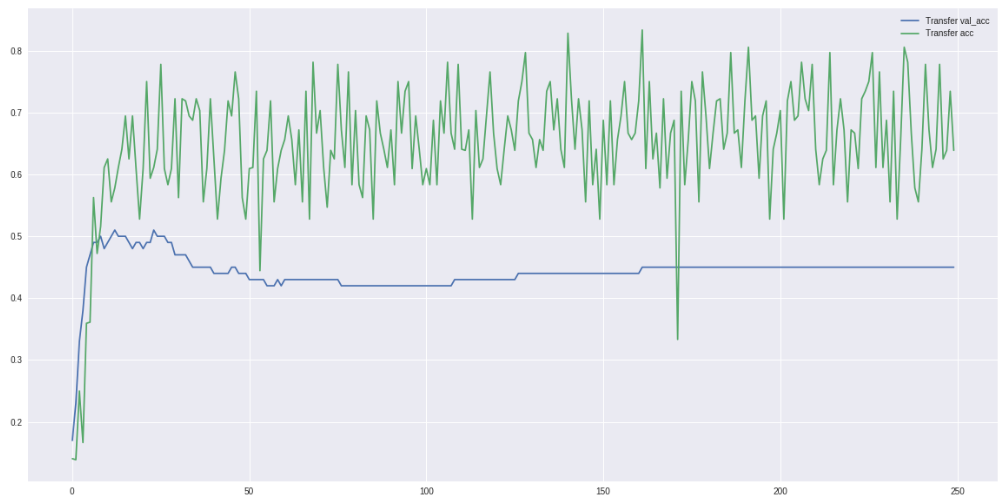


Figure 2 - Accuracy plot for the first transfer method and for a 100-sample data set. As can be seen, the net over-fitted the train set even with all the regularization and dropout

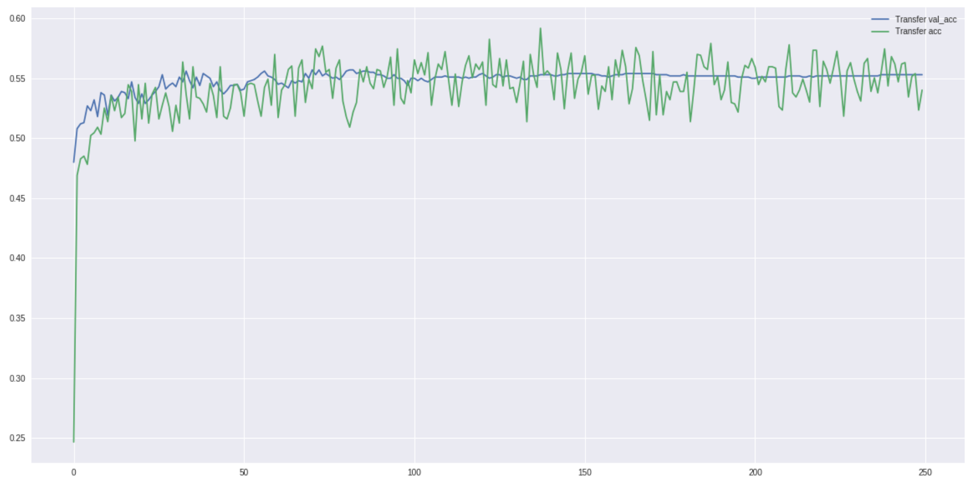


Figure 3 - Accuracy plot for the 1000 sample set size. No over-fitting present yet the net converged to only 55% accuracy

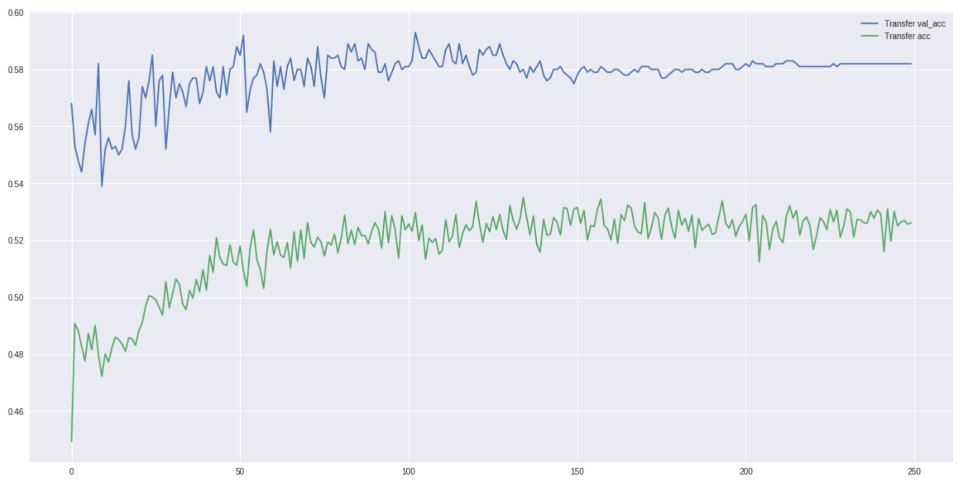


Figure 4 - Accuracy plot for the 10K data set, here even though the train set was larger, as it was only tuning ~5000 parameters, the net couldn't really obtain better results than for the 1K set.

b. In the second part, we implemented a KNN classifier based on the first fully connected layer, again under the assumption it contains a variety of features extracted from the image and based on these a classification can be inferred.
In practice, we used the VGG-16 network with the pre-trained weights from the recitation, only stopping now at the layer-before-last, hence having an output of 512 features extracted by the network. We then used the SKlearn tool box to map features extracted from a subset of the training set to the KNN space and ran the test set through the network, then predicting their classification using the KNN map (again, using the SKlearn library). As ascribed in the assignment, we ran the classifier for subsets of sizes 100, 1000 and 10K. For the K parameter in the KNN algorithm, we ran it from k=1 to k=40 and the results were as follows:

For train set size = 100 (and test set also of 100 samples) we get:

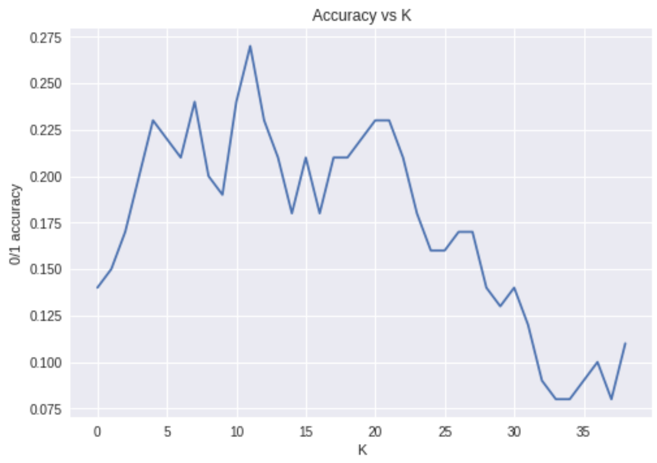


Figure 5 - Tests set accuracy vs K. The optimal k was 12 giving an accuracy of: 0.27

For train set size = 1000 (and test set also of 1000 samples) we get:

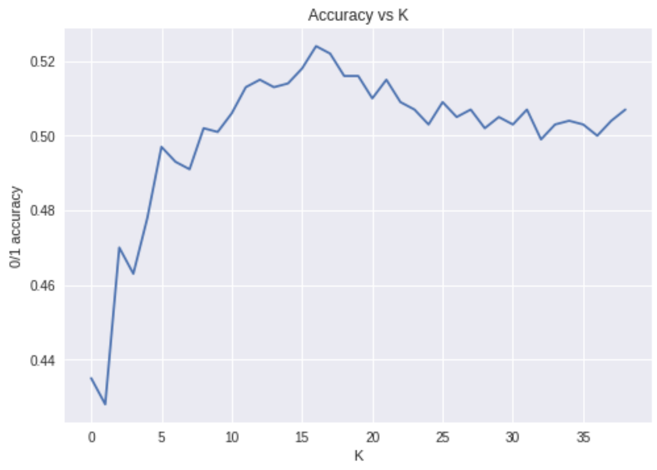


Figure 6 - Tests set accuracy vs K. The optimal k was 17 giving an accuracy of: 0.524

For train set size = 10000 (and test set of 1000 samples) we get:

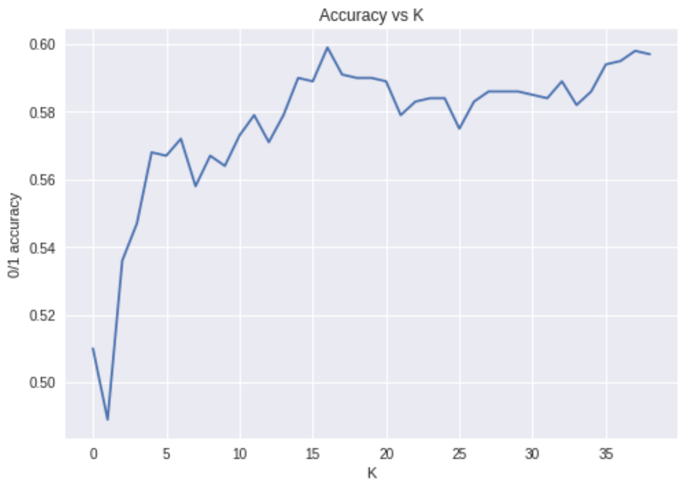
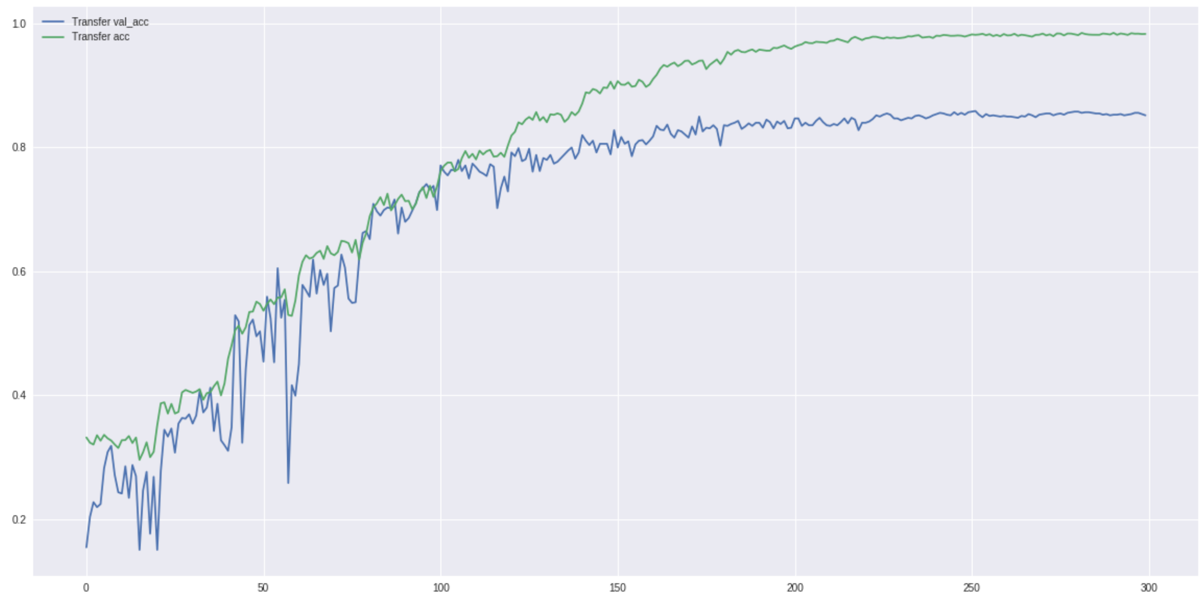


Figure 7 - Tests set accuracy vs K. The optimal k was 17 giving an accuracy of: 0.599

As we can see for the EKNN method, the results seem to be bounded in the ~0.6 accuracy level. This might be related to the different method features are combined in the FC layer (weighted sum) and spatially in the KNN (or just the lack of the architecture to do the task).

c. In the last section, we searched for a different method for transfer learning based on the CIFAR data base. First of all, to be honest, we did not manage to find a very good one (or at-least one that we managed to implement). Having said that, and after reading a lot of different approaches, a few (we assume common) methods seemed interesting to us, main one was "freezing" only some of the weights or heavily regulating them in the convolutional layers, hence allowing, some or all, of weights to be trained based on the new and limited data set. This is a common yet effective method (which is often prone to catastrophically forgetting of the previous data it was trained for, i.e. it "forgets" how to classify the original data set it was trained for; the CIFAR 100 data set. Running this method, we ran into to a lot of technical issues and ended up running the net while allowing all weights to be trained but with heavy regularization (0.01 weight_decay).
Running this for 300 epochs with 10K gave us the following results:



```
Epoch 300: loss: 0.4430 - acc: 0.9837 - val_loss: 0.9463 - val_acc: 0.8520
```

Running with more regularization might derive better results (or add dropout)

We also tried running the convolutional part as it was (up to the flattening layer) and then using a second FCN to predict the CIFAR-10 but (maybe due to implementation errors) this did not give us any results over the 0.4-0.5 accuracy on the test set for 10K samples (it did though, over-fit the train set pretty nicely for 100 and 1000 samples)

Overall, we didn’t manage to find a very effective method, either due to bad implementations or just not-so-good architecture ideas. That said, we did learn a lot reading different articles on this topic.