Understanding and expediting learning in Convolution Neural Networks using feature maps

Anand, Mayank
my321532@dal.ca

Bhupathiraju, Akhilesh Varma
ak445438@dal.ca

Chandrala, Rohini
rh359742@dal.ca

April 2022

1 Abstract

Neural networks are heavily used in image processing and classification. However, understanding the internal workings of neural networks has always been a tricky task. One way to look at how a convolutional neural network (CNN) works is by visualizing the feature maps which are obtained after passing a filter through an image. To better understand these feature maps and their purpose in CNN, we experimented with multiple ways of utilizing these in accelerating and enhancing the training of CNNs. Firstly, we proposed a similarity-based classification in which, while training the network, the feature vector is computed using the feature map of the input image data and this is compared with the previously cached feature vector for images trained so far for each label after every batch of training. If a valid match, on comparison with the true label is found, the image will be assigned the default label, thus skipping the further training process of subsequent convolution layers, which boosts and increases the efficiency of the neural network. Secondly, to achieve improved classification, all the feature maps obtained from a fully trained CNN model were once again passed for training to the CNN with the same model architecture. This can add regularization effect to the model and helps when a model overfits the training data.

2 Introduction

Recently, deep learning has attracted researchers' attention for image classification tasks. Convoluted neural networks have immensely improved the performance of classification tasks. However, most of the composed models have a vast number of layers. For example, Google Net has 22 layers, and more layers count for more weights and biases. Which indirectly leads to more computation required to update these weights and biases.

To train these huge models, a large amount of computation is required, leading to significant computation time while training. The deployment of training these huge models on office machines is challenging. Therefore, we use GPUs to train these models to overcome these drawbacks. However, using GPUs constitutes more problems. A tremendous amount of power is used to keep these GPUs cool which is environmentally and financially expensive. Furthermore, it leads to some serious environmental concerns.

Our research focuses on filtering those images that might not be required for model training; rather, these images might increase training time. Also, similar images might not help much in changing weights. Moreover, if we do not feed these similar images, we are trying to figure out will this technique will decrease our training accuracy. If it does not affect much accuracy, then we can filter these images and should not send them for further training, which will help us accelerate the training. Matthew D. Zeiler and Rob Fergus [6] in their research mentioned how we could examine what happened to these images when they send over a CNN layer using feature maps. These feature maps can give us much information. Utilizing the power of feature maps, we tried to filter those similar images and wanted to experiment with how training affects if we exclude those similar images.

Secondly, these feature maps can be beneficial in analyzing results. However, we wanted to play

with these extracted feature maps and see what happens if we use these feature maps for training. We found out that these feature maps are nothing but blurred images for our original images on doing explanatory analysis. This intrigued us to see whether training a model with these feature maps will add regularization. Furthermore, we want to check how the model that is trained on feature maps react to an adversarial images. To find answers to these questions, we performed quite a few experiments that are briefly explained in further sections.

3 Exploratory Analysis

While doing exploratory analysis we were very curious to see what if some images are very different from others. For example, few people write 7 with bar in middle so, how our model will classify these outliers. Finding outliers for numerical data is very straight forward but for images we were looking for a though so that we could find these outliers, if exist. On doing some research, we got to point where thought we will never get to a conclusion if we think in the direction of images. So, we moved to the basic unit of an image which is pixel. As we are using MNIST dataset, on above analysis we found out that in our image we have total 784 pixels in 1D array or 28x28 shape in 2D array. And each pixel value is between 0-255. Having this idea, we then thought of generating an image that will have average pixel values of all pixels from images of that class known as centroid images.

Figure 1: *Showing centroid images for all class lables.*

As from Figure 1 we can see that these centroid images look almost like numbers that we usually correct. As we created these images with average pixel values these came out to be blurred and this is expected.

After finding these centroid images, we found Euclidean distance of all images of a class to the centroid image that we generated for that class. And we had an intuition that those images that will have more Euclidean distance will look totally different to our centroid images. And we will get our outliers. From our findings, we got the same results that we expected. The images that were more distant from centroid images looks much different from others. Below are the top 5 images of two classes that had greatest Euclidean distance from the centroid image of that class. From these findings, we can find out that on training our model if these images are classified correctly which will show that our model is not generalizing well and model is overfitting the training data.

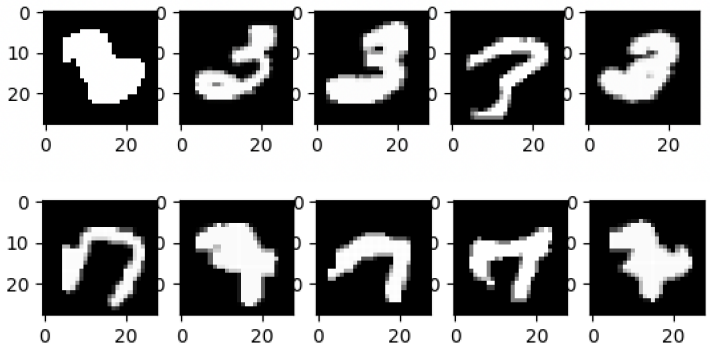

Figure 2: *Most different images for label 3 and 7*

3.1 Visualizing the data in higher dimension

Plotting t-sne [5] (t-distributed stochastic neighbour embedding) for MNIST would help us visualize the data in a higher dimension by giving each image a location in a 2D space. We see that all the similar-looking digits fall under the same cluster and the majority of the clusters seem to be having well-defined cluster boundaries. This indicates that the data can be classified easily without needing many transformations. Thus, simple neural networks would be sufficient to classify most images accurately.

Although both PCA and t-sne serve the same purpose of reducing the high dimensionality data into low dimensions, t-sne was used to visualize MNIST data as it gave better separation of classes. One of the reasons for this is how PCA and t-sne work. In PCA, the maximum variance along the n-dimension is captured and plotted in lower dimensions. It uses the Eigen decomposition of the covariance matrix. T-sne, on the other hand, takes

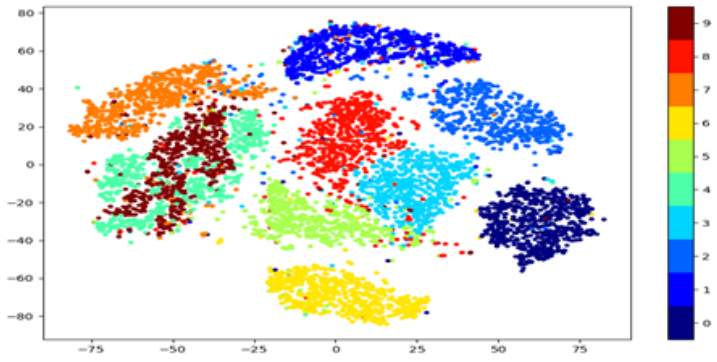

Figure 3: *t-nse plot for MNIST dataset.*

the high dimension data and gives each point in it a location in lower-dimensional space. It does so by finding clusters in the data, thereby ensuring that the embedding preserves the meaning in the data. T-sne helps keep the similar distance close and different distances apart, and it works by minimizing the distance between the points using t-distribution.

4 Related Work

Visualizing feature maps and gaining intuition about the network is standard practice. Matthew D. Zeiler and Rob Fergus [6], in their research, showed the feature maps that are generated from every layer during training. Similarly, Matthew Y.W. Teow [4], in his research, visualized feature maps for handwritten digits. These visualizations gave valuable insights into how a model learns while training, which intrigued our research. What if these feature maps are used while training our model. Keunyoung Park and Doo-Hyun Kim worked on how they can use feature maps to accelerate the testing or predictions of a model, and their research gave promising results that it can accelerate image classification. All these researches mainly focus on either visualizing feature maps or using feature maps for classification for the testing phase. In our research, we use these feature maps to find similar images. And we were then passing only images that were very different from the model for training.

5 Caching Technique

Park and Kim [2] did feature vector extraction in their research. We will be performing something similar to that. However, our research did cache process over the batches while training our model. This process mainly contains two steps feature vector extraction and cosine similarity check.

5.1 Feature Vector Extraction

Feature maps are obtained by convolving a trained kernel over an image. Moreover, images that classify in the same class have similar features because the intensity of each feature map extracted is similar.

As, Park and Kim [2] did in their research, we will be performing something similar to that in which for feature maps, a feature vector will be defined as $F_i(x, y)$ of size $P \times Q$ and Energy E can be defined $E_i \sum_x \sum_y |F_i(x, y)|$ where (i) is index of feature map for that convolution layer. Form here we will calculate mean $\mu_i = \frac{E(i)}{PQ}$ and standard deviation

$$\sigma_i = \frac{\sqrt{\sum_x \sum_y (|F_i(x,y)| - \mu_i)^2}}{PQ}$$

So, feature vector is mean and S.D of features for each convolution layer as:

$$\mathrm{V} = (\mu_0\sigma_0,\ \mu_1\sigma_1,\ \mu_2\sigma_2 \ldots\ldots\ldots \mu_{i-1}\sigma_{i-1})$$

The feature vector that we got is generated for a batch before and after training. The vector that is generated after the training of that batch is stored in the cache. Therefore, it is undesirable to write feature vectors of all batches because we use a cache. We have exactly one feature vector cache for each class label. A representative feature vector corresponding to each class must is generated. So, whenever a new feature vector V_{new} is cached in the same class, the mean and feature vector is V_{cached} is updated, as shown in the below equation.

$$V_{cached} = \frac{(V_{cached} \times n) + V_{new}}{n + b}$$

where, n = total images cached so far and, b = batch size

5.2 Cosine Similarity Check

The cosine similarity between feature vector of new batch $V_{query}(Q)$ with the cached feature

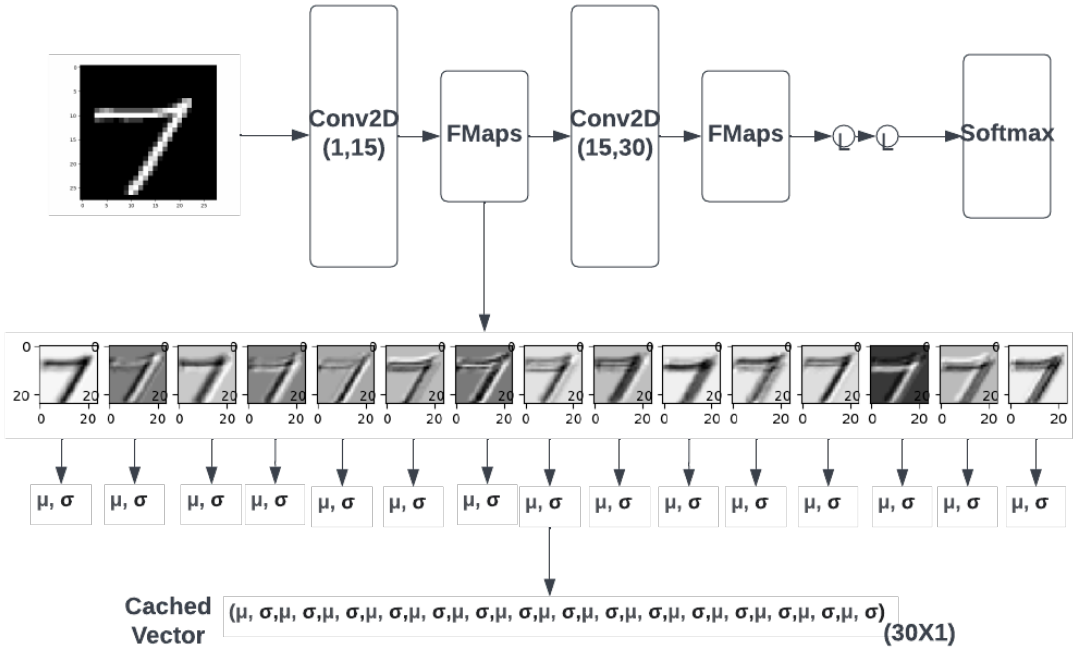

Figure 4: *Diagram shows how feature vector is calculated.*

vector $V_{cached}(C)$ so far in training is calculated. Below cosine similarity check is performed on both the vectors

$$\theta = \frac{\sum_i (Q_i \times C_i)}{\sqrt{\sum_i (Q_i)^2} \times \sqrt{\sum_i (C_i)^2}}$$

The value obtained from cosine similarity is between 0 to 1. The more closer the value is to 1 the more similar the feature vectors are.

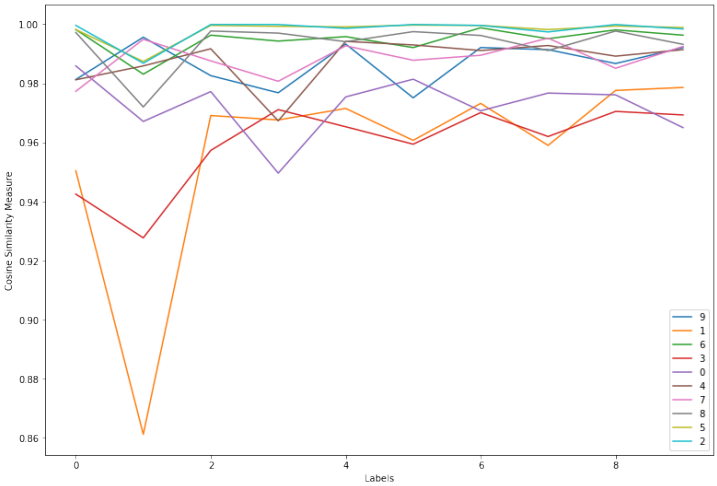

Figure 5: *Cosine Similarity Visualization at very early stage of training.*

However if keep on caching vectors in iteration loop at the end of training loop it will show mostly all *Cache Hit*. As in Figure 5 we can see the line plot for class 1 gives the least cosine similarity with the cached vector for that class. That shows it results in false positive. To overcome this we have include a dynamic threshold for each class which is updated every time when we update V_{cache} for each batch. So if value of that is less than threshold than we gave it as *Cache Miss*.

So Threshold (ε) is calculated as cosine similarity between $V_{new}(U)$ and $V_{cached}(C)$ when V_{cached} is calculated.

$$\varepsilon_{new} = \frac{\sum_i (U_i \times C_i)}{\sqrt{\sum_i (U_i)^2} \times \sqrt{\sum_i (C_i)^2}}$$

And the same way we update threshold on each new batch iteration as we update V_{cached}.

$$\varepsilon_{cached} = \frac{(\varepsilon_{cached} \times n) + \varepsilon_{new}}{n+b}$$

where, n = total images cached so far and, b = batch size

$$\begin{cases} Hit & if(\theta(V_{query}, V_{Cached} <= \varepsilon) \text{ and} \\ & \quad \text{argmax}_\theta\{\theta(V_{query}, V_{Cached})\} == truelabel \\ Miss & otherwise \end{cases}$$

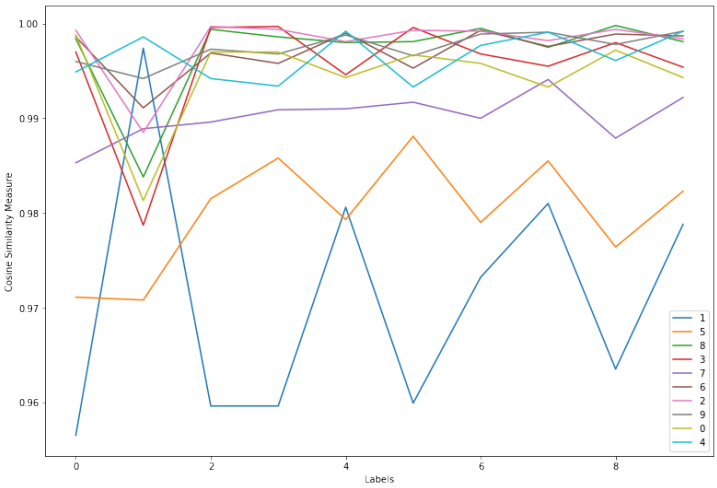

Figure 6: *Cosine Similarity Visualization at later stage of training*

In addition to the threshold, if a particular image matches the cache, we predict the label for that image. Moreover, if that label matches that image's true label, we remove that image from that batch. If the predicted label does not match the true label, we do not remove that image from that batch.

6 Experiments with feature maps

Below are the experiments we tried by training our model on feature maps. For these experiments we have used NUMPY [1] and Pytorch [3]

6.1 Training on feature maps

This experiment is mainly to see if training a model on the feature maps can act as a regularizer, thereby helping not to overfit the data. As a part of the experiments related to the training model on feature maps, we only used a simple architecture with two convolution layers and max pool layers after each of these convolution layers and an activation layer after the Max pool layer. We then included two linear layers with an activation layer between them and a softmax.

We have chosen this because we do not want to include any regularization in our architecture, and we would like to see if our approach of training the model on the feature maps itself acts as a regularization technique. So, in these experiments, we will first train a base model on a dataset(either MNIST or Fashion MNIST). We then extract feature maps for all the images and train the base model on the generated feature maps. In the end, we evaluate base model and base model trained on feature maps against the test dataset.

6.2 Testing base model trained on feature maps on perturbed images

In this experiment, we would like to verify that training a model on feature maps can help better perform on perturbed images. As a part of this experiment, we have first trained a base model(used in the feature map experiment) on the MNIST dataset.

We then extracted feature maps for the images. We also generated perturbed images using the same model. The base model is evaluated on the generated perturbed images, and the feature map model is trained on the feature maps. Once done, the feature map model is evaluated on the generated perturbed images.Perturb images are generated using Fast Gradient Signed Method(FGSM) and with an epsilon value of 0.25

7 Results and Analysis

7.1 Accelerated Learning Results

We used the caching technique to filter images and restrict them to send for further training in this experiment. We applied a cosine check of each feature vector of the image in batch with the cached feature vector in the training loop. Furthermore, if a successful Cache Hit occurred based on threshold and true values label comparison, we remove that image from the batch and send only different images for training.

As we can see from the below results for Batch 4096 in figure 7., accuracy, when the model is trained using the caching technique, it outperformed the standard training in which the model is trained on the whole dataset. Even after the total cache hit in each epoch was on an average

Epochs	Batch 4096			
	Normal Training	Cache Accelerated Training		
Epoch-1	accuracy=11	accuracy=25.3	successful cache hits =18133	total images trained =41867
Epoch-2	accuracy=12.7	accuracy=62.1	successful cache hits =19376	total images trained =40624
Epoch-3	accuracy=18.6	accuracy=72.6	successful cache hits =19675	total images trained =40325
Epoch-4	accuracy=34.3	accuracy=85.9	successful cache hits =20413	total images trained =39587
Epoch-5	accuracy=67.1	accuracy=90	successful cache hits =20711	total images trained =39289
Epoch-6	accuracy=82.5	accuracy=91.6	successful cache hits =20851	total images trained =39149
Epoch-7	accuracy=85.2	accuracy=92.9	successful cache hits =20947	total images trained =39053
Epoch-8	accuracy=85.6	accuracy=93.7	successful cache hits =21197	total images trained =38803
Epoch-9	accuracy=91.2	accuracy=94.2	successful cache hits =21247	total images trained =38753
Epoch-10	accuracy=91.8	accuracy=94.2	successful cache hits =21181	total images trained =38819

Figure 7: *Table showing accelerated learning in each epoch with Batch Size 4096.*

of 18,000 images, which shows with caching technique though the total number of trained examples decreased as we removed similar images; nevertheless, caching technique outperformed standard training.

We performed the same caching technique with different batch sizes as shown in Figure 8. and found out that with caching technique, we either get more accuracy or somewhere near to the standard training accuracy. This shows that the removed images did not make much of a difference to our training process. Moreover, in some cases, with less training data, we outperformed the standard training technique.

	Models		Batch- 32	Batch- 64	Batch-128	Batch-256	Batch-512	Batch-1024	Batch-2048	Batch-4096
MNIST	Train	Base Model	99.32	98.20	99.20	98.44	95.46	97.41	93.17	91.24
	Test		98.52	97.63	98.07	97.92	94.85	97.33	93.10	91.78
	Train	Base Model with Caching	**99.685**	94.71	97.59	96.01	90.55	94.57	**94.52**	**94.17**
	Test		**98.77**	93.71	96.49	95.30	90.42	94.04	**94.58**	**94.16**

Figure 8: *Table summarizing the results of training base model using caching technique on different batch size.*

7.2 Results of model trained on feature maps

This experiment showed that models usually overfit the dataset when trained multiple times(i.e. with more epochs). When we trained the feature map model, we noticed that base model trained on feature maps performed better on the testing set after training on the generated feature map images, for very minimal epochs. Results are summarized in Figure 9. However, the overall accuracy is decreased if the model is trained multiple times (i.e. with more epochs). This is because when the model got trained on feature maps with only edges and curves, the model tries to adjust the weights to recognize the edges and curves; hence, the decrease in overall accuracy is explainable. During the experiments on feature map training, we found that training the model on feature maps gives undefined values for loss and makes the model not learn anything. Investigating this further, we found that the extracted feature maps almost have negative pixel values, which is due to kernel values in the convolution layer. When we extract the feature maps after the convolution layer and pass them to the model again for training, the output of the loss function (Negative Log Likelihood) is resulting in Nans values. One reason for this could be, the feature maps obtained after the convolution layer may contain lot of negative values meaning the weights are negative. When there are a lot of weights which are negative and the output of convolution is passed to ReLU activation function, the negative values get converted to 0 resulting in lot of zero values. Now, since there are a lot of zero values, very few values get passed further inside the network and only few of the parameters get updated every time using gradient descent and this may cause gradients to become zero thus causing Nans' in the loss function output. To deal with this situation, the feature maps can be extracted after the activation layer coming immediately after the convolution layer or we can also use a different activation function like Leaky ReLU and this helps

MNIST Dataset:

Batch Size	Accuracy of Vanilla Model trained on original dataset (Train Accuracy)	Accuracy of Vanilla Model trained on original dataset (Test Accuracy)	Accuracy of Vanilla model trained on feature maps (Train Accuracy)	Accuracy of Vanilla model trained on feature maps (Test Accuracy)
64	98.98	98.13	94.24	94.34
256	96.45	96.34	90.57	90.75

Fashion MNIST Dataset:

Run S.No	Accuracy of Vanilla Model trained on original dataset (Train Accuracy)	Accuracy of Vanilla Model trained on original dataset (Test Accuracy)	Accuracy of Vanilla model trained on feature maps (Train Accuracy)	Accuracy of Vanilla model trained on feature maps (Test Accuracy)
1	72.89	71.6	72.18	72.7

Figure 9: *Table summarizing the results of training a model on feature maps*

because it helps in making the values positive.

7.3 Results of model trained on feature maps on Perturbed images

With this experiment, we observed that the model trained on feature maps did show better results on the perturbed images. Results are summarized in Figure 10. In perturbed images, we are ideally changing the inputs features(also we can say that the input values have shifted) and it ideally means it is a different distribution. So when there is a change in the input values(even though they are normalized values) the output of the hidden layers largely varies and these values will be on a different scale. However, the model trained on feature maps can recognize the curves and edges of the image even when there is a shift in the input values. For this reason, we think the model trained on feature maps is able to perform better than a normal model.

MNIST Dataset:

Epsilon	Accuracy of Vanilla Model on Perturbed Images	Accuracy of Vanilla model trained on feature maps on Perturbed Images
0.15	28.09	26.74
0.25	10.76	23.82

Figure 10: *Table summarizing the results of training a model on feature maps*

8 Conclusion and Future Work

To conclude, we have used the feature maps which are extracted from the initial layers of the convolution, on these feature maps we are finding out the cosine similarity to identify similar ones and passing the remaining to the training loop, this lead to expedite learning as the model is able to reach the accuracy quicker and also learn better than the vanilla model. In addition, when training is done on the feature maps itself keeping the weights of the trained vanilla model, the model is able to regularize much better and thereby increasing the training efficiency. To further, expand this idea the above model is also tested on perturbed images which resulted in increased accuracy as compared to the base model.

To concretize our results, we plan to expand this idea and perform experiments on a more complex dataset with coloured images like CIFAR 10 instead of grayscale images. Also, we would like to use a deeper convolution neural network for training, where we can try to extract feature maps from further convolution layers within the network and use them to match similarity measures and train the CNN based on these feature maps. One way to think about this is that since we have used only simple datasets like MNIST, FashionMNIST which are just grayscale and have less variety of images, the CNN model within the first layer itself would be able to extract the low-level features like edges and curves which are essentially what the numbers as labels consists.

However, when we use more complex data, the initial layers would similarly extract only the low-level features, whereas the deeper layers would be able to extract high-level features like shapes and regions, which are nothing but the combinations of these low-level features. Thus, using features maps from these deeper networks would make sense for such images. This idea can be further extended across domains, particularly in the music field, where the Mel spectrogram and Chromagram are generated for each audio clip, and a convolution neural network can be built in the same way. After training the network, the feature maps can be obtained from the final convolution layers, and a similarity caching technique can be applied to these feature vectors to help find similar music. So, now based on this similarity score, analogous music can be recommended to a user.

References

[1] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant.

Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[2] Keunyoung Park and Doo-Hyun Kim. Accelerating image classification using feature map similarity in convolutional neural networks. *Applied Sciences*, 9(1):108, 2019.

[3] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[4] Matthew YW Teow. Understanding convolutional neural networks using a minimal model for handwritten digit recognition. In *2017 IEEE 2nd International Conference on Automatic Control and Intelligent Systems (I2CACIS)*, pages 167–172. IEEE, 2017.

[5] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[6] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.