Identifying the Higgs Boson with Convolutional Neural Networks

Anton Apostolatos
Stanford Computer Science Department
antonaf@cs.stanford.edu

Leonard Bronner
Stanford Department of Statistics
lbronner@stanford.edu

Abstract

This paper proposes a Convolutional Neural Network architecture for tackling the problem of identifying the Higgs boson subatomic particle from colorflow energy images, as modeled by the ATLAS Experiment at the Large Hadron Collidor. Current methods which utilize Fisher Discriminant Analysis on jet pull superstructure data are used as a baseline for the algorithms we explored. We were able to substantially improve on the current state of the art, achieving an AUC score of 0.904.

1. Introduction

This project focuses on the identification of the Higgs boson subatomic particle from jet pull energy colorflow images of the particles' decay, as modeled by the ATLAS Experiment at the Large Hadron Collider at the European Center for Nuclear Research (CERN) [1].

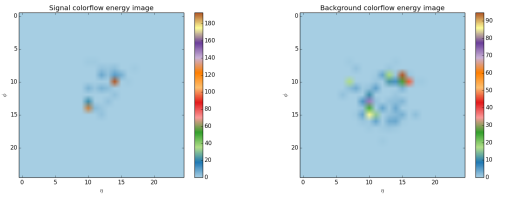
The Higgs field is an hypothesized energy field thought to permeate the entire universe. Without it, the Standard Model of particle physics would break down, as atomic particles would not have the required mass to attract each other, leading them to simply float around in the universe at the speed of light [20]. Its proof would completely alter our understanding of mass as a physical property, making the discovery of the Higgs field the fundamental unanswered question in particle physics in the last half-century [12].

The Standard Model suggests that if the Higgs field were to exist, then its quantum excitation, a particle referred to as the Higgs boson, would also have to exist [20].

The Large Hadron Collider (LHC) is tasked with finding this particle. Consisting of multiple superpowered electromagnets, the LHC collides charged particles traveling at near lightspeed. These collisions deform space upon impact, breaking the charged particles into their subatomic constituents. It has been hypothesized that with a proton collision at high enough energy, the Higgs boson would decay in observable ways.

The ATLAS detector at the LHC records 40 million proton collisions a second, making human curation of these events unfeasible [13]. An accurate classification system that would label the most promising observations as Higgs boson particle decays is, therefore, required.

In collaboration with the Stanford Linear Accelerator (SLAC) and the ATLAS Experiment at CERN we were given access to energy images for both Higgs boson and gluon decay, referred to as signal and background respectively, seen in Figure 1.



(a) Signal image (b) Background image

Figure 1: 25×25 pixel colorflow energy images

The purpose of this project is to build a Convolutional Neural Network (CNN) which, given the colorflow energy image of the decay of an unknown particle, can accurately distinguish whether or not that particle is a Higgs boson or not. We are building on the work from our CS229: Machine Learning class project, where we used conventional machine learning techniques to tackle this problem.

2. Related Work

2.1. FDA

Current approaches to Higgs boson identification involve the use of jet pull features, which is a class of features used to characterize the superstructure of a particle decay event [6]. Jet pull information provides insight as to whether an event was initiated by a quark or a gluon, or if it came from a single object's decay, as would be

[0]Special thanks to Ariel Schwartzman and Benjamin Nachman at SLAC and Michael Kagan at CERN for their guidance and advice.

the case for a Higgs boson particle decay, by describing the angle between energy decay patterns [17, 9]. The current state of the art model leverages this feature-set, while using Fisher Discriminant Analysis (FDA) for classification.

The approach extracts discriminating information between different classes of jets, similar to techniques used in computer vision [16]. The algorithm uses a representation of jets as images, applies preprocessing techniques to construct a consistent set of jet images, and applies a linear discriminant, which has been trained on a collection of example jets [4].

FDA identifies the plane in the high dimensional feature space which maximizes the separation between the jet classes, simultaneously minimizes the scatter within each jet class. Since FDA uses knowledge of the within-class variations, it is not significantly influenced by data fluctuations present in both classes [4, 16].

FDA is trained using a set of preprocessed example jet images from two classes and produces a discriminant that has the same dimensionality as the example jet images and thus can be viewed as a jet image itself. Discrimination between classes for a jet image is then achieved by projecting this image onto the Fisher-jet.

This method achieved an accuracy of 0.654 for Higgs boson classification from colorflow jet images. We will be comparing our results with the results achieved by this model.

2.2. Adaptive Boosting

In CS229, we set out to construct a binary classifier for the identification of Higgs boson decay events from colorflow jet images using a more traditional classification system. Our most successful classifier was an Adaptive Boosting, or AdaBoost, classifier.

AdaBoost is a meta-algorithm that combines multiple weak classifiers into a more accurate classifier. AdaBoost runs these weak classifiers multiple times, adapting each time so that subsequent classifiers are used to favor misclassified labels made by previous discriminants [5]. Algorithm 1 presents the method with more detail and rigor. For our particular case, this weak classifier $\mathcal{L}$ was a Random Forest classifier and the number of iterations T was 50.

The Random Forest algorithm is a general ensemble learning classification method, relying on decision tree models [8]. The Random Forest classifier works by constructing multiple classification trees, where leaves represent either $+1$ or -1 labels and branches represent conjunctions of features. The classifier builds B trees during training. When an unseen sample requires classification, the input vector is passed through every single tree constructed, where every specific tree T_b outputs a prediction [3]. The classifier returns the majority label.

Algorithm 1 AdaBoost algorithm [14, 15]

1: Initialize the distribution as

$$D_1(i) = 1/M, i = 1, \ldots, M$$

2: **for** $t = 1$ to T **do**

3: Get weak hypothesis $h_t : \mathcal{H} \mapsto -1, +1$ from training weak learner $\mathcal{L}$ using distribution D_t

4: Compute the error rate

$$\varepsilon_t = \sum_{i=1}^{N} D_{t-1}(i) \mathbb{1}(h_t(x_i) \neq y_i)$$

5: Compute the weight α_t as

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$$

6: Update the distribution, for $i = 1, \ldots, m$ as

$$D_t(i) = \frac{1}{Z_t} D_{t-1}(i) \exp(\alpha_t \mathbb{1}(y_i \neq h_t(x_i))$$

where

$$Z_t = \sum_{i=1}^{N} D_{t-1}(i) \exp(\alpha_t \mathbb{1}(y_i \neq h_t(x_i))$$

7: **end for**

8: Construct and return the final classifier

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

With this method we were able to achieve an AUC of 0.873, already a large improvement from the current state of the art. For our work this quarter, we are using this number as a benchmark for our CNN model results.

3. Dataset and Features

We were given access to energy images for both Higgs boson and gluon decay. The two dimensions of these images corresponded to the spherical coordinates called η and ϕ, where ϕ is the azimuthal angle in the x-y plane perpendicular to the beam direction and η is the angle in x-z. These were preprocessed to center the jet, with resonance being kept constant in every sampled data point. As is evident in Figure 1 there appears to be a stark difference between signal and background colorflow images, so automatic classification seems feasible. However, these images were generated from overlapping multidimensional probability distributions. Therefore, perfect classification is probabilistically impossible.

We had a dataset of 100,000 images composed of 625 floats (25×25 images), with half of the images belonging to each of the two classes. Our methodology consists of separating our dataset into a training set, a validation set, and a test set, and feeding randomized batches of the training set to our CNN.

We also zero-centered the images with the training mean and normalized all values to be between [-1, 1]. Furthermore, we zero-padded the images to 32×32 to allow for multiple even-divisions of Max-Pool, which would allow for larger network architectures.

4. Methods

We had initial doubts about the greater effectiveness of CNNs for this task compared to traditional machine learning techniques. This is because the added benefit of CNNs is the ability to pick up on multidimensional patterns in images. Our problem lacks both the depth (colorflow images just have 1 layer, instead of 3 RGB channels) and form that complex objects are made up of. Instead, as can be seen in Figure 1, our images look a lot noisier, with feature interaction limited to 1 dimension – thus, we thought it possible that normal machine learning classifiers have enough predictive power for this task.

Furthermore, this difference to common CNN tasks meant that we had to design, build and learn a CNN from scratch. As described previously, there are few similarities between these colorflow energy representations and common image datasets, making transfer learning less effective.

In the following sections we will describe the various models designed and built for this task, all of which were implemented on Tensorflow [2].

4.1. SimpleModel

Due to the limitations we foresaw with this project we decided to first build a simple CNN model to understand the potential effectiveness of CNNs for this task, and generate a baseline.

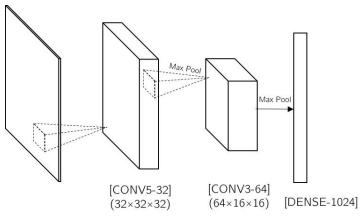


Figure 2: SimpleModel

As can be seen in Figure 2 our network consisted of a $32 \times 32 \times 32$ convolution layer, followed by a Max-Pool layer and a second $64 \times 64 \times 64$ convolution layer with another Max-Pool layer. We decided to increase the depth at each layer to counteract the down-sampling performed by the Max-Pooling layers. Furthermore, we set all our filters to be of size 3×3, because, as we discussed in class, enough 3×3 filters have the same power as one larger filter with the added benefit of fewer parameters. Finally, we decided on a stride of 1, given that our images were already very small and we wanted to loose the least amount of information possible.

A single filter convolution is a tensor w, in our case of size $3 \times 3 \times x$ where x was the depth of the incoming image, which slides along the input, at each step computing an inner product with the local area of the image. This single number is the local output at the new position, and all these local outputs are placed into a matrix at their respective positions to create w's output (biases are also added to this output). Multiple such filter outputs are stacked to create one convolution layer (in our case, for example 32 in the first layer, or 64 in the second layer). A toy example can be seen in Figure 3. In this way, our algorithm is able to to combine local information in different ways to better understand patterns between the pixels.

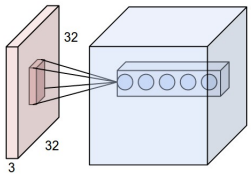


Figure 3: A convolution layer [11]

A Max-Pooling layer is another filter that moves across the new input (often the output from a convolution) and filters each pixel in the local area to be equal to the largest of all surrounding pixels. In this way the network is able to pick up on most important features at each set of convolutions. As before, we decided to minimize the amount of information lost, which meant using 2×2 Max-Pool filters with a stride of 2, which can be seen in Figure 4.

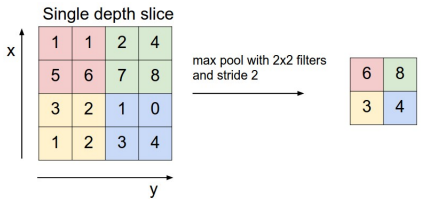


Figure 4: A Max-Pool layer [11]

After the convolution and Max-Pooling layers we place one 4092 dimensional fully connected layer with Softmax as the final layer to allow for a classification.

A fully connected layer can be seen in Figure 5, where the middle (hidden layer) is fully connected because every neuron receives input from every neuron in the input layer and passes on information to every layer in the output layer. We decided to use at least one fully connected layer because we wanted our final classifica-

tion decision to take account information from every part of the network.

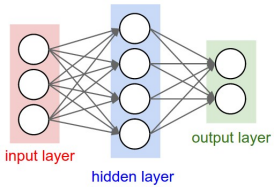


Figure 5: Fully Connected Layer (middle) [11]

As described above, our final layer was a Softmax, or in this binary case a simple logistic regression:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Softmax normalizes the input between $[0, 1]$, squashing the output of the neural network into a probability. The corresponding output can be interpreted as the confidence our algorithm has in classifying a given input.

During this project, a central mantra that we followed was: .

"Recklessness has no place in either love or weight initialization"

Following this advice, we used He et al.'s weight initialization as described in lecture [7].

4.2. LaNet

After verifying that CNNs were in fact competitive at this task, we moved on to create a larger network. We decided to model our network on the VGG architecture [18]. Given the constraint of having to train networks from scratch, this network architecture seemed the most reasonable, both in terms of training time and in terms of number of variables.

A second issue we faced when designing a CNN architecture was the relative small size of our input images. This limited the amount of down-sampling we could perform. Since our images were of padded size 32×32, we were only able to run them through at most 5 Max-Pool layers.

The second network that we built, called L(eonard)-A(nton) Net, as presented in Figure 6, is substantially larger than the SimpleModel we had built for verification.

It has two $32 \times 32 \times 32$ convolution layers, followed by a Max-Pool. Then two $64 \times 16 \times 16$ convolution layers and another Max-Pool. These are followed by three $128 \times 8 \times 8$ convolution layers with a Max-Pool layer and then another three $256 \times 4 \times 4$ convolution layers with a final Max-Pool layer. We then had two 4096-dimensional dense layers with a final 1000-dimensional dense layer with a Softmax classifier on top.

4.3. LaNetTwo

The final complete model we built was an extension of LaNet. As discussed in the following section, we built this network to see if a larger network would be any more successful at tackling this problem, or if we had reached the limit of this task.

As can presented in Figure 7 our LaNetTwo is very similiar in implementation to LaNet – it simply took the design of our first large network to its logical conclusion. We added one more convolutional layer to both the first and the second set of convolutional layers to make three $32 \times 32 \times 32$ and three $64 \times 16 \times 16$ layers with a Max-Pool to separate them. We also added a final set of convolutional layers at the end of our network before the densely connected layers. This set of layers were now of size $512 \times 2 \times 2$, which made the images as small as we thought would make logical sense in a convolution. At the end of that layer, we added a Max-Pool that led into the dense section of the network, where the features were of size $512 \times 1 \times 1$, before being expanded again to be of size 4096 before Softmax classification.

5. Experiments and Results

5.1. Training and Testing Methods

Since we had large datasets at our disposal we used hold-out cross validation. Namely, we split our dataset into three sets, a training set which composed of 65% of the data, a validation set which comprised 17.5% of the data and a test set, which was made up of the remaining 17.5% of the data. We would train each model on the training set and would evaluate the current hypothesis function of the model after every epoch on the validation set in order to tune hyperparameters on the go (this was necessary because our model took a very long time to run, which meant that it was unfeasible to allow every model to run until completion). After our CNN had finished learning, we evaluated our model only once on an previously unseen test set.

5.2. Evaluation Metrics

Classifiers will be evaluated by their receiver operating characteristics, or ROC curves. We will also be quantifying the performance of all binary classifiers tested by calculating the area under the ROC curve, or AUC. We use this method because the AUC represents the probability that a randomly chosen signal example will be classified correctly, which is exactly what we wish to optimize towards. It was also the metric which was used in all previous literature for Higgs boson classification problems, and so in order to compare our classifiers against the state of the art, this metric was important. As explained above, the AUC is the area under

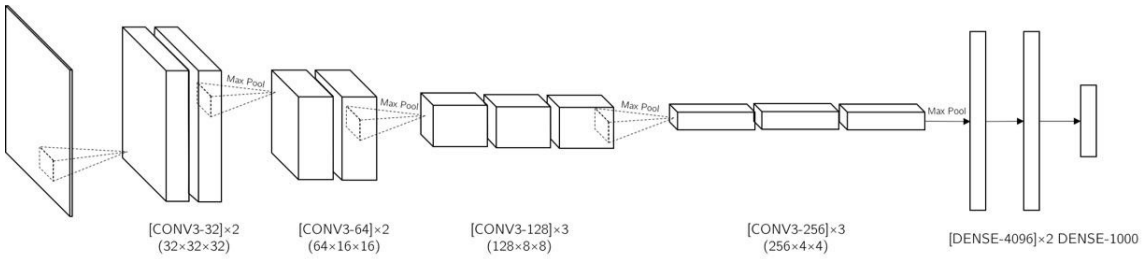


Figure 6: LaNet

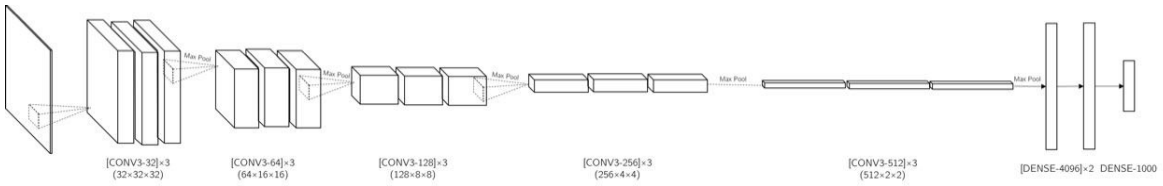


Figure 7: LaNetTwo

the ROC curve, which shows a plot of the true positive rate (TPR) versus the false positive rate (FPR). A perfect classification would mean that the TPR would be 1.0 at every value of the FPR, which would then give us an area under the curve of 1.0. On the other hand, if our TPR always exactly the same as the FPR, our model would not be any better than randomly guessing, giving us an AUC of 0.5 (any AUC less than 0.5 would simply mean we would have to flip the classifier, so nothing is worse than FPR $=$ TPR).

5.3. Experiments

5.3.1 Overfitting the Simple Model

The first experiment we performed was making sure that our CNN was able to learn from the data. As a sanity check, we took a small sample from the training set and ran our SimpleModel until we had clearly overfit the data. This meant a training accuracy of 1.0 for all batches. Within a couple dozen epochs this was achieved. We continued to use this as an initial sanity check for each of our models.

5.3.2 Learning Rate

As we have discovered, the learning rate is by far the most important hyperparameter which can make or break an artificial learning experience. We therefore spent most of our computation time on fine-tuning this hyperparameter. Because of the time it took to train a single model, we were unable to perform random search on our learning rate. Instead, we took the approach of increasing the learning rate by an order of magnitude until our algorithm diverged, and then halved the learning rate until we saw a model that converged.

As we can see in Figure 8 in the "No Dropout" line, we saw validation accuracy jump up very early in the learning process, but then peter out quickly. Our takeaway from this was the necessity of annealing the learning rate.

We tried two different methods for this. One was an exponential decay of the learning rate, where in each subsequent epoch we multiplied the learning rate by a factor γ where $0 < \gamma \leq 1.0$. We tested multiple values of γ and found no evidence that it improved learning or convergence.

For this reason, we attempted a step decay approach to learning rate annealing, similar to that of Simonyan and Tisserman [18]. If validation accuracy did not decrease sufficiently over multiple epochs we would halve the learning rate going forward.

Interestingly, neither of these approaches had a significant positive impact on our model's convergence or classification power. We assume this means that our models have therefore reached a local optima, described by the hyperparameters we used, without decay.

5.3.3 Dropout

Our model had significant over-fitting over the training set. We therefore introduced a dropout on the densely connected layers as proposed by Srivastava et al. [19]. In essence, during training dropout turns off connections between neurons with a probability p. This is equivalent to sampling different neural networks from a larger, completely dense neural network, not unlike traditional ensemble methods such as AdaBoost. The problem with this approach is that dropout is only performed on the dense layers, allowing for overfitting in the convolution

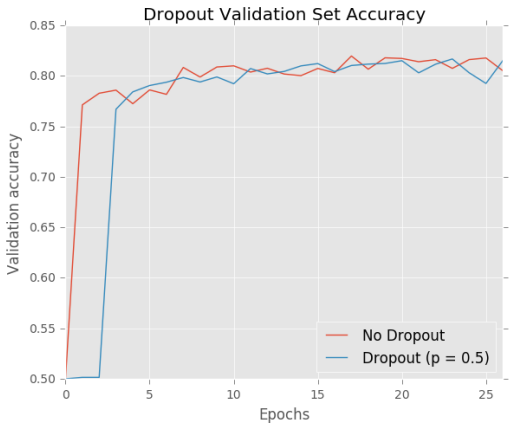


Figure 8: Validation set accuracy with and without dropout

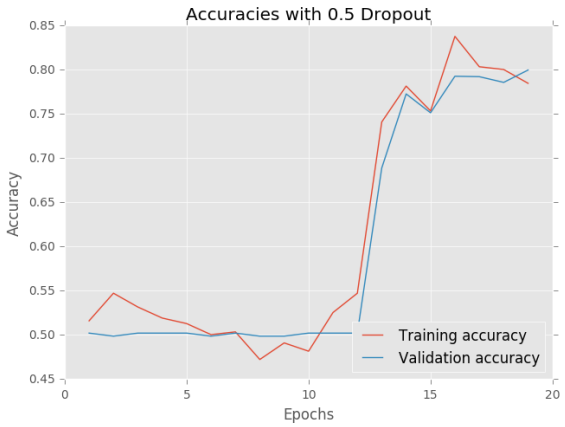


Figure 9: Validation and train set accuracies with 0.5 dropout

layers. The hope, however, is that backpropagation from the dropped out dense neurons will impact the training of the convolutional filters in a desired manner.

Training with dropout took a significantly longer time. Either convergence took longer, but what we also often saw was a accuracy cliff where the validation accuracy would remain at around 0.50 for multiple epochs before convergence set in. This can be seen in Figure 8 This is similar to what has been seen when it comes to bad weight initialization which may show an underlying connection between dropout and initialization, something that we found interesting and may warrant further investigation.

During our experimentation we found that while dropout did help combat overfitting the training set, as is evident in Figure 9, we did not see any significant improvements on the power of classification of our model on the validation set. Admittedly, this is something we have a hard time explaining and may warrant further testing. However, the simple explanation would again be that our model had reached it classification potential even without dropout.

Ultimately our final model included dropout of $p = 0.5$.

5.3.4 Batch Normalization

Ioffe and Szegedy introduced into the literature a batch normalization technique which should decrease the necessity of perfect weight initialization [10]. The general idea of batch normalization is to subtract the batch mean and divide by the batch standard deviation before every layer to normalize the inputs to convolutions. The intuition for that is to zero-center and standardize the inputs which stops the weights from being vastly affected by the data they see.

We applied this technique to our architectures, adding a batch normalization layer before every convolution set. Surprisingly enough, this had no impact on our validation accuracy, except for making training significantly slower. Our intuition for this is two-fold. On the one hand, we think that our weight initialization from He et al. was good enough. Concurrently, our experimentation on the learning rate reduced the potential gains provided by batch normalization.

For these reasons we did not include batch normalization in our final model.

5.3.5 Larger Networks

Our move from LaNet to LaNetTwo was an attempt to exhaust the number of layers that our input size allowed us to have, and see whether this would improve our results. As can be seen in the following section, our best LaNet model achieved an AUC score of 0.895 while our best LaNetTwo model achieved an AUC score of 0.896 – not the improvement we expected nor were looking for.

As a result we decided to try a different approach. Instead of increasing the amount of layers, we wanted to extend the power of the LaNet setup. For this reason we doubled the number of filters at every single convolution layer, leading to the conception of what will affectionately henceforth be referred to as LaNetThree. For example, in the first set of convolution layers, LaNetThree has 64 filters instead of LaNet's 32. We have not included a graphical depiction of this model in the paper as it is identical to LaNet in every single other way.

We found that this was a better extension of the classification power of CNNs, with LaNetThree achieving a AUC of 0.904 on the test set.

5.4. Results

There are six models which we ultimately want to compare. These include the FDA on pull features approach, AdaBoost, SimpleModel, LaNet, LaNetTwo, and LaNetThree. For the sake of brevity, we have only included the results of the best models for each of these. The results are found in Figure 10.

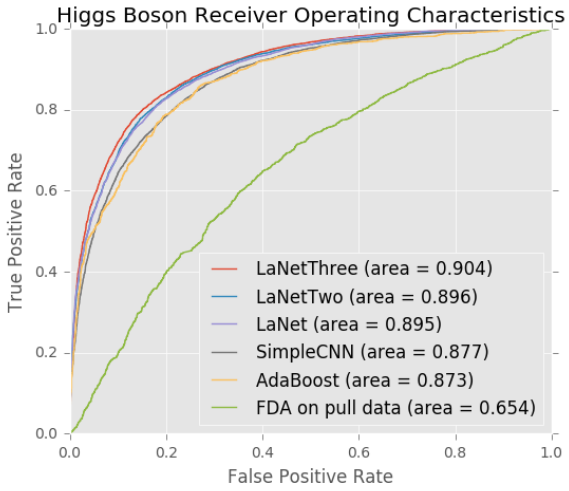


Figure 10: Comparing five main models

As this ROC graph shows us, a Convolutional Neural Network is easily able to achieve a high score on this task. Comparing just our SimpleModel against the AdaBoost classifier, we can see that they both achieved a comparable AUC of around 0.87 (0.873 for AdaBoost vs. 0.877 for SimpleModel). Given our images are preprocessed to be as standardized as possible, a traditional classification model that doesn't leverage the image-like component of the data can still be competitive. It seems that an AUC of 0.87 is the transition point between these two different types of learning. It took us a considerable amount of time, effort and resources to achieve anything beyond that.

Our more complex models, starting with LaNet, were able to achieve a greater AUC than the SimpleModel or than AdaBoost. Namely, we achieved an AUC of 0.895.

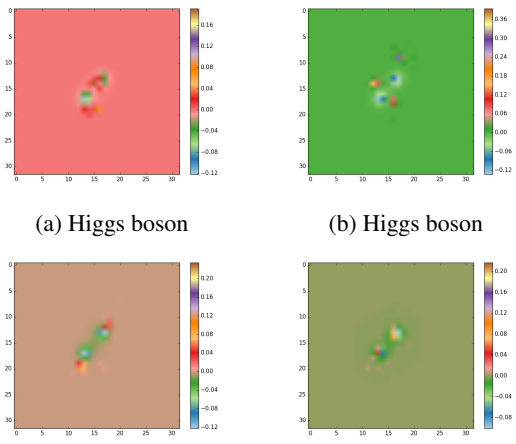
As explained previously, we extended LaNet in two different ways. LaNetTwo, representing the deepest possible network for our data, achieved a marginally better AUC of 0.896. An intuitive explanation for why a much deeper model was not significantly more effective is that there aren't as many layers of object complexity in our colorflow jet images as in traditional object recognition, meaning that each layer has less necessity to abstract away complicated attributes. A classifier for cars, for example, needs to look at many details at many layers, from the edges of the windscreen to the lighting of the car, to be able to work well. We believe that our objects do not have such complicated or varying features as to necessitate as deep of a network.

On the other hand, doubling the number of filters for every layer had a more considerable effect on classification power, with LaNetThree achieving an AUC 0.904. This may point to the idea that there is more interaction effect between pixels in our images than we had previously thought when doubting the potential of CNNs for this task.

These results are also observed when measuring accuracy:

Model	AUC	Accuracy
FDA	0.654	-
AdaBoost	0.873	0.798
SimpleModel	0.877	0.774
LaNet	0.895	0.812
LaNetTwo	0.896	**0.825**
LaNetThree	**0.904**	0.820

It's difficult to look at misclassified colorflow images and understand what the reasons were for the CNN to classify them incorrectly without having a much deeper understanding of the physics governing these phenomena. Notwithstanding this, we know that the Higgs boson decays into very specific decay channels, of which there are always two particles present. In Figure 11 we can see examples of these incorrect misclassifications from LaNet. Notice that these examples all generally present this general structure of having two distinguishable maxima. It is this ambiguity that challenges the classification power of our model.



(a) Higgs boson (b) Higgs boson

(c) Background (d) Background

Figure 11: Misclassified images with their correct nature

6. Conclusion

While our results, especially compared to non-deep convolutional learning approaches, do not seem like great progress, it is important to stress that this work in general represents a breakthrough in the Higgs boson identification process through jet colorflow images. We were able to vastly increase the probability of correctly identifying the presence of a Higgs boson subatomic particle. To place our results in context, our final CNN model achieved a 24.4% proportional increase from AdaBoost to a perfect oracle and a 72.3% proportional increase from the state of the art FDA. As discussed previously, perfect classification is probabilistically impossible given the multidimensional probability distributions that these data are sampled from. In light of this, our results are very good.

7. Evaluation & Future Work

While our collaborators at SLAC and CERN think that these results are already publishable, there are a few avenues that we want to explore in the future, especially in connection with utilizing CNNs for this task. Primarily, we would like to run our models on more diverse data. Currently, the dataset has been sampled from the same event and is preprocessed (e.g. shifted, rotated, normalized) to make the classification task as easy as possible. We assume this is why non-CNN approaches are able to rival neural networks for this task. This would also mean that the scores that we are seeing above are as good of a classification as we can expect – especially considering that larger networks were not able to vastly improve our scores from LaNet.

Once our data is less standardized we expect non-CNN classifiers to perform substantially worse than CNNs, as neural networks should be able to pick up on rotations and translations. Furthermore, this non, or at least, less processed data will have more complex relationships between the pixels in the colourflow images – something that a CNN should be able to detect.

All in all, we very much are looking forward to continue working on this problem and hope to be able to publish our results and contribute to work at CERN at identifying the Higgs boson.

References

[1] G. Aad, E. Abat, J. Abdallah, A. Abdelalim, A. Abdesselam, O. Abdinov, B. Abi, M. Abolins, H. Abramowicz, E. Acerbi, et al. The atlas experiment at the cern large hadron collider. *Journal of Instrumentation*, 3(08):S08003, 2008. 1

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 3

[3] L. Breiman and A. Cutler. Random forests-classification description. *Department of Statistics, Berkeley*, 2007. 2

[4] J. Cogan, M. Kagan, E. Strauss, and A. Schwartzman. Jet-images: computer vision inspired techniques for jet tagging. *Journal of High Energy Physics*, 2015(2):1–16, 2015. 2

[5] Y. Freund, R. E. Schapire, et al. Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156, 1996. 2

[6] J. Gallicchio and M. D. Schwartz. Seeing in color: jet superstructure. *Physical review letters*, 105(2):022001, 2010. 1

[7] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015. 4

[8] T. K. Ho. The random subspace method for constructing decision forests. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(8):832–844, 1998. 2

[9] A. Hook, M. Jankowiak, and J. G. Wacker. Jet dipolarity: top tagging with color flow. *Journal of High Energy Physics*, 2012(4):1–15, 2012. 2

[10] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. 6

[11] F.-F. Li and A. Karpathy. Cs231n: Convolutional neural networks for visual recognition, 2015. 3, 4

[12] J. Lucio et al. Proceedings of the ii mexican school of particles and fields. Technical report, Teaneck, NJ; World Scientific Pub. Co., 1987. 1

[13] L. Mackey and A. Schwartzman. Physics event reconstruction at the large hadron collider. *Stanford Data Science Workshop*, 2015. 1

[14] J. Matas and J. Sochman. Adaboost. *Center for Machine Perception, Czech Technical University, Prague*, 2001. 2

[15] R. E. Schapire. Explaining adaboost. In *Empirical inference*, pages 37–52. Springer, 2013. 2

[16] B. Scholkopft and K.-R. Mullert. Fisher discriminant analysis with kernels. *Neural networks for signal processing IX*, 1:1, 1999. 2

[17] J. Shelton. Tasi lectures on jet substructure. *arXiv preprint arXiv:1302.0260*, 2013. 2

[18] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 4, 5

[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014. 5

[20] M. SStrassler. The known particles - if the higgs field were zero. October 2011. 1