Image Classification

Enes Demirağ, Abdullah Enes Bedir, Kübra Neyiş

I. Introduction

IMAGE classification task is one of the most widely studied computer vision problems. Self-driving cars use image classification to identify what's around them such as trees, people, traffic lights etc. Image classification can also help in healthcare. For instance, it could analyse medical images and try to classify them as while helping healthcare workers to identify symptoms of illness. Or image classification could help people organise their photo collections.

There are many studies on this problem proposing different algorithms and structures. Early studies in this field mostly consist of mathematical models. With these analytical approximations and approaches, it was quite hard to come up with a generalized system which can classify multiple classes to each other effectively.

With the rise of the machine learning methods and the increasing accessibility to more digital images, better methods are proposed every year. These methods mainly consist of a dimension reduction, a feature extraction, and a classification part. As an example, machine learning methods such as KNN [1] and SVM [2] can be used on features from a lower dimensional space generated with the help of a dimension reduction algorith like PCA [3].

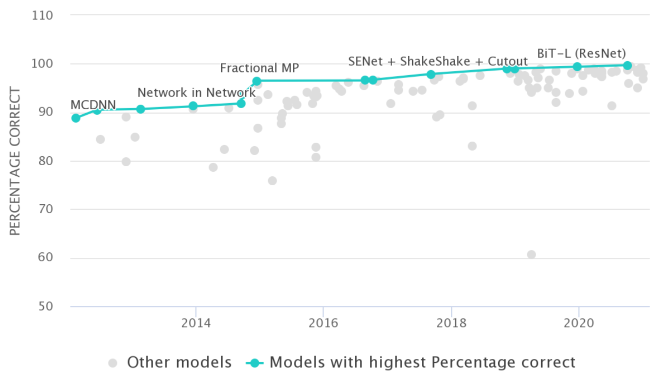


Fig. 1. Performance of different methods on CIFAR-10 dataset

After the convolutional neural network (CNN) architectures emerged [4], alongside with the other computer vision problems, they used on image classification works. Because of their performance on image data with the help of convolutional layers, it became possible to classify multiple classes by training them on large image datasets like CIFAR-10 [5]. With these supervised learning models, there was no longer a need for a seperate feature extraction step due to its capabilities.

In this study, we reviewed all these mentioned steps and gave results. Using Python programming language along side with scikit-learn [6] and tensorflow [7] library and the tools such as Kaggle and Google Colab we implemented multiple image classification models with variaty of methods. In continuing chapters, designing and training phases and the decisions and choses made will be explained.

A. Dataset

The datasets are important part of the Deep Learning models. The datasets are used to training and test the model. Main goal of the datasets is obtain best parameters in the model. The datasets are divides two main group generally, which are train and test set. In some cases, the validation set can be added to among these groups. The dataset needs to be as much as possible large and various.The variety is important concept of the network to prevent the overfitting in the model. To do the best test, The test set should be independent from training set.

For the project, we choose CIFAR-10 dataset [5]. The CIFAR-10 dataset includes 60000 color images. The shape of these images is 32x32. In the dataset, there are 10 classes and 6000 images for each class. In the dataset, the testset and trainingset are already divided as 10000 for test and 50000 for training. The images of the dataset vary as much as possible. The dataset is collected by Alex Krizhevsky, Vinod Nair and Geoffrey Hinton. Although the size of the dataset is small, the dataset being colorful reveals valuable features in the dataset. Also, since the dataset is suitable for MLP and CNN models, it is one of the best selection for this project.

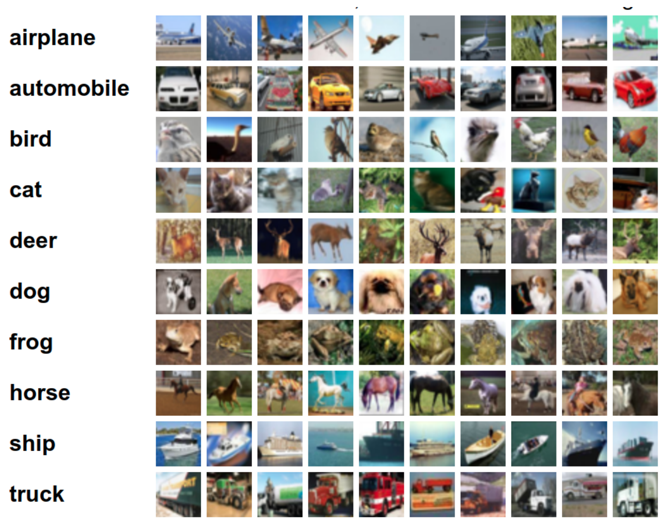


Fig. 2. Images from CIFAR-10 dataset [5]

The classes of the dataset are like in figure 2. The general information of the dataset is given in this section. In the project, when the validation data set is used, it is randomly selected from training set and test test.

II. Related Works

CIFAR-10 dataset is a quite popular dataset among computer vision tasks and commonly used in academic studies throughout years. Figure 1 shows rising of the state-of-the-art methods with respect to time.

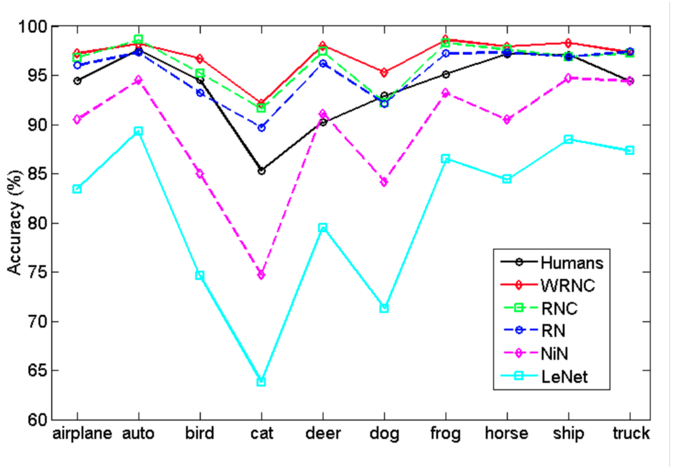


Fig. 3. Comparison of different models of accuracy by each class [8]

In a study, [8], recognition percentages of human versus the deep neural network models compared and results was proposed as in figure 3. Even though the recognition rate of the humans are quite good, it can be seen that this problem is quite challanging by looking the result of leNet [4] which was one ot the early convolutional neural networks. Of course, nowadays, there are models that achieve more than 99% accuracy in the image classification problem with this dataset.

Some of the studies stand out with their different approaches onto problem. Before trying to design and implement our own models, we studied many different academic papers as a literature review. Some of them were using traditional machine learning algorithms [1] [2]. And others were using more complex neural network based structures like MLP and CNN [9] [10] [11]. In the following chapters, these models will be explained deeply.

III. Traditional Image Classification Methods

In our study, before using artificial neural network models, we tried traditional machine learning algorithms that do not require training. After the feature extraction phase, we designed simple classifier models with widely used machine learning algorithms such as KNN and SVM using the histogram information of the images in training set. These histograms binned into 24 features to be able to simplify the input image with a lower dimensional input features.

A. K-Nearest Neighbors

The K-Nearest Neighbors (KNN) algorithm is a supervised machine learning algorithm that is simple and easy to implement. It is a method that can be used in the solution of both classification and regression problems. This algorithm assumes that the data points with the same labels are in close proximity in feature space. An example of 2D dataset can be seen in figure 4. Data points with the same labels are located at close distance to each other. KNN decides which class a new input data belongs to by calculating its distance from other known data points in the dataset.

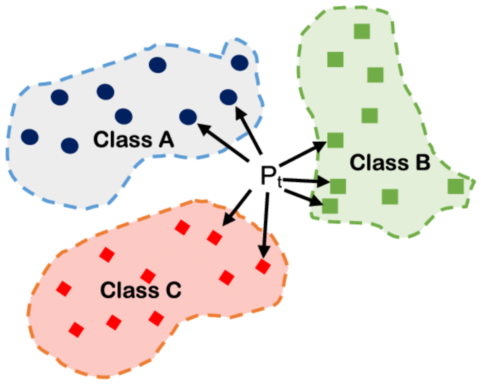
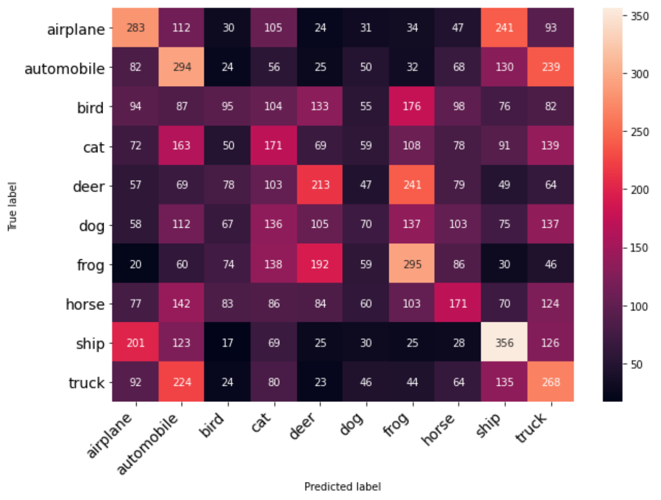


Fig. 4. 2D representation of KNN algorithm

To explain the KNN algorithm step by step, first, when a new input data comes, the distance to each existing data is calculated. This distance could be, for example, an euclidean distance which is the square root of the sum of every distances of dimensions. Then, the closest k samples are taken from the collection sorted according to these distances. Here, an odd number is usually chosen for the k parameter when working on classification problems. Afterwards, among the selected data points, which class has the most members, the new input is classified as the same class.



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	283	112	30	105	24	31	34	47	241	93
automobile	82	294	24	56	25	50	32	68	130	239
bird	94	87	95	104	133	55	176	98	76	82
cat	72	163	50	171	69	59	108	78	91	139
deer	57	69	78	103	213	47	241	79	49	64
dog	58	112	67	136	105	70	137	103	75	137
frog	20	60	74	138	192	59	295	86	30	46
horse	77	142	83	86	84	60	103	171	70	124
ship	201	123	17	69	25	30	25	28	356	126
truck	92	224	24	80	23	46	44	64	135	268

Fig. 5. Confusion matrix of our KNN model

In our implementation, first of all, we performed the feature extraction stage by getting histogram information of the images in HSV color space. Then using these features, we fit several different KNN models with different k values to achieve the best result. By examining the accuracy scores obtained with the test set, we determined the most appropriate value for our dataset as k = 25. After this value, the increase in the model accuracy was negligible. With this approach, we achieved 22.16% accuracy. Because of the simpicity of this model and the complexness of the problem KNN approach performed quite poorly as we guessed. You can see the confusion matrix of KNN in figure 5

B. *Support Vector Machine*

Support Vector Machine (SVM) is another popular supervised machine learning algorithm. The purpose of this method is to specify boundaries in n dimensional space that will divide classes from each other in the best way. In this algorithm, extreme data samples called support vectors are determined and a hyperplane is created to differentiate classes with the help of these support vectors. Any new input data is classified according to its location to these hyperplanes. The simplest SVM model is obtained using a linear hyperplane.

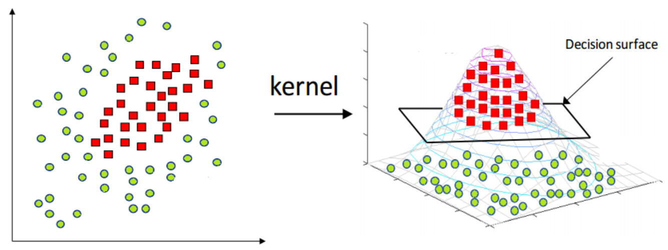


Fig. 6. SVM Kernel Trick

In the case of a linearly non-seperable dataset, using a linear hyperplane will not give sufficient results. To solve this problem, a method called kernel trick can be used. Kernel is a non-linear function that transforms a low dimensional input space into a higher dimensional space. It is possible to make a linearly non-seperable problem seperable with the kernel trick. An example of SVM kernel trick can be seen in figure 6. For example, the gaussian radial basis kernel function is given in equation 1.

$$k(x_i, x_j) = \exp(-\gamma \left\| x_i - x_j \right\|^2), \qquad \gamma = 1/2\sigma^2 \tag{1}$$

In our case, we used radial basis function (RBF) as a kernel function. Using the same features like KNN, we utilized SVM classifier model and got slightly better accuracy. Our achieved accuracy with SVM model is 23.71%. But as predicted, compared to a neural network model, these results are quite poor due to the simplicity of the methods.

IV. Neural Network Based Image Classification Methods

In this section, methodology under MLP and CNN network architectures explained and our designed models presented.

A. *Multi Layer Perceptron*

Perceptron is a machine learning algorithm that classifies an input. Perceptronperforms this classification with a linear function which uses weight and bias values.With this feature, perceptron forms the basis of all neural networks.

$$Out = \sum_{i=1}^{N} I_i \omega_i + b_i \tag{2}$$

Perceptron consists of 4 basic parts as input value, weight, bias and activation function. As can be seen, the innovation that perceptron adds to neural networks is that the bias value is added to each process as shown in 2.

As was the case when neural networks were first launched, the perceptron could not achieve the expected success due to technical limitations. There are 2 types of Perceptron models which are called single-layered perceptron model and multi-layered perceptron model. It contains a feedforward network within the single-layer perceptron model. It is the simplest neural network that can be implemented. An illustration of a Feed-Forward network structure is shown in 7

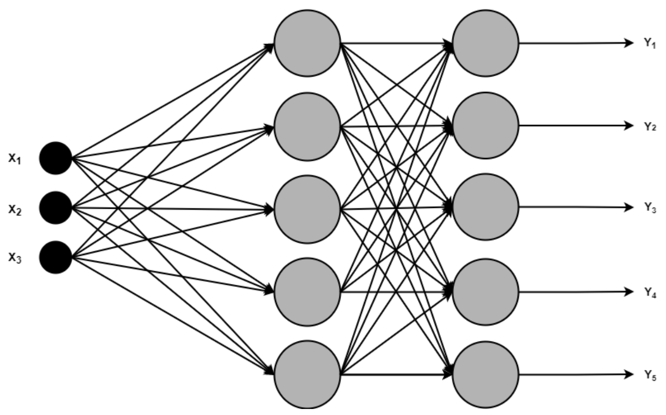


Fig. 7. Feed-Forward Network Structure

As seen in figure 7, each neuron in the first stage is connected to the neurons in the other layer. Combinations between all neurons require considerable high processing power which allow the solution of more complex problems. All neurons in the input layer direct their result to the output layer. If all the neurons between the layers are connected in a network, the network that is obtained can be called a fully connected network. On the other hand, if it is stated that the weight value between two different neurons, it is understood that there is no connection between that two neurons.

It is also pretty significant to state that single-layered perceptron models are only applicable to problems with linearly separable classes. Later on in the future, it is discovered that multi-layered perceptron networks can be used for non-linearly separable problems.

Even though multi-layered perceptron networks are similar to single-layered perceptron networks, they contain fundamental differences. There are many more hidden layers in multi-layered perceptron. Also, another important difference is that the multi-layer perceptron has a backward stage in addition to the forward stage.

In forward stage, a path from the input layer to the output layer is followed. In oppose to that in the backward stage, an error is obtained from the value from the generated output and the value that should actually be obtained, and this error is used to change the parameters in the neurons which are weights and biases.

One other important addition is the usage of non-linear activation functions in multi-layer networks. With these improvements, multi-layered networks are able to work on non-linearly separable problems.

After studying variaty of MLP network structures, we designed our own model in order to understand image classification task in depth. After training MLP model with different hyperparameters we achieved maximum 51.5% test accuracy.

B. Convolutional Neural Network

1) Layers of Neural Network: Convolutional neural networks (CNN) are a sub-branch of deep learning and it is often used in visual information analysis. Common usage areas are image and video recognition, image classification, medical image analysis and natural language processing. CNN consists of several layers such as convolutional layer, non-linearity layer, pooling (downsampling) layer, flattening layer, fully-connected layer. Convolution process is applied to the image matrix with the filter matrix at the convolutional layer so that specific features in the image are identified such as edges. Different filter processes are applied to determine the features in the picture. Each filtering process gives a feature matrix which determines a type of feature. Non-linearity layer is named as activation layer because in this layer one of the activation codes are used. This layer prevents the neural network from acting like a single perception. Pooling layer is located between consecutive convolution layers. This layer reduces the number of calculations by selecting certain parameters within the network. Pooling layer is not preferred in high productivity models. Flattening layer prepares data at the input of the fully-connected layer as a string.

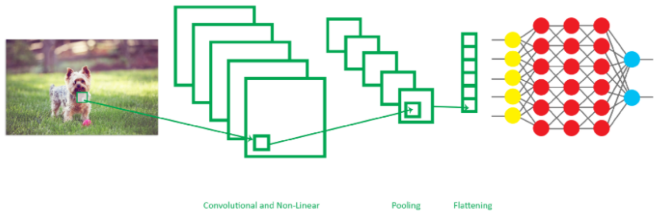


Fig. 8. CNN Architecture

In this architecture, six layers comprising alternating sequence of convolution and three pooling layers are used. All convolution layers contain filters of size 3x3 and number of channels are 32, 32, 64, 64, 128, 128 respectively. Stride of all convolution layers are one and padding is 'same' so dimension of the output matrix is same after convolution. As the number of filters changes, only the 3rd dimension of the matrix changes. 2x2 max pool layers are applied with stride of two after second, third and sixth convolution layer. Dimension of output matrix after pooling and convolution is given by the equation 3.

$$(\frac{n+2p-f}{s}+1)\times(\frac{n+2p-f}{s}+1) \tag{3}$$

Where n is the input dimension with nxn, f is the filter and pooling dimension with fxf and s is the stride. Pooling of all concolution layers is calculated with the equation 4. So dimension of input matrix is preserved after each convolution.

$$p=\frac{f-1}{2} \tag{4}$$

In all layers except fully connected layer, the 'elu' activation function is used. The input size of the network is (32x32x3). The number parameters at the output of flatten layer is 2048. Ten output neuron is added as fully-connected with the output parameters of flatten layer.

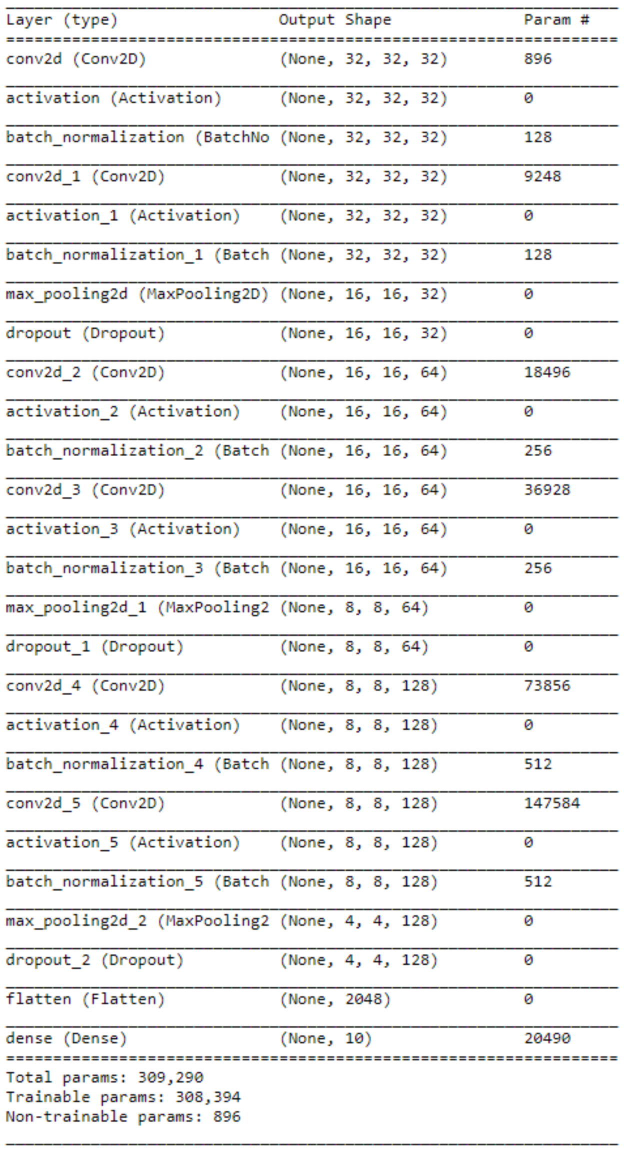

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896
_________________________________________________________________
activation (Activation)      (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 32)        128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        36928
_________________________________________________________________
activation_3 (Activation)    (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
activation_4 (Activation)    (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_4 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 128)         147584
_________________________________________________________________
activation_5 (Activation)    (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_5 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 10)                20490
=================================================================
Total params: 309,290
Trainable params: 308,394
Non-trainable params: 896
_________________________________________________________________
```

Fig. 9. Number of parameters and layer sizes

2) Activation Function: After multiplying with filter and pooling matrices, the output matrices are passed through an activation function that maps the input signals to the required output signals. Activation function must be non-linear. In the case of using linear activation function, the neural network behaves like a single perceptron. After every layer except fully

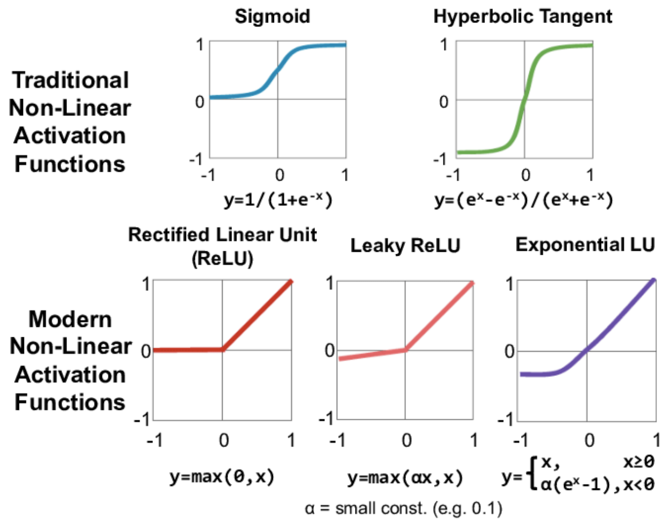

Sigmoid
Hyperbolic Tangent
Traditional Non-Linear Activation Functions
y=1/(1+e^-x)
y=(e^x-e^-x)/(e^x+e^-x)
Rectified Linear Unit (ReLU)
Leaky ReLU
Exponential LU
Modern Non-Linear Activation Functions
y=max(0,x)
y=max(αx,x)
y={x, x≥0; α(e^x-1), x<0
α = small const. (e.g. 0.1)

Fig. 10. Activation Functions [12]

connected layer the 'elu' activation function is used. At the final layer, fully-connected layer, 'softmax' activation function is used. Some commonly used activation functions can be seen in figure 10.

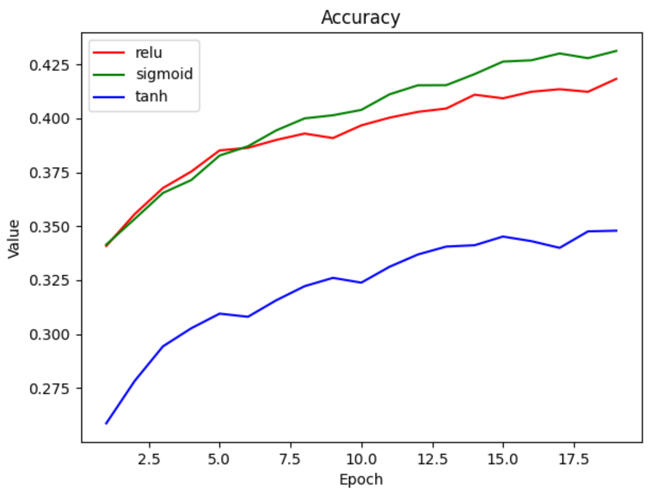


Fig. 11. Activation Function Comparison

3) Optimizer: In this network, RMSProp (Root Mean Square Propagation) is used. It was devised by Geoffrey Hinton. In RMS prop, each update is done according to the equations given in 5, 6, and 7 where n is initial larning rate, ω_t is exponential average of squares og gradients, and g_t is the gradient at time t along ω_j. This update is done separately for each parameter w^j.

$$v_t = \rho v_{t-1} + (1-\rho) * g_t^2 \tag{5}$$

$$\Delta\omega_t = -\frac{n}{\sqrt{v_t + \varepsilon}} * g_t \tag{6}$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t \tag{7}$$

4) Loss Function: The loss function is a measurement way of model prediction performance. It compares the predicted and true outputs. In this network, Categorical Cross entropy is used because it performs well in classification problems. As an example categorical cross entropy loss function is given in equation **??**.

$$CE = -\sum_{c=1}^{M} y_{i,c} log(p_{i,c}) \tag{8}$$

5) CNN Network 1: In previously mentioned model, %86.9 accuracy is obtained. Related training metrics are shown in figure 12.

6) CNN Network 2: CNN Network 2 is obtained by adding a hidden layer after flatten layer to the CNN Network 1. The hidden layer contains 512 neurons and fully-connected. This network gives %85 accuracy.

7) CNN Network 3: In our first CNN model, three layers comprising alternating sequence of convolution and three pooling layers are used. All convolution layers contain filters of size 3x3 and number of channels are 8, 16, 32 respectively. There are 32 neurons which named as hidden layer. In concolution layer, the 'relu' activation function is used. Sigmoid activation function is used for flatten and hidden layer, also softmax function is used for hidden and output layer. In this network %66 accuracy is obtained. After that result we try to fine-tune our model with different hyperparameters.

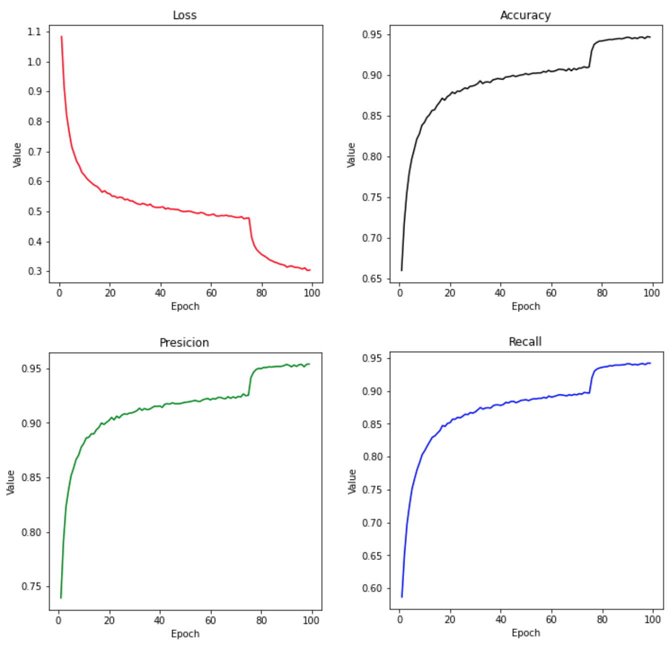


Fig. 12. Training metrics of tuned CNN

V. Hyperparameter Tuning

In this section our hyperparameter selection approaches and decisions will be explained. Given studies was applied on our

MLP model first, then using the known information from it becomes easier to design and tune our CNN model.

The process of finding most optimal hyperparameters in machine learning is called as hyperparameter tuning. The model parameters like weights are learnt during training by model. Unlike the model parameters, hyperparameters cannot learnt during training part. Hyperparameters are the only parameters in model which set by engineers. Changing the hyperparameters affects various metrics in the model. Some of the metrics are accuracy, speed and fitting of the model. Some of the important hyperparameters are number of nodes in the hidden layers, learning rate, number of hidden layers and number of epoch. To find best hyperparameters, the most important way that applied in the project is implementing theoretical knowladge. The other methods benefited in the project are Grid Search and Random Search. Also, various hyperparameters tried to see if the expectations are satisfied.

The trials of different hyperparameters are applied on the MLP model which explains in detail at methodology part of the report. Although the accuracy of the MLP model is not too high. It provides to observe the changes according to hyperparameters. In the trails, both loss and accuracy is observed to see changes more accurate. The hyperparameters that observed in the project are learning rate, number of hidden layers, neuron number in hidden layer, effects of difference optimizers, effects of different loss functions, dropout rate and the activation function in hidden layer.

Since the learning rate is crucial to find global minima, it is the first parameter tried in the project. Best learning rate is observed as 0.001. Some of the noted learning rate tests shown in below table as an example.

Hyperparameter	Accuracy (%)	Loss
Learning Rate = 0.1	35.0	1.80
Learning Rate = 0.01	45.0	1.60
Learning Rate = 0.001	52.5	1.30

The other trials are number of hidden layers and neuron number in hidden layer. Since the number of hidden layers and neuron number in hidden layer are increase complexity of the model, they are important for such a non linear dataset. The trials of these two parameters were made independently. Best number of hidden layer is observed as 2, and the neuron number in hidden layer is observed as 256. As an example data, some of our trails of different hidden layer and neuron numbers were given in below tables.

Hyperparameter	Accuracy (%)	Loss
Hidden Layer = 1	52.5	1.40
Hidden Layer = 2	55.0	1.28
Hidden Layer = 3	54.5	1.30

Hyperparameter	Accuracy (%)	Loss
Neuron Number = 128	46.0	1.50
Neuron Number = 256	48.0	1.55
Neuron Number = 512	42.0	1.70
Neuron Number = 1024	40.0	1.75

Another trial is about optimizers. Optimizers are methods used to change the attributes of the neural network such as weights and learning rate in order to reduce the losses. SGD, RMSProp, ADAM [13] are the optimizers tried in the trial. Best optimizer is observed as RMSProp in our case.

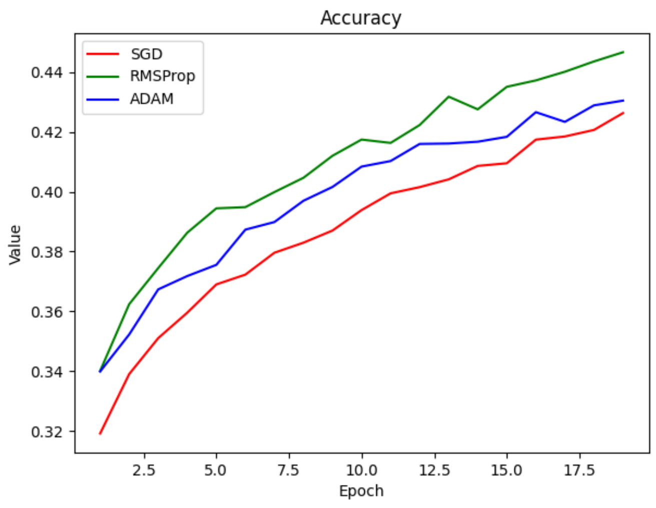


Fig. 13. Optimizer Comparison

The other trial is on Loss functions. Loss functions is used to estimate the loss of the model so that the weights can be updated to reduce the loss on the next iteration. MSE, Categorical Cross-Entropy, Kullback-Leiber Divergence and Categorical Hinge are the loss functions tried in the trial. In this trial, since loss functions are evaluated, the loss values are unimportant. The characteristic of the loss functions can be seen with loss values. End of the trial Categorical-Cross Entropy is observed as best loss function.

Loss Function	Accuracy (%)
MSE	40.0
Categorical Cross-Entropy	42.5
Kullback-leiber Divergence	42.0
Categorical Hinge	35.0

The other trial is on Dropout Rate. Dropout layers basically prevent overfit to the training dataset. CNN model was trained multiple times with different dropout rates.

The effect of learning rate, hidden layer and number of neurons in any hidden layers are noted and can be see in above tables respectively. Data in these tables were gathered with training MLP network for 20 epochs.

VI. Conclusion

In this study, we designed multiple custom algorithms using traditional machine learning methods such as KNN and SVM alongside with modern neural network structures like MLP and CNN. After that we optimized them for image classification task on CIFAR-10 dataset, we tuned our parameters such as number of layers, number of neurons, activation functions, optimizer to obtain maximum accuracy with minimal training time. We conclude that convolution neural network models outperformed previous KNN, SVN and MLP models.

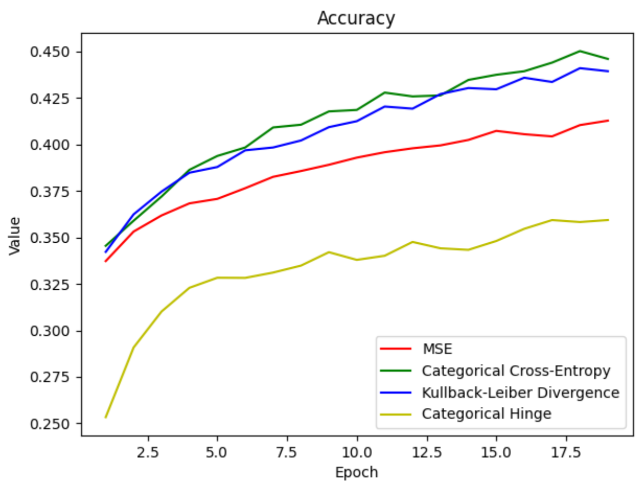


Fig. 14. Loss Function Comparison

A. Results

In figure 15 both previously mentioned MLP and this CNN network metrics can be seen throughout 100 epochs. Here the red lines represents the MLP model and blue lines is for CNN model. While it was possible to get better result in training phase with some changes, the increase of the test accuracy was negligible. Instead of overfitted models with high training accuracies, we worked on to achieve a well generalized model with relatively close train and test accuracies.

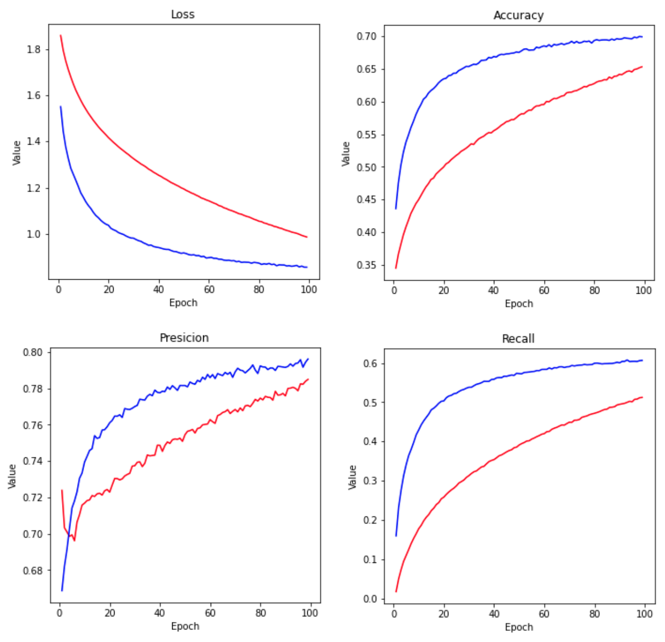


Fig. 15. Training metrics of MLP and CNN

After optimizing our CNN model and fine-tuning with mentioned hyperparameters, %86.96 accuracy obtained. Loss, precision, recall, and accuracy metrics plotted while training. These plots are given in figure 12.

In below table, we gather all of the implemented models and using the test set of CIFAR-10 we noted the accuracy values like shown.

	KNN	SVM	MLP	CNN
Accuracy (%)	22.16	23.71	51.53	86.96

The CNN model we design can performed quite exiting result on test set. Confusion matrix on the CNN model can be seen in figure 16. Also some predicted results are shown in figure 17.

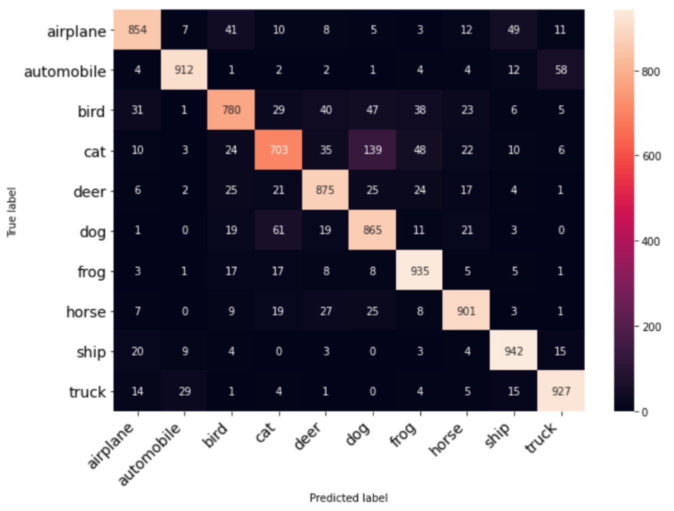


Fig. 16. Training metrics of tuned CNN

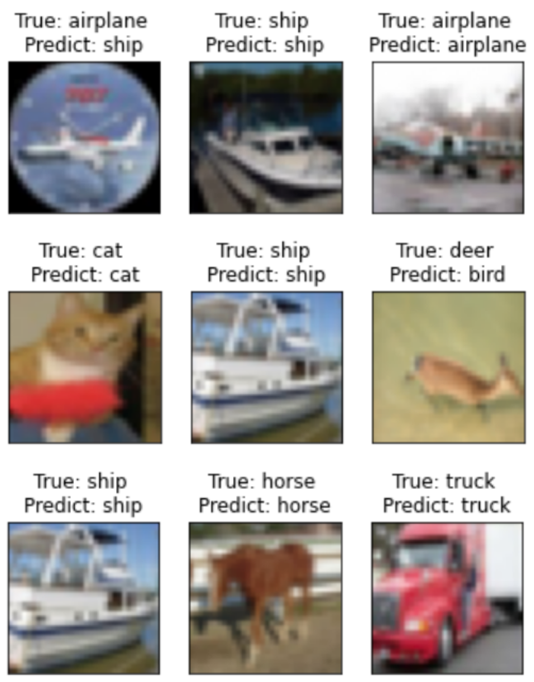


Fig. 17. Predicted labels vs. ground truths

Acknowledgment

We would like to thank our advisor *Associate Professor İsa Yıldırım.*

References

[1] Y. Abouelnaga, O. S. Ali, H. Rady, and M. Moustafa, "Cifar-10: Knn-based ensemble of classifiers," 2016.

[2] Y. Tang, "Deep learning using linear support vector machines," 2015.

[3] I. Jolliffe, "Principal component analysis," in *International Encyclopedia of Statistical Science*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1094–1096.

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE Inst. Electr. Electron. Eng.*, vol. 86, no. 11, pp. 2278–2324, 1998.

[5] A. Krizhevsky, "Convolutional deep belief networks on cifar-10," 05 2012.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[7] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[8] T. Ho-Phuoc, "Cifar10 to compare visual recognition performance between deep neural networks and humans," 2019.

[9] R. C. Çalik and M. F. Demirci, "Cifar-10 image classification with convolutional neural networks for embedded systems," in *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, 2018, pp. 1–2.

[10] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, "Sharpness-aware minimization for efficiently improving generalization," 2020.

[11] T. Ridnik, H. Lawen, A. Noy, E. B. Baruch, G. Sharir, and I. Friedman, "Tresnet: High performance gpu-dedicated architecture," 2020.

[12] V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, 03 2017.

[13] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.