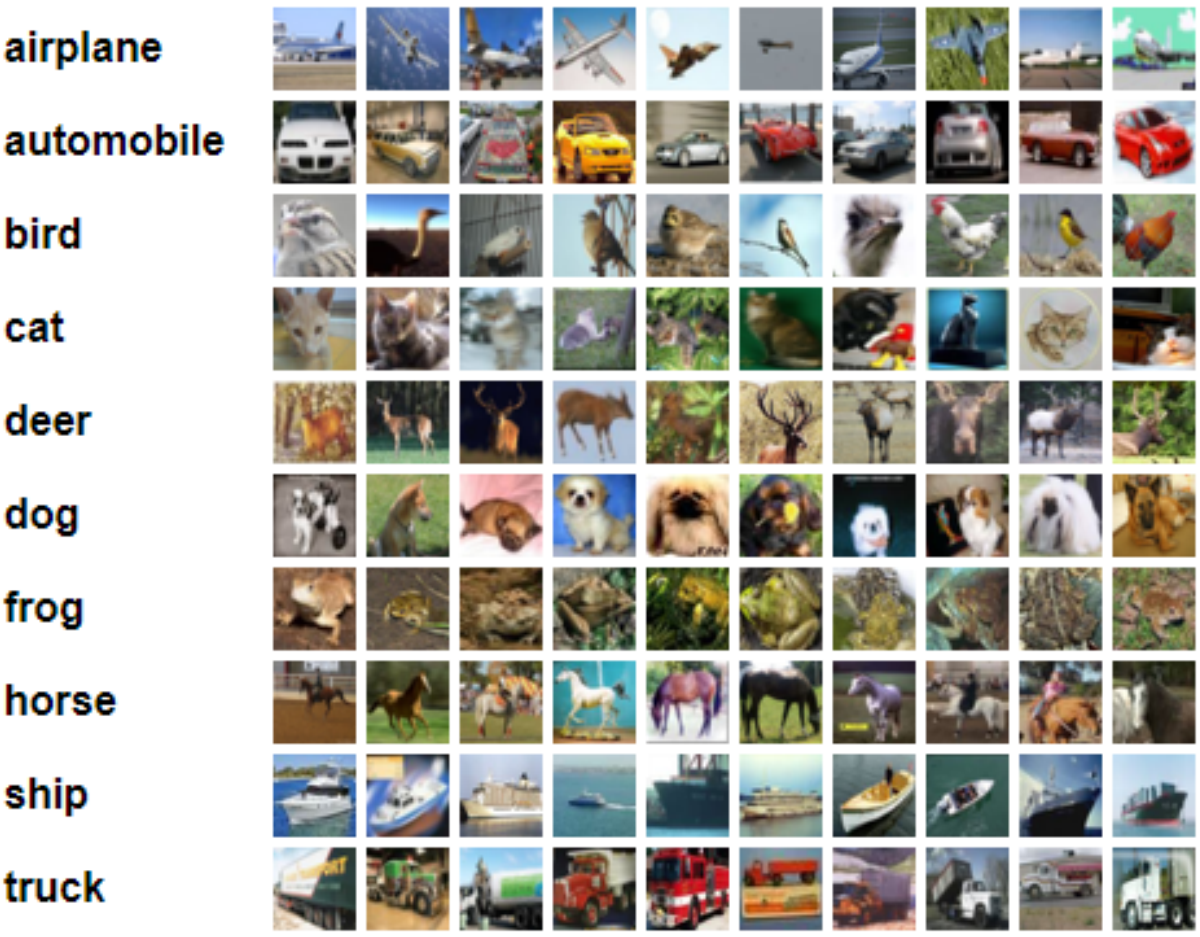
CIFAR-10 Dataset: Image Classification

Contributors: Ashley Maddix, Forest Pearson, Ian Wasson, Eben Weisman, JingJing Zhao



Specific Contributions

Multi-Layer Perceptron, CNN: Forest Pearson, Ian Wasson

K-Means: Ashley Maddix, Eben Weisman, JingJing Zhao

Project Report: Ashley Maddix, Forest Pearson, Ian Wasson, Eben Weisman, JingJing

Description

In this project, we set out to develop a set of algorithms to read and classify images from the CIFAR-10 Dataset. This dataset contains 60,000 32x32 color images in 10 classes, with 6,000 images per class. 50,000 images are for training and 10,000 are for testing. The data is further divided evenly into 6 batches, 5 training, 1 test. We processed this data through several different algorithms and compared their accuracies. Our goal was to compare 2 different supervised learning algorithms and 1 unsupervised learning algorithm against each other.

Evolution

In our initial designs, we endeavored to compare supervised vs. unsupervised learning algorithms. This paradigm was followed, the final project containing 3 learning algorithms to compare to one another: two neural networks, and one K-Means algorithm. In regards to the neural networks, primary development was towards a multi-layer perceptron, using back-propagation to update the weights. Following this, a convolutional network was developed using *tensorflow.keras*, and *CUDA/CUDNN* to increase process time drastically.
The development of K-Means remained relatively consistent, however special attention was paid to matrix formations, as the *CIFAR* data held many more dimensions in its raw format. The increase in dimensionality over the team's prior experiences was a major hurdle to overcome.

Limitations

The primary limitation for this experiment was computational power. The backpropagation necessary for training neural networks requires a significant amount of computational power, which meant that it required many hours of processing to obtain results. Due to the amount of time required to train these networks, we were only able to run 6 different tests, and were unable to add additional hidden layers.

Describing the Algorithm

Multilayer Perceptron

We decided to use a very basic three layer perceptron algorithm to analyze the CIFAR-10 datasets. The basic idea behind the algorithm is we accept every pixel and its three color layers (RGB) as inputs. These inputs are connected to a hidden layer via a network of weights, such that each input maps to each hidden unit. These hidden units are then mapped to an output layer. These output units correspond to the 10 different classes of images, whichever unit has the highest value is accepted as the prediction. After this process is complete, we analyze the accuracy of the prediction and update the weights between each of the layers for better accuracy.

Convolutional Neural Network

Our convolutional Neural Network was mainly used as a way to compare with a more advanced version of our MLP, even if it required some research into deep learning. This CNN primarily made use of tensorflow, keras, CUDA, and CUDNN to create all of the layers required for the project, with those being the initial convolutional layers, and the output pooling layers. These layers can be viewed in *Fig. 7*. The main difference between this and the MLP was its ability to reference neighboring data in the previous layer, such as pixels in an image, to learn from then share weights. The CNN also was capable of reducing model complexity, via flattening or other methods.

K-means

Our K Means algorithm finds a set of random X, Y, Z values equivalent to the length of clusters we aim to find. We use this random initial set with the central point of our plot to find a new set of centroids relevant to our data. We loop through our list of data, comparing that image to each centroid, then find the centroid closest to that point. Once we have a list containing the cluster grouping for each image in our training set, we use that list to update our centroid values, which are the central points of each cluster.

Data and Results

Fig. 1 - Fig. 5 are output data for K-Means : Fig. 6 and Fig. 7 are Convolutional Network output : Fig. 8 - Fig. 16 are results from the Multi-Layer Perceptron

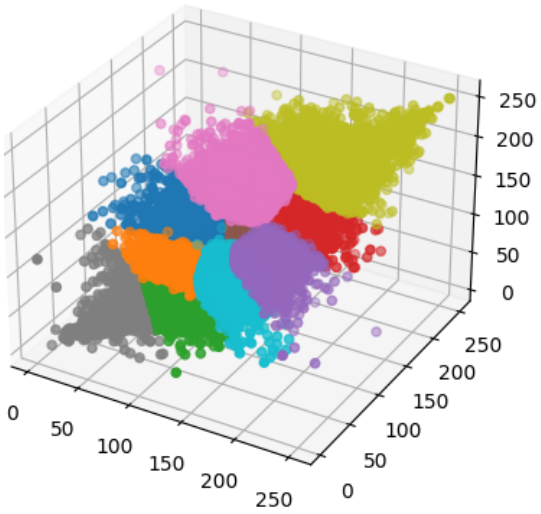


Figure 1.

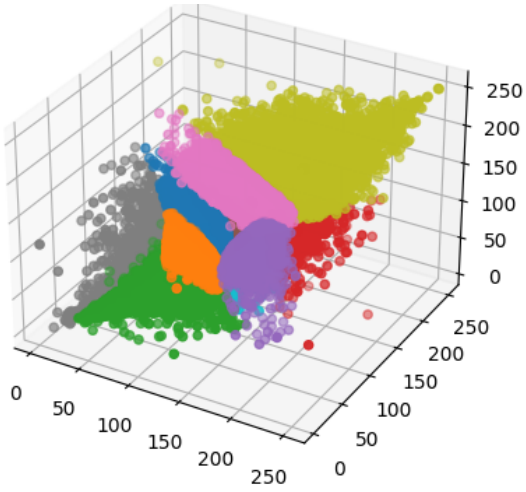


Figure 2.

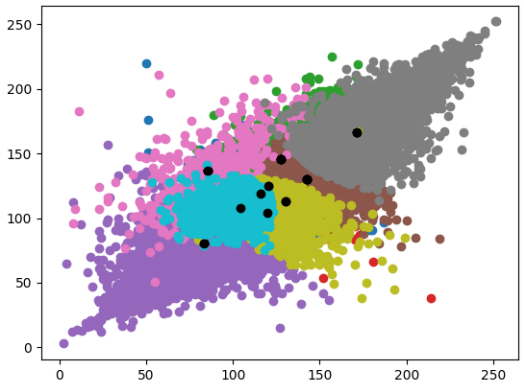


Figure 3.

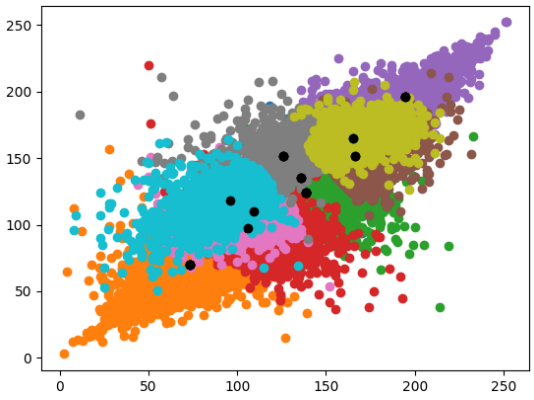


Figure 4.

K-Means accuracy			
	1	2	3
10	0.235601917	0.208728299	0.24843744675155
20	0.242334378	0.224885608	0.232441651264951
30	0.231812138	0.229232514	0.233703395785564
40	0.228795735	0.228957872	0.233807795375812
50	0.228269143	0.228902696	0.233633193919847
60	0.229597881	0.228912954	0.232670765481017
70	0.230305221	0.228354928	0.231834911955578
80	0.231014634	0.228104435	0.230913073424513
90	0.231851919	0.229405208	0.230895982310343
100	0.233521263	0.230541812	Converges
Avg	0.232310423	0.226602633	0.234259801807686

Figure 5.

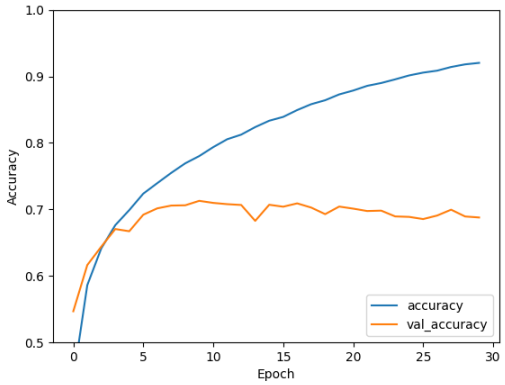


Figure 6.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36928
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 64)	65600
dense_1 (Dense)	(None, 10)	650

Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0

Figure 7.

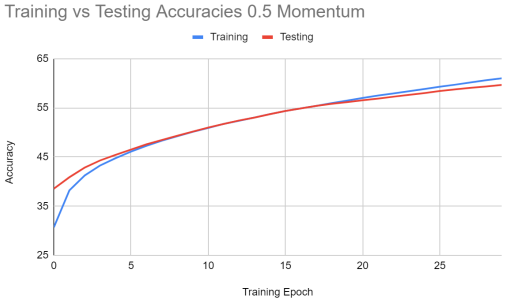


Figure 8.

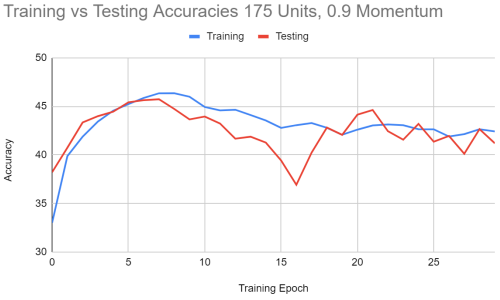


Figure 10.

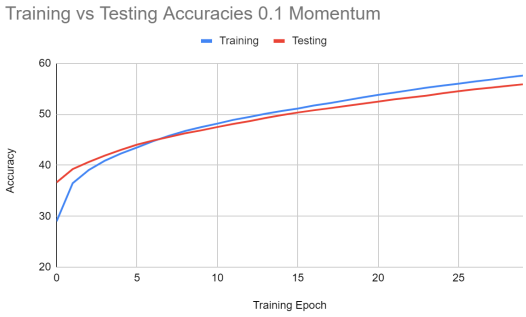


Figure 9.

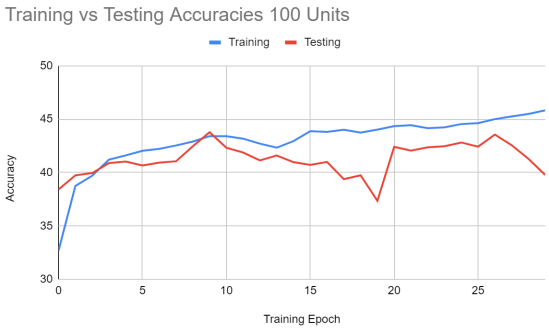


Figure 11.

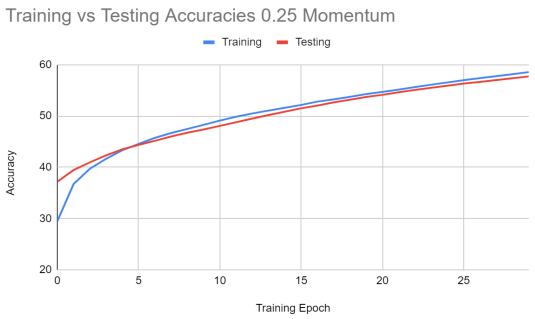


Figure 12.

```
[[3051  630  364   33  283   87   14   22  478   38]
 [ 267 3860  118   28  239  108   22   22  210  126]
 [ 740  373 2086   87 1211  285   60   52   85   21]
 [ 431  597  914  792  839 1160  102   33  105   27]
 [ 498  218  961   37 2878  221   52   42   72   21]
 [ 333  399 1026  215  820 2019   59   30   80   19]
 [ 191  398 1247  139 1349  515 1076   17   50   18]
 [ 596  379  919   96 1383  369   23 1076   80   79]
 [1010  776  146   37  221  154    7   10 2582   57]
 [ 491 1992  212   58  338  262   35   63  370 1179]]
```

Figure 14.

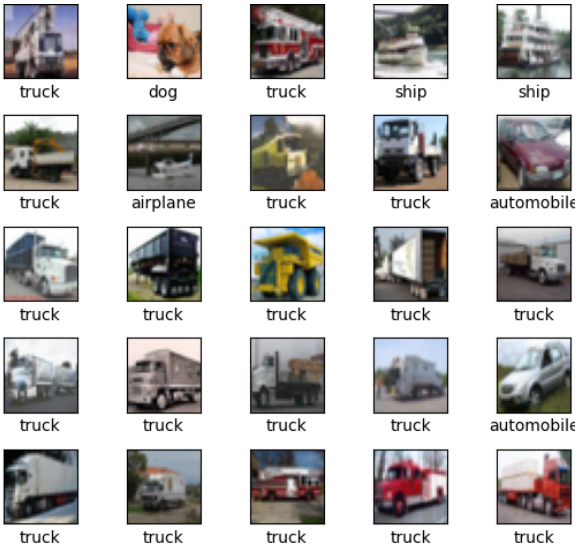


Figure 15.

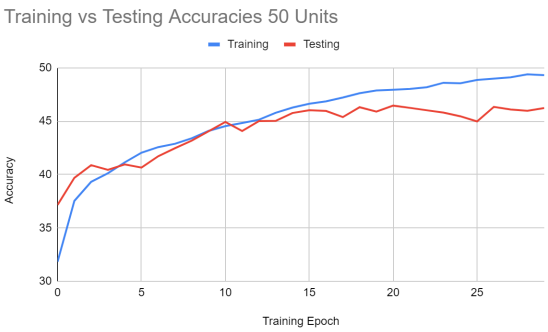


Figure 13.

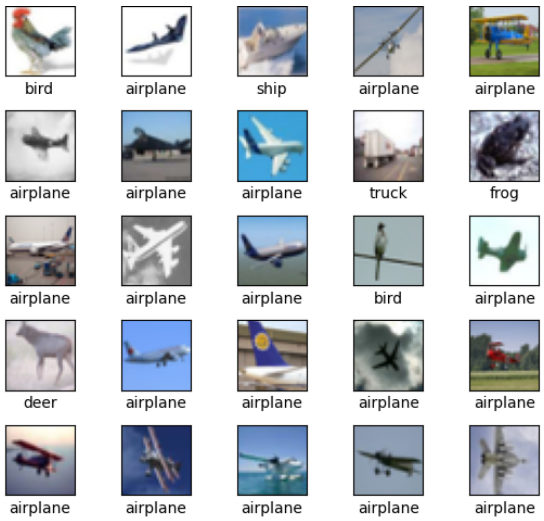


Figure 16.

Multi-Layer Perceptron

The MLP very quickly obtained an accuracy of 40%. The baseline that we used for the majority of the tests was 175 hidden units, with a learning rate of 0.01 and a momentum of 0.9. After training this for 30 epochs, it quickly plateaued at around 43% accuracy. We ran 5 additional tests, 3 of these were varying the momentum values, and the other 2 were varying the number of hidden units. The first hidden unit test was to reduce the number of units down to 100. This test obtained the same relative accuracy as our baseline. The second test was to reduce it further to 50 units. This was able to obtain a slightly higher accuracy of around 47%. In addition to its higher accuracy, it was also significantly faster to train.

Varying the momentum proved to have better results. The first test was to reduce the momentum down to 0.5, which achieved an accuracy of around 60%. The second test was to reduce it down to 0.25, which achieved an accuracy of around 57%. Finally, we reduced the momentum to 0.1, this gave us a result of about 54% accuracy.

All of the executed tests resulted in oscillations, as well as no obvious signs of overfitting. The testing accuracy was very similar to the training accuracies, and in most cases performed better. As for the data after processing, the MLP had significant trouble identifying cats and frogs, seen as classes 3 and 6 in Fig. 14, likely due to these classes having less distinct colors.

Convolutional Neural Network

This allowed it to quickly obtain a 70%+ testing accuracy within 5 epochs, a significant increase compared to even the best MLP results. Of course overfitting was also seen in this program, with some of our examples achieving near 94% training and 68% testing, as seen in *Fig. 6*, showing its exceptional ability to learn the data a little too well. This network also made use of CUDA and CUDNN Nvidia software, allowing for more advanced integration with GPUs as opposed to our other work. This change from our previous software allowed epochs to be performed in roughly 6 seconds on a consistent basis, even with the CIFAR data fully loaded.

K-Means

In this design, the project randomly selects ten data as centroids. Then, all data are grouped into classes in which each centroid is located according to the principle of minimum distance from the initial centroid. There is several data in each class, and the mean of all sample points in K classes is calculated as the K centroids for the second iteration. The steps are then repeated according to this center until convergence (the centroids no longer change), or the specified number of iterations has been reached. In this project, the value of K is set to 10 because there are 10 different kinds of images. The accuracy of the set images is calculated by having the most data in each class in the total amount of data. The accuracy rate is around 23%

Conclusion and Final Thoughts

The three algorithms we implemented all attained significantly different results. Of the two neural networks (our measure of supervised learning), the convolutional network performed the best. Both neural networks outperformed the K-Means algorithm, however seeing as K-Means is unsupervised, this was to be expected. The convolutional networks' accuracy tended to peak between 70% and 80% before becoming subjected to severe over-fitting, as seen in *Fig. 6*. Through the series of tests performed involving the MLP, optimal accuracy tended to peak near 60%, with the test involving 175 hidden units, and 0.5 momentum performing the best.
The K-Means algorithm maintained a significantly lower accuracy, with *Fig. 5* showing a final accuracy of ~23%. As stated earlier, this was expected to be low, as the process is unsupervised. K-Means also held some added complexity, as our experience with the algorithm was limited to a 2 feature dataset, and CIFAR had several orders of magnitude above this. However, K-Means was significantly faster to train on the dataset. It was able to obtain results relatively quickly, taking about 10-20 minutes to complete. This is in contrast to the MLP speed of around 12 hours.
Overall, it is clear that supervised learning is better suited to processing the CIFAR-10 dataset, performing, at a minimum, twice as well as our unsupervised learning algorithm.

References

The CIFAR-10 dataset. CIFAR-10 and CIFAR-100 datasets. (n.d.). Retrieved May-June, 2022, from https://www.cs.toronto.edu/~kriz/cifar.html

Chollet, F. C. (2022, April 12). *The Sequential model*. Keras. https://keras.io/guides/sequential_model/

Maddix, Pearson, Wasson, Weisman, & Zhao. (n.d.). *CS445-545-ML-Project*. GitHub. Retrieved June 9, 2022, from https://github.com/ForestPearson/CS445-545-ML-Project