Supplemental Digital Content containing Supplemental Methods describing the technical details of the deep learning algorithm and the accompanying Supplementary Table 1. The Supplemental Digital Content was not copyedited by *Archives of Pathology & Laboratory Medicine*.

Supplementary Methods

Here, we describe the procedure for developing the LYmph Node Assistant (LYNA) algorithm. We used the Cancer Metastases in Lymph Nodes 2016 challenge data set (Camelyon16)[1].

Data preparation

We begin by describing the method of processing a whole-slide image (WSI) into image patches for both algorithm development and usage. Each digitized WSI is on the order of 100,000×100,000 pixels at 40X magnification. Because images of this size cannot be easily processed (most machine learning models operate on images smaller than 1,000×1,000 pixels), we divided each WSI into a grid. This grid is of size 128×128 pixels, which is large enough to include several tumor cells, and was also used by Litjens et al[2]. For the purposes of this supplementary methods section, we will refer to each of these 128×128 pixels as a region of interest (ROI).

Next, we filtered this grid to remove non-tissue regions by discarding ROI image patches with an average grayscale value above 0.8[3]. These patches contain pixels that are on average too bright ("white"), and comprise about 80% of image on average. After filtering for tissue, 10,000 to 400,000 patches (median of 90,000) remained in each WSI; slides containing a larger tissue specimen generated more patches. To ensure that the algorithm was provided with sufficient context to make each prediction, we cropped image patches of size 299×299 pixels (75 µm) centered on each ROI in the grid as input to the algorithm.

Algorithm development requires both an input image and a label, or desired algorithm prediction for the image. In this instance, the algorithm should output 1 ("yes") if the image contains tumor, and 0 ("no") otherwise. The Camelyon16 dataset provided pathologist-annotated pixel-level outlines of tumors on each of 110 tumor-containing slides in the training set. Based on the grid described in the previous paragraph, each tumor slide contained between 20 and 150,000 tumor patches (median of 2,000). In the tumor slides, this corresponded to 0.01% to 70% (median 2%) of the patches. In other words, tumor sizes varied dramatically from the smallest micrometastasis to large macrometastasis that have effaced most of the lymph node. Patches containing tumor were labeled "1" and "0" otherwise for algorithm development.

Dataset split

Next, unbiased algorithm evaluation requires that a set of images be used to develop the algorithm, and a separate set for evaluation. In the Camelyon16 challenge, the pathologist-annotated labels were provided for the “training” set: 110 tumor slides with pixel-level annotations and 160 slides annotated at the slide-level as “normal”. The “test” set of 130 slides were originally provided as images; without annotations at either the slide level or pixel level. The labels were later released at the conclusion of the challenge in preparation for a followup challenge (Camelyon17), but those released labels were used only for evaluation of our algorithm (Table 1 in the main text).

Thus, to make decisions during algorithm development about which hyperparameters (“knobs”) to tune, we further divided the training dataset of 270 slides into a training split (216 slides, 80%) and a tuning split (54 slides, 20%). This split was generated randomly, while ensuring that the ratios of the two models of whole-slide scanners used to digitize these images and the ratio of tumor to normal slides were consistent in both splits. The deep learning algorithm described in this manuscript was “trained” on the train split, with periodic internal validation on the tuning split to select various hyperparameters (described in the *Neural network training* section below).

Neural network architecture

The deep learning algorithms that we developed are a type of multi-layered (“deep”) convolutional neural network[4]. For the remainder of this technical section, we will use the term “network” to refer to the deep learning algorithm, to avoid confusion with “algorithms” which are used in computer science and deep learning for a different purpose. Our network takes an image as input and outputs a predicted value between 0.0 and 1.0. We used the Inception (V3) neural network architecture, as detailed in Supplemental Table 1. Briefly, the network contains many sequential layers of artificial “neurons”, each of which performs a specific type of computation using the internal weights (also called “parameters”) in the neuron. Although the architecture is pre-determined, the weights are learned during the training process (described in the *Neural network training* section below). We slimmed down the Inception (V3) architecture by reducing the number of neurons in each layer. This helps to control overfitting by reducing the number of weights in the neural network, and reduces computation, thus speeding up both training and algorithm usage. In addition, this speedup allowed us to quickly experiment with different “hyperparameters” (such as data augmentation magnitudes and whether to apply color normalization, described in the *Data augmentation* section below).

Layer type	Kernel size	Stride	Number of filters (“neurons”)	Input shape
Convolution	3×3	2	16	299×299×3
Convolution	3×3	1	16	149×149×16
Convolution	3×3	1	16	147×147×16
Max pool	3×3	2	n.a.	147×147×16

Convolution	1×1	1	16	73×73×16
Convolution	3×3	1	19	73×73×16
Max pool	3×3	2	n.a.	71×71×19
Inception block (repeat 3X)	Mixed	1	Mixed	35×35×19
	Mixed	1	Mixed	35×35×64
	Mixed	1	Mixed	35×35×64
Inception block (repeat 1X)	Mixed	2	Mixed	35×35×64
Inception block (repeat 4X)	Mixed	1	Mixed	17×17×118
	Mixed	1	Mixed	17×17×76
	Mixed	1	Mixed	17×17×76
	Mixed	1	Mixed	17×17×76
Inception block (repeat 1X)	Mixed	2	Mixed	17×17×76
Inception block (repeat 2X)	Mixed	1	Mixed	8×8×127
	Mixed	1	Mixed	8×8×203
Average pool	8×8	n.a.	n.a.	8×8×203
Dropout	n.a.	n.a.	n.a.	1×1×203
Classifier	1×1	1	2	1×1×2

Supplementary Table 1. Neural network architecture: slimmed-down Inception (V3) that contains fewer “neurons” per layer (*depth_multiplier=0.1, min_depth=16*). Inception blocks contain multiple branches consisting of combinations of convolutions and average or max pooling, and concatenation operations to merge the branches again. Complete code to create this architecture can be found at the open source code repository github[5].

Neural network training

Training the neural network typically involves first initializing the weights of the network in one of two ways. The simplest is random, based on heuristics to ensure the gradient, or learning “signal” doesn’t vanish through the many layers of the network. A second option uses a trained network of the same architecture as a starting point (“pre-training”), a form of transfer learning. This second network may have been trained using another dataset, including those comprising other types of images. For example, published networks for retina fundus images in ophthalmology[6] and clinical images of the skin in dermatology[7] were initialized from networks trained on ImageNet[8], which contains 1,000 classes of objects such as cars, cats, and dogs. The effectiveness of transferring from such disparate images is thought to be because the basic features learned by earlier layers in the network (such as edges and shapes) generalizes to different object recognition tasks. In general, this type of transfer learning becomes less important when sufficient data are available to train the network “from scratch”. In our work, 214 WSI’s produce tens of millions of smaller image patches after cropping, yielding a large dataset for training accurate deep learning algorithms even without this pre-training procedure. In addition, pre-trained versions of our slimmed-down Inception V3 model were not available.

After initialization, the network was provided with randomly selected images and the weights of the network were adjusted to reduce the prediction error (as measured by the softmax cross-entropy loss) using the TensorFlow software[9]. This process, called stochastic gradient descent, was repeated millions of times. To speed this process up, we used multiple (8) computers, each of which computes the current prediction error with a separate randomly selected batch of images (“batch size” of 32), with asynchronous updates to the weights. The version of stochastic gradient descent we applied is called RMSProp[10], with a momentum of 0.9, decay of 0.9, and ϵ of 1.0. These settings were the default used in training Inception (V3) on ImageNet. Another critical setting is the speed with which the network weights were adjusted, also termed the learning rate. A high learning rate tends to “over-shoot” the ideal weights, while a low learning rate takes too long to converge. We used an initial learning rate of 0.05, with a decay of 0.5 every 2 million examples seen, as tuned based on the performance of the model on tuning split in the training dataset (5 million patches).

The process of selecting image patches for training purposes also makes a difference because of the wide variety of specimen and tumor sizes on different slides. Specifically, each slide contained between 10,000 to 400,000 tissue-containing patches, and for tumor slides between 20 to 150,000 of these patches contained tumor. As such, simply taking these patches as independent image examples would have biased the network towards slides with more tissue or larger tumors. Because the characteristics of small and large tumors are different (for example lobular and ductal subtypes of breast adenocarcinoma tend to grow in different patterns), we used a sampling procedure that was unbiased at the slide level using three steps. We first selected the “tumor” category with probability P, and “normal” with probability $1-p$. In step two, we selected a random slide containing patches of the selected category, and in step three, selected from that random slide a random patch containing that category. The probability p allowed us to show the network more examples of either tumor or non-tumor. Although we first started with $P=.50$, we later found in the tuning split that $P=.20$ produced fewer false positive predictions.

Readers who wish to recreate this process may find similar code available on github[11]. The remainder of our code comprises custom pipelines specific to our internal infrastructure (such as data storage and computer cluster).

Color normalization

As discussed in the main text, we preprocessed each image to standardize the color distributions despite factors such as different laboratory preparation conditions and digital scanner color balances. This process contains two steps; first computing the color statistics for each WSI, and next normalizing the color statistics for each image to that of the reference. In our work, we used the median color statistics as a reference. The formulas used in this approach was described previously[12], and examples can be found in Figure 2.

Data augmentation

Data augmentation improves the performance of neural network performance by artificially perturbing the images (such as rotating the image by 90 degrees) during training. First, increasing the size of the training dataset typically improves a neural network's performance. Data augmentation artificially inflates the size of the dataset. In addition, it is thought that these augmentations enable the trained network to be insensitive to the perturbations used. For example, because the orientation of the tissue is based on its (frequently random) orientation in the tissue cassette and sectioning protocol, the network should be insensitive to the rotations or mirroring of the image. The specific perturbations we used were:

- Randomly rotate and/or mirror the image to one of the 8 possible orientations: the original image and 90, 180, 270-degree rotations, and a mirroring followed by all 4 rotations.
- Randomly perturb the brightness of the pixels by a random magnitude (tensorflow.image.random_bightness with a maximum delta of 0.25)
- Randomly perturb the saturation of the image by a random magnitude (tensorflow.image.random_saturation with a maximum delta of 0.25)
- Randomly perturb the hue (average color) of the image by a random magnitude (tensorflow.image.random_hue with a maximum delta of 0.04)
- Randomly perturb the contrast of the image by a random magnitude (tensorflow.image.random_contrast with a maximum delta of 0.75)
- "Jittering" the original image patch location around the center 128×128 pixel region of interest by up to 8 pixels.

The magnitudes of these perturbations were determined using the performance of the deep learning algorithm on the tuning split of the training dataset.

Neural network application

Finally, application of the algorithm results in a prediction value between 0.0 and 1.0 for each input image. A higher value indicates a higher likelihood of that particular tissue patch containing tumor. To obtain the algorithm's interpretation of the entire slide, we apply the algorithm to each tissue-containing 128×128 pixel ROI on each slide. Similar to the training setup, we provide the algorithm with additional context around the ROI for a total of 299×299 pixels, or 75 µm. Each slide contains 90,000 such patches on average. This generates a two-dimensional table of predictions, each representing the algorithm's predicted likelihood of the corresponding tissue patch containing tumor. As shown in Figure 1 of the main text, these predictions can be visualized using a heatmap that indicates high tumor likelihood with red and low likelihood with blue, or by highlighting tumor areas using an outline or circle.

The prediction for the WSI is simply the maximum prediction across all patches in the slide. Intuitively, this corresponds to a hypothetical tireless pathologist who reviewed the entire WSI at high magnification, then reporting the "suspiciousness score" of the most suspicious ROI in the slide. The higher this score is, the more likely the slide contains tumor (based on the most suspicious patch). However, because each WSI contains tens of thousands of patches, this strategy is prone to errors if the underlying patch predictions are not accurate enough. In practice, we have found that our simple approach yields high area under receiver operating characteristic curve (AUCs) that are comparable to more complex methods that require a second machine learning classifer (such as a random forest) for a "second-stage" slide-level prediction.

A technique termed ensembling tends to improve neural network performance by averaging multiple predictions. We apply this technique in three ways. First, we averaged the weights of the network across the previous steps, using an exponential moving average decay of 0.9999. Intuitively, this stabilizes the network's weights by preventing drastic changes. Second, for each ROI-containing patch, we applied the algorithm to all 8 random orientations (see *Data augmentation* section above), resulting in the algorithm seeing the same image from different perspectives. We take the geometric mean of these predictions. The geometric mean was used because it yielded slightly better results than the arithmetic mean on the tuning split in the training set. Finally, we trained 10 independent deep learning algorithms and averaged their outputs for each patch.

Of the three averaging techniques mentioned, the first is computationally the cheapest because it uses a single set of weights and requires only one computational inference for each image input. Depending on the number of augmentations, the second and third can increase or decrease in computational burden. In practice, the visual appearance of the heatmap does not change dramatically with these techniques, but the quantitative metrics (such as area under the receiver operating characteristic curve (AUC) for performance on patches) improve with more averaging, albeit with diminishing returns.

References

1. Ehteshami Bejnordi B, Veta M, Johannes van Diest P, et al. Diagnostic Assessment of Deep Learning Algorithms for Detection of Lymph Node Metastases in Women With Breast Cancer. *JAMA*. 2017;318(22):2199-2210.
2. Litjens G, Sánchez CI, Timofeeva N, et al. Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis. *Sci Rep*. 2016;6:26286.
3. Janowczyk A, Madabhushi A. Deep learning for digital pathology image analysis: A comprehensive tutorial with selected use cases. *J Pathol Inform*. 2016;7:29.
4. LeCun Y, Bengio Y, Hinton G. Deep learning. *Nature*. 2015;521(7553):436-444.
5. Github TensorFlow Inception V3. https://github.com/tensorflow/tensorflow/blob/master/tensorflow/contrib/slim/python/slim/nets/inception_v3.py. Accessed June 6, 2018.
6. Gulshan V, Peng L, Coram M, et al. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. *JAMA*. 2016;316(22):2402-2410.
7. Esteva A, Kuprel B, Novoa RA, et al. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*. 2017;542(7639):115-118.
8. Russakovsky O, Deng J, Su H, et al. ImageNet Large Scale Visual Recognition Challenge. *Int J Comput Vis*. 2015;115(3):211-252.
9. Abadi M, Barham P, Chen J, et al. TensorFlow: A System for Large-Scale Machine Learning. In: *OSDI*. Vol 16. ; 2016:265-283.
10. Hinton G, Srivastava N, Swersky K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. 2012.
11. Github Training Inception. https://github.com/tensorflow/models/tree/master/research/inception. Accessed June 6, 2018.
12. Liu Y, Gadepalli K, Norouzi M, et al. Detecting Cancer Metastases on Gigapixel Pathology Images. *arXiv [csCV]*. March 2017. http://arxiv.org/abs/1703.02442.