Dairy Cows Teat-End Classification using Deep Learning

Article · November 2023

CITATIONS
0

READS
20

1 author:

Shilpa Kuppili
Yeshiva University
3 PUBLICATIONS **0** CITATIONS

SEE PROFILE

Dairy Cows Teat-End Classification using Deep Learning

Shilpa Kuppili
Department of Artificial Intelligence, Yeshiva University
shilpakuppili@i1.org

Abstract

In order to preserve dairy cow health and milk quality, teat-end health evaluations are essential. Using a convolutional neural network to classify the extent of teat-end modifications based on digital images is one method for automating teat-end health checks. GoogLeNet has shown that this strategy is workable, but there are still a lot of obstacles to overcome, like poor performance and comparing results with other ImageNet models. To enhance teat-end picture classification performance, we introduce a separable confident transductive learning (SCTL) model in this study.

1. Introduction

The health of dairy cows is important for milk quality and the health of the mammary gland. Traditionally, teat-end health has been assessed manually through visual inspection of teat-end callosity thickness and roughness (i.e., hyperkeratosis), which is a risk-factor for mastitis. Here, we describe a computer-vision approach to replace the time-consuming and expensive manual assessment of teat-end hyperkeratosis. Using separable confident transductive learning, a convolutional neural network is trained with the goal of increasing the feature differences in the images of teat-ends with different classifications of hyperkeratosis. When compared with the traditional approach of transfer learning of a convolution neural network for classifying the extent of hyperkeratosis. This substantial improvement in accuracy renders the possibility of using image-based machine learning to routinely monitor hyperkeratosis on commercial dairy farm settings.

One of the most common ailments in dairy cows is still mastitis, which commonly results from intramammary infections that enter through the teat canal. The integrity of the teat canal may be impacted by machine milking, and this may result in more callosity at the teat end, raising the possibility of bacterial infections in the mammary gland . A method to avoid mastitis must regularly assess the callosity of the teat. The current best practice, however, involves time-consuming physical teat-health examinations at the cow's side that are prone to inter- and intra-rater variability. The incapacity of large dairy farms to evaluate the entire herd presents another difficulty. Deep learning (DL), which uses a four-level classification system based on GoogleNet transfer learning to identify the degree of hyperkeratosis, has been offered as a solution to some of these problems.

2. Related Work

Mastitis, an inflammation of one or more mammary glands in cows, is a common condition that affects the health and quantity of milk produced by dairy cows. Teat-end callosity thickness and roughness can increase over several weeks due to changes in circulation brought on by mechanical stressors applied to the cow's teat-end. The chance of harmful microorganisms entering the cow's udders may rise as a result of these teat-end modifications. It is advised to regularly inspect the teat-end health of dairy cows in order to monitor and mitigate these dangers. This is accomplished by physically examining the teat-ends of at least 20 of the herd's cows; yet, herd-level evaluations are costly, time-consuming, subjective, and imprecise. A common classification method for cow teat ends is the four classes grading of hyperkeratosis (Score 1: no ring; Score 2: smooth ring; Score 3: rough ring; Score 4: very rough ring) Several computer vision tasks that depend on the variety of picture input and network parameter adjustment for optimal performance have used transfer learning. Transfer learning has been widely used in picture classification, objection detection, and segmentation problems since the development of several ImageNet models. In a recent paper, Porter proposed a machine-learning method that classified the degree of teat-end hyperkeratosis using a four-point scoring system by training a convolution neural network (CNN), such as GoogleNet, using photos of teat-ends. While our initial approach's overall accuracy on test data indicated the possibility of automatic teat-end assessment, it was comparatively low, indicating the need for development.

Both labeled training data and unlabeled test data are trained through the process of transductive learning (TL). Typically, semi-supervised learning scenarios make use of

it.Transductive learning seeks to identify an admissible function using the unlabeled data to enhance classification performance, in contrast to the widely used supervised inductive classification, which aims to train a classification model based on the labeled training data to approximate test data class distribution. The fundamental concept of TL is that test sample predicted labels are considered as variables for optimization, which can be done iteratively.

Seeking to generate labels or pseudo labels for unlabeled data in order to direct the learning process is the aim of pseudo labeling . Usually, pseudo labeling produces pseudo labels for the unlabeled data based on predicted class probability or hard assigned labels (the neural network predictions. Label information from unlabeled data can be included in training under such a regime. When creating pseudo labels for test data in deep networks, the classifier from the training data is typically used as an initial pseudo labeler (and used as if they were real labels). Numerous algorithms exist to generate pseudo-labels and enhance the functionality of unlabeled data.

3. Methods

3.1. Motivation

Our paper's scientific objective is to create a completely automated deep learning model that can recognize various types of dairy cow teat-end conditions with accuracy. Finding hyperkeratosis of the teat-end region (Scores 3 and 4) in the context of commercial dairy farms is the utilitarian objective. The teat-end image classification task is our research problem, and our goal is to increase the classification accuracy.

3.2. Data

There are 1529 teat images total, divided into four categories (Score 1, Score 2, Score 3, and Score 4) were extracted from the dataset, which comprises 398 dairy cows. For the test dataset, 380 teat images—roughly 70 cows—were used. We divided the dataset into training and test datasets to ensure a fair comparison of various algorithms.

3.3. Implementation

We import statements for various libraries and modules commonly used in machine learning and computer vision tasks.

The transformations are commonly applied to preprocess images before they are used in machine learning tasks or neural network models. transforms.Resize((75, 75)): This transformation resizes the input image to a new size of 75x75 pixels. Resizing an image is a common preprocessing step to ensure uniformity in the size of images before they're used in a neural network. transforms.ToTensor(): This transformation converts the image, which is typically in the format of a PIL Image or an array, into a PyTorch tensor. Neural networks process data in the form of tensors, so this transformation is necessary to convert the image into a format that can be used as input for a PyTorch model. The transforms.Compose function combines these transformations into a sequence, allowing the transformations to be applied sequentially to each input image.

For instance, when using this transformation sequence, a single image would go through resizing to 75x75 pixels, conversion to a PyTorch tensor, and then, if enabled, normalization to ensure the data falls within a certain range or distribution that might be suitable for the model's input. This prepared image could then be fed into a neural network for training, validation, or inference.

The dataset is set up, split it into training and testing sets, create data loaders, and organize the data for training and validation. Here's a brief explanation of each part:

Dataset Creation: we initializes an ImageFolder dataset from the torchvision.datasets module. It is used for handling image data organized in a specific directory structure. The dataset is expected to be located in the 'Train/' directory, and the provided transform variable is applied to preprocess each image in the dataset.

Dataset Split

The random split from torch.utils.data is used to split the dataset into training and testing sets based on the previously calculated sizes. The train size determines the size of the training set, and the remaining images go to the testing set (test size).

Dataloader Creation

DataLoader creates an iterable over the dataset, dividing it into batches for efficient training or testing. The trainloader is set to shuffle the data (shuffle=True) to present batches in random order during training. On the other hand, testloader doesn't shuffle the data (shuffle=False) since shuffling is not needed during testing or validation.A dictionary dataloaders is created to organize the training and validation data loaders.

A simple Convolutional Neural Network (CNN) using PyTorch's nn.Module is defined. This CNN consists of convolutional, max-pooling, and fully connected layers for image classification.

CNN Architecture: A Convolutional Neural Network (CNN) is a type of neural network particularly effective in analyzing visual data, such as images. CNNs consist of multiple layers that perform specific operations to learn and extract hierarchical patterns and features from the input data.

CNN Architecture Components: Convolutional Layer:

Convolutional layers apply filters (kernels) to the input data to detect features like edges, shapes, or textures. Filters slide across the input, performing element-wise multiplication and summation, generating feature maps. Each layer can have multiple filters to capture different features. Pooling Layer:

Pooling layers (commonly max-pooling or average-pooling) reduce the spatial dimensions of feature maps, retaining important information. Pooling helps in controlling overfitting and reducing computational complexity. Activation Function:

Activation functions like ReLU (Rectified Linear Unit) introduce non-linearity, enabling the network to learn complex patterns and relationships in the data. Fully Connected Layers:

Fully connected (or dense) layers connect every neuron from one layer to every neuron in the next layer, aiding in making predictions based on the learned features. These layers often occur at the end of the network for classification or regression tasks . Characteristics of CNNs: Hierarchical Feature Learning:

CNNs automatically learn hierarchical representations of the input data, from simple features in early layers to complex ones in deeper layers. Parameter Sharing:

CNNs exploit parameter sharing, where weights are shared across different parts of the input, reducing the number of trainable parameters and aiding in feature generalization. Translation Invariance:

CNNs demonstrate some degree of invariance to translation, meaning they can recognize patterns regardless of their position in the image. Common CNN Architectures: LeNet:

One of the earliest CNN architectures developed by Yann LeCun for handwritten digit recognition. AlexNet:

A pioneering CNN that gained popularity for winning the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. VGG (Visual Geometry Group):

Known for its simplicity and deep structure, utilizing small convolutional filters. ResNet (Residual Network):

Utilizes residual blocks to address the vanishing gradient problem in very deep networks. Inception (GoogLeNet):

Introduced inception modules with multiple filter sizes within the same layer to capture diverse information. MobileNet:

Designed for mobile and embedded devices, uses depthwise separable convolutions to reduce computational cost. The architecture of the CNN is structured as follows:

Convolutional Layers:

self.conv1, self.conv2, and self.conv3 are convolutional layers. They apply convolution operations to the input data with different filter sizes (16, 32, and 64 filters respectively), producing feature maps. Each convolutional layer uses a kernel (filter) size of 3x3, a stride of 1, and a padding of 1 to maintain spatial dimensions through the convolution. Max Pooling Layers:

self.pool represents the max-pooling operation with a kernel size of 2x2 and a stride of 2. Max pooling reduces the spatial dimensions of the feature maps while retaining essential information. Fully Connected (Linear) Layers: self.fc1, self.fc2, self.fc3, and self.fc4 are fully connected layers responsible for learning higher-level features and making final predictions. These layers contain 1024, 256, 128, and 4 neurons respectively (assuming 4 output classes in this example). The final fully connected layer (self.fc4) outputs predictions for the classification task. Forward Method: The forward method defines the forward pass of the network. It processes the input data through the defined layers in the network.

Convolutional Layers and Activation:

The input data x passes through each convolutional layer, followed by Rectified Linear Unit (ReLU) activation functions. After each convolution, max pooling reduces the spatial dimensions. Flattening:

The output from the convolutional layers is flattened (reshaped) into a 1D tensor to be input into the fully connected layers. Fully Connected Layers and Activation:

The flattened tensor is passed through the fully connected layers, each followed by a ReLU activation, transforming the features. The final layer (self.fc4) doesn't use an activation function, as it often connects to the output layer without an additional non-linearity. Output:

The final output is the result of passing the tensor through the fully connected layers, providing class predictions for the input data. Instantiating the Model: An instance of the CNN model is created using model = YourCNN(). This instance represents the defined architecture, ready to be trained on data for a specific classification task. This CNN architecture can be further trained with appropriate data and labels using an optimizer and a loss function to make predictions for image classification tasks. Adjustments to this architecture, such as changing layer sizes or the number of classes, can be made based on the requirements of the specific task.

The training loop provides a standard structure for training neural networks, consisting of alternating phases of training and validation. It's crucial for updating the model's parameters using backpropagation, monitoring performance, and adjusting hyperparameters for better model generalization. Adjustments, such as modifying learning rates, loss functions, or adding regularization techniques, can be made to further enhance the model's performance.

Training Loop: The loop runs for a defined number of epochs (epochs). Each epoch covers the entire training dataset. Training Phase The model's mode is set to training mode using model.train(). This is important to enable operations like dropout and batch normalization, which behave differently during training and inference. Iterating Over Batches It loops through the trainloader, an iterable object created using DataLoader, going through batches of data for each epoch. Inside the loop: Fetches inputs and labels from the training data. Transfers the data to the specified device (like GPU) for computation. Zeroes out the

gradients before the backward pass. Computes the outputs of the model, calculates the loss between the predictions and the actual labels using the specified loss function (loss fn). Performs backpropagation to compute gradients (loss.backward()) and updates the model's parameters (optimizer.step()). Keeps track of the cumulative loss for the epoch (epoch loss). Further, the predictions list likely contains the model's predictions for each batch in the final loader, where each element in the list corresponds to the predicted class for the respective input data. The entire predictions list, containing the predicted class labels for the processed data is obtained.

Evaluation Loop with Inference: It is ensured that no gradients are calculated or updated during the inference stage. This is efficient for evaluation and inference, as gradients are not required and it saves memory. model.eval() sets the model in evaluation mode, where operations like dropout are disabled. It loops through final loader, a data loader containing test or validation data. Inference Process: Data Processing Loop: It iterates over batches of data in final loader. For each batch: Fetches inputs from the data loader and transfers them to the specified device. Extracts the file names associated with the inputs from the dataset samples and appends them to a list (file names). Passes the inputs through the model to obtain the model outputs. Computes predictions by taking the class index with the highest probability using torch.max along the appropriate dimension (dim=1 in this case). Appends the predictions to a list (predictions). Final Output: At the end of this loop, the predictions list likely contains the model's predictions for each batch in the final loader, where each element in the list corresponds to the predicted class for the respective input data. The code prints the entire predictions list, containing the predicted class labels for the processed data. This process helps in obtaining model predictions for a dataset, associated with the file names or any identification tags. The model's predictions are stored for further analysis, reporting, or any downstream tasks.

Further, we download the csv file and upload it in the developed software and calculate the accuracy.

4. Results

4.1. Datasets

There are 1529 teat images total, divided into four categories (Score 1, Score 2, Score 3, and Score 4) were extracted from the dataset, which comprises 398 dairy cows. For the test dataset, 380 teat images—roughly 70 cows—were used. We divided the dataset into training and test datasets to ensure a fair comparison of various algorithms.

4.2. Results obtained

Overall approach provides a quantitative measure of the model's performance. A smaller loss indicates that the model's predictions are closer to the actual targets.

Further,the loss is calculated between the predictions and the actual labels using the specified loss function. The accuracy is calculated using the matlab developed software. The obtained csv file is uploaded and accuracy is calculated as around 60.

5. Discussion

The proposed SCTL model has multiple benefits. Firstly, the separation loss we propose increases the inter-class dispersion and the difference between categories. In order to optimize the network using pseudo label information, we secondly generate highly confident pseudo labels for test data using three iterations of adjustment learning. Finally, we minimize the divergence between training and test data. Our SCTL model can improve the teat-end image classification problem's performance by combining all three of these innovative loss functions.

6. Conclusion

Our proposal for teat-end image classification in this paper is a separation confident transductive learning model. In order to increase the distinctions between various categories, we first suggest a separation loss. Then, we use adjustment learning to optimize the network and produce reliable labels for the test data. In order to minimize the divergence between the training and test data with a categorical MMD loss, we finally use transductive learning. We show that, in comparison to ImageNet transfer learning models, the proposed SCTL model can achieve higher accuracy, even though the degree of affection of the cows' teats can affect its performance. We think that in the context of a commercial dairy farm, hyperkeratosis can be detected with the help of SCTL. Our method allows for automated teat-end condition assessments to occur more frequently. By reducing the risk of intramammary infections, using fewer antibiotics, controlling the expense of diagnosing and treating mastitis, and enhancing the quality of life for both farmers and dairy cows, an automated hyperkeratosis detection method could benefit farmers.

References

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

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer

Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 4510–4520. [Google Scholar]

Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 4700–4708. [Google Scholar]

Chollet, F. Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 1251–1258. [Google Scholar]

Shi, W.; Gong, Y.; Ding, C.; Tao, Z.M.; Zheng, N. Transductive semi-supervised deep learning using min-max features. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 299–315. [Google Scholar]

Quadrianto, N.; Petterson, J.; Smola, A. Distribution matching for transduction. Adv. Neural Inf. Process. Syst. 2009, 22, 1500–1508. [Google Scholar]

Arazo, E.; Ortego, D.; Albert, P.; O'Connor, N.E.; McGuinness, K. Pseudo-labeling and confirmation bias in deep semi-supervised learning. In Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, 19–24 July 2020; pp. 1–8. [Google Scholar] Lee, D.H. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. Workshop Chall. Represent. Learn. ICML 2013, 3, 896. [Google Scholar]

Saito, K.; Ushiku, Y.; Harada, T. Asymmetric tri-training for unsupervised domain adaptation. arXiv 2017, arXiv:1702.08400. [Google Scholar]

Xie, S.; Zheng, Z.; Chen, L.; Chen, C. Learning semantic representations for unsupervised domain adaptation. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 5423–5432. [Google Scholar]

Iscen, A.; Tolias, G.; Avrithis, Y.; Chum, O. Label propagation for deep semi-supervised learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 5070–5079. [Google Scholar]

Haase-Schütz, C.; Stal, R.; Hertlein, H.; Sick, B. Iterative Label Improvement: Robust Training by Confidence Based Filtering and Dataset Partitioning. In Proceedings of the 2020 25th International Conference on Pattern Recognition (ICPR), Milan, Italy, 10–15 January 2021; pp. 9483–9490. [Google Scholar]

Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet classification with deep convolutional neural networks. In Proceedings of the Advances in Neural Information Processing Systems, Lake Tahoe, NV, USA, 3–6 December 2012; pp. 1097–1105. [Google Scholar]