Group 30 Date: 12/07/2019

ECEP 596, Autumn 2019 Homework 4: Final Project

Exploring machine learning models for object classification

Group Members: Sairam Tabibu
Mukund Bharadwaj
Niraj Porecha

TABLE OF CONTENTS

1. Introduction

The proposed project is a variation of “Option 2: Object Recognition” suggested for the assignment. The aim of the project is to apply different interest operators and Machine Learning algorithms and compare their performance in multiclass object classification. Further, we would like to study the variation in accuracy due to boost and ensemble operations in comparison to naive ML algorithms.

2. Related work

2.1. Convolutional Neural Networks

Deep Convolutional Neural Networks (CNNs) are a special type of Neural Networks, which have shown state-of-the-art performance on various competitive benchmarks. The powerful learning ability of deep CNN is largely due to the use of multiple feature extraction stages (hidden layers) that can automatically learn representations from the data. Availability of a large amount of data and improvements in the hardware processing units have accelerated the research in CNNs, and recently very interesting deep CNN architectures are reported[6,7,8]. The recent race in developing deep CNNs shows that the innovative architectural ideas, as well as parameter optimization, can improve CNN performance. In this regard, different ideas in the CNN design have been explored such as the use of different activation and loss functions, parameter optimization, regularization, and restructuring of the processing units[9,10,11].

2.2. Random Forests

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.[3] It has become a commonly used tool in multiple prediction scenarios due to their high accuracy and ability to handle large features with small samples. Random Forest combines the two concepts of Bagging and Random Selection of Features by generating a set of T regression trees where the training set for each tree is selected using Bootstrap sampling from the original sample set and the features considered for partitioning at each node is a random subset of the original set of features. For each node, the optimal node splitting feature is selected from a set of m features that are picked randomly from the total M features. For $m \ll M$, the selection of the node splitting feature from a random set of features decreases the correlation between different trees and thus the average response of multiple regression trees is expected to have lower variance than individual regression trees. Larger m can improve the predictive capability of individual trees, but can also increase the correlation between trees and void any gains from averaging multiple predictions.

The bootstrap resampling of the data for training each tree also increases the independence between the trees.[2]

2.3. Support Vector Machines

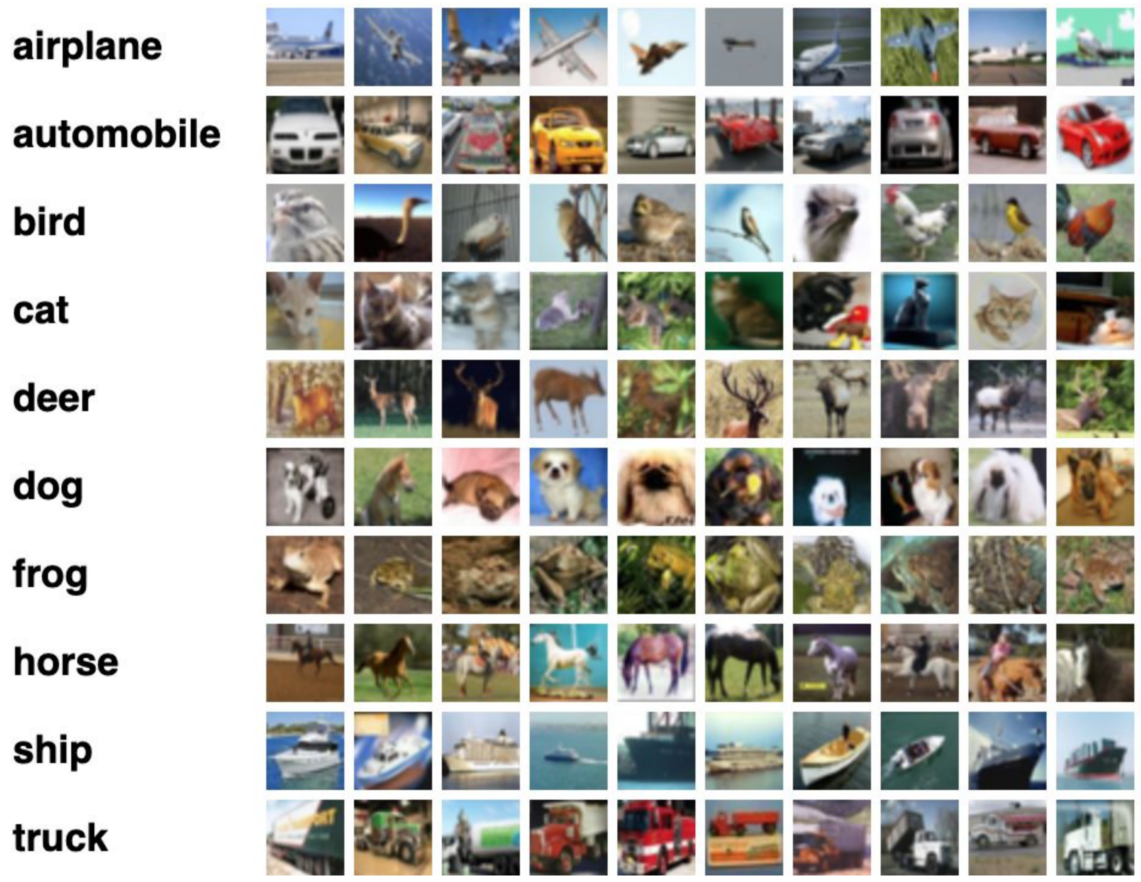
Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection. The basic idea of support vector machines is just like 1-layer or multi-layer neural nets. SVMs calculate an optimal hyperplane for linearly separable patterns. They extend to patterns that are not linearly separable by the transformations of original data to map into new space - the Kernel function. Support vectors are the data points that lie closest to the decision surface (or hyperplane). They are the data points most difficult to classify and have a direct bearing on the optimum location of the decision surface.

3. Dataset - CIFAR10:

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Below are the classes in the dataset, as well as 10 random images from each:



The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

The dataset downloaded was a version for python. Each file consisted of a pickled dictionary containing the following key-value pairs:

- **data** -- a 10000x3072 numpy array of *uint8*s. Each row of the array stores a 32x32 colour image. The first 1024 entries contain the red channel values, the next 1024 the green, and the final 1024 the blue. The image is stored in row-major order, so that the first 32 entries of the array are the red channel values of the first row of the image.

- **labels** -- a list of 10000 numbers in the range 0-9. The number at index *i* indicates the label of the *i*th image in the array **data**.

The dataset contains another file, called *batches.meta*. It too contains a Python dictionary object. It has the names of each class, stored as a list.[5]

4. Experiments and results

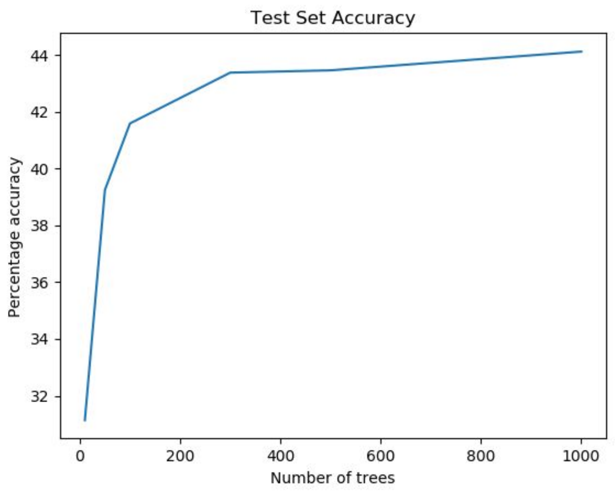
4.1. Data Processing

For the machine learning models used apart from the CNNs, input images are fed as a single channel flattened array. While colour is an important aspect of classification, we decided to use a grayscale image instead, depending on the shape and texture of the objects. The grayscale image used consisted of converting the 3-channel RGB image into YUV colour space and using the Y channel (illumination) as the input to the image. This essentially made the 3x32x32 image provided in the dataset into a 1x1024 feature set. The Standard Scaler was used from Scikit-Learn to produce a dataset with no mean and scaled to unit variance. While this would be sufficient to train the models, it is resource intensive and difficult for hyper-parameter tuning. Instead Principal Component Analysis (PCA) was used to reduce the number of feature sizes to reduce the complexity of model training while retaining the most important features. Experimenting with different values of transformed components, it was found that a model number of ~35 was sufficient to capture the majority of features with less than 0.1% accuracy loss. Therefore, all Random Forest and SVM models (which showed performance increases in some cases) were therefore trained on these components instead.

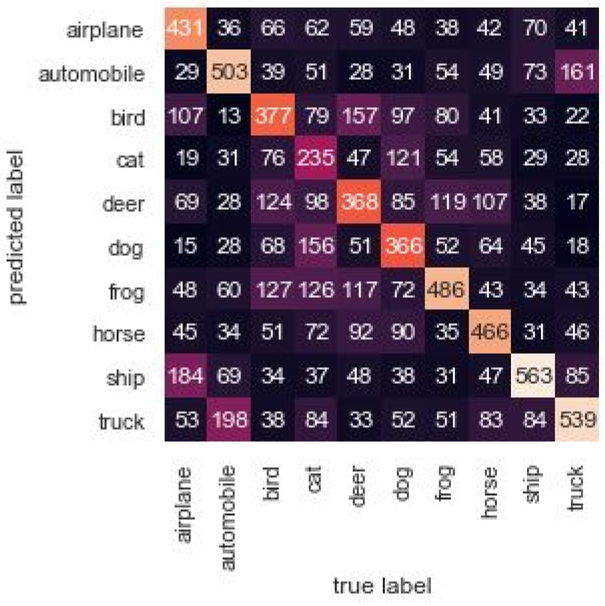
4.2. Random Forest

The classifier was implemented using the implementation available on Scikit-Learn. Literature available typically suggest that Random Forests are not easily prone to overfitting due to the large number of trees used as well as the bootstrapping method of growing each individual tree and that the default parameters usually yield good results. However, during implementation, it was seen that the models trained had overfit to the training data while showing much poorer scores on the testing set. To counter this effect, the default parameters were modified to reduce overfitting while improving testing accuracy. To this effect, it was found that increasing the number of trees (n_estimators) and decreasing the maximum sample pool available to build each new tree (max_samples) yielded the best results. Similarly, Scikit Learn does not use out of bag scores to compute the performance of the model by default which was also enabled to provide a better measure of the model.

Initially, multiple random forests models were trained with varying number of trees. It can be seen from the graph that the number of trees makes a big difference when using a small number of trees. However, after crossing a few 100s of trees, it can be seen that the returns of increasing the number of trees reduces. The maximum accuracy achieved was 42.35%



Variation in Test Set Accuracy with number of trees



Confusion Matrix for a Random Forest Model using 1000 trees
and maximum sample of 80% of training set for building each tree

4.3. Support Vector Machine (SVM)

We used scikit learn for the implementation of SVM for Object Classification. Various parameters like *C, gamma, kernels* are used for tuning the SVM.

The *gamma* parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'.

The *C* parameter trades off correct classification of training examples against maximization of the decision function's margin. or larger values of *C*, a smaller margin will be accepted if the decision function is better at classifying all training points correctly. A lower *C* will encourage a larger margin, therefore a simpler decision function, at the cost of training accuracy.

The *kernel function* can be any of the following:

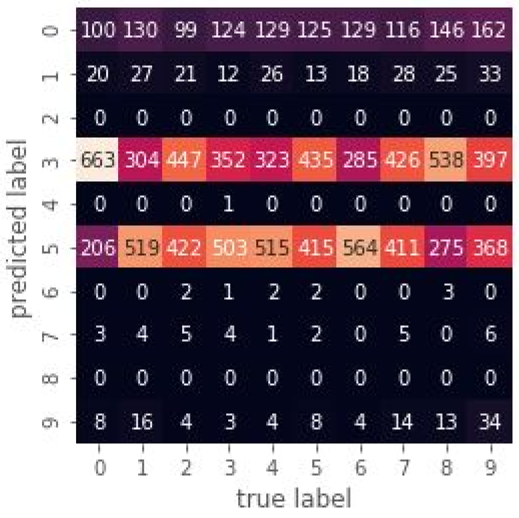
- Sigmoid
- Linear
- Polynomial
- Radial Basis Function (rbf)

By training the SVM model using all of the above kernels, Accuracy for each was about 12-15%. Thus, further we tuned the *C* and *gamma* parameters to increase the accuracy and achieved the following results (gamma=0.1, C-1 was found to be optimal after tuning):

gamma=0.1, C=1	**Kernel**	**Accuracy**
SVM	Sigmoid	13.00%
	Linear	27.40%
	Polynomial (deg=4)	41.03%
	RBF	45.61%

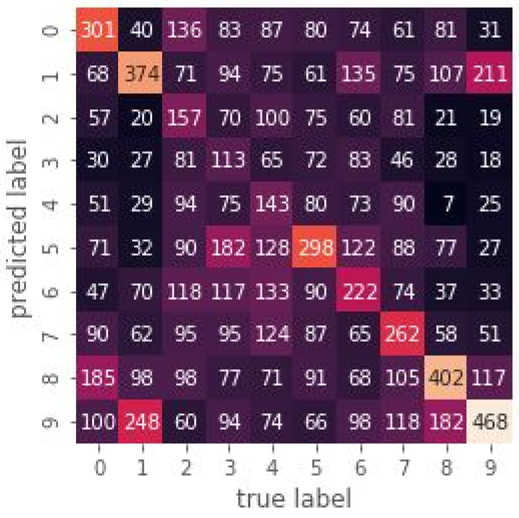
- Confusion Matrices for each kernel

Sigmoid:



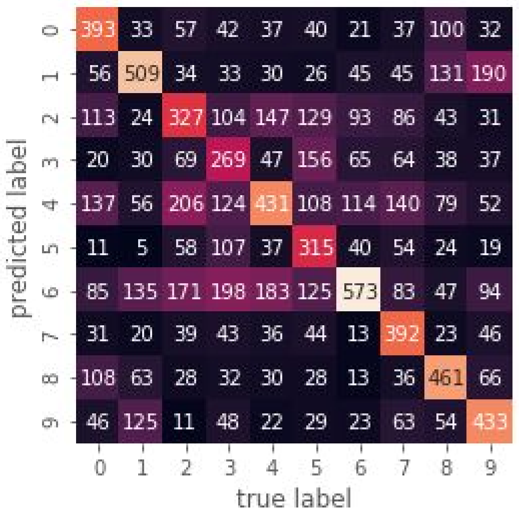
	0	1	2	3	4	5	6	7	8	9
0	100	130	99	124	129	125	129	116	146	162
1	20	27	21	12	26	13	18	28	25	33
2	0	0	0	0	0	0	0	0	0	0
3	663	304	447	352	323	435	285	426	538	397
4	0	0	0	1	0	0	0	0	0	0
5	206	519	422	503	515	415	564	411	275	368
6	0	0	2	1	2	2	0	0	3	0
7	3	4	5	4	1	2	0	5	0	6
8	0	0	0	0	0	0	0	0	0	0
9	8	16	4	3	4	8	4	14	13	34

Linear:



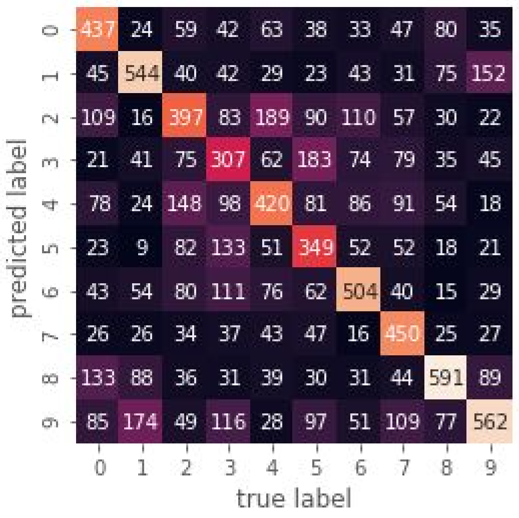
	0	1	2	3	4	5	6	7	8	9
0	301	40	136	83	87	80	74	61	81	31
1	68	374	71	94	75	61	135	75	107	211
2	57	20	157	70	100	75	60	81	21	19
3	30	27	81	113	65	72	83	46	28	18
4	51	29	94	75	143	80	73	90	7	25
5	71	32	90	182	128	298	122	88	77	27
6	47	70	118	117	133	90	222	74	37	33
7	90	62	95	95	124	87	65	262	58	51
8	185	98	98	77	71	91	68	105	402	117
9	100	248	60	94	74	66	98	118	182	468

Polynomial:



	0	1	2	3	4	5	6	7	8	9
0	393	33	57	42	37	40	21	37	100	32
1	56	509	34	33	30	26	45	45	131	190
2	113	24	327	104	147	129	93	86	43	31
3	20	30	69	269	47	156	65	64	38	37
4	137	56	206	124	431	108	114	140	79	52
5	11	5	58	107	37	315	40	54	24	19
6	85	135	171	198	183	125	573	83	47	94
7	31	20	39	43	36	44	13	392	23	46
8	108	63	28	32	30	28	13	36	461	66
9	46	125	11	48	22	29	23	63	54	433

RBF:



	0	1	2	3	4	5	6	7	8	9
0	437	24	59	42	63	38	33	47	80	35
1	45	544	40	42	29	23	43	31	75	152
2	109	16	397	83	189	90	110	57	30	22
3	21	41	75	307	62	183	74	79	35	45
4	78	24	148	98	420	81	86	91	54	18
5	23	9	82	133	51	349	52	52	18	21
6	43	54	80	111	76	62	504	40	15	29
7	26	26	34	37	43	47	16	450	25	27
8	133	88	36	31	39	30	31	44	591	89
9	85	174	49	116	28	97	51	109	77	562

4.4. Convolutional Neural Network (CNN)

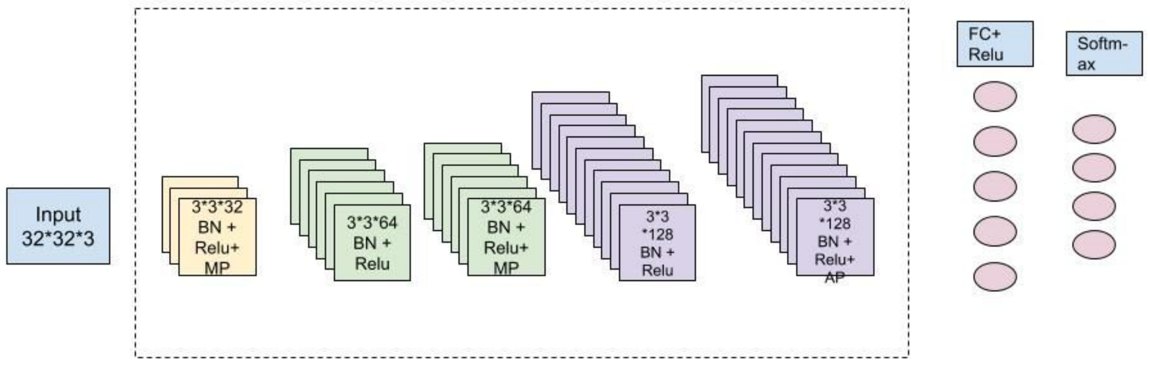
Convolutional neural networks (CNN, or ConvNet) is a class of Deep Neural Networks, most commonly applied to analyzing visual imagery for tasks such as classification, segmentation, tracking etc.

We used a custom 5 layers CNN for training our data and experimented on 4 different types of architectures varying the number of channels in several layers.

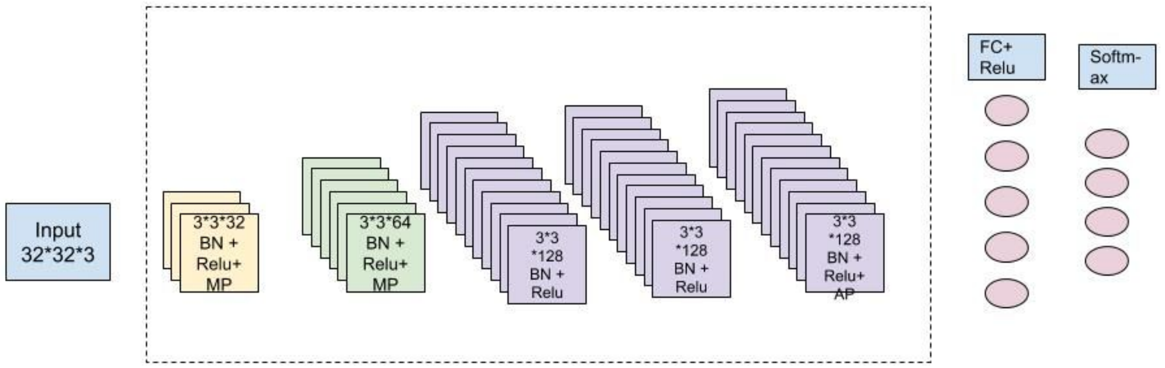
We kept the training parameters for these custom CNN's similar as shown below. We also performed transfer learning on popular architecture **Resnet18** and compared it with our custom architectures.

<u>**Training parameters**</u>
Learning Rate - 0.1 (*0.5 after every 10 epochs)
Optimiser - SGD optimiser
Momentum - 0.9
Loss function - Cross Entropy Loss
Data Augmentation - Horizontal flip and Random crop
Regularization weight - 0.005
Batch Size - 64

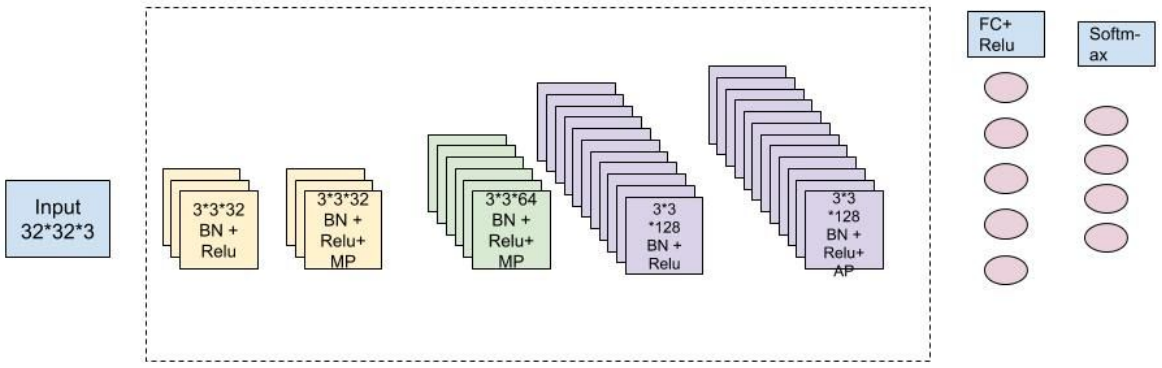
CNN Architecture 1



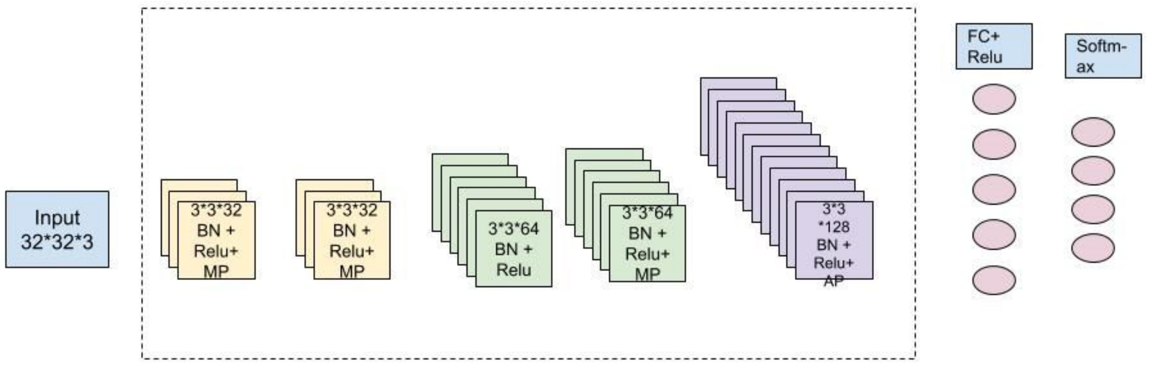
CNN Architecture 2



CNN Architecture 3



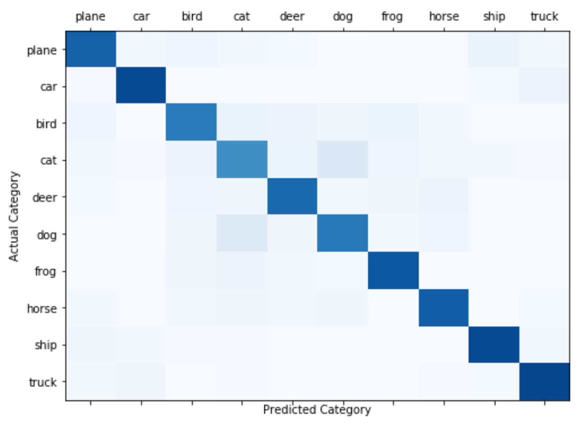
CNN Architecture 4



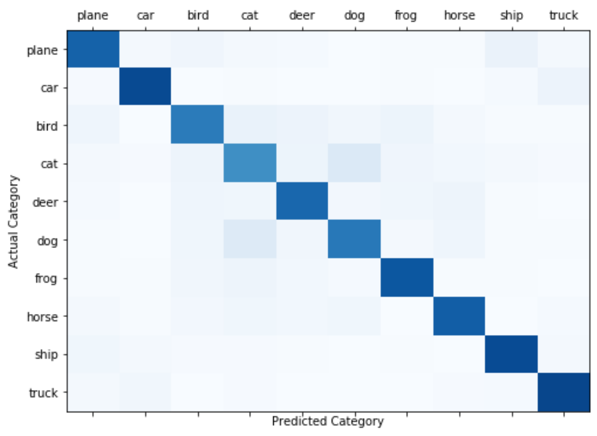
- **CNN Results:**

Category	Accuracy CNN1	Accuracy CNN2	Accuracy CNN3	Accuracy CNN4	Accuracy Resnet-18
Plane	80.9	80.8%	77.3%	74.0%	94.9%
Car	90.4	89.8%	91.2%	87.4%	96.7%
Bird	73.7	71.0%	66.7%	63.9%	90.9%
Cat	61.7	63.4%	68.1%	58.9%	85.3%
Deer	76.3	78.8%	78.7%	72.2%	94.0%
Dog	74.3	72.0%	69.0%	64.2%	90.2%
Frog	84.5	85.1%	84.0%	80.6%	96.6%
Horse	84.0	82.4%	83.6%	80.0%	94.5%
Ship	90.1	89.2%	90.6%	84.5%	96.1%
Truck	89.9	91.1%	86.5%	86.5%	95.4%
Total	**80.44%**	**80.36%**	**79.57%**	**75.22%**	**93.46%**

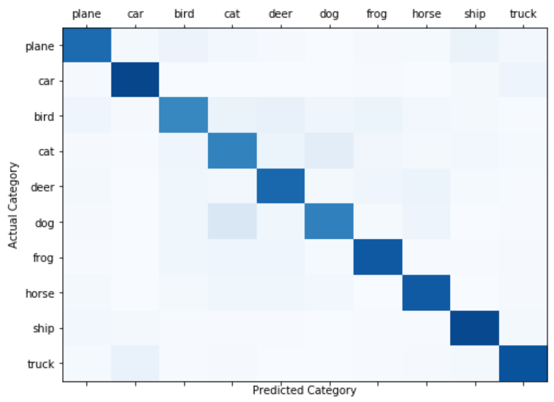
- **Confusion Matrices**



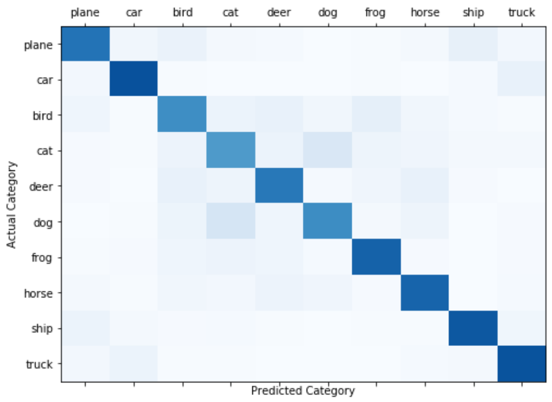
CNN1



CNN2



CNN3



CNN4

4.5. Analysis of the performance

Looking at the confusion matrices produced by the various models, there was one main observation seen. that cause classification to be harder between certain classes. Looking at the major errors found, all classifiers showed good ability of classifying at broad categorical level, i.e., they mostly struggle with classifying within animals, but not between animals and non-species. With Random Forests and SVMs, there was a noticeable error with the following:

- Airplanes being miss-classified as ships and birds
- Automobiles being classified as trucks
- Sub-animal categories especially between (also seen on CNNs)
    - Cat and Dog
    - Deer and Horse

These can be explained by their similar shapes and textures. For example, both cats and dogs are four-legged and furry. Similarly, airplanes and ships are typically made of shiny metals and have similar smaller windows.

From using various kernels in SVMs it can be observed that on CIFAR10, RBF kernel performs the best, and sigmoid performs the worst. This maybe due to the fact that SVM with sigmoid can be compared to a single layer perceptron NN.

In CNN's, when there are more shallow layers in the beginning of the network and less deep layers in the end the performance of the model performance tends to decrease. This shows that more discriminative features are learnt in the later layers.

5. Future work

One possible extension of this project can be to leverage the differentiating capabilities of different machine learning models by using an ensemble or cascading structure on the the of CNN or perform a decision level fusion.

6. Summary and Conclusion

In conclusion the CNN's outperforms all the other Machine learning models due to it's ability to learn complex features.

The Link to the Code - EE596-codes

7. References

1. The CIFAR-10 dataset https://www.cs.toronto.edu/~kriz/cifar.html
2. Pal, R., 2017, "Predictive Modeling of Drug Sensitivity", Pages 149-188
3. Scikit-learn, RandomForestClassifier, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
4. Scikit-learn, Principal Component Analysis, https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
5. Scikit-learn, Support Vector Machines, https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
6. He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.
7. Chatfield, Ken, et al. "Return of the devil in the details: Delving deep into convolutional nets." *arXiv preprint arXiv:1405.3531* (2014).
8. Szegedy, Christian, et al. "Going deeper with convolutions." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.
9. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929-1958.
10. Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014).
11. Agarap, Abien Fred. "Deep learning using rectified linear units (relu)." *arXiv preprint arXiv:1803.08375* (2018).