

---

# Comparing Influence of Varied Layers, Pooling, and Optimization Methods on CIFAR-10 Classification

---

**Rebecca Du**  
(rrdu@ucsd.edu)

## Abstract

In this paper, a convolutional neural network is constructed to perform image classification on the CIFAR-10 dataset. The impact of changing the number of layers, the pooling method, and optimization function in the CNN on model performance was investigated. Results indicated that the best performing combination was a 3-layer CNN using AvgPool and SGD as the optimization method.

## 1. Introduction

In this final project, the CIFAR-10 dataset was used for the classification task. The dataset was downloaded directly from TensorFlow (<https://www.tensorflow.org/datasets/catalog/cifar10>).

The code for the final project (including loading, formatting, processing, and classifying the datasets) was written in the Python programming language (<https://www.python.org/>). Various Python libraries were used to assist the coding process, including Numpy (<http://www.numpy.org/>), PyTorch (<https://pytorch.org/>), and Matplotlib (<https://matplotlib.org/>). The code was written in Jupyter notebooks (<https://jupyter.org/>).

The algorithm used was a uniquely constructed Convolutional Neural Network made specifically for this task. The pooling and optimization methods were coded based on frameworks provided in class and resources online.

The Jupyter notebooks used in this project can be found in the Github repository linked below.

<https://github.com/rrdu/COGS-181-Final-Project>

## 2. Methodology

### 2.1 General Approach

The overall goal of this project was to assess how changing the number of layers, pooling methods, and optimization functions altered the performance of a CNN on the CIFAR-10 classification task.

The CIFAR-10 dataset consists of 60,000 32x32 pixel color images separated into 10 classes (plane, car, bird, cat, deer, dog, frog, horse, ship, truck). There are 6,000 images per class and the dataset is split into 50,000 training images and 10,000 test images.

The specific architecture (with output dimensions throughout the layers) of the CNN's used can be seen in **Figure 1** and **Figure 2**. Generally, they follow the sequence below:

- Input image
- $N$  number of convolutional blocks, with each block having:

- 3 x 3 convolutional layer (stride 1, padding 1)
- Batch normalization layer
- ReLU activation function
- 2 x 2 pooling layer (AvgPool or MaxPool)
- Flattening layer
- Dropout layer
- Fully-Connected layer (aka Linear layer) 1
- ReLU activation function
- Fully-Connected layer (aka Linear layer) 2
- Output
- Optimization function (SGD or Adam)

Due to multiple layers causing long runtimes, only CNNs with 2 convolutional blocks and 3 convolutional blocks were tested for this report.

The specific function of each of the layers in the CNN are discussed further below.

### 2.2 3x3 Convolutional Layer

The 3x3 convolutional layer applies a kernel of size 3x3 over the input image or the previous layer's feature map. The kernel convolves (i.e. slides) across the input (whether it be the input image or the resultant feature map from a previous layer), moving 1 stride at a time (in this case, 1 pixel). At each position, the kernel performs element-wise multiplication with the portion of the image it covers, the results of which are summed up and combined with a bias term. This results in a single value that represents one element. The elements accumulated from the kernel moving across the entire image are collected in the feature map matrix. The map represents features of the input, such as edges, corners, and textures. This map is then outputted for the next step in the CNN.

### 2.3 Batch Normalization Layer

A batch normalization layer was included in the CNN to normalize the inputs of the neural network layer for each mini-batch. The process includes computing the mean and variance across the mini-batch then subtracting the mean and dividing it by the mini-batch standard deviation. In short, it applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1. This stabilizes the learning process and aids in reducing the number of epochs required to train the CNN.

### 2.4 ReLU Activation Function

The ReLU (Rectified Linear Unit) activation function introduces non-linearity into the model, It is defined as

$$f(x) = \max(0, x)$$

Where  $x$  is the input to a neuron in the neural network. By thresholding values at 0, the ReLU function improves computational efficiency, especially in neural networks with many layers. Additionally, the gradient of the ReLU activation function is 1 for all positive inputs, thus enabling the model to learn faster. Overall, the ReLU activation layer serves to improve the complexity and efficiency of the CNN.

### 2.5 2x2 Pooling Layer

A 2x2 pooling layer separates the feature map it receives as an input from the previous layer into non-overlapping 2x2 size regions and then summarizes the value in each region into 1 output per region. This results in a feature map that has been reduced both spatially and in regards to the parameters that were previously present. This makes the representation more abstract. There are two types of pooling explored in this report:

- **MaxPool:** for each 2x2 region of the feature map, MaxPool takes the

maximum value. For example, if the values present were {3, 5, 2, 6}, MaxPool would return 6.

- This highlights the most prominent feature in the region, making it robust to noise and generally better for capturing edges, corners, and textures
- **AvgPool:** for each 2x2 region of the feature map, AvgPool computes the average of all the values. For example, if the values present were {3, 5, 2, 6}, AvgPool would return 4.
  - This gives equal importance to all the features in the region, performing more background information though at the cost of making it more sensitive to noise.

### 2.6 Flattening Layer

The flattening layer transforms the multi-dimensional feature map it receives as an input into a 1-dimensional array. It does this by stacking each of the arrays one after another into a single long vector. For example, if the input from a previous layer is a feature map of size 64 x 8 x 8 (an 8x8 feature map with 64 dimensions), the flattening layer would result in an output of a 1-dimensional vector with  $64 * 8 * 8 = 4,096$  elements. This prepares the data for the fully-connected (aka linear) layers.

### 2.7 Dropout Layer

The dropout layer for this CNN has a rate of 0.5, meaning it randomly sets half of the output units of the previous layer to 0 during each training update. The values that are not set to 0 are doubled to maintain the scale of the weight updates. This serves to prevent overfitting and makes the network learn more robust features. Without it, some neurons may rely too much on

other neurons. Dropout ensures that the neurons are able to operate independently.

### 2.8 Fully-Connected Layers

Fully-connected layers combine the features learned by previous layers to learn which features contribute the most to a particular output. In the CIFAR-10 classification task, FC layer 2 represents the final output. The current network structure uses two FC layers:

- **FC layer 1 (FC1):** FC1 takes the flattened output that has gone through dropout and begins deriving the final product. It begins to map the learned features to a space where relationships between the features and output labels can be formed
  - In the CNN of this paper, FC1 reduces the dimensionality of its input for a more compressed representation
- **FC layer 2 (FC2):** FC2 finishes mapping the features to the labels.
  - For classification tasks like CIFAR-10, the size of FC2 corresponds to the number of classes/labels in the task

In short, FC1 compartmentalizes and compresses the features from the previous layers while FC2 maps these features to the appropriate labels. As stated previously, ReLU activation function between the layers captures non-linear relationships and improves the network's performance on complex patterns in the data.

### 2.9 Optimization

The optimization step of the neural network serves to minimize the loss function during the training process of the neural network and thus improve accuracy. In general, they adjust the weights of the network to reduce the error

between the predicted and actual outputs. Two popular optimization methods: stochastic gradient descent (SGD) and adaptive moment estimation (Adam) were tested in this paper:

- **SGD:** SGD works by updating the network's weight using the derivative of the loss function with respect to each weight. Specifically, it calculates the gradient on a small batch of data rather than the whole dataset. It employs a fixed learning rate to control the step size during the weight updating process and the weights are updated in the opposite direction of the gradient to minimize the loss
  - Though SGD is simple and efficient, it can converge slowly on complex data.
- **Adam:** Adam also calculates gradients on small batches of data but keeps track of the mean and uncentered variance of the gradients. This enables Adam to adjust the learning rate for each weight individually, resulting in faster and more stable convergence. Adam utilizes the concept of "momentum" (moving average of past gradients) to accelerate optimization in the right direction.
  - Adam is less sensitive to initial learning rate and other hyperparameters and performs well overall.

### 3. Results

The performance of the CNN on the testing is summarized in **Appendix A.2** in three tables:

- **Table 1** lists the training loss curves for each of the CNN's.
  - Though they mostly followed the same pattern (with the most drastic change in mini-batch

loss occurring before index 150), CNNs that used MaxPool had lower losses sooner.

- **Table 2** shows the test accuracies on test samples for each of the CNN's. All the CNN's were tested on the same 4 images with GroundTruth labels of cat, ship, ship, and plane.
  - Based on the information in Table 2, it seems that the more layers the CNN has, the better its performance (with the 3-layer CNN's having 75-100% accuracy)
- **Table 3** displays the test accuracies for each image category across the CNN's
  - Interestingly, the category with the highest accuracy across all the CNN's was the car category.
- **Table 4** lists the overall test accuracies across the different categories for all the CNNs
  - The overall accuracies indicate that 3-layer AvgPool SGD is the CNN that performed the best on the test set.
- Table 5 compares the average overall accuracies of the pooling layers across optimization methods, and the average overall accuracies of the optimization methods across the pooling methods
  - For 2-layer CNNs, MaxPool and Adam perform better
  - For 3-layer CNNs, AvgPool and SGD perform better
  - This is reflect in Table 4, where 2-layer MaxPool Adam has the 2nd highest accuracy (74.1%) and 3-layer AvgPool SGD has the highest accuracy (75.3%)

### 4. Discussion

The results are mostly as expected. Naturally, CNNs with more layers are more complex and thus can capture the subtle nuances in images better. This explains why in general, the 3-layer CNNs outperformed the 2-layer CNNs - due to the increased abstraction capabilities resulting from the additional convolutional block.

The difference in performance between the pooling methods is a bit more difficult to distinguish. Though both performed roughly equally well, MaxPool outperformed AvgPool in the 2-layer CNNs and AvgPool outperformed MaxPool for the 3-layer CNNs. In the 2-layer CNNs, MaxPool's better performance can be explained by the fact that it extracts more prominent features and thus compensates for the lack of an additional convolutional block to provide the complexity necessary to properly classify the image. AvgPool underperforms in 2-layer CNNs because it smoothes the images too much and makes it difficult for the convolutional block to distinguish between the aspects of the image. On the other hand, the smoothing nature of AvgPool may explain why it outperformed MaxPool in the 3-layer CNN because it reduces the noise present in the low-resolution CIFAR images. It also prevents the overfitting that MaxPool is prone to, which may cause MaxPool to pick up the most prominent but not the most *relevant* feature for the classification task.

Finally, the difference in performance between Adam and SGD was also a bit surprising. Though Adam's fast convergence would make it seem like a better algorithm, the results indicate that Adam only works better than SGD in the 2

convolutional block CNN, whereas SGD outperforms it in the 3 convolutional block CNNs. Adam's faster convergence may cause it to generalize and overfit, which is especially a concern for the CIFAR-10 dataset due to the low-resolution of the images. SGD's simplicity, on the other hand, could have resulted in a more stable training process. One thing of note is that in general, the Adam CNNs took longer to run than the SGD CNNs, likely due to the more complex nature of the optimization method.

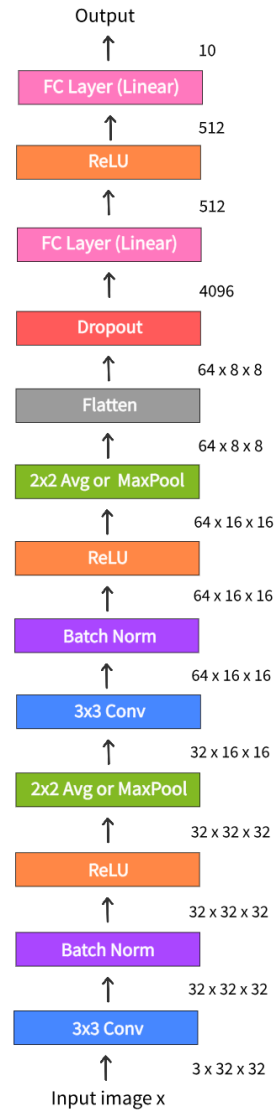
### 5. Conclusion

This paper investigated the impact of changing the number of layers, the pooling method, and the optimization function of a CNN on the CIFAR-10 image classification task. The results indicated that CNNs with more layers outperformed those with less layers. However, regarding pooling and optimization methods, the best method depended on the layers in the CNN. Smaller CNNs would prefer a combination of MaxPool to extract prominent features in the images and compensate for the comparative lack of network complexity, and Adam due to its fast convergence. In contrast, CNNs with more layers prefer a combination of AvgPool to reduce noise in the low-resolution CIFAR-10 images and SGD for a simple and stable training process. Overall, it seems that there is no one "right answer" as to which pooling and optimization is the absolute "best" for the CIFAR-10 classification task as it is more dependent on the structure of the individual networks themselves. Further research on the subject could consider the impact of different normalization and activation functions on network performance.

## A. Appendix

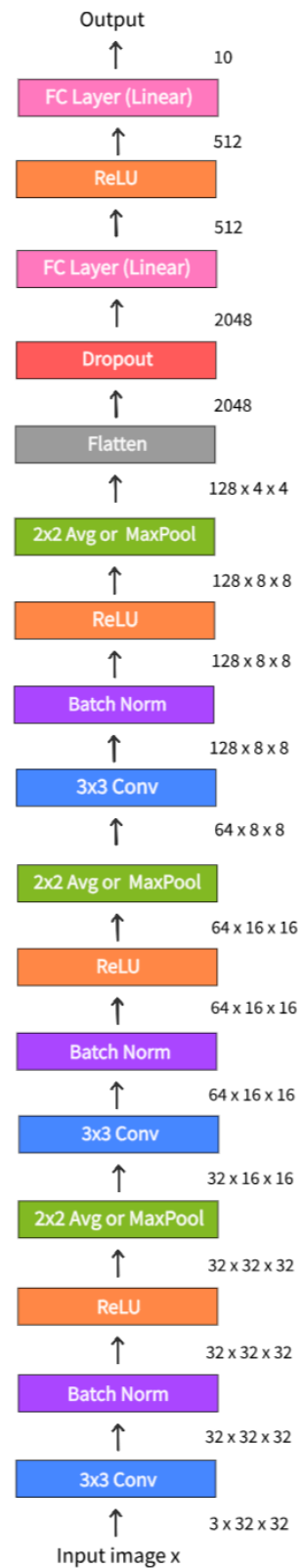
### A.1. Figures

Figure 1: 2-layer CNN Architecture



## Comparing Supervised Learning Algorithms on Various Datasets

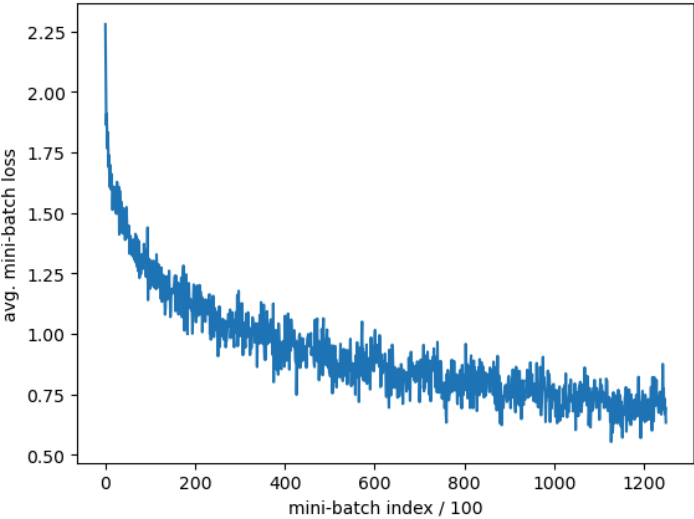
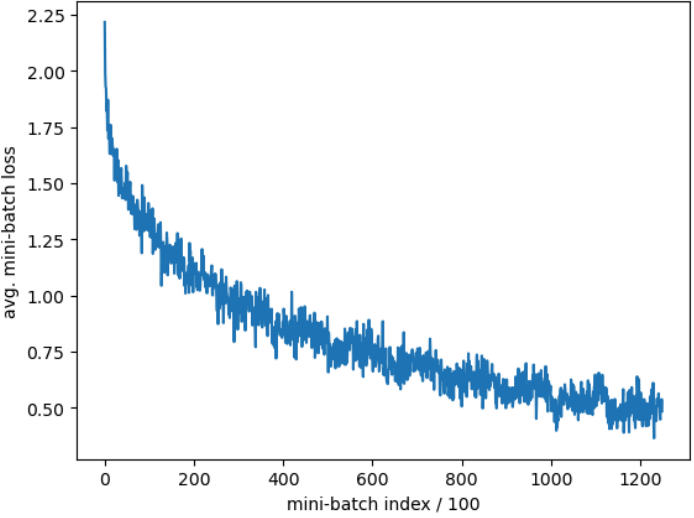
Figure 2: 3-layer CNN Architecture



Comparing Supervised Learning Algorithms on Various Datasets

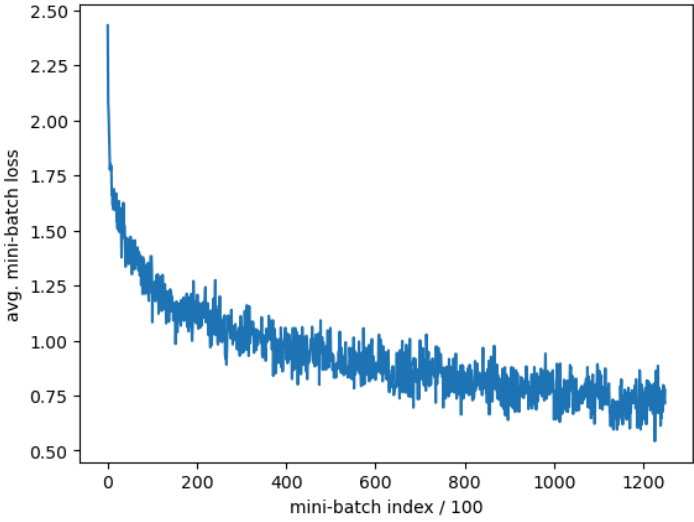
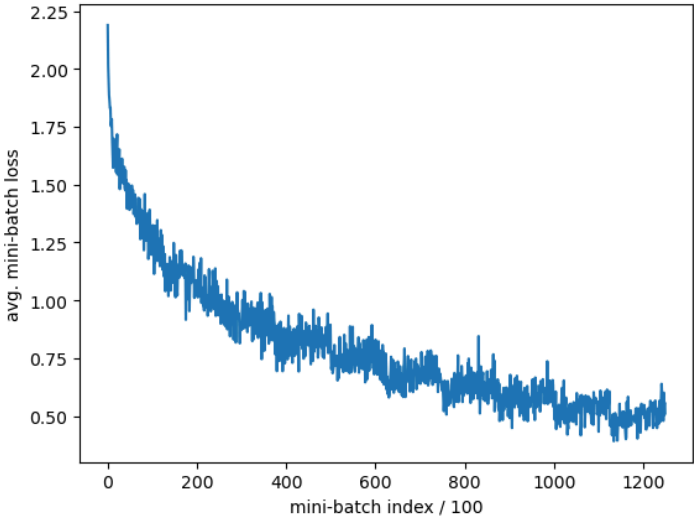
A.2. Tables

Table 1: Training Loss Curves Across CNNs

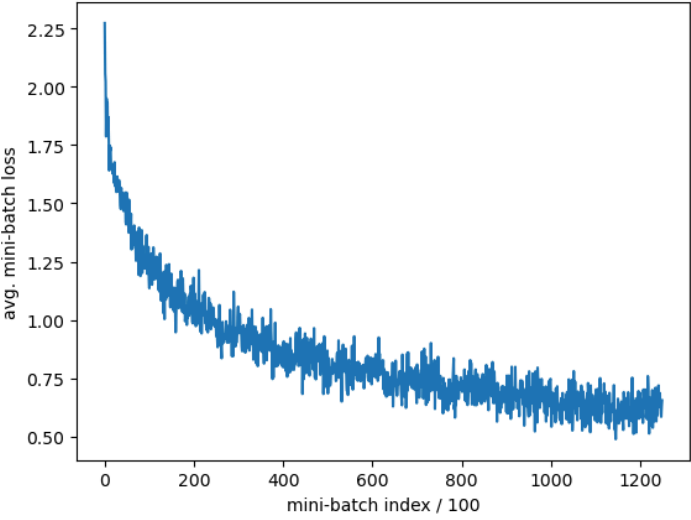
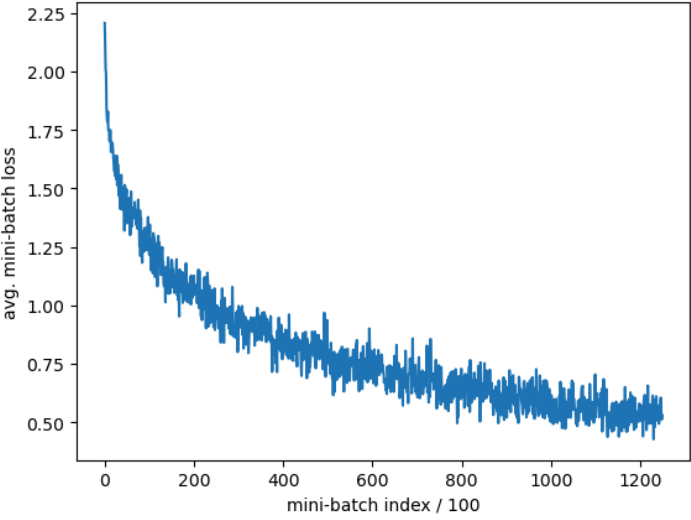
Layers	Pooling	Optimization	Training Loss Curve
2	AvgPool	Adam	
		SGD	



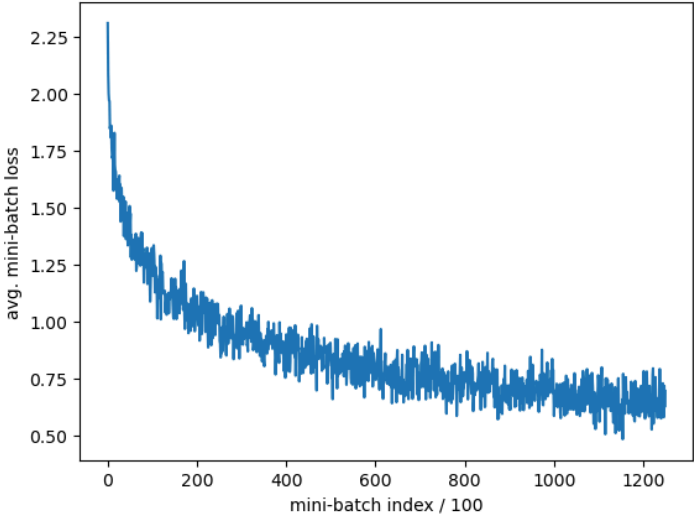
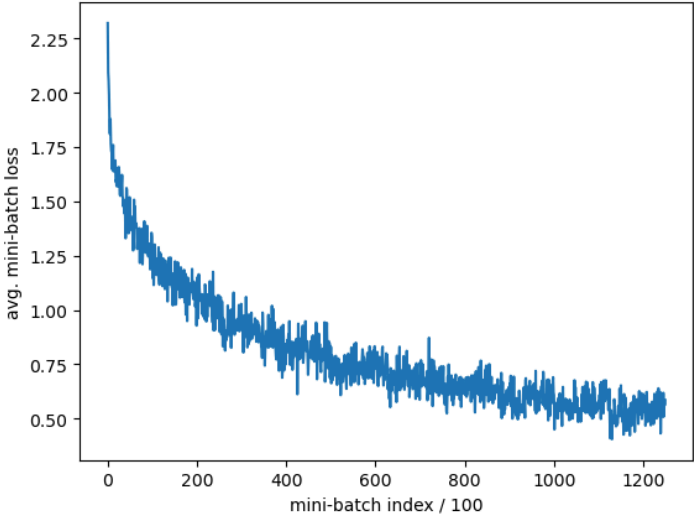
Comparing Supervised Learning Algorithms on Various Datasets

	MaxPool	Adam	
		SGD	

Comparing Supervised Learning Algorithms on Various Datasets

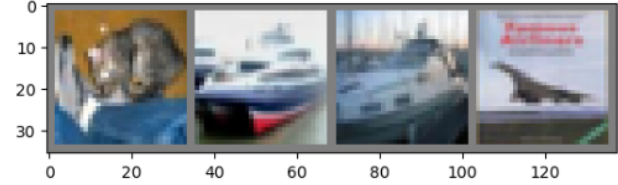
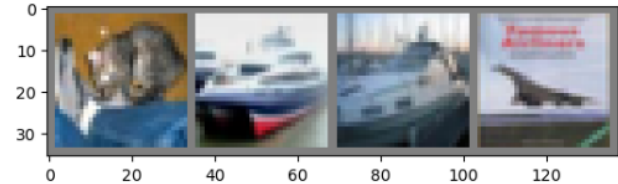
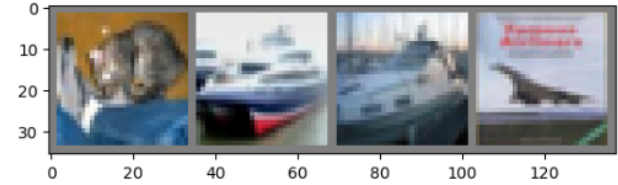
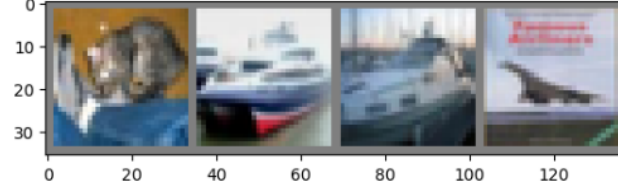
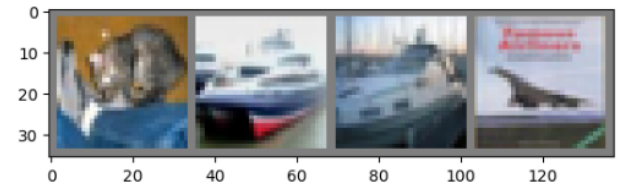
3	AvgPool	Adam	
		SGD	

Comparing Supervised Learning Algorithms on Various Datasets

	MaxPool	Adam	
		SGD	

## Comparing Supervised Learning Algorithms on Various Datasets

Table 2: Test Accuracies for Test Sample Across CNN's

Layers	Pooling	Optimization	Test Sample	Test Sample Accuracy
2	AvgPool	Adam	 <p>GroundTruth: cat ship ship plane Predicted: cat car ship ship</p>	50%
		SGD	 <p>GroundTruth: cat ship ship plane Predicted: dog ship ship plane</p>	75%
	MaxPool	Adam	 <p>GroundTruth: cat ship ship plane Predicted: cat ship plane plane</p>	75%
		SGD	 <p>GroundTruth: cat ship ship plane Predicted: dog ship ship plane</p>	75%
3	AvgPool	Adam	 <p>GroundTruth: cat ship ship plane Predicted: cat ship ship plane</p>	100%

## Comparing Supervised Learning Algorithms on Various Datasets


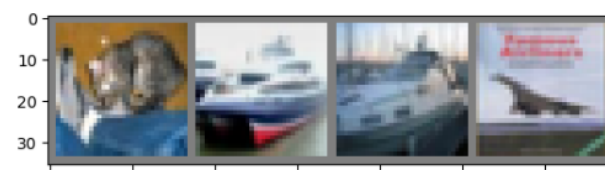
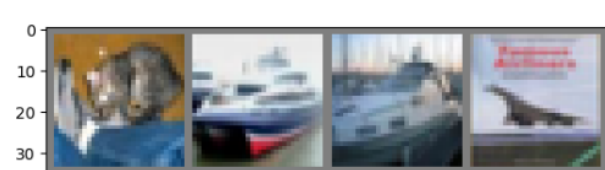
		<b>SGD</b>	 <p>GroundTruth:    cat    ship    ship    plane Predicted:    cat    car    ship    plane</p>	75%
	<b>MaxPool</b>	<b>Adam</b>	 <p>GroundTruth:    cat    ship    ship    plane Predicted:    cat    ship    plane    plane</p>	75%
		<b>SGD</b>	 <p>GroundTruth:    cat    ship    ship    plane Predicted:    cat    ship    ship    plane</p>	100%

Table 3: Test Accuracies for Each Category Across CNN's

Layers	Pooling	Optimization	Categories									
			Plane	Car	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck
2	AvgPool	Adam	73%	82%	55%	49%	72%	62%	70%	76%	82%	79%
		SGD	76%	83%	56%	51%	71%	62%	79%	76%	81%	82%
	MaxPool	Adam	72%	80%	58%	45%	53%	65%	80%	76%	81%	77%
		SGD	74%	74%	61%	56%	70%	58%	56%	72%	80%	84%
3	AvgPool	Adam	81%	83%	65%	58%	73%	60%	82%	75%	82%	81%
		SGD	76%	80%	71%	53%	75%	68%	78%	78%	85%	89%
	MaxPool	Adam	76%	87%	57%	63%	76%	55%	77%	72%	84%	82%
		SGD	79%	89%	69%	63%	67%	59%	74%	82%	77%	76%

Comparing Supervised Learning Algorithms on Various Datasets

Table 4: Overall Test Accuracies Across Categories

Layers	Pooling	Optimization	Overall Accuracies
2	AvgPool	Adam	70.0%
		SGD	71.7%
	MaxPool	Adam	74.1%
		SGD	68.5%
3	AvgPool	Adam	74.0%
		SGD	75.3%
	MaxPool	Adam	72.9%
		SGD	73.5%

Table 5: AvgPool vs. MaxPool and Adam vs. SGD

Layers	Pooling	Average Overall Accuracies across Optimizations	Optimization	Average Overall Accuracies across Pooling Methods
2	AvgPool	70.85%	Adam	72.05%
	MaxPool	71.3%	SGD	70.1%
3	AvgPool	74.65%	Adam	73.45%
	MaxPool	73.2%	SGD	74.4%