Comparison of Two Convolutional Neural Networks in Jet Image Classification

Anni Li,* Adolfo Partida,† Quinn Picard,‡ and Daniel Primosch§
University of California San Diego
(Group 4)
(Dated: March 24, 2023)

Abstract: In this PHYS 139/239 Final Project, we applied two variations of Convolutional Neural Networks (CNNs) to identify Higgs Bosons from the background (QCD jets). The first model is CVN (Convolutional Visual Network) [1], which is used to identify neutrino interactions based on data provided by NOvA. The second model is B-CNN (Bilinear CNN) [2], which is used to perform fine-grained classifications. We used the dataset of jet images [3], and the result shows that B-CNN performs better than CVN in this classification task, achieving 87.39% accuracy and 0.86 AUC for 2,500 test images.

I. INTRODUCTION

Particle physics is a highly complex and data-intensive field that involves the study of the fundamental building blocks of matter and their interactions. With the advent of advanced technologies and experimental techniques, particle accelerators can generate an enormous amount of data, which needs to be analyzed effectively to draw meaningful conclusions. Convolutional Neural Networks (CNNs) have emerged as a powerful tool for data analysis in particle physics due to their ability to extract features automatically and efficiently from large datasets. CNNs have been successfully applied in particle physics for tasks such as event classification, object recognition, and anomaly detection. In this paper, we will discuss the advantages of using different CNN models outlined in *Convolutional visual network for neutrino events* (Sutton and Barto 2016) [1], and *Bilinear cnn models for fine-grained visual recognition* (Lin, RoyChowdhury, and Maji 2015) [2], for data analysis in particle physics.

II. DATASET

The CVN network that we were originally trying to emulate was trained on a neutrino images dataset that we couldn't gain access to. We eventually settled on another image based dataset which aimed to separate Higgs Boson decay jets from background noise. The dataset, *Sample with jet, track and secondary vertex properties for Hbb tagging ML studies*, consists of Pythia 8 simulated proton-proton collision events at a center-of-mass energy of 13 TeV (Duarte 2019) [4]. We pulled 20,000 images from the dataset for training, but it had an imbalance of Higgs Boson and QCD data, we reduced the number of QCD images to match the number of Higgs Boson images which brought our sample size down to just over 5000. The data was processed into PyTorch tensors and the labels were converted from a one-hot format to a single binary vector for better compatibility with our neural network. Testing images were uploaded in a similar manor with a sample of 2500 images. Labels were truncated from the original six types to a binary system labeled if the image was a Higgs Boson or QCD event.

III. METHODS

CONVOLUTIONAL VISUAL NETWORK

We reproduced this network according to the paper A Convolutional Neural Network Neutrino Event Classifier by A.Aurisano et.al [1], but made some modifications because we used a different dataset. The original model uses a similar structure as GoogLeNet but has two different views of the image, X view and Y view. The model takes two parallel frames to pass these two views simultaneously and combines them by an inception module at the end followed by an average polling and softmax output. However, the original dataset used in the paper is not publicly available, so we chose the Jet Image Datase [4], which does not include two views of the image. Therefore, we only constructed one branch of this model.

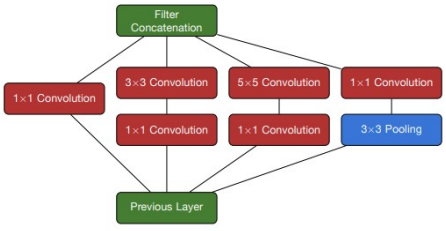


FIG. 1. Model of an inception module.Takes a set of feature maps produced by the previous layer as input then distributes those feature maps to branches, each with filters at different scales. The outputs from these branches are then concatenated to be passed to the next layer with same number of feature maps and dimensions as input .

* a5li@ucsd.edu
† apartidalizarraga@ucsd.edu
‡ qpicard@ucsd.edu
§ dprimosc@ucsd.edu

The model uses a NIN (Network-in-Network) structure, in which a main network contains several sub-networks. The sub-network, called inception modules, is shown in figure 1. Each branch of the inception module takes a set of feature maps produced by the previous layer as input, then distributes those feature maps to branches, each with filters at different scales [1]. The outputs from these branches are then concatenated to be passed to the next layer with the same number of feature maps and dimensions as the input. Using several inception modules allows for a diverse range of pattern learning without adding too much complexity, which increases the efficiency and capacity to learn.

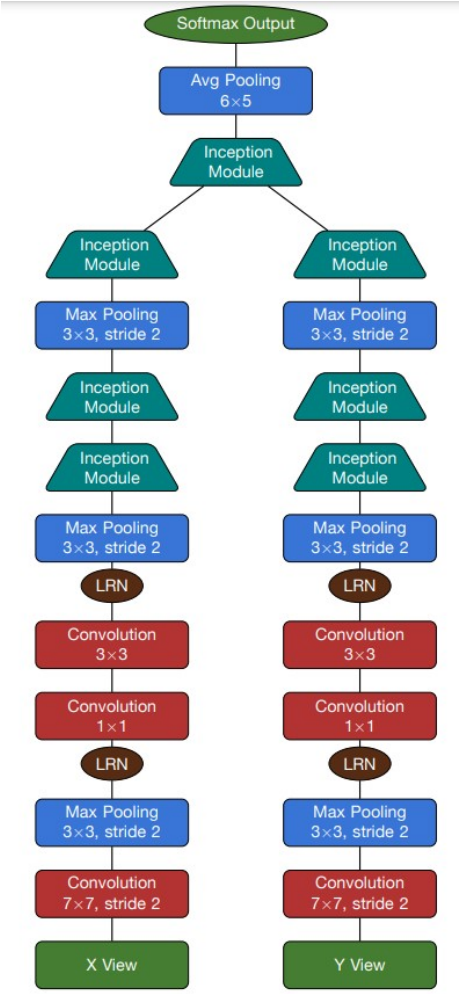


FIG. 2. Diagram of the CVN architecture used. Only a single branch was used in our model unlike the dual branch architecture used here.

The main network figure 2 starts with a 7x7 convolution layer, then passes to a 3x3 max pooling layer followed by an LRN (Local Response Normalization). The output then is passed to a 1x1 convolution layer, which down-samples the number of feature maps and maintains the dimensionality of the input maps. Then the output is passed through a 3x3 convolution layer, an LRN, and a 3x3 max pooling layer. Further, two inception modules are inserted to extract complex features. After a 3x3 max pooling layer and two inception modules, the output is processed by a 1x1 average pooling and a softmax for the final output. The original paper sets the final average pooling layer to 6x5 because they have three classes with two views. For our dataset, we only have one view and one label class, therefore we set the average pooling layer as 1x1.

The hyperparameters for this model are chosen with manual experiments. Since the model keeps predicting all zeros for every point, we think tuning hyperparameters cannot be much helpful. The final settings are learning rate=0.0005, batch size=32, loss function=NLLLoss, and optimizer=SGD with momentum=0.9. We trained the model 20 epochs and it takes less than 5 minutes in total. Since we noticed that data in the test set are not balanced, we added a threshold in the test function. We found that when the threshold < 0.5, the model would predict everything as 1, and when the threshold > 0.5, the model would predict everything as 0. We therefore determined to use another model to see if we can find a better solution to this classification task.

BILINEAR CNN

Since the predictions using the CVN are too sensitive to unbalanced data, we assume CVN's are not effective at distinguishing trivial differences between images. This reminds us of the fine-grained classification problem, which is to distinguish subordinate categories within a larger category, such as recognizing species of birds or models of cars. Since the input objects in this kind of problem are very similar in shape, models used for fine-grained classification would be experts at distinguishing trivial differences between classes. Unlike a traditional CNN, which uses a single feature extractor, B-CNN leverages two feature extractors to capture different aspects of the input images. The two feature extractors in the B-CNN model share their weights, allowing for more efficient learning of the image features. By combining these two feature extractors, the model can learn more expressive and discriminative representations of the input data. In our implementation, the VGG16 model with batch normalization is used as the backbone feature extractor. We set `pretrained=False` so that the VGG16 model is initialized with random weights to be trained from scratch on our own dataset.

The forward pass of the model starts by duplicating the single-channel input to create a 3-channel input, which is required by the VGG16 model. The input is then passed through both feature extractors. Each feature extractor generates a feature map, and these maps are pooled using global average pooling to obtain compact feature representations. Next, the model computes the outer product of these two feature representations using a bilinear layer. This bilinear operation is designed

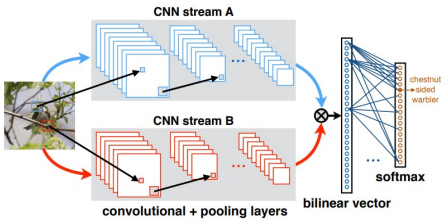
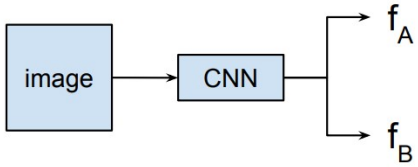


FIG. 3. The image is passed though CNN A and B and then their outputs are combined using a matrix outer product and average pooled to obtain the bilinear feature representation. This is then passed through a linear and softmax layer to get class predictions. [1] .

to capture higher-order interactions between the features learned by the two extractors. The output of the bilinear layer is used to make predictions for the specified number of classes. In the training loop, we added a validation process with early stopping to prevent the model from over-fitting too much. We set `patience=10` so that if the validation loss is higher than the best validation loss ten times, the training would stop. In practice, we notice that the model heavily overfits the data, so we set a weight decay=`1e-5` in our optimizer Adam with learning rate `0.0001`. Furthermore, we used cross entropy loss to calculate the training loss. The time it takes per epoch is much longer, which is approximately one minute per epoch. For possible improvements, we consider using optuna to automatically find the best hyperparameters. However, since it takes too long for the optimization process to finish, we did not apply this in our project. Furthermore, we can also modify the layers with different settings to make the model fit our jet dataset better and explore other loss functions for binary classification.

For details about the our code used, please visit our GitHub repository or go to the url `https://github.com/danprim/phys239_project.git`



(c) fully shared

FIG. 4. Model showing a B-CNN feature function where all computations are shared (VGG-M).

IV. RESULTS

The CVN model was trained using Stochastic Gradient Descent (SGD) with a learning rate of 0.0005 with momentum 0.9 and ran for only 20 epochs. The training loss was computed using Negative Log-Likelihood (NLLLoss). The training and testing data was loaded in batches of 32. However as mentioned before, the network predicted only ones for threshold > 0.5 or 0 if threshold < 0.5. This gave us a constant loss of 0.685 for all epochs and a AUC roc curve of around 0.55, meaning that the classifications were almost entirely random.

The B-CNN model was trained using the ADAM optimizer with a learning rate of 0.0001 and a decay of $1e^{-}5$. The loss was computing using CrossEntropyLoss. The data was once again fed in batches of 32. The model was trained for a maximum of 100 epochs but early stopping was implemented, determined by null improvements on validation loss, giving us an average of 22 epochs completed in total. The model gave us a final training loss of 0.27, with an accuracy of 87.39% and a AUC roc curve of 0.86.

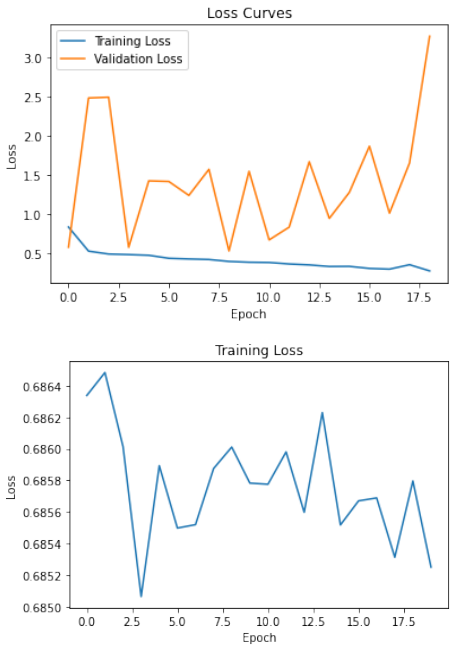


FIG. 5. Graphs comparing the loss from 17 epochs between the B-CNN (top) and the CVN (bottom) model.

As seen here the B-CNN showed a great improvement on classification when compared to the CVN model. The B-CNN model outperforms the CVN model in terms of loss, accuracy, and AUC. It also shows that the B-CNN model is much better suited for fine-grained classification problems.

V. CONCLUSION

In this project, we compare the performance of Convolutional Visual Networks (CVN) and Bilinear Convolutional Neural Networks (B-CNN) for classifying Higgs

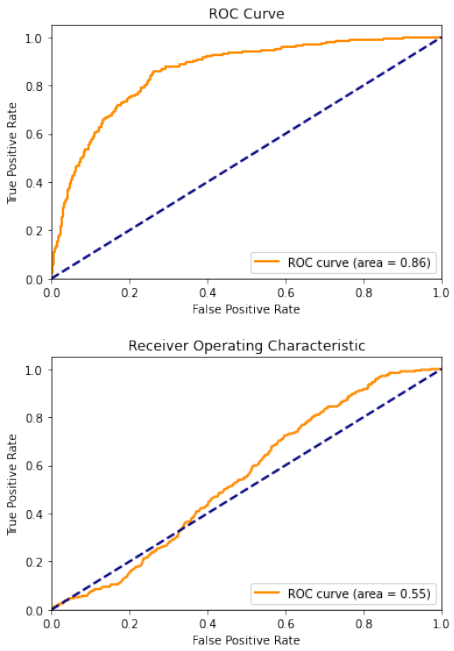


FIG. 6. Graphs comparing the ROC curves of the B-CNN (top) and the CVN (bottom) model.

Boson Events from background noise in jet images created in proton-proton simulated collisions. The CVN showed poor performance in the classification, classifying everything as a Higgs Boson Event or everything background noise. This returned an almost entirely random classification with an AUC of 0.55. Meanwhile the B-CNN proved to be much more accurate, achieving an AUC of 0.86, and correctly classifying the events with an accuracy of 87.39%.

We further conclude from this work that the performance of a neural network can depend sensitively on the type of images under consideration. Much care needs to be applied when translating a model that worked for one dataset to another, even if the general type of output, e.g. particle detector images, is the same. Our approach can be further optimized by applying the optuna library for possible improvements to training performance and improving other hyperparamter and/or loss function choices.

[1] R. S. Sutton and A. G. Barto, "Convolutional visual network for neutrino events," *arXiv preprint arXiv:1604.01444*, 2016.

[2] T.-Y. Lin, A. RoyChowdhury, and S. Maji, "Bilinear cnn models for fine-grained visual recognition," *arXiv preprint arXiv:1504.07889*, 2015.

[3] J. Duarte, "Sample with jet, track and secondary vertex properties for hbb tagging ml studies," *CERN Open Data Portal*, 2019.

[4] J. A. Maestre *et al.*, "Jet-images: Computer vision inspired techniques for jet tagging," *arXiv preprint arXiv:1511.05190*, 2015.