STAG: Smart Tools and Applications in Graphics (2021)
P. Frosini, D. Giorgi, S. Melzi, and E. Rodolà (Editors)

An Information-theoretic Visual Analysis Framework for Convolutional Neural Networks

Jingyi Shen[†] and Han-Wei Shen[‡]

The Ohio State University

Abstract

Despite the great success of Convolutional Neural Networks (CNNs) in Computer Vision and Natural Language Processing, the working mechanism behind CNNs is still under extensive discussion and research. Driven by strong demand for the theoretical explanation of neural networks, some researchers utilize information theory to provide insight into the black-box model. However, to the best of our knowledge, employing information theory to quantitatively analyze and qualitatively visualize neural networks has not been extensively studied in the visualization community. In this paper, we combine information entropies and visualization techniques to shed light on how CNN works. Specifically, we first introduce a data model to organize the data that can be extracted from CNN models. Then we propose two ways to calculate entropy under different circumstances. To provide a fundamental understanding of the basic building blocks of CNNs (e.g., convolutional layers, pooling layers, normalization layers) from an information-theoretic perspective, we develop a visual analysis system, CNNSlicer. CNNSlicer allows users to interactively explore the amount of information changes inside the model. With case studies on the widely used benchmark datasets (MNIST and CIFAR-10), we demonstrate the effectiveness of our system in opening the black-box of CNNs.

CCS Concepts

• ***Human-centered computing*** → ***Visual analytics;***

1. Introduction

The Convolutional Neural Networks (CNNs) is a type of deep neural networks that have shown impressive breakthrough in many application domains such as computer vision, speech recognition and natural language processing [KSZQ19]. Different from the traditional multilayer perceptrons which only consist of fully connected layers, CNNs have additional building blocks such as the convolution layers, pooling layers, normalization layers, dropout layers, etc. Through a combination of these layers, CNN models can extract features of different levels to generate the final result. Despite its great success, because of the lack of explanation of CNNs, designing and evaluating the model is still a challenging task.

As a result, the interpretation of the black-box systems in deep neural networks (DNN) models has received a lot of attention lately. Researchers proposed various visualization methods to interpret DNNs such as CNNVis [LSL*17], LSTMVis [SGPR18], GANViz [WGYS18], etc. By making the learning and decision making process more transparent, the reliability of the model can be better confirmed which allows researchers to improve and diagnose the models more effectively.

Since a CNN model can extract features of different levels, it can be thought of as an information distillation process. Measuring the information change during this information distillation process makes it possible to open the black box of neural network models. In 2017, the work by Shwartz-Ziv and Tishby [ST17] tackle the problem using information theory to analyze the learning process of DNNs via a technique called *Information Plane*. However, their focus is to give a theoretical bound of neural networks instead of a though visual analysis of information flow in the network at different granularities. Although plenty of works have been done to visualize DNNs, to the best of our knowledge, using information theory as a comprehensive analytic tool to visualize and analyze deep learning models has not been fully studied in the visualization community. In this paper, we aim to bring together information theory and visual analytics for analyzing CNN models.

To better understand the information distillation process of CNNs, we start with treating the model as a black box and analyze its input and output. Then we get into the details of the model's building blocks such as layers and channels. There are multiple angles to assess the information, e.g., measure the amount of information inside the input data, between the input and output, between the intermediate layers, and among the channels. To systematically formulate the queries, we introduce a data model in the form of a four-dimensional hypercube that allows users to systematically

† e-mail: shen.1250@osu.edu
‡ e-mail: shen.94@osu.edu

DOI: 10.2312/stag.20211486

query the data from various stages of CNN models. The dimensions of this data model represent the input, layers, channels, and training epochs. Different slicing or aggregation operations on this hypercube are introduced to glean insights into the model. To calculate the information flow through CNN models, we propose two different types of entropy: inter-sample entropy and intra-sample entropy. Inter-sample entropy is the entropy from a set of samples, including the input and the intermediate results, in their original high-dimensional space, and intra-sample entropy is the entropy for each data sample such as an image or a feature map. We design and develop a visual analysis system, CNNSlicer. CNNSlicer aims to reveal the information inside the model and help users understand the learning process of a CNN model or evaluate a well-trained CNN model in the testing phase.

In summary, the contributions of our work are: (1) We propose a novel hypercube data model to help users make more systematic queries to the information available in CNN models. (2) We propose two types of entropy, inter-sample entropy and intra-sample entropy, each reveals a different insight into the information distillation process of CNNs. (3) We combine visual analysis of CNNs with information theory and develop a system, CNNSlicer that allows users to perform visual queries into the CNN models.

2. Related work

2.1. Visual Analytics for Deep Learning

In both visualization and machine learning fields, an increasing number of researchers are focusing on the visual analysis of deep learning models to diagnose, evaluate, understand and even refine the model. In the visualization community, scientists integrate multiple visual components with proper interactions to let user explore and understand the model. For example, CNNVis [LSL*17] was proposed to assist users in exploring and understanding the role of each neuron in the model. By only analyzing the input and output, Manifold [ZWM*19] was designed to help the development and diagnosis of the model. To understand and interpret the Deep Q-Network (DQN), DQNViz [WGSY19] encodes details of the training process in the system for users to perform comprehensive analysis. In the machine learning field, there are some popular works on Explainable Artificial Intelligence (XAI) [ADRS*19], such as saliency map [FV17], loss landscape visualization [LXTG17], sensitivity analysis [CE11] and deconvolution [ZF13].

2.2. Information Theory and Applications in Deep Learning

Information theory, first introduced by Claude Shannon in 1948, is a theory focusing on information quantification and communication [Sha48]. Shannon defined several fundamental concepts such as entropy, relative entropy and channel capacity to measure information in a statistical way [Sha48]. Even to date, information theory is still popular in areas like mathematics, computer science (e.g., feature selection [HXZ20]) and electrical engineering (e.g., compression [HXZ20]), etc. It also has been employed in deep learning field. For example, cross-entropy, in information theory, measures how different two probability distributions are. In deep learning, the cross-entropy loss is widely used in classification tasks to measures the distance between the predicted labels and the true labels.

Besides the loss design, there has been some attempt to open the black box of deep learning models via information theory. One of the pioneering works done by Tishby and Zaslavsky [TZ15] is the Information Bottleneck. In their work, a layered neural network model is taken as a Markov Chain with every layer only depends on the output of the previous layer. The evolution of the mutual information between the input and hidden layers, and between hidden layers and the output during training is depicted in the information plane to investigate the training dynamics. With this information plane, the authors observed that DNNs aim to first compress the input into a compact representation and then learn to increase the model's generalizability by forgetting some information. However, their goal is more on giving a theoretical background of neural networks from an information perspective. The information plane visualization is limited to show only the mutual information trajectories between several layers. Different from information plane, we want to utilize information theory and some visualization methods to open the black box of CNNs from different granularities such as layers, channels, and different training epochs.

3. Background

3.1. Convolutional Neural Network

Convolutional Neural Networks (CNNs) have various building blocks including convolutional layers, pooling layers, non-linear activations, dropout, normalization, etc. With these layers stacked up multiple times, CNNs are able to automatically extract hierarchical features from the input. Consider a standard CNN model as an information distillation process, each layer is extracting and purifying the input information into concise representations. Initially, the model has no idea what to filter out given a large amount of information. During training, the model's weights get updated through backpropagation and its output converges to the expected result. When the model is well trained, it can reduce and summarize the input by keeping only the most salient characteristics of the data to perform the underlying inference task. This fundamental nature of CNNs is what motivates us to develop a framework to evaluate and explain how a neural network model process information and how the information flows through the model.

3.2. Information Theory

In information theory, entropy (Shannon's entropy [Sha48] to be more specific) is a widely used metric to measure the amount of disorder (information) in the system. In a random process, if an event is more common, the occurrence of this event contains less information than a rare event. That is to say, given a random event, higher probability means lower information content. From this intuition, the information content of a stochastic event x can be defined as the negative log of its probability $p(x)$:

$$I(x) = -log_b(p(x)) \tag{1}$$

In Shannon's entropy [Sha48], the logarithmic base of the log is 2, so the resulting unit is "bit".

For a stochastic system, if each random event can happen with almost the same probability, this system is almost unpredictable. From the information theoretic point of view, the system is more

disorder. On the other hand, when only a few events are more likely to happen, the output will be less surprising and hence the system contains less information. Thus, the amount of information for a system, also known as the entropy [Sha48], is defined as the expected value of all random events' information content:

$$H(X) = E[I(X)] = E[-log_b(P(X))] = -\sum_{i=1}^{n} P(x_i) log_b P(x_i) \quad (2)$$

where n is the number of possible stochastic events of the system.

In [Sha48], Shannon also defined a theoretical upper bound for a communication system. The system has six components as shown in Fig. 1. The *information source* produces the original information to be transmitted to the destination. A *transmitter* encodes the information into a suitable representation that can be transmitted from the transmitter to the *receiver* through the *communication channel*. The receiver is a decoder which decodes the received information and sends it to the *destination*. During transmission, the addition of *channel noise* will cause signal interference. Shannon defined the capacity (i.e., mutual information) of a noisy channel by [Sha48]:

$$MI = H(X) - H(X|Y) \quad (3)$$

where $H(X)$ measures the amount of information in the information source and the conditional entropy $H(X|Y)$ gives us the amount of uncertainty for the source information X given the destination information Y. In other words, $H(X|Y)$ measures the amount of information loss during the transmission. If Y has exactly the same amount of information as X, then given Y there would be no uncertainty about X. In this case, $H(X|Y)$ equals zero. However, as we have inevitable noise added during transmission, in practice $H(X|Y)$ is not zero.

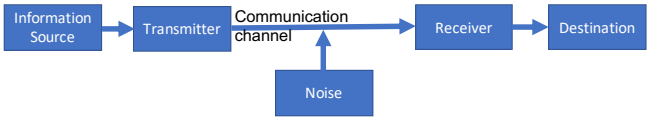

Figure 1: *A communication system transmits encoded source information to the destination through a noisy communication channel.*

As we discuss in Section 3.1, CNNs can be seen as extracting and purifying the input information through consecutive layers. In information theory, entropy can measure the amount of information in a system. In our case, the system is a neural network model. Since the information is "compressed" into the output-relevant information, one may wonder whether the entropy keep decreasing as we get into deeper layers. On the other hand, through iterative training, the optimized model is more sure about what to output given an input. This means the randomness of the system is decreasing when making inferences. Can the entropy of the model reflect this change? Furthermore, how does the entropy of the feature maps from a specific layer change during training? If we can answer the above questions, we will improve the transparency of the CNN model through information theory.

4. A CNN Information Analysis Framework

4.1. Requirement Analysis

Our work tries to help users analyze and visualize CNN from an information theoretic point of view. After thoroughly discussing with a machine learning and information theory expert, we come up with the following visual analysis requirements:

- **R1: Provide an overview as well as the basic building blocks of a CNN model.** CNN models have hierarchical structures. To go along with this property, our visual design also requires such a hierarchy. From the overview of the model to the details of the architecture, this top-down visualization can assist users to understand the model in a more intuitive way.
- **R2: Develop a unified data model for CNNs.** As we discuss before, there are various information queries one can make to improve the transparency of the CNN model. Visualizing all possible queries in a theoretical way would help users to gain a deeper understanding of the model. However, a large amount of data are generated during the training process and CNN architectures vary depends on the application. Thus, a unified CNN model representation is needed.
- **R3: Visualize information statistics of the model.** To give a clue about how the CNN model makes decisions from an information theoretic perspective, the statistics associated with each building block of the model need to be analyzed. Since there is a huge amount of statistical information, we need better data organization and effective visual designs to avoid visual clutter. With detailed statistics, users can find interesting directions for further analysis and potential explanations for the model.
- **R4: Support multifaceted analysis of the model's learned features.** Besides the statistical information, to get insight into the functionality of the intermediate layers, comprehensive visualization to display detailed multifaceted information in the model is required. For example, the visualization of the various features that the model learned.

4.2. dCNN: A Data Model for CNNs

As we stated before, it is important to formulate the queries systematically to evaluate and interpret the information flow inside a CNN model. In this section, we formalize a data model to represent the information available in CNN models in a comprehensive way, regardless of the specific type of CNN architecture. This data model helps us organize the information queries and make the CNN explanation process more systematic.

The data available in the entire CNN model can be thought of as a four-dimensional array, denoted as $dCNN(X, L, C, T)$ where each dimension of the array serves as a key. A specific value of the key is used to slice the array and obtain the information stored within.

The first dimension of the array is the input data (training or testing) dimension, denoted as X. Within this dimension, each specific instance, or a value in the dimension represents an input, for example, an input training image. In the case of classification neural networks, the dimension X can be further divided into subgroups where each subgroup represents data in a specific class.

The second dimension in the multidimensional array is the layer dimension, denoted as L. As we know, a CNN contains multiple layers, from the input to hidden to the output layers, and each of the layers plays a specific role. For example, the earlier layers are responsible for extracting low-level features such as edges and colors. As we go deeper into the model, low-level features are com-

bined into high-level features such as contours or textures and then objects can be extracted. As each layer acts like a 'function' whose output only depends on the input from the previous layer, the input information gets distilled layer by layer.

Given a particular layer in the CNN, there exist multiple convolution kernels, also known as filters and the output of which is often called a channel. If we fix at a layer, there can be multiple channels to be chosen for analysis, which prompts the necessity of having the next dimension in the array that represents the channels, denoted as C. For any CNN model, it is crucial to have multiple filters to be trained, and as a result, multiple feature maps will be produced for a given input. Intuitively, each channel is activated by some specific features in the input. It is also known that more filters do not necessarily guarantee more information from the input to be captured since some filters may be 'dead' where no information is extracted. Since the capability of a filter can be checked by the corresponding feature maps, it is reasonable that we focus on feature maps when evaluating the filters.

Finally, a CNN model is trained through many iterations of backpropagation and gradient descent. When the input training data are divided into many small batches, called mini-batches, a backpropagation is conducted for each mini-batch and an iteration that goes through all mini-batches is called an epoch. To track the progress of training for a CNN model over time, we index the data in the last dimension of the four-dimensional array by its epoch number, denoted as T.

As a result, we propose to use a four-dimensional hypercube to represent the data related to a CNN model. The four dimensions are data (X), layer (L), channel (C), and training epoch (T). We denote this data structure as $dCNN(X,L,C,T)$. Inside the four-dimensional array, each specific entry is often an array, for example, an input image, a feature map in the hidden layer, or an array of probability values, one for each possible output label. Different slicing or aggregation operations on this hypercube lead to different information queries and facilitate the evaluation and interpretation of the CNN model.

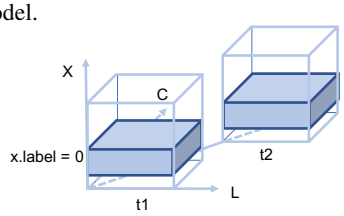

Figure 2: *Slice the data model by input (X) dimension to get all intermediate data in the CNN model, given input class label 0.*

4.3. Slicing dCNN as Information Query

The data model described in Section 4.2 defines a 4-dimensional hypercube to represent all data that can be extracted from a CNN model. To analyze the data in a CNN model, we can first slice the hypercube based on the analysis need, followed by calculating various entropy measures from the slicing result. Below we first explain the meaning of data slicing. The calculation of entropies is explained in Section 4.4.

- **Slicing by Input** X**:**$dCNN(x,-,-,-)$ means we slice the hypercube on the X dimension where x is a particular instance or a set of instances. The notation $-$ indicates all values in the respective dimension. For example, $x = \{x' | x'.label = 0\}$ returns all the data with label 0 from all epochs, all layers and channels. $H(dCNN(x,-,-,-))$ denotes its entropy. It may not be very helpful to calculate $H(dCNN(x,-,-,-))$ across so many layers and channels, we need to further constrain the query.
- **Slice by Input, Layer, and/or Epoch:** When slicing on both X and L dimensions, we have $dCNN(x,l,-,-)$, where $l \in [0,|L|]$ specifies which layer we are interested in for a given input x, and $|L|$ is the index of the last CNN layer. $dCNN(x,l,-,-)$ represents all feature maps at layer l for all epochs given input x. To track the training process across different epochs, we need to further index the data on the T dimension. $H(dCNN(x,l,-,t))$ is the entropy of output from layer l during training epoch t given the input x. $dCNN(X,0,-,0)$ is the initial condition where the input is the entire set X which have not gone through any layer of the model. So the layer index and the training epoch are all zeros. In this case, $H(dCNN(X,0,-,0))$ measures the diversity of the input X. Besides entropy, it is also possible to calculate the conditional entropy $H(a|b)$, where $a = H(dCNN(x,l_1,-,t))$ and $b = dCNN(x,l_2,-,t))$. It describes given the output of layer l_2, how much we know about the output of layer l_1 at training epoch t. The mutual information $H(a) - H(a|b)$, as defined in Equation 3, measures how much information gets lost between layer l_1 and l_2 at training epoch t. This fulfill our goal of querying the information change between layers.
- **Slice by Input, Layer, Channel, and/or Epoch:** Previously, we have $H(dCNN(x,l,-,-))$ and $H(dCNN(x,l,-,t))$ measure the amount of information that a layer contains. However not every channel in this layer is useful. We can further slice the four dimensional array along the C dimension to investigate a particular channel. $dCNN(x,l,c,t)$ returns the output (feature map) of channel c in layer l at training epoch t given input x. $c \in [0,|C_l|]$ specifies which channel we are focused on, and $|C_l|$ is the number of channels in layer l. $H(dCNN(x,l,c,t))$ measures the information content of the output in channel c.

4.4. Entropy Calculation for dCNN

In this work, we adopt entropy as a measure of information content for CNNs. The calculation of entropy, however, needs to be specially tailored to the need of analysis. Specifically, for the four-dimensional data model mentioned above, the calculation of entropy depends on how the array is sliced.

We propose two types of entropy calculations for our data model. The first entropy that can be calculated is called inter-sample entropy, which is to calculate how diverse the data are in the high dimensional space. For example, the entropy of the training image set for classification where each image is a sample from a high dimensional space. This type of entropy is often an indicator of whether the data from the input or in the immediate layers of CNNs is sufficiently diverse, and how the information represented in a set of samples is distilled across the different layers of a CNN. To calculate this entropy, we need to consider the distribution of the data samples in the high-dimensional space. The second type of entropy is intra-sample entropy. The intra-sample entropy considers the entropy within each data sample. For example, the entropy of an input image or the entropy of a particular feature map. To calculate the

intra-sample entropy, we use histograms of the data sample. In this case, an image is not considered as a high dimensional point but a one-dimensional histogram of the pixel values where the entropy can be efficiently calculated. Below we describe the calculation methods in detail.

4.4.1. Inter-Sample Entropy Calculation

The purpose of inter-sample entropy calculation is to measure the diversity of the data samples in their original space, which is often high-dimensional. As the set of samples are going through the different layers of a neural network, redundant or irrelevant information for the inference task is discarded which will, in turn, change the distribution of the layer output. Monitoring how the entropy is changed often provides an important hint on how the neural network is doing. As an example, considering our *dCNN* data model and assuming X is the set of *all* training images in the MNIST dataset where each data sample in X is an image of size 28*28 in a 784-dimensional space. If we want to measure the diversity of the training set X, we can first slice the CNN data array by $dCNN(X,0,-,0)$, and then calculate the inter-sample entropy, denoted as $H(dCNN(x,0,-,0))$, to measure whether the input samples have sufficient diversity. Below we explain how the inter-sample entropy is computed.

As defined in Equation 2, when calculating the entropy, we need the probability for each of the states in the system. To obtain the probabilities for calculating the inter-sample entropy, we adopt an idea inspired by a dimensionality reduction technique, Uniform Manifold Approximation and Projection (UMAP) [LMM18]. In UMAP, to create a distribution for high dimensional points, the distance between a pair of samples i and j is converted into a probability by an exponential distribution function. With the probabilities between all sample pairs, we can then calculate the inter-sample entropy as the expected value of the information content for all sample pairs.

The exponential distribution is widely used to model relationships between random variables. For any data point x_i, the similarity between x_i and another point x_j is given by the conditional probability $P_{j|i}$:

$$P_{j|i} = exp(\frac{-\|x_i - x_j\|}{\sigma_i}) \tag{4}$$

where σ_i is the scale parameter of the distribution for each x_i. An adaptive exponential kernel for each point are more powerful to model the real data distribution in high-dimensional space, as a result, each σ_i is set to satisfy Equation 5:

$$\sum_{j=1}^{k} exp(\frac{-\|x_i - x_j\|}{\sigma_i}) = log_2(k) \tag{5}$$

where k is the number of data points.

To make the above probability calculation computationally efficient, for each sample, we only consider its k-nearest neighbours in the high-dimensional space. Before computing entropy, the probabilities are symmetrized and normalized. To make them symmetric, we average $P_{j|i}$ and $P_{i|j}$:

$$P_{i,j} = \frac{P_{j|i} + P_{i|j}}{2N} \tag{6}$$

where N is the number of data points. Then we normalize the joint probability table. Now we have probability for each of the states, we plug Equation 6 into the entropy Equation 2 to calculate inter-sample entropy. The inter-sample entropy calculation is similar to the joint entropy, which considers the probabilities of all pairs of data samples in dataset X.

$$H(X) = -\sum_{i=1}^{n}\sum_{j=1}^{n} P_{i,j} log_2 P_{i,j} \tag{7}$$

4.4.2. Intra-Sample Entropy Calculation

The intra-sample entropy measures the randomness of the values in each data sample, regardless of the data sample's dimensionality. For example, when estimating the entropy for an image or a feature map, we only take the value distributions of the pixels into consideration. We use histograms for probability estimation which is also often adopted to measure the quality of images for image processing applications.

The calculation details are as follows. First, we normalize the value range to $(-1,1)$, divide the range into $B(e.g., B = 32)$ equal-size intervals (bins), and then put the values of a data sample (e.g. an image's or a feature map's pixel values) into these bins. The resulting frequency distribution is a histogram. Then we use the normalized frequency distribution as the probability distribution to calculate the intra-sample entropy. This approach is straightforward and computationally efficient which reflects the sharpness or blurriness of value distributions. One example of using intra-sample entropy is when we want to calculate the randomness within a group of feature maps, which is denoted as $H(dCNN(x,l,c,t))$ where x is a set of input images from the same class. This is useful for the evaluation and understanding of CNN's filters. Because a 'dead' filter will not be activated by different inputs, the value distribution on one channel can give some idea about the corresponding filter. It is feasible that we use intra-sample entropy for each feature map and use the averaged result as the final $H(dCNN(x,l,c,t))$.

5. Information Query via CNNSlicer

In the previous section, we formulate a four-dimensional hypercube $dCNN(X,L,C,T)$ to represent a CNN model. By slicing on the hypercube, we are able to perform information analysis of a CNN. To meet our requirements stated in Section 4.1, we develop a visual analysis system, called *CNNSlicer*, with four visual components. Our framework is developed based on d3.js with Flask framework as backend to support the interactions with the data and views. The neural network model is implemented using Pytorch [PGM*19]. Wee use Matplotlib to pre-process and generate some analytic figures. In this section, we describe the four visual components of CNNSlicer and show how various information measures can assist the analysis of CNN models.

5.1. Training Performance View

To help users evaluate the training performance of the model **(R3)**, we design the training performance view in Fig. 3(A). In this view, users can analyze the model's quality, the diversity of the input data and the training loss.

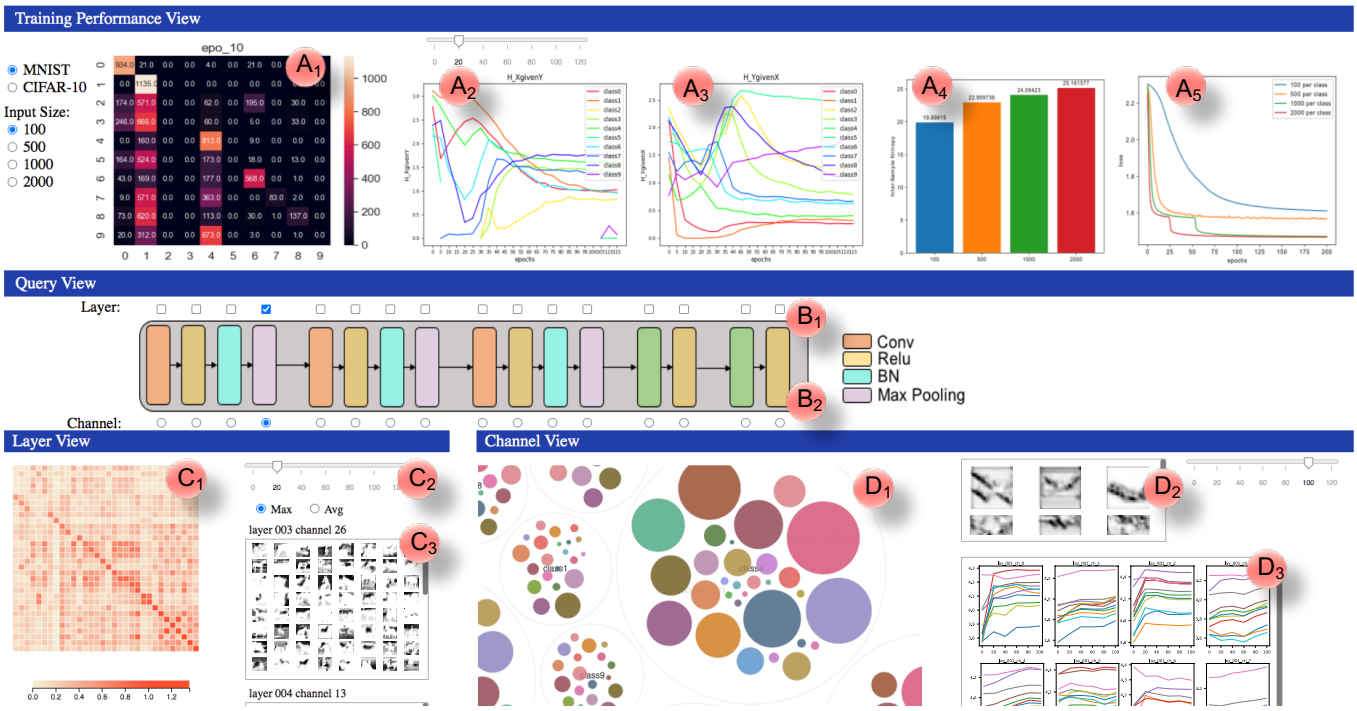

Figure 3: *System Overview. (A) Performance overview of the selected model; (B) Query View: for user to perform queries; (C) Layer view: display multifaceted information of layers; (D) Channel view: show detailed intra-sample entropies and deconvolutional results of channels.*

5.1.1. Evaluate CNN Model Quality

The quality of the CNN model is evaluated by a confusion matrix in Fig. 3(A_1), and the information queries between the model's input and output as shown in Fig. 3(A_2-A_3).

The confusion matrix (Fig. 3(A_1)) shows how confused the model is between two classes. Each row of the confusion matrix represents all instances of predicting an actual class to all classes. The cell color indicates how many instances are predicted as the column class but actually belong to the row class. The higher value the brighter is the color.

Consider the CNN model as a communication system, where the input x is transmitted to the output layer through the model. The information queries between input x and output y will reflect the model's information distillation ability. To perform information queries about input and output layers, we slice the data model $dCNN$. $dCNN(x,0,-,0)$ returns the input x. Given the input x, the output y of the CNN model at training epoch t can be denoted as $dCNN(x,|L|,-,t)$, where $|L|$ is the index of the output layer.

As discussed in Section 3.2, $H(x.label|y=i)$ measures given the predicted labels y being class i, how much uncertainty we have about the ground truth labels $x.label$. On the other hand, $H(y|x.label=i)$ measures the randomness of the model's predictions in the output for the data samples in class i at epoch t. The calculation of $H(x.label|y=i)$ and $H(y|x.label=i)$ can be done via Shannon's entropy. That is, for a given class i, to calculate $H(y|x.label=i)$ we need $p(y=j|x.label=i)$ where $j \in [0,9]$. These probabilities can be obtained from the model output. Then we can directly use the Shannon's entropy equation to evaluate.

The conditional entropy plots (Fig. 3(A_2-A_3)) shows the uncertainties of the model. The horizontal axis in each plot is the training epochs and the vertical axis is the conditional entropy ($H(x.label|y=i)$ or $H(y|x.label=i)$). These line charts can assist users to monitor the uncertainty change over training epochs.

5.1.2. Evaluate Input Diversity

Increasing the training data size will usually improve the model's robustness and accuracy. However, if the training data is not well distributed, it can lead the model to optimize in a non-optimal direction and affect the speed of convergence. Therefore, it is worth to evaluate the diversity of the input (i.e., inter-sample entropy $H(dCNN(x,0,-,0))$ in high-dimensional space). The inter-sample entropy of input is displayed in Fig. 3(A_4).

5.2. Query View

There are various information queries users can make according to different slicing operations. To assist users to make proper information queries from different granularities, we design a query view as shown in Fig. 3(B). This view is linked to the layer view and channel view. In this view, all layers of the CNN architecture are visually available. To support the top-down analysis **(R1)** of the CNN model, users can select layers by checking the checkboxes (Fig. 3(B_1)) or select channels from one layer by clicking one of the radio buttons (Fig. 3(B_2)).

5.3. Layer View

In the previous sections, we have discussed about slicing the data model and performing information queries about the input ($dCNN(x,0,-,0)$) and the output ($dCNN(x,|L|,-,t)$). In this section, we focus on the information flow between the intermediate layers. To visualize the amount of information transmitted between layers, we design a layer view as shown in Fig. 3(C). After slicing the layer dimension of the hypercube by selecting layers in the query view, the layer view will be updated.

We adopt mutual information to evaluate the information between layers and show the result in a heatmap (Fig. 3(C_1)). The slider in Fig. 3(C_2) is used to visualize the mutual information over training epochs. The heatmap can be sorted by rows or columns based on the average or maximum value within the row or column vectors. The layer view also contains a visualization of feature maps **(R4)** to help users gain more insight of the layer.

5.3.1. Evaluate the Information Between Layers

Suppose users are interested in how the information flows from layer l_i to layer l_j during training. Then all layers before l_i can be taken as a transmitter and all layers after l_j can be taken as a receiver. The stacked layers between layer l_i and l_j represent the noisy communication channel. The mutual information (Equation 3) between layer l_i and layer l_j at epoch t is used to measure the transmitted information, i.e., $H(X)-H(X|Y)$ where $X = dCNN(x,l_i,-,t)$ and $Y = dCNN(x,l_j,-,t)$.

Normally, each layer's output has multiple channels generated by different filters. To reduce the analysis complexity and make the explanation process more clear, we consider each channel as one representation of this layer's output. That is, given a group of input instances, the mutual information between layer l_i and layer l_j is calculated based on the feature maps from all channel pairs, one channel from layer l_i and another from layer l_j.

However, feature maps from layer l_i and layer l_j have different dimensionality. To compute their joint distribution, we adopt a similar idea that uses distances in high-dimensional space between two samples as the similarity measurement. Instead of converting distances into probabilities using exponential or Gaussian distributions, we map the distances into a 2-dimensional histogram, where one axis of the histogram represents the bins that discretize the pairwise feature map distances in one channel from layer l_i, and the other axis represents the bins of the distances in one channel from layer l_j. Any entry in the 2D histogram records the frequency of the joint distance pairs from the two channels, one from layer i and another from layer j, computed from the feature maps. After the 2D histogram is established, we can compute the joint entropy of two channels, the entropy of each channel, the conditional entropy between channel pairs, and the mutual information between one channel in layer i and one channel in layer j.

The 2D histograms are useful in this case since it combines these two high-dimensional spaces and studies the distance correlation between two filters. One alternative approach is to use a 2D-Gaussian distribution that take the distance pairs from two layers as input to do the probability mapping. Due to the computational complexity, our approach is based on 2D histograms.

The sorted heatmap in Fig. 3(C_1) is used to visualize the matrix of mutual information between the channels from two layers. The horizontal axis of this heatmap are channels of layer l_i and the vertical axis are channels of layer l_j. The color at position (x,y) in the heatmap encodes the mutual information between layer l_i and layer l_j using the feature maps from the x-th filter of layer l_i and the y-th filter of layer l_j. A brighter color indicates a smaller mutual information between these two channels which means the channel undergoes higher information loss or less correlation between the two channels' outputs. One column of this heatmap represents the mutual information calculated based on feature maps from one filter in layer l_i and feature maps from all filters in layer l_j. If there are 64 filters in layer l_j, then there will be 64 mutual information values in this column. By sorting the matrix columns by the maximum values we mean: (1) compute the maximum value of each column; (2) sort these maximum values and keep the sorted indexes; (3) rearrange the columns based on these indexes. We perform similar operations for rows. After sorting, patterns of mutual information between layers become clear. When users hover over one rectangle of the heatmap, the text showing layer index, channel index and mutual information will be updated in Fig. 3(C_3). When they click on one rectangle, feature maps from the corresponding"column filter" and "row filter" will also show up in Fig. 3(C_3).

5.4. Channel View

If we further slice $dCNN(x,l,-,t)$ on the channel dimension, we get $dCNN(x,l,c,t)$ which are all the feature maps generated by filter c. Since filters are the most fundamental building blocks of a CNN model, it is important to investigate the information of filter c's output (i.e., channel c). When users click one of the radio buttons layer in the query view, the information statistics for each channel of this layer **(R1, R3)** is updated in channel view, as shown in Fig. 3(D). To reveal what information gets filtered out during the information distillation process, we adopt a Deconvolutional Network (deconvnet) [ZTF11] to project a filter's output back to the input image space **(R4)** in Fig. 3(D_2). In the channel view, we utilize a small multiple chart as shown in Fig. 3(D_3) to visualize all channels' intra-sample entropies' changes over training epochs. We also employ circle-packing diagrams with two levels of hierarchy, as shown in Fig. 3(D_1) to compare intra-sample entropies between different input classes.

5.4.1. Evaluate the Information in Channels

To calculate $H(dCNN(x,l,c,t))$ (i.e., the intra-sample entropy of channel c), we first compute the intra-sample entropy at channel c for every input, then we take the average of these entropies. To evaluate a channel's performance on different classes, we compute the channel's intra-sample entropy for input with different classes.

In the small multiple chart (Fig. 3(D_3)), each box represents a channel, the x-axis of each line in the box represents epochs, y-axis presents the entropy, and different colors represent different classes. To reduce visual clutter, the system only shows four channels at a time and users can scroll down to analyze other channels.

The input data are further divided into subgroups according to their true labels and every subgroup is represented as a circle in the

circle-packing diagram. In Fig. 3(D_1), the first-level circle represents the class. The size of first-level circle represents the entropy of all feature maps belong to this class. Inside each first-level circle, there are smaller second-level circles representing channels. The size of the second-level circle has a positive correlation with the intra-sample entropy of the channel (i.e., weighted by a power function to make circles more distinguishable). Assuming a CNN has 32 channels in this layer, then there will be 32 second-level circles inside each first-level circle. The color of the second-level circles represents the channel index. If users click on the first-level circle of one class, the circle-packing diagram will zoom in and show the inner second-level circles. By hovering over the second-level circles, users can check the channel's intra-sample entropy.

We utilize a deconvnet to visualize what information the feature map contains about the input. In [ZF13], given one input image, feature maps of all channels in a layer are pushed back to the input image space by successively unpooling and filtering to get the deconvolutional result. In our work, given one input image, we only use the feature map from the channel we are interested in to get the deconvolutional result in the input image space, this can show how much information from the input remains in this channel. When users click on one second-level circle, the deconvolutional results of this filter will be updated in Fig. 3(D_2).

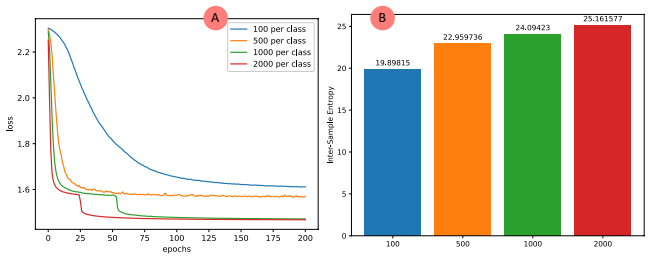

Figure 4: *(A) Training loss for datasets with 100, 500, 1000, and 2000 samples per class; (B) inter-sample entropy for datasets with 100, 500, 1000, and 2000 samples per class.*

6. Case Studies

6.1. Diversity of the Input

Take the MNIST dataset as an example. This dataset has ten classes. Experts random sample 100, 500, 1000, and 2000 samples respectively from each class resulting in four different datasets ($x1, x2, x3, x4$). To analysis the diversity of the datasets, they calculate the inter-sample entropy $H(dCNN(x_i, 0, -, 0))$ with $i \in [1,4]$. In Fig. 4(B), the horizontal axis shows the number of samples per class and the vertical axis is the inter-sample entropy for the dataset. They train the model based on these datasets separately. The training loss is shown in Fig. 4(A) where the horizontal axis represents the training epoch and the vertical axis is the loss value. Fig. 4(A) and Fig. 4(B) share the same color scheme which makes it easier for users to connect the training performance with the diversity of training data. For example, blue represents the dataset with 100 samples per class. The blue line in Fig. 4(A) takes more than 150 epochs before the loss becomes flat and stable (a slower convergence speed compare with others). The blue bar in Fig. 4(B) shows a lower inter-sample entropy (i.e., 19.89815) compared to other datasets (i.e., 22.959736, 24.094230 and 25.161577), meaning the dataset is less diverse. On the other hand, the dataset with 2000 samples per class has the highest diversity, and the training loss is steeper and converges much faster (at about epoch 50). Thus, the experts have the conclusion that datasets with high inter-sample entropy, i.e., diversity, do converge faster.

6.2. Information Between Input and Output

Users can select different input sizes in the training performance view. After the selection, the confusion matrix and line charts get updated. Fig. 5 shows a juxtaposition comparison of $H(x.label|y = i)$ with input size 1000 (Fig. 5(A)) and 10000 (Fig. 5(B)). For $H(x.label|y = i)$, when the input size is 10000, it gets stabilized faster (at about epoch 25) compared with input size 1000 which still dramatically goes ups and downs until epoch 70. This confirms that training with larger input data gets stabilized faster, and "stabilized" here means the uncertainty about the input class given output gets stable.

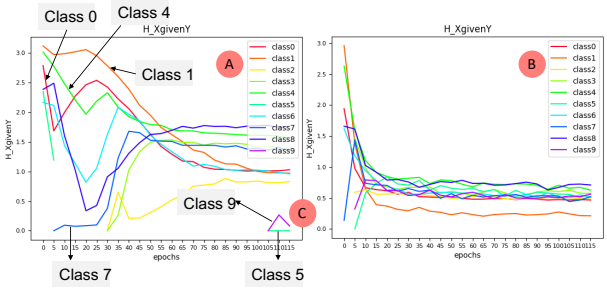

Figure 5: *(A) $H(x.label|y = i)$ with input size 1000, $i \in [0,9]$; (B) $H(x.label|y = i)$ with input size 10000, $i \in [0,9]$.*

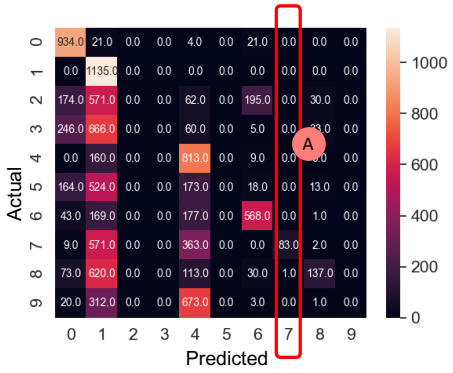

Figure 6: *Confusion matrix at training epoch 10.*

In Fig. 5(A), at the beginning, $H(x.label|y = 0)$, $H(x.label|y = 1)$ and $H(x.label|y = 4)$ are high indicating some instances belong to other classes get wrongly classified into class 0, 1 and 4. In Fig. 6, the brighter columns (column 0, 1 and 4) in confusion matrix verifies this. In Fig. 5(A), $H(x.label|y = 7)$ is low at the beginning. This means the model does not have much randomness during prediction for this class. The low entropy here is a reflection of a high precision score. As shown in Fig. 6(A), among all instances classified into class 7, most of them are belong to class 7. The segments in Fig. 5(C) shows $H(x.label|y = 5)$ and $H(x.label|y = 9)$ are infinity in earlier epochs. In Fig. 6, we can see column 5 and column 9 are all zeros (no instances get classified into these two classes).

The model has a hard time learning to predict class 9 and 5. This is the information we can not get from a single accuracy score.

From several case studies, we show that it is effective to use entropy as the evaluation metric to visualize the training performance. The visualization clearly tells us when the training gets stabilized. Besides, unlike the accuracy measure which only tells how many instances get correctly classified, entropy encodes information about wrong predictions such as how uncertain (random) the predictions are. Entropy ($H(x.label|y = i)$, $i \in [0,9]$) can help users locate when the model has high precision. However, low entropy does not indicate good performance since all samples can be wrongly classified into one class and still has low entropy, although the chance for this to happen is not high.

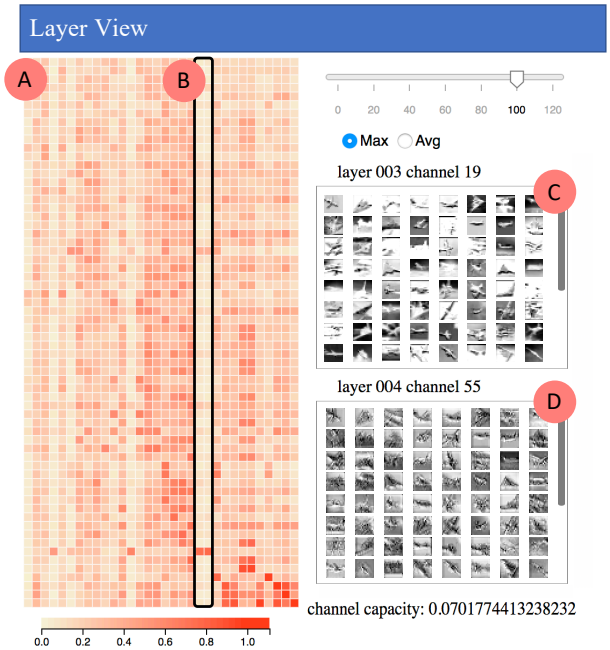

Figure 7: *Layer view. (A) Heatmap of mutual information between layer 3 (horizontal axis) and layer 4 (vertical axis); (B) two filters with similar patterns; (C) feature maps from filter 19 in layer 3; (D) feature maps from filter 55 in layer 4.*

6.3. Information Between Layers

The experts adopt the CNN classification model trained on the CIFAR-10 dataset for this case study. First, they choose layer 3 (horizontal axis) and layer 4 (vertical axis) in the query view. They select epoch 100 and sort the heatmap by the maximum value for each dimension. In Fig. 7(A), they observed two filters (filter 8 and filter 30 in layer 3) highlighted in the back rectangle in (B) have similar patterns. Their hypothesis is that these two filters are similar. To verify their hypothesis, they visualize the feature maps from these two filters and the other two randomly selected filters in layer 3 as shown in Fig. 8. The first row (A) in Fig. 8 are the input images. From the second row to the bottom row, they are feature maps from filter 0, filter 24, filter 8 and filter 30 respectively. We can see filter 8 and filter 30 (the last two rows (D) and (E)) perform similarly which verifies their hypothesis.

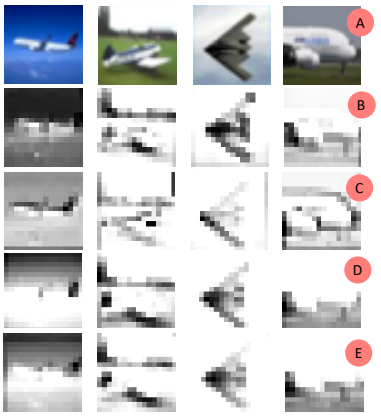

Figure 8: *(A) original input; (B) to (E) are feature maps at layer 3 from filter 0, filter 24, filter 8, filter 30 respectively.*

Experts are interested in the information flow between layer 2 and layer 3. There is a max-pooling between these two layers. When they drag the epoch slider, they can see the change of mutual information over the training time in Fig. 9. From left to right, they are heatmaps of mutual information at epoch 0, 40, and 120 respectively. Because experts are investigating the information change before and after a max-pooling, the corresponding filters are expected to be similar which makes the diagonal of these heatmaps have the highest mutual information. In the earlier training epochs, many rows and columns have similar patterns, indicating some filters are redundant. Through iterative training, there are fewer dark square patterns and similar rows and columns are decreasing, indicating filters in the convolutional layer before this activation are learning to extract useful features.

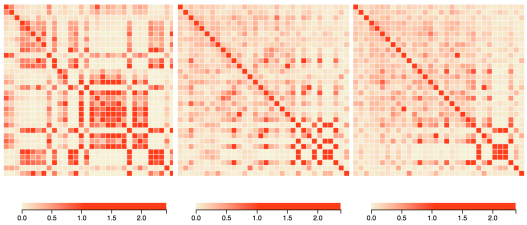

Figure 9: *From left to right, they are heatmaps of mutual information between layer 2 and layer 3 sorted by maximum value in each dimension at epoch 0, epoch 40 and epoch 120.*

To investigate the connection between formation flow and the model's depth, experts compare mutual information between layer 1 and latter layers (i.e., layer 3, layer 5, layer 7) at epoch 120. Intuitively, with each layer of the network learning to reduce the irrelevant information in input, the mutual information between the input and deeper layers will decrease. This explains the phenomenon that the mutual information heatmap between these layers are getting brighter as we approach deeper layers, as shown in Fig. 10.

6.4. Information Inside Channels

Experts use the CNN model trained on CIFAR-10 dataset to investigate the information inside channels. Fig. 11(A) is the small

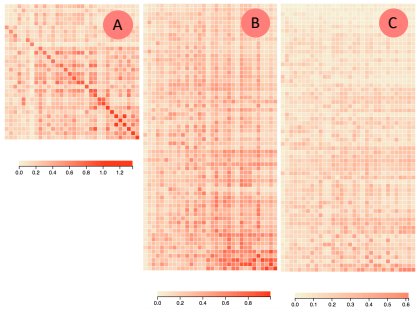

Figure 10: *(A) Mutual information between layer 1 and layer 3; (B)mutual information between layer 1 and layer 5; (C) mutual information between layer 1 and layer 7. All at epoch 120.*

multiples chart for layer 1 and Fig. 11(B) is for layer 11. By comparing these two plots, they conclude that filters in lower layers are more consistent among all input classes. However, filters of higher layers perform more differently and the behavior depends on the class of the input. This may be because normally filters in lower convolutional layers are trained to extract diverse and detailed features which are common for all classes. But in higher layers, filters will combine low-level features into class-related high-level representations. Besides, since low-level feature maps have more high frequency details (contains more irrelevant information), channels in lower layers have relatively higher entropies comparing to higher layers as shown in Fig. 11(C) and Fig. 11(D).

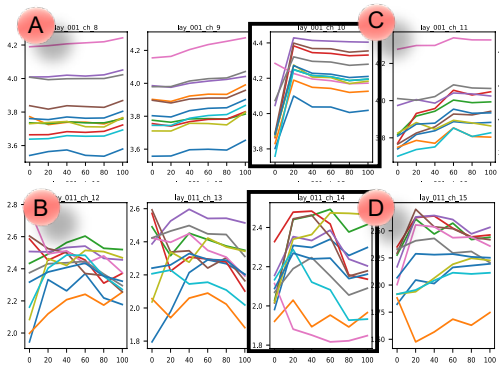

Figure 11: *Small multiples charts for layer 1 (A) and layer 11 (B).*

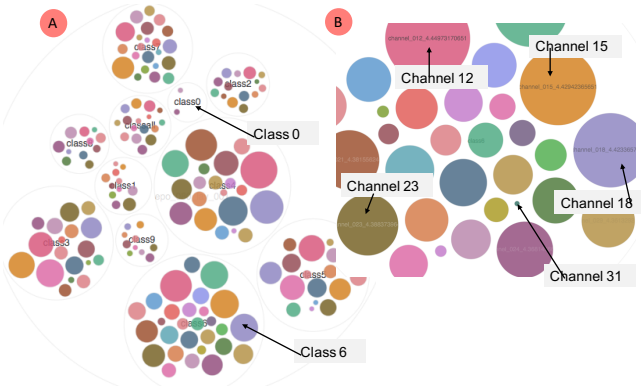

Figure 12: *Channel view. (A) circle-packing diagram for all channels in one epoch; (B) zoom-in of class 6.*

Experts are interested in the channel's performance differences when classifying different classes. Fig. 12(A) is the circle-packing diagram of layer 3 at epoch 80. Among these 11 big circles, 10 of them representing 10 different classes and the remaining one representing the whole dataset. In Fig. 12(A), class 0, which is the class "airplane", has the smallest intra-sample entropies. They also zoom in the circle of class 6 as shown in Fig. 12(B), by comparing the size of circles and hovering over the circles to see the intra-sample entropy values, they locate some channels with higher entropies such as channel 23, 18 and 12. They notice in Fig. 12(B), there is a small second-level circle in the lower left region. By hovering over it, they find it is channel 31 with entropy 1.795243. They find for every class, the intra-sample entropy of this channel is small. To gain insight into the function of this filter, they adopt deconvnet to visualize this filter's learned features. In class 6, they click on the small circle of channel 31. Fig. 13(A) is the deconvolutional result. They also visualize other filters such as filter 23 who has higher inter-sample entropy in Fig. 13(B). From Fig. 13 we can see the result of filter 31 contains fewer details compared to filter 23. This means filter 31 distills the input information too much. From the case studies, we demonstrate the hierarchy of circle-packing diagram gives users insight into the performance of the filter.

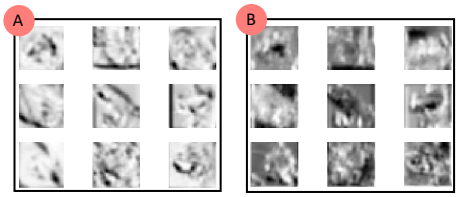

Figure 13: *Deconvolutional result for feature maps in channel 31 (A) and channel 23 (B) in layer 3.*

7. Conclusion and Future Work

In this work, we combine visualization with an information-theoretic approach to help evaluate and understand the information distillation process of CNNs. To begin with, we formalize a data model to store the information available in a CNN model. A four-dimensional hypercube is derived from the data model, consisting of four dimensions (i.e., input(X), layer(L), channel(C) and epoch (T)). By slicing on the hypercube, we are able to systematically formulate various information queries of a CNN. Based on the analysis need, we propose two types of entropies: inter-sample entropy and intra-sample entropy. We also develop a visual analysis system, CNNSlicer, from which users can explore the flow of information inside a CNN. We use the MNIST and CIFAR-10 datasets to demonstrate how to use CNNSlicer to evaluate and analyze the CNN model, such as comparing the convergence speed of training, and the information changes between layers.

In the future, we plan to extend our framework to analyzing other types of neural networks such as Recurrent Neural Networks (RNNs), Generative Adversarial Networks (GANs), and Attention-based transformers. We will also look into the roles of various neural network components using information theory, such as the activation functions, batch normalization and skip connection. We believe our information theoretical framework will provide a unique perspective to opening the black-box of deep learning neural networks.

References

[ADRS*19] ARRIETA A. B., DÍAZ-RODRÍGUEZ N., SER J. D., BENNETOT A., TABIK S., BARBADO A., GARCÍA S., GIL-LÓPEZ S., MOLINA D., BENJAMINS R., CHATILA R., HERRERA F.: Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai, 2019. arXiv:1910.10045. 2

[CE11] CORTEZ P., EMBRECHTS M. J.: Opening black box data mining models using sensitivity analysis. In *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)* (2011), pp. 341–348. 2

[FV17] FONG R., VEDALDI A.: Interpretable explanations of black boxes by meaningful perturbation. *CoRR abs/1704.03296* (2017). URL: http://arxiv.org/abs/1704.03296, arXiv:1704.03296. 2

[HXZ20] HUANG S., XU X., ZHENG L.: An information-theoretic approach to unsupervised feature selection for high-dimensional data. *IEEE Journal on Selected Areas in Information Theory* (2020), 1–1. 2

[KSZQ19] KHAN A., SOHAIL A., ZAHOORA U., QURESHI A. S.: A survey of the recent architectures of deep convolutional neural networks. *CoRR abs/1901.06032* (2019). URL: http://arxiv.org/abs/1901.06032, arXiv:1901.06032. 1

[LMM18] LELAND MCINNES J. H., MELVILLE J.: Umap: Uniform manifold approximation and projection for dimension reduction. *ArXiv e-prints* (2 2018). arXiv:1802.03426. 5

[LSL*17] LIU M., SHI J., LI Z., LI C., ZHU J., LIU S.: Towards better analysis of deep convolutional neural networks. *IEEE Transactions on Visualization and Computer Graphics 23*, 1 (Jan 2017), 91–100. doi:10.1109/TVCG.2016.2598831. 1, 2

[LXTG17] LI H., XU Z., TAYLOR G., GOLDSTEIN T.: Visualizing the loss landscape of neural nets. *CoRR abs/1712.09913* (2017). URL: http://arxiv.org/abs/1712.09913, arXiv:1712.09913. 2

[PGM*19] PASZKE A., GROSS S., MASSA F., LERER A., BRADBURY J., CHANAN G., KILLEEN T., LIN Z., GIMELSHEIN N., ANTIGA L., DESMAISON A., KOPF A., YANG E., DEVITO Z., RAISON M., TEJANI A., CHILAMKURTHY S., STEINER B., FANG L., BAI J., CHINTALA S.: Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, Wallach H., Larochelle H., Beygelzimer A., d'Alché-Buc F., Fox E., Garnett R., (Eds.). Curran Associates, Inc., 2019, pp. 8024–8035. 5

[SGPR18] STROBELT H., GEHRMANN S., PFISTER H., RUSH A. M.: Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE Transactions on Visualization and Computer Graphics 24*, 1 (2018), 667–676. 1

[Sha48] SHANNON C. E.: A mathematical theory of communication. *Bell System Technical Journal 27*, 3 (1948), 379–423. doi:https://doi.org/10.1002/j.1538-7305.1948.tb01338.x. 2, 3

[ST17] SHWARTZ-ZIV R., TISHBY N.: Opening the black box of deep neural networks via information. *CoRR abs/1703.00810* (2017). URL: http://arxiv.org/abs/1703.00810, arXiv:1703.00810. 1

[TZ15] TISHBY N., ZASLAVSKY N.: Deep learning and the information bottleneck principle. *CoRR abs/1503.02406* (2015). URL: http://arxiv.org/abs/1503.02406, arXiv:1503.02406. 2

[WGSY19] WANG J., GOU L., SHEN H., YANG H.: Dqnviz: A visual analytics approach to understand deep q-networks. *IEEE Transactions on Visualization and Computer Graphics 25*, 1 (2019), 288–298. 2

[WGYS18] WANG J., GOU L., YANG H., SHEN H.: Ganviz: A visual analytics approach to understand the adversarial game. *IEEE Transactions on Visualization and Computer Graphics 24*, 6 (2018), 1905–1917. 1

[ZF13] ZEILER M. D., FERGUS R.: Visualizing and understanding convolutional networks. *CoRR abs/1311.2901* (2013). URL: http://arxiv.org/abs/1311.2901, arXiv:1311.2901. 2, 8

[ZTF11] ZEILER M. D., TAYLOR G. W., FERGUS R.: Adaptive deconvolutional networks for mid and high level feature learning. In *2011 International Conference on Computer Vision* (2011), pp. 2018–2025. 7

[ZWM*19] ZHANG J., WANG Y., MOLINO P., LI L., EBERT D. S.: Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models. *IEEE Transactions on Visualization and Computer Graphics 25*, 1 (Jan 2019), 364–373. URL: http://dx.doi.org/10.1109/TVCG.2018.2864499, doi:10.1109/tvcg.2018.2864499. 2