Astroparticle Physics 104 (2019) 42–53

Contents lists available at ScienceDirect

Astroparticle Physics

journal homepage: www.elsevier.com/locate/astropartphys

Particle identification in camera image sensors using computer vision

Miles Winter[a,b], James Bourbeau[a,b], Silvia Bravo[a,b], Felipe Campos[b,c], Matthew Meehan[a,b,*], Jeffrey Peacock[e], Tyler Ruggles[a], Cassidy Schneider[a,b], Ariel Levi Simons[d], Justin Vandenbroucke[a,b]

[a] Department of Physics, University of Wisconsin-Madison, Madison, WI 53706, USA
[b] Wisconsin IceCube Particle Astrophysics Center, Madison, WI 53703, USA
[c] University of California, Berkeley, CA 94720, USA
[d] University of Southern California, Los Angeles, CA 90007, USA
[e] Sensorcast, Boulder, CO 80305, USA

ARTICLE INFO

Article history:
Received 11 March 2018
Revised 21 June 2018
Accepted 19 August 2018
Available online 20 August 2018

Keywords:
Cosmic rays
Deep learning
Convolutional neural network
Classification
Citizen science

ABSTRACT

We present a deep learning, computer vision algorithm constructed for the purposes of identifying and classifying charged particles in camera image sensors. We apply our algorithm to data collected by the Distributed Electronic Cosmic-ray Observatory (DECO), a global network of smartphones that monitors camera image sensors for the signatures of cosmic rays and other energetic particles, such as those produced by radioactive decays. The algorithm, whose core component is a convolutional neural network, achieves classification performance comparable to human quality across four distinct DECO event topologies. We apply our model to the entire DECO data set and determine a selection that achieves $\geq 90\%$ purity for all event types. In particular, we estimate a purity of 95% when applied to cosmic-ray muons. The automated classification is run on the public DECO data set in real time in order to provide classified particle interaction images to users of the app and other interested members of the public.

1. Introduction

The ubiquity of smartphone devices worldwide has sparked an explosion in the field of distributed sensors; their widespread adoption has effectively instrumented global population centers with a variety of detectors. The CMOS image sensors in modern smartphones are based on similar semiconductor technology to that found in professional telescopes and particle physics detectors, enabling them to detect cosmic rays and other ionizing charged particles. These particles have long been a background nuisance for CCDs used in astronomical cameras [1], however several recent projects including the Distributed Electronic Cosmic-ray Observatory [2] seek to use this background as signal for both scientific and educational purposes. It may be possible for such networks of smartphones to detect extensive air showers created by ultra-high energy cosmic rays (UHECR) above 10^{20} eV, if challenging user density targets are met [3]. This is a powerful and cost-effective way to extend UHECR measurements to higher energies, but there are substantial hurdles to achieving this goal [4]. Since it is also possible to detect local radioactivity with camera sensors [5], networks of smartphones could be used as radiation monitors. More exotic analyses have also been proposed, such as searching for correlated extensive air showers created when an ultra-high-energy photon interacts with the heliosphere [6]. One major hurdle limiting these scientific pursuits is accurate and efficient particle identification, which is necessary to reject the radioactive background for cosmic-ray measurements or vice-versa for radiation measurements. In this paper we describe a computer vision algorithm developed to identify the charged particles detected by camera image sensors. We then apply it to the data set produced by the Distributed Electronic Cosmic-ray Observatory (DECO) [2,7], the first publicly available cosmic-ray smartphone application.

DECO detects cosmic rays by way of an Android application that began beta testing in October 2012 and was released publicly in September 2014. DECO is designed to detect ionizing radiation that traverses silicon image sensors in smartphones. The resulting dataset consists of images recorded by users worldwide (Fig. 1) that contain evidence of charged particle interactions. Due to the diverse ecosystem of Android phones on the market, the systematic variation in data taking conditions, and the variety of particle event morphologies, classification of DECO events

* Corresponding author at: University of Wisconsin-Madison, Wisconsin IceCube Particle Astrophysics Center, 222 West Washington Ave., Suite 500, Madison, WI 53703, United States.
E-mail addresses: winter6@wisc.edu (M. Winter), jbourbeau@wisc.edu (J. Bourbeau), mrmeehan@wisc.edu (M. Meehan).

https://doi.org/10.1016/j.astropartphys.2018.08.009
0927-6505/

Fig. 1. World map showing the global network of DECO users. Dots indicate data taking locations and span 80 different countries. Every continent including Antarctica is represented. Lines of data points, such as those in Antarctica and west of the Americas, indicate users running DECO on plane flights. Map plotted with a Kavrayskiy VII projection and up to date as of December 2017.

presents a unique challenge. Our initial work using straight cuts to classify events in the highly heterogeneous dataset was moderately successful in classifying some event types, but identifying a cosmic-ray muon sample with high purity proved challenging. We present a computer vision algorithm based on a convolutional neural network for classifying DECO events. Additional cosmic-ray cell phones apps mentioned above could also benefit from the approach described here. We presented initial results from our CNN classification in [8]. More recently, during preparation of this paper, Borisyak et al. [9] appeared and describes a CNN algorithm intended for use as an online cosmic-ray muon trigger.

2. DECO App

The DECO detection technique uses similar ionization-detecting semiconductor technology to that found in the silicon trackers of professional particle physics experiments [10,11]. Ionizing charged particles that travel through the sensitive region (i.e. depleted region) of a phone's image sensor are detected via the electron-hole pairs they create. The DECO app, which can be run on any Android device with Android version $\geq$ 2.1, is designed to be run with the camera face down or covered in order to minimize contamination from background light. While running, the app repeatedly takes long-duration ($\sim$50 ms) exposures and runs them through a two-stage filter to search for potentially interesting events. This filter first searches a low-resolution image for N pixels above an intensity threshold, and if passed, analyzes a high-resolution image in the same manner. The intensity is the sum of the red, blue, and green color values (RGB) for each pixel. Images that pass both filters are tagged as "events" and are automatically uploaded to a central database for offline analysis. Additionally, the app has a "minimum bias" data stream that saves one image every five minutes per device for offline calibration and noise studies. In particular, they are used to determine the appropriate value of N for the online filter to select potentially interesting events. The app's online filter is simple and efficient in order to maximize livetime, while more detailed analyses of images are performed offline. The DECO data can be browsed using a public website [12], where users can perform queries using various metadata including time stamp (UTC), latitude and longitude (rounded to nearest 0.01° for privacy), event vs. minimum bias categorization, Android phone model, and device ID.

Offline analysis of images that pass the app's online filter begins with a contour-finding algorithm to locate clusters of bright pixels. We use the marching squares algorithm, a special case of the marching cubes algorithm [13,14], to search for groups of at least

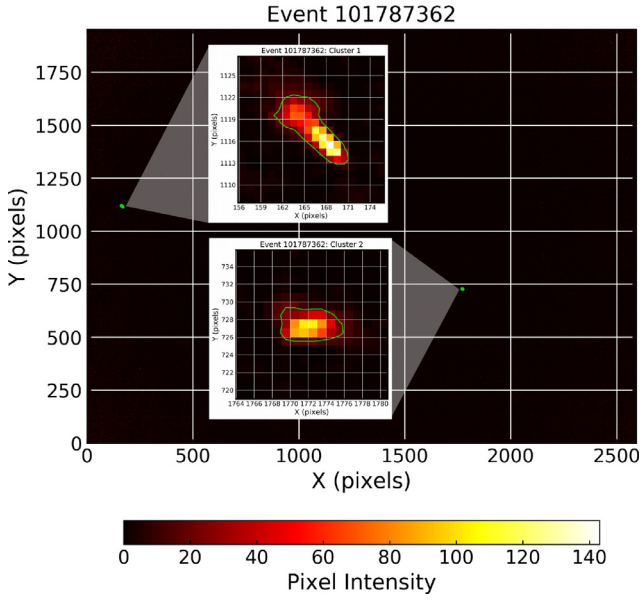

Fig. 2. Example of a full camera image that passed online filtering. During offline analysis, a contour-finding algorithm is used to identify hit clusters of pixels. In this event, two clusters (shown with green contours) were identified for further analysis and classification. The color scale represents the pixel intensity, scaled to the brightest pixel, after a conversion to grayscale. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

10 pixels with a minimum RGB sum of 20. These clusters of pixels are then grouped together at a higher level: any clusters within 40 pixels of one another are considered a single group. This grouping is to account for electrons which can scatter in and out of the camera sensor, creating multiple nearby clusters of pixels with distinct contours. Fig. 2 shows an example of the contours found in a DECO image with this algorithm.

2.1. Event types

There are three categories of charged particle events in the DECO dataset: tracks, worms, and spots. These are named according to the convention in [1], which categorizes events based on their morphology. Tracks are long, straight clusters of pixels in an image created by high-energy (GeV) minimum-ionizing cosmic rays. These are predominantly cosmic-ray muons at sea level and primary cosmic rays (mostly protons) above $\sim$20,000 ft altitude [15]. Worms are named for the curved clusters of pixels caused by the meandering paths of electrons that have undergone multiple Coulomb scattering interactions. These electrons are likely the result of local radioactivity. Worms can also be seen as two or more nearby, disconnected clusters of pixels, which are the result of an electron scattering in and out of the sensitive region of the camera sensor. Spots are smaller, approximately circular clusters of pixels that can be created by various interactions. They are likely predominantly caused by gamma rays that Compton scatter to produce a low energy electron that is quickly absorbed. Spots can also be produced by alpha particles, which also have a very short range in silicon, or by cosmic rays incident normal to the sensor plane. Fig. 3 shows the characteristic camera sensor response for each of the three interaction signatures detected by DECO. In addition to the three particle interaction categories, there are also events due to light in the sensor occurring when it is not sufficiently shielded, and several categories of noise: hot spots, thermal noise fluctuations, and large-scale sensor artifacts such as rows of bright pixels [2]. While non-particle events, shown in Fig. 4, are not partic-

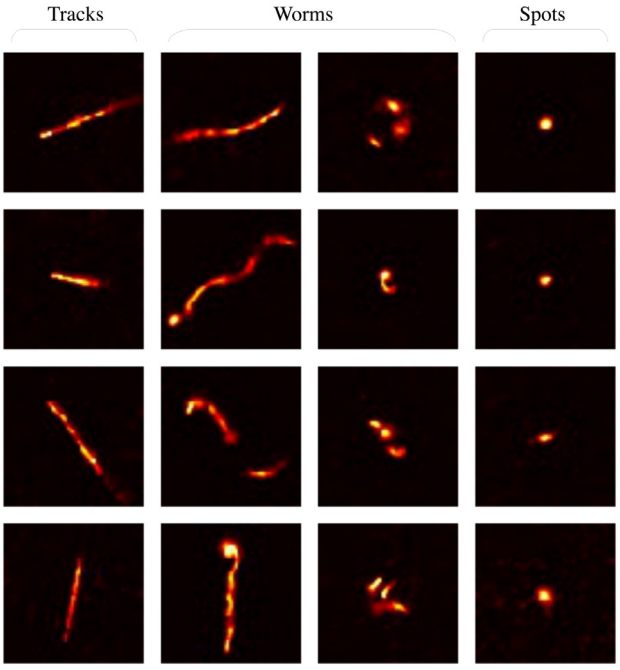

Fig. 3. Representative sample of the three distinct types of charged particle events that require classification. Tracks and spots, left and right columns, respectively, are generally observed to have consistent and predictable features. Worms, middle two columns, are observed to have a much wider variety of features, many of which present potential classification confusion when compared to track-like and spot-like features. Each image above has been converted to grayscale and cropped to 64×64 pixels.

ularly of interest from an analysis standpoint, they do cause potential classification confusion. It is worth noting that these event categories are motivated both by their morphologies and the potential physics analyses that would utilize different categories of events as signal or background. For example, efficient track identification (and worm rejection) is required to detect UHECRs using networks of smartphones or to perform cosmic-ray experiments in a classroom setting. Worms, on the other hand, would need to be identified in order to use DECO or a similar app as a radiation monitoring system.

2.2. Initial classification approach

Given the numerous event types, both particle and non-particle, and the increasing number of images being collected by DECO, there is a growing need for a reliable computerized event classification system. However, there are several challenges associated with characterizing the DECO dataset in a way that requires little human intervention. Due to the inhomogeneity in hardware[1] and data acquisition conditions, otherwise identical events may be detected differently, for example due to fluctuations in brightness, background noise, or number of pixels hit. Additionally, DECO particle events possess rotational and translational symmetry, which must be accounted for by classification algorithms.

An initial algorithm that classified DECO events used straight cuts applied to geometric metrics that were combined to make a binary classification: track or non-track. Clusters of pixels were identified using the marching squares algorithm described in Section 2. The binary classification identified low-noise images with a single cluster of pixels, not containing any sub-clusters (i.e. evidence of an electron scattering out of the sensor plane), with a minimum area of 10 pixels, and an eccentricity > 0.99, where eccentricity is calculated using image moments as described in [16]. The last two requirements were intended to select larger, line-like events, such as tracks. This method accurately distinguished tracks from spots, but struggled to separate tracks and worms, presumably due to their similar morphology. Many worms only curve slightly and have a high eccentricity. These events are unlikely to be high-energy muons due to their curvature, but the classification based on straight cuts could not distinguish them from tracks. Fortunately, advances in the quickly developing field of machine learning offer techniques to overcome these classification challenges.

3. Deep learning

3.1. Background

Deep learning is a subset of machine learning focused on building models that are capable of learning how to describe data at multiple levels of abstraction. This is achieved with a nested hierarchy of simple algorithms that when combined can form highly complex and diverse representations. At each layer of the nested hierarchy, a non-linear transformation of the previous layer's out-

[1] Users have run DECO on 604 distinct phone models to date.

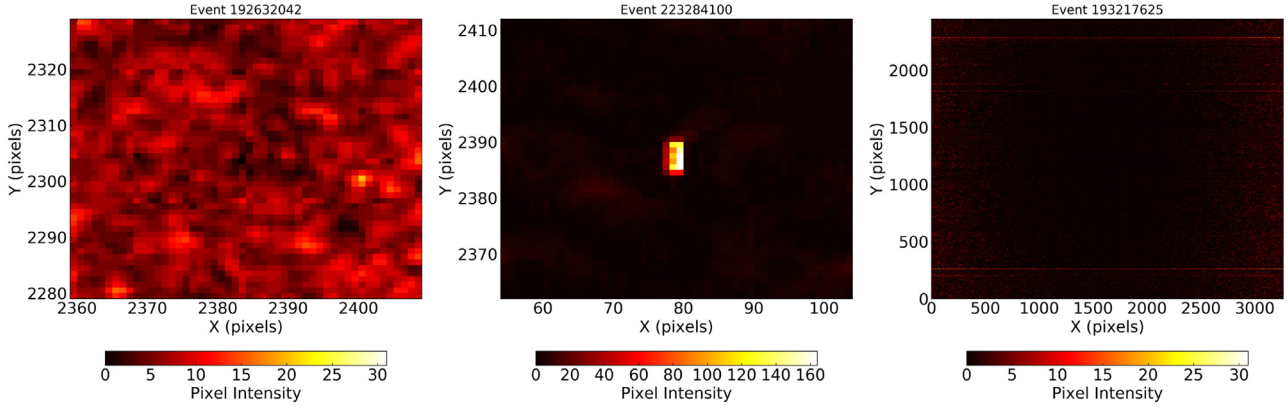

Fig. 4. Examples of non-particle (noise) events in the DECO dataset. Left: noise due to thermal fluctuations. Center: hot pixels, i.e., pixels that have regular, geometric shapes and typically repeat in the same location. Right: row of bright pixels, likely an artifact of the image sensor readout. The color scale represents the pixel intensity, scaled to the brightest pixel in each image, after a conversion to grayscale. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

put is typically performed, which results in the deeper layers of the model seeing a progressively more abstract representation of the original input. By learning features at multiple levels of abstraction, the model has the ability to learn complex mappings between the input and output directly from data [17]. This is particularly advantageous when dealing with higher-level abstractions that humans may not know how to explicitly describe in terms of the available input.

Deep learning models are typically constructed with four basic components in mind: (1) a specific dataset, (2) an objective function[2], i.e. the function that will be maximized or minimized, (3) the optimization procedure to be used on the objective function throughout the learning process, and (4) an appropriate structure for the model given the analysis goals and dataset characteristics. For our purposes, a particularly relevant and widely used example of such a model is the feedforward neural network, also known as the multilayer perceptron [18,19], which can be used to perform a number of tasks, including classification.

For classification, we begin by assuming that there exists some function, f^*, that describes the true mapping between input vector, $\boldsymbol{x}$, and category, y, such that $y = f^*(\boldsymbol{x})$. In this case, the goal of the feedforward neural network is to construct a mapping, $\boldsymbol{y} = f(\boldsymbol{x};\boldsymbol{\theta})$, then learn which value of the parameter vector, $\boldsymbol{\theta}$, provides the best approximation between f^* and f [20]. The categorical label, $\boldsymbol{y}$, is a unit vector containing all zeros, except for the index that corresponds to the yth category in the model, which has a value of 1. The function f is typically a series of nested functions, $f(\boldsymbol{x}) = f_n(f_{n-1}(\ldots f_2(f_1(\boldsymbol{x}))\ldots))$, with depth n, where f_1 corresponds to the input layer, f_2 through f_{n-1} are hidden layers[3], and f_n is the layer that provides the desired output (e.g. probabilities for input $\boldsymbol{x}$ belonging to each individual category in $\boldsymbol{y}$).

Each layer consists of a specified number of units, called neurons, that each compute a weighted linear combination of the inputs followed by a non-linear function which outputs a single, real-valued input for the next layer. Traditionally, layers have a dense, fully connected structure where the output of each neuron in a given layer is connected as input to all the neurons in the next layer. In this case, the output of the nth layer, $\boldsymbol{x}_n$, has the following vector representation:

$$\boldsymbol{x}_n = g(\boldsymbol{W}_n\boldsymbol{x}_{n-1} + \boldsymbol{b}_n), \tag{1}$$

where $\boldsymbol{x}_{n-1}$ is the output of the previous layer's neurons, $\boldsymbol{W}_n$ is a matrix of weights, $\boldsymbol{b}_n$ is a vector of biases, and g is the non-linear function, also known as the activation function. The weights and biases constitute the model's parameters, which are optimized during the learning process. Note that for the first layer in the model, $\boldsymbol{x}_{n-1} = \boldsymbol{x}_0$, which is simply the initial model input, $\boldsymbol{x}$. With the exception of the output layer, the typical choice for the activation function is the rectified linear unit, or ReLU [21], defined by $g(z) = \max(0, z)$, which outputs the maximum between the input and zero. A common variant is the leaky ReLU [22], where negative inputs are not set to zero, but are instead multiplied by a small constant α. In the output layer, the *softmax* function (multi-class generalization of the logistic sigmoid, see for example [20]) is used to produce a multinoulli distribution representing the probability that input $\boldsymbol{x}$ belongs to each of the K different categories represented in the model. The category with the greatest probability is generally taken to be the classification, however specific threshold cuts for each category can also be used.

During the learning process, the model is presented with a large number of training examples where each input, $\boldsymbol{x}$, has a single human assigned categorical label, y, which is taken by the model to be the ground truth. The ground truth label, y, is then typically represented in a conditional probability distribution, q, such that the conditional probability for the kth category in the model is given by $q(k|\boldsymbol{x}) = \delta_{ky}$, which is the Kronecker delta. A loss function is used to compute the error between the model predictions and the ground truth. Modern neural networks are typically trained using the principle of maximum likelihood. In this approach, the loss function is the negative log-likelihood, which can be equivalently described as the cross-entropy between the training examples and the modeled distributions [20]. In the case of multinomial logistic regression (i.e., classification with multiple categories), the cross-entropy loss function for a single training example is:

$$H(p, q) = -\sum_{k=1}^{K} q(k|\boldsymbol{x}) \log(p(k|\boldsymbol{x})), \tag{2}$$

where K is the total number of categories in the model, $q(k|\boldsymbol{x})$ is the ground truth, human-assigned probability for the kth category, and $p(k|\boldsymbol{x})$ is the probability output by the model for the kth category. The gradient of the loss, as a function of the weights and biases, is calculated using the back-propagation algorithm [23]. The loss is then minimized by updating the weights and biases for all the neurons in each layer using the method of mini-batch stochastic gradient descent (SGD) [24,25]. When using mini-batches, the gradient of the loss function is estimated as the average instantaneous gradient over a small group of training examples (25–100, typically), which serves to balance gradient stability with computing time. This procedure is then repeated, iterating through mini-batches of training examples, until the error between the modeled and ground truth distributions reaches a satisfactory level. A single cycle through all of the mini-batches contained in the training set is typically referred to as an epoch.

3.2. Convolutional neural networks

Convolutional neural networks (CNNs) [26] are a subclass of neural networks in which standard matrix multiplication is replaced with the convolution operation in at least one of the model's layers. CNNs have shown extraordinarily good performance learning features from datasets that are characterized by a known grid-like topology, such as pixels in an image or samples in a waveform. The core concept behind CNNs is to build many layers of "feature detectors" that take into account the topological and morphological structure of the input data [27]. Throughout the training process, the model learns how to extract meaningful features from the input, which can then be used to model the contents of the input data. The first stages of a CNN typically contain two types of alternating layers that are used to perform "feature extraction": convolutional layers and pooling layers.

Convolutional layers take a stack of inputs (e.g. color channels in an image) and convolve each with a set of learnable filters to produce a stack of output feature maps, where each feature map is simply a filtered version of the input data (input image, in our case). A given input image, I, convolved with a $n \times m$ filter, F, will produce an output according to:

$$X_{p,q} = (F * I)_{p,q} = \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{c} F_{i,j,k} \cdot I_{p+i,q+j,c}, \tag{3}$$

where $X_{p,q}$ is the (p, q) pixel of the feature map (prior to applying the non-linear function), n and m correspond to the filter's height and width in units of pixels, and c is the number of color channels

[2] In the case of minimization, the objective function is commonly referred to as the cost, loss, or error function.

[3] Intermediate layer outputs are always connected as inputs for other layers and are therefore never visible as network outputs, hence the term "hidden".

in the input image.[4] With this transformation in mind, a slightly modified version of Eq. (1) can be constructed such that the lth of L total feature maps output by the nth layer, $\boldsymbol{X}_n^{(l)}$, can be expressed with the following matrix representation:

$$\boldsymbol{X}_n^{(l)} = g\left(\sum_{k=1}^{K} \boldsymbol{W}_n^{(k,l)} * \boldsymbol{X}_{n-1}^{(k)} + b_n^{(l)}\right), \tag{4}$$

where $\boldsymbol{X}_{n-1}^{(k)}$ is the kth of K total feature maps output by the previous layer,[5] $\boldsymbol{W}_n^{(k,l)}$ is a set of matrices containing the weights for the learnable filters, $b_n^{(l)}$ is the bias for the lth feature map, $*$ is the two-dimensional convolution operation shown in Eq. (3), and g is the activation function that performs a non-linear transformation of each pixel to produce the resulting feature map.

Feature maps are essentially abstract representations of the input image, where each individual feature map is tasked with learning how to extract a specific feature from the input, such as edges, corners, contours, parts of objects, etc. It should be noted that the specific features learned by each feature map are not predetermined, but, rather, are selected solely by the model during the learning process. The feature maps nearest the input tend to resemble the original image. At layers further from the input, the feature maps gradually become more abstract and specialized.

Replacing the matrix product with a sum of convolutions results in a series of additional benefits [20]: (1) a restricted connectivity pattern where each neuron is only connected to a local subset of the input, which reduces the number of computations, (2) the model learns a single set of parameters for each filter that can then be shared via convolution by all pixels in the input, which reduces the number of model parameters and improves the model's generalization performance,[6] and (3) the form of parameter sharing used in convolution also results in translation equivariance, meaning a translation in the input results in the same translation in the output. The restricted connectivity pattern results in the model learning predominantly from only local interactions in the input, meaning that features at distant locations of the input are less likely to interact. To combat this, convolutional layers are often used alongside pooling (subsampling) layers.

Pooling layers [28] reduce the dimensionality of a feature map by using an aggregation function to compute a summary statistic across a small, local region of the input. The dimensional reduction gives the deeper layers of the model the ability to learn correlations between increasingly larger, yet lower resolution, regions of the input. For example, max pooling [29] computes the maximum output located within a rectangular region of the input, then reduces that rectangular region to a single value equal to the maximum. A common choice is to divide each feature map into non-overlapping 2×2 grids of pixels that are then each reduced to a single pixel, converting a feature map from, say, 32×32 pixels to 16×16 pixels. As a result, only the most pronounced features in each rectangular region are forwarded to the deeper layers of the model. The pooling operation also gives rise to translation invariance[7] across small regions of the input. This is a desirable benefit when one is primarily interested in whether certain features are present in the input, rather than knowing precisely where they are located.

Finally, the features extracted from convolutional and pooling layers are typically used as input for a standard, fully connected, feedforward neural network (as explained in Section 3.1) where the desired output is then produced, which, in this case, is the CNN classification of the input image.

4. Constructing a DECO CNN

In the sections that follow, we describe the construction and optimization of a DECO-specific convolutional neural network. We begin by introducing the dataset and the challenges associated with both human classification error and the small number of training images. We explain how data augmentation was used to make the model approximately invariant to rotations as well as artificially boost the number of training images. We then discuss the problem of overfitting and the techniques used to address it. Next, we summarize the model structure and training process used. Finally, we present the classification results, evaluate the performance of the model, and discuss the model's role in current and future DECO analyses.

4.1. Image database and human labels

As discussed in Section 3.1, the model must not only be presented with a large number of training examples, but also with a set of corresponding human-determined categorical labels. However, assigning human labels to large datasets is time consuming and, depending on the dataset, difficult to do accurately. Previous deep learning models within the astronomy and particle physics communities have constructed labeled datasets by using a crowd-sourcing approach, for example by Galaxy Zoo [30,31], or large-scale Monte Carlo event simulations, for example by the NOvA neutrino experiment [32,33]. Both approaches require considerable human labor. At present, the DECO image database contains $\sim$45,000 events (images that passed the online filter), each of which potentially contains one or more clusters. Assigning human labels to each event cluster would be a very time consuming task. With this in mind, rather than labeling the entire dataset, we instead opted for an iterative approach in which the number of labeled training examples was successively increased in parallel with the optimization of the CNN model structure.

To accomplish this, individual event clusters were inspected by eye, by multiple people, and assigned labels of track, spot, worm, noise[8], or ambiguous. Additionally, if a clear identification could not be made or if humans disagreed on the classification, which occurred $\sim$10% of the time, the image was labeled as ambiguous and excluded from the training set. During the optimization process, the model was trained and used to classify events that were not in the original training sample. These classified images were then searched by eye for likely false positives, i.e., instances where the model reports a high probability that an event belongs to a certain category but appears to be wrong. These incorrectly classified events were then assigned a correct human label, added to the existing set of training images, and used to train the next iteration of the model. This process was repeated on increasingly larger sets of images. As shown in Fig. 5, with each new iteration, the examples that the model found most difficult to categorize were added to the labeled dataset, thus addressing the remaining weaknesses in the classifier.

[4] In our application, we sum the three color channels R, G, and B to produce a single grayscale color channel.

[5] The input layer, $\boldsymbol{X}_{n-1}^{(k)} = \boldsymbol{X}_0$, isn't a feature map but is simply the input image for the model.

[6] Generalization performance is a model's ability to perform well on previously unseen examples that were not included in the training set.

[7] To be clear, f is translation *equivariant* if $f(T(x)) = T(f(x))$, and translation *invariant* if $f(T(x)) = f(x)$, where $T(x)$ is a translation operation.

[8] The noise category was added during the iterative training process when it was found to drastically improve the model's overall classification accuracy.

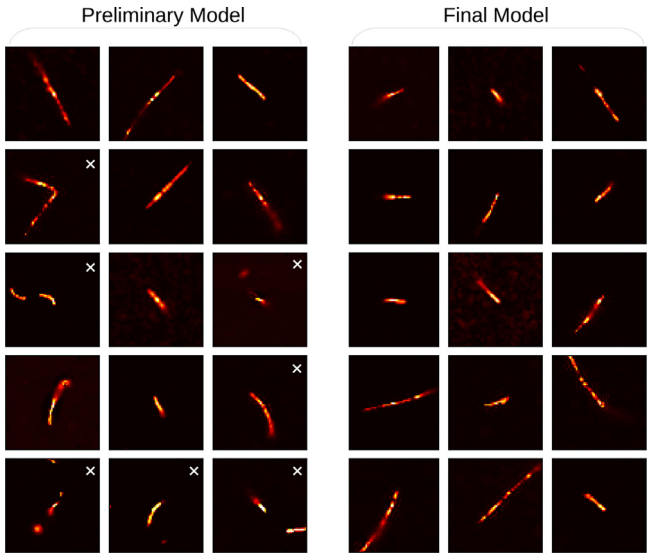

Fig. 5. Left: random sample of images with track probability > 0.95 according to a preliminary version of the CNN model (presented in [8]). This version of the model struggled to correctly identify tracks that had similar features to other event types, particularly worms. Incorrectly classified images, denoted with a white '×', were assigned a human label (worm, in each example shown) and added to the training set for the next iteration of the model. Right: random sample of images with track probability > 0.95 according to the final version of the CNN model. The CNN classification agrees with the human classification for every single event in this sample.

4.2. Preprocessing and data augmentation

Image-to-image variations in position, scale, and rotation pose a challenge to DECO event classification. When a DECO user collects data, both the position and orientation (at least in azimuth – zenith typically corresponds to phones operating flat on a table) of the phone is arbitrary. Both orientation and location data are collected in the app's metadata. However, the (x, y) position of a given event cluster within the camera sensor, as far as the model is concerned, should be considered a meaningless feature. Similarly, the orientation of a hit cluster within the (x, y) plane, as well as reasonable variations in scale (e.g. the length of a track) should also be considered meaningless by the model. Fortunately, CNNs naturally handle translations in the input quite well [34,35]. However, invariance to features such as scale and rotation need to be learned.

For a given input image, the apparent size of the event with respect to the camera sensor can be affected by a number of factors such as the underlying hardware in the specific phone model (including the image sensor resolution), the energy of the particle, and the angle of incidence. The pooling operation provides resiliency to minor changes in shape and scale [36], however, variations larger than a few pixels must be addressed by other means. Sophisticated solutions to this problem have been proposed [37], however, the simplest method is to introduce scale-jittering via data augmentation, which is in widespread practice today [38,39]. Data augmentation consists of randomly transforming training images while preserving their human-assigned category labels. Similar to scale invariance, data augmentation can also be used to learn rotation invariance. While rotation-invariant CNN architectures exist [31] and have been shown to outperform data augmentation in certain cases [40], the small number of training images in this study prohibited the use of such methods. Finally, due to the limited number of training images available, data augmentation was also used to artificially inflate the number of "unique" images seen by the model during training.

In general, data augmentation has been shown to be the simplest way to achieve approximate invariance to a given set of transformations [27]. Assuming the model has the capacity to do so (i.e., enough feature maps), the model should be able to learn a wide variety of invariances directly from the data [41]. An additional benefit of data augmentation is that a single set of transformations can be used to address multiple different issues. With that in mind, the following operations were applied to each training image:

- **Grayscale conversion and normalization**: a dimensional reduction over the channel axis of each image was performed by calculating an unweighted sum of each pixel's R+G+B value. The resulting grayscale images were then normalized to 1, taking the maximum possible R+G+B value to be 765 (i.e., 255×3). Grayscale reduces the variation seen from phone to phone and is also computationally more efficient. Furthermore, while color provides essential information for other image classification tasks, it does not for particle tracks.
- **Translation**: random left/right and up/down shifts, each by an integer number of pixels uniformly sampled between −8 and +8 with respect to the image center.
- **Rescale**: random zoom in/out uniformly sampled between 90% and 110% of the original image size, used for learning scaling invariance.
- **Reflection**: random horizontal and vertical reflections, each with a probability of 50%.
- **Rotation**: random rotation uniformly sampled between 0° and 360°; used for learning rotation invariance. After the rotation, any remaining pixels outside the boundaries of the original input were assigned a value of 0.
- **Crop**: crop from 100×100 pixels to 64×64 pixels; used to reduce the amount of empty space created on the boundaries of the image as a result of rotation, translation, and rescaling.

With the exception of normalization and the conversion to grayscale, which could be performed ahead of time, all data augmentation was done in real time during the training process. Prior to the start of each training epoch (full cycle through all training images, as defined in Section 3.1), a new random set of perturbations are applied to each image. Applying data augmentation in this way ensures that the model is never presented with the exact same version of a training example more than once. Real-time data augmentation is performed in Python using the Keras neural network application programming interface [42], which makes use of tools contained within the SciPy library [43].

4.3. Avoiding overfitting through regularization

Deep neural networks typically have anywhere from tens of thousands to tens of millions of trainable parameters. The advantage of such a large number of parameters is that the model has the ability to fit extremely complex and diverse datasets. However, the downside of a model with such tremendous freedom is that there is considerable risk of over-fitting, which occurs when the model simply memorizes the training images. As a result, the model is overly sensitive to the specific features that were memorized during training and therefore generalizes poorly to new data. Over-fitting is of particular concern when dealing with a small number of training images, as is the case in this study. To combat this phenomenon, we used several regularization techniques [20,44], which are modifications to the learning process that are intended to reduce generalization error while leaving training error[9] unaffected. These techniques are as follows:

[9] The error between the true and predicted classification for images in the training set.

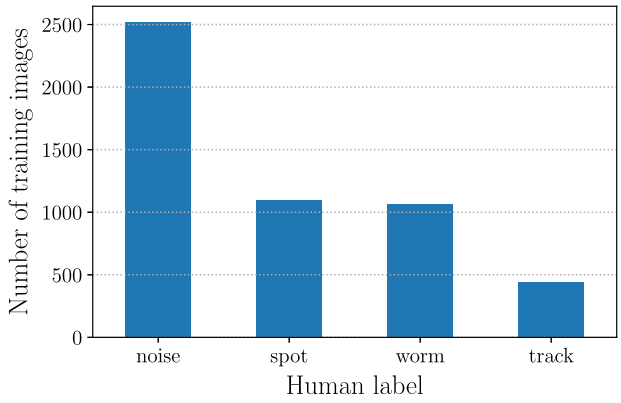

Fig. 6. Number of training images for each event type contained in the final dataset that was used to train the best performing model. Out of the 5119 total images, there are 2520 (49%) noise, 1094 (21%) spot, 1063 (21%) worm, and 442 (9%) track images.

- **Data augmentation**: artificially increasing the number of training examples by modifying the images in such a way that they look different for each particular training instance while still maintaining the correctness of the underlying human assigned label. The particular perturbations used are outline in Section 4.2.
- **Label smoothing**: accounting for the uncertainty in human assigned labels by replacing the hard 0, 1 (false, true) label distribution, $q(k|\boldsymbol{x}) = \delta_{ky}$, with $q(k|\boldsymbol{x}) = (1-\epsilon)\delta_{ky} + \frac{\epsilon}{K}$, where k is the kth of K total categories in the model, ϵ is a small constant representing the probability of an incorrect label, and y is the human label. This modification results in an additional penalty term being introduced into the loss function, Eq. (2). Assuming that ϵ is reasonably small, this technique reduces the effect of incorrect labels while still encouraging correct classification [45].
- **Dropout**: at every step of the training process, each individual neuron in a given layer has a probability, P, of being temporarily set to zero, or "dropped out" [46,47]. The purpose of dropout is to prevent the co-adaptation of neuron outputs such that each individual neuron depends less on other neurons being present in the network. To preserve the total scale of inputs, the neurons that weren't dropped out are rescaled by a factor of $1/(1-P)$. Dropout is only applied during training and turned off afterwards.
- **Max-norm constraint**: to prevent weights from blowing up, a max-norm constraint is applied to each neuron's weight vector, $\boldsymbol{W}$, such that $\|\boldsymbol{W}\| \leq r$, where $\|\cdot\|$ is the L^2 vector norm and r is a user specified constant dictating the maximum value. After each training step the constraint is checked and, when necessary, the weights are updated according to $\boldsymbol{W} \rightarrow \boldsymbol{W}\frac{r}{\|\boldsymbol{W}\|}$. The max–norm constraint, both with and without dropout, has been shown to help reduce over-fitting [47,48]. This constraint was applied to fully connected layers only.
- **Early stopping**: during the training process, testing loss (error) typically decreases, reaches a minimum value, and then begins to increase again once over-fitting has set in. To avoid using an overfit model, we capture running snapshots of the best version of the model during training, which correspond to the epochs where testing loss reaches a new minimum value [49,50].
- **Categorical weights**: As seen in Fig. 6, certain event types, tracks in particular, have fewer training images than others. As a result, the model sees more training examples from the abundant categories than the under-represented ones, which introduces bias into the classifier. To account for this imbalance, each category is assigned a weight, according to its abundance, which is applied to the loss function (Eq. (2)) to ensure that all categories are represented equally during optimization.

Table 1
Layer-by-layer summary of the best performing network. Each layer name is given followed by the number of feature maps (convolutional layers) or neurons (dense layers), the size of the convolutional filter or pooling region, the activation function used, and, lastly, the amount of dropout applied. For the leaky ReLU activation function, the value of α was set to 0.3 in all cases. A max-norm constraint of 3 was used for both 2048 dense (fully connected) layers. Dropout with a probability $P = .2$ was also applied to the input layer (not listed in the table).

	Layer	Features	Size	Activation	Dropout
1	Convolution	64	3×3	Leaky ReLU	–
2	Convolution	64	3×3	Leaky ReLU	–
3	Max pooling	–	2×2	–	0.2
4	Convolution	128	3×3	Leaky ReLU	–
5	Convolution	128	3×3	Leaky ReLU	–
6	Max pooling	–	2×2	–	0.2
7	Convolution	256	3×3	Leaky ReLU	–
8	Convolution	256	3×3	Leaky ReLU	–
9	Max pooling	–	2×2	–	0.2
10	Convolution	512	3×3	Leaky ReLU	–
11	Convolution	512	3×3	Leaky ReLU	–
12	Max pooling	–	2×2	–	0.2
9	Dense	2048	–	Leaky ReLU	0.4
10	Dense	2048	–	Leaky ReLU	0.4
11	Dense	4	–	softmax	–

4.4. Model structure and training

The best performing model trained in this study begins by taking a normalized, 100×100 grayscale image (zoomed in on the hit pixel cluster) as input. The input is then transformed via data augmentation (Section 4.2), cropped to 64×64, and subjected to dropout with a probability $P = 0.2$. Next, feature extraction is performed using four three-layer-deep blocks, each of which consists of the following operations: 3×3 convolution followed by a leaky ReLU activation, a second identical 3×3 convolution with leaky ReLU, and, lastly, 2×2 max pooling. For the leaky ReLU nonlinearity, a constant multiplier $\alpha = 0.3$ is applied for all negative inputs. Following max pooling in each block, dropout is applied with probability $P = 0.2$. For each of the four blocks, the number of feature maps is doubled, starting with 64 in the first block and ending with 512 in the last. The model structure is loosely based on the VGG-16 network [38], which used only 3×3 convolutional filters and 2×2 max-pooling throughout the network. Following feature extraction, the feature maps are flattened to a single, one-dimensional vector that is used as input for a three-layer fully connected network (Section 3.1). The first two layers are identical dense (fully connected) layers with 2048 neurons, leaky ReLU activation with $\alpha = 0.3$, and a max-norm constraint with $r = 3$ (see Section 4.3). Each dense layer is also followed by dropout with a probability $P = 0.4$. Finally, the output layer performs softmax regression, which outputs the probability for each of the 4 categories in the model (track, spot, worm, and noise). Fig. 7 shows a block diagram of the model structure and workflow. Specific details for each layer are summarized in Table 1.

To train the model, we used a variant of mini-batch SGD (see Section 3.2) known as Adadelta [51]. For our model, Adadelta was found to converge slightly faster than both SGD and Adam [52], another widely used variant of SGD. At the beginning of each training epoch, a new set of random data augmentation perturbations are applied to each image in the training set. The model was programmed in Python using the Keras neural network application programming interface [42] operating with a Theano [53] backend. The final model contains approximately 25 million trainable

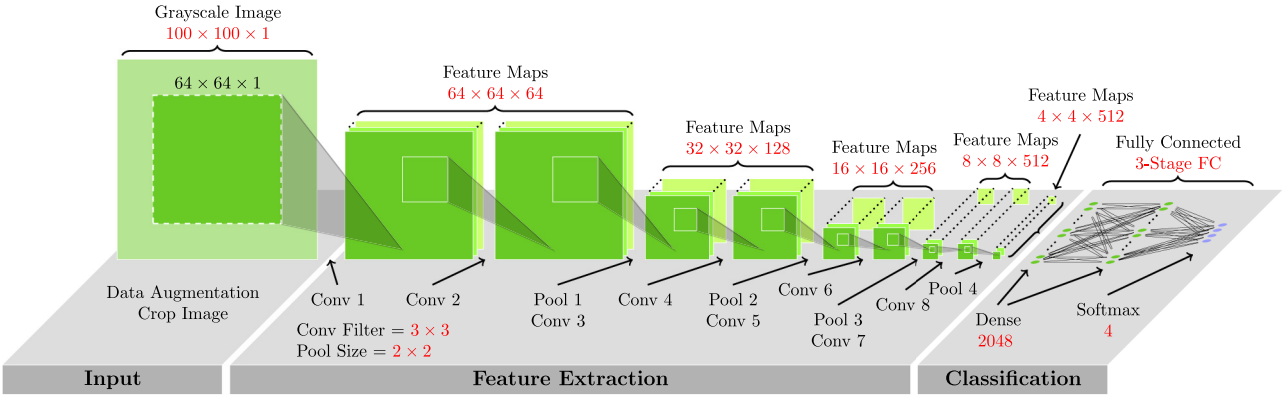

Fig. 7. Block diagram of the best performing network trained in this study. The input and output dimensions for each operation are shown to the left and right of the arrows, respectively. All convolutional filters are 3×3 and all pooling operations are 2×2 max pooling. Following the fourth pooling layer, the feature maps are flattened to a single 1-dimensional vector of length 8196, which is then used as input for the first dense layer.

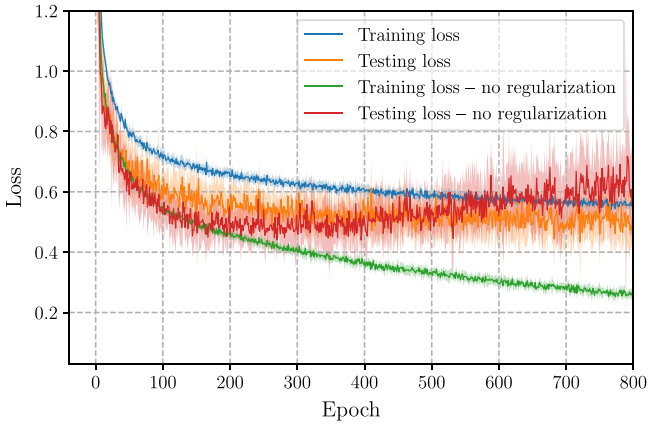

Fig. 8. Loss as a function of epoch for two different versions of the model, one trained with regularization techniques and one trained without. The loss is averaged over the 10-fold cross validation of the entire dataset and shaded error bands indicate the 1σ spread across the 10-folds. An epoch refers to one full cycle through all available training images.

parameters and was trained on a single NVIDIA Quadro M4000 graphics processing unit (GPU) with 8 GB of RAM.

5. Results and analysis

5.1. Model performance

To estimate the overall performance of the model, independent sets of human-classified images were evaluated using the method of stratified k-fold cross-validation [54]. In this procedure, the set of training images is split into k groups, where each group contains a roughly equal number of images from each of the categories represented in the model. k otherwise identical versions of the model are then trained, each time setting aside one group for testing and $k-1$ for training the model. Selecting a value of 10 for k, we trained each individual fold for a total of 800 epochs, where each epoch consists of a single cycle through the full set of training images. The loss (defined below) for both training and testing sets, averaged over the 10 folds as a function of training epoch, is shown in Fig. 8. The loss[10] for a set of examples is defined to be:

$$\mathcal{L} = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} q_n(k|\boldsymbol{x}) \log(p_n(k|\boldsymbol{x}))w_k, \tag{5}$$

where N is the number of training or testing images, $K = 4$ is the number of categories in the model, p and q are the respective CNN and human assigned categorical distributions for each image (defined in Section 3.1), and w is a categorical weight term to account for the categorical imbalance in the training set (see Section 4.2). Conceptually, the loss can be thought of as the average error between the human and CNN classifications.

Early stopping (Section 4.3) was used to obtain the best performing (lowest testing loss) versions of the model throughout each 800-epoch training session, which, on average, occurred near epoch 650. The training and testing loss as a function of training epoch can be seen in Fig. 8. The gap between the training and testing loss is caused by the regularization techniques used to prevent overfitting, which are only applied to the training set (see Section 4.3). Lower testing loss than training loss can also be indicative of an underfit model. To test this, an alternate version of the model was trained with dropout removed from all layers, the max-norm constraint removed from the fully-connected layers, and no label smoothing. The results of this test revealed that the gap between testing and training loss disappeared until overfitting set in at epoch $\sim$200. This explains the gap between training and testing loss and also confirms that the regularization techniques are effectively preventing the model from overfitting the data. To investigate the potential benefits of a longer training duration, an additional model was trained for 10,000 epochs. While training loss was observed to decrease slightly, no benefit was seen in the testing set, thus confirming that 800 epochs was sufficient. A value of $\epsilon = 0.004$ was used for label smoothing. Setting ϵ to 0 as well as using larger values of 0.1 and 0.01 all resulted in marginally higher testing loss. We also tested an alternate, simpler version of the model which is described in Section 5.3.

5.2. Model accuracy

Fig. 9 shows a category-by-category summary, known as a confusion matrix, quantifying the error between human and CNN classifications for each category in the model. Each square of this confusion matrix is calculated by averaging the testing set results over the 10 folds in the cross validation. It should be noted that the resulting distribution is not normalized and is biased according to the relative occurrence of each category in the training set. For example, noise events make up almost half of the training set (Fig. 6).

[10] Note that this is technically the logarithm of the loss and therefore is not expressed as a percentage.

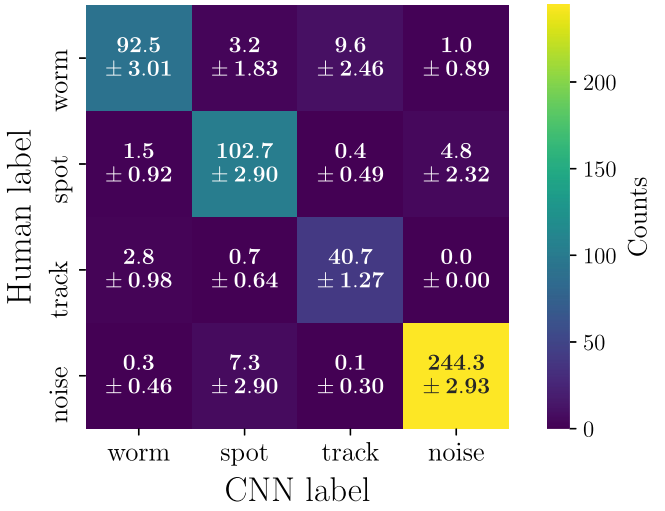

Fig. 9. Confusion matrix summarizing the CNN categorization accuracy. The vertical axis shows the ground truth (human-determined) classification and the horizontal axis shows the classification from the CNN. The values shown in the confusion matrix are the average and standard deviation of the testing set results from the 10-fold cross validation.

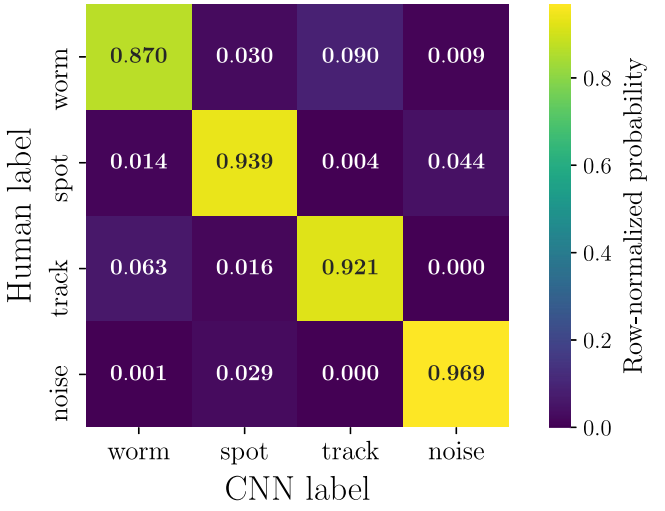

Fig. 10. Row-normalized confusion matrix that accounts for the relative imbalance in the number of testing examples for each category in the training set. Normalization is performed independently for each row and is calculated by dividing each row of the unnormalized confusion matrix (see Fig. 10) by the total number of events in that row.

This bias can be removed by normalizing each row of the confusion matrix to the total counts contained in each row, i.e. the total number of human-labeled events for each category. The resulting row-normalized confusion matrix describes the conditional CNN probability distributions for each of the four human-assigned labels in the model. The probability of the CNN correctly identifying each event type, along with the probability of mis-identifying each category, can be read directly off of the row-normalized confusion matrix in Fig. 10. For example, the model correctly identifies human-labeled tracks as tracks 92% of the time, while incorrectly identifying them as worms 9% of the time. This confusion in the classifier is both expected and comparable to human performance, given that, out of the four categories in the model, track and worm event morphologies are among the most similar. The model accurately labels noise events 97% of the time, which is the highest accuracy of any event type. This is also expected due to the vast differences between charged particle events and noise. Moreover,

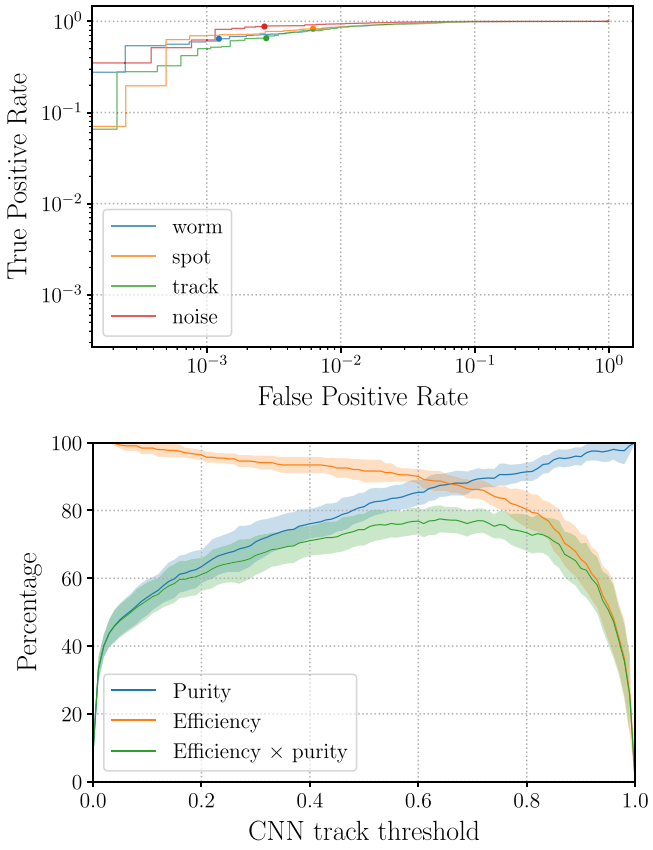

Fig. 11. (Top) Receiver operating characteristic (ROC) curve displaying the true positive rate vs. false positive rate for a variety of threshold values. A threshold of 0.9 is indicated with a dot for each category. (Bottom) Purity, efficiency, and their product as a function of CNN track probability threshold, averaged over the 10-fold cross validation. For each curve, the average and standard deviation are indicated by the thin solid line and corresponding band, respectively. For each threshold value, the purity and efficiency are calculated for events with a CNN track output, p_{track}, above the track threshold.

this also confirms that the model successfully learned the concept of noise, justifying the inclusion of this category in the model.

These results assume that a single classification is assigned to each image by choosing the category with the highest CNN output probability. We explore the performance of alternative choices below.

We further evaluate the model's classification performance by calculating the true and false positive rates for each category, assuming a binary classification scheme (e.g. track and non-track). The true and false positive rates for each category are parameterized according to a threshold applied to its CNN output probability and plotted as a receiver operating characteristic (ROC) curve, as seen in the top panel of Fig. 11. For example, requiring a track probability of at least 0.9 results in a true positive rate of 60% and a false positive rate of 0.3%. While the trade-off between efficiency and purity[11] can be inferred from the ROC curve, these quantities were also explicitly calculated for tracks, which is the primary category of interest for most DECO users. The resulting efficiency, purity, and efficiency × purity curves, averaged over the 10 folds and plotted as a function of track probability threshold, are shown in the bottom panel of Fig. 11. For a given fold and threshold, the efficiency is calculated from the testing set and defined to be the ratio of the number of tracks that pass the threshold to the to-

[11] The definitions of purity and efficiency used here are generally referred to as precision and recall, respectively, within the machine learning community.

tal number of tracks. Likewise, for a given fold and corresponding test set, the purity is defined as the ratio of the number of human-labeled tracks that pass the threshold to the total number of events, regardless of event type, that pass the threshold. The product of the resulting curves is one metric that can be used to determine a threshold value that balances the efficiency vs. purity trade-off.

5.3. Comparison with simpler model

In the previous sections, we have shown that the model exhibits excellent performance across all four categories when classifying unseen data. However, one might wonder if the complexity of our model, which contains 25 million trainable parameters, is necessary to achieve this level of performance. In order to test this, we trained a simpler version of the model, containing 140 thousand parameters, with the same efforts described in Sections 4.2 and 4.3. The simpler model contained only two blocks of convolutional and pooling layers, followed by significantly smaller dense layers than those described in Section 4.4. The performance of this model was evaluated using the same 10-fold cross-validation described in Section 5.1. Compared to our more complex model, the simple model was equally accurate when classifying spots and noise, but 17% less accurate at classifying worms and 7% less accurate at classifying tracks. Furthermore, when evaluating the track performance in a binary fashion (see Section 5.2), a 0.8 track threshold cut with the simple model resulted in a track sample with $<80\%$ purity and only 40% efficiency. This suggests that a more complex model is necessary in order to distinguish tracks and worms, which are the most interesting events scientifically.

5.4. Comparison with straight cuts

Early classification attempts, described in Section 2.2, sought to separate tracks from non-tracks in a binary fashion using straight cuts on simple metrics. This method, which used each image's area, number of clusters, and eccentricity, can be directly compared to the CNN model. To accomplish this, we treat the CNN output as a binary classification scheme (track or non-track) and evaluate both classification methods on the same set of testing images and corresponding human-assigned labels. The initial, straight-cuts model yielded a track selection with an efficiency of 69% and a purity of 37%. The low purity is likely due to small differences in the event topologies of many tracks and worms, which can be difficult to capture with simple geometric metrics. Moreover, optimization of the straight-cuts approach required aggressive cuts on these metrics, which also contributes to its poor efficiency in identifying tracks. The CNN classification, on the other hand, identifies tracks with 80% efficiency and 91% purity (cutting at a track probability threshold of 0.8, to be explained in Section 5.5), and can also accurately identify worms, spots, and noise with similar performance. Furthermore, the output probabilities of the CNN model enable us to design an event selection with a desired efficiency and/or purity in mind.

5.5. Application to full dataset

While the CNN model has a number of uses, providing real-time classifications for the events listed in the public DECO data browser [12] is perhaps the most important. For this purpose, we seek to maintain a high-purity set of events identified as tracks. After evaluating constant cut-off values of 0.7, 0.8, and 0.9 on the testing set, we opted for a probability threshold of 0.8, which yields an event selection with a track efficiency of 80% and, most importantly, a track purity of 91%. As a result of applying a threshold cut rather than the maximum-probability criterion, there are some events with probability below threshold for every single category, which are therefore assigned a label of "ambiguous". More aggressive threshold cuts result in more events being labeled "ambiguous".

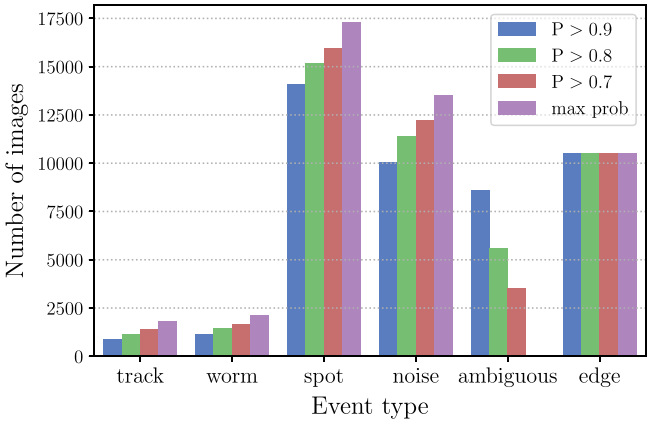

Fig. 12. Distribution of event types in the full data set (45,316 images) for different threshold choices applied to the CNN output probabilities. Each threshold is applied uniformly to all four categories and any event that does not have a probability greater than the threshold for any category is labeled "ambiguous". The fourth selection classifies events according to their maximum probability, which is why there are no ambiguous events in that scheme. "Edge" events are images with event clusters located less than 32 pixels from the camera sensor edge, which is incompatible with the CNN input requirement of 64×64 pixel images. The relatively high rate of edge-type images may be due to light leakage around the edges of the image sensor when DECO is run under sub-optimal data-taking conditions, such as in a well-lit room.

To investigate the effect of a given threshold choice on the full dataset we ran every event in the DECO database ($\sim$45,000 images) through the CNN model and used the resulting output probabilities to classify each event according to several different threshold choices. The resulting distributions for all event types, shown in Fig. 12, confirm that a threshold of 0.8 is indeed reasonable and results in ambiguous images $\sim$10% of the time, which is consistent with human categorization ambiguity (Section 4.1). With this in mind, the classification scheme based on a threshold of 0.8 was implemented in the public database, which can now be queried by event type as determined by the CNN [12].

Given the classification assigned to any event using this scheme, it is desirable to know the probability that the CNN classification is in fact correct for each event type. As an example, for tracks this corresponds to the conditional probability $P(H = track|CNN = track)$, where H is the human label and CNN is the CNN label. This probability depends on the relative rate of each event type in the data set, i.e., the prior probability that a given event belongs to a given category. The conditional probability could be calculated directly from the testing data sets used in the 10-fold cross validation, however, the distribution of event types in this set of images is biased in comparison to the full dataset. This is because the training set was intentionally enriched with tracks and worms; tracks are the most interesting events from an astrophysical perspective and worms are the primary source of confusion for tracks. Compared to the training set, the full data set has relatively fewer worms and tracks and more spots and noise events. Fortunately, this bias can be corrected by rescaling the testing set results. To accomplish this, we begin with the approximation that the CNN classifications for the full dataset are entirely correct, an approximation that is justified by the excellent performance of the CNN. We then use the abundance of each event type in the full dataset according to the CNN classification to determine the prior probability that an event belongs to a given category. Next, we apply a threshold cut of 0.8 to the testing set and construct a new

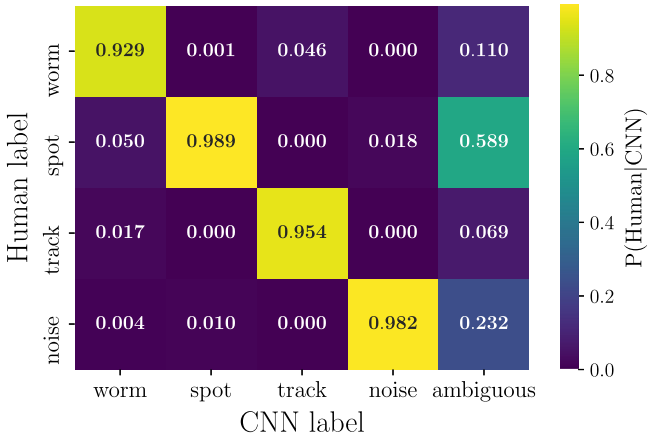

Fig. 13. Column-normalized confusion matrix re-weighted to account for the relative rate of each event type in the full data set. In order for an event to be classified as a particular category, the corresponding CNN probability must be > 0.8. Events that do not meet this threshold for any probability are classified as "ambiguous".

confusion matrix (similar to Fig. 9). We rescale each row of this confusion matrix by the ratio of the number of events for each event type in the full data set (Fig. 12 with a 0.8 threshold) to the number of each event type in the training set (Fig. 6). Finally, we rescale the confusion matrix column-wise in order to calculate the conditional probability, $P(H = i|CNN = j)$, for each category. By necessity, a 5th column for "ambiguous" events was added to the confusion matrix, which shows the distribution of events that don't meet any of the CNN threshold requirements. The resulting confusion matrix, shown in Fig. 13, suggests that all four event types in the full dataset are likely to be classified correctly $\geq$ 90% of the time. Most notably, we estimate that an event classified as a track by the CNN has a $\sim$95% probability of being a track according to human classification. Note that this quantity is the expected observable purity in the dataset, which differs from the 91% purity estimated on the training set that was described at the beginning of this section.

6. Conclusions and future work

We have described the development and validation of a convolutional neural network for the classification of images obtained by users running the DECO application. This new approach to image classification resulted in significant improvements over previous classification of DECO images using straight cuts. Event classification using the straight-cuts approach produced a track sample with 20% purity after applying the rescaling procedure described in Section 5.5. The CNN model, on the other hand, yields a data set with an estimated purity of 95% after rescaling to the full DECO data set. This classification algorithm has been integrated into the standard DECO processing pipeline and the resulting classification of each event is available along with the event's image and metadata on the public web site within several hours of detection. The CNN classification can be used in queries, allowing users to select a sample of images of any particle identity, or multiple identities, for analysis and outreach purposes.

In addition to improving the overall experience of DECO users, the new model opens the door for new and improved analyses. For example, the model provides efficient rejection of the radioactive background (i.e., worms), which is necessary to detect extensive air showers using DECO or a similar application. Additionally, the measurement of the depletion depth (i.e., sensitive region) of a phone's camera sensor requires a large, pure sample of cosmic-ray muon tracks. Without a robust method of identifying tracks, the analysis published in [7] was limited to a single phone. The new classification enables us to extend this analysis to multiple phones with a lower non-cosmic-ray background in the data set. Once the thickness of the depletion region is known for a particular phone model, it can be used to constrain the incident zenith angle of individual cosmic rays. Together with the azimuthal direction of the track within the sensor plane, this will enable reconstructing the direction of DECO tracks. Constraining the direction of detected muons would improve the sensitivity of a multi-phone coincidence analysis, since the direction of muons from the same extensive air shower should be correlated. Measuring the direction of events could also enable measurement of the East-West effect.

One shortcoming of the analysis presented in this paper is the human labeling method of assembling a sample of training images. There is an inherent bias in the model due to potentially mis-labeled images in the training sample. Although the efforts described in Section 4.4 should mitigate some of this bias, further work could quantify it. Beam line data from a particle physics accelerator and data collected from running DECO with radioactive sources would yield unbiased samples of tracks and worms, respectively, to further evaluate the performance of the model. Additionally, coincidence experiments with DECO and scintillators could provide a similar data set of tagged cosmic-ray tracks, though with far lower statistics.

While the model was developed exclusively using images in the Android DECO data set, we expect it to generalize to similar data sets with minimal changes. DECO for iOS, which is currently in development, will have a data set consisting of images created by the same charged-particle interactions discussed here. Although the overall camera response will differ from Android phones, the resulting event types are expected to be the same. It is worth emphasizing that the Android data set consists of images from hundreds of different phone models, with wide variation in camera sensor response to DECO events. The data augmentation applied during training (Section 4.2) mitigates the effects of model-to-model variation by building invariances into the classification that should enable it to generalize to the iOS data set. It is also possible that including the phone model as a feature in the neural network could help further reduce the effects of model-to-model variation. The excellent performance of our CNN in identifying particle types in the DECO data set indicates that the same approach would be powerful for identifying particles detected by other projects that use distributed camera sensors. Finally, our approach (and perhaps our particular model architecture) could be well suited for other experiments (such as the DAMIC [55] dark matter project) that use CCD and CMOS sensors for particle detection.

Acknowledgements

DECO is supported by the American Physical Society, the Knight Foundation, the Simon Strauss Foundation, QuarkNet, and by National Science Foundation Grant #1707945. We are grateful for beta testing, software development, and valuable conversations with Colin Adams, Raaha Azfar, Keith Bechtol, Segev BenZvi, Andy Biewer, Paul Brink, Patricia Burchat, Duncan Carlsmith, Alex Drlica-Wagner, Mike Duvernois, Brett Fisher, Lucy Fortson, Stefan Funk, Mandeep Gill, Laura Gladstone, Giorgio Gratta, Jim Haugen, Kenny Jensen, Kyle Jero, Peter Karn, David Kirkby, Matthew Plewa, David Saltzberg, Marcos Santander, Delia Tosi, and Ian Wisher. We would also like to thank Ilhan Bok, Adrian Cisneros, Alex Diebold, Tyler Dolan, Blake Gallay, Emmanuelle Hannibal, Heather Levi, and Owen Roszkowski for their contributions to the DECO project through our QuarkNet DECO high school internship program.

References

[1] D. Groom, Cosmic rays and other nonsense in astronomical CCD imagers, Exp. Astron. 14 (1) (2002) 45–55, doi:10.1023/A:1026196806990.

[2] J. Vandenbroucke, S. Bravo, P. Karn, M. Meehan, M. Plewa, T. Ruggles, D. Schultz, J. Peacock, A.L. Simons, Detecting particles with cell phones: the distributed electronic cosmic-ray observatory, PoS ICRC2015 (2016) 691.

[3] D. Whiteson, M. Mulhearn, C. Shimmin, K. Cranmer, K. Brodie, D. Burns, Searching for ultra-high energy cosmic rays with smartphones, Astropart. Phys. 79 (2016) 1–9, doi:10.1016/j.astropartphys.2016.02.002.

[4] M. Unger, G. Farrar, Feasability of studying ultra-high-energy cosmic rays with smartphones, 2015. arXiv:1505.04777v1.

[5] J.J. Cogliati, K.W. Derr, J. Wharton, Using CMOS sensors in a cellphone for gamma detection and classification, 2014. arXiv:1401.0766v1.

[6] P. Homola, et al., Search for extensive photon cascades with the cosmic-ray extremely distributed observatory, in: Photon 2017: International Conference on the Structure and the Interactions of the Photon and 22th International Workshop on Photon-Photon Collisions and the International Workshop on High Energy Photon Colliders CERN, Geneva, Switzerland, May 22–26, 2017, 2018. arXiv:1804.05614.

[7] J. Vandenbroucke, S. BenZvi, S. Bravo, K. Jensen, P. Karn, M. Meehan, J. Peacock, M. Plewa, T. Ruggles, M. Santander, D. Schultz, A. Simons, D. Tosi, Measurement of cosmic-ray muons with the distributed electronic cosmic-ray observatory, a network of smartphones, J. Instrum. 11 (04) (2016) P04019.

[8] M. Meehan, S. Bravo, F. Campos, J. Peacock, T. Ruggles, C. Schneider, A.L. Simons, J. Vandenbroucke, M. Winter, The particle detector in your pocket: the distributed electronic cosmic-ray observatory, in: Proceedings, 35th International Cosmic Ray Conference (ICRC 2017): Bexco, Busan, Korea, July 12–20, 2017, 2017. arXiv:1708.01281.

[9] M. Borisyak, M. Usvyatsov, M. Mulhearn, C. Shimmin, A. Ustyuzhanin, Muon trigger for mobile phones, J. Phys. Conf. Ser. 898 (3) (2017) 032048, doi:10.1088/1742-6596/898/3/032048.

[10] M. Ackermann, et al., The fermi large area telescope on orbit: event classification, instrument response functions, and calibration, Astrophys. J. Suppl. Ser. 203 (1) (2012) 4.

[11] The CMS Collaboration, The CMS experiment at the CERN LHC, J. Instrum. 3 (08) (2008) S08004.

[12] https://wipac.wisc.edu/deco.

[13] W.E. Lorensen, H.E. Cline, Marching cubes: a high resolution 3d surface construction algorithm, Comput. Graph 21 (4) (1987) 163–169.

[14] S. van der Walt, J.L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J.D. Warner, N. Yager, E. Gouillart, T. Yu, The Scikit-Image Contributors, scikit-image: image processing in Python, PeerJ 2 (2014) e453, doi:10.7717/peerj.453.

[15] C. Patrignani, et al., Review of particle physics, Chin. Phys. C40 (10) (2016) 100001, doi:10.1088/1674-1137/40/10/100001.

[16] Y.D. Khan, S.A. Khanand, F. Ahmad, S. Islam, Iris recognition using image moments and k-means algorithm, Sci. World J. 2014 (2014) 9.

[17] Y. Bengio, Learning deep architectures for AI, Found. Trends Mach. Learn. 2 (1) (2009) 1–127, doi:10.1561/2200000006.

[18] F. Rosenblatt, Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms, Spartan Books, Washington, 1962.

[19] R.D. Reed, R.J. Marks, Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks, MIT Press, Cambridge, MA, USA, 1998.

[20] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016. http://www.deeplearningbook.org.

[21] V. Nair, G.E. Hinton, Rectified linear units improve restricted Boltzmann machines, in: Proceedings of the 27th International Conference on International Conference on Machine Learning, in: ICML'10, Omnipress, USA, 2010, pp. 807–814.

[22] A.L. Maas, A.Y. Hannun, A.Y. Ng, Rectifier nonlinearities improve neural network acoustic models, ICML Workshop on Deep Learning for Audio, Speech and Language Processing, 2013.

[23] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1, MIT Press, Cambridge, MA, USA, 1986, pp. 318–362.

[24] Y. LeCun, L. Bottou, G.B. Orr, K.-R. Müller, Efficient backprop, in: Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop, Springer-Verlag, London, UK, UK, 1998, pp. 9–50.

[25] L. Bottou, F.E. Curtis, J. Nocedal, Optimization methods for large-scale machine 935 learning, 2016. arXiv:1606.04838v3.

[26] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2323, doi:10.1109/5.726791.

[27] P.Y. Simard, D. Steinkraus, J.C. Platt, Best practices for convolutional neural networks applied to visual document analysis, in: Proceedings of the Seventh International Conference on Document Analysis and Recognition - Volume 2, in: ICDAR '03, IEEE Computer Society, Washington, DC, USA, 2003. 958–.

[28] Y.L. Boureau, J. Ponce, Y. Lecun, A theoretical analysis of feature pooling in visual recognition, in: ICML 2010 - Proceedings, 27th International Conference on Machine Learning, 2010, pp. 111–118.

[29] Y.T. Zhou, R. Chellappa, Computation of optical flow using a neural network, in: IEEE 1988 International Conference on Neural Networks, 1988, pp. 71–78 vol.2, doi:10.1109/ICNN.1988.23914.

[30] K.W. Willett, C.J. Lintott, S.P. Bamford, K.L. Masters, B.D. Simmons, K.R.V. Casteels, E.M. Edmondson, L.F. Fortson, S. Kaviraj, W.C. Keel, T. Melvin, R.C. Nichol, M.J. Raddick, K. Schawinski, R.J. Simpson, R.A. Skibba, A.M. Smith, D. Thomas, Galaxy Zoo 2: detailed morphological classifications for 304 122 galaxies from the Sloan Digital Sky Survey, MNRAS 435 (2013) 2835–2860. arXiv:1308.3496, doi:10.1093/mnras/stt1458.

[31] S. Dieleman, K.W. Willett, J. Dambre, Rotation-invariant convolutional neural networks for galaxy morphology prediction, MNRAS 450 (2015) 1441–1459. arXiv:1503.07077, 10.1093/mnras/stv632.

[32] D.S. Ayres, et al., The NOvA Technical Design Report, 2007, doi:10.2172/935497.

[33] A. Aurisano, A. Radovic, D. Rocco, A. Himmel, M.D. Messier, E. Niner, G. Pawloski, F. Psihas, A. Sousa, P. Vahle, A convolutional neural network neutrino event classifier, J. Instrum. 11 (2016) P09001. arXiv:1308.3496, 10.1088/1748-0221/11/09/P09001.

[34] Y. LeCun, Y. Bengio, The Handbook of Brain Theory and Neural Networks, MIT Press, Cambridge, MA, USA, 1998, pp. 255–258.

[35] Y. Gong, L. Wang, R. Guo, S. Lazebnik, Multi-scale Orderless Pooling of Deep Convolutional Activation Features, 2014. arXiv:1403.1840v3.

[36] D. Scherer, A. Müller, S. Behnke, Evaluation of pooling operations in convolutional architectures for object recognition, in: Proceedings of the 20th International Conference on Artificial Neural Networks: Part III, in: ICANN'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 92–101.

[37] Y. Xu, T. Xiao, J. Zhang, K. Yang, Z. Zhang, Scale-invariant convolutional neural networks, 2014. arXiv:1411.6369v1.

[38] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014. arXiv:1409.1556v6.

[39] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, in: NIPS'12, Curran Associates Inc., USA, 2012, pp. 1097–1105.

[40] D. Marcos, M. Volpi, D. Tuia, Learning rotation invariant convolutional filters for texture classification, 2016. arXiv:1604.06720v2.

[41] K. Lenc, A. Vedaldi, Understanding image representations by measuring their equivariance and equivalence, 2014. arXiv:1411.5908v2.

[42] F. Chollet, et al., Keras, 2015. (https://github.com/fchollet/keras)

[43] E. Jones, E. Oliphant, P. Peterson, et al., SciPy: Open Source Scientific Tools for Python, 2001-, http://www.scipy.org/ [Online; accessed 2018-08-20].

[44] J. Kukačka, V. Golkov, D. Cremers, Regularization for deep learning: a taxonomy, 2017. arXiv:1710.10686v1.

[45] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, 2015. arXiv:1512.00567.

[46] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R.R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors, 2012. arXiv:1207.0580.

[47] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (2014) 1929–1958.

[48] N. Srebro, A. Shraibman, Rank, trace-norm and max-norm, in: Proceedings of the 18th Annual Conference on Learning Theory, in: COLT'05, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 545–560, doi:10.1007/11503415_37.

[49] C.M. Bishop, Regularization and complexity control in feed-forward networks, in: F. Fougelman-Soulie, P. Gallinari (Eds.), Proceedings International Conference on Artificial Neural Networks ICANN'95, 1, 1995, pp. 141–148.

[50] J. Sjöberg, L. Ljung, Overtraining, regularization, and searching for minimum in neural networks, IFAC Proceedings Volumes 25 (14) (1992) 73–78. 4th IFAC Symposium on Adaptive Systems in Control and Signal Processing 1992, Grenoble, France, 1–3 July. doi:10.1016/S1474-6670(17)50715-6.

[51] M.D. Zeiler, ADADELTA: an adaptive learning rate method, 2012. arXiv:1212.5701v1.

[52] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, 2014. arXiv:1412.6980v9.

[53] Theano Development Team, Theano: a Python framework for fast computation of mathematical expressions arXiv:1605.02688v1.

[54] R. Kohavi, A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection, Morgan Kaufmann, 1995, pp. 1137–1143.

[55] A.E. Chavarria, et al., Damic at snolab, Phys. Procedia 61 (2015) 21–33. 13th International Conference on Topics in Astroparticle and Underground Physics, TAUP 2013. doi:10.1016/j.phpro.2014.12.006.