EE046746: Computer Vision
HW-2
Classification

Daniela Boguslavsky 315720003
Shahar Rashty 312465305

Due date: **15.5.23**

1. 5 images from the training set with their labels as title

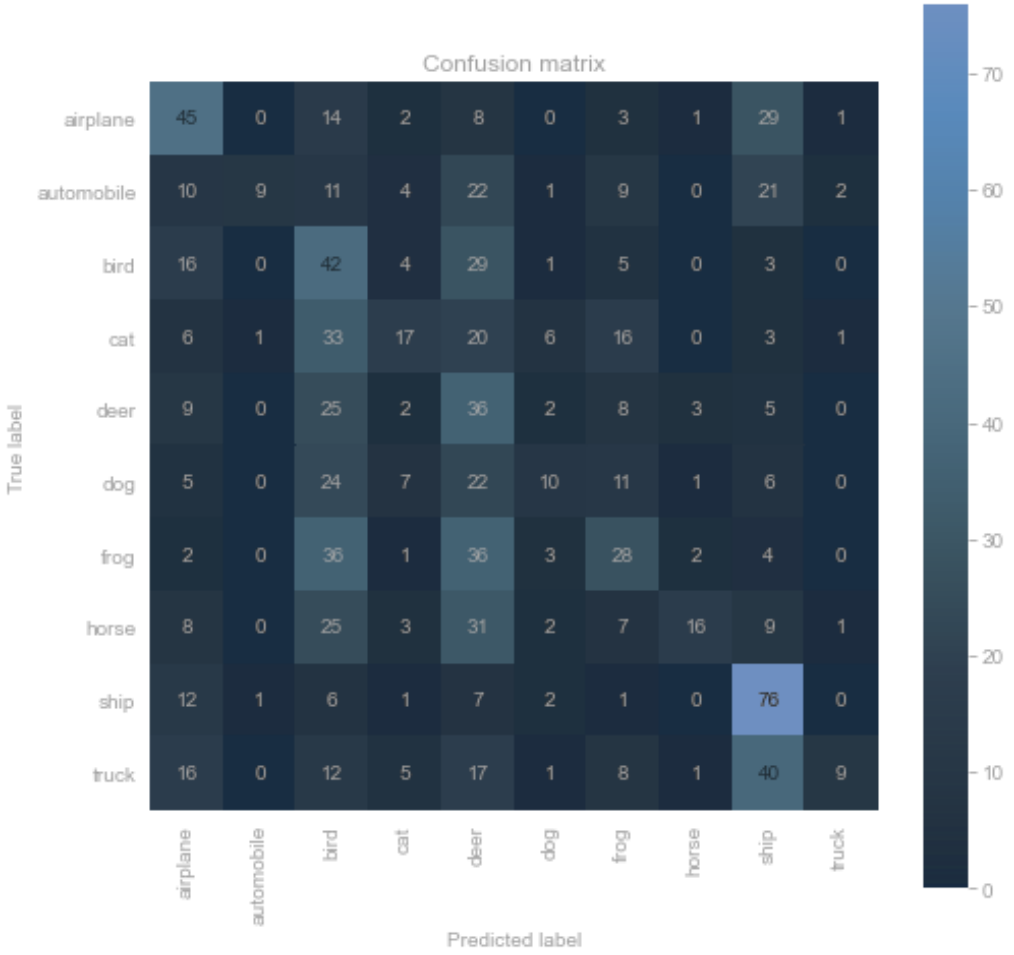
We loaded the dataset and transform the images to tensors then created a dataloader for the training set and displayed 5 images .

2. we built a K-Nearest Neighbors (K-NN) classifier , loaded 10k images from the train dataset and trained the model using fit method

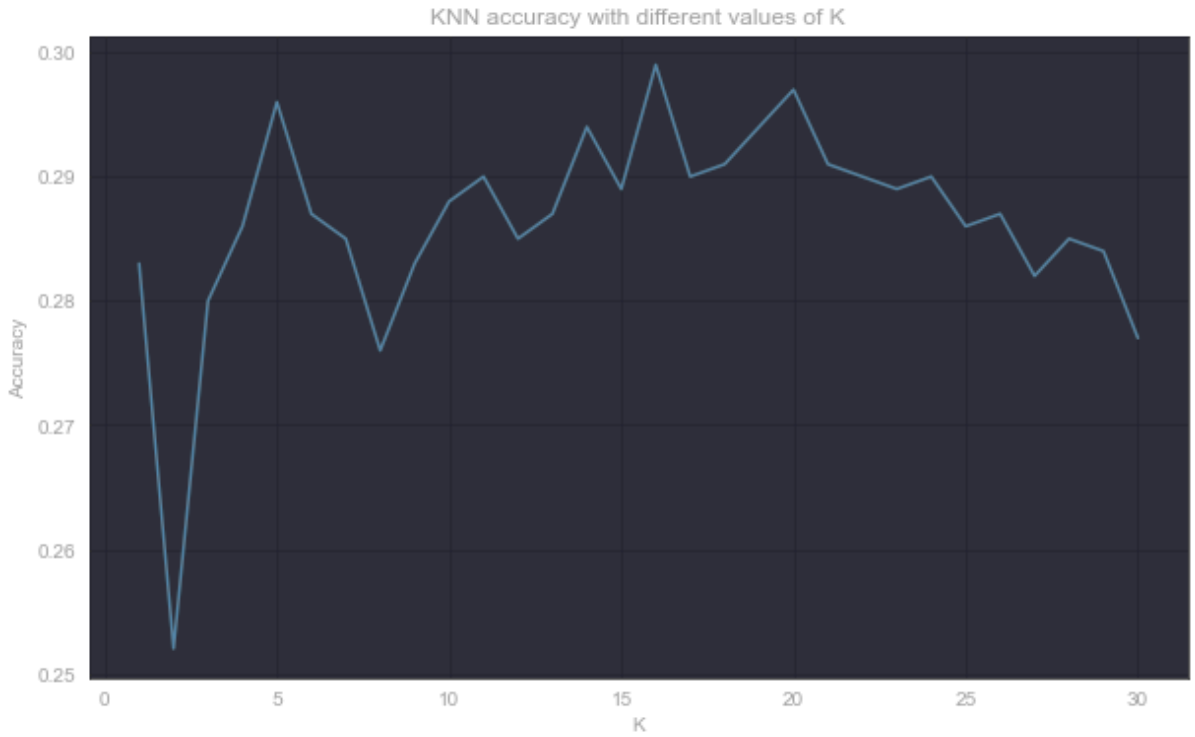
3. we loaded 1000 images from the testset and checked the model performance by creating y_preds using predict

method , we got Test accuracy: 0.2880 very poor results.

We received the folowing confusion matrix



4.

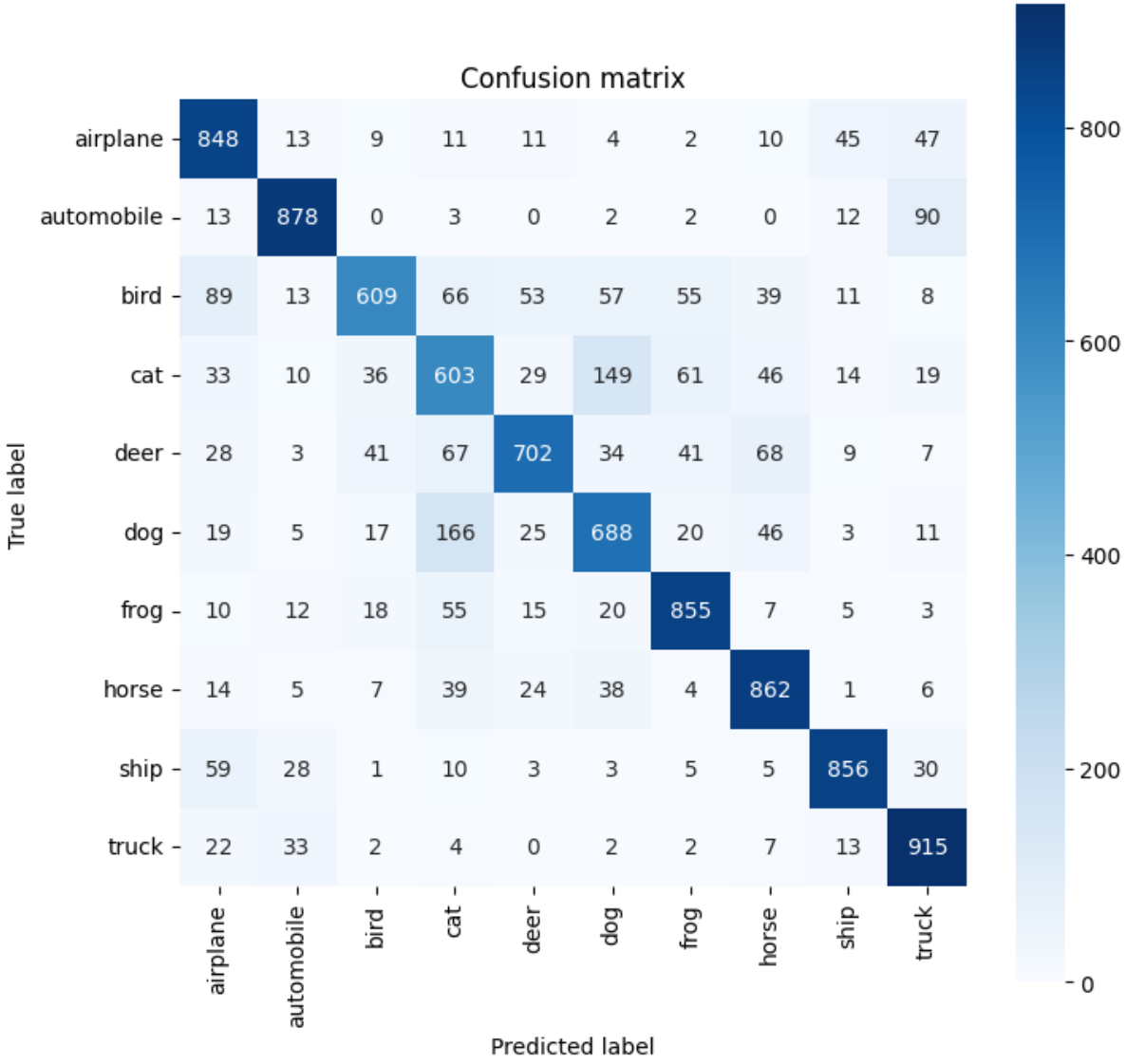


We used a for loop to iterate range 1,31 of k , at each iteration we initialized a model with k neighbors, trained the model and made prediction , we can see that the accuracy is small for all values of k , we can conclude that KNN classifier do not perform well on the dataset which might be to complex.

PART 2:

1.We created the SvhnCNN model from the tutorial trained and test is those are the results:

```
test accuracy: 78.160%
```



we can see that the cats,dogs and birds are the most confusing classes to the model .

2.

We created the DaniellaShaharNet it’s similar to the model from the tutorial but it is deeper and we added 4 residual blocks(where written as a separate class) that uses skip connections , each residual block apply 2 convolutions with kernel size 3 at the direct path and a convolution with kernel 1 at the skip path to keep features from former layers.

Architecture description:

- Input size: (batch_size, 3, height, width), where 3 is the number of input channels (RGB), and height and width are the dimensions of the input image.
- Convolutional layers: The network starts with a convolutional layer with 32 output channels, followed by batch normalization and leaky ReLU activation function. Then, a residual block with 32 input channels and 64 output channels is used, followed by another leaky ReLU activation function. The process is repeated with a convolutional layer with 128 output channels and a residual block with 128 input channels and 256 output channels. Then, a dropout layer is used to prevent overfitting. A residual block with 256 input channels and 512 output channels is used, followed by another max pooling layer. Finally, a residual block with 512 input channels and 256 output channels is used.
- Fully connected layers: After the convolutional layers, the output is flattened and passed through two fully connected layers, with 64 and 10 output neurons, respectively. The final layer produces the output of the network, which is a vector of size (batch_size, 10), representing the predicted class scores.
- Activation functions: Leaky ReLU is used as the activation function in all the convolutional and fully connected layers.

- Dropout: Dropout with a probability of 0.5 is used after the second max pooling layer to reduce overfitting.

We chose residual block because created a deeper network with more convolutions layers and Residual blocks are used in convolutional neural networks (CNNs) to help address the problem of vanishing gradients during training. Vanishing gradients occur when the gradient becomes very small during backpropagation, making it difficult for the network to learn.

We also used Leaky ReLU instead of ReLU since it can help prevent the "dying ReLU" problem, which occurs when a large number of neurons in a network become inactive (i.e., produce zero output) due to their inputs falling below zero. LeakyReLU with slope of -0.1 instead of ReLU.

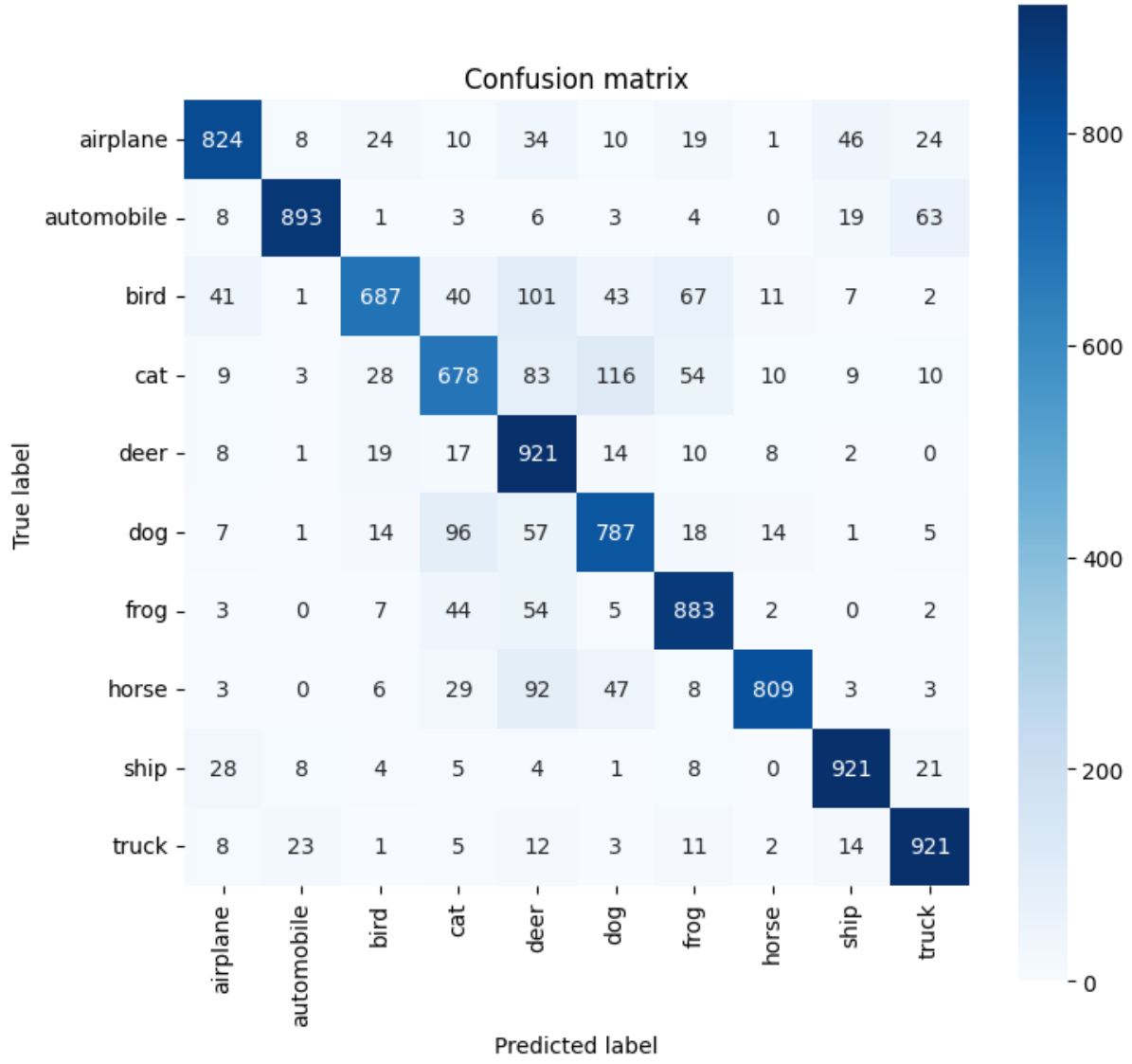
We kept the out_channels=128 from the conv layers since the dataset images are pretty small , this way we also reduce the input of the FC which can take to much space at the GPU memory otherwise.

And we calculated the input shape of the FC we got torch.Size([5, 128, 6, 6]) at the output of the conv layers then this was flatten and we got 128*6*6 = 4608 features entering the FC layer. Our FCs contains 2 layers similar to the model from the tutorial.

Number of parameters in the model: 6956874

We ran the model with the same hyperparameters as in the tutorial and received better results on the test:

```
test accuracy: 83.240%
```



3.

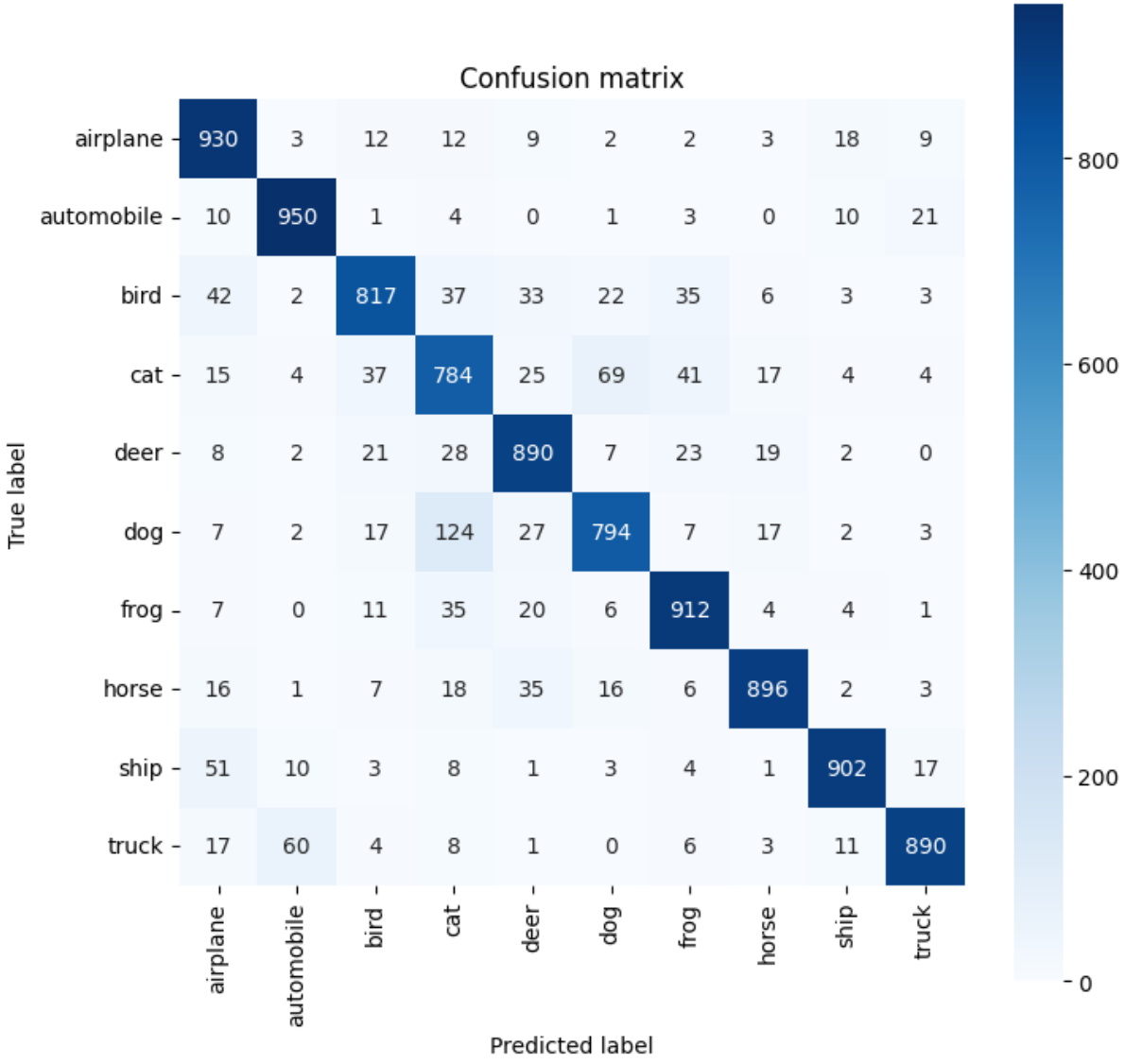
Then we created validation set to tune the hyperparameters We chose to adjust the learning rate and the batch size.

We trained the model with different lr and different batch size and checked the results in the val set , we kept the best hyperparameters and then initialize new model , trained it with the chosen hyperparameters and test it, since this code runtime was too long we marked out this part in our code and manually wrote the parameters. those are the results , we chose batch_size=64 and lr=1e-3

And run the training loop for 60 epochs.

And got better results:

```
test accuracy: 87.650%
```



PART 3:

1.we loaded the pre-trained VGG-16 model and put it in evaluation mode.

2.we wrote a function to load all jpg images inside a folder as PIL Image object.

Used it to load the birds folder :


3.we wrote a function to preprocess the images to fit to the VGG-16 , the steps we take:
Resize the images to 224x224 pixels.
Convert the images to the RGB color space.
Normalize the pixel values based on the mean and standard deviation of the ImageNet dataset- using mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225].

4. we opened the text file and saved the labels as an array, then we forward the images through the network and used soofmax to convert the output to probabilities then we took the argmax of the probabilities and the labels at this index is the predicted class label .

We got :

Predicted label: 94: 'hummingbird',

Probability: 0.7039

Predicted label: 90: 'lorikeet',

Probability: 0.6463

5.

We downloaded image of a cute dog preprocessed it and forward to the network similar to what we did at the last section and we got :

6.

We plotted the 3 first filters of the first layer of the VGG-16,

and their response (their output) for the image from section 5.

We used the code from the tutorials and made a few adjustments to display the first layer and only 3 filters.

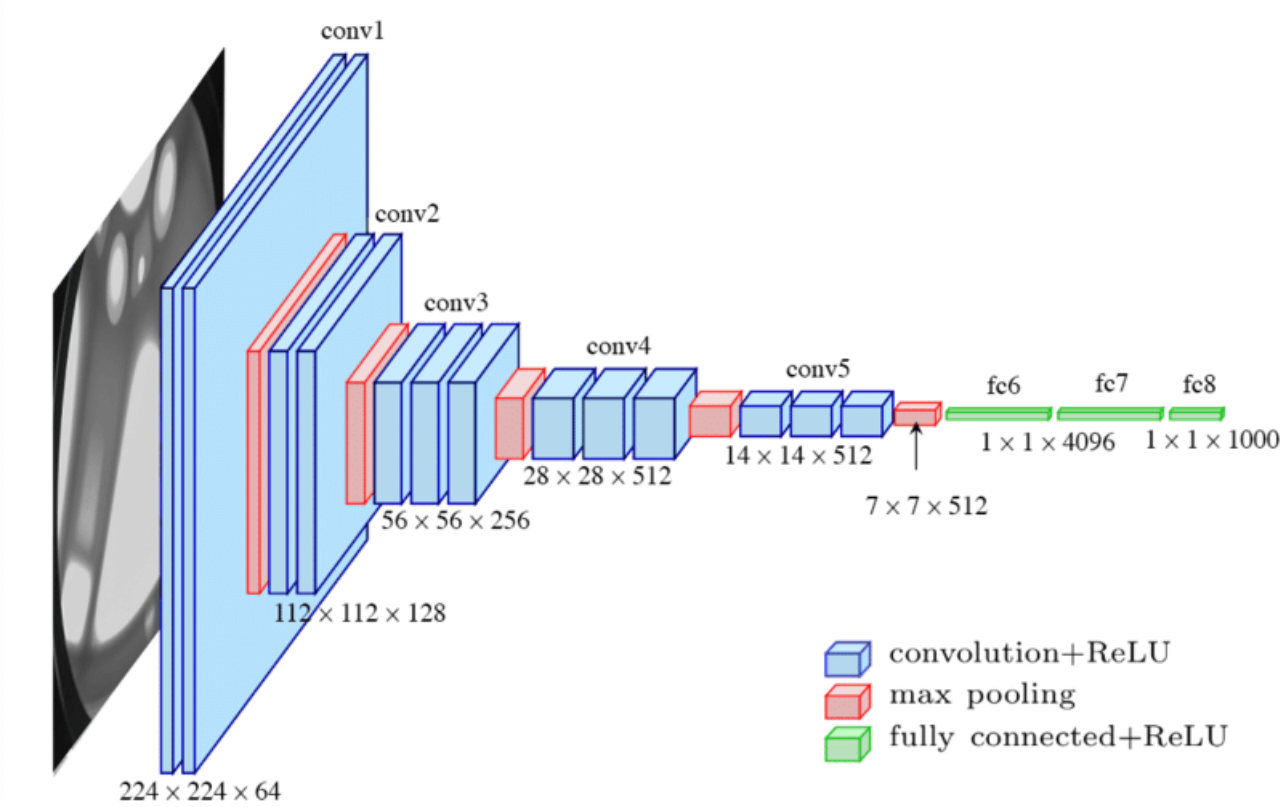
We can see that each filter is 3*3 as the conv kernel size the images are the results of the convulotion with the image , we can see that each filter put more importance to different texture at the image but all got the “shape” of the dog since this is the first layer, if we would have done it for the last layer we might view more delicate features.

7.

We loaded all images from the two folders using load_images

Function we wrote .

We created a function to forward an image through the VGG-16 net without the last FC which called FC8



The way we do that is we created new model that contains all conv layers as VGG-16 and the FC contains all layers except the last one index -1 ,

new_classifier = nn.Sequential(*list(model.classifier.children())[:-1])
model_copy.classifier = new_classifier

then we forward each image through the new network and stack all output to a single tensor .

so we picked FC7 and the features space size is :

torch.Size([4096])

8.

We built SVM classifier , we created X_train which is the concatenation of the preprocessed dogs and cats images features(first all the dogs and then all the cats)

Y_true_train is [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] so dogs are 1 and cats are 0.

We trained the SVM model using fit method.

Then we took 4 images for test set and preprocessed them and ran them one by one thought the all layers off the model till FC7 to extract 4096 features for each image. we used those feature

To predict with the SVM and got :

The image was classified as: cat

The image was classified as: dog

The image was classified as: dog

100% accuracy on test set.

9.

We chose this cow image from the internet:

We forward it through the VGG network similar to what we did in the last sections and the cow was classified as an ox we checked and there is no “cow” label in the txt file so ox is probably the correct label.

10.

We augmented the image and created 3 different images:

- rotated by 45 degrees
- Changed space color to YCrCb
- Blured – convolved with a Gaussian filter

Our images where PIL Image objects so we converted them to Numpy array , applied the augmentation using CV2 and then transformed back to PIL Image(to match the object type required in the functions we wrote earlier)

11.

We preprocessed each image and forward through the VGG-16 network in a similar way as we did in last sections and we got :

We can see that 2 out of 3 images where not classified correctly.

We deduct that the model is not robust to rotation , It is possible that it was not trained with enough rotated data so the model is sensitive to changes in orientation or perspective of an image , it is also possible that the model was not trained with cow images standing in this uphill angle .

It is possible that VGG-16 was trained on RGB images, and not with YCrCb color space.Thus,the model isn’t able to handle this change in color space, and thus failed to recognize the object in the image.

It seems that the model gave weight to the pinkish colors of this image probably at the training process most pinkish images where real Volcano leading to the incorrect classification as a volcano.

Blurring the image can help reduce noise and emphasize certain features in an image. In your case, blurring may have made the edges of the cow image less prominent and distinguishable, but the shape of the cow is still there and the colors are of a natural habitat of a cow so the VGG-16 model was able to recognize the cow and classify it correctly.

We were wondering what would have happened If we test the model with image of a cow that is not in it’s natural habitat , will he be able to classified it ?

The image was classified as: sandbar, sand bar

The image was classified as: cliff, drop, drop-off

Seems It ignores the cow..