Convolutional K-Nearest Neighbors: An End-to-End Hybrid Approach

***Abstract*—This paper presents the use of a hybrid end-to-end approach to replace the fully connected layers of a Convolutional Neural Network (CNN) model, and train the hybrid model using the Neighborhood Component Analysis loss function. This hybrid model was trained using various standard Machine Learning data sets such as MNIST Digits, to evaluate the performance against standard CNN models. Experiments were also conducted by varying the size of the data sets and using different batch sizes. Our results and experiments have shown that our hybrid models are more robust to data set sizes and the convolutional layers alone can perform just as well as standard CNN models. Our approach takes a step in producing more explainable hybrid Deep Neural Network models.**

***Index Terms*—Convolutional Neural Networks, K-Nearest Neighbors, Explainable Deep Learning, Neighborhood Component Analysis, Hybrid Methods,**

I. Introduction

A growing problem in Artificial Intelligence (AI) and Machine Learning (ML) is the inability to explain or fully understand the reasons by which state-of-the-art ML algorithms perform as well as they do [1]. We constantly see AI/ML researchers attempting to outperform these algorithms but at the cost of adding further complexity to the models. In the past decade, we have seen the rise of Deep Neural Networks (DNNs) and the incredible feats they have accomplished in the sub-fields of AI such as Computer Vision (CV) and Natural Language Processing (NLP). DNNs are considered black-box or opaque methods due to their complex interactions and non-linear activations. This is also what allows DNNs to have the flexibility to learn complex functions that classical ML algorithms struggle to learn [1]. However, interpreting and explaining their learned function and results is a big problem that AI researchers still struggle to accomplish.

Convolutional Neural Networks (CNNs) have helped increase the popularity of CV and have accelerated methods such as object detection and classification in remarkable ways. They consist of convolutional layers, that mimic the visual cortex, to learn feature embeddings from images. CNNs are usually paired with fully connected or dense layers to learn a metric function using the features embeddings from the images.

Methods such as Grad-CAM aim to provide a visualization technique of explainability for the convolutional layers of the CNN; we are still left trying to understand the dense layers. This is done by calculating heat maps of the activations in the convolutional layer which highlight the important regions the CNN looks at to make its final prediction. Though this provides a visualization technique for the explainability for the convolutional layers of a CNN, we are still left with the opaque portion of the dense layers.

Methods such as model simplification have been proposed to provide explainability in dense models. But techniques such as these increase in complexity as the number of layers increase [1]. We see models such as AlexNet, VGG16, and VGG19 perform well in various settings. But most of the trainable parameters in these models are located in the dense layers of the model and simplifying these layers could result in decreased performance.

Hybrid methods pairing opaque models with transparent models have increasingly become a topic of research. Methods such as these usually pair a trained CNN with a transparent model such as a decision tree or K-Nearest Neighbors (KNN) classifier and thus are necessarily done post-hoc. Though these methods add explainability to CNNs, the use of the dense layers are still present during training time and still influence the black-box nature of the model.

In this paper, we propose a hybrid end-to-end approach that omits the use of the fully connected layers of a CNN and train the convolutional layers alone with a KNN inspired Neighborhood Component Analysis (NCA) loss function. Furthermore during test time, we use a KNN classifier in place of the fully connected layers. We propose that our models perform just as well with the convolutional layers alone and therefore with less trainable parameters. We also propose our hybrid model is robust to smaller data sets and less prone to overfitting. Our goal is to remove a portion of the black-box nature of CNNs by eliminating the dependency of the dense layers for classification.

The remainder of this paper is outlined a follows. In Section II we discuss related work on the use of hybrid methods and the use of the NCA loss function. Section III, we explain our use of the NCA loss and also formulate and explain a few research questions to explore. In Section IV, we give descriptions of the model architectures and data sets used, as well as a detailed explanation of our experiments. Section V showcases results from the experiments. In Section VI, we answer the research questions brought up in Section II, using results from the experiments. Lastly, we conclude this paper in Section VII and give some insight into future work we would like to explore.

II. Related Work

For a model to be explainable, it needs to have two characteristics; ***explainability*** and ***interpretability***. [1] were the first to formulate a proper definition of explainable models.

Explainability is the details and reasons a model gives to make its functioning understandable to a certain audience. Interpretability is the level at which a given model makes sense for a human observer. Interpretability, also known as ***transparency***, is a passive characteristic which comes from the design of the model itself. Explainability is an active characteristic which is done post-hoc.

Though CNNs cannot have interpretability by design, we seek the use of hybrid methods that pair transparent models with opaque models to add explainability. In our case, we pair CNNs with a KNN classifier by omitting the fully connected layers. Most hybrid methods are implemented post training, we seek to train the convolutional layers with the KNN selection rule to further aid the performance of the KNN classifier during inference time. One problem with training a CNN with a KNN classifier is that the KNN selection rule is not differentiable. Thus we seek a loss function that is similar to the KNN selection rule to be able to train our hybrid models.

Neighborhood Component Analysis (NCA) is usually an unsupervised dimensionality reduction technique for data sets with high complexity. [3]'s method learns a quadratic distance metric which optimizes the Leave-One-Out classification error on the training set. [3] introduce a differentiable cost function based on stochastic neighbor assignments in a transformed space that seeks to maximize the expected number of points correctly classified.

Just like [3], [15] propose a differentiable relaxation of the KNN selection rule by interpreting the KNN rule as the limit distribution of k categorical distributions. [15] then replace the one-hot encoded label vector in stochastic nearest neighbors with their continuous expectations. This yields a continuous and deterministic relaxation that converges to the hard KNN selection rule.

[16] took the NCA algorithm further by adopting it as a supervised loss function for their CNNs by taking advantage that the NCA loss is a special case of the Contrastive loss function. They seek to train their CNNs with the NCA loss in order to learn feature embeddings that perform well for the KNN. However, [16] did not omit the use of the fully connected layers and trained the entire CNN with the NCA loss function. Furthermore, they evaluate their model by taking the results of the output layer of the CNN and passing it to a K-Nearest Neighbor model. Their method was able to out perform their baseline models trained using the Cross Entropy loss function.

[9] takes a post-hoc approach to improve the explainability of DNNs by replacing the output layer of a DNN with a KNN classifier to produce both a predicted label and an explanation to its prediction. The explainability of the new hybrid DNN+KNN comes from the inherent transparency of the KNN and the simplicity of the nearest neighbor assumption.

[14] takes on a post-hoc approach as well, by evaluating a trained DNN with a KNN classifier at each layer. They identify patterns in the output of each layer and compare them to those found during training to ensure that the prediction made is supported by the training data.

III. Background

A. NCA Loss

Distance Matrix: We begin by transforming a batch of N input samples into a feature space, by passing them through the convolutional layers of the CNN. As a result of the flattening layer, an $N \times M$ matrix is formed where M is the number of features calculated by the CNN. This feature matrix is then used to calculate the pairwise Euclidean distance between each feature vector in the batch to produce a distance matrix D.

Probability Matrix: Following [3], a softmax over the distance matrix D is used to produce an $N \times N$ probability matrix P, where each row represents the probability p_{ij} such that, sample X_i selects sample X_j as a nearest neighbor and thus having a similar class label. The actual Leave-One-Out classification error of the KNN and NCA comes from setting the diagonals p_{ii} of the probability matrix P to zero, i.e. removing the probability that sample X_i selects itself as a nearest neighbor [3].

Bit Matrix: With the construction of the probability matrix P, we then proceed to produce a bit matrix B. We begin by one-hot encoding the class labels for each sample in the batch to produce an $N \times C$ matrix L, where N is the total number of samples in the batch and C is the number of class labels in the data set. We take on a one vs all approach for multi-class classification. To calculate the bit matrix B, we simply do a matrix multiplication with L and its transpose ($B = LL^T$), where each row in matrix B represent a bit vector for sample X_i such that a bit is turned on if X_i's class label is equal to X_j's class label, i.e. $L_i \cdot L_j = 1$.

Loss Calculation:

$$\mathcal{L}_{NCA} = 1 - \frac{1}{N} \sum_{i,j}^{N} p_{ij} * b_{ij} \tag{1}$$

Having calculated the probability matrix P and the bit matrix B, we have the necessary tools to calculate the NCA Loss as seen in equation (1). In optimizing the NCA loss function, we maximize the probability p_i denoting the probability that sample X_i is correctly classified [3].

As mentioned before, we omit the use of the dense layers in a CNN and only train the convolutional layers. This approach in training just the convolutional layers is inspired by [6] in which their models lack the use of fully connected layers and [10] in which they instead add fully connected convolutional layers in between convolutional layers. In contrast, we propose a simple method that uses standard convolutional, batch normalization, dropout, or pooling layers.

B. Research Questions

In this section we explain a few research questions we think are important to explore.

1) ***Can we eliminate the fully connected (dense) layers?*** Fully connected layers add complexity to the model. These layers increase the performance of DNNs but at the cost of lower model interpretability [1]. To explore this question, we cut out the fully connected layers of

CNNs and feed the features calculated by the convolutional layers directly to the NCA loss function during the training process. During the inference process, we feed the new sample into the convolutional layers and feed the outputting features from the CNN to a KNN classifier. The KNN classifier uses features learned from the training set to predict a label for the testing sample. We look to see if this hybrid approach performs as well as a CNN with dense layers.

2) ***How much learning can we do in the convolutional layers?***
Convolutional layers learn kernels that are specific to the prediction problem at hand. These kernels are used to map an image to a feature space by producing feature maps, which provide insight into the internal representation for an image. Learning is typically over for a CNN when the loss function used is optimized. Since standard CNNs consist of convolutional (feature learning) and dense (metric learning) layers, parameters in both areas aid in the learning of the model. We seek to explore the amount of learning convolutional layers can achieve by training the convolutional layers alone. We then will train a standard CNN counterpart model that consist of the same convolutional layers but with added dense and output layer. We will use this as our baseline to our performance and compare the result of our hybrid models.
3) ***How does the training data set size affect learning?***
A rule of thumb in neural-based ML is to use large data sets. This allows for the ML model to train its parameters to perform well to the classification problem. Typical classification image data sets consist of 50,000+ training images. We explore this problem by training hybrid and standard CNN models using subsets of the MNIST Digits and Fashion data sets to learn a binary classification problem. We also take it a step further by using the Horses or Humans data set and compare the result to the standard CNN model.
4) ***Can we use non-linear activation functions in the convolutional layers to increase learning?***
Activation functions are regarded as essential hyperparameters for CNNs. They drastically influence the overall performance of models by handling linear and non-linear data classifications alike [20] [12]. Determining the appropriate activation function for a given model is a trial and error process. To answer this question, we can examine comparisons of popular non-linear functions and their influence on network learning. We make these comparisons by utilizing a high-performing deep CNN and swapping the activation functions. We will implement a non-bias dense layer to accurately measure convolutional-learning.

IV. Methodology

A. Architecture

Our objective is to remove the dependency of the densely connected layers in CNNs. Architectures of our hybrid Conv-KNN models can be seen in Tables I and II. Tables III and IV show the baseline architectures we compared our hybrid models against. These architectures were chosen from [4] and [5] whom were able to create models that performed well on the data sets used in our experiments.

B. Data Sets

The data sets used are shown in Table V along with their respective training and testing size. We chose these data sets to test our method on simple and standard problems.

Modified National Institute of Standards and Technology (MNIST) Digits is a dataset of handwritten digits with training set size of 60,000 images and a testing set size of 10,000 images [8]. The MNIST Digits dataset is a fairly simple benchmark that has become a standard in the ML community to validate models.

The MNIST Fashion dataset increases the difficulty by replacing the images of hand written digits with images of clothing items [19]. MNIST Fashion aims to be a data set that represents a more modern CV task [19].

The Street View House Numbers (SVHN) is a another dataset in the style of MNIST Digits, but it presents a significantly more difficult and real-world problem by using snippets of house numbers from Google Street View images [13]. SVHN is a larger dataset than MNIST Digits and Fashion and the images are in color rather than greyscale.

Horses or Humans is a small data set consisting of 500 rendered images of various species of horses in various poses in multiple locations [11]. The data set also includes 527 rendered images of male and female humans of various diverse backgrounds and races presented in the training set. [11]. Images in this dataset are also larger compared to the other data sets used, with color images of size 300×300.

Lastly, we used the Canadian Institute For Advanced Research-10 (CIFAR-10) dataset which consists of 60,000 color images with ten mutually exclusive classes [7]. CIFAR-10 is a significantly more complex dataset and it is widely used as a benchmark for CV algorithms.

C. Experiments

Before starting the training and testing process, we begin by normalizing the pixel values of the images in the data sets. This is done to ensure we have stability when performing floating point operations.

1) Training Process: The training process of our hybrid model is similar to that of a standard DNN in that we do a forward pass of a batch of inputs through the layers of the network. Again, since we are omitting the use of the dense layers, our models include convolutional layers and end with a flattening layer as the output layer. We perform Pooling and Batch Normalization but don't use Drop Out layers in our hybrid models. We feed the output features directly to the

TABLE I
OUR ARCHITECTURE FOR MNIST DIGITS, FASHION, AND SVHN DATA SETS [4]

Layer	Number of Kernels	Kernel Size	Stride	Padding
Convolution	32	(5×5)	(1, 1)	None
Convolution	32	(5×5)	(1, 1)	None
Batch Normalization				
Max Pooling		(2×2)	(2, 2)	None
Convolution	64	(3×3)	(1, 1)	None
Convolution	64	(3×3)	(1, 1)	None
Max Pooling		(2×2)	(2, 2)	None
Flattening				

TABLE II
OUR ARCHITECTURE FOR CIFAR-10 DATA SET [5]

Layer	Number of Kernels	Kernel Size	Stride	Padding
Convolution	96	(5×5)	(1, 1)	Same
Convolution	80	(5×5)	(1, 1)	Same
Max Pooling		(2×2)	(2, 2)	None
Convolution	96	(5×5)	(1, 1)	Same
Convolution	64	(3×3)	(1, 1)	Same
Max Pooling		(2×2)	(2, 2)	None
Flattening				

TABLE III
BASELINE ARCHITECTURE FOR MNIST DIGITS, FASHION, AND SVHN DATA SETS [4]

Layer	Number of Kernels	Kernel Size	Stride	Padding
Convolution	32	(5×5)	(1, 1)	None
Convolution	32	(5×5)	(1, 1)	None
Batch Normalization				
Max Pooling		(2×2)	(2, 2)	None
Convolution	64	(3×3)	(1, 1)	None
Convolution	64	(3×3)	(1, 1)	None
Max Pooling		(2×2)	(2, 2)	None
Flattening				
Dense	256			
Dense	128			
Dense	84			
Dense	10			

TABLE IV
BASELINE ARCHITECTURE FOR CIFAR-10 DATA SET [5]

Layer	Number of Kernels	Kernel Size	Stride	Padding
Convolution	96	(5×5)	(1, 1)	Same
Convolution	80	(5×5)	(1, 1)	Same
Max Pooling		(2×2)	(2, 2)	None
Convolution	96	(5×5)	(1, 1)	Same
Convolution	64	(3×3)	(1, 1)	Same
Max Pooling		(2×2)	(2, 2)	None
Flattening				
Dense	256			
Dense	10			

NCA loss function to predict and calculate the training error. We then use the Stochastic Gradient Descent (SGD) algorithm as our optimizer to update the parameters with respect to the NCA loss.

Since the NCA loss is dependent on the batch size, we discovered to use a batch size $B >= 2C$, where C is the number of classes in the data set. We found this to be the minimum batch size for our experiments in order to increase the chances that each sample in the batch can be paired with another sample of the same class, i.e. each sample can have a nearest neighbor.

Our hybrid model uses the Rectified Linear Unit (ReLU) activation function; any activation function can be used. It is important to note that our implementation of the NCA loss uses the Euclidean Distance to calculate the distance between two feature vectors. We found that ReLU and ReLU like activation functions would result in large distances between feature vectors. Due to the exponential function in the Softmax, this would result in dividing by zero and thus lead to underflow.

To avoid running into the chance of dividing by zero, we clip the values to a minimum threshold of $\epsilon = 10^{-12}$ for any probability p_{ij} in the Softmax cover. Note that the diagonals are still set to zero to include the Leave-One-Out classification error.

2) Testing Process: For our hybrid approach, we evaluate the trained model with a standard KNN algorithm. In order to predict a label for new samples, we use the features from the training set to compute a distance to the new sample.

To implement the algorithm, we pass the new sample through the CNN and receive a feature vector from the flattening layer. We then iterate through the training set and use the feature vectors of the training set to calculate the Euclidean distance to the new sample. Next we sort the distance values from smallest to largest and find the K nearest neighbors to the new sample and use the KNN voting method to select a label for the new sample.

In order to speed up inference time when using large data sets, we opted to use a subset of the training set with balanced classes. We randomly sample from the training set until we reach a bin size of b for each of the C classes. We found this to be the most efficient way to reduce inference time while maintaining performance.

3) Varying The Training Data Set Size: Our approach to evaluate the models in these experiments was different than the one explained above. For these experiments, we went with a $K = 1$ nearest neighbor approach, varied the size of each data set, and used only two classes from the the MNIST Digits and Fashion data sets for binary classification. Training set sizes used for these experiments can be seen in Table VI.

First we train with the full training set size, then we train the models with half the dataset. This is then followed by training the models using only 160 samples from the training set. For the MNIST Digits and Fashion data sets, we use batch sizes of 1024, 512, and 32 images for each training set size. For the Horses or Humans data set we use batch sizes of 64, 32, and 16 images. The testing set also follows the same scheme as the training set size except the last testing set size is only 15 samples. This was done to evaluate the performance of both the hybrid models and the baseline models when the training size decreases.

4) Non-Linear Orientation: We have selected two model location variation where we will implement non-linear activation functions called Iterative and Post. Iterative utilizes activation functions inside the second convolutional layer in each convolutional block. Post utilizes activation functions after each batch normalization. These locations were decide upon due to previous work done in activation function comparisons [2]. With these two activation locations declared, we can implement more variations of models with a combination of these locations to measure the overall feature learning from the convolutional layers. We selected the sigmoid, tanh, and ReLU activation functions strictly for their standard use in DNNs. The Cross Entropy (CE) loss function was used to train the Non-Bias models in this experiment.

TABLE V
MULTI-CLASS DATA SETS USED

Data Set	Train Size	Test Size
MNIST Digits	60,000	10,000
MNIST Fashion	60,000	10,000
SVHN	73,257	26,032
CIFAR-10	50,000	10,000

TABLE VI
BINARY DATA SETS USED

Data Set	Train Size	Test Size
MNIST Digits (0s and 1s)	12,665	2,115
MNIST Fashion (0s and 1s	12,000	2,000
Horses or Humans	1,027	256

V. RESULTS

A. Multi-Class Classification Experiments

We trained a model for each of the four data sets with their respective architectures as seen in Tables I and II. We trained each model using batch sizes of 32, 64, 128, and 256], started with a learning rate of $\eta = 10^{-3}$ and decreased it by a factor of 10^{-1} when the model was stuck at a local minima. We set a maximum of five learning rate decreases before the training process would be stopped.

As mentioned before, to decrease the inference time while maintaining performance, we randomly sampled the training set to a subset of 10,000 samples of equally sized bins. Furthermore we used a batch of testing samples to reduce the inference time. A opted to use a value of $K = \sqrt{10,000} = 100$ for the KNN classifier.

We compared our hybrid end-to-end model results with their baseline counter parts that included the densely connected layers. The baseline architectures can be seen in Tables III and IV, all the baseline models were individually trained using the same data set, learning rates, batch sizes, and SGD optimizer. Each baseline model was trained using the CE loss function.

Tables VII-X show the results of our experiments, pairing our hybrid end-to-end model with their baseline counterparts.

With MNIST Digits being a relatively simple problem, the performance of our model was neck to neck with its baseline counterpart. We can see in Table VII that both our hybrid model and the baseline model were able to achieve a testing accuracy of over 99% in all four trials. More importantly, our method was able to reduce overfitting compared to the baseline. We based this by subtracting the testing accuracy from the final training accuracy. As we can see in Table VII, our hybrid model had a higher testing accuracy than its final training accuracy; the baseline overfitted by a small amount.

With MNIST Fashion dataset being a more complicated problem than MNIST Digits, we see both our hybrid model and its baseline counterpart overfitting the training set. As seen in Table VIII, with a batch size of 32, we see our hybrid model having a train and test accuracy of 91.43%. In the other trials we are still overfitting less than the baseline.

In Table IX we see the results of of our hybrid model and the baseline model trained on the SVHN data set. We see that

TABLE VII
CONV-KNN VS BASELINE (MNIST DIGITS)

	Conv-KNN			Baseline		
Batch Size	Train	Test	Difference	Train	Test	Difference
32	95.71%	99.44%	-3.71%	99.65%	99.32%	0.33%
64	99.14%	99.31%	-0.17%	99.69%	99.41%	0.28%
128	99.18%	99.29%	-0.11%	99.71%	99.21%	0.50%
256	98.85%	99.14%	-0.29%	99.74%	99.13%	0.61%
Number of Parameters	81,952			274,246		

TABLE VIII
CONV-KNN VS BASELINE (MNIST FASHION)

	Conv-KNN			Baseline		
Batch Size	Train	Test	Difference	Train	Test	Difference
32	91.43%	91.43%	0.00%	99.50%	90.96%	8.54%
64	94.34%	90.73%	3.61%	99.76%	90.37%	9.39%
128	95.56%	89.86%	5.70%	99.74%	90.25%	9.49%
256	90.63%	89.13%	1.50%	99.91%	89.38%	10.53%
Number of Parameters	81,952			274,246		

TABLE IX
CONV-KNN VS BASELINE (SVHN)

	Conv-KNN			Baseline		
Batch Size	Train	Test	Difference	Train	Test	Difference
32	84.98%	93.52%	-8.54%	99.93%	92.41%	7.52%
64	91.32%	93.35%	-2.03%	99.83%	91.55%	8.28%
128	92.03%	92.95%	-0.92%	99.91%	90.77%	9.14%
256	90.41%	91.97%	-1.56%	90.48%	88.17%	2.31%
Number of Parameters	83,552			390,534		

TABLE X
CONV-KNN VS BASELINE (CIFAR-10)

	Conv-KNN			Baseline		
Batch Size	Train	Test	Difference	Train	Test	Difference
32	93.50%	80.30%	13.20%	99.86%	75.90%	23.96%
64	96.78%	79.49%	17.29%	99.95%	75.70%	24.25%
128	96.72%	78.18%	18.54%	99.98%	75.59%	27.39%
256	81.34%	75.36%	5.98%	99.97%	70.57%	29.40%
Number of Parameters	446,832			1,498,234		

the hybrid model significantly overfits less than the baseline model. Following the same trend as when trained with the other data set, the hybrid model has a higher test accuracy with a batch size of 32. What is most interesting is that the hybrid model performs better when trained on the SVHN data set. We can see that the hybrid model has the lowest accuracy differences for this data set than it does with the others. SVHN is suppose to be a more complex data set than MNIST Digits but the model overfits less on the SVHN data set overall.

CIFAR-10 is the most complicated dataset we used since it consists of images of real objects in different settings. We can see in Table X that our model outperforms the baseline in every trial. We have been able to reduce overfitting significantly as well with the best trial being the model having a batch size of 256. As we can see, the hybrid model only overfits by 5.98% while the baseline overfits by 29.40%.

It's evident that overfitting will become a problem as the data sets become more complicated, but we aim to reduce overfitting by reducing the number the trainable parameters.

TABLE XI
DATA SET SIZE VARIATION (MNIST DIGITS)

Train Size	Batch Size	Conv-KNN Accuracy	Baseline Accuracy
12665	1024	99.93%	99.34%
6332	1024	99.91%	99.04%
160	1024	99.80%	93.31%
12665	512	99.88%	99.61%
6332	512	99.86%	99.33%
160	512	99.88%	92.13%
12665	32	99.63%	99.64%
6332	32	99.60%	99.49%
160	32	99.65%	97.50%

As seen in Tables VII-X, our hybrid models have less than half the number of trainable parameters than the baseline models. We can see that the fully connected layers account for most of the trainable parameters. We argue that we can eliminate the fully connected layers since our hybrid models perform just as well and at times better than the baseline models.

TABLE XII
DATA SET SIZE VARIATION (MNIST FASHION)

Train Size	Batch Size	Conv-KNN Accuracy	Baseline Accuracy
12000	1024	96.16%	94.69%
6000	1024	96.50%	92.90%
160	1024	92.87%	84.86%
12000	512	96.72%	96.22%
6000	512	96.81%	95.26%
160	512	92.70%	83.33%
12000	32	88.22%	96.86%
6000	32	88.51%	95.51%
160	32	91.02%	78.38%

TABLE XIII
DATA SET SIZE VARIATION (HORSES OR HUMANS)

Train Size	Batch Size	Conv-KNN Accuracy	Baseline Accuracy
1027	64	95.35%	87.73%
513	64	95.98%	73.09%
160	64	96.05%	50.00%
1027	32	92.89%	90.55%
513	32	94.26%	75.86%
160	32	94.65%	50.00%
1027	16	88.24%	89.41%
513	16	90.27%	82.11%
160	16	91.29%	50.00%

B. Varying The Training Data Set Size

Results for our small data set experiments can be seen in Tables XI-XIII. Note we went with a binary classification approach in these experiments and use classes 0 and 1 of the MNIST Digits and Fashion data sets.

Again, since MNIST Digits is a simple problem, the differences in accuracy between the models is fairly small. Our hybrid model was still able to outperform its baseline counterpart by 0.29% when comparing their highest accuracy. We can see that the baseline's accuracy begins to decline as the number of training samples decreases. As for our hybrid model, there is relatively no change in accuracy.

We begin to observe a larger difference between our hybrid model and its baseline counterpart when using the MNIST Fashion data set. When comparing the highest accuracies achieved, our hybrid model was still able to outperform the baseline by 0.44% and never falling below 88.00%. On the contrary, the baseline's lowest accuracy fell to 78.38% with a batch size of 32 and a training set size of 160.

We begin to see a sharper decline in accuracy by the baseline model when using the Horses or Humans dataset. The baseline model achieves a maximum accuracy of 90.55% with a train set size of 1,027 and batch size of 32. It achieves a minimum accuracy of 50.00% with a train set size of 160. Our hybrid model achieves a maximum accuracy of 99.49% and a minimum accuracy of 86.44% with train set size and batch size of 1027 & 16 and 1027 & 64 respectively.

From these results we can see that our hybrid models are more robust to the size of the training data set.

C. Non-Linear Orientation

The architecture of the Non-Bias CNN model used in these experiments can be seen in Table XIV. The architecture of this model is similar to the CNN in [4], but the complexity is minimized by the reduction in dense layers. We utilize architecture closely resembling [4] because we need an accurate modern baseline model to compare against our convolutional-only learning results. It is also one of the most optimized and high performing CNN for our data set.

Table XV shows the results of our activation function comparison experiments. Besides Iterative and Post, we compared against a single activation function and no activation function orientations. As mentioned above with MNIST Digits being a simple problem, we are still able to yield strong results with differing orientations. It is apparent that orientation does not influence a higher model evaluation. Furthermore, we notice that *None* yields comparable evaluation percentage to that of the activation functions

VI. DISCUSSION & FUTURE WORK

*A. **Can we eliminate the fully connected (dense) layers?***

Our goal is to remove a portion of the black-box nature of CNNs by eliminating the use of the dense layers for classification. As seen in our results, our Convolutional-KNN hybrid approach was able to perform just as well and at times outperform the baseline CNN models. The trade off between interpretability and performance is seen in both the Digits and Fashion data sets by a small amount. To our surprise, the trade off is not seen in our SVHN experiment with our hybrid model performing better than the regular CNN model. When trained on four data sets, we can see that the dense layers could account for most of the overfitting done by the model.

From our results we can safely argue that when it comes to these classification problems, we can exclude the use of the fully connected layers in CNNs. Our models achieved a higher testing accuracy and also had less overfitting as opposed to the baseline models. Further work needs to be done using larger data sets and more complex classification problems to give a complete answer.

*B. **How much learning can we do in the convolutional layers?***

In our experiments, we take on a hybrid end-to-end approach of training CNNs by omitting the use of the dense layers. By optimizing the NCA loss function, we exhaust the tuning of the kernels of each convolutional layer in the hybrid model. Therefore we say that we exhaust the learning of the convolutional layers for that particular model. In all instances, the convolutional layers alone (hybrid model) have a lower training accuracy than the standard CNN (baseline). However, when evaluating the hybrid model with a KNN classifier, we achieved testing accuracies higher than the standard CNN in most cases.

The amount of learning a DNN is depends on the task at hand and the loss function we seek to minimize. We can see from our experiments that the convolutional layers trained with the NCA loss function can learn just as much as the baseline models trained with a CE loss function. To give a complete

TABLE XIV
ARCHITECTURE FOR NON-BIAS CNN

Layer	Number of Kernels	Kernel Size	Stride	Padding
Convolution	32	(5×5)	(1,1)	None
Convolution	32	(5×5)	(1,1)	None
Batch Normalization				
Max Pooling		(2×2)	(2,2)	None
Dropout				
Convolution	64	(3×3)	(1,1)	None
Convolution	64	(3×3)	(1,1)	None
Batch Normalization				
Max Pooling		(2×2)	(2,2)	None
Dropout				
Flatten				
Dense	10			

TABLE XV
ACTIVATION FUNCTION COMPARISON

Activation Function	1-Final Layer	2-Iterative	2-Post Batch	4-Iterative and Post
ReLU	98.56%	99.26%	98.69%	99.08%
Tanh	97.83%	98.07%	98.47%	98.82%
Sigmoid	97.96%	98.48%	92.94%	96.53%
None	98.25%			

answer, we would still need to test our hybrid method using various architectures.

C. *How does the training data set size affect learning?*

To explore this question, we used subsets of the MNIST Digits and Fashion data sets along with the Horses or Humans data set. We reduced the Digits and Fashion data sets to only two classes (0 and 1) which in return reduced the size to 12,000 images for the training sets. Furthermore, we shrunk the number of images in each data set to half for other trials and went to as low as 160 images for the training set. We also varied the testing set size in a similar fashion.

We saw a decline in testing accuracy from the standard CNN models but our hybrid models were able to be consistent as the set sizes varied. From these results we can conclude that our hybrid method is robust to the data set size. This could be as a result of the NCA loss function which seeks to maximize the expected number of points correctly classified. Further work would need to be done using multi-class classification as well as data sets that are designed to be smaller.

D. *Can we use non-linear activation functions in the convolutional layers to increase learning?*

Activation functions are widely considered an essential part of most modern day neural nets and can largely influence the performance of the model [12]. As we try to answer our above question, it is important to note that although our experiments show that non-linear activation functions do not demonstrate a significant increase in learning, the experiments were done on a fairly simple data set with only three activation function comparisons. For this question to be fully answered, we must examine a larger variety of non-linear activation functions and their evaluation in respect to orientation as well as using more complex data sets [2]. Furthermore, it may be necessary to experiment with different model architectures.

VII. CONCLUSION

In this paper, a Hybrid End-to-End Convolutional K-Nearest Neighbors model has been proposed by replacing the fully connected layers with a KNN classifier. The convolutional layers were trained alone using the Neighborhood Component Analysis loss function to learn feature embeddings that will further aid the KNN classifier.

Our results show that our approach produces hybrid models that are robust to data set sizes and perform just as well and at times outperform the standard CNN models. Our models are more transparent than standard CNNs and, with post-hoc techniques, can provide more explainability. We have shown that we can eliminate the fully connected layers in CNNs and begin to use transparent methods and loss functions to train the convolutional layers alone.

By removing a portion of the inherent black-box nature of DNNs, we hope our hybrid approach could help promote trust in using hybrid DNN methods for real-world problems outside of the research world.

VIII. FUTURE WORK

To further explore our research questions we aim to use several well known deeper architectures like VGG-16 and train them with our hybrid approach. We would also like to use large data sets such as ImageNet to increase the complexity of the learning process as well as smaller data sets to evaluate the amount of learning the convolutional layers can learn. We also reserve the use of Grad-CAM for future testing to further compare the kernels of the convolutional layers trained with our approach.

REFERENCES

[1] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information fusion*, 58:82–115, 2020.

[2] Disha Gangadia. Activation functions: Experimentation and comparison. In *2021 6th International Conference for Convergence in Technology (I2CT)*, pages 1–6. IEEE, 2021.

[3] Jacob Goldberger, Geoffrey E Hinton, Sam Roweis, and Russ R Salakhutdinov. Neighbourhood components analysis. In L. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 17. MIT Press, 2004.

[4] Jay Gupta. Going beyond 99% — mnist handwritten digits recognition, 2020.

[5] Md Mosharaf Hossain, Douglas Talbert, Sheikh Ghafoor, Ramakrishnan-Ramki Kannan, et al. Fawca: A flexible-greedy approach to find well-tuned cnn architecture for image recognition problem. Technical report, Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States), 2018.

[6] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[7] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[8] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[9] Ritchie Lee, Justin Clarke, Adrian Agogino, and Dimitra Giannakopoulou. Improving trust in deep neural networks with nearest neighbors. In *AIAA Scitech 2020 Forum*, page 2098, 2020.

[10] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[11] Laurence Moroney. Horses or humans dataset, feb 2019.

[12] Arijit Nandi, Nanda Dulal Jana, and Swagatam Das. Improving the performance of neural networks with an ensemble of activation functions. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2020.

[13] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[14] Nicolas Papernot and Patrick McDaniel. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv preprint arXiv:1803.04765*, 2018.

[15] Tobias Plötz and Stefan Roth. Neural nearest neighbors networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[16] Weiqiang Ren, Yinan Yu, Junge Zhang, and Kaiqi Huang. Learning convolutional nonlinear features for k nearest neighbor image classification. In *2014 22nd International Conference on Pattern Recognition*, pages 4358–4363, 2014.

[17] Ruslan Salakhutdinov and Geoff Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Artificial Intelligence and Statistics*, pages 412–419. PMLR, 2007.

[18] Daniel Tarlow, Kevin Swersky, Laurent Charlin, Ilya Sutskever, and Rich Zemel. Stochastic k-neighborhood selection for supervised and unsupervised learning. In *International Conference on Machine Learning*, pages 199–207. PMLR, 2013.

[19] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

[20] Wei You, Changqing Shen, Dong Wang, Liang Chen, Xingxing Jiang, and Zhongkui Zhu. An intelligent deep feature learning method with improved activation functions for machine fault diagnosis. *IEEE Access*, 8:1975–1985, 2019.