




046746 wet HW 2

submitted by:

Nir Sassy: 316491737



Part 1- Classic Classifier

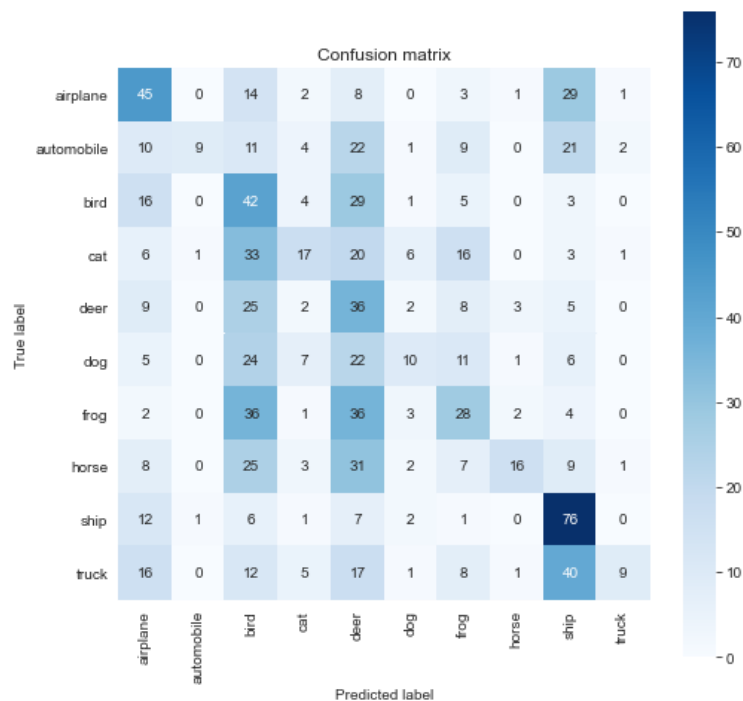
Please note that the classifier was trained on a shuffled 'train-set' and tested on an unshuffled test-set, this can be changed by changing the value of `shuffle_data_base` to `False`.

1. Showing 5 images of the CIFAR10 dataset with the appropriate labels:



CIFAR 10 Dataset example images with labels

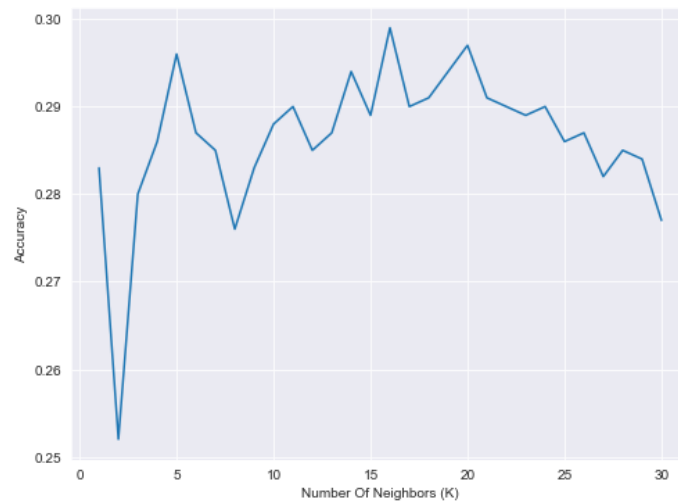
2. After building the K-NN classifier with 10 nearest neighbors trained on 10,000 examples and tested on 1000 examples from the test-set we saw a prediction accuracy of 0.286 with the nest confusion matrix:



KNN Classifier Confusion Matrix

We can conclude that as expected the K-NN classifier isn't doing a very good job on this large dataset. In general, it's clear that the values on the diagonal aren't high compared to other values on the matrix, hinting on high misclassifications ratio.

3. K is a hyper-parameter that describes the number of neighbors in K -NN algorithm. Like every hyper-parameter, it can be tested to check which value yields the 'best' result (best in terms of some metric – here we used accuracy). Testing K values in the range $[1,30]$, we received the following chart:

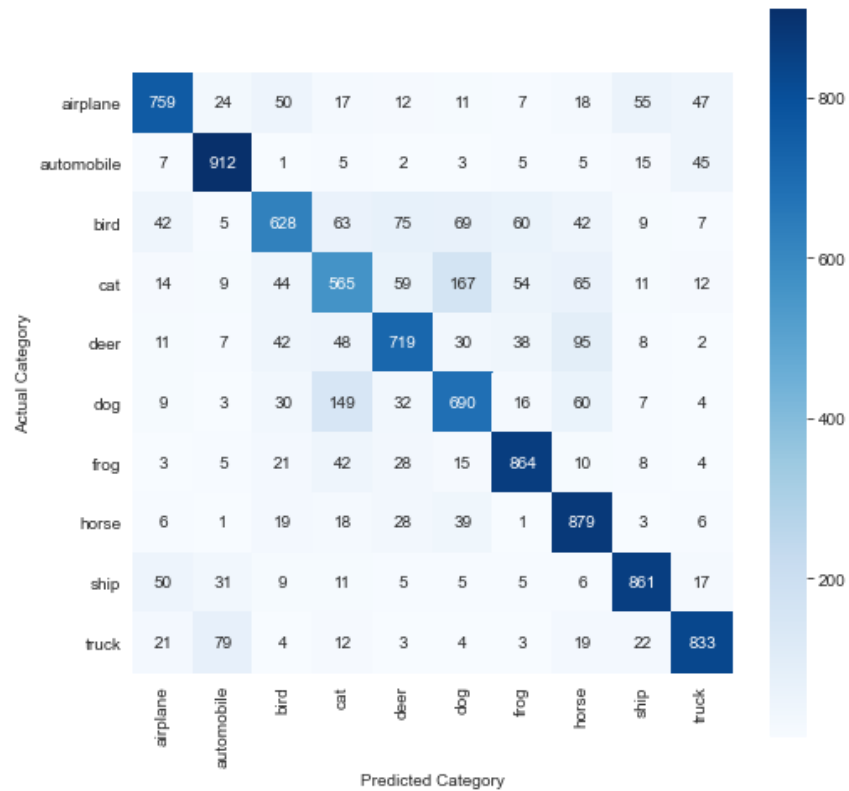


KNN accuracy as a function of K

The best accuracy was 0.299 and was achieved with $K=16$ neighbors.

Part 2- Design and Build a CNN Classifier

We trained the 'SvhnCNN' Network from tutorials 3-4 for 20 epochs and received a prediction accuracy of 77.100% and the following confusion matrix:



The most confusing classes for the model are classes cats and dogs, but also bird has a very high misclassification rate with other labels.

2. Our CNN classifier:

Our CNN Classifier used LeakyReLU activation with ResidualBlocks.

The CNN part has 3 convolutional layers each followed up by a batch normalization layer, 4 residual blocks, 7 leaky ReLU activation layers (after every conv2d layer and residual block layer), 3 max pooling layers and 1 dropout layer ($p=0.5$).

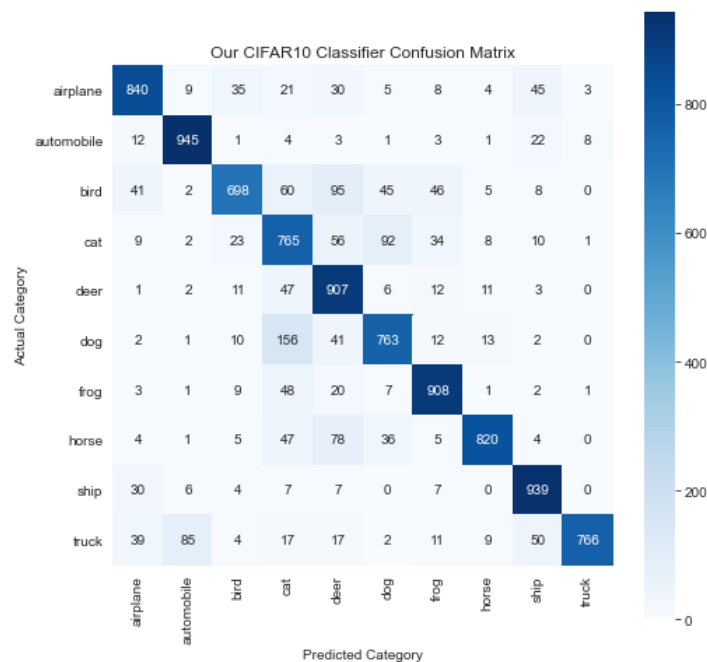
The classifier head of the network contains 2 fully-connected layers with one leaky ReLU activation layer and one dropout layer. The total number of layers in the network is 26.

The filter sizes are a standard 3x3, the output from the last layer of the CNN part is shaped [126x6x6] which is then flattened for a feature vector with size of 4608 that goes into the classifier head first layer. This layer receives an input of size 4608 and outputs a vector with size of 64 which goes to the last FC layer that outputs a vector with size 10, which represents the classifier scores for each label.

The Leaky ReLU activation function prevents some neurons in the network to 'die' because of negative weights (Regular ReLU function is zero for negative values), which can help the model to fully utilize all of its layers. The Residual Blocks helps to prevent the vanishing gradient problem where the gradients become too small in the back propagation phase (especially in deeper neural networks), causing the weights updates to be too small and the training ineffective. It does it by allowing the gradients to skip connections and bypass some of the layers in the network.

The hyper-parameters chose to be the same as the previous network (batch_size = 128, learning rate = $1e-4$, 20 epochs). The total number of trainable parameters are 6,956,874.

Our network got a score of 83.510% test accuracy with the following confusion matrix:



We can see that overall, the parts where the previous network failed are the parts where our network fails, but the missclassification rate mostly reduced. The model still have hard time classifying between dogs and

cats, birds are still confused with bunch of other classes (cats, deer, dog) and an interesting point: there are some classes which are more likely to be missclassified, despite the overall improvement. For example, horse are being missclassified as deer, cat or frog way more than the previous network, hinting on overfitting of our model.

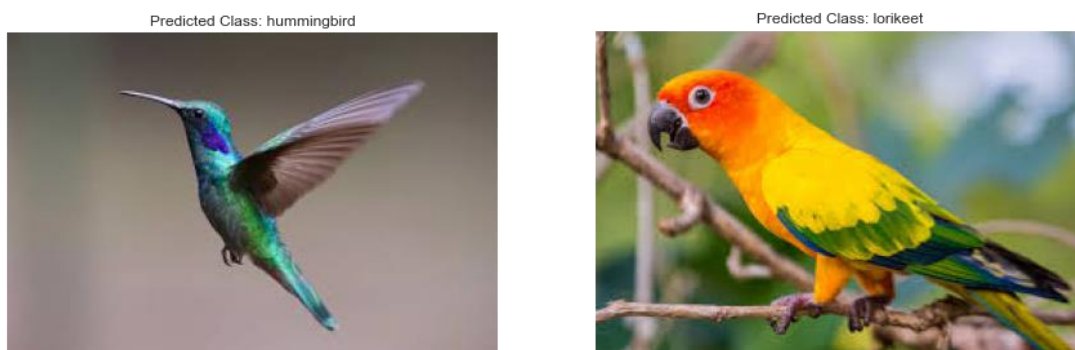
Part 3 - Analyzing a Pre-trained CNN & Adversarial Examples

1. Loaded the pretrained vgg16 model from torchvision.models and set it to eval mode.
2. Created a 'load' function that receives a folder path and returns an array of the images inside that ends with '.jpg'. We used the function to load the photos in data/birds folder and showed them:



Birds Images from 'data/birds' folder

3. We defined a function called 'preprocess' that preprocesses an image before sending it through vgg16 pretrained net. The steps this function does are (in order):
 - Resizing the image to size 224x224 using bilinear interpolation.
 - Converts the image to a torch.tensor object which also scales the values inside to the range [0, 1] by dividing all the values by 255.
 - Normalize the values by a mean of [0.485, 0.456, 0.406] and std of [0.229, 0.224, 0.225]
 - Add an extra dimension at the beginning of the tensor, effectively creating a batch of size 1 as that's what the VGG16 network expects to receive.
4. Created a labels array with the provided txt file. We then passed forward the birds images in the network and plugged the 1000-sized output vector to a SoftMax function to get probabilities. We took the argmax on the probabilities array to get the index of the prediction and plugged this index to the labels array to get the label. The results are:



Birds - Classified

We can see that the pretrained model didn't have any difficulty to properly classify those images.

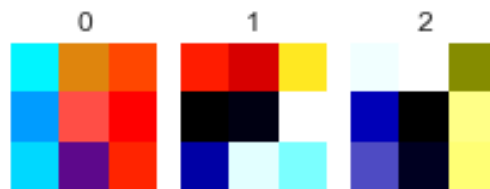
5. We found a picture of a boxer dog on the internet and fed it to the network. The result is:



Image of a dog from the internet - classified.

The model correctly classified the image.

6. The first 3 filters of the first layer of VGG16 Network are:



First 3 filters of the first layer of VGG16 pretrained CNN

And their output:



Outputs of the first 3 filters for the dog image from the internet

We can clearly see that the overall shape of the dog is still present, as these are the outputs of the filters from the first layer. We can see that the filters focus and emphasize different aspects of the image, s.t going deeper in the network those aspects would translate into features that have some meaning and can be translated to a prediction.

7. We decided to remove the last FC layer (FC-8) and extracted the features of the cats and dogs images from the FC7 layer output of the VGG16 Network. We wrote a function that returns the output of the FC7 layer by deep copying the original model and removes the last layer from the copied model. We then fed the copied network the dogs and cats images and stacked the features for each image in a tensor. The feature-space size (output of fc7 layer) is 4096, meaning that 4096 features are given for each image.

8. We trained a Random Forest Classifier to classify cats and dogs based on the features extracted in section 7. We labeled the cats images '0' and the dogs images '1' and made an array to supervise the training called 'y_train'. We tested the model on some test data from the internet and got the following results:



Test data for the RFC model - Classified.

We can see that our random forest classifier made 100% accuracy on this test data. This is quite remarkable because we trained the model on features from only 18 images (9 cats and 9 dogs) and got perfect score, even for test data of only 4 images.

9. We picked an image of a zebra in the safari and fed it to the network. The result is:

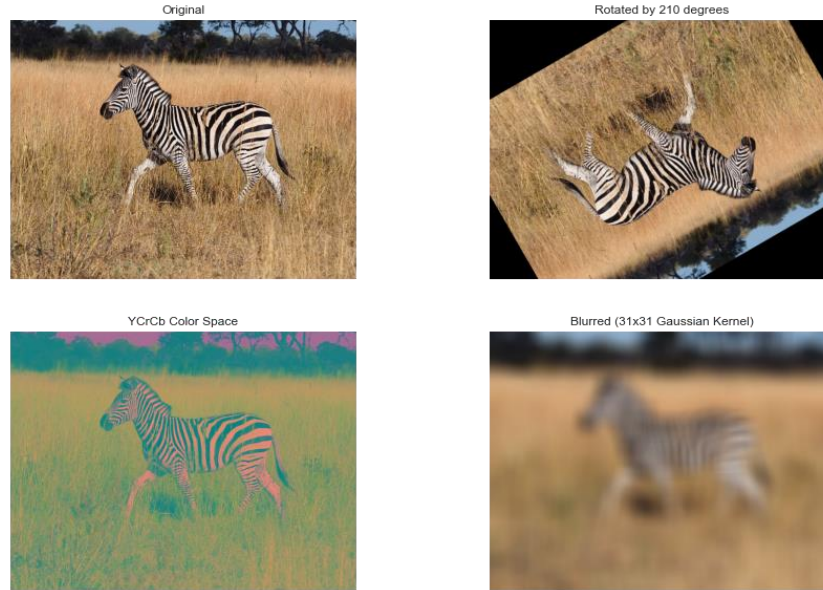


As expected, a zebra in its natural habitat shouldn't be any problem for the pretrained VGG16 Network to classify correctly.

10. We applied the following 3 transformations on the zebra image:

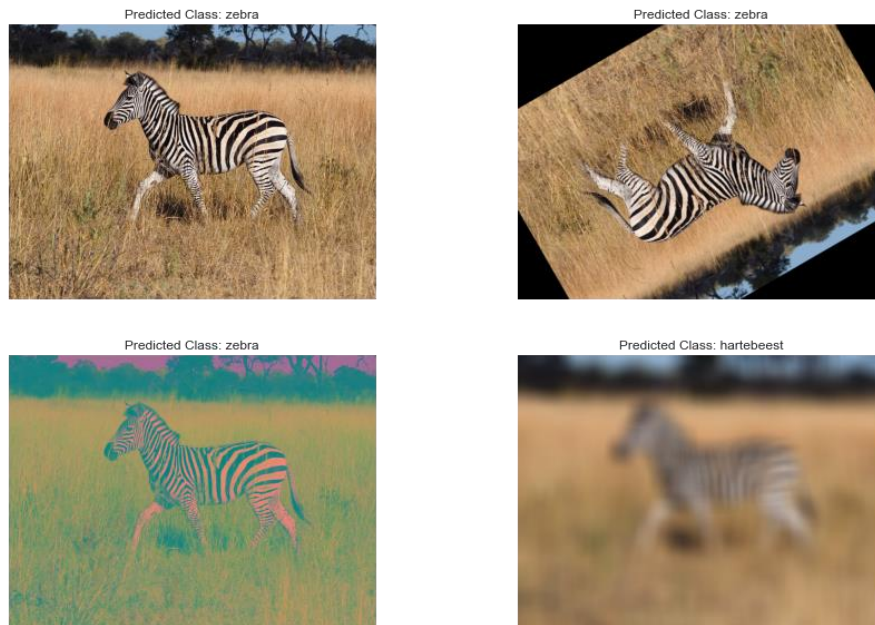
- Rotation by 210 degrees
- Blurring (By 31x31 gaussian kernel with std=20)
- Changed the color space to 'YCrCb'.

The transformed images look like this:



Original zebra image and its transformations

11. We fed the transformed images to the network and got the following results:



Original zebra image and its transformations – classified.

Well, guess VGG16 pretrained Network is robust to zebras. The model properly classified the image as a zebra after these transformations, and we even had to raise the gaussian kernel to 31x31 and the std to 20 for it to not be able to classify it correctly.

So, we tried the given cow image with the same transformations:



Given cow image with the same transformations – classified.

This time, we successfully ‘fooled’ the pretrained model by rotation and color-space change, but the blurred image was correctly classified.

This happens because the model doesn’t necessarily learn how to extract features about cows or zebras specifically but also features about their typical background, lighting, and other aspects during training, s.t changing those aspects alone might change the predicted class altogether.