Deep Learning Classification of Lipid Droplets in Quantitative Phase Images

L. Sheneman[1,*], A. E. Vasdekis[2,*]

[1]Northwest Knowledge Network, University of Idaho, Moscow, ID, 83844, USA

[2]Department of Physics, University of Idaho, Moscow, ID, 83844, USA

* Correspondence should be addressed to LS (sheneman@uidaho.edu) or AEV (andreasv@uidaho.edu).

Abstract

We report the application of supervised machine learning to the automated classification of lipid-droplets in label-free, quantitative-phase images. By comparing various machine learning methods that are publicly available and commonly used in biomedical imaging and remote sensing, we found convolutional neural networks to outperform others, both quantitatively and qualitatively. We describe our imaging approach, the implementation of all machine learning methods that we compared, and their performance in computational requirements, training resource needs, and accuracy. Overall, our results indicate that quantitative-phase imaging coupled to machine learning enables accurate lipid droplet classification in single living cells. As such, the present paradigm presents an excellent replacement candidate of the more common fluorescent and Raman imaging modalities by enabling label-free, ultra-low phototoxicity and deeper insight into the thermodynamics of metabolism at the single-cell level.

Introduction

Quantitative-phase imaging (QPI) of biological systems has met with significant success in recent years, both in fundamental and biomedical investigations [1-5]. In QPI, an image is formed by quantifying the optical path length (or optical phase delay) difference between the specimen and its background without any fluorescent staining [6-8]. This type of imaging confers unique information about cellular physiology and metabolism, including cellular dry-mass and dry-density that are not attainable via conventional, volumetric, bioimaging methods [9]. Further, QPI enables high-contrast imaging between cells and their background, which has found applications in localizing the contour of individual cells (i.e., performing cell segmentation) without any computationally intensive approaches [10, 11]. More recently, QPI has been interfaced with deep learning methods in various applications ranging from image reconstruction to object recognition [12-18]. In the context of object recognition, applications such as screening histological targets [19], colorectal cancer [20] and hematological disorders [21, 22], as well as the detection of nuclei in structural biology applications [23] have been demonstrated.

Here, we report the application of deep-learning in recognizing and localizing lipid droplets (LDs) directly in quantitative-phase images without any staining. LDs are an important cytosolic organelle where all eukaryotes and some bacteria store neutral lipids in the form of triacylglycerols, steryl and retinyl esters [24-26]. Catabolic utilization of these compounds provides cells with membrane building blocks and energy in the form of ATP, indicating an important role of LDs in energy homeostasis and metabolism [27, 28]. Recent evidence has expanded the role of LDs as a key trafficking node of

proteins, transcription factors and chromatin components [29-33]. The implications of LDs in disease have also been better understood in recent years, primarily in the context of infection by intracellular pathogens [34, 35] and viruses [36, 37], as well as fatty-liver disease [38] and various forms of cancer [39]. Further, LDs have found tremendous applications as a sustainable source of biodiesel production [40-42].

The ability to visualize LDs at the single-cell level has greatly advanced our understanding of the key roles that LDs play in metabolism, disease and biotechnology, as well as has unmasked the underlying effects of cellular noise [43-45]. While fluorescence and Raman microscopy have been the most common imaging methods in such investigations [46-51], LD localization by QPI has also been recently demonstrated [52, 53]. The advantages of QPI over other imaging methods pertain primarily to label-free imaging with ultra-low phototoxicity (typically at $\mu W/cm^2$ illumination densities) that can confer key thermodynamic parameters, such as the ensemble number-density of molecular assemblies. However, due to the relatively low refractive index contrast between LDs and other organelles, LD localization in single-living cells by QPI exhibits insufficient discriminatory power and, thus, low specificity (**Figure 1**). As such, QPI is not compatible with automated high-throughput image processing in LD localization LDs, with the exception of coupled deconvolution- and correlation-based post-processing schemes, albeit at 86% success rates [53].

To overcome these shortcomings and enable fully automated LD recognition in QPI, we explored various supervised machine learning methods. Using *Yarrowia lipolytica*, (**Figure 1**), a tractable oleaginous yeast [54-56], we explored simple ensemble decision tree models, including random forest [57, 58] and gradient boosting [59, 60], as well as

deep learning convolutional neural network (CNN) [61-64]. We found that CNNs outperform decision tree models in a number of metrics, including normalized template cross-correlation and the Sørensen-Dice coefficient (i.e., “Dice scores” [65-68]). In the following sections, we describe the generation of binary training images, detail each machine learning method, and compare both quantitatively and qualitatively their performance.

Methods

Building our Curated Labeled Image Library

We started by labeling a large image library of individual *Y. lipolytica* cells that we have previously reported [53]. The library consists of two *Y. lipolytica* strains, Po1g and MTYL038 [54] grown under conditions selected growth conditions that yield lipid content greater than 1.5% per weight. These conditions pertain to time points greater than 28 hours in a defined YSM medium with a carbon-to-nitrogen ratio (C/N) equal to 150. As detailed in [53], cells were transferred from rich YPD medium (20 g/L Bacto Peptone from BD, 10 g/L yeast extract from Alfa Aesar, 20 g/L glucose from Fisher) to the defined YSM medium (1.7 g/L yeast nitrogen base without amino acids and without ammonium sulfate from BD Difco) 0.69 g/L complete supplement mixture without Leucine from Sunrise Science Products, 1.1 g/L ammonium sulfate from Fisher and 75 g/L glucose from Fisher) at 50 × dilution (dilutions yielding starting ODs at 0.01) and centrifugation (490 × g) and washing in YSM three times. For the Po1g strain, 0.1 g/L Leucine was added in the YSM medium. All growth experiments were performed in a shaking incubator at a temperature of 29 °C.

To perform quantitative phase imaging, we placed 2 μL from a growing culture between two coverslips and pressed gently to minimize the distance between the two coverslips prior to imaging. We acquired all images using a quantitative-phase imaging system (Phi Optics) that relies on the spatial light interference microscopy (SLIM) modality. In SLIM, images are formed with the aid of a liquid crystal phase modulator that shifts the optical phase of the light wavefront scattered by the specimen with respect to un-scattered light [70]. This system was coupled to an inverted microscope (DMi8, Leica) equipped with an automated *xyz* stage, a 100× magnification objective (NA 1.3, PH3), and a sCMOS camera (Orca Flash 4.0, Hamamatsu). 3D images were acquired by scanning the objective along the imaging path (*z*-axis) with a step of 400 nm and the stage in the *xy* plane. Due to the lower axial resolution of our imaging set-up than the lateral one, we compressed the acquired 3D images into 2D via the maximum projection method. Using the 2D images, we detected the cell contour by direct optical-phase thresholding and no further pre-processing [10].

To localize LDs and generate a binary training data-set, we first 3D deconvoluted two phase-modulated intensity images [$I_{\pi/2}(x,y,z)$ and $I_{\pi}(x,y,z)$] using the AutoQuant X3 procedure (MetaMorph), and subsequently cross-correlated these two images as we have previously described [53]. To minimize errors during training and classification, we further evaluated each single-cell image and excluded subsets with noticeable defects (e.g. specks of dust occluding the image), apoptotic cells, small buds, and cells that spatially overlapped. Our final usable image library consisted of approximately 7,000 32-bit grayscale images of single *Y. lipolytica* cells. With the goals of performing controlled tests while evaluating the effect of training set size on performance and

accuracy, we randomly split our image library into a fixed set of 2,000 test images and 5 different training sets ranging between 1,000 images and 5,000 images (**Figure 2**). These test and training sets were then used in all evaluations.

Random Forest and Gradient Boosting

We first implemented random forest and gradient boosting via *XGBoost* [71], two ensemble machine learning methods that have proven effective with the semantic segmentation of images across a number of disciplines including bioimage analysis [72-79] and remote sensing [80-85]. Unlike neural networks and deep learning methods, models trained with these algorithms benefit from simplicity and a relatively high level of interpretability [86-88]. The individual estimators (decision trees) within these models can be understood and interpreted without significant effort. As such, the contribution and importance of individual features can be estimated [89, 90] and used to subsequently optimize feature selection [91-93].

Both of these supervised machine learning methods are based on the principle that one can build a strong learner from a collection, or an *ensemble* of weak learners such as decision trees [94]. During model training, both random forest and XGBoost methods construct a set of individual binary decision trees. During classification, these two methods evaluate each pixel by each tree within the ensemble and a simple majority-vote is performed to determine the final classification for each pixel (**Figure 3**).

Random forest and gradient boosting differ in important ways. The random forest

method builds a forest of decision trees using a form of bootstrap aggregation ("bagging") in which random subsamples of both the input data and feature classes are used when constructing any given tree [95, 96]. Individual decision trees are weak learners primarily because they are high-variance estimators that tend to overfit the training data. By building an ensemble of individual *bagged* trees in which the information used to construct the trees is derived from a random subset of both data and features, the correlation between any two trees in the forest is reduced and the overall ensemble becomes less prone to overfitting the training data.

Gradient boosting differs from the random forest method primarily in how the decision trees are constructed [97]. Rather than using bootstrap aggregation during decision tree construction, gradient boosting uses *boosting* [98] by starting with a set of trivially short decision "stumps" and sequentially building out new decision trees. Each new tree in the ensemble is constructed to minimize the *residuals* (differences between observed and predicted values) of previously constructed trees. In this way, gradient boosting is a form of iterative refinement of the overall ensemble, as each new tree is sequentially constructed to reduce the remaining error in the overall model. To help prevent overfitting, trees constructed via gradient boosting are purposely shallow and typically constrained by a hyperparameter depth value of 3 or 4. In our evaluation, we specifically use XGBoost, a popular modern gradient boosting implementation. XGBoost differs from conventional gradient boosting algorithms by computing the second order derivative of the loss function for improved optimization efficiency while using different forms of *regularization* [99] to prevent the model from becoming overly complex and prone to overfitting. [100] XGBoost has a very computationally efficient implementation

and integrates directly with our *scikit-learn* Python framework [101].

The training and classification workflows are the same for both our random forest and XGBoost implementations (***Figure 3, Sup. Figure 1***). During training, we read all *Y. lipolytica* images from our training set as a collection of single-band 32-bit grayscale TIFFs. Each image (**Figure 3a**) is sequentially passed into a central preprocessing routine which first converts the image to a normalized 8-bit internal representation to improve memory and computational efficiency in exchange for an acceptable loss in precision. In order to extract features for every pixel of a given image, we then apply a static set of parametrized image filters to the input image. These are common image filters as implemented in OpenCV2 [102], SciPy [103], and the Python Imaging Library (PIL) and include gaussian smoothing, intensity thresholding, Sobel edge detection [104], Laplacian of Gaussian Edge Detection) [105], Structure Tensor Eigenvalues [106] and more (**Figure 4** and **Sup. Fig. 2**). These filters effectively extract new per-pixel features by computing intensity value changes to any given pixel as a non-linear transformation of the current pixel in the spatial context of the overall image with weighted contributions from neighboring pixels.

Our preprocessing function expands our original 2D single-band image of width *w* and height *h* into a 3D array with a feature depth *d* equal to the number of applied filters (**Figure 3b**). As it contains valuable feature information in its original form, we also retain the original 8-bit image as one of the feature layers. In total, we preprocess every 2D image into a 3D array with a depth of 80 features.

We sequentially preprocess every image in our training set in this way and *flatten* our

internal 3D representation of the training set data with extracted per-pixel features into a long 2D array that represents all of the pixels and all 80 features extracted per pixel (**Figure 3c**). We separately perform the same flattening operation with our binary training labels into a distinct internal data structure. While this data transformation is computationally expensive, this 2D internal representation of data points and associated extracted features is required for our scikit-learn and XGBoost model training functions. We then train our two types of models by fitting them to the training data (**Figure 3d**) using only basic hyperparameters (**Table 1**).

Table 1: *Hyperparameters for our ensemble machine learning models. We use ensembles of 100 trees and use multi-threading for improved performance on multi-CPU systems. For XGBoost, we explicitly specify a maximum decision tree depth of 5 and we subsample training data to further mitigate overfitting.*

Random Forest	
Number of Estimators	100
Number of Threads	25
XGBoost	
Number of Estimators	100
Number of Threads	25
Max Depth	5
Subsample	0.5

The model training output is an optimized model that consists of an ensemble of optimized decision trees (**Figure 3e**) that fits our raw training images to the provided binary training labels. Each optimized decision tree with the trained model are represented by internal nodes, each corresponding to a specific feature type along with an associated quantitative expression that evaluates into a left-or-right bifurcating branching decision. Decisions are therefore represented as branches between nodes in the decision tree. The order of the internal nodes and the branching decisions related to those internal nodes were optimized during model training. Because of the compute-intensive data flattening step needed prior to model fitting (**Figure 5**), the computational complexity of our overall random forest and XGBoost training program is quadratic at $O(n^2)$, where n is the total number of input pixels. Trained models are written to disk for later use in classification (semantic segmentation) of new images. These stored machine learning models can also be inspected to report metadata such as relative feature importance (**Supplementary Figures, Sup. Fig. 2**).

During the classification step, archived trained models are read from disk into memory and new raw images are individually preprocessed using the same collection of image processing filters that were employed for feature extraction prior to model training. Similarly, the preprocessed input image data is flattened prior to use by the trained random forest or XGBoost classifiers. Each pixel and its corresponding vector of extracted features is passed through each decision tree within a model. This is repeated as the pixel feature vector traverses the decision tree from the root down to a single labeled leaf node (**Figure 3g**). This process is repeated for every decision tree in the model, each of which built in a unique way. The final classification for any given pixel is

made by performing a simple majority vote across all decision trees in the model (**Figure 3f**). Features within any given tree that contribute more to classification accuracy will occur closer to the root of the tree. Averaging the relative depth of a feature type across all trees in the ensemble approximates the relative importance of that feature to the overall classifier.

Convolutional Neural Networks with the U-Net Architecture

In recent years, deep learning methods such as convolutional neural networks (CNNs) have demonstrated remarkable accuracy in difficult computer vision applications such as image classification [107-115]. More recently, powerful CNN architectures have been introduced to perform semantic segmentation of images [116-118] . Originally developed for biomedical image analysis, U-Net is one of the most widely utilized, studied, and cited CNN architectures for image segmentation [119]. We implement U-Net as a deep learning method for performing binary classification of LDs in images of *Y. lipolytica* cells and quantitatively and qualitatively compare accuracy and computational speed to our results from the ensemble machine learning methods discussed earlier.

The U-Net layout is *fully convolutional* and therefore has no fully connected layers (**Figure 6**). The first layer starts with an original image of a fixed dimensionality, in our case this is a 256x256 single-channel grayscale image. As our *Y. lipolytica* image library contains images of differing widths and heights, we first pad all of our images to be a consistent 256x256 size. The next two convolutional layers are configured to apply 64

image filters using a 3x3 kernel that convolves over the image data from the previous layer. We specifically use padding in our convolutional layers to preserve the original dimensions of the previous layer. This convolution process of sliding a 3x3 kernel over our image data extracts features in a way that is analogous to our use of image filters in the ensemble methods described earlier in this paper. The result of the first three layers is a 256x256x64 *feature map tensor* that represents the original image dimensions, encoded now with 64 extracted features. This tensor is now passed through a 2x2 max pool layer that down-samples the original image width and height by a factor of 2, leading to a 128x128x64 tensor. This is then passed through additional image filters and down-sampled again. This pattern is repeated several times, reducing the tensor's width and height while greatly expanding its feature depth. At the lowest level of the U-Net, we apply dropout layers to randomly subsample from the prior convolutional layer in an attempt to mitigate model overfitting to our training data [120].

We then begin to scale up our tensors using 2x2 up-sampling layers which are followed immediately by concatenation with the tensors from prior layers at the same depth. This concatenation step after every up-sampling layer ultimately combines important spatially-correlated elements of the original 2D image with the more deeply transformed feature-rich tensors. This pattern of applying convolutional layers, upsampling, and concatenation continues until we result in a tensor with our original image 256x256 image dimensions. After a couple additional convolutional layers, our final output layer is implemented as a 2D convolutional layer with a sigmoid activation function as a continuous approximation to a binary step function. When using the trained model during classification, we apply a simple intensity threshhold cutoff (0.5) to post-process

the continuous output values from the model to final binary classification output.

As with our prior tests, we used different training set sizes between 1,000 and 5,000 training images and we trained each model for 1000 epochs. The learning curve, or rate of change of the loss function[1], was very rapid with the U-Net CNN (**Figure 7**). Notably, a training set size of 1000 images resulted in a shallower learning curve that prematurely reached stationarity within localized optima at approximately 500 epochs. The classification output from models trained with only 1000 images never classified *any* inputs as LDs, and we determined that a non-augmented training set size of 1000 or less was too small for these inputs. However, CNN models trained with 2,000 or more training images provided steeper learning curves and improved accuracy scores.

As an illustrative exercize to visualize the incremental improvements made by the U-Net CNN during the training phase, we instrumented our Python framework to save trained models after every epoch and perform a classification on test input (**Figure 8**). This qualitative visualization of the rate of improvement correlates to our quantitative learning curves (**Figure 7**), as there is exceptionally rapid improvement within just the first few epochs. By just the tenth epoch, it is visually obvious that in most cases the model is already closely approximating our binary labels for our test data. However, our learning curves indicate that our models continue to asymptotically improve across all 1,000 epochs when the training set contains 2,000 or more images. We found that in most cases, relatively small improvements in binary cross entropy loss resulted in noticeable improvements to Dice accuracy scores.

---

[1] Our CNN uses *binary cross entropy loss*, a common loss function for binary classification problems.

Results

We constructed our tests to provide a consistent framework to compare our three methods: Random forest, XGBoost, and U-Net convolutional neural networks. With some notable exceptions, all three methods performed similarly from a quantitative accuracy standpoint. We used the same stochastically grouped training and test sets for each method as shown earlier in **Figure 2** and used the same or analogous hyperparameters across models wherever possible (**Table 1**). While a number of metrics are available for quantifying image segmentation accuracy, we settled on using the commonly used Sørensen-Dice (*i.e.* "Dice" [65-68]) score to report model classification accuracy against our test set. Our scoring function computes an image's Dice score as:

$$Dice = \frac{2TP}{2TP + FP + FN}$$

Where *TP*, *FP*, and *FN* represent the number of accumulated true positives, false positives, and false negatives respectively when doing a pixel-by-pixel comparison between a "true" labeled image and binary classified output from our machine learning models.

While random forest and XGBoost were able to successfully train useful semantic

segmentation classifiers with a small non-augmented training set of only 1,000 images, we found that the deep learning CNN method can become trapped in local optima during gradient descent optimization and produce a classifier that consistently produced blank output (i.e. no true *or* false positives). The lack of any positives resulted in uninterpretable Dice scores. Because of the high ratio of non-LD pixels in these images, it is likely that the neural network partially optimized the classifier to always predict non-LD pixels, resulting in a reasonably low but far from globally optimal binary cross entropy loss during model training. We fully expect that this could be mitigated by simply augmenting our data during training to synthetically increase our pool of training images.

In all other cases, the deep learning U-Net CNN method produced superior accuracy scores over random forest and XGBoost (**Figure 9**). In all tests, median Dice accuracy scores ranged in an exceptionally narrow band between 0.87 and 0.89. In general, XGBoost produced slightly more accurate classifiers than random forest and all methods improved as training set size increased, with the notable exception of the training set size of 5,000 images. With this relatively high number of training images, both U-Net CNN and XGBoost Dice scores declined slightly, likely indicative of model overfitting.

In addition to quantifying the relative accuracy of the different machine learning methods under different training set sizes, we also characterized the computational complexity and performance of these methods as a function of training set size. We demonstrated earlier in **Figure 5** that the random forest and XGBoost methods were $O(n^2)$ with respect to the number of input pixels due to the expensive but required internal data re-

structuring prior model training. Using the Keras framework for our CNN implementation, we instead stream our training data though our model during training, resulting in a linear *O(n)* complexity *(***Figure 10***).* Despite the improved asymptotic scalability of our CNN implementation, it takes considerably less wall-clock time to train ensemble methods such as random forest or XGBoost due to the high fixed complexity of the U-Net architecture. In fact, we found that the only practical way to train our U-Net CNN in a reasonable timeframe was to use Keras [121] on a GPU-enabled TensorFlow backend [122].

While training was computationally expensive, all of our tested supervised machine learning methods produced models which could perform full semantic segmentation on new images in a fraction of a second per image (**Figure 11**). Unsurprisingly, the U-Net CNN classifier implemented on a GPU accelerator significantly outperformed all conventional CPU-based methods with a median time of 15.4 msec per image. XGBoost classifiers were also relatively fast, largely due to the shallow depth of the individual decision trees within the model, leading to fast classification of individual pixels. The slowest method tested was the U-Net CNN implemented on a conventional CPU, certainly due to the fixed high complexity of the U-Net CNN architecture. In all, each of these methods took less than half a second per full image segmentation, and all methods are practical candidates for integration into a complete high-throughput optical imaging and analysis workflow.

Discussion

We tested three different supervised machine learning semantic segmentation methods to classify pixels that correspond to LDs within images of *Y. lipolytica* cells. We assessed and compared classification accuracy and computational efficiency as a function of training set size. From a purely quantitative standpoint, these methods performed similarly with some noticeable differences. First, we found that a convolutional neural network using the popular U-Net architecture performed better than simpler ensemble methods, but required a sufficient training set size greater than 1,000 images. In reality, this is unlikely to be a significant problem for U-Net CNN as training set sizes of 2,000 and above were sufficiently rich to develop a robust classifier and synthetic data augmentation would almost certainly suffice to construct enough training data.

In general, classification accuracy increased as a function of training set size until our training set exceeded 5,000 images at which point there was a slight drop in classification accuracy scores for both the CNN and XGBoost methods. In general, adding more training data mitigates overfitting; it is possible that this case was a minor anomaly and adding even more training data would result in modest increases in accuracy. It is also plausible that the specific additional 1,000 images did not have a sufficiently high signal-to-noise ratio to improve model generalization. However, in terms of quantitative Dice accuracy scores, all tested methods performed remarkably similarly for training set sizes of 2,000 images or more.

Despite the similar median Dice scores, we found that the deep learning CNN method provided more biologically interpretable and realistic image segmentations. We reviewed the output of the trained models for all three methods to elucidate the

qualitative differences between the methods (**Figure 12**). While these are hand-picked demonstrative examples, it is clear that the U-Net CNN more often classifies LDs as round, smooth, continuous objects that are biologically interpretable as realistic and monolithic organelles. Random forest and XGBoost do create similar output to U-Net CNN in most cases, but are more likely to produce noisy mis-classifications with rough and irregular edges, spurious single-pixel misclassifications, or holes and openings within otherwise complete and round droplets. In addition to the higher Dice accuracy, this qualitative advantage for the U-Net CNN is important to consider when deciding which method to adopt.

Because of the fixed complexity of the U-Net model, we found that random forest and XGBoost were much faster to train despite their greater asymptotic computational complexity as a function of the number of training images. In fact, training a U-Net CNN using multicore CPUs was impracticaly slow for larger training sets. However, using GPU support with our Keras/TensorFlow implementation led to 40X speedups in both training and classifications time. Thus, using a U-Net CNN executed on modern GPUs to classify LDs produces the fastest, most accurate and interpretable segmentations in QPI images.

Figure Captions

Figure 1*: Our supervised machine learning system can classify lipid droplets within 32-bit grayscale raw images of individual Y. lipolytica cells (a). Curated binary training images (b) are used to train machine learning classifier models that can successfully classify lipid pixels given new image data (c). We assess classification accuracy by comparing (c) to (b) using a number of metrics including normalized template cross-correlation and the Sørensen-Dice coefficient (i.e. "Dice scores" [65-68]). While this classification example looks reasonably accurate, it has a relatively low Dice accuracy score of 0.68.*

Figure 2*: The overall method design. We use approximately 7,000 images and randomly choose 5000 images as a training pool and 2000 as our test set. From the 5000 images in our training pool, we then randomly construct training sets of size 1000, 2000, 3000, 4000, and 5000. With these training sets, we train three types of machine learning models: convolutional neural networks (CNN), gradient boosting models with XGBoost, and random forest models. For CNNs, we train for 1000 epochs[2]. For the other methods we use a collection of 100 decision trees/stumps. During training, we use binary labeled data associated with each raw image to train our models. Each method outputs a segmentation map which can then be compared to the binary labeled template to produce an accuracy score (e.g. Sorensen-Dice coefficient).*

Figure 3: *Training the random forest or XGBoost models and classifying new inputs with the trained models.* ***Training Step****: Each raw image in the training set (**a**) is*

[2] An *epoch* is one full pass of the training data set through the neural network model during training.

*expanded into a set of k images through the application of image filters (see **Figure 4** below) in order to extract informative features for each pixel (**b**). Each pixel is now described not by a single intensity value, but by a feature vector of length k. In our configuration, k=80 features. We then reorganize these data into a long two-dimensional array (**c**) wherein rows represent all pixels in the training set, and all columns represent an extracted feature for that pixel. This n x k matrix, along with corresponding binary labels for each pixel, is passed as training input to a random forest or XGBoost model fit function (**d**). The output of the training procedure is either a trained random forest or XGBoost model (**e**), which is an ensemble of optimized decision trees.* ***Classification Step****: When classifying new raw images, we repeat steps (**a**), (**b**), and (**c**) for each input image to build an n x k matrix of pixels and features. The feature vector for each pixel is evaluated by each of the j decision trees in our ensemble and a simple majority vote across all trees determines the final classification (**f**) for any given pixel. To classify a pixel within an individual decision tree (**g**), the pixel's features are evaluated by simple learned decision expressions at each internal node. Each pixel then follows a path of decisions through the tree until we traverse to a leaf node corresponding to a classification label (e.g. "LD" or "not LD").*

Figure 4: *A subset of the image filters used for feature extraction. We apply a number of common digital image filters such as Gaussian smoothing and Sobel edge detection to each input image, using multiple σ parameter values. This extracts a total of k features per pixel. These features are then used for either training or classification with random forest or XGBoost machine learning methods.*

Figure 5: *Computational time for training a random forest model is a superlinear function of training set size.*

Figure 6: *The U-Net convolutional neural network (CNN) architecture was developed specifically for semantic segmentation. Our grayscale input image is consistently padded to 256x256. This image os passed through convolutional layers using a Rectified Linear Unit ("ReLU") activation function with a 3x3 kernel. Each convolutional layer specifies padding to prevent gradual image shrinkage. The original image (now transformed into a feature map tensor) passes through a series of convolutional and 2x2 max pool layers until the tensor is finally reduced to 16x16 with a feature depth of 1024. At the lowest levels, we perform a 0.5 dropout to mitigate overfitting. We then iteratively up-sample (2x2) the tensor and perform a second dropout while concatenating it with the earlier tensor of the same dimension at the same level. We perform this same concatenation operation at every up-sample layer. The final convolutional output layer uses a sigmoid activation to approximate a binary classification for each pixel.*

Figure 7: *The U-Net CNN learning curves for different sizes of training sets. In neural networks, a learning curve is the rate of model improvement during training for the chosen loss function. Here, we use a binary cross entropy loss function, as is common with binary classification problems. With a small training set size of 1000, the learning curve for U-Net CNN is smooth and gradual but often becomes trapped in local optima: we enter stationarity at around 500 epochs, but our loss scores never approach those for larger training sets. The learning curves for training set sizes over 2000 are very similar as they approach zero loss at a similar rate. Given that the model trained using a*

large training set size of 5000 scored worse than other models built with smaller training sets (possibly due to chance or model overfitting, see ***Figure 10****). For this binary segmentation problem with these data, a training set size of 2000 images may produce the best balance of accuracy vs. computational speed.*

Figure* 8*: *Watching the U-Net convolutional network learn. This figure demonstrates how the model noticeably improves very early during the initial 10 epochs of training. While we trained our models to 1000 epochs, we show that even by epoch 10 with a training set size of 5000 images, our model has already started approximating the true image segmentation (as shown in the rightmost column).*

Figure 9: *A quantitative accuracy comparison of our three machine learning methods as a function of training set size. Here we compute the median Sørensen-Dice coefficient (i.e. "Dice" score) given by each method for each training set size. Note that our CNN was unable to consistently train an effective classifier with only 1000 non-augmented training images; however, the deep learning CNN method otherwise consistently outperformed other machine learning classifiers. XGBoost generally outperformed random forest. Overall, in absolute terms as shown by the range on the y-axis, the quantitative differences between methods is minimal.*

Figure 10*: The time required to train one epoch as a function of training set size. For neural networks, an epoch is defined as one full pass of the training set through the model. Using the Keras Python framework, we stream our training data through our U-Net CNN model in batches, and thus the time required to train per epoch is essentially a linear function of the training set size. In this case, we trained using Keras/TensorFlow*

on a consumer-level GPU (Nvidia GeForce GTX 1080 Ti).

Figure 11: *Classifying all pixels within an image using a trained model is relatively fast across all machine learning methods used. The fastest approach was the U-Net CNN executed on a Nvidia 1080 Ti GPU, with a median time per image segmentation of 15.4 msec. XGBoost classifiers were also relatively fast at a median rate of 76 msec per image segmentation. Random forest classifiers took a median time of 181 msec per image, while U-Net classifiers implemented on a CPU took significantly longer at a median time of approximately 484 msec per image.*

Figure 12*: Qualitative argument for the use of the U-Net CNN. While random forest or XGBoost can sometimes score similarly to deep learning methods such as the U-Net CNN, we found that deep learning methods produces smoother and more biologically interpretable segmentations in almost all cases. Computed Dice scores are shown above the images. These images are acute examples where random forest and XGBoost produced reasonably high scoring but qualitatively unrealistic or noisy classifications of lipid droplets. Among other reasons, this is likely because the U-Net CNN directly persists and integrates original 2D spatial information while building the segmentation map. With the random forest and XGBoost methods, this 2D information is only indirectly inferred in a lossy way via the particular image filters used during feature extraction.*

References

1. Park, Y., C. Depeursinge, and G. Popescu, *Quantitative phase imaging in biomedicine.* Nature Photonics, 2018. **12**(10): p. 578-589.

2. Majeed, H., et al., *Quantitative phase imaging for medical diagnosis.* Journal of Biophotonics, 2017. **10**(2): p. 177-205.

3. Gjörloff-Wingren, A., *Quantitative phase-contrast imaging—A potential tool for future cancer diagnostics.* Cytometry Part A, 2017. **91**(8): p. 752-753.

4. Jin, D., et al., *Tomographic phase microscopy: principles and applications in bioimaging [Invited].* Journal of the Optical Society of America B, 2017. **34**(5): p. B64-B77.

5. Lee, K.C.M., et al., *Quantitative Phase Imaging Flow Cytometry for Ultra-Large-Scale Single-Cell Biophysical Phenotyping.* Cytometry Part A, 2019. **95**(5): p. 510-520.

6. Barer, R., *Interference Microscopy and Mass Determination.* Nature, 1952. **169**(4296): p. 366-367.

7. Choi, W., et al., *Tomographic phase microscopy.* Nat Methods, 2007. **4**(9): p. 717-9.

8. Wolf, E., *Three-dimensional structure determination of semi-transparent objects from holographic data.* Optics communications, 1969. **1**(4): p. 153-156.

9. Zangle, T.A. and M.A. Teitell, *Live-cell mass profiling: an emerging approach in quantitative biophysics.* Nature Methods, 2014. **11**(12): p. 1221-1228.

10. Alanazi, H., et al., *Robust microbial cell segmentation by optical-phase thresholding with minimal processing requirements.* Cytometry A, 2017. **91**(5): p. 443-449.

11. Nguyen, T.H., et al., *Halo-free Phase Contrast Microscopy.* Scientific Reports, 2017. **7**(1): p. 44034.

12. Rivenson, Y., Y. Wu, and A. Ozcan, *Deep learning in holography and coherent imaging.* Light: Science & Applications, 2019. **8**(1): p. 85.

13. Jo, Y., et al., *Quantitative Phase Imaging and Artificial Intelligence: A Review.* IEEE Journal of Selected Topics in Quantum Electronics, 2019. **25**(1): p. 1-14.

14. Deng, M., et al., *Learning to synthesize: robust phase retrieval at low photon counts.* Light: Science & Applications, 2020. **9**(1): p. 36.

15. Yoo, J., et al., *Deep learning diffuse optical tomography.* IEEE transactions on medical imaging, 2019.

16. Sun, Y., Z. Xia, and U.S. Kamilov, *Efficient and accurate inversion of multiple scattering with deep learning.* Optics express, 2018. **26**(11): p. 14678-14688.

17. Kamilov, U., et al., *Learning approach to optical tomography.* Optica, 2015. **2**: p. 517-522.

18. Lim, J., et al., *High-fidelity optical diffraction tomography of multiple scattering samples.* Light: Science & Applications, 2019. **8**(1): p. 82.

19. Rivenson, Y., et al., *PhaseStain: the digital staining of label-free quantitative phase microscopy images using deep learning.* Light: Science & Applications, 2019. **8**(1): p. 23.

20. Zhang, J.K., et al., *Label-free colorectal cancer screening using deep learning and spatial light interference microscopy (SLIM).* APL Photonics, 2020. **5**(4): p. 040805.

21. Kim, G., et al., *Learning-based screening of hematologic disorders using quantitative phase imaging of individual red blood cells.* Biosensors and Bioelectronics, 2019. **123**: p. 69-76.

22. Singh, V., V. Srivastava, and D.S. Mehta, *Machine learning-based screening of red blood cells using quantitative phase imaging with micro-spectrocolorimetry.* Optics & Laser Technology, 2020. **124**: p. 105980.

23. Lee, J., et al., *Deep-Learning-Based Label-Free Segmentation of Cell Nuclei in Time-Lapse Refractive Index Tomograms.* IEEE Access, 2019. **7**: p. 83449-83460.

24. Martin, S. and R.G. Parton, *Lipid droplets: a unified view of a dynamic organelle.* Nature reviews Molecular cell biology, 2006. **7**(5): p. 373-378.

25. Thiam, A.R., R.V. Farese Jr, and T.C. Walther, *The biophysics and cell biology of lipid droplets.* Nature Reviews Molecular Cell Biology, 2013. **14**(12): p. 775-786.

26. Hashemi, H.F. and J.M. Goodman, *The life cycle of lipid droplets.* Current opinion in cell biology, 2015. **33**: p. 119-124.

27. Wang, C.W., *Lipid droplets, lipophagy, and beyond.* Biochim Biophys Acta, 2016. **1861**(8 Pt B): p. 793-805.

28. Garcia, E.J., J.D. Vevea, and L.A. Pon, *Lipid droplet autophagy during energy mobilization, lipid homeostasis and protein quality control.* Front Biosci (Landmark Ed), 2018. **23**: p. 1552-1563.

29. Ploegh, H.L., *A lipid-based model for the creation of an escape hatch from the endoplasmic reticulum.* Nature, 2007. **448**(7152): p. 435-8.

30. Cermelli, S., et al., *The lipid-droplet proteome reveals that droplets are a protein-storage depot.* Curr Biol, 2006. **16**(18): p. 1783-95.

31. Welte, M.A., *Expanding roles for lipid droplets.* Current biology, 2015. **25**(11): p. R470-R481.

32. Ueno, M., et al., *Fat-specific protein 27 modulates nuclear factor of activated T cells 5 and the cellular response to stress.* J Lipid Res, 2013. **54**(3): p. 734-43.

33. Li, Z., et al., *Lipid droplets control the maternal histone supply of Drosophila embryos.* Curr Biol, 2012. **22**(22): p. 2104-13.

34. Saka, H.A. and R. Valdivia, *Emerging roles for lipid droplets in immunity and host-pathogen interactions.* Annual review of cell and developmental biology, 2012. **28**: p. 411-437.

35. Roingeard, P. and R.C. Melo, *Lipid droplet hijacking by intracellular pathogens.* Cell Microbiol, 2017. **19**(1).

36. Miyanari, Y., et al., *The lipid droplet is an important organelle for hepatitis C virus production.* Nat Cell Biol, 2007. **9**(9): p. 1089-97.

37. Samsa, M.M., et al., *Dengue virus capsid protein usurps lipid droplets for viral particle formation.* PLoS Pathog, 2009. **5**(10): p. e1000632.

38. Gluchowski, N.L., et al., *Lipid droplets and liver disease: from basic biology to clinical implications.* Nat Rev Gastroenterol Hepatol, 2017. **14**(6): p. 343-355.

39. Cruz, A.L.S., et al., *Lipid droplets: platforms with multiple functions in cancer hallmarks.* Cell Death & Disease, 2020. **11**(2): p. 105.

40. Stephanopoulos, G., *Challenges in engineering microbes for biofuels production.* Science, 2007. **315**(5813): p. 801-4.

41. Fukuda, H., A. Kondo, and H. Noda, *Biodiesel fuel production by transesterification of oils.* Journal of Bioscience and Bioengineering, 2001. **92**(5): p. 405-416.

42. Scott, S.A., et al., *Biodiesel from algae: challenges and prospects.* Current Opinion in Biotechnology, 2010. **21**(3): p. 277-286.

43. Vasdekis, A.E., A.M. Silverman, and G. Stephanopoulos, *Origins of Cell-to-Cell Bioprocessing Diversity and Implications of the Extracellular Environment Revealed at the Single-Cell Level.* Scientific Reports, 2015. **5**(1): p. 17689.

44. Herms, A., et al., *Cell-to-cell heterogeneity in lipid droplets suggests a mechanism to reduce lipotoxicity.* Curr Biol, 2013. **23**(15): p. 1489-96.

45. Coradetti, S.T., et al., *Functional genomics of lipid metabolism in the oleaginous yeast Rhodosporidium toruloides.* Elife, 2018. **7**.

46. Wilfling, F., et al., *Triacylglycerol synthesis enzymes mediate lipid droplet growth by relocalizing from the ER to lipid droplets.* Dev Cell, 2013. **24**(4): p. 384-99.

47. Daemen, S., et al., *Microscopy tools for the investigation of intracellular lipid storage and dynamics.* Molecular Metabolism, 2016. **5**(3): p. 153-163.

48. Hu, F., L. Shi, and W. Min, *Biological imaging of chemical bonds by stimulated Raman scattering microscopy.* Nature Methods, 2019. **16**(9): p. 830-842.

49. Nan, X., J.X. Cheng, and X.S. Xie, *Vibrational imaging of lipid droplets in live fibroblast cells with coherent anti-Stokes Raman scattering microscopy.* J Lipid Res, 2003. **44**(11): p. 2202-8.

50. Greenspan, P., E.P. Mayer, and S.D. Fowler, *Nile red: a selective fluorescent stain for intracellular lipid droplets.* Journal of Cell Biology, 1985. **100**(3): p. 965-973.

51. Noothalapati Venkata, Hemanth N. and S. Shigeto, *Stable Isotope-Labeled Raman Imaging Reveals Dynamic Proteome Localization to Lipid Droplets in Single Fission Yeast Cells.* Chemistry & Biology, 2012. **19**(11): p. 1373-1380.

52. Jung, J., et al., *Label-free non-invasive quantitative measurement of lipid contents in individual microalgal cells using refractive index tomography.* Scientific Reports, 2018. **8**(1): p. 6524.

53. Vasdekis, A.E., et al., *Eliciting the impacts of cellular noise on metabolic trade-offs by quantitative mass imaging.* Nature Communications, 2019. **10**(1): p. 848.

54. Tai, M. and G. Stephanopoulos, *Engineering the push and pull of lipid biosynthesis in oleaginous yeast Yarrowia lipolytica for biofuel production.* Metabolic Engineering, 2013. **15**: p. 1-9.

55. Blazeck, J., et al., *Harnessing Yarrowia lipolytica lipogenesis to create a platform for lipid and biofuel production.* Nature communications, 2014. **5**(1): p. 1-10.

56. Beopoulos, A., et al., *Control of lipid accumulation in the yeast Yarrowia lipolytica.* Appl. Environ. Microbiol., 2008. **74**(24): p. 7779-7789.

57. Tin Kam, H. *Random decision forests*. in *Proceedings of 3rd International Conference on Document Analysis and Recognition*. 1995.

58. Breiman, L., *Random Forests.* Machine Learning, 2001. **45**(1): p. 5-32.

59. Friedman, J.H., *Stochastic gradient boosting.* Computational Statistics & Data Analysis, 2002. **38**(4): p. 367-378.

60. Friedman, J.H., *Greedy function approximation: A gradient boosting machine.* Ann. Statist., 2001. **29**(5): p. 1189-1232.

61. Fukushima, K., *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.* Biological Cybernetics, 1980. **36**(4): p. 193-202.

62. Lecun, Y., et al., *Gradient-based learning applied to document recognition.* Proceedings of the IEEE, 1998. **86**(11): p. 2278-2324.

63. Khan, A., et al., *A survey of the recent architectures of deep convolutional neural networks.* Artificial Intelligence Review, 2020.

64. Krizhevsky, A., I. Sutskever, and G.E. Hinton. *Imagenet classification with deep convolutional neural networks*. in *Advances in neural information processing systems*. 2012.

65. Sørensen, T., et al., *A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons.* 1948.

66. Dice, L.R., *Measures of the Amount of Ecologic Association Between Species.* Ecology, 1945. **26**(3): p. 297-302.

67. Bertels, J., et al. *Optimizing the Dice Score and Jaccard Index for Medical Image Segmentation: Theory and Practice*. 2019. Cham: Springer International Publishing.

68. Fidon, L., et al. *Generalised Wasserstein Dice Score for Imbalanced Multi-class Segmentation Using Holistic Convolutional Networks*. 2018. Cham: Springer International Publishing.

69. Liu, X., Z. Deng, and Y. Yang, *Recent progress in semantic image segmentation.* Artificial Intelligence Review, 2019. **52**(2): p. 1089-1106.

70. Kim, T., et al., *White-light diffraction tomography of unlabelled live cells.* Nature Photonics, 2014. **8**(3): p. 256.

71. Chen, T. and C. Guestrin. *Xgboost: A scalable tree boosting system*. in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016.

72. Berg, S., et al., *ilastik: interactive machine learning for (bio)image analysis.* Nature Methods, 2019. **16**(12): p. 1226-1232.

73. Gu, L., et al. *Semi-supervised Learning for Biomedical Image Segmentation via Forest Oriented Super Pixels(Voxels)*. 2017. Cham: Springer International Publishing.

74. Jog, A., et al., *Random forest regression for magnetic resonance image synthesis.* Medical Image Analysis, 2017. **35**: p. 475-488.

75. NEDJAR, I., et al., *RANDOM FOREST BASED CLASSIFICATION OF MEDICAL X-RAY IMAGES USING A GENETIC ALGORITHM FOR FEATURE SELECTION.* Journal of Mechanics in Medicine and Biology, 2015. **15**(02): p. 1540025.

76. Désir, C., et al. *A Random Forest Based Approach for One Class Classification in Medical Imaging*. 2012. Berlin, Heidelberg: Springer Berlin Heidelberg.

77. Bakas, S., et al. *GLISTRboost: Combining Multimodal MRI Segmentation, Registration, and Biophysical Tumor Growth Modeling with Gradient Boosting Machines for Glioma Segmentation*. 2016. Cham: Springer International Publishing.

78. Yang, T., W. Chen, and G. Cao, *Automated classification of neonatal amplitude-integrated EEG based on gradient boosting method.* Biomedical Signal Processing and Control, 2016. **28**: p. 50-57.

79. Lemaitre, G., et al., *A boosting approach for prostate cancer detection using multi-parametric MRI*. The International Conference on Quality Control by Artificial Vision 2015. Vol. 9534. 2015: SPIE.

80. Pal, M., *Random forest classifier for remote sensing classification.* International Journal of Remote Sensing, 2005. **26**(1): p. 217-222.

81. Belgiu, M. and L. Drăguţ, *Random forest in remote sensing: A review of applications and future directions.* ISPRS Journal of Photogrammetry and Remote Sensing, 2016. **114**: p. 24-31.

82. Feng, Q., J. Liu, and J. Gong, *UAV remote sensing for urban vegetation mapping using random forest and texture analysis.* Remote sensing, 2015. **7**(1): p. 1074-1094.

83. Gislason, P.O., J.A. Benediktsson, and J.R. Sveinsson, *Random Forests for land cover classification.* Pattern Recognition Letters, 2006. **27**(4): p. 294-300.

84. Georganos, S., et al., *Very High Resolution Object-Based Land Use–Land Cover Urban Classification Using Extreme Gradient Boosting.* IEEE Geoscience and Remote Sensing Letters, 2018. **15**(4): p. 607-611.

85. Moisen, G.G., et al., *Predicting tree species presence and basal area in Utah: A comparison of stochastic gradient boosting, generalized additive models, and tree-based methods.* Ecological Modelling, 2006. **199**(2): p. 176-187.

86. Lee, D., et al., *Attitudes on Autonomous Vehicle Adoption using Interpretable Gradient Boosting Machine.* Transportation Research Record, 2019. **2673**(11): p. 865-878.

87. Wang, Y., et al., *Stacking-based ensemble learning of decision trees for interpretable prostate cancer detection.* Applied Soft Computing, 2019. **77**: p. 188-204.

88. Chakraborty, S., et al. *Interpretability of deep learning models: A survey of results*. in *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. 2017.

89. Altmann, A., et al., *Permutation importance: a corrected feature importance measure.* Bioinformatics, 2010. **26**(10): p. 1340-1347.

90. Fontana, J.M., M. Farooq, and E. Sazonov. *Estimation of feature importance for food intake detection based on Random Forests classification*. in *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. 2013.

91. Mursalin, M., et al., *Automated epileptic seizure detection using improved correlation-based feature selection with random forest classifier.* Neurocomputing, 2017. **241**: p. 204-214.

92. Rao, H., et al., *Feature selection based on artificial bee colony and gradient boosting decision tree.* Applied Soft Computing, 2019. **74**: p. 634-642.

93. Pan, F., et al. *Feature selection for ranking using boosted trees*. in *Proceedings of the 18th ACM conference on Information and knowledge management*. 2009.

94. Kearns, M. and L. Valiant, *Cryptographic limitations on learning Boolean formulae and finite automata.* Journal of the ACM (JACM), 1994. **41**(1): p. 67-95.

95. Breiman, L., *Bagging predictors.* Machine Learning, 1996. **24**(2): p. 123-140.

96. Bühlmann, P. and B. Yu, *Analyzing bagging.* The Annals of Statistics, 2002. **30**(4): p. 927-961.

97. Dietterich, T.G., *An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization.* Machine Learning, 2000. **40**(2): p. 139-157.

98. Schapire, R.E., *The Boosting Approach to Machine Learning: An Overview*, in *Nonlinear Estimation and Classification*, D.D. Denison, et al., Editors. 2003, Springer New York: New York, NY. p. 149-171.

99. Ng, A.Y. *Feature selection, L 1 vs. L 2 regularization, and rotational invariance*. in *Proceedings of the twenty-first international conference on Machine learning*. 2004.

100. Dietterich, T., *Overfitting and undercomputing in machine learning.* ACM computing surveys (CSUR), 1995. **27**(3): p. 326-327.

101. Pedregosa, F., et al., *Scikit-learn: Machine learning in Python.* the Journal of machine Learning research, 2011. **12**: p. 2825-2830.

102. Zelinsky, A., *Learning OpenCV---Computer Vision with the OpenCV Library (Bradski, G.R. et al.; 2008)[On the Shelf].* IEEE Robotics & Automation Magazine, 2009. **16**(3): p. 100-100.

103. Virtanen, P., et al., *SciPy 1.0: fundamental algorithms for scientific computing in Python.* Nature Methods, 2020. **17**(3): p. 261-272.

104. Kanopoulos, N., N. Vasanthavada, and R.L. Baker, *Design of an image edge detection filter using the Sobel operator.* IEEE Journal of Solid-State Circuits, 1988. **23**(2): p. 358-367.

105. Sharifi, M., M. Fathy, and M.T. Mahmoudi. *A classified and comparative study of edge detection algorithms*. in *Proceedings. International Conference on Information Technology: Coding and Computing*. 2002.

106. Khan, A.R., et al., *3D structure tensor analysis of light microscopy data for validating diffusion MRI.* NeuroImage, 2015. **111**: p. 192-203.

107. Moeskops, P., et al. *Deep Learning for Multi-task Medical Image Segmentation in Multiple Modalities*. 2016. Cham: Springer International Publishing.

108. Kensert, A., P.J. Harrison, and O. Spjuth, *Transfer Learning with Deep Convolutional Neural Networks for Classifying Cellular Morphological Changes.* SLAS DISCOVERY: Advancing the Science of Drug Discovery, 2019. **24**(4): p. 466-475.

109. Nanni, L., S. Ghidoni, and S. Brahnam, *Ensemble of convolutional neural networks for bioimage classification.* Applied Computing and Informatics, 2018.

110. Giben, X., V.M. Patel, and R. Chellappa. *Material classification and semantic segmentation of railway track images with deep convolutional neural networks*. in *2015 IEEE International Conference on Image Processing (ICIP)*. 2015.

111. Soukup, D. and R. Huber-Mörk. *Convolutional Neural Networks for Steel Surface Defect Detection from Photometric Stereo Images*. 2014. Cham: Springer International Publishing.

112. Kang, L., et al. *Convolutional Neural Networks for Document Image Classification*. in *2014 22nd International Conference on Pattern Recognition*. 2014.

113. Gopalakrishnan, K., et al., *Deep Convolutional Neural Networks with transfer learning for computer vision-based data-driven pavement distress detection.* Construction and Building Materials, 2017. **157**: p. 322-330.

114. Slavkovikj, V., et al. *Hyperspectral image classification with convolutional neural networks*. in *Proceedings of the 23rd ACM international conference on Multimedia*. 2015.

115. Rawat, W. and Z. Wang, *Deep convolutional neural networks for image classification: A comprehensive review.* Neural computation, 2017. **29**(9): p. 2352-2449.

116. Audebert, N., B. Le Saux, and S. Lefèvre. *Semantic Segmentation of Earth Observation Data Using Multimodal and Multi-scale Deep Networks*. 2017. Cham: Springer International Publishing.

117. Guo, Y., et al., *A review of semantic segmentation using deep neural networks.* International Journal of Multimedia Information Retrieval, 2018. **7**(2): p. 87-93.

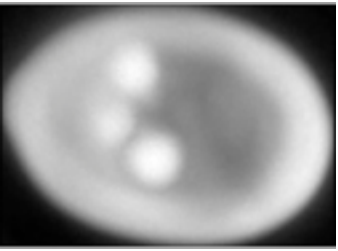
118. Badrinarayanan, V., A. Kendall, and R. Cipolla, *SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017. **39**(12): p. 2481-2495.

119. Ronneberger, O., P. Fischer, and T. Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. Cham: Springer International Publishing.

120. Srivastava, N., et al., *Dropout: a simple way to prevent neural networks from overfitting.* The journal of machine learning research, 2014. **15**(1): p. 1929-1958.

121. Chollet, F. and others. *https://github.com/fchollet/keras*. 2015.

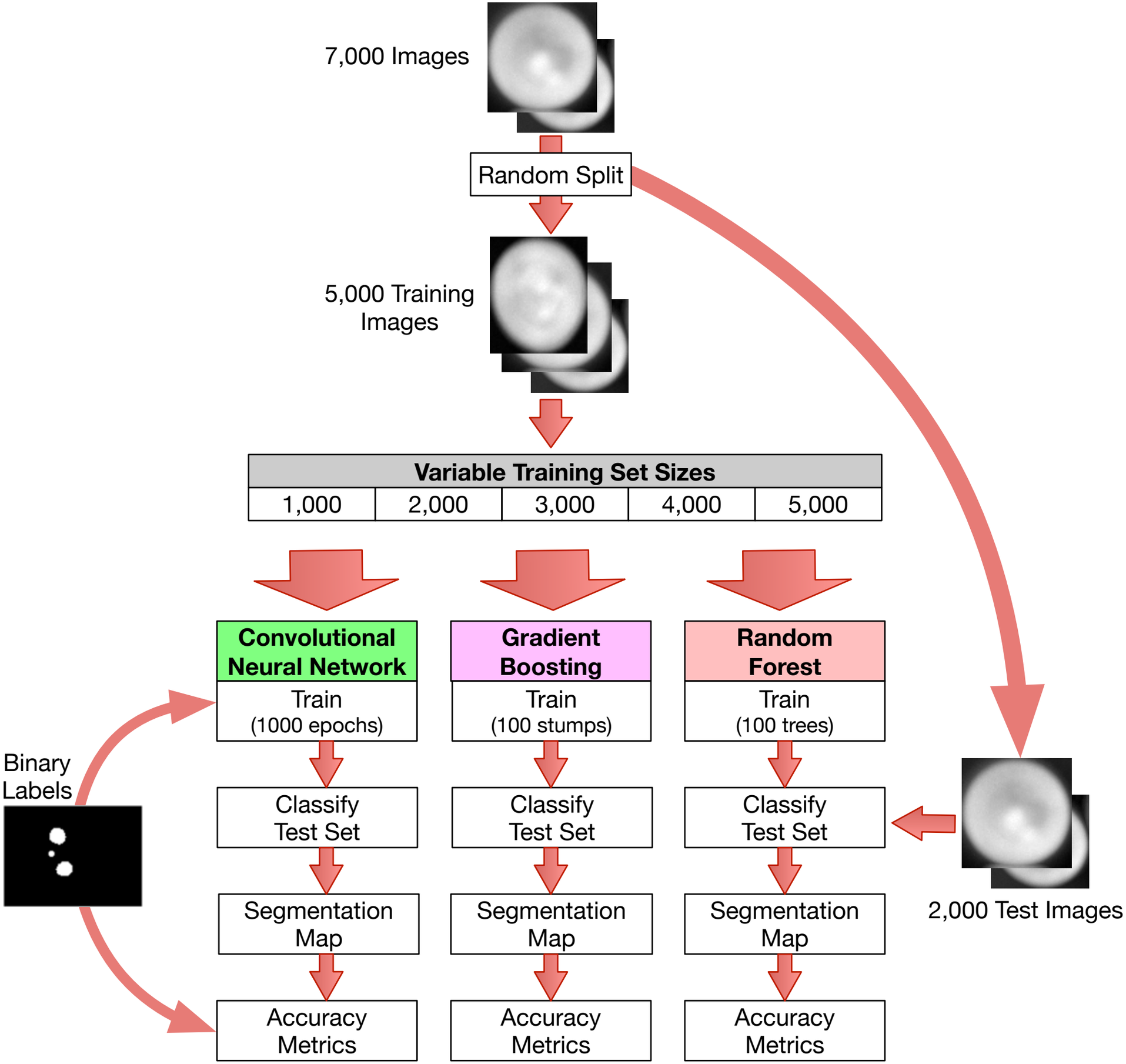
122. Pang, B., E. Nijkamp, and Y.N. Wu, *Deep Learning With TensorFlow: A Review.* Journal of Educational and Behavioral Statistics, 2020. **45**(2): p. 227-248.

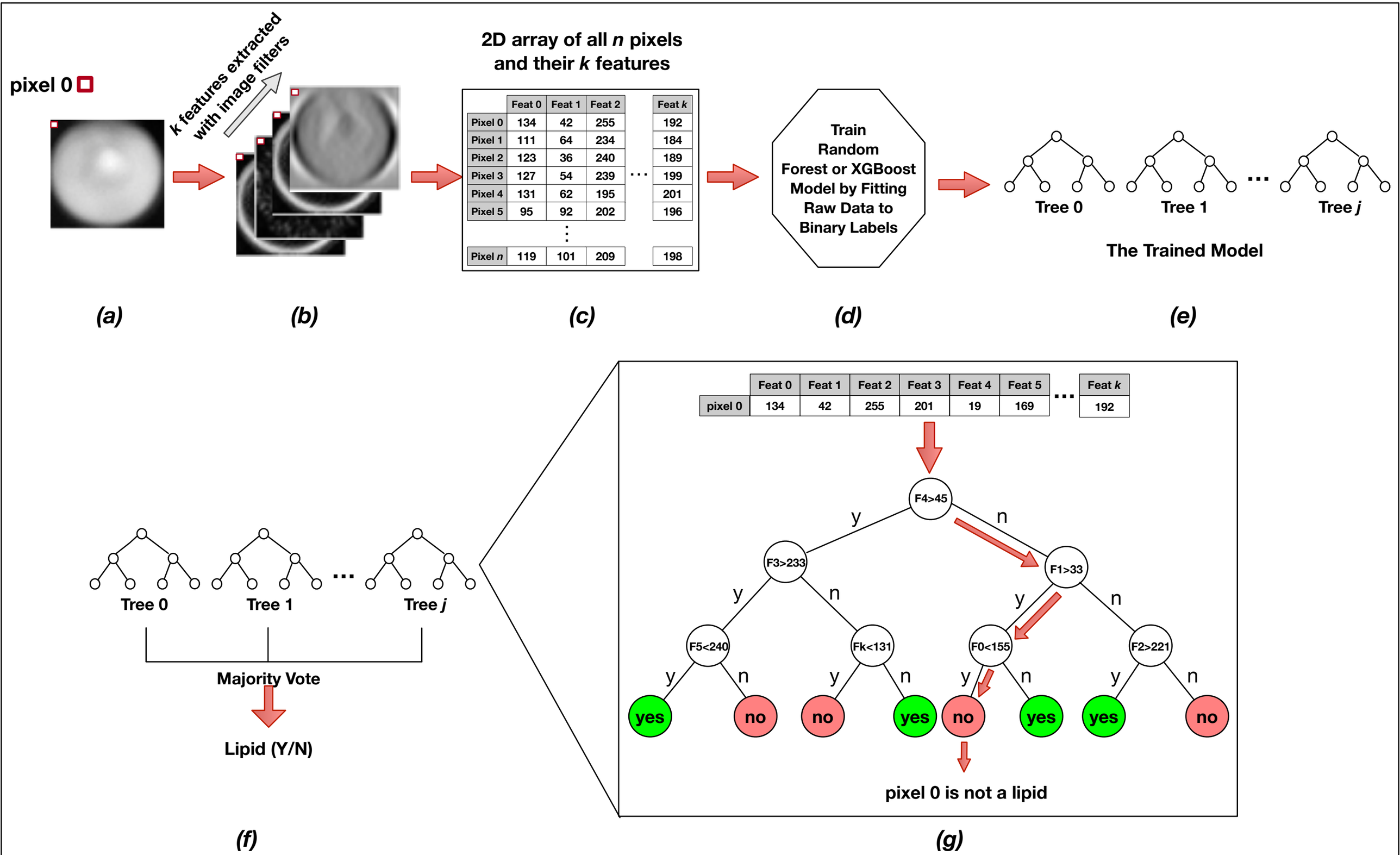


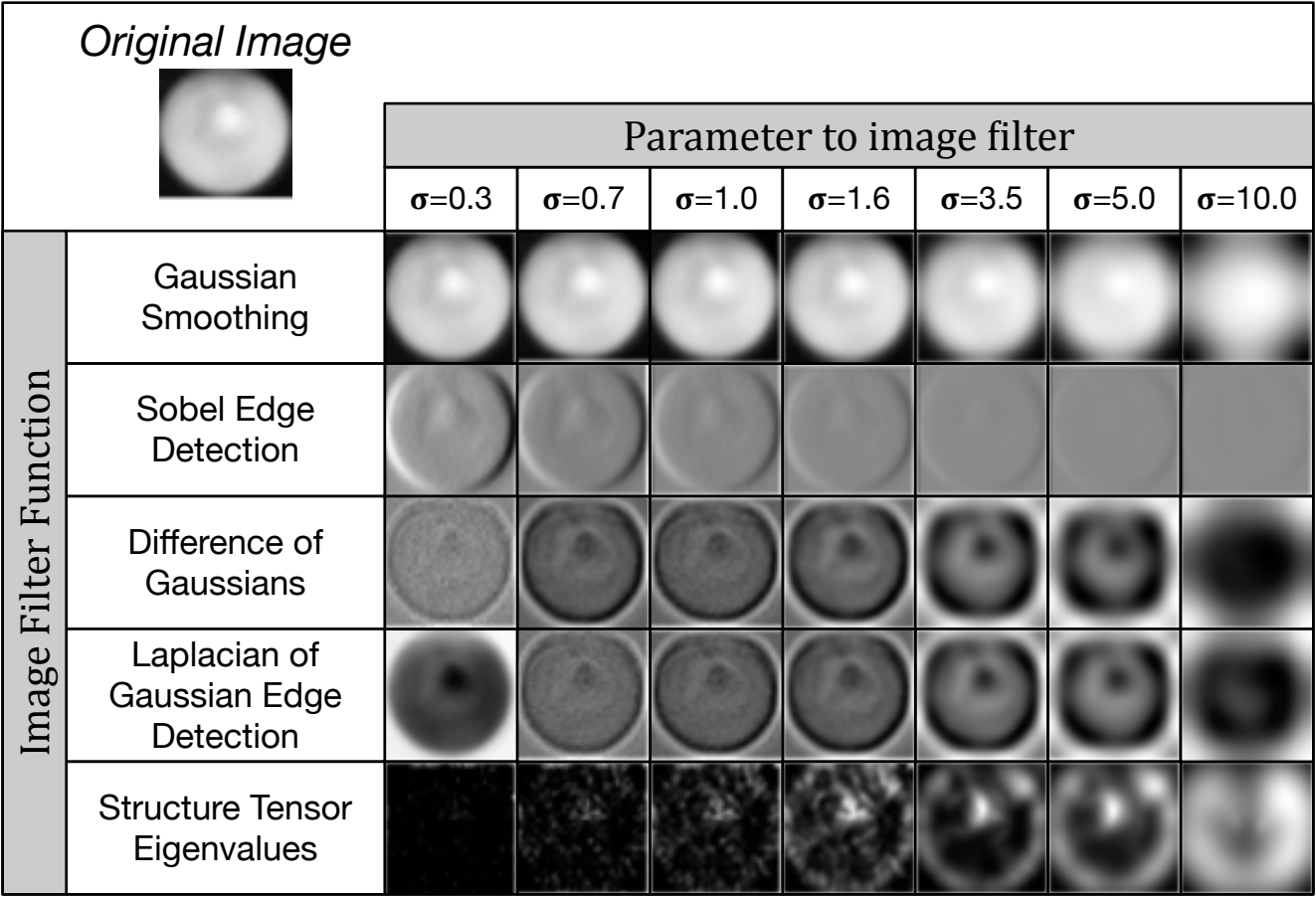
(*a*)

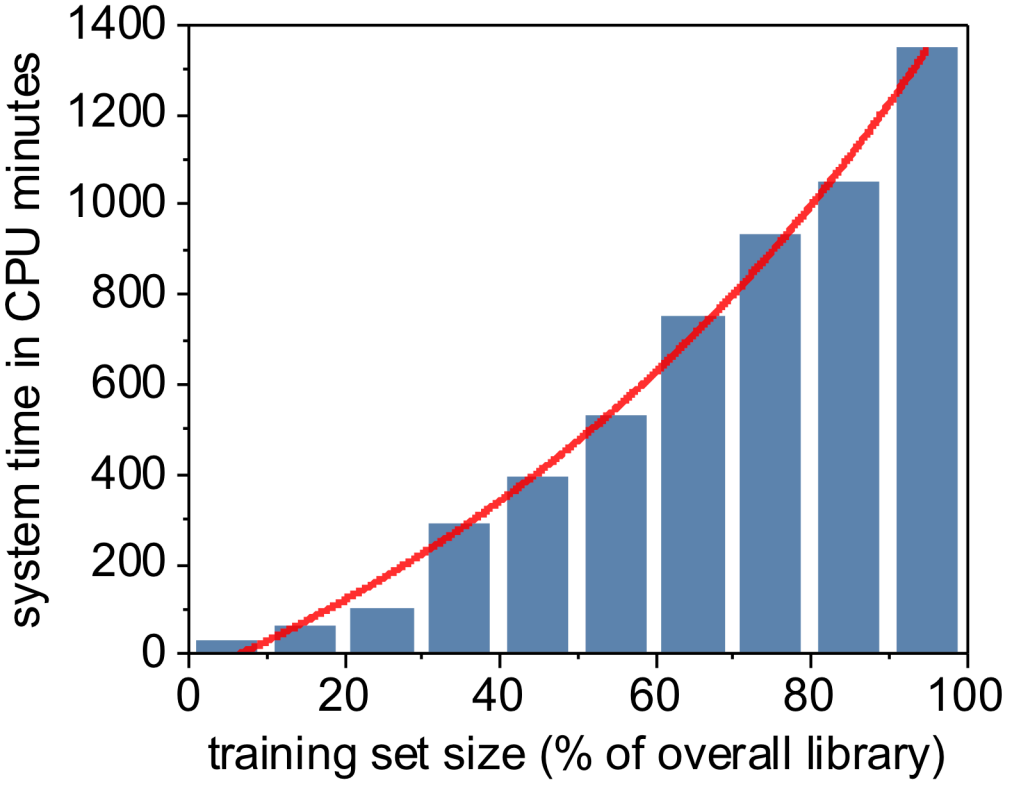
(*b*)

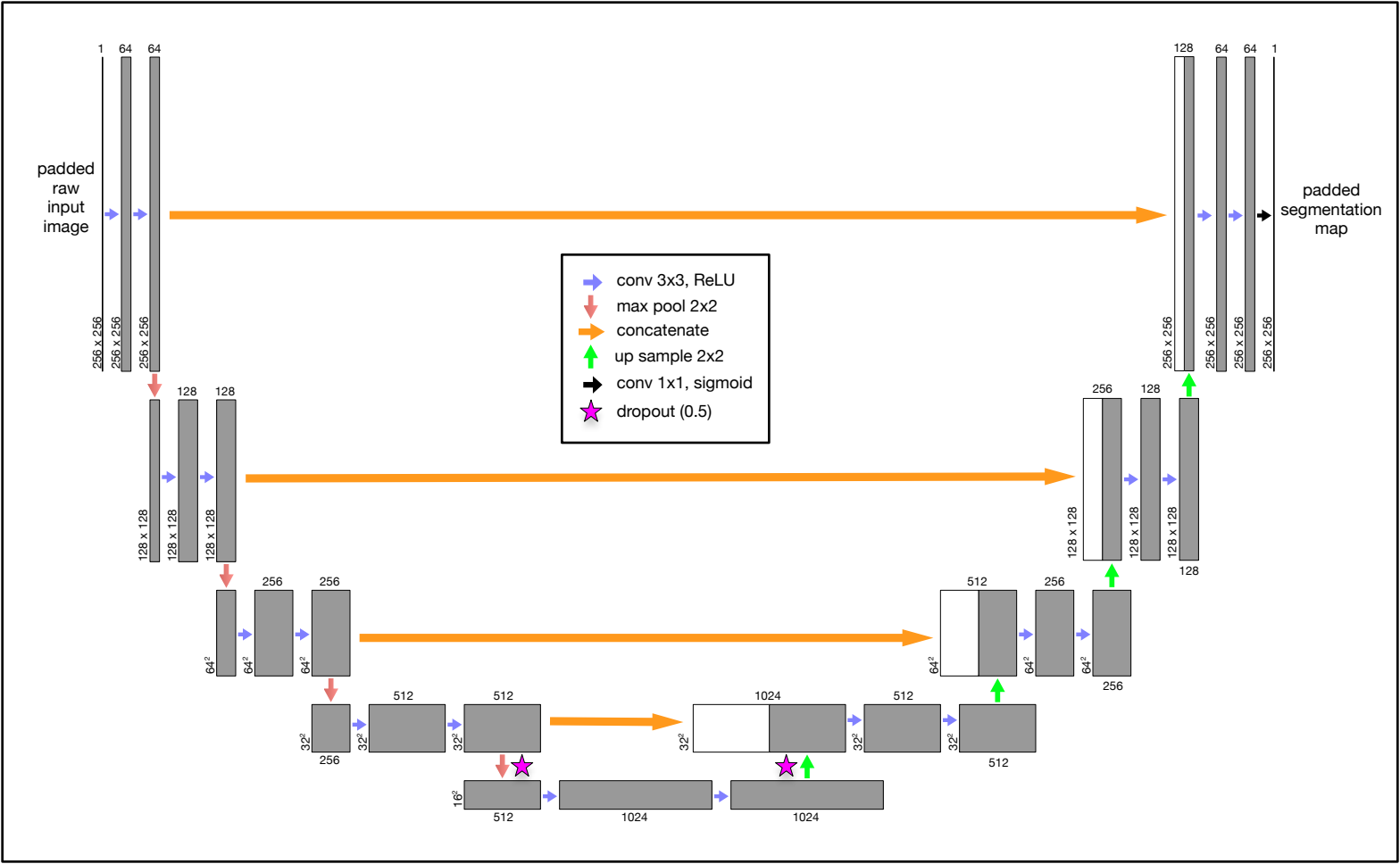
(*c*)

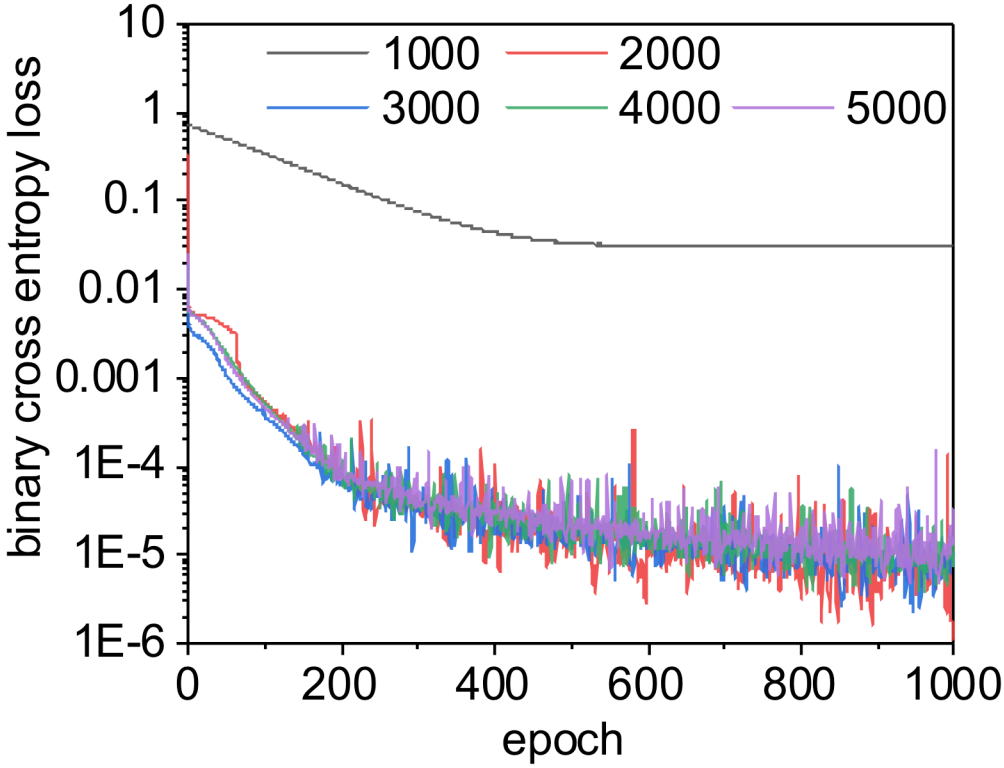

7,000 Images
Random Split
5,000 Training Images
Variable Training Set Sizes
1,000
2,000
3,000
4,000
5,000
Convolutional Neural Network
Train (1000 epochs)
Classify Test Set
Segmentation Map
Accuracy Metrics
Gradient Boosting
Train (100 stumps)
Classify Test Set
Segmentation Map
Accuracy Metrics
Random Forest
Train (100 trees)
Classify Test Set
Segmentation Map
Accuracy Metrics
Binary Labels
2,000 Test Images

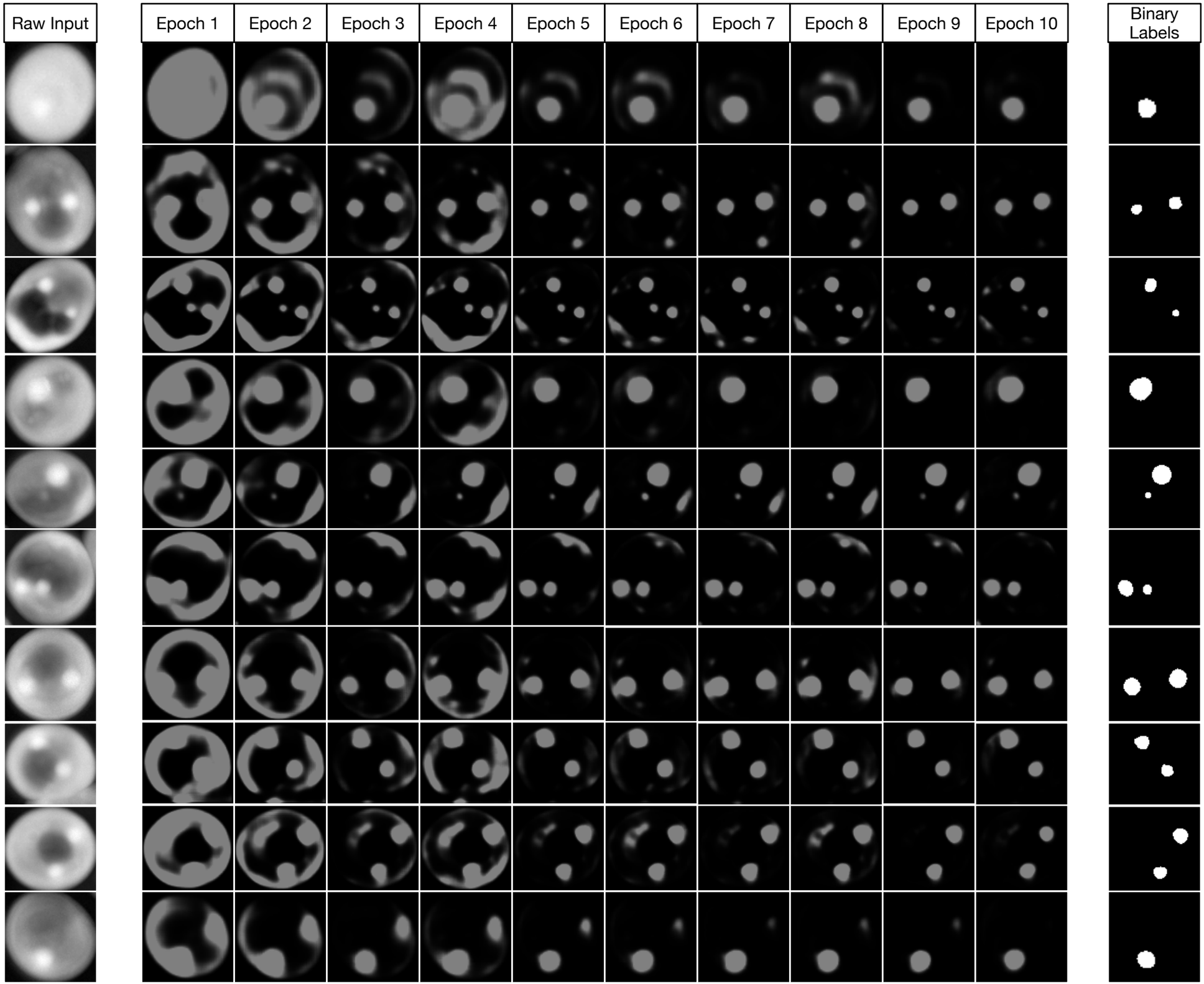

pixel 0
k features extracted with image filters
2D array of all n pixels and their k features
Feat 0 Feat 1 Feat 2 Feat k
Pixel 0 134 42 255 192
Pixel 1 111 64 234 184
Pixel 2 123 36 240 189
Pixel 3 127 54 239 199
Pixel 4 131 62 195 201
Pixel 5 95 92 202 196
Pixel n 119 101 209 198
Train Random Forest or XGBoost Model by Fitting Raw Data to Binary Labels
Tree 0
Tree 1
Tree j
The Trained Model
(a)
(b)
(c)
(d)
(e)
Tree 0
Tree 1
Tree j
Majority Vote
Lipid (Y/N)
(f)
Feat 0 Feat 1 Feat 2 Feat 3 Feat 4 Feat 5 Feat k
pixel 0 134 42 255 201 19 169 192
F4>45
F3>233
F1>33
F5<240
Fk<131
F0<155
F2>221
y
n
yes
no
pixel 0 is not a lipid
(g)

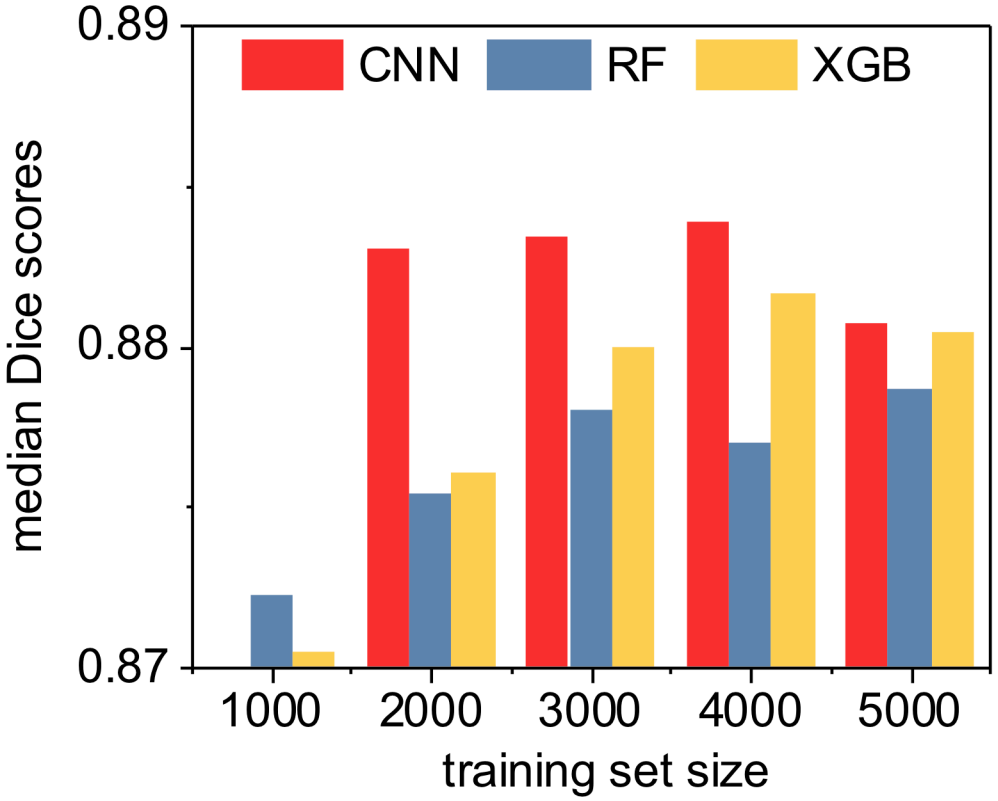

Original Image
Parameter to image filter
σ=0.3
σ=0.7
σ=1.0
σ=1.6
σ=3.5
σ=5.0
σ=10.0
Image Filter Function
Gaussian Smoothing
Sobel Edge Detection
Difference of Gaussians
Laplacian of Gaussian Edge Detection
Structure Tensor Eigenvalues

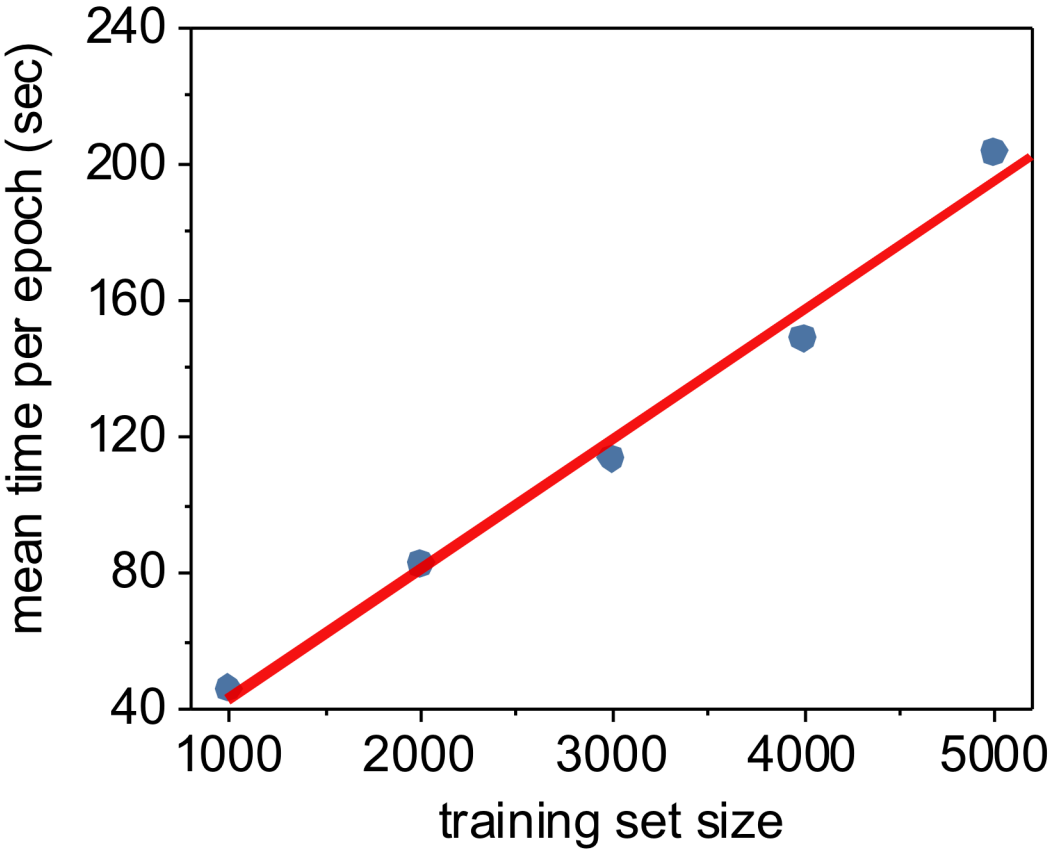

1400
1200
1000
800
600
400
200
0
system time in CPU minutes
0
20
40
60
80
100
training set size (% of overall library)

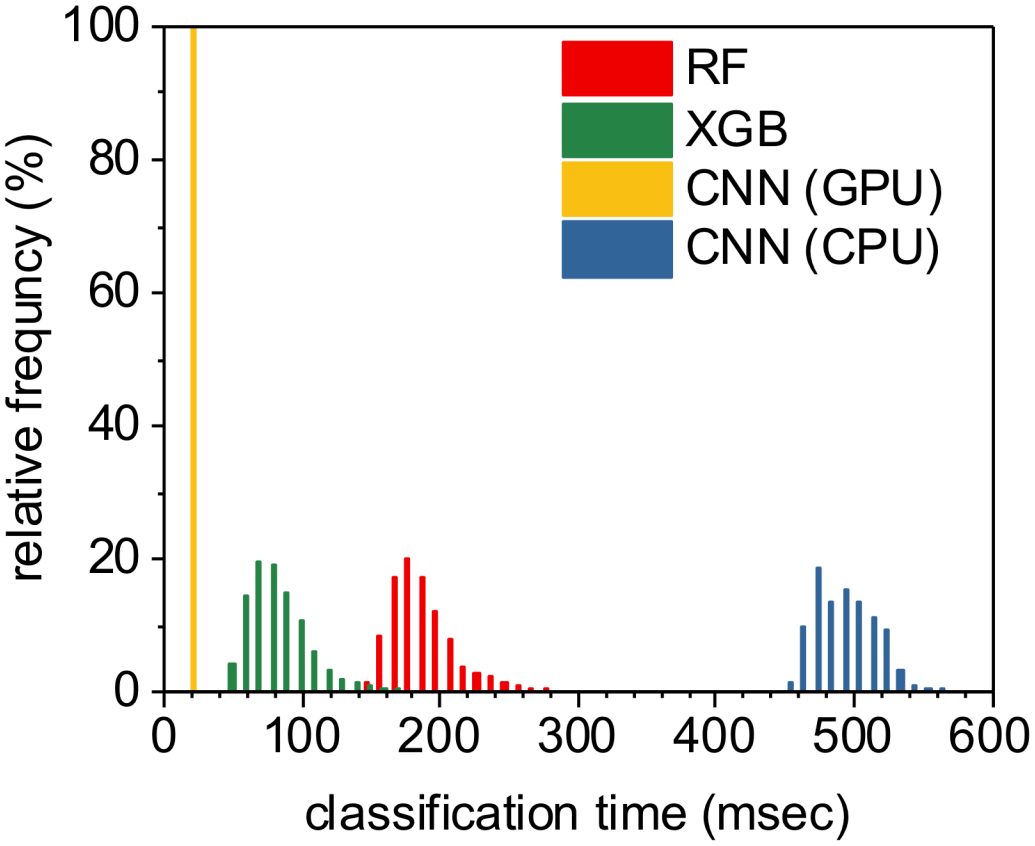

padded raw input image
padded segmentation map
conv 3x3, ReLU
max pool 2x2
concatenate
up sample 2x2
conv 1x1, sigmoid
dropout (0.5)
1
64
64
128
128
256
256
512
512
1024
1024
512
512
256
256
128
128
64
64
1
256 x 256
128 x 128
64^2
32^2
16^2

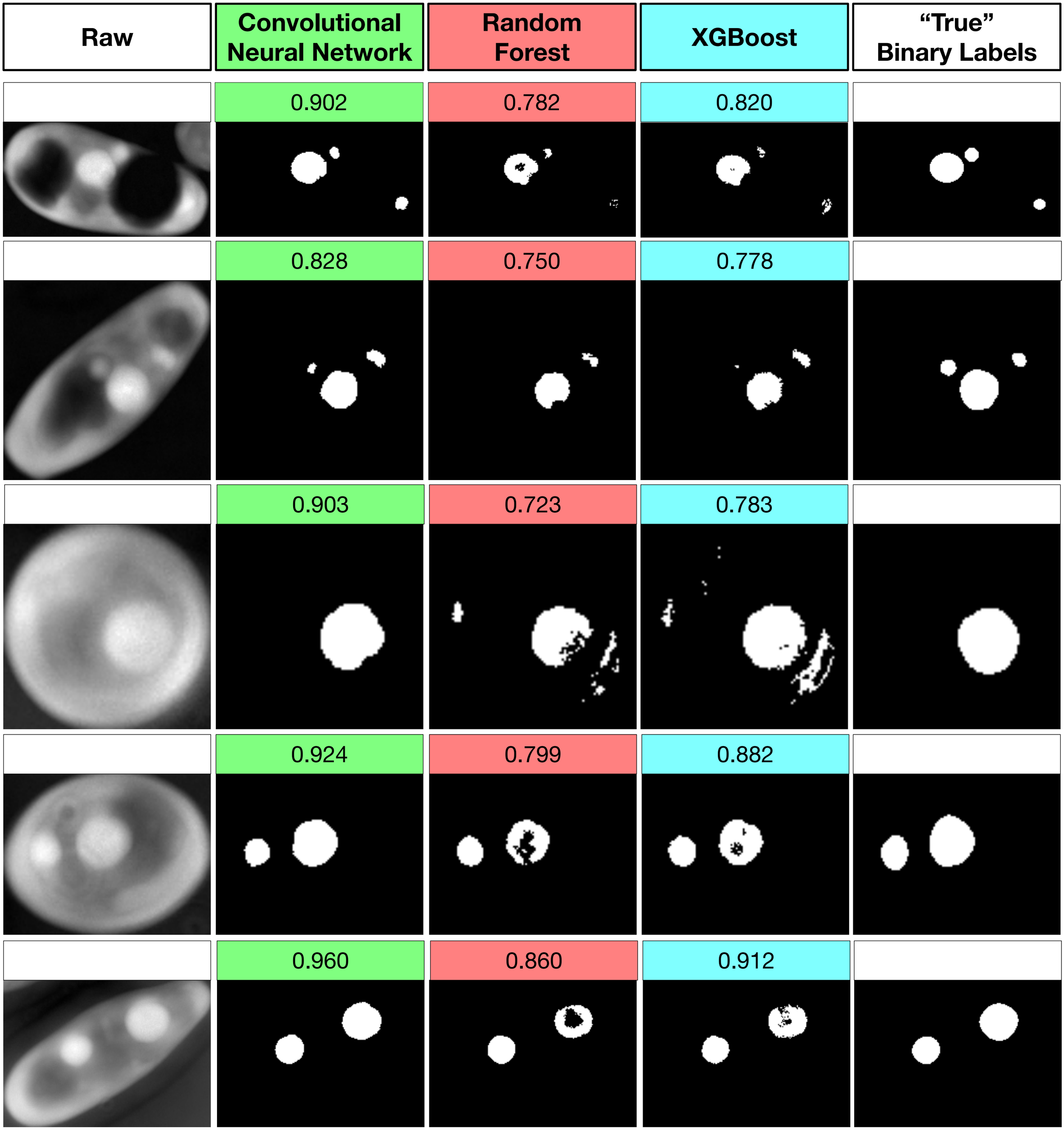

10
1
0.1
0.01
0.001
1E-4
1E-5
1E-6
binary cross entropy loss
0
200
400
600
800
1000
epoch
1000
2000
3000
4000
5000


Raw Input
Epoch 1
Epoch 2
Epoch 3
Epoch 4
Epoch 5
Epoch 6
Epoch 7
Epoch 8
Epoch 9
Epoch 10
Binary Labels


0.89
0.88
0.87
median Dice scores
CNN
RF
XGB
1000
2000
3000
4000
5000
training set size


240
200
160
120
80
40
mean time per epoch (sec)
1000
2000
3000
4000
5000
training set size


RF
XGB
CNN (GPU)
CNN (CPU)
relative frequncy (%)
classification time (msec)
0
20
40
60
80
100
0
100
200
300
400
500
600


Raw
Convolutional Neural Network
Random Forest
XGBoost
"True" Binary Labels
0.902
0.782
0.820
0.828
0.750
0.778
0.903
0.723
0.783
0.924
0.799
0.882
0.960
0.860
0.912