ENPH 455: Classifying Particle Interactions with LiquidO using Deep Learning

Kyle Singer

Supervisor: Mark Chen

Department of Physics, Engineering Physics and Astronomy, Queen's University, Kingston, Ontario, Canada, K7L 3N6

April 3, 2022

Abstract

Liquid scinitillator detectors (LSDs) often use transparent scinitllators to detect neutrino collisions. While they are proven effective, one of the drawbacks of these detectors is their inability to distinguish event-by-event interactions, especially at energies below 10 MeV. LiquidO is a newly proposed form of LSD where the scintillator used is opaque (highly scattering). With this new form of LSD, photon patterns are produced via fibre optic cables running through the liquid and transporting photons created during collisions to the end of the fibres. These images can be classified, though too many are produced for people to classify them one by one. Therefore, a deep learning approach was considered due to the minimal limits placed on memory and training time, as well as the essentially infinite amount of images available since they are simulated. Thus, a convolutional neural network (CNN) was designed with an Adam optimizer and cross-entropy cost function, and trained on various datasets. The datasets contain images of positrons which come from electron flavoured anti-neutrino collisions, and electrons and gamma rays, which come from backgrounds (3 classes total). The goal was to have the CNN classify the simulated images with a 99% accuracy rate meaning no more than 1% of the testing image set would be misclassified. Some of the challenges with this included a shared GPU with limits on how much memory could be used by a single user at a given time and general maintainability considerations. The top performing trained model was able to reach 99.8% accuracy though fell to 75% when tested on images shifted from starting at the (0,0) graph point. After being trained on the shifted images combined with the 99.8% model's parameters being used as an initial starting point for the new model, the results showed a more reliable 99.29% and 99.33% accuracy when tested on shifted and non-shifted image sets respectively.

Acknowledgements

Reflecting back on my work, I would like to thank my supervisor Mark Chen for his role in helping to guide and support me through this project.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

In 1956, the first ever electron flavoured anti-neutrino was observed using a (hydrocarbon) liquid scintillator detector (LSD) [1]. Since then, various forms of these types of neutrino detectors have been created all differing in size, type of doping, and overall structure. Some very successful ones include SNO [2], Borexino [3], and NOvA [4] which all use transparent liquids as their scintillator. LSDs work by the liquid being excited to the state of luminescence where the number of photons emitted is correlated to the energy of the particle colliding with it [5]. One example of this which will be focused on in this paper is inverse-β decay, $\bar{\nu_e} + p \rightarrow e^+ + n^0$ where the positron's decay produces an identifiable photon pattern [6]. When the positron breaks down, it creates two gamma rays making it unique and distinguishable.

Transparent LSDs work by placing light emission detectors, typically photo-multiplier tubes (PMTs) around the outside of the system to detect the photons produced during collisions with the liquid. The produced photons propagate towards the PMTs where, based on the number of photons and time before detection, the particle type and position of interaction can be recognized. One of the major drawbacks of LSDs are their ability to distinguish event-by-event interactions, especially when the particle collision occurs at energies lower than 10 MeV [6]. Current transparent LSDs can use segmentation (splitting the volume) to try to recover information about neutrino interactions. Segmentation works by dividing the volume into a series of cells where the light produced during an interaction is contained within the given cell. The resulting pattern created when different cells produce light gives a form of image/pattern that is used to identify the class of particle, such as positrons from neutrinos or electrons and gamma rays from backgrounds (BGs). Despite the positive effects of segmentation, this setup tends to fail at lower energies (less than 10 MeV) as there is less energy deposition [6].

In order to better distinguish event-by-event interactions, a new form of detector was proposed called LiquidO where the LSD uses an "opaque" (highly scattering) scintillator as opposed to the usual transparent. To detect photons, optical fibres run through the liquid instead of around the outside as the light would not be detected. The resulting photons created near the emission point are transported to the ends of the fibres where the fibre-hit pattern creates a photon image pattern. The more photons that connect with an optical fibre, the brighter the point in the overall image. This method of neutrino detection does not require segmentation as the opaque liquid performs its own light confinement, producing better spatial imaging as the photons can be better contained. The resulting images show unique patterns for both BGs and neutrinos as seen in Fig. 1. Electrons tend to make a single point energy deposition while gamma rays Compton scatter to produce multiple points. Positrons make a bright central point with two annihilation gamma rays that are emitted back-to-back (180°, in a line).

The number of interactions and resulting images produced from LiquidO become too many for a person to classify as they are on the scale of thousands to millions. Thus an automated method is needed to distinguish between neutrino and BG interactions. To do this, a machine learning (ML) approach was proposed to speed up the rate at which images are classified while maintaining a degree of accuracy equal to or better than of a human. The results of this classification problem are discussed and analyzed in this paper.

1.2 Goal

The goal of this project was to design and implement a ML based software capable of correctly classifying photon images produced from a LiquidO neutrino detector. The aim was to have a 99% correct classification rate where for 9,000 images, no more than 90 are misclassified. Some challenges in classifying the images include the Compton scattering patterns produced during gamma ray and positron decay seen in Fig. 1(b) and (c) making them potentially difficult to distinguish.

2 Theory & Motivation

2.1 Data

Different versions of the data were constructed at various points. For a convolutional neural network (CNN), two datasets are required, training and testing, as well as three forms of sets: training, validation, and testing. The training set is the data fed into the CNN on each iteration of the epochs to train on. An epoch is the number of times the training data is passed through the model during training. The validation set is used to validate the model during training. The images in the validation set are not the same as in the training set and for this case are 10% of the training dataset. This portion helps to give an idea of how the network is performing on data not in the training set. With a validation set, overfitting becomes significantly more obvious if the accuracy on the training data is high and low on the validation. During training, after each epoch the algorithm should tend to improve on accuracy and reduce the loss with the validation set results being slightly lower than the training set. The testing set is used to determine the CNN's performance on a set of data that it has not been trained or validated on. This gives more definitive results as to the performance of the CNN. For this case, there a three

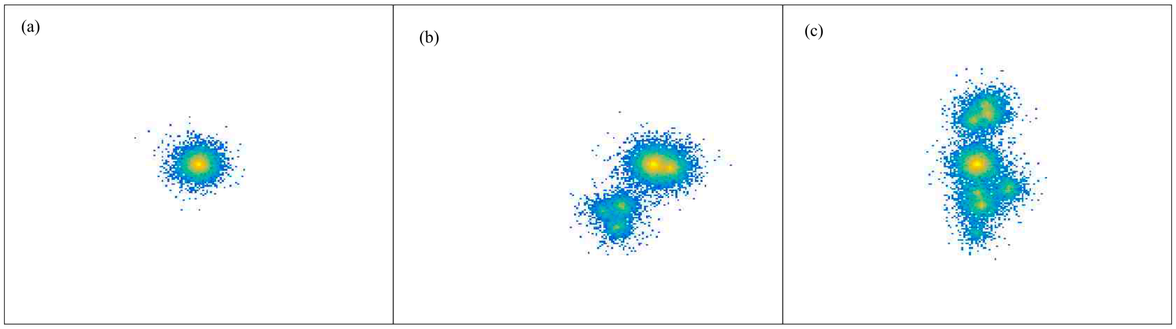


Figure 1: Photon patterns produced by different source collisions with a LiquidO neutrino detector. The brighter (more yellow) a pixel, the more photon collisions occured with the fiber at that point. (a) simulation results for an electron with a 1 MeV system. (b) simulation results for a gamma ray with a 2.5 MeV system. (c) simulation results for a positron with a 1 MeV system.

classes of images: electron, gamma ray, and positron as seen in Fig 1. The data consists of images of size 224x224x3 (the third number refers to the number of channels where 3 means coloured). The images are generated using the Monte Carlo simulation package provided by the LiquidO collaboration on the Queen's "neutrino" Linux machine. The electron, gamma ray, and positron images are simulated under the parameters: scattering length of 0.5 cm, lightyield (photons/MeV) of 10,000, fiber spacing (pitch) of 1.0 cm, and a fiber diameter of 0.50 mm. The energies of the particles simulated for each class are 1.0 MeV, 2.5 MeV, and 1.0 MeV for electron, gamma ray, and positron respectively. Note that the gamma ray is simulated with a higher energy as this energy is closer to that of a common gamma ray background. Additionally, for the positron energy, the 1.0 MeV is only for the central ball, with two 0.511 MeV gamma rays added from the positron annihilation bringing this closer to 2.0 MeV similar to the gamma ray.

2.2 CNN: VGG16 Model

When considering the method to classify electron, gamma ray, and positron photon images, a deep learning (DL) approach was taken as opposed to a classical one. While classical methods are good for non-class-specific algorithms, they require manual adjustments to select features (filtering out unwanted features by thresholding). The images that are being used to train the models (Fig 1) do not contain an "obvious" set of features and thus a DL approach was considered. DL requires an abundance of data for an artificial neural network (ANN) to train on [7]. Since photon images are simulated, there is essentially an infinite number of images that could be produced. In this paper, data sizes of 9,000 and 15,000 images were used to train on. A CNN specifically was chosen as they are shift and space invariant meaning they are not significantly impacted by the perspective/orientation of the input images.

To determine the structure of the CNN to train with, pre-existing models of standardized networks were considered due to their proven effectiveness with image classifications problems. AlexNet, created in 2012 is one of the more popular neural networks with its strong performance in the ImageNet Scale Visual Recognition Challenge (ILSVRV) having an 84.7% top-5 test error rate performance on the ILSVRC-2010 dataset. The structure consists of 5 convolutional layers and 3 fully-connected layers (total of 8 layers) [8]. VGG-16, which was brought forward in 2014 at the ImageNet Challenge which training on the ImageNet dataset, had a 92.7% top-5 test accuracy. The VGG structure as a whole is known to improve on the AlexNet structure as it increases the number of layers creating a denser network while decreasing the size of filters used and having an overall improved performance on different datasets. VGG16 in particular outperformed previous VGG models on ILSVRC-2012 and 2013 competitions. With a VGG model however, the drawbacks come in the time to train and amount of space the model takes up due to its denser network structure [8].

Since there were minimal restrictions on the amount of time to train the model and sufficient space for a VGG model to be stored on the supplied server (10.83 TB server), a VGG16 model was chosen. The overall design and structure for this CNN can be seen in Fig. 2. The convolutions collect and store features using filters in a feature map. The

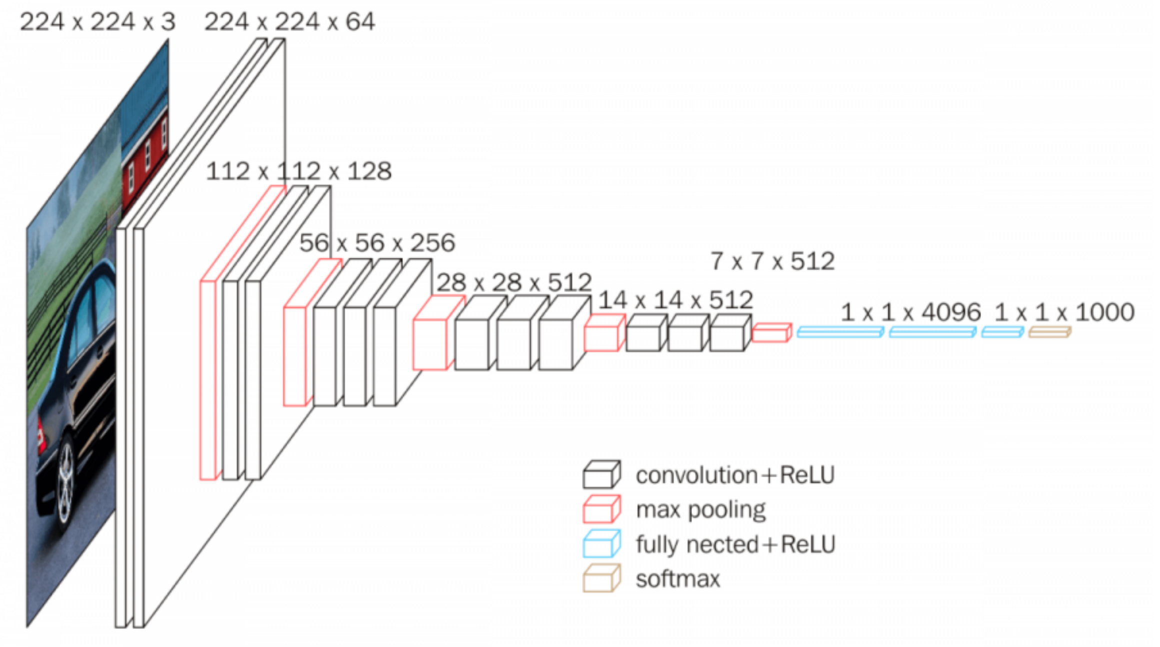


Figure 2: Visual Geometry Group 16 (VGG16) structure for a convolutional neural network (CNN). All convolutions (blue) are performed with a 7x7 kernel (filter) size [8].

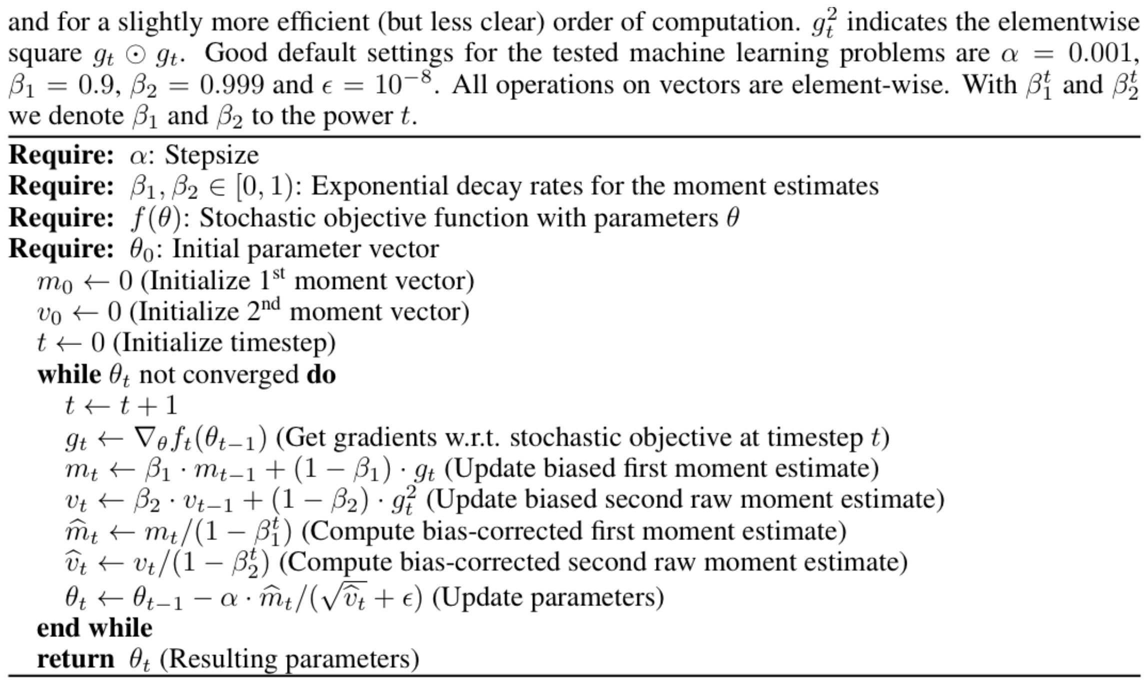
ReLU activation function introduces non-linearity into the system. Max pooling takes the largest value in a section of the created feature map, pooling the sections into a down-sampled structure. The fully connected layers, which takes in a flattened result following the last max pooling layer, are a small feed-forward neural network (NN) where all the nodes from one level, connect to all the nodes of the next. The softmax layer reduces the results from the last NN layer into values ranging from 0 to 1 for the number of possible classes. The results represent the confidence the input image is in each of the classes and a prediction can therefore be made. The predicted class is whichever has the highest confidence on the output.

2.3 Adam Optimizer

An optimizer in a NN is an algorithm or function used within each epoch (iterations through the training set) to adjust the weights and learning rates of the network thus improving the loss and accuracy of the model being trained [9]. The optimizer modifies the parameters the cost function is dependent on to minimize the function and thus achieve accurate prediction abilities. This is often done in a form of gradient descent.

The Adam (adaptive moment estimation) optimizer, was designed based on the methodology around stochastic gradient descent (SGD) but with some added improvements. Adam computes individual learning rates for different parameters that can be modified and are based on approximations for first and second moments of the gradients [10]. The structure for the optimizer can be seen in Fig. 3 where g_t is the gradient of the cost function, the first moment vector of the gradient is the mean of the moving average, the second moment is the uncentered variance of the gradient, the βs (between 0 and 1) control the exponential decay rates of the moving averages. The setup continuously updates the exponential moving averages of the gradient to update the moments, followed by the parameters of the cost function [10]. The bias-correction used at the end of the while loop in Fig. 3 is implemented based on the derivation of the second moment estimate shown by Kingma in section 2 of ref. [10].

Adam was chosen as the optimizer as it is well-suited for machine learning based



and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1$^{\text{st}}$ moment vector)
  $v_0 \leftarrow 0$ (Initialize 2$^{\text{nd}}$ moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t/(1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t/(1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t/(\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

Figure 3: Pseudo code structure for the Adam optimizer algorithm. The algorithm takes in the stepsize, α, the exponential decay rates for the two moments, β_1, β_2, the cost function, $f(\theta)$, and a vector of initialized parameters, θ_0, and outputs converged parameters to update the weights and biases of the nodes in the NN. [10]

problems which utilize large datasets in combination with high-dimensional parameters [10]. VGG16, the chosen CNN requires large data sets to function and creates high-dimensional features making this optimizer a strong fit. An additional reason for the choice is based on the logistic regression analysis shown by Kigma in ref. [10] where they compare the rate of decay of training costs for different optimizers with various datasets. The results of the comparison show Adam as the leading contender when compared to AdaGrad, RMSProp, SGDNesterov, and AdaDelta using MNIST and CIFAR10 datasets with different multi-layered neural networks.

2.4 Cross-Entropy Cost Function

Cost functions, which for individual events are known as a loss functions, give the ability to estimate the performance of a machine learning model. By estimating the difference/distance between the predicted and actual value of the output, the function is able to measure the incorrectness of the model [11].

For the chosen VGG16 model and Adam optimizer, a cross-entropy cost/loss function was selected. Cross-entropy measures the difference between probability distributions (usually 2) for a set of events of random variables [12]. In regards to information theory, there are lower and higher probability events where the low contain more information and the high, less since the low are less likely to occur. Given an event, the information can be calculated as in Eq. (1) where $x, h(x)$, and $P(x)$, are an event, the information for the event, and probability for the event respectively [12].

$$h(x) = -log(P(x)) \tag{1}$$

Entropy refers to the minimum number of bits needed to transmit a (random) event from a probability distribution. A skewed distribution would have a low entropy while a balanced distribution would have a high entropy [13]. Given a discrete set of states, X, the entropy (H(x)) can be represented as seen in Eq. (2) [11].

$$H(x) = \sum_{x}^{X} P(x) * log(P(x)) \tag{2}$$

Cross-entropy refers to the the average number of bits required to represent an event from one distribution (Q) in another (P) and is represented by $H(P, Q)$. Here, H is the cross-entropy function, P is the target distribution and Q is the approximation of P. The cross-entropy can then be calculated as per Eq. (3) [11].

$$H(P, Q) = -\sum_{x}^{X} P(x) * log(Q(x)) \tag{3}$$

This type of cost function is particularly good, and often used with classification models. For a known class label, the probability is 1 while the rest of the class probabilities are 0. So for a known electron, the probability distribution would be [1.0, 0.0, 0.0], where the other probabilities are for the gamma ray and positron classes. For this case, the random variable is the predicted class and the events are the classes to be predicted [11]. As a result, the cross-entropy model in Eq. (3) can be used where P is the expected probability, Q is the predicted probability, and x is the class. Given that this is a multi-class classification problem (more than 2 classes), the predictions for classes are converted into distributions of probabilities across each event [11]. Thus, the average of the distribution across the dataset can be calculated. The results are reported in units of nats (since a log of base 10 is used). When looking at loss, for an arbitrary problem a loss below 0.20 is considered ok, whereas a loss less than 0.02 suggests at very good predictions being made where 0 is perfect classification [11].

A cross-entropy cost function was chosen to train with for this model of CNN as the model is built to solve a multi-class classification problem. It is a well-documented, thoroughly used and effective method of calculating loss and for the given style of problem, fits very well with the trend.

3 Maintainability

The software designed and implemented was done so with a focus on readability and maintainability. All variables, objects, classes, and files have been named appropriately such that they can be understood by any engineer and/or developer. A description of the project, instructions on running the code and reproducing the results, as well as some resources and the code itself are all recorded in a GitHub repository found here. In order to keep track of the libraries used and their versions, a `requirements.txt` file is included in the repository such that the exact versions can be used to prevent backwards compatibility issues with newer versions. Additionally, a `README.MD` file is included to walk the user through the different aspects of the repository and how to use it. In terms of design, one of the main libraries used in the code is PyTorch, an open-source machine learning library. Some others for model setup and testing analysis include `sklearn`, `torchvision`, `numpy`, and `matplotlib`.

Some concerns still exist within the setup that could not be addressed. Firstly, a heavy reliance on an open-source library does come with risk if the versions of the code are updated and older ones become unavailable. Note that this can be avoided such that the software is maintained and updated. Additionally, if the code is to be updated with newer versions, updates can be non-backwards compatible meaning sizeable changes could be needed to maintain the same results. The files take up minimal memory though the size of trained model is approximately 719.60 MB. While there is no current issue, if memory requirements for system change in the future, a smaller CNN model may be needed which does not guarantee the same level of accuracy.

4 Constraints

Majority of constraints set on this project came from the services available to develop with. While the amount of data supplied was not an issue nor the amount of space to store files on the supplied server (or personal computer), the GPU came with restrictions. Two CUDA version 11.4 GPUs were available for use, one with a memory of 12,207 MiB (12,800 MB) and the other with 24,268 MiB (25,447 MB). While this appears to be a significant amount of space, there are limitations on how much memory can be taken by a process considering it is a shared GPU. The GPU allowed for a maximum of 352.56 MiB with 11.85 GiB set aside where PyTorch took up 11.26 GiB of the 11.85. As the number of training images grew, so did the batch size where batch size refers to the number images used in each epoch. For 9,000 training images, a batch size of 64 worked well (the initial choice). However for 15,000 images, the GPU would run out of memory. Therefore a smaller batch size of 60 was substituted. As a result, the limitations of the GPU directly effected how the CNN could be trained.

5 Process & Results

The algorithm construction began with developing an understanding of how to build a CNN as well as how it can be implemented with PyTorch. Once there was confidence in how to use the library, construction of the VGG16 CNN began. At this point, the Adam optimizer and cross-entropy cost function were chosen for training. The neural network was then tested with a miniature dataset (10 images per class) to check for syntax, runtime, and logic errors. After some initial debugging and the creation of a testing file, the first iteration of the dataset was simulated.

The results begin with the first set of 9,000 images being trained with and tested on (3,000 per class, 18,000 total for training and testing). The images used were generated such that each photon pattern began at the center of the plot (position (0,0)) with a randomized rotation applied to the image before training. For the training loop, a batch size of 64 was used. The accuracy on the trained model was 99.4% meeting the goal of this project. The classification results for this model can be seen in Fig. 4. Despite the initial success of the model, there was still significant room for improvement. A new set of images were generated in the same format ((0,0) starting position) however, now with 5000 images per class (15,000 images for training and 15,000 for testing). The accuracy of the new model improved to 99.8%. The classification values can be seen in Fig 5. Note that for this model, a batch size of 60 was used due to the memory constraints of the GPU. The model appears to have improved with the adding of images though due to time constraints, a larger image set of the (0,0) starting images was not generated. Due to the strong performance of the CNN, it was decided to give the model a more difficult set of data to test on to ensure the correct features from the images were being recognized and the model was not overfitting to non-shifted ((0,0) starting) images. Therefore, a 15,000 image testing set was generated where the starting position of the photon pattern (collision position) was uniformly and randomly shifted by up to ± 20 units along the x- and/or y-axis. After testing the model trained with 15,000 non-shifted images on the 15,000 shifted

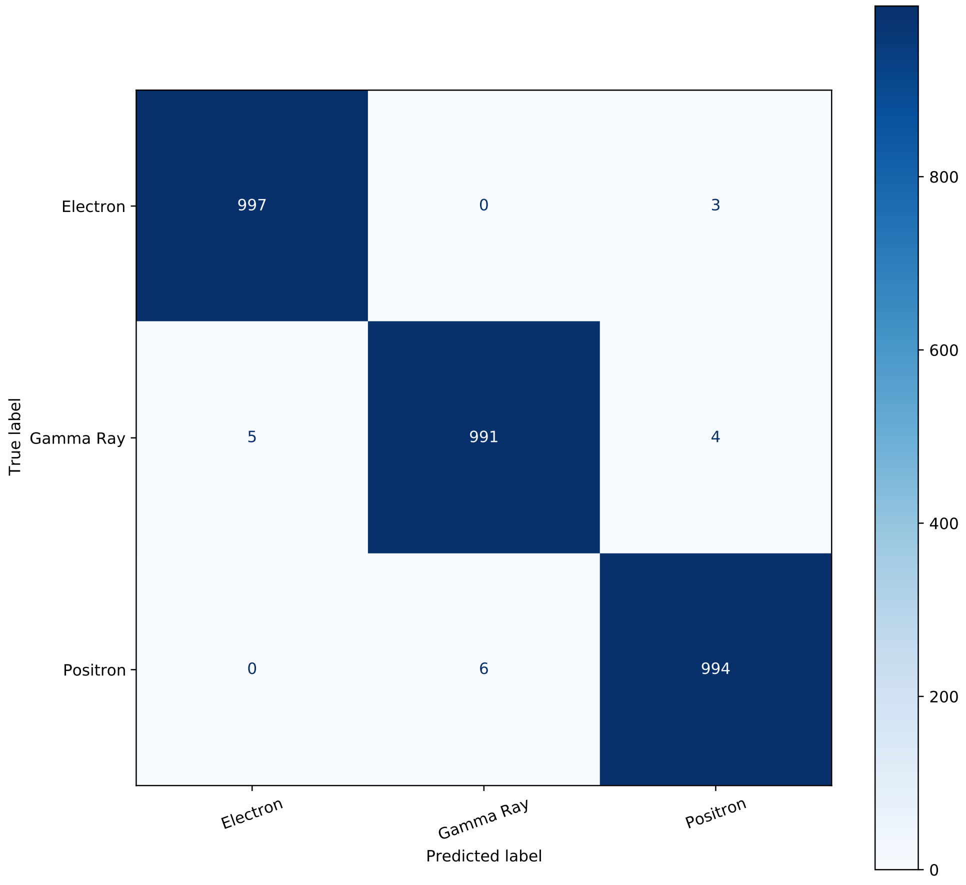


Figure 4: Confusion matrix of the initial model plotting the true class label of the image against the predicted label. The matrix shows the results of the VGG16 model trained with 9,000 non-shifted images and tested with 9,000 non-shifted images.

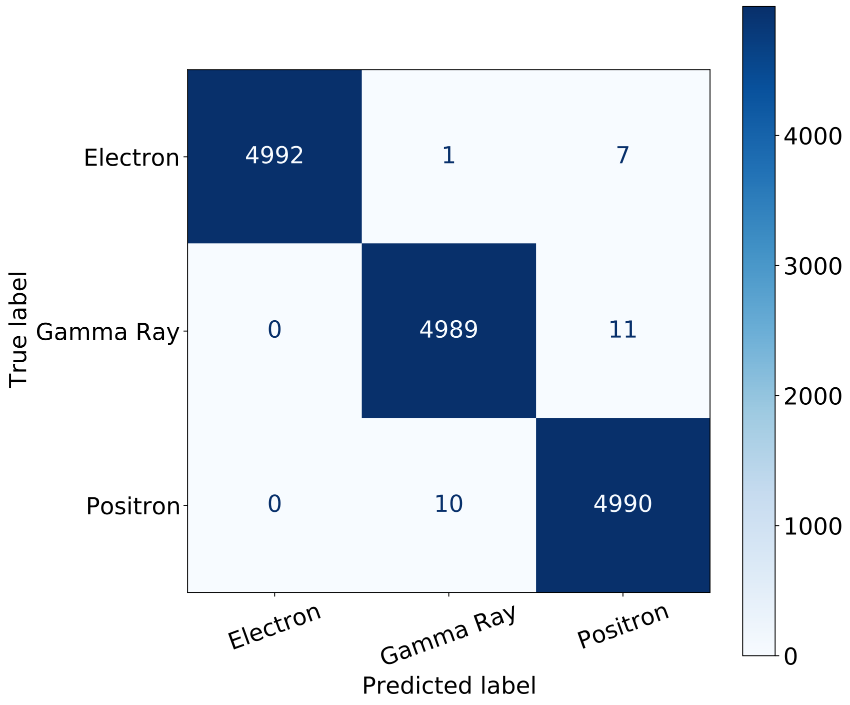
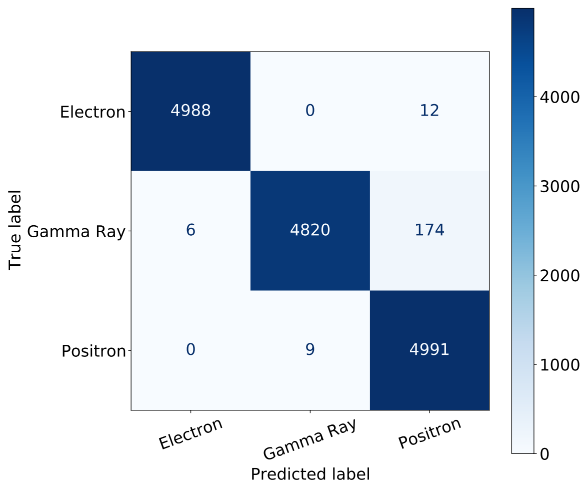


Figure 5: Confusion matrix plotting the true class label of the image against the predicted label. The matrix shows the results of the VGG16 model trained with 15,000 non-shifted images and tested with 15,000 non-shifted images.

images set, the accuracy of the model dropped quite drastically to a value of 75.3%. This result confirmed the idea that the model may be focusing on the wrong features of the model, clearly relying on the (0,0) position more than previously considered. To counteract the drop in accuracy, a new set of testing and validation images were generated with shifting and rotations applied. Following this, the shifted image testing set was re-tested with the new model trained on shifted images. The results showed a 98.67% testing accuracy for the the non-shifted images and 98.50% on the shifted images with test losses of 0.035 and 0.042 respectively. The relative confusion matrices can been seen in Fig. 6. With the model now falling just short of the goal set out, it was decided to change the initial set of parameters used to train the model. Before, the NN was initialized with a set of arbitrary values trained from the MNIST training set to save time, rather than starting from randomized values. To improve the model, instead of this set, the new model was initialized with final parameters taken from an earlier model generated using 15,000

(a) 

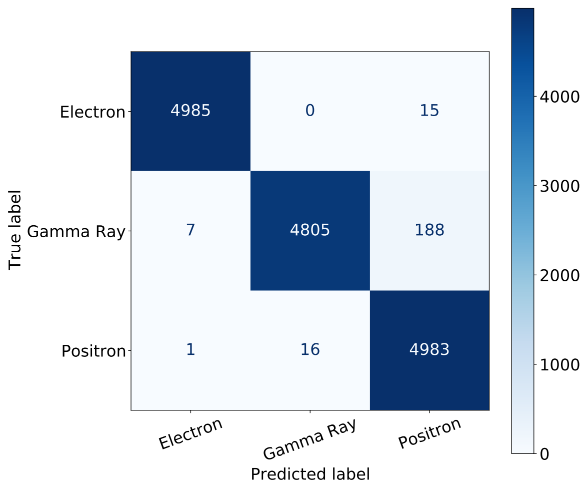
(b) 

Figure 6: Confusion matrices plotting the true class label of the image against the predicted label. (a) results of the VGG16 model trained with 15,000 shifted images and tested with 15,000 non-shifted images. (b) results of the VGG16 model trained with 15,000 shifted images and tested with 15,000 shifted images.

non-shifted images (the one with Figure 5 results). The model training with the 15,000 shifted images was then re-trained with this new starting point and the model ended with an accuracy of 99.29% and test loss of 0.027. The confusion matrix can be seen in Fig. 7. The size of the final model ended with 134,281,283 parameters (weights and biases) all of which are trainable, and with a total model size of approximately 719.60 MB. The full structure breakdown with size outputs and parameter counts can be seen in Appendix B. To validate the results of the shifted model, it was tested with the non-shifted 15,000 image testing set. The accuracy of classification was shown to be 99.33% with a test loss of 0.024. The confusion matrix can be seen in Fig 8. As suspected, the accuracy of the model on the non-shifted images was higher than on the shifted images despite the model being trained on shifted images as they are easier classify. While all of the different model variations were being tested, a set of misclassified images were also plotted along with the confusion matrices to ensure that the images being misclassified were “reasonable” misclassifications. A reduced version of the created images can be seen in Fig. 9. The full-sized figure can be seen in Appendix B.

After going through many of the misclassified images with Mark Chen, there were no

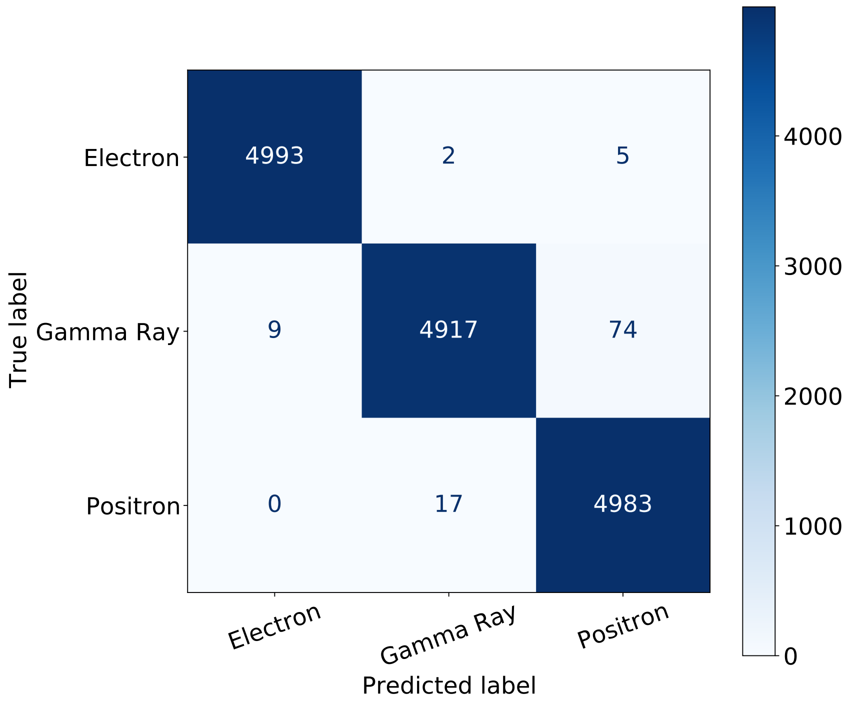


Figure 7: Confusion matrix plotting the true class label of the image against the predicted label. The matrix shows the results of the VGG16 model trained with 15,000 shifted images and tested with 15,000 shifted images where the initial parameter values began with the last values of the non-shifted 99.8% accuracy model.

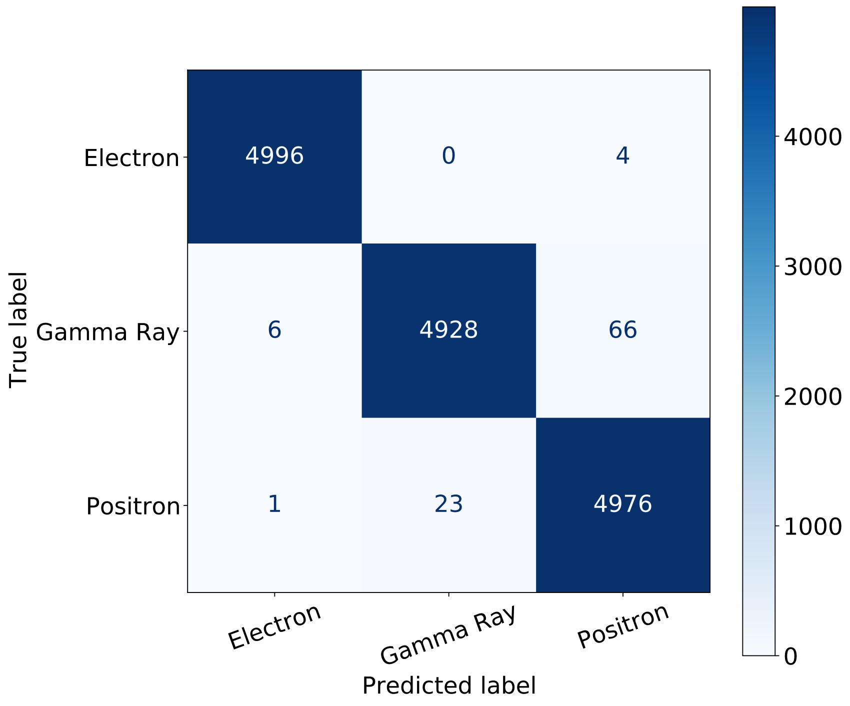


Figure 8: Confusion matrix plotting the true class label of the image against the predicted label. The matrix shows the results of the VGG16 model trained with 15,000 shifted images and tested with 15,000 non-shifted images where the initial parameter values began with the last values of the non-shifted 99.8% accuracy model.

explicit misclassified images that a human might not make themselves. For the images that are electrons and were classified as gamma rays/positrons, the image does appear to follow the Compton scattering pattern of a gamma ray/positron as seen in the top left of Fig. 9. This can be a direct result of a Bremsstrahlung interaction where an electron loses speed next to another atom and undergoes an interaction which causes a gamma ray to shoot off, thus creating a Compton scattering pattern similar to a gamma ray or positron. For the case of a gamma ray appearing as an electron, this can result from the photoelectric effect where a gamma ray is absorbed on its first interaction and causes the release of a single electron making the pattern appear as an electron. A good example of this can be seen in the top row of Fig. 4. The last type of observed misclassification are positrons being classified as gamma rays. One possibility for this is that one of the two gamma rays produced during the positron's annihilation could have scattered vertically (on the z-axis) or absorbed quickly such that the second pattern cannot be identified and the image appears more like a gamma ray. Examples of this can be seen in the bottom row of Fig. 9.

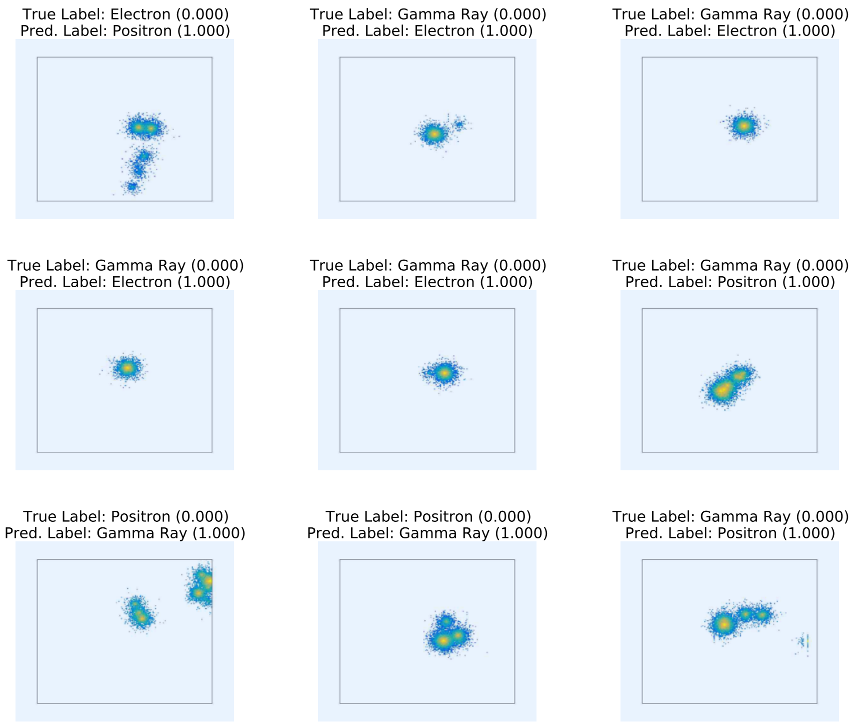


Figure 9: A subset of images misclassified by the VGG16 model from Fig. 7. The numbers show the confidence the model had in the predicted and true labels (output from the softmax layer of the CNN).

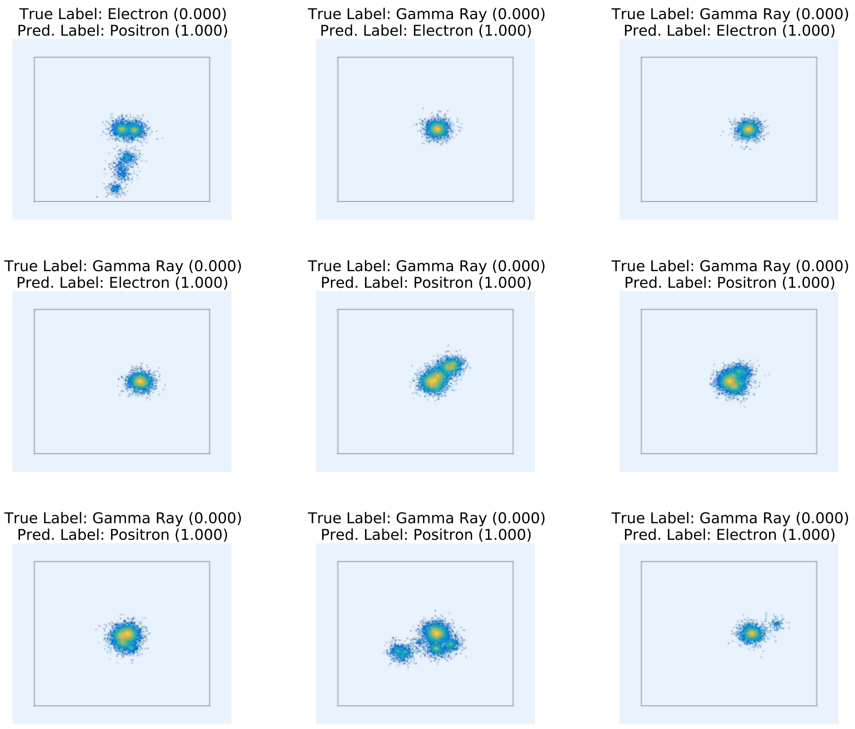


Figure 10: A subset of images misclassified by the VGG16 model from Fig. 8. The numbers show the confidence the model had in the predicted and true labels (output from the softmax layer of the CNN).

6 Next Steps

Moving forward, the goal would be to improve on the abilities of the CNN classifier by increasing the size of and number of variations in the dataset as well as attempting to reduce the memory requirements of the trained model. To increase the size of the dataset, simply modifying the generating files to simulate more images can be done with ease. For variations, adding in particle collisions with different energies and collision paths to each classes' image sets would achieve this. This can be done again by modifying the generating files' parameters or even to randomize for each created image within certain ranges.

To reduce the amount of memory the trained model occupies, a different form of CNN could be considered. While VGG16 has proven itself capable of meeting the 99% accuracy goal, it is known for taking up sufficient amounts of memory, in this case, approximately 719.60 MB. While the neutrino servers currently have no issues with this size of file, this is subject to change as it is a shared server. Considering a smaller CNN, such as a smaller VGG structure, AlexNet, or even a ResNet model, will take up less space than VGG16 though there is no guarantee it can maintain the same level of accuracy. To do this, development and testing will need to be done.

Another possibility to improve the performance of the model is to introduce three dimensional images into the dataset (potentially replacing the two dimensional ones). This could help to reduce the number of misclassified images. For example, where a positron may be classified as a gamma ray due to one of the gamma rays produced during the positron's decay propagating along the z-axis thus its Compton scattering pattern not being visible in a two dimensional image.

The last thing to take into account when moving forward is to keep software versions up to date. Keeping track of when PyTorch (among other library dependencies) are updated will help to know when new features that may be useful become available in addition to when new updates are no longer backwards compatible. Therefore, if the software is being improved on and/or updated, some minor and potentially major changes may be needed to implement the newer versions. Since all the code has been documented in

GitHub, this helps with version control and to create different versions of the same code thus maintaining older versions as a cautionary.

7 Conclusion

Using a VGG16 convolutional neural network (CNN) combined with an Adam optimizer and cross-entropy cost function for training, the 99% accuracy goal, where no more than 90 images would be misclassified for a 9,000 image dataset was not only achieved but exceeded on. With 9,000 images to train on, the CNN model was able achieve a 99.4% accuracy. Expanding the problem to a 15,000 image dataset, with only non-shifted images a 99.8% accuracy was reached. Further challenging the trained model to perform, using a 15,000 image dataset consisting of photon images shifted off from the (0,0) axis position, the model under performed reaching only a 75.3% accuracy. To improve, the model was trained on a new shifted image dataset, though still falling short with only a 98% accuracy when tested on both the shifted and non-shifted dataset. Finally, the model was able to reach the goal on both shifted and non-shifted images with one model, trained on shifted images where the initial parameter values began with the final parameters of model trained on 15,000 non-shifted images.

While the combination of the VGG16 CNN with the Adam optimizer and cross-entropy cost function has worked well when classifying images from LiquidO, the model can be improved. For example, diversifying and increasing the size of the training set to images with particle collisions of various energies as well as potentially adding in three dimensional images. Another improvement could be reducing the size of model from 719.60 MB by potentially changing the overall CNN structure to a smaller one though testing would be needed to ensure performance metrics are still met.

Considering the overall setup, documentation, and design of software, there are many areas that can be improved and expanded on, and the storage of the code in Github allows this to be done with ease.

Word Count

4,612 words.

References

[1] C. L. Cowan, F. Reines, F. B. Harrison, H. W. Kruse, and A. D. McGuire, "Detection of the free neutrino: A Confirmation," *Science*, vol. 124, pp. 103–104, 1956. DOI: `10.1126/science.124.3212.103`.

[2] V. Albanese, R. Alves, M. Anderson, *et al.*, "The SNO experiment," *Journal of Instrumentation*, vol. 16, no. 08, P08059, 2021. DOI: `10.1088/1748-0221/16/08/p08059`. [Online]. Available: `https://doi.org/10.1088%2F1748-0221%2F16%2F08%2Fp08059`.

[3] G. Alimonti, C. Arpesella, H. Back, *et al.*, "The borexino detector at the laboratori nazionali del gran sasso," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 600, no. 3, pp. 568–593, 2009. DOI: `10.1016/j.nima.2008.11.076`. [Online]. Available: `https://doi.org/10.1016%2Fj.nima.2008.11.076`.

[4] P. Adamson, C. Ader, M. Andrews, *et al.*, "First measurement of muon-neutrino disappearance in NOvA," *Physical Review D*, vol. 93, no. 5, 2016. DOI: `10.1103/physrevd.93.051104`. [Online]. Available: `https://doi.org/10.1103%2Fphysrevd.93.051104`.

[5] J. B. Birks, *The theory and practice of Scintillation Counting*, ser. International Series of Monographs in Electronics and Instrumentation. Pergamon, 1964.

[6] A. Cabrera, A. Abusleme, J. dos Anjos, *et al.*, "Neutrino physics with an opaque detector," *Communications Physics*, vol. 4, no. 1, 2021. DOI: `10.1038/s42005-021-00763-5`. [Online]. Available: `https://doi.org/10.1038%2Fs42005-021-00763-5`.

[7] N. O. Mahony, S. Campbell, A. Carvalho, *et al.*, "Deep learning vs. traditional computer vision," *CoRR*, vol. abs/1910.13796, 2019. arXiv: `1910.13796`. [Online]. Available: `http://arxiv.org/abs/1910.13796`.

[8] V. Kurama, *A review of popular deep learning architectures: Alexnet, vgg16, and googlenet*, 2020.

[9] A. Gupta, *A comprehensive guide on deep learning optimizers*, 2021. [Online]. Available: `https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/#:~:text=An\%20optimizer\%20is\%20a\%20function,loss\%20and\%20improve\%20the\%20accuracy.`.

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[11] C. Mc., *Machine learning fundamentals (i): Cost functions and gradient descent*, 2017. [Online]. Available: `https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220`.

[12] J. Brownlee, *A gentle introduction to cross-entropy for machine learning*, 2019. [Online]. Available: `https://machinelearningmastery.com/cross-entropy-for-machine-learning/`.

[13] I. Koppert-Anisimova, *Cross-entropy loss in ml*, 2021. [Online]. Available: `https://medium.com/unpackai/cross-entropy-loss-in-ml-d9f22fc11fe0#:~:text=Entropy\%20is\%20the\%20number\%20of,multi\%2Dclass\%20cross\%20entropy\%20loss.`.

A Statement of Work and Contributions

The work completed in the fall term for this project consisted of research into the structure of convolutional neural networks (CNNs), how they are implemented, how to use PyTorch, and into how LiquidO works. Additionally, some test scripts were run in the fall term to ensure libraries being used were completely understood. The winter term consisted of implementing the VGG16 CNN, simulating all used datasets, training and testing the models, and writing the final paper.

B Structure & Results

Table 1: Final structure of the model trained on 15,000 shifted and rotated images. The parameters refers to the number of weights and biases in the CNN. The parameters total 134,281,283 with all them being trainable. The estimated total size of the model is 719.60 MB.

Layer (type:depth-idx)	Output Shape	Param #
Sequential: 1-1	–	–
Conv2d: 2-1	[-1, 512, 7, 7]	–
Conv2d: 2-1	[-1, 64, 224, 224]	1,792
BatchNorm2d: 2-2	[-1, 64, 224, 224]	128
ReLU: 2-3	[-1, 64, 224, 224]	–
Conv2d: 2-4	[-1, 64, 224, 224]	36,928
BatchNorm2d: 2-5	[-1, 64, 224, 224]	128
ReLU: 2-6	[-1, 64, 224, 224]	–
MaxPool2d: 2-7	[-1, 64, 112, 112]	–
Conv2d: 2-8	[-1, 128, 112, 112]	73,856
BatchNorm2d: 2-9	[-1, 128, 112, 112]	256
ReLU: 2-10	[-1, 128, 112, 112]	–
Conv2d: 2-11	[-1, 128, 112, 112]	147,584

BatchNorm2d: 2-12	[-1, 128, 112, 112]	256
ReLU: 2-13	[-1, 128, 112, 112]	–
MaxPool2d: 2-14	[-1, 128, 56, 56]	–
Conv2d: 2-15	[-1, 256, 56, 56]	295,168
BatchNorm2d: 2-16	[-1, 256, 56, 56]	512
ReLU: 2-17	[-1, 256, 56, 56]	–
Conv2d: 2-18	[-1, 256, 56, 56]	590,080
BatchNorm2d: 2-19	[-1, 256, 56, 56]	512
ReLU: 2-20	[-1, 256, 56, 56]	–
Conv2d: 2-21	[-1, 256, 56, 56]	590,080
BatchNorm2d: 2-22	[-1, 256, 56, 56]	512
ReLU: 2-23	[-1, 256, 56, 56]	–
MaxPool2d: 2-24	[-1, 256, 28, 28]	–
Conv2d: 2-25	[-1, 512, 28, 28]	1,180,160
BatchNorm2d: 2-26	[-1, 512, 28, 28]	1,024
ReLU: 2-27	[-1, 512, 28, 28]	–
Conv2d: 2-28	[-1, 512, 28, 28]	2,359,808
BatchNorm2d: 2-29	[-1, 512, 28, 28]	1,024
ReLU: 2-30	[-1, 512, 28, 28]	–
Conv2d: 2-31	[-1, 512, 28, 28]	2,359,808
BatchNorm2d: 2-32	[-1, 512, 28, 28]	1,024
ReLU: 2-33	[-1, 512, 28, 28]	–
MaxPool2d: 2-34	[-1, 512, 14, 14]	–
Conv2d: 2-35	[-1, 512, 14, 14]	2,359,808
BatchNorm2d: 2-36	[-1, 512, 14, 14]	1,024
ReLU: 2-37	[-1, 512, 14, 14]	–
Conv2d: 2-38	[-1, 512, 14, 14]	2,359,808
BatchNorm2d: 2-39	[-1, 512, 14, 14]	1,024

ReLU: 2-40	[-1, 512, 14, 14]	–
Conv2d: 2-41	[-1, 512, 14, 14]	2,359,808
BatchNorm2d: 2-42	[-1, 512, 14, 14]	1,024
ReLU: 2-43	[-1, 512, 14, 14]	–
MaxPool2d: 2-44	[-1, 512, 7, 7]	–
AdaptiveAvgPool2d: 1-2	[-1, 512, 7, 7]	–
Sequential: 1-3	[-1, 3]	–
Linear: 2-45	[-1, 4096]	102,764,544
ReLU: 2-46	[-1, 4096]	–
Dropout: 2-47	[-1, 4096]	–
Linear: 2-48	[-1, 4096]	16,781,312
ReLU: 2-49	[-1, 4096]	–
Dropout: 2-50	[-1, 4096]	–
Linear: 2-51v	[-1, 3]	12,291

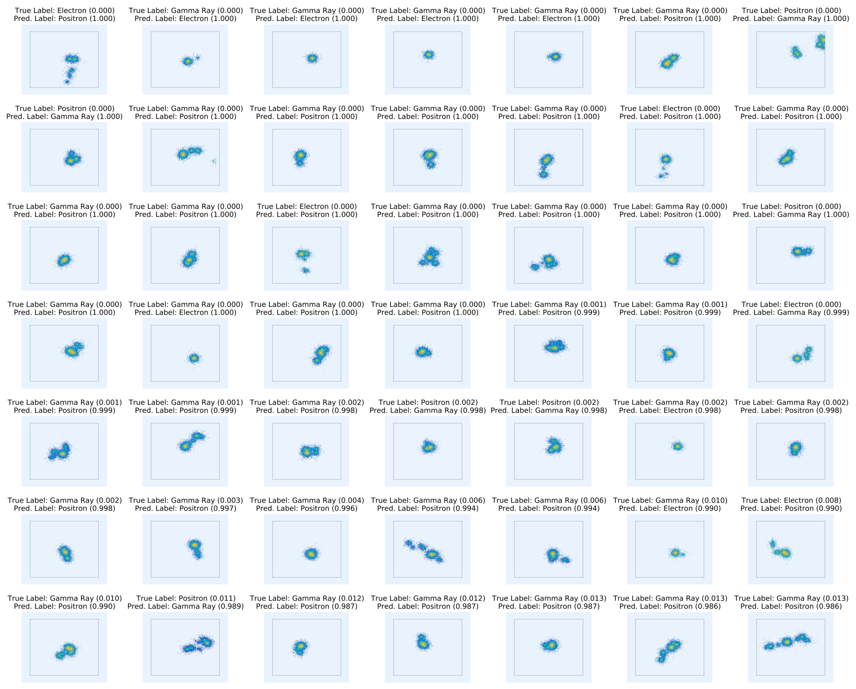


Figure 11: Images misclassified by the VGG16 model from Fig. 7. The numbers show the confidence the model had in the predicted and true labels (output from the softmax layer of the CNN).