Project Summary

Students

Matan Yarin Shimon – 314669342

Yarin Hindi - 313357337

Background

In this project we tried the approach of turning the images into grayscale (the grayscale.ipynb notebook), in the presentation professor Lee-Ad told us to try the approach of keeping it RGB (the RGB.ipynb notebook), and we added another notebook that changes the labels into general Animal or Vehicle and predicts it (Animal vs Vehicle.ipynb).

We also added a more detailed confusion matrics conclusion after Lee-Ad told us to do so (you can see it here under the "**Confusion matrics conclusion**" section).

DataSet representation

CIFAR-10 dataset. Contains 60,000 32x32x3 (RGB) images in 10 classes, with 6000 images per class. There are 50,000 training images and 10,000 test images.

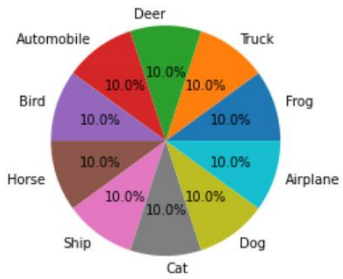
The classes are: Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship and Truck.

Main goal

Predict the class label of a given image according to the pixels value it has.

1) Notebook we have presented to Liad (plus liad comments):

Data distribution



As we can see, our data is perfectly balanced.

Models prediction

We took two approaches, turning the pixels into grayscale and without.

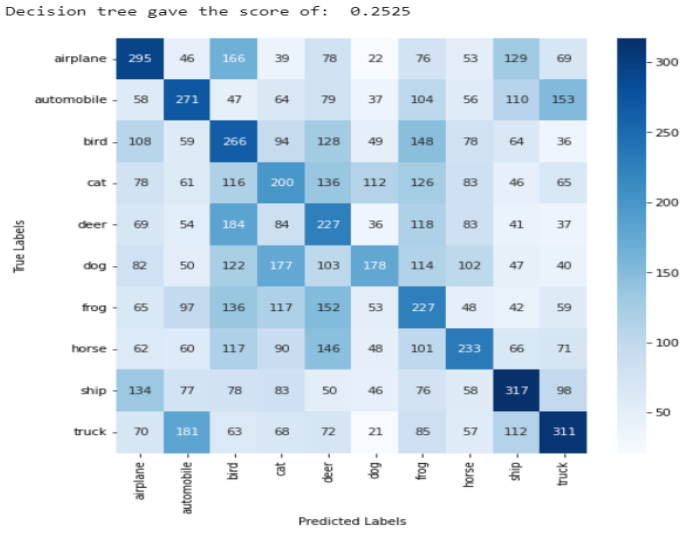
Decision tree

Challenges

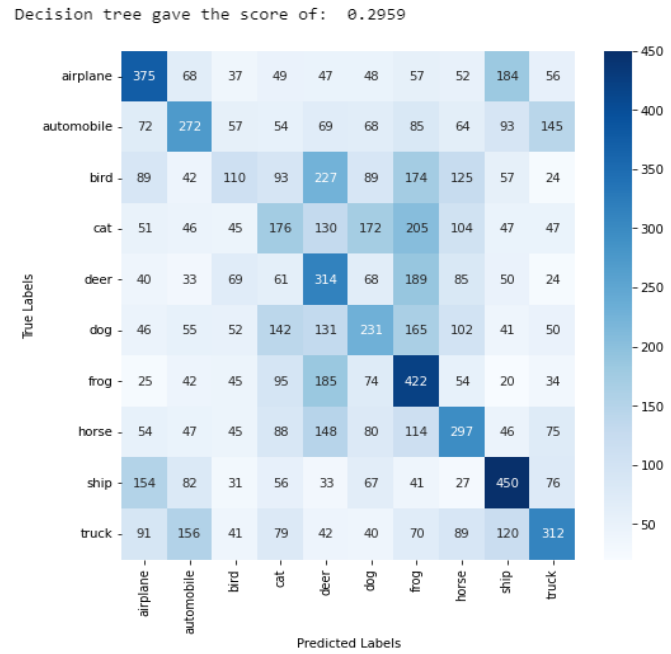
First we ran the model on it's default parameters and the model just kept running without stopping. To overcome this problem we changed the parameters of this model and tried to maximize the prediction outcome.

Eventually, we saw that the max depth of 10 gave us the best score.

grayscale



RGB



AdaBoost

Challenges

First we didn't know which base estimator and how many num estimators will maximize out score predictions. For knowing this we ran the model on several different parameters for the base estimators.

Eventually we saw that from 250 estimators and more, the model gave lower score and for that reason we stopped at 200 estimators in our final run.

grayscale

```
AdaBoost with max depth:  5  and n_estimators:  50  gave the score of:  0.3219
AdaBoost with max depth:  10  and n_estimators:  50  gave the score of:  0.2608
AdaBoost with max depth:  5  and n_estimators:  100  gave the score of:  0.3245
```

Note: In the cell block above we ran the ada boost on our data, this cell takes a lot of hours to run. We already ran it and acciedently ran it again and this is the reason this is the cell that does'nt have it's final outcome. The outcome of this was around 0.3 ~ 0.33.

RGB

```
AdaBoost with max depth:  5  and n_estimators:  50  gave the score of:  0.3636
AdaBoost with max depth:  10  and n_estimators:  50  gave the score of:  0.2993
```

Note: In the cell block above we ran the ada boost on our data, this cell takes a lot of hours to run. We already ran it and acciedently ran it again and this is the reason this is the cell that does'nt have it's final outcome. The outcome of this was around 0.3 ~ 0.35.

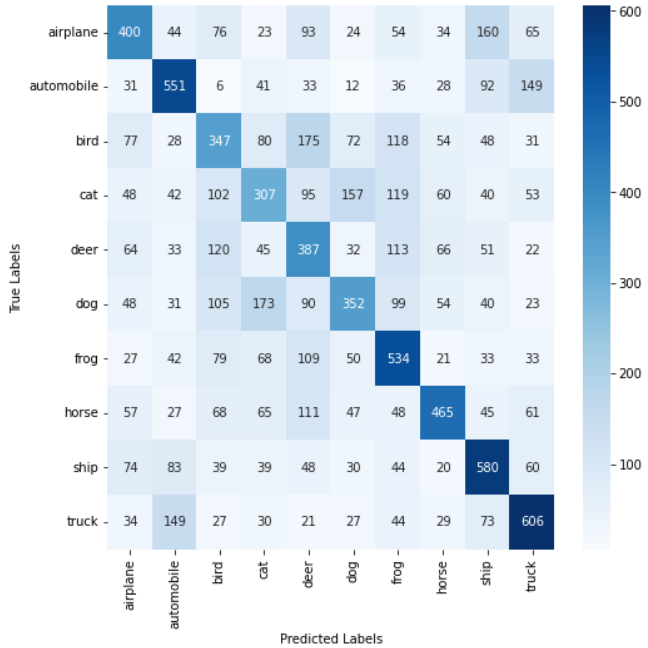
SVM

Challenges

To make the SVM fit our data, we saw that we need to observe some different kernel function and see which one will give us the best score.

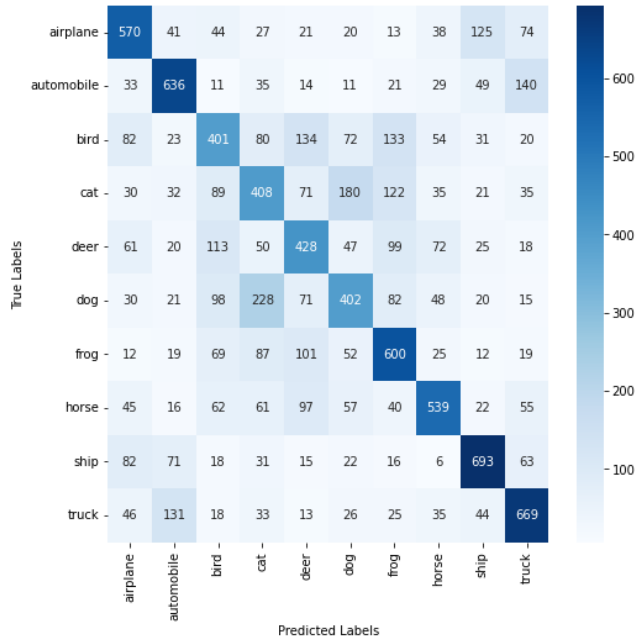
grayscale

```
linear  kernel gave the score of:  0.1504
poly  kernel gave the score of:  0.3545
rbf  kernel gave the score of:  0.4529
kernel:  rbf  gave the best score of:  0.4529
```



RGB

```
linear  kernel gave the score of:  0.2508
poly  kernel gave the score of:  0.4544
rbf  kernel gave the score of:  0.5346
kernel:  rbf  gave the best score of:  0.5346
```



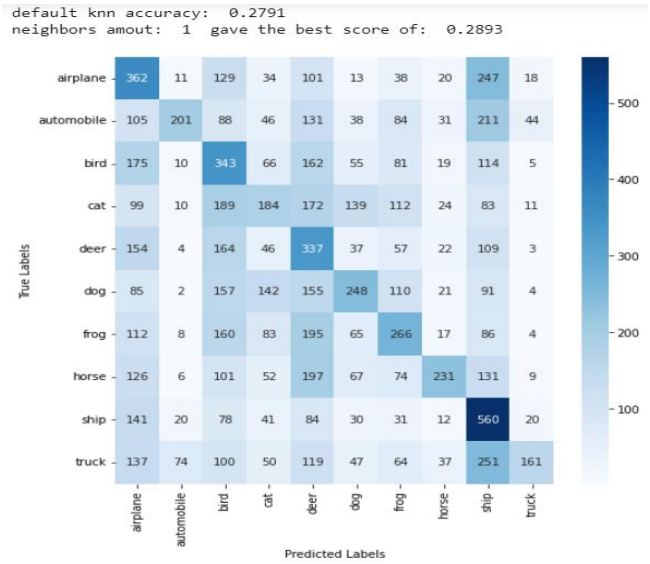
KNN

Challenges

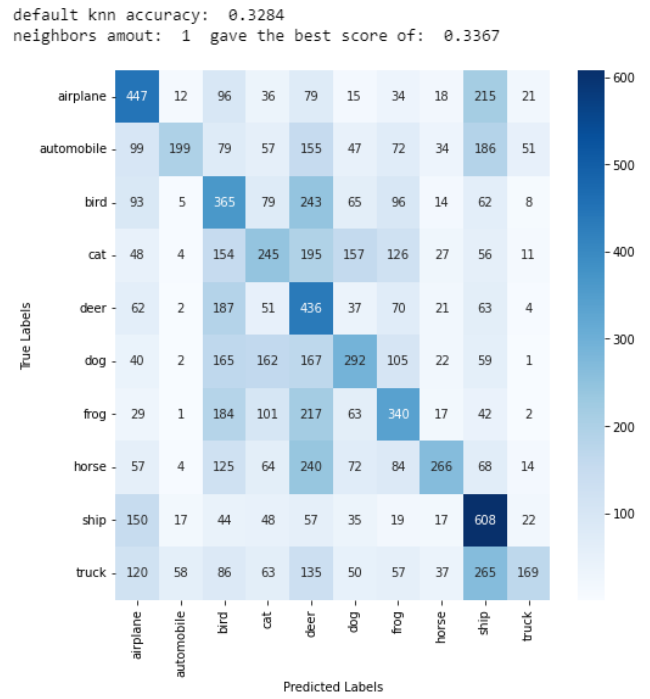
For maximizing the KNN we need to check what number of neighbors will give us the best predictions, for that reason we ran on several number of neighbors, and recieved the number of neighbors that gave us the best score.

Eventually we saw that from 11 neighbors and more, the prediction gets worse, and for that reason we stopped checking in 9 neighbors in our final run.

Grayscale

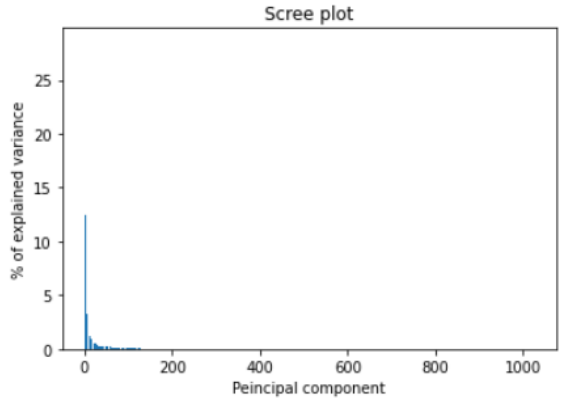


RGB



PCA

Each image has a lot of pixels, by that our model can be really complex and may cause overfitting. let's see if we can improve our predictions while using dimentional reduction (PCA).



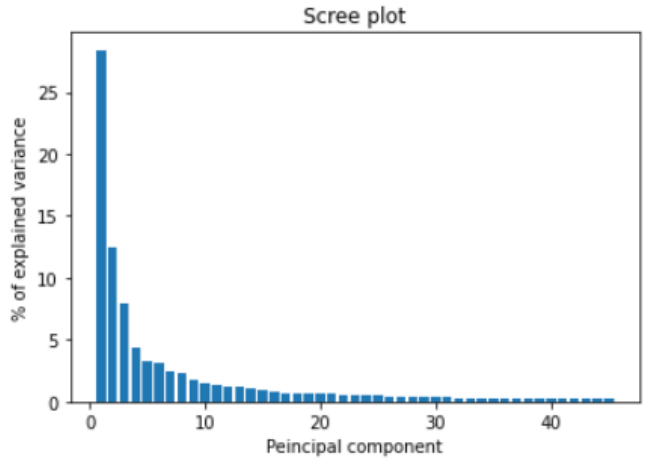
As we can see from the Scree plot above, there's alot of pixels that are not playing a significal part in the label prediction.

For that reason we will perform dimentional reduction

To get a 85% of our data accuracy, we need only 45 columns.

Let's check the variance percentage in each of the 45 pixels

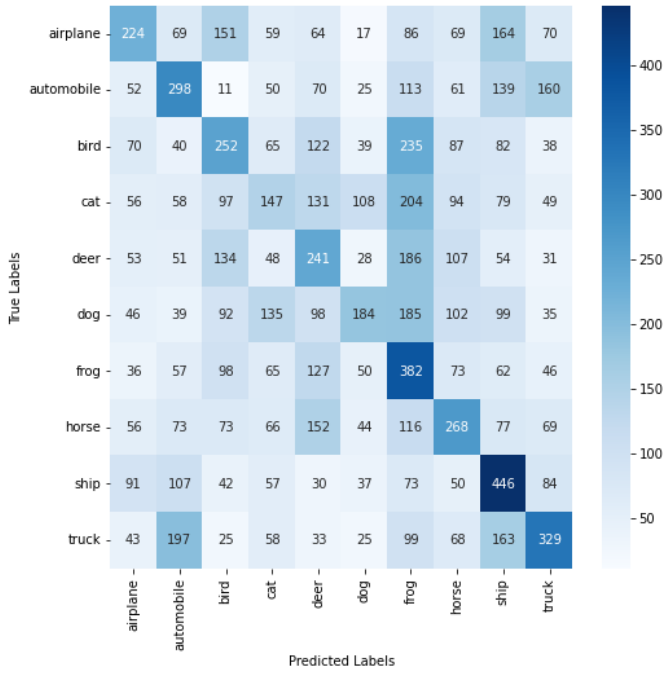
```
per_var = np.round(pca.explained_variance_ratio_*100, decimals=1)
plt.bar(x=range(1,len(per_var)+1), height=per_var)
plt.ylabel('% of explained variance')
plt.xlabel('Peincipal component')
plt.title('Scree plot')
plt.show()
```



Now the Scree plot looks way better.

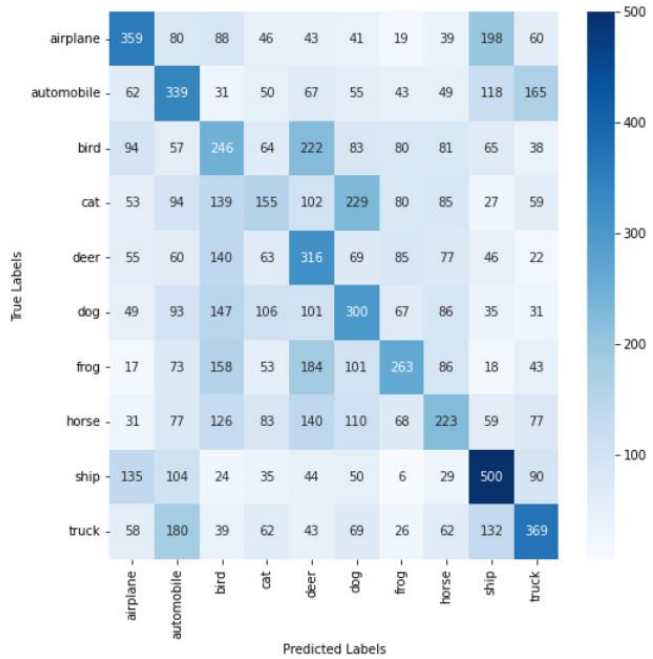
Decision tree on the reduced data (grayscale)

```
Decision tree gave the score of:  0.0967
```



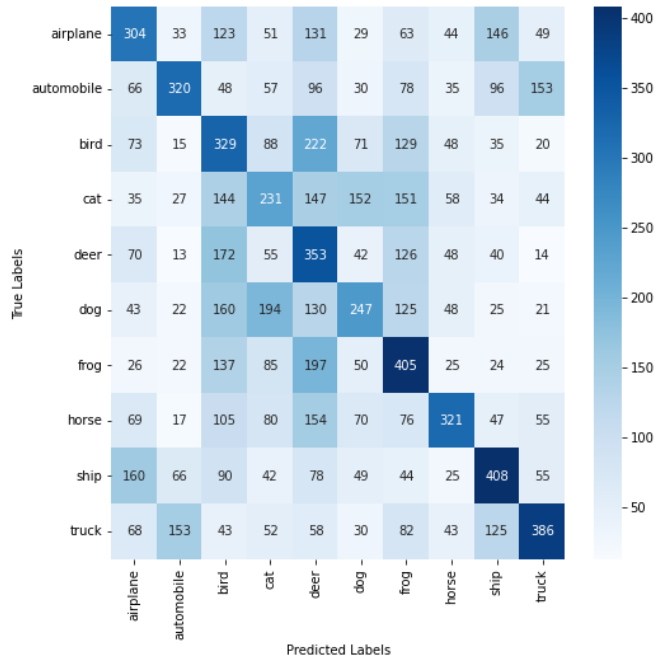
Decision tree on the reduced data (RGB)

```
Decision tree gave the score of:  0.0983
```



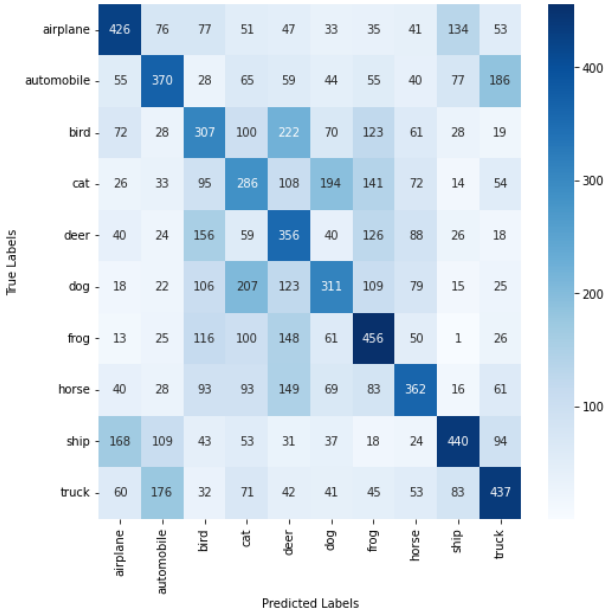
AdaBoost on the reduced data (grayscale)

```
AdaBoost with max depth:  5  and n_estimators:  50  gave the score of:  0.3206
AdaBoost with max depth:  10  and n_estimators:  50  gave the score of:  0.2622
AdaBoost with max depth:  5  and n_estimators:  100  gave the score of:  0.3141
AdaBoost with max depth:  10  and n_estimators:  100  gave the score of:  0.296
AdaBoost with max depth:  5  and n_estimators:  150  gave the score of:  0.305
AdaBoost with max depth:  10  and n_estimators:  150  gave the score of:  0.3164
AdaBoost with max depth:  5  and n_estimators:  200  gave the score of:  0.296
AdaBoost with max depth:  10  and n_estimators:  200  gave the score of:  0.3304
AdaBoost with depth:  10  and n_estimators:  200  gave the best score of:  0.3304
```



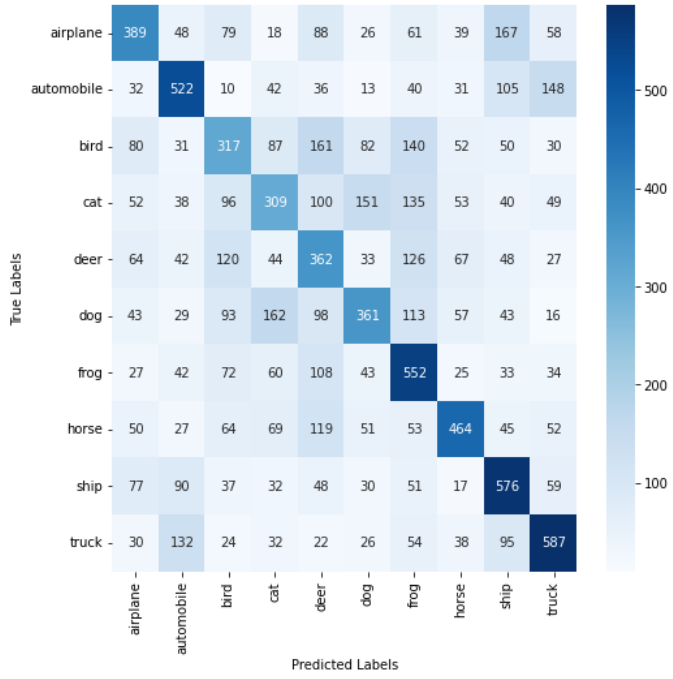
AdaBoost on the reduced data (RGB)

```
AdaBoost with max depth:  5  and n_estimators:  50  gave the score of:  0.337
AdaBoost with max depth:  10  and n_estimators:  50  gave the score of:  0.3119
AdaBoost with max depth:  5  and n_estimators:  100  gave the score of:  0.3267
AdaBoost with max depth:  10  and n_estimators:  100  gave the score of:  0.3497
AdaBoost with max depth:  5  and n_estimators:  150  gave the score of:  0.327
AdaBoost with max depth:  10  and n_estimators:  150  gave the score of:  0.3678
AdaBoost with max depth:  5  and n_estimators:  200  gave the score of:  0.326
AdaBoost with max depth:  10  and n_estimators:  200  gave the score of:  0.3751
AdaBoost with depth:  10  and n_estimators:  200  gave the best score of:  0.3751
```



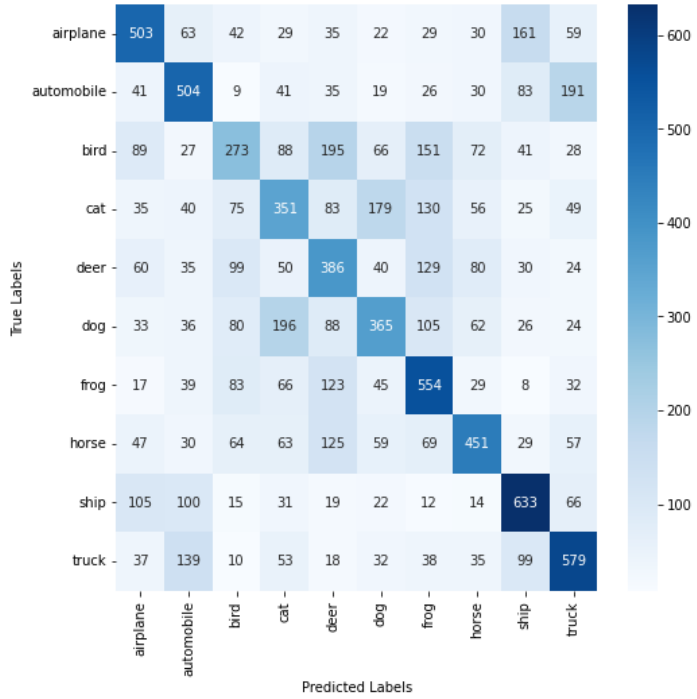
SVM on the reduced data (grayscale)

```
linear  kernel gave the score of:  0.1347
poly  kernel gave the score of:  0.362
rbf  kernel gave the score of:  0.4439
kernel:  rbf  gave the best score of:  0.4439
```



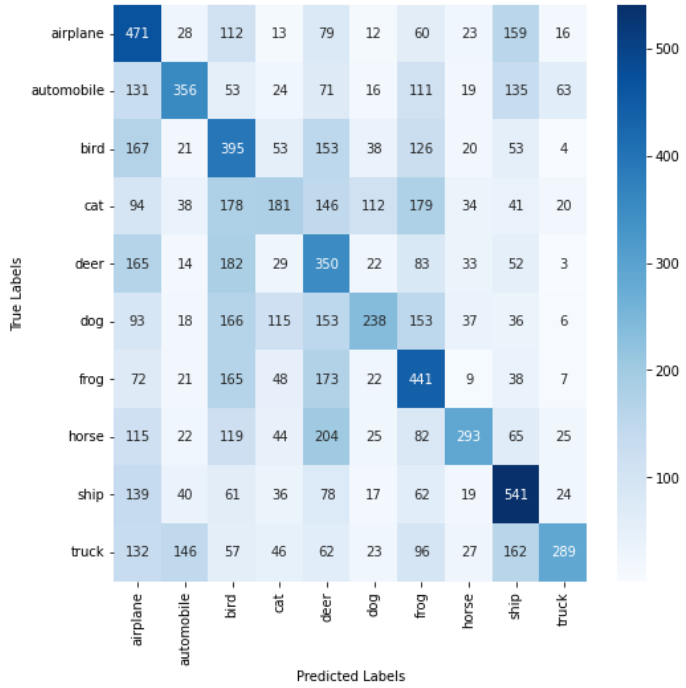
SVM on the reduced data (RGB)

```
linear  kernel gave the score of:  0.1529
poly  kernel gave the score of:  0.4057
rbf  kernel gave the score of:  0.4599
kernel:  rbf  gave the best score of:  0.4599
```



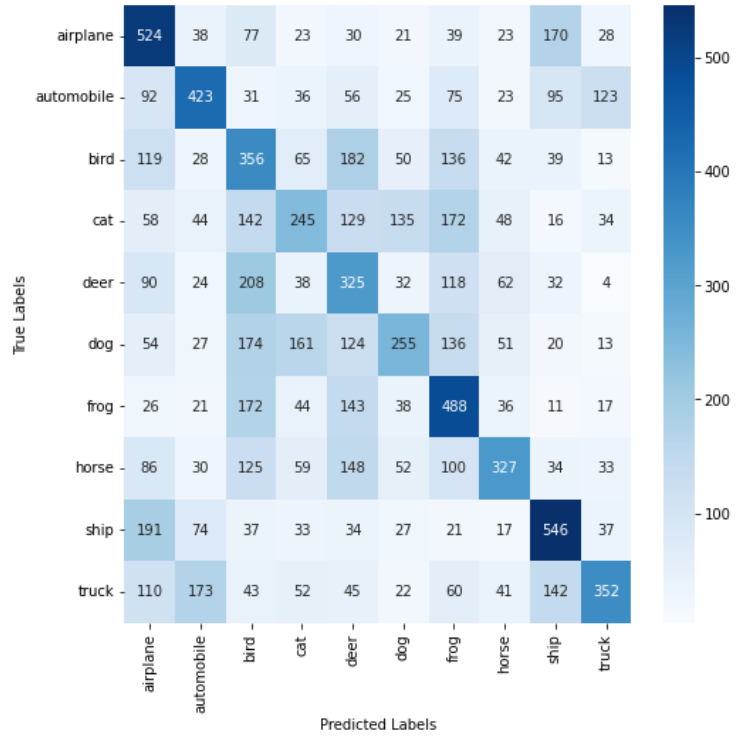
KNN on the reduced data (grayscale)

```
default knn accuracy:  0.3478
neighbors amout:  9  gave the best score of:  0.3555
```



KNN on the reduced data (RGB)

```
default knn accuracy:  0.3764
neighbors amout:  9  gave the best score of:  0.3841
```



Models conclusion

The best score so far we have accomplished was made by the SVM, by using rbf kernel (without the PCA) that gave the score of: 0.4529 (grayscale) and 0.5346 (RGB).

Why we received such lower performances?

<u>DecisionTree</u>

It may be that the decision tree is not suitable for this particular dataset and that the hyperparameters of the model need to be tuned further than the one we performed.

<u>Adaboost</u>

Maybe the reason Adaboost didn't perform well on our data is that we didn't use the right base estimator for this mission.

<u>SVM</u>

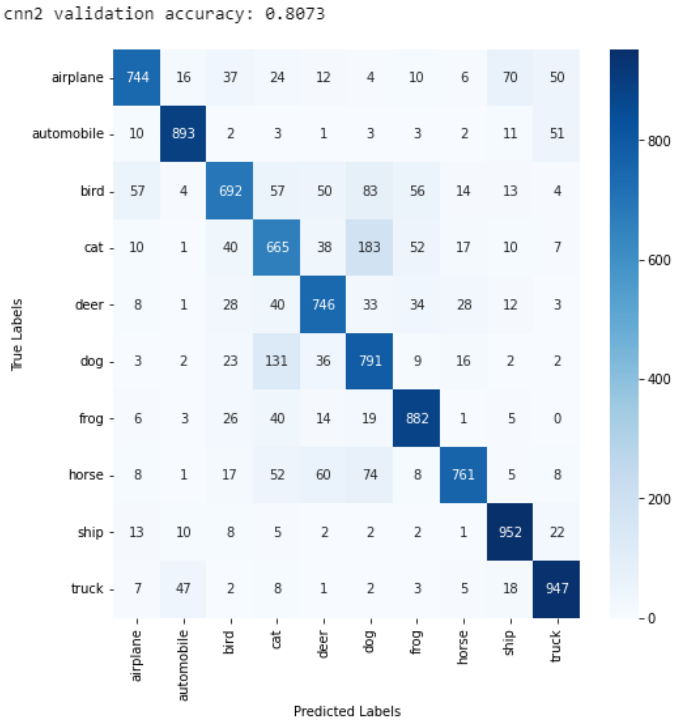
SVM works well on datasets with a clear margin of separation, it could be that our data is highly overlapping or noisy.

<u>KNN</u>

One possible reason that KNN didn't perform well on our data is that the dataset is high-dimensional, and KNN tends to perform poorly in high-dimensional spaces.

Let's see if we can beat that score with a model we have not learned.

CNN



As we can see, the CNN gave us the best score of: 0.8073 and for that reason, we will predict the test by using this model.

Why did CNN perform well?

One possible reason could be that our dataset consists of images, and CNNs are known to perform well on image classification tasks because they can extract meaningful features from the raw pixel data.

Confusion matrics conclusion

We can see even though our model has a high prediction score, it predicted a lot of cats as dogs (183) and vice versa (131), this means that the cat and the dog label classes have a lot of common features, which works out logically.

In the same way for the vehicles, it predicted a lot of automobiles as trucks (51) and vice versa (47), this means that the automobile and the truck label classes have a lot of common features, which works out logically.

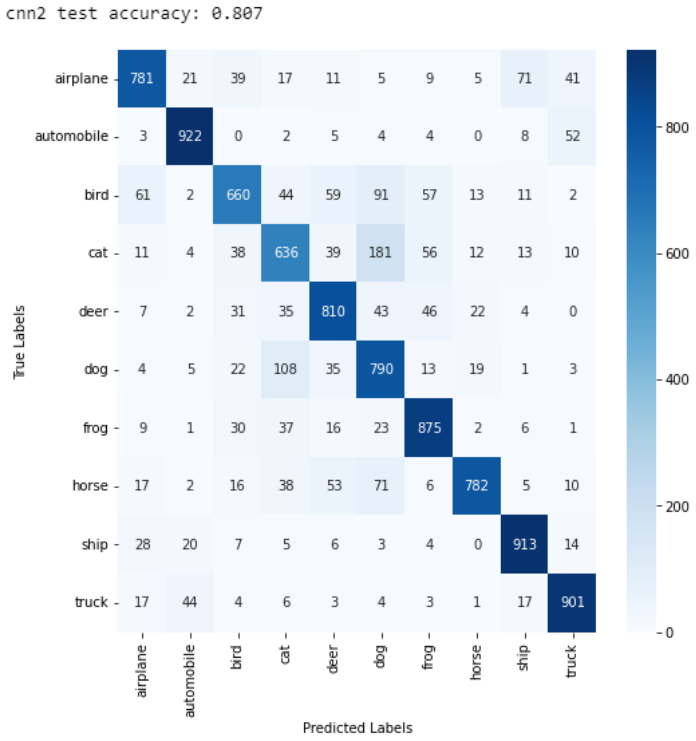
It works out logically that there's a higher confusion between the cats and dogs rather than the automobiles and trucks because the cats and dogs share a higher number of common features.

Moreover, we can see that generally, we predicted a lot of animals as other animals and a lot of vehicles as other vehicles.

While predicting a low percentage of vehicles as animals, and vice versa.

For that reason, we performed another full notebook of predicting our data based on the labels of Animal / Vehicle (see "**Notebook we added for showing the confusion matrics conclusions**" section).

Predicting the test



Final conclusion

1. While using data that stays as RGB and didn't move to grayscale we got better scores.
2. The model that gave us the best score is CNN.

2) **Notebook we added for showing the confusion matrics conclusions**:

We won't show here all of the models predictions for convenience (you can see everything in the notebook itself we added).

Our models prediction was great (around 0.8 – 0.85), and the best of them was also the CNN which gave a 0.95 score, let's present the test prediction: