Philipp Hohlfeld, Stefan Bielmeier, and Bob Ni
CS294-082: Experimental Design for Machine Learning on Multimedia Data
Prof. Gerald Friedland

Estimating the complexity of CIFAR-10 and training a compact and accurate model

0. Context, Background, and Preview of Methodology

CIFAR10 is a labeled image set commonly used by machine learning researchers as a benchmark for model performance. It consists of a total of 50,000 labeled images in the training set and 10,000 labeled images in the test set. In **section 1**, we provide an overview of the dataset and weigh in on potential questions about the dataset itself.

Because CIFAR10 is such a commonly used dataset, there is a plentitude of papers published on increasing model accuracy to higher and higher bounds. In **section 2**, we examine the objectives of the model, what accuracy and performance success mean to us, as well as any other factors that influence the performance of our model.

We take a more analytical lens in **section 3**, where we look at measuring the MEC of the data set. For reasons explained further in this section, we subset and modify the original dataset in order to create a functioning model:

1) We compress all of the images to JPEG Q20 in grayscale, in lieu of a convolutional layer in an attempt to extract important image information for our model.
2) We further eliminated all but two classes for classification; this is because the 10-class problem turns out to require an inordinate amount of expected MEC in a model in order to memorize, and we are bounded by computing and time constraints in our environment.

Based on our analysis of the data set as well as model performance, we select a model that maximizes validation set accuracy, tune this further, and produce and measure the best model produced.

In **section 4**, we further explore a potential model's ability to generalize and its resilience.

In **section 5**, we examine where the dataset is sufficient to generalize, and any resulting limitations inherent to the data set.

We explore the bounds of our machine learning model by training it to memorization in **section 6**; we also describe the issues we had encountered with training a model in the case of the multiclass problem with unscaled data.

In section 7, we explore the generalization curve of our machine learner, and explain how we arrived at the model we chose in section 3. We examine the considerations taken in light of capacity constraints and other principles in machine learning experimental design.

In section 8, we discuss steps that can be taken toward quality assurance in the context of this dataset and model.

In section 9, we cover the reproducibility and repeatability of our findings.

1. Variables, Labeling, and Annotator agreement

We use CIFAR10 for our data set, which consists of 50,000 training set images and 10,000 test set images. Each image belongs to one of ten classes: cat, dog, plane, car, ship, deer, bird, frog, horse, or truck. As explained later in the paper, we first explore the data set as a 10-class dataset, but will later narrow it down to a binary problem (cat and dog) in order to perform all of the analyses required and derive performance metrics measured in class.

The goal of our machine learner is to predict which class the input image belongs to. Within the data set, the label "image type" describes the generally agreed upon class labeling, which we are targeting in our modeling.

In terms of expected accuracy of the labeling, this data set has been vetted by numerous academic research teams; we can reasonably expect the labeling on these images to be as correct as possible. We did not go through the images and verify the correctness of all of them, except for a handful of examples.

There is no published source on annotator agreement for CIFAR-10, as far as our team can ascertain. However, given the proliferation of papers published on this data set, we assume that there are no known issues on annotator agreement. We have seen no issues across papers or forums that indicate any level of disagreement on the labeling, so we will assume that annotator agreement is near 100% for the purposes of our experiment.

2. Success accuracy, data, classes, noise, and bias

Success accuracy and data

Because CIFAR10 is a commonly cited data set used for benchmarking machine learning models, there are many published papers indicating incremental improvements on test set accuracy.

In an initial paper on the topic, 82% accuracy was demonstrated through an initial experiment with CNNs[1]. Most recently in 2021, a team achieved 99% accuracy through the use of the Transformer architecture alongside CNNs[2].

While we have no target for the level of accuracy required for success, we do strive for as high of an accuracy level as possible given the constraints of generalization and expected MEC of the data set. When selecting our model, we used the conventional guidance of maximizing validation set accuracy; however, we seek to understand and measure the implications this has on overall model generalization. We observe this trade-off first-hand in this experiment.

There are 50,000 training images in the training set, with 5,000 training images for each of the 10 possible classes. As a result, the classes are perfectly balanced. We split the training set into a 50-50 split between training and validation, which might result in slight class imbalances while we try to determine the optimal model.

All images are 32x32 full RGB images in the original data set, but we compress these images into JPEG Q20 and convert them to grayscale for the majority of this paper. In either case, these images are represented as 1,024-column (from 32x32 pixels) data points.

[1] Google Code Archive - Long-term storage for Google Code Project Hosting.
[2] https://arxiv.org/pdf/2010.11929v2.pdf

As we will explore later in the paper, we actually narrow down the data set later to a binary problem (cat and dog). In this case, we end up with 10,000 training images, with 5,000 images of each class. As a result, we have 2,000 images in our test set, 1,000 of each class. We still hold the 50-50 training/validation split within our training dataset. The same class balance holds in this case.

Modalities

The modalities that we can exploit in this data set are relatively sparse. We only have images depicting ten different objects in quite low resolution: frog, truck, ship, plane, car, bird, deer, dog, cat, horse. The original dataset contains R/G/B values, which are another form of modalities; however, for our paper, we will be compressing images by using JPEG of quality 20, and switching to grayscale, which will eliminate the R/G/B modality.

We cannot foresee any more information in the image that could be exploited other than the object shape or color, and the corresponding label with that object. The exception could be ships, trucks, cars, or planes might have words on them as they're man-made. It's very unlikely that we can exploit any words on these objects for our classification task as the resolution is so low (32x32). We can reasonably assume the words are not detectable, especially for JPEGs of quality 20. Furthermore, any background information will likely be eliminated too with our compression and conversion to grayscale.

Temporal Information, Noise, and Bias

Furthermore, there is no temporal information in this data, as far as we are able to ascertain, since this is static image data.

We expect there to be a relatively high amount of noise in the data. Much of the color in the object might be irrelevant for classification accuracy, which is why we switch to grayscale. The background also is probably largely composed of noise, as well. We assume this because JPEG compression of the original image dataset removes color in the background first.

With respect to bias in the dataset, there are likely to be sources underlying the data that can be both quantified and non-quantified. For example, the images are likely to be universally agreed upon images, meaning that certain types of cats (for example "big" cats like tigers) might not be included in the dataset, or might be less represented than in the real world's data sets. Other sources could include selecting only clear images, images that were in the public domain already, or images with enough light to illuminate the subject. Via Brainome, -biasmeter tells us that the dataset is likely weighted 77.66% towards class 0 (cats) away from class 1 (dogs).

3. MEC for the data

In order to calculate MEC of the original dataset, we leverage the following formula derived from the in-class formula[3] for the two-class scenario, but adapted for the multiclass nature of this dataset:

*Σ (1/p_i) * log_2(thresholds_i + 1) * d*

[3] https://piazza.com/class/kss3z4d4x027ag?cid=90

Where we sum for every single class i; the probability of each class (p_i) is 1/10, thresholds_i is the number of thresholds associated with class i (explained below), and d is the number of columns in this dataset.

We can count the number of thresholds for each class by using Algorithm 1 from Chapter 9, with the following change: since the dataset we are using is multiclass, we count the number of thresholds associated with each class for everytime a class changes.

Working with the 10-class dataset (RGB)

For the full RGB 10-class dataset, we derive a total of 44,315 thresholds across the ten classes. Using the formula above yields approximately ~13 bits, which is the bits needed with a perfect machine learner. We multiply this by the number of data columns in the data set, which is 3 * 1,024, yielding a total of 39,937 bits needed to memorize this data set as a dictionary.

Working with the 10-class dataset (JPEG Q20, grayscale)

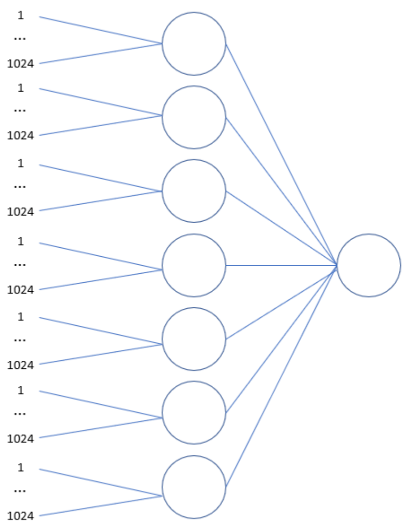
For a compressed JPEG (Q20) grayscale version of the 10-class CIFAR-10 dataset, we derive a total of 44,214 thresholds across the ten classes. Taking log_2 of this value yields approximately ~13 bits, which is the bits needed with a perfect machine learner. We multiply this by the number of data columns in the data set, which is 1,024, yielding a total of 13,313 bits needed to memorize this data set as a dictionary.

Working with the 2-class dataset (JPEG Q20, grayscale, standard-scaled)

For a compressed JPEG (Q20) grayscale version of the CIFAR-10 dataset with only two classes (“cat” and “dog”), we derive a total of 4,916 thresholds across the two classes. Taking log_2 of this value yields approximately ~13 bits, which is the bits needed with a perfect machine learner. We multiply this by the number of data columns in the data set, which is 1,024, yielding a total of 13,313 bits needed to memorize this data set as a dictionary.

This dataset includes a scaling using Scikit learn’s StandardScaler, which results in pixel or column values with mean = 0, and unit variance.

Measuring the MEC of a neural network



We built a neural network with 7 neurons in the hidden layer (we explain how we determined this is the optimal model in section 7). In this model, the expected Memory Equivalent Capacity ended up being approximately 7,183 bits, based on the four engineering rules for calculating MEC (our calculation simplifies into 7 * (1,024 + 1) + 1 * (7 + 1)). This is about half of the dataset MEC.

Our evaluated model structure consists of an initial layer of 7 perceptrons (each of which accept 1,024 inputs), which feed into 1 output perceptron. The diagram for our selected neural network is to the right. We chose this model based on validation set accuracy, further expanded upon in section 7.

Throughout this document, we also consider the implications of choosing a network with 1 neuron in the hidden layer, for the sake of maximizing generalization. Based on the four engineering rules, this model's expected MEC would be 1 * (1,024 +1) + 1 + 1 = 1,026 bits, which we will reference later.

4. Expected Generalization and Resilience

In this section, evaluate the expected generalization of a neural network classifier for our dataset. We leverage the following formula[4] for calculating the generalization of a balanced binary classifier:

G = # correctly classified instances / MEC.

For our standard-scaled (mean = 0, unit variance) cat and dog 2-class dataset with a total of 10,000 training examples, we divide that number by the expected Neural Network MEC of 13,313. The resulting generalization is 0.75 bits/bit, which is equivalent to the corresponding brainome pre-measurements for that dataset. With a benchmark of 2 indicating whether a model is generalizing, we do not expect to generalize training a neural network on either dataset. But, we will see later. When we consider our 10-class JPEG compressed dataset, the expected generalization for a neural network would be (according to brainome in lieu of a formula for a multi-class problem) at 10.02 bits/bit.

In terms of resilience, we can utilize the following formula[5] for calculating the average resilience of a balanced binary classifier:

*Resilience = -20 * log_10(G)*

Again in lieu of a formula for a multi-class set, we consider only the binary dataset with expected G = 0.75. For resilience, we calculate -20 * log_10(0.75) = + 2.498 dB for the scaled cat and dog dataset. We do not believe that our resilience is at par with what we would need for accurately completing the CIFAR10 classification task. Given the -22.4 dB benchmark we discussed in class, this indicates that a future model probably will memorize, or can only find a small amount of pattern in the dataset, most of which is noise. Because of our compression algorithm (JPEG Q20), we likely were not able to extract much feature information via our model. Changing the compression to, e.g. a CNN, we may receive more learnable data; in the appendix, we also discuss the initial results of a CNN that might be able to generalize more, and have a better resilience to noise.

Given the low expected generalization (<1) and *positive* resilience, we think that adversarial examples could present a very bad situation. The model may just classify adversarial images as any category because the model shows such low resilience to new data – probably because we are using gray-scale and JPEG compression.

With respect to data drift, we are working with a set CIFAR10 dataset that is likely not going to be changed in the near future. However, if the authors chose to update the images in

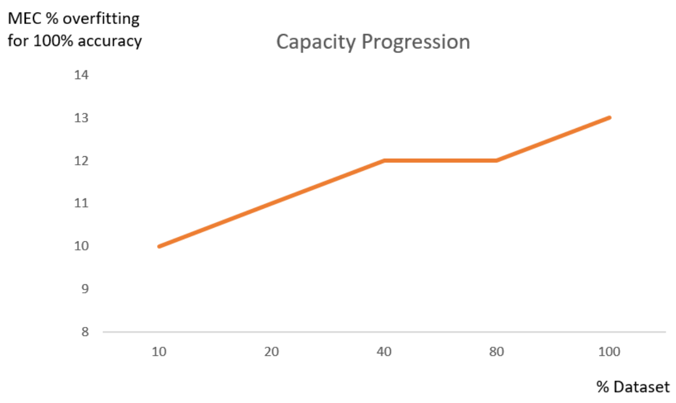
[4] CS294 Lecture 1 Notes cs294-1 (berkeley.edu), page 29
[5] Resilience, Glossary - Brainome.ai

the dataset, this could present some challenges. For certain subjects, such as frogs, deer, birds, or horses, we expected image representation of these to be fairly consistent, as these subjects do not tend to change over time. However, with automobiles, trucks, ships, or airplanes, these subjects could change over time (with new models, designs, or colors), and it could become difficult to identify these using our older machine learner. We would need to retrain on a new set of updated images. For example, if there is a new type of car introduced that resembles a frog, we would need to retrain and the model might not be resilient to these new differences.

5. Sufficient data and capacity progression

According to Brainome, the overall CIFAR10 dataset is at the level of “yellow” data sufficiency: “Maybe enough data to generalize”.

However, when we look at just the scaled version of the two-class training set, Brainome indicates this is “red”. Indeed, we see that the capacity progression to the right does not appear to level off at any point; it looks like it is about to level off, but rather depicts a linear function. This indicates that there is not enough data to generalize. This is confirmed by both Brainome and our analysis in sections 4 and 7.



6. Training for memorization

We can create a neural network that trains for memorization (100% training accuracy) by applying the engineering principles for neural networks. Our goal is to build a network whose capacity equals our previous estimation of the MEC of the dataset.

In this section, we discuss our attempts at training neural networks to reach 100% accuracy on the unscaled 10-class dataset (compressed to JPEG Q20, grayscale) with PyTorch, and why we ultimately chose to pivot into working on the standard-scaled 2-class dataset.

Working with the 10-class dataset (JPEG Q20, grayscale)

We initially tried to use Brainome to reach memorization of the 10-class dataset; however, Brainome reached an accuracy of 40% with neural networks when run with -nosplit. As a result, we decided to build our own neural network to reach 100% memorization.

Our initial assessment of the full CIFAR10 training dataset yielded an estimate of 13,313 bits required for memorization. Assuming a simple architecture of one hidden layer, and 10 output neurons (one per class), we only need to determine the number of neurons in the first hidden layer, given by *x.*

The following formula calculates the capacity of such a network if we include a bias term for each neuron. The input dimensionality of our dataset is 1,024 (grayscale), resulting from images with one channel per pixel with a total size of 32x32 pixels each:

*13,313 bits = (input dims + 1) * x + (x + 1) * 10.*

Solving for x, we get a result for x = 12.88. We round up and get a result of x = 13 neurons. We thus need to build a network of 13 neurons in the first layer, and 10 neurons in the second layer to memorize the dataset (training at memory equivalent capacity). We implement it in Pytorch via a feed-forward neural network with the following structure.

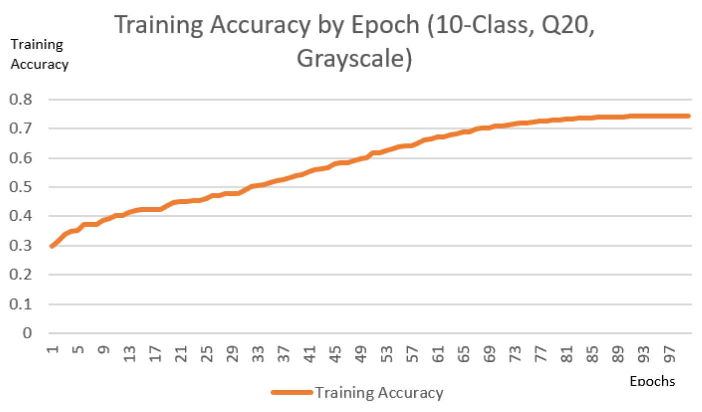
```
#first layer, 1024
self.fc1 = nn.Linear(1024, 13)
self.fc2 = nn.Linear(13, 10)
```

We used the following hyperparameters:

Hyperparameter (Pytorch)	**Value**
Loss Function	Cross Entropy Loss With Reduction = “Mean”
Activation Function	Sigmoid
Batch Size	64
Optimizer	ADAM
Learning Rate	0.00001
Regularization	None
Betas	0.9, 0.98
Eps	1e-08

However, running with these parameters ended up converging to an accuracy of about 40%. We did sample from multiple different hyperparameter values to try to improve upon this, but to little or no change in this result.

We then attempted to build a model to reach the calculated maximum required capacity of the dataset, which was 45,319,351 bits. As a result, we built a model with 44,214 neurons in order to reach the maximum required capacity. However, this model converged as well after 100 epochs to 75%, as seen on the right.



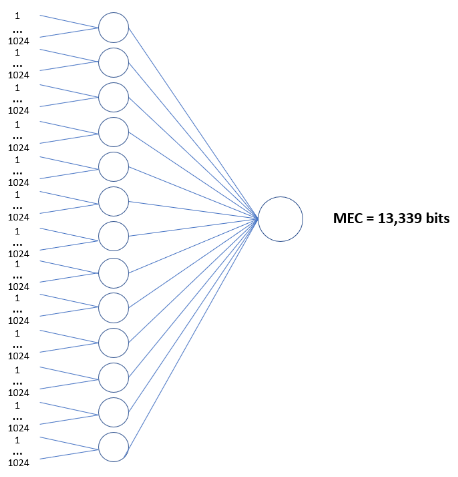
Although we determined that training a model of this size should be able to memorize the entire data set, it ended up only being able to memorize ~75% of the data set. We're unsure of why this occurred, but we have an intuition of why this has happened:

- First, the multiclass problem might have a higher upper bound on MEC needed than we were able to compute with our current understanding of MEC calculations. One deeper reason could be that the generalization capacity is C/C-1, which with 10 classes is 1.1111. That may imply that also much more capacity is required to memorize the dataset, compared to a binary classification problem.
- Second, and most likely, the values in the dataset range from 0 to 255 (grayscale). Given this larger space of real numbers compared to, say, a dataset that only contains numbers between -1 and 1, the machine learner has to train way longer to find fitting weights that memorize the dataset (i.e. find the local minimum). This seems most probable, as we above use increasing network size as a workaround to achieve faster convergence. However, we lack the computational power (and time) to get to full memorization for the unscaled values.

Working with the 2-class dataset (JPEG Q20, grayscale)

As a result, we concluded in switching to the binary problem of classifying cats and dogs within CIFAR10 in a standard-scaled version, still using JPEG Q20 in grayscale.

In this case, we leverage the formula for calculating the capacity of the network above again. However, we know that with this dataset, the MEC expected to memorize is 13,312 bits. As a result, with 13,312 bits = x * (1,024 +1) + 1*x + 1, we can choose the number of neurons in the first hidden layer to be 13 (rounded up from 12.97). We note that this value is incredibly close to the expected MEC value from the 10-class problem, which might hint that the issue with the 10-class problem was due to an undersizing of the model.



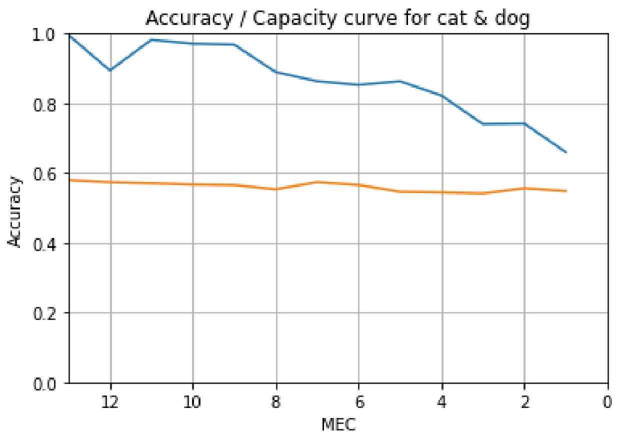
We split the training data into a training and validation set with a ratio of 50/50, resulting in 5,000 images in each set. The reason we do this is so that each set contains about the same complexity of images to learn from and evaluate with.

Using a similar methodology as above with additional feature engineering, including libraries and hyperparameters, we were able to get to 100% memorization of the training set by using 13 neurons. We illustrate the capacity progression vs accuracy curve in the following section. The right is a diagram of the structure of our 100% memorization MLP model.

7. Training for generalization

We trained our model for the 2-class problem on the training data while varying the number of neurons from 1 to 13, and evaluated the accuracy of the training data and validation data. With 13 neurons, we reached 100% memorization with 13 neurons. We use the number of neurons as the proxy for MEC on the x-axis in the graph below.

Our selected model (from section 3) is where the number of neurons is equal to 7. This model configuration was selected because validation accuracy is maximized at this point (57.4%), indicating that this is the best point for when the model is generalizing without overfitting. We also chose the above model based on our hyperparameter tuning, and ended up selecting the following:



Solver = “adam”
Alpha = 0
Max_Iter = 4,000

These parameters yielded the highest validation accuracy. We should note that there is a great amount of latitude available to do further hyperparameter tuning, such as testing with the “sgd” optimizer (we tested with “adam” and “lbfgs”), or other parameters we have not tested.

With 7 neurons, we recorded a 57.4% validation accuracy vs 86.3% training accuracy. We then trained this model on the entire training dataset (training and validation), and received an accuracy of 58.7% on the independent testing set. This is relatively in line with what we expected, as it performs only slightly better on the test than on the validation set.

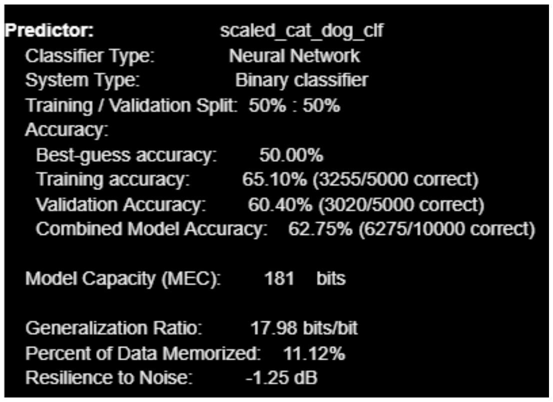
Again calculating the generalization of the mode using G = # correctly classified instances / MEC, we divide the correctly classified number of instances in the training set, 8,016 by our model MEC of 7,183, reaching G = 1.116 bits/bit. The resilience associated with this (given the formula from section 3) is -1.279 dB. This does not reach the targeted benchmark for generalization (2 bits/bit) or resilience (-22.4 dB), confirming our earlier analysis.

Given this generalization level and our test accuracy of 58.7% being only slightly better than chance (50%), we are confident that this model is extracting and learning some level of features from the underlying dataset, but is mostly memorizing on the training set. This is in line with our observations of the validation set as well. This is likely due to the fact that we do not do any actual feature extraction in this model, so the neural network is likely looking for patterns in what may be noise. We explored the implications of a CNN model in the appendix of this article, which yielded initial promising results. We could also explore a random forest for this exercise in order to see if that can uncover patterns and trends easier than a neural network is able to.

Alternatively, we could also choose a model with only 1 neuron (see section 7 for details and rationale). If we trained this network on the entire training set, we would find 6,264 correctly classified instances with an MEC of 1 * (1024 +1) + 1 + 1 = 1,026 bits. With this we find that G = 6.105 bits/bit and so resilience = -15.7 dB. While it still does not reach the -22.4 dB benchmark discussed in class, at least it is closer, and provides the best generalization. As a result, when building a machine learner, we need to think about whether we care more about higher validation accuracy or higher generalizability.

Consequently, if we choose a model to maximize the generalization ratio, we would build it to have 1 neuron in the hidden layer. With this configuration, there are 2,740 correctly classified instances in the validation set with an MEC of 1 * (1,024 +1) + 1 + 1 = 1,026 bits. At this point, G = 2.67, which means it is at its best generalization ratio. Since validation set accuracy does not vary significantly between MEC size, the generalization ratio shrinks the larger the network gets.

To corroborate this notion, we observe that when Brainome generates a model for our dataset, it chooses a model with MEC lower than we're able to generate here (see below). We're unsure of how Brainome is able to generate a model below the size of a neural network with a single neuron, but we assume that some level of model tuning is involved here, such as principal component analysis, which allows model capacity to decrease while maintaining a comparable validation accuracy. Moreover, the Brainome output demonstrates a closeness between training and validation accuracy, which leads us to believe that this model is doing a better job of generalizing. We would like to understand more about how Brainome optimizes this model.



```
Predictor:                    scaled_cat_dog_clf
  Classifier Type:              Neural Network
  System Type:                  Binary classifier
  Training / Validation Split:  50% : 50%
  Accuracy:
    Best-guess accuracy:        50.00%
    Training accuracy:          65.10% (3255/5000 correct)
    Validation Accuracy:        60.40% (3020/5000 correct)
    Combined Model Accuracy:    62.75% (6275/10000 correct)

  Model Capacity (MEC):         181    bits

  Generalization Ratio:         17.98 bits/bit
  Percent of Data Memorized:    11.12%
  Resilience to Noise:          -1.25 dB
```

8. Other quality assurance measures

We did not explore any metrics beyond accuracy and other concepts learned in class in this paper, meaning that we have the further opportunity to look at model-building from the lens of metrics like recall, precision, or other conventionally used metrics. Most of the papers that we evaluated utilized accuracy as the primary metric as well.

For further quality assurance, we could find even more image sets corresponding to cats and dogs (as there are likely to be many on the internet), and apply our model on these to see if the model continues to perform at the same level as expected.

We could also randomize the training/validation/test set again to see if we get significantly different results as a result. We would not expect this to have a significant effect on accuracy, but it is possible given the limited sample size.

9. Repeatability and reproducibility

Because we are leveraging a very publicly available dataset, which is not only widely distributed on the internet, but also built into libraries such as PyTorch, we expect there to be little issues in accessing the data leveraged in the model.

We have provided our code in the form of notebooks in the following GitHub repository: bobni/CS294_Final_Project (github.com). We believe we have thoroughly commented throughout the code so that future readers will be able to follow the steps we have done, and have also included an overview of the files in our README. We performed all of our calculations in Google CoLab, so that our environment is not unique to our machines.

We have also included our training and test sets in these files, which have the level of compression and grayscale that we used directly in this project, as well as the scaled version of the dataset.

10. Learnings and Reflections

In this project, we were able to both build a machine learner for the CIFAR-10 dataset as well as apply the learnings from this class to understand and analyze the learner we had built. We had the chance to really ask why we were doing the things we were doing and to really dig into the details. While we ultimately decided to pivot to the binary problem in this paper, we believe that in the long-term we could understand how to effectively approach the full 10-class problem using similar methodology (e.g. engineering and scaling data appropriately upfront). We learned that although things work in theory, issues such as how Python libraries are implemented or computing limits can inhibit ideal results, and we would seek ways of overcoming or avoiding these in more informed ways the next time around.

In terms of teammate contributions, we believe that our team members (Philipp, Stefan, and Bob) had equal contributions in this project. Philipp focused a great deal on finding the right JPEG compression, getting data engineered correctly, and researching and operating the key tools like Brainome and SciKit Learn that would help us answer many parts of 3, 5, 6, and 7. Stefan built much of the code for the two machine learners that would be trained to 100% memorization using PyTorch and SciKit Learn, created much of the visualizations involved throughout the paper and provided a lot of mathematical thoughtwork on the project itself, encompassing a lot of work in 1, 2, 6, and 7. Bob built a lot of the code for measurement on metrics like MEC generalization and resilience, developed and tested the range of models in the experiment, and developed and analyzed the CNN approach, all of which provided much of 1, 3, 7, 8 and 9, along with compiling, editing, and formatting this paper and Github.

Appendix (Optional)

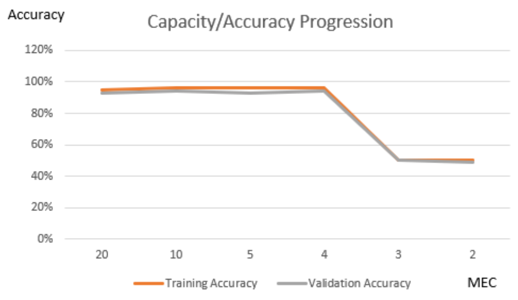
In this section, we detail an alternative approach to JPEG Q20 compression, where we leveraged a CNN instead of JPEG. Here we share details about the MEC of the dataset, the expected MEC of the model, training the model to 100%, and the capacity/accuracy of this model. Note that we performed this on the plane/car dataset.

Our CNN structure is as follows:

Layer	Parameters and Steps
Convolution	In = 3 channels, Out = 50 channels, 5x5 kernel -> Relu -> Dropout (0.1)
Convolution	In = 50 channels, Out = 50 channels, 5x5 kernel -> Relu -> Dropout (0.1)
MaxPool	2x2, Stride of 2
Convolution	In = 50 channels, Out = 50 channels, 5x5 kernel -> Relu -> Dropout (0.1)
Convolution	In = 50 channels, Out = 50 channels, 5x5 kernel -> Relu -> Dropout (0.1)
MaxPool	4x4, Stride of 4
Linear	In = 800, Out = X
Linear	In = X, Out = 2

We vary the value of X (the number of neurons in the fully-connected layer.

When we calculate the MEC of the output of the CNN, we derive a total of 3,787 thresholds across the two classes. Taking log_2 of this value yields approximately ~12 bits, which is the bits needed with a perfect machine learner. We multiply this by the number of data columns in the data set, which is 800 (the output of the CNN), yielding a total of 9,600 bits needed to memorize this data set as a dictionary.



Using the formula derived above to determine the number of neurons needed to reach this MEC, we have 9,600 bits = x * (800 +1) + 1*x + 1, where x solves to be 12 neurons needed to reach expected MEC of the data set.

Interestingly, when we look at the capacity/accuracy of this model, we see the graph to the right, where the model shows a clear spike in accuracy once the number of neurons reaches 4. Afterwards, both the training and test accuracy seem to level off, slowly reaching 100%.

We believe this spike in accuracy of the model well below the expected MEC needed indicates that our pre-trained CNN is incredibly powerful in extracting features from the images, and therefore the neural network itself does not have to do much work, as it learns the patterns from the features easily.

We select the model where neurons = 4, because there appears to be no incremental benefit to the validation set accuracy following this. With 9,520 correctly labeled instances (across test and control) and an MEC of 4 * (800 +1) + 1 * 4 + 1 = 3209 MEC, we derive a G = 9,520/3,209 = 2.967 bits/bit, and a resilience of -9.45 dB. Therefore, this model is generalizing well, though it still doesn't reach the mark of -22.5 dB in resilience.