Particle Detection in Bone Marrow Using CNNs and Transformer Networks

Marcus Williams and Anna Källén

May 2023

Supervisors:
Jesper Jönsson, 070 141 45 87, jejo@cellavision.se, CellaVision
Mattias Näsström, 076 329 85 62, mana@cellavision.se, CellaVision
Anders Heyden, 046 222 85 31, anders.heyden@math.lth.se, LTH

Abstract

The critical role of hematological stem cells, concentrated in bone marrow, in disease diagnosis makes the detection of these cells an imperative task. The current process, however, is slow and heavily reliant on expert interpretation, underscoring the need for automation.

In this thesis, we collected a 20x microscopy dataset of real and fake bone marrow particles and fine-tuned several CNNs and Transformer networks on the task of classifying them. We performed a hyperparameter investigation, explored whether illuminating the sample from an angle improves performance and analysed what the networks are looking for with several interpretability methods.

The best results were obtained by fine-tuning a Swin Transformer V2S, using images captured standard illumination settings. This model reached 97.19% on the test set. Different hyperparameters were evaluated with the result that most hyperparameters are of little significance, apart from dataset size where the accuracy increased logarithmically with the dataset size.

A much faster model was made by quantising ResNet18 to 8-bits, which had an accuracy of 96.75% while only taking 59ms per image for inference on microscope hardware, making it feasible to use in an automated bone marrow examination process in the future.

Acknowledgements

We want to thank our supervisors Jesper Jönsson and Mattias Näsström at CellaVision for their valuable input and guidance throughout our thesis project. Veronica Jenei also deserves a special thanks for labeling all of the data and answering our questions about bone marrow theory.

Thanks to Anders Heyden at Lunds University, for great supervision and input on academic writing.

Contents

1 Introduction and Background

Analysing bone marrow samples is an important part of diagnosing many haematologic disorders [1]. One such analysis is a Nucleated Differential Count or NDC which involves identifying and counting different types of nucleated cells (cells containing a nucleus) in a sample to help diagnose and monitor various health conditions. This test currently consists of a trained haematologist or pathologist first locating bone marrow particles to determine regions of interest at low magnification (10x). The next step is to examine these regions with an oil-immersed 100x objective lens in order to discriminate diverse cell types. Manually investigating and finding cells in a bone marrow NDC is labour-intensive, time-consuming, requires considerable training, and is poorly reproducible [1]. It would therefore be beneficial if this process could be automated. The cells of most interest are the ones found near bone marrow particles as these are more likely to be in an earlier stage of development than the more mature cells in the blood stream. Immature cells provide important information about haematopoiesis, the derivation of many different blood cell types from haematopoietic stem cells. The first step of the analysis pipeline is therefore to find these bone fragments in the bone marrow sample. Since finding these particles is the basis for the bone marrow analysis, it is important to find actual particles and not misidentify other objects or artifacts as bone fragments. The currently used method for finding particles tends to mistake artefacts or things resembling particles for actual particles, largely due to limitations in illumination and optics.

This thesis will consist of collecting a large amount of training, validation and test data of both real bone marrow particles and fakes. This data will consist of 20x microscope images using different ways to illuminate the sample. The main focus of the report will be fine-tuning the pretrained networks ResNet, Swin Transformer V2 and ConvNeXt V2 and seeing how their performance changes based on network size, image type, image size and on freezing certain layers. We will also train them from scratch and see how this impacts performance. Furthermore we will perform a interpretability investigation into what these networks focus most at when deciding if an image contains a real particle or not.

2 Biological Theory

2.1 Bone Marrow

Bone marrow is a type of spongy tissue that is found inside bones. There are two types of bone marrow, red bone marrow and yellow bone marrow. Yellow bone marrow mainly consists of fat, while red bone marrow consists of stem cells that can develop into red blood cells, white blood cells and platelets. There are also blood vessels in the bone marrow which transport the mature blood cells into the peripheral blood circulation. [2] Bone marrow analysis could be performed for a variety of different reasons including:

- Diagnosing blood disorders: The procedure helps in the diagnosis of blood-related conditions like anaemia, leukaemia, lymphoma, myelodysplastic syndromes, and multiple myeloma, among others. [3]
- Monitoring treatment response: It can be used to assess the effectiveness of treatments like chemotherapy or stem cell transplantation for blood-related diseases.[4]
- Investigating abnormal blood cell counts: If a blood test reveals an abnormal count of red or white blood cells or platelets, a bone marrow analysis may help determine the cause.[5]

- Detecting infections: In some cases, bone marrow analysis can detect infections that are difficult to identify using other diagnostic methods, particularly those that affect the bone marrow itself.[6]
- Staging cancer: In some cases, the procedure can help determine the extent or stage of a cancer, which is crucial for determining the appropriate treatment plan.[7]

There are two main types of bone marrow examinations, a bone marrow trephine biopsy and a bone marrow aspiration. [5] A bone marrow biopsy means extracting a solid piece of bone, while the aspirate is the liquid part of the bone marrow and extracted by puncturing a bone, typically the hip or sternum, with a hollow needle so liquid can be drawn out. The aspirate consists of mature blood cells, various stages of immature blood cells as well as bone fragments. The immature blood cells, i.e., bone marrow cells or haematopoietic stem cells, are concentrated near the bone fragments, since that is where they are produced. The bone fragments, also known as particles, are therefore of interest while looking for bone marrow cells. [8] Figure 1 and 2 show examples of particles and artifacts that may be confused with particles.

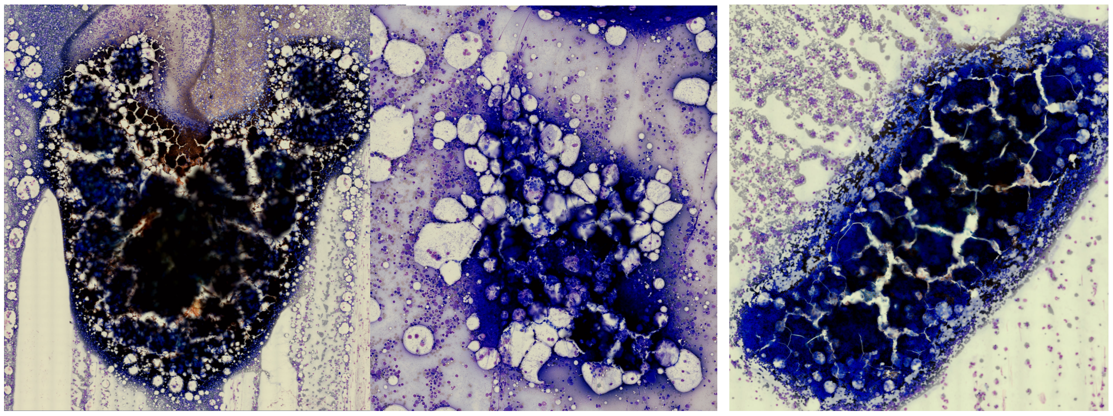

(a) (b) (c)

Figure 1: Example images of particles.

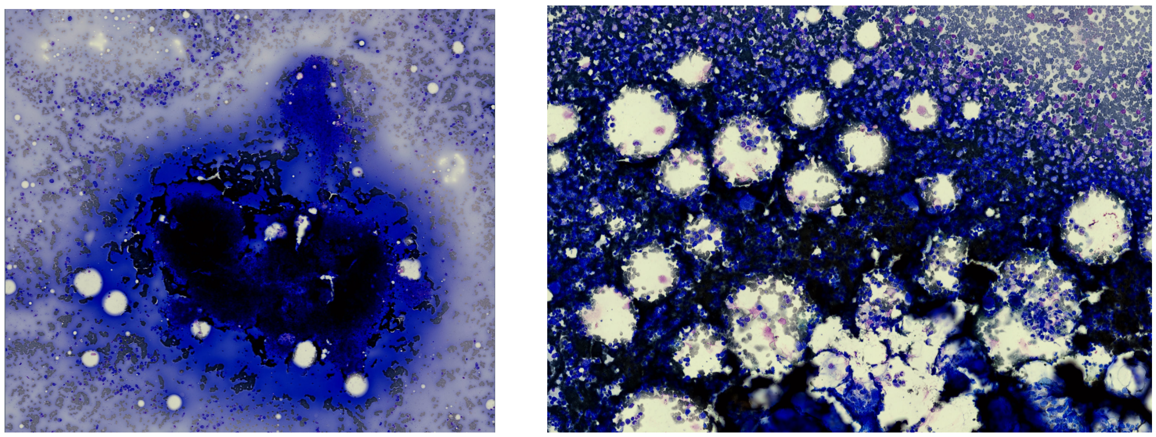
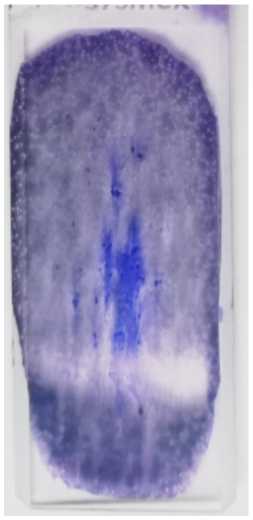

(a) (b)

Figure 2: Example images of regions which are not particles but might be confused with particles.

2.2 The Examination Procedure

After the aspiration, the bone marrow aspirate is prepared for analysis. Firstly, the aspirate is smeared on a test slide using a smearing technique. Smearing techniques are used in microscopy and laboratory analysis to prepare a thin layer of a specimen on a glass slide, making it suitable for examination under a microscope. This allows for better visualisation of individual cells and their components. There are primarily two different smearing techniques, the wedge/long smear and the squash smear. For the squash smear, a droplet of blood is placed in the centre of a slide and then another slide is placed on top of this pushing the liquid out towards the edges. The wedge smear by contrast, is created by placing a drop of blood at the edge of the slide and then spreading it out along the slide using the edge of another slide. Examples of the results of the two smearing techniques can be seen in Figure 3. The squash smear results in most bone particles ending up near the centre of the smear, while for the wedge smear, the particles are typically mostly located at the end of the smear. [8]

After smearing, the sample is allowed to dry for a brief period. This step is essential to ensure that the cells adhere well to the slide and that the subsequent staining process is effective. The air-dried smear is then usually fixed using a fixative solution, such as methanol or formaldehyde. [8]


(a) Squash smear.

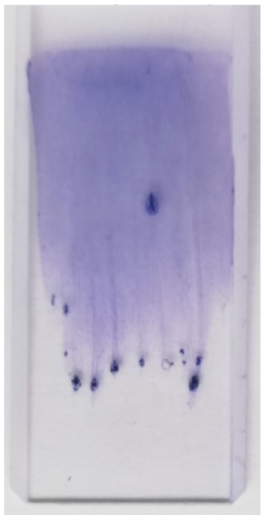
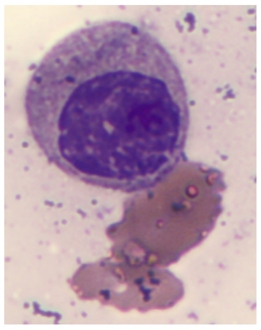
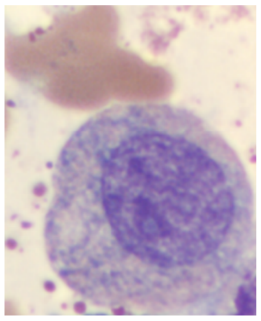

(b) Wedge smear.

Figure 3: Smearing techniques.

After this, the smears are stained to make it easier to distinguish and count the cells. There are several different types of stain used for different purposes, for example May-Grünwald-Giemsa (MGG), Wright (W) and Wright-Giemsa (WG). Figure 4 shows examples of the stains. Sometimes a thin plastic film, known as a coverslip, is added on the slide for protection. Adding the coverslip involves placing a few drops of mounting medium, onto the stained slide and then carefully lowering it onto the mounting medium to avoid air bubbles. The medium helps preserve the stained cells and secures the coverslip in place. [8]


(a) Wright (W)


(b) Wright-Giemsa (WG)

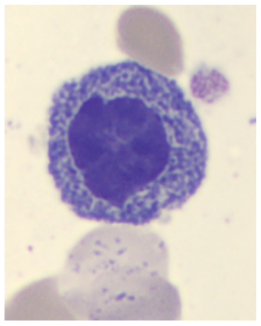

(c) May-Grünwald-Giemsa (MGG)

Figure 4: Stain techniques.

The stained sample is then ready for analysis. The first step of the analysis is to look at the slide with a 10x or 20x objective to find particles in regions of interest for the examination. The interesting regions are those with good staining quality, bone marrow cells and not too much peripheral blood. The stem cells are often located around the bone fragments, and this is the reason why finding the particles is of interest. The selected regions are then investigated in high-resolution with a 50x-100x objective, and cells of different types are differentiated. Around 500 cells are counted during the differentiation and the ratio between different cell types is finally reported in an aspirate report. [8]

The analysis step is time consuming and requires an experienced haematologist or pathologist. There are not many people with this type of expertise and their time is very valuable, which provides motivation for developing an automated process. An automated process could consists of something like the following: First, the entire slide is scanned to produce the preview scan which is used to identify particles. The selected regions are then scanned once again, this time in low resolution to make it possible to select regions for a differential count. Then, the regions are scanned in high-resolution, the cells are counted, and the differential is sent back to the user for review before an aspirate report. An overview of the manual and automated process can be seen in Figure 5.

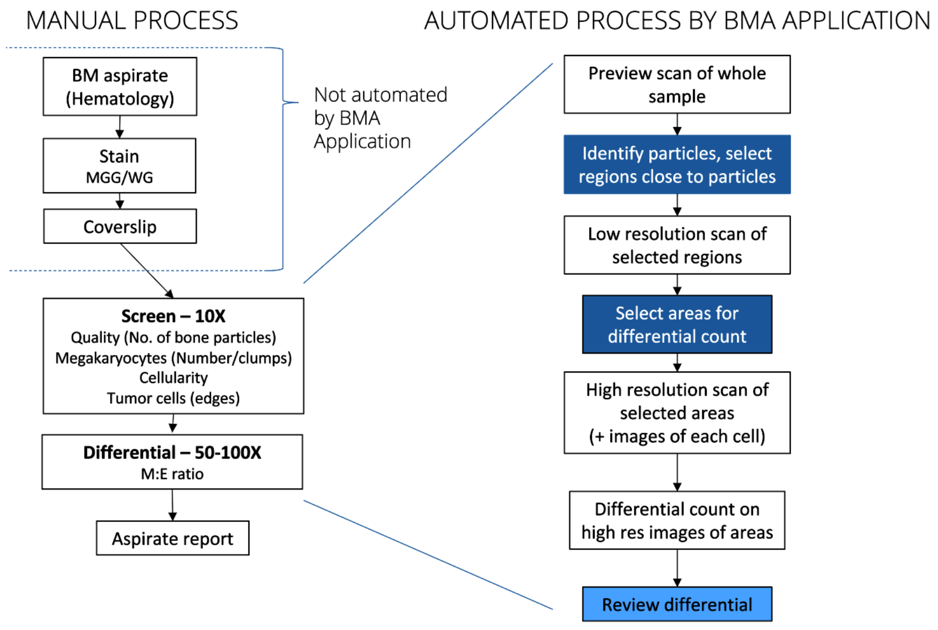

MANUAL PROCESS
AUTOMATED PROCESS BY BMA APPLICATION
BM aspirate (Hematology)
Stain MGG/WG
Coverslip
Not automated by BMA Application
Screen – 10X
Quality (No. of bone particles)
Megakaryocytes (Number/clumps)
Cellularity
Tumor cells (edges)
Differential – 50-100X
M:E ratio
Aspirate report
Preview scan of whole sample
Identify particles, select regions close to particles
Low resolution scan of selected regions
Select areas for differential count
High resolution scan of selected areas (+ images of each cell)
Differential count on high res images of areas
Review differential

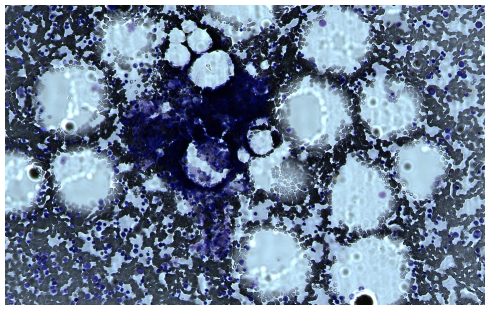
Figure 5: The bone marrow aspirate (BMA) analysis process.

2.3 Dark Field Microscopy

Due to optical properties of an objective, only light within a cone defined by a specific angle can enter the lens. This angle, θ_{NA}, depends directly on the numerical aperture (NA) of the objective and can be calculated with the following formula: [9]

$$NA_{obj} = n \cdot sin(\theta_{NA}) \tag{1}$$

where n is the refractive index of the medium separating the sample and objective. All images taken with illumination angles which lie within this cone are known as bright field images. Light outside this angle will not directly enter the lens, however some light can be scattered of the sample and still enter the lens. Images taken with light outside this angle are known as dark field images. Dark field images often contain structural information of the sample and enhance the contrast of the image. [9] Examples of a sample illuminated from a bright field angle as well as an angle from the dark field can be seen in Figure 6.


(a) Bright field image

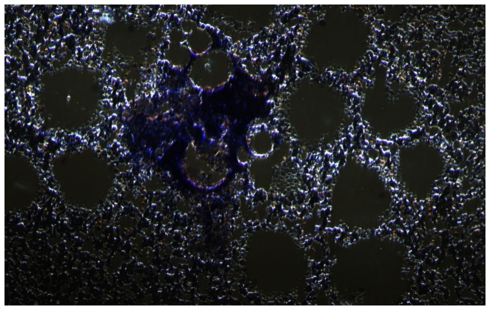

(b) Dark field image

Figure 6: Example of a bright- and dark field image.

2.4 Fourier Ptychography Microscopy

In traditional microscopy, the resolution of an image is limited by the physical properties of the lens used to capture the image. However, Fourier Ptychography Microscopy or FPM uses multiple images captured at different illumination angles which can be combined in order to create a high-resolution image. The sample is illuminated from various angles, and a series of low-resolution images are captured. These images are then combined and processed using a phase retrieval algorithm [10] to create a high-resolution image that contains more information than can be captured by a single microscope image. Several images, captured with a 20x objective, can be combined with the FPM technique so that it results in an image with as good resolution as an image captured by a 100x objective. This implies that the images captured at different angles, i.e., point light source (PLS) images, contain a lot of extra useful information. [10]

In this thesis we will not be using the algorithm directly to generate a high-resolution image, since this is very computationally expensive, but will instead feed the images which would be needed by the algorithm into a neural network and let the network decide how best to use all the additional information.

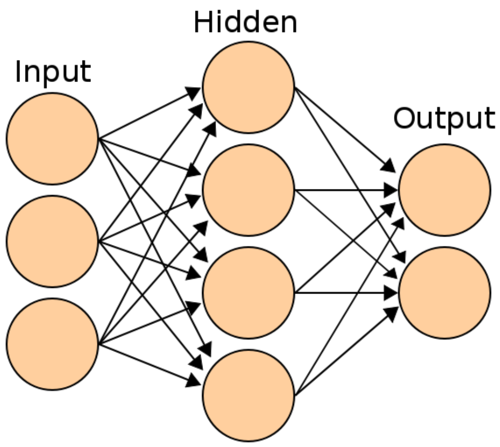
3 Machine Learning Theory

3.1 Artificial Neural Networks

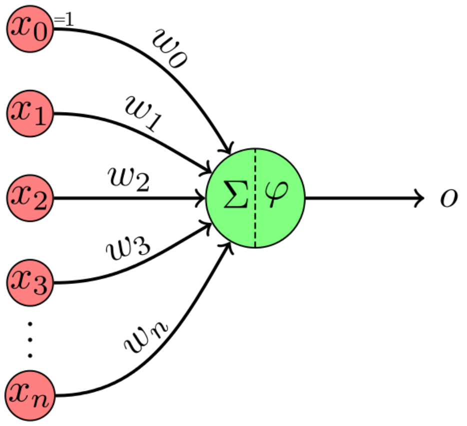
Artificial neural networks (ANNs) are a type of computational model that is inspired by the structure and functioning of biological neurons in the brain. ANNs are composed of layers of interconnected nodes, each node representing an artificial neuron. Each neuron takes in a set of inputs, multiplies each input by a corresponding weight, and applies an activation function to the weighted sum of the inputs. [11]

The output of a neuron is then passed on to the next layer of neurons as input. This process continues until the final layer produces an output that represents the predicted result of the network. The weights between neurons are initially set randomly, and then adjusted during training to minimise the loss function, typically to some form of difference between the predicted output and the actual output. [11]

Weights are typically updated using backpropagation, which is the process of computing the gradient of the loss function with respect to the weights in the network, starting with the final layer and propagating the errors back through the network. Illustrations of a simple neural network and a single artificial neuron can be seen in Figure 7.



(a) ANN. Image taken from [12]



(b) A single artificial neuron. Image taken from [13]

Figure 7: Illustration of a simple neural network and a single artificial neuron.

The equation for a single artificial neuron can be seen below:

$$y = \phi(\sum_{i}^{n} w_i \cdot x_i + b)$$

where the weights w, input x and bias b are fed through an activation function ϕ.

3.2 Optimisers

Optimisers are methods used to adjust the parameters of a machine learning model in order to minimise the loss function or error. The goal of optimisation is to find the best possible set of parameters that results in the lowest possible error on the training data while also generalizing well to new, unseen data. [11]

The simplest optimiser is probably gradient descent. The idea behind gradient descent is to update the model's parameters iteratively in the direction of the steepest decrease in the loss function. This is done by calculating the gradient of the loss function with respect to the model's parameters and using it to update the parameters. [14]

Stochastic Gradient Descent, or SGD, is a common choice of optimiser. SGD is a variation of gradient descent, but instead of computing the gradient for the entire training set SGD computes the gradient on only a subset, or mini-batch, of the training data during each iteration. This helps to speed up the training process and is more computationally efficient than standard gradient descent. It does, however, introduce noise in the gradient updates, which can cause the loss function to fluctuate. However, this can sometimes be beneficial in order to avoid getting stuck in local minima. [15]

Something which can improve SGD is adding a momentum term. Inspired by physics, momentum introduces a term proportional to the previous gradient. Momentum helps the optimiser accelerate in directions with consistent gradients and dampen oscillations in directions with fluctuating gradients, overall, it typically increases convergence speed. [16]

ADAM, or ADAptive Moment estimation, is a more advanced optimiser which maintains separate learning rates for each parameter and adjusts them based on the first and second moments, mean and variance, of the gradients. This leads to more efficient and stable optimisation. This is especially important for sparse and noisy datasets or non-convex optimisation problems [16].

The weight update rule for Adam is given by:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\ w_t &= wt - 1 - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \end{aligned}$$

where:

- t is the iteration number
- w_t is the weight vector at time t
- g_t is the gradient of the loss function with respect to the weights at time t
- m_t and v_t are the first and second moment estimates of the gradient at time t, respectively
- $\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected estimates of m_t and v_t, respectively
- α is the learning rate
- β_1 and β_2 are hyperparameters that control the decay rates of the first and second moment estimates, respectively

- ϵ is a small constant added to the denominator to avoid dividing by 0.

[16]

3.3 Learning Rate Scheduling

There are many different ways to schedule the learning rate for the training. In most approaches, the learning rate is decreasing with the epochs. One way of scheduling the learning rate is REX, or Reflected Exponential, described by equation 2. A plot of the learning rate schedule with $\beta = 1$ can be seen in Figure 8.

$$\beta \cdot \frac{\frac{1-t}{T}}{\frac{1}{2} + \frac{1}{2} \cdot \frac{1-t}{T}} \tag{2}$$

Where β is the initial learning rate t is the current epoch and T is the total number of epochs. [17]

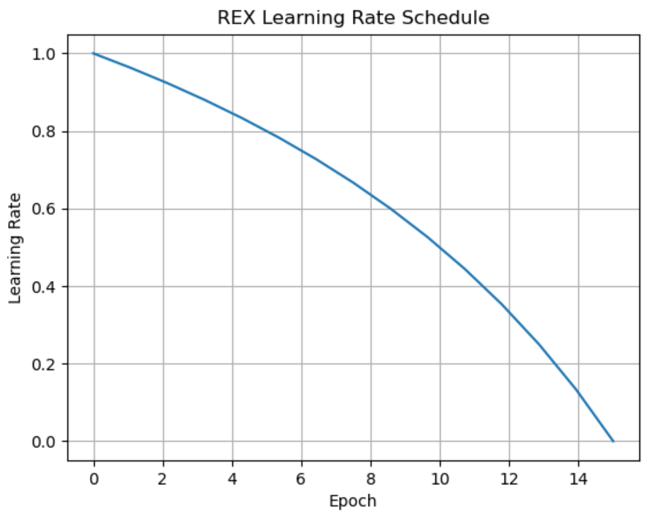


Figure 8: Plot of learning rate schedule for REX and 15 epochs with $\beta = 1$.

3.4 Loss Functions

The goal of the optimisation process in training machine learning models is to minimise the loss function. The classification accuracy might seem like an intuitive choice for a loss function; however, this function would be non-differentiable and non-continuous making it inappropriate for gradient based methods. Instead, we will use the cross-entropy loss.

3.4.1 Cross-Entropy Loss

Cross-entropy loss is a loss function commonly used for classification problems. The cross-entropy loss for a binary classification (i.e., when there are only two classes as in our case) is calculated as: [18]

$$L(y, p) = -(y \log(p) + (1 - y) \log(1 - p)) \tag{3}$$

Where y is the true label (0 or 1) and p is the predicted probability of the instance being in class 1.

Cross-entropy originates from information theory. If we think of the predictions as a distribution and the true values as another distribution, cross-entropy is the measure of dissimilarity between these two distributions. [18]

3.5 Activation Functions

An activation function is a non-linear function applied to the input of each neuron. The non-linearity is important since otherwise our entire network could be reduced to a single linear matrix multiplication. [19]

3.5.1 ReLU vs GELU vs Sigmoid

The sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ is a widely used activation function which is smooth and differentiable making it suitable for gradient based optimisation methods. One drawback of the sigmoid function is that the gradient becomes very small for large positive or negative weights. This means that the weight updates in these regions will be very small, requiring a large number of steps to escape these regions. [19]

ReLU or Rectified Linear Unit is a piece-wise linear activation function which outputs the input directly if it is greater than 0, and outputs 0 when it is less than 0 (see Figure 9). Unlike the Sigmoid activation function the gradient does not vanish for large positive weights, meaning training is faster. ReLU is the activation function used by ResNet. [20] [19]

The GELU, or Gaussian Error Linear Unit, activation function is defined as follows: [21]

$$GELU(x) = x \cdot \Phi(x) \tag{4}$$

Where Φ is the cumulative distribution function, or CDF, of the standard Gaussian distribution. The GELU activation function is a smooth approximation of the ReLU activation function. Like Sigmoid, GELU provides a smooth transition between positive and negative input values while mitigating the vanishing gradient problem. GELU has been shown to have better performance than ReLU in certain situations [21]. GELU is the activation function used by Swin Transformer [22] and ConvNeXt [23].

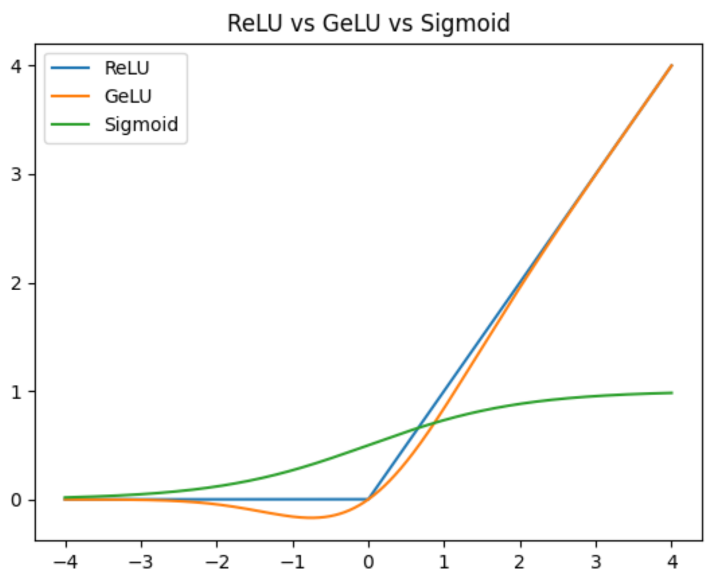


Figure 9: The Sigmoid, ReLU and GeLU activation functions.

3.6 Fully connected Layers

A fully connected layer, also known as a dense layer, could be seen as the simplest possible layer and is a key component of many types of ANNs. A fully connected layer works by connecting every neuron in one layer to every neuron in another layer. Each neuron calculates the weighted sum of its inputs, adds a bias term, and then passes the result through a non-linear activation function. The weights and biases are learned during the training process. [11]

3.7 Convolutional Layers

A convolutional layer involve convolving learned filters, or kernels, with the input image. This operation can be seen as sliding the filter over the input image, computing the element-wise matrix product between the filter and the portion of the image that is currently overlapped by the filter. This produces a single output value, which is then placed into the output feature map at the corresponding spatial location. [24] See Figure 10 for a visual explanation.

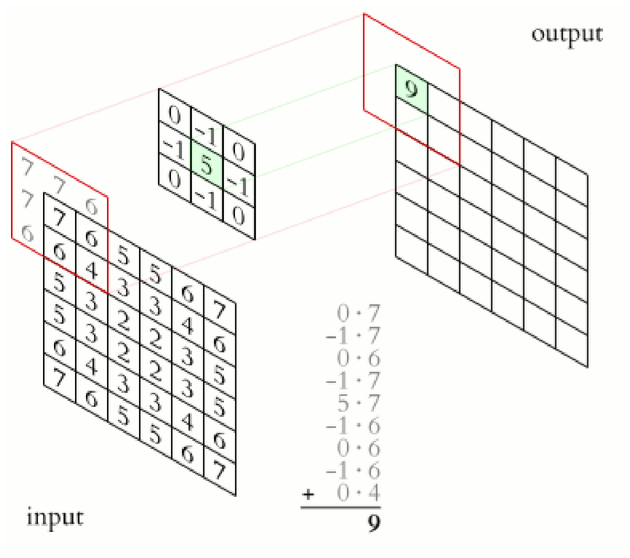


Figure 10: Convolutional filter. [25]

By repeating this process for all possible spatial locations in the input image, we can produce an entire output feature map that represents the filtered version of the input image. The filter parameters are learned during the training process using backpropagation and gradient descent, allowing the network to automatically learn features that are relevant for the given task. [24]

One important property of convolutional layers is that they are translation invariant, meaning that the learned filters can detect the same features regardless of their location in the input image. This is because the same filter weights are used at every spatial location in the input image. This is in contrast to fully connected networks where a one-pixel shift of an image results in a completely different output. [24]

3.8 Pooling Layers

Pooling layers are used to down-sample the feature maps by replacing each small neighbourhood, say 2x2, of values with a single value, usually either the maximum (max pooling) or the average (average pooling) of the values in the neighbourhood. This has the effect of reducing the spatial resolution of the feature maps, while retaining their most salient features. Pooling helps to reduce the number of parameters in the network, which can help to prevent overfitting and improve generalisation performance. [24]

3.9 Residual Connections

In a standard deep neural network, each layer receives the output of the previous layer as its input. However, as the depth of the network increases, it becomes increasingly difficult for the network to learn useful representations, due to vanishing gradients, i.e., when the gradients propagated backwards through the network become very small, making it difficult to update the parameters of the earlier layers. [26]

Residual connections help to address this problem by providing a shortcut path for the gradients to flow directly from the output to the input of a block of layers. This allows the network to more easily learn the identity mapping between the input and output, as well as any additional features that are learned by the block of layers. [26]

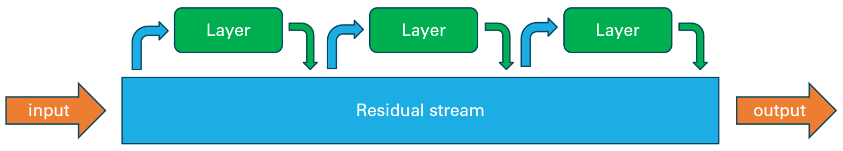


Figure 11: Illustration of the residual stream concept.

When networks become very deep it makes more sense to see the residual connection as the default path since the magnitude of the residual activations will be much larger than the activations of the actual layers. Figure 11 depicts the residual stream which can be thought of as a very high dimensional data storage. Each layer copies data from the residual stream performs some form of computation on the data and then inserts the result back into the residual stream. With this framing it is clear that a network layer does not exclusively interact with the output of the previous layer, but rather combines features from the outputs of many different previous layers creating complex circuits.

3.10 Batch Normalisation and Layer Normalisation

Normalisation is a critical step in neural networks as it standardises the feature ranges, which facilitates network convergence [27]. It also helps mitigate issues in deep networks like the vanishing/exploding gradient problem, where repeated multiplication of numbers less than 1 or greater than 1 results in rapid decay or explosion, respectively. Normalisation layers address this problem by scaling outputs into a consistent range. Two common normalisation techniques are batch normalisation and layer normalisation, each having learned parameters, often denoted as γ and β, for scaling and shifting the normalised values.[27] [28]

Batch normalisation works by rescaling each feature in a batch to have a mean of 0 and a variance of 1 during the training phase [27]. For instance, if we have a batch size of 32, for each feature x, we subtract the mean and divide by the variance of that feature over the 32 samples. However, during the evaluation phase, it uses the moving average of the mean and variance computed during training. This approach is computationally efficient, improves performance, and accelerates network convergence. Networks like ResNet utilise batch normalisation after every convolutional layer [20]. However, batch normalisation encounters challenges with small batch sizes, as the mean and variance estimates may be inaccurate, reducing the benefits of normalisation. In an extreme case with a batch size of 1, the variance is 0, causing division by zero. Larger batch sizes are often memory-constrained, particularly with large inputs like high-resolution images. [27]

Contrarily, many modern neural networks such as ConvNeXt and Swin Transformer utilise layer normalisation [22] [23]. Layer normalisation operates differently by normalising the features within each data point, rather than across a batch. That is, for each individual data point, it computes the mean and variance over all the features in a layer. For instance, if we have a layer with 64 features, this would mean subtracting the mean and dividing by the variance for each feature across all 64 features in that data point. Unlike batch normalisation, layer normalisation behaves the same way during both training and evaluation phases [28].

3.11 Regularisation

Regularisation is a set of techniques used to decrease overfitting of a model. Overfitting occurs when a model is too complex and has been trained for too long. The network has learned to model the noise in the training data instead of the underlying patterns that generalize to new, unseen data. [11]

3.11.1 L2 Norm

L2 regularisation adds a penalty term to the loss function proportional to the square of the weights: [11]

$$\lambda \sum_{j=1}^{n} w_j^2 \tag{5}$$

Where λ is the regularisation parameter and $w_j \in 1...n$ are the weights. This discourages the model from assigning too much importance to any single feature, thereby distributing the weights more evenly. λ, plays a crucial role in this process. If λ is set to zero, the regularisation term will have no effect, while if λ is set to a very large value, the weights will be heavily penalized, leading to a model that is too simple and likely underfitting the data. [11]

3.11.2 Dropout and Stochastic Depth

Dropout is a regularisation technique where a certain percentage of neurons are randomly dropped out (i.e., set to zero) during each training iteration. This technique helps to prevent overfitting and encourages the model to learn more robust features. By randomly dropping out neurons during training, dropout forces the remaining neurons to learn more robust and generalized features. This is because the neurons cannot rely on the presence of specific other neurons to make correct predictions, as they might be dropped out. As a result, the neurons become less co-adaptive and more capable of processing inputs independently, leading to better generalization. During validation and testing, all neurons are used, and the output is scaled by the dropout probability to ensure that the expected output is the same as during training. [29]

The idea behind stochastic depth is like to that of dropout, but instead of dropping individual neurons, entire layers are dropped. This technique can be seen as a more aggressive form of dropout, as it allows for larger parts of the network to be turned off during training. The probability of dropping a layer is typically increased as the network depth increases, as dropping early layers would be catastrophic for performance. Again, no layers are dropped and the output is scaled during validation and testing.[30]

3.12 Fine-Tuning Pre-Trained Networks

Fine-tuning a pre-trained image classification model is a form of transfer learning, where the pre-trained model is used as a starting point for training a new model on a smaller, domain-specific dataset. Instead of training the new model from scratch, fine-tuning the pre-trained model allows us to leverage the knowledge and features learned by the pre-trained model on a larger, more general dataset, and apply it to the new, more specific task at hand. Training a deep neural network from scratch on a small dataset can result in overfitting, where the model learns to memorize the training data rather than generalize to new, unseen data. Using a pre-trained model as a starting point for fine-tuning can help to mitigate this problem, as the model has already learned relevant features from the larger dataset. Fine-tuning a pre-trained model can also be much more computationally efficient than training a model from scratch. It is possible that only some layers need to be fine-tuned and the required amount of data needed to reach a certain performance might be much lower. [31]

In this report we will be fine-tuning ResNet, Swin Transfomer V2 and ConvNeXt V2.

3.13 ResNet

ResNet, short for Residual Network, is a deep neural network architecture that was introduced by researchers at Microsoft in 2015 [20]. ResNet is designed to address the problem of degradation in very deep neural networks, where the accuracy of the network starts to decrease due to the vanishing/exploding gradient problem.

In practice, a ResNet consists of a series of residual blocks, each of which contains several convolutional layers, batch normalization, and ReLU activation functions. Each residual block has a shortcut connection that bypasses the convolutional layers and adds the input of the block to its output.

ResNet comes in many different sizes from ResNet18 with 18 layers to ResNet152 with 152 layers. In this report we will be evaluating fine-tuned versions of ResNet18, ResNet34 and ResNet50 which have 12, 22 and 26 million parameters respectively.

3.14 Transformers and Natural Language Processing

To understand how transformer networks work for image processing we first need a history lesson about how transformer networks for natural language processing were developed and in order to do that we need to understand RNNs and LSTMs.

3.14.1 RNNs and LSTMs

RNNs or Recurrent Neural Networks, are a type of neural network designed to work with sequential data. They use feedback loops to carry information through time, allowing them to process input sequences of varying length and produce corresponding output sequences. One major challenge with RNNs is the difficulty of retaining information over long time periods due to the "vanishing gradient" problem, and it can cause the network to forget important information from earlier time steps. [32] LSTMs or Long Short-Term Memory networks aim to reduce this problem. In an LSTM, the hidden unit is replaced by a more complex cell that contains multiple "gates" that control the flow of information. These gates allow the network to selectively remember or forget information from previous time steps. [33]

3.14.2 Transformer Networks

Transformer networks are a type of ANN which learn context and thus meaning by investigating relationships in ordered data such as sentences or images. Transformers have become very popular for natural language processing (NLP) tasks and now increasingly for computer vision (CV) tasks. First introduced by Vaswani et al. in "Attention is all you need" 2017 [34], transformers work by using "self-attention". Previous NLP networks such as RNNs and LSTMs had issues with vanishing gradients and with hardware parallelisation. The introduction of attention mechanisms let a network draw from the state at any preceding point along the sequence. The attention layer can access all previous states and weigh them according to a learned measure of relevance, providing relevant information about far-away tokens. While RNNs would need to perform many steps to relate words far apart in a sentence a transformer would only need a single step [32].

3.14.3 Decoder Only Transformer Networks

A classic transformer network consists of an encoder block and a decoder block, however for many applications, including ours, a simpler decoder only network is more appropriate.

A decoder only transformer consists of an input embedding and positional encoding layer, followed by many decoder blocks and finally an output embedding layer [35]. Each decoder block consists of masked multi-head attention layers, feed forward and “Add + Norm” layers. The architecture can be seen in Figure 12 below.

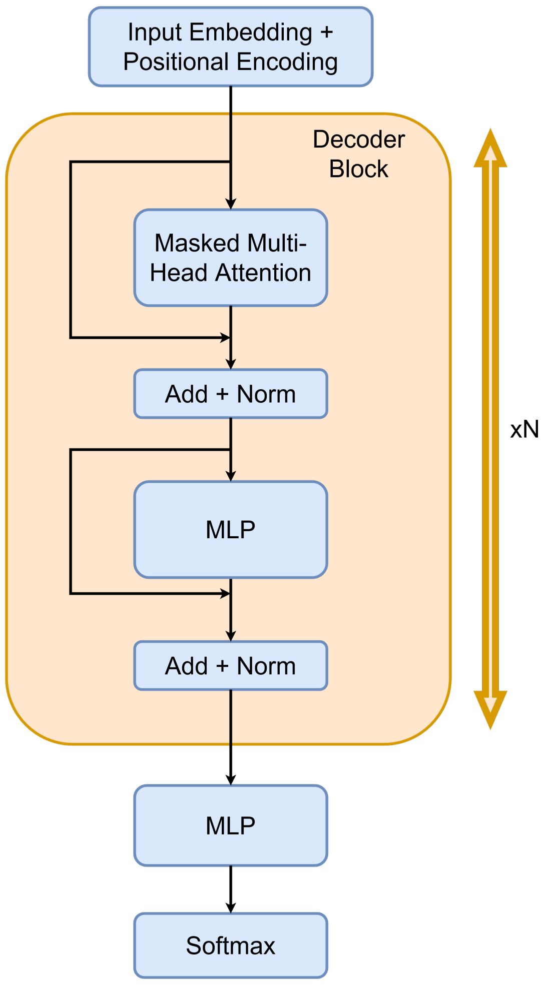

Input Embedding + Positional Encoding
Decoder Block
Masked Multi-Head Attention
Add + Norm
MLP
Add + Norm
xN
MLP
Softmax

Figure 12: A decoder only transformer network. The network consists of an input embedding and positional encoding, followed by N identical decoder blocks, followed by a MLP (fully connected layer) and a softmax. The decoder block itself consists of a masked multi-head attention layer followed by an MLP.

3.14.4 Input and Output Embeddings

The input embedding layer maps the input sequence into a higher dimensional vector space, where each token in the sequence is represented as a vector. The output embedding layer instead maps these high dimensional vectors back into tokens. It used to be the case that the output embedding matrix was simply the inverse of the input embedding matrix, but this is no longer the case for state of the art transformer networks [35].

Embeddings can either be made “manually” by assigning each word in a dictionary a one-hot encoding which used to be the standard or more recently by having the network learn how best to tokenise sentences. Learned embeddings often split words into several with a word on average being 4/3 tokens long. [35]

3.14.5 Positional Encodings

While RNNs and LSTMs implicitly know the order of words in a sentence since they are fed the tokens one at a time, transformers are fed the entire input sequence at the same time, meaning they lose the order of the words. In order to solve this, the ordering is explicitly added back to the input. A naive way to do this would be to simply assign each position a unique integer, but this would make it difficult for the model to generalize to sequences of different lengths. If we were instead to assign each position a number between 0 and 1, this would cause issues with normalisation. Instead one could use sine and cosine functions to generate unique vectors for each position, which can be seen in equation 6: [34]

$$\mathrm{PE}(i,j) = \begin{cases} \sin\left(\frac{i}{10000^{2j/d_{\text{model}}}}\right) & \text{if } j \text{ is even} \\ \cos\left(\frac{i}{10000^{(2j-1)/d_{\text{model}}}}\right) & \text{if } j \text{ is odd} \end{cases} \tag{6}$$

Where $\mathrm{PE}(i,j)$ represents the positional encoding for the i-th token and j-th dimension of the embedding, d_{model} represents the dimensionality of the embedding space, and i ranges from 0 to the length of the sequence minus one. In other words, the positional encodings are a NxM dimensional matrix where N is the sequence length and $M = d_{\text{model}}$ is the dimensionality of the embedding space. Recently some networks have stopped using sine and cosine based positional encodings in favour of letting the network learn its own positional encoding functions.[34]

3.14.6 Attention

Attention is a mechanism used in neural networks to selectively focus on different parts of an input sequence while processing it. It is inspired by the way humans selectively attend to different parts of sensory input [34].

After tokenising a sequence into high dimensional embedding vectors, as described in the previous section, each token is transformed into a set of “Query”, “Key”, and “Value” vectors using three different learned linear projections. The query vector represents the current position being processed, while the key and value vectors represent the other positions in the input sequence. The attention is computed according to the following equation [34]:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_{model}}}\right)V \tag{7}$$

Where Q is the query matrix, K is the key matrix and V is the value matrix. The coloumn vectors of these matricies are the query, key and value vectors respectively, for each token in the input. Again d_{model} is the dimensionality of the embedding space.

In other words, a similarity score is computed between the query vector and each of the key vectors using a dot product. The similarity scores are then normalised using a softmax function to obtain attention weights, which represent how important each position in the input sequence is for the current position. The $\sqrt{d_k}$ in the softmax function is a scaling factor which has been found to improve performance. The attention weights are then used to compute a weighted sum of the value vectors, which produces the final output at the current position. This weighted sum gives more importance to the parts of the input sequence that are most relevant for the current position.

3.14.7 Multi-head Attention

With a multi-head attention mechanism, the query, key and value vectors are then split into several parts and then fed to separate attention “heads” which apply different learned transformations to the vectors. The outputs of each attention head are concatenated together and then transformed, which forms the final output of the multi-head attention mechanism. [34]

The main advantage of using multi-head attention is that it allows the model to attend to multiple parts of the input sequence simultaneously and to learn different representations of the input sequence. This helps in capturing more complex patterns and dependencies in the input sequence, leading to better performance [34].

3.14.8 Masked Attention

Self-attention computes how the tokens in the output sequence relate to each other. During training this layer will by default have access to future tokens in the output sequence, which is obviously undesirable as the network will then only learn to use the real token to predict the output token. Therefore, we need a masked attention layer, where the effect of future tokens is nullified in order to force the network to predict the next token in the output sequence based only on the input sequence and previous tokens in the output sequence. For computational efficiency reasons the entire output sequence is still fed into the network, but future tokens are later masked in the masked-attention layer. [34]

3.15 Transformers and Image Processing, Swin Transformer

Despite first being developed for natural language processing, the transformer architecture has, with some modifications, been applied to images as well. While images are not sequences, they still have positional, typically 3D (height, width and RGB), structure. A large problem that arises is that (at least without strategies to reduce it) the computational complexity of attention is quadratic to the input size. This is much worse for images than for text since an image typically contains many more pixels than a text contains tokens. Therefore, attention typically is not applied to the entire image at the same time, but to smaller patches that might later be merged, in a similar way to how CNN filters do not look at the entire image at the same time. [22]

3.15.1 Swin Transformer V1

Swin, or Shifted WINdow transformer, was introduced by Liu et al. in their 2021 paper, “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows [22]. The Swin Transformer is designed for computer vision tasks, such as image classification, object detection, and semantic segmentation, and has achieved state-of-the-art performance on various benchmarks [22]. Swin has linear computational complexity as a function of the number of pixels in the image. In order to achieve this, it uses hierarchical feature maps and shifted window attention. The high level structure can be seen in Figure 13 below.

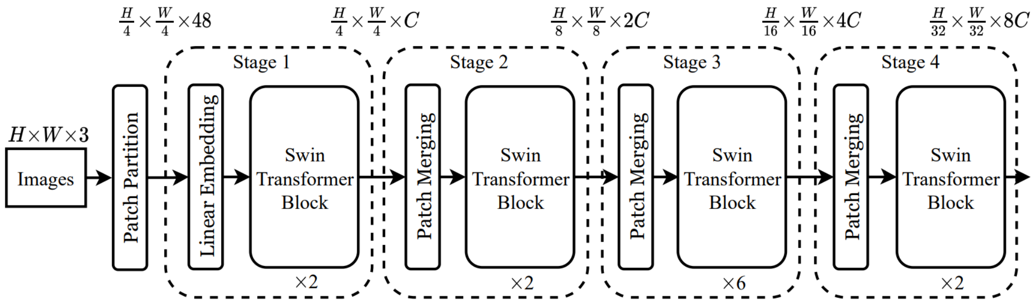


Figure 13: Depiction of the Swin Transformer V1 architecture, taken from [22].

3.15.2 Hierarchical Feature Maps and Patch Merging

Hierarchical feature maps refer to the representation of input images at multiple scales, with a gradual reduction in spatial resolution and therefore an increase in the receptive field. This allows the model to learn features at different levels of granularity, from local to more global representations. [22]

In CNNs, the downsampling of feature maps is often achieved using pooling layers. These layers reduce the spatial dimension by aggregating the values within small, non-overlapping regions, say 2x2, in the feature map.

In Swin Transformers, by contrast, the downsampling is performed using patch merging. The patch merging process works as follows: The input image is divided into non-overlapping patches, say 4x4 pixels, at the beginning of a block. These patches are then linearly embedded into vectors and treated as tokens. [22]

After processing the tokens through a series of Swin Transformer layers, the feature map is effectively down-sampled using patch merging. Patch merging reduces the spatial dimension by concatenating the features of neighbouring patches. To down-sample the feature maps by a factor of n, patch merging groups n x n neighbouring patches and concatenates their features into a single new patch. For example, to down-sample a feature map by a factor of 2, a 2x2 group of neighbouring patches is combined into a single new patch. [36]

3.15.3 Shifted Window Attention

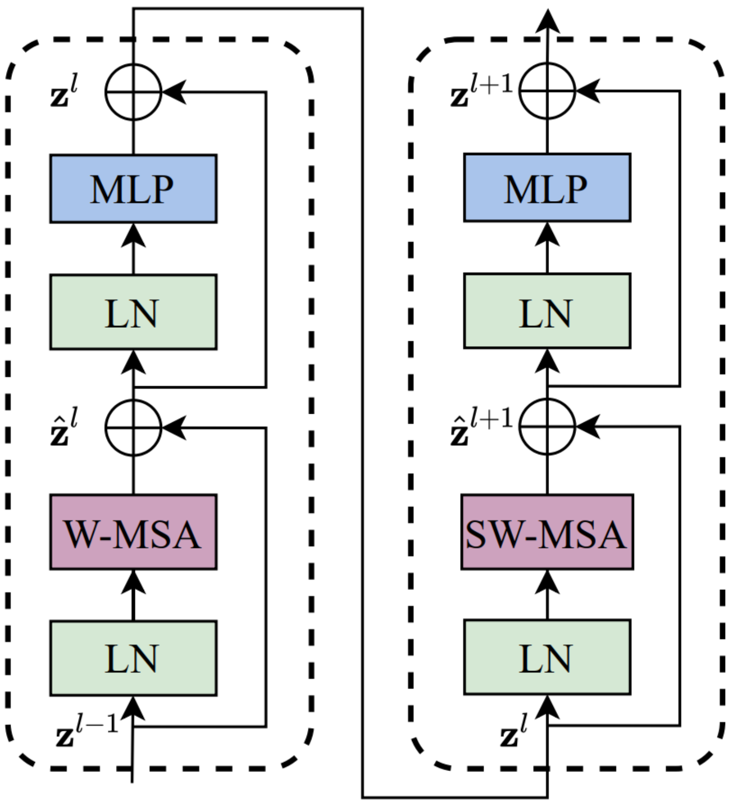


Figure 14: A diagram of a single Swin Transformer block, taken from [22].

Zooming into a Swin Transformer block, which is depicted in Figure 14 above, we see that Swin Transformer block utilises a modified version of the standard multi-head self-attention (MSA) module used in other transformers. It uses Window MSA (W-MSA) and Shifted Window MSA (SW-MSA) modules instead. Each Swin Transformer block consists of two sub-units, with a normalization layer, an attention module, another normalization layer, and an MLP layer. The first sub-unit uses W-MSA, while the second sub-unit uses SW-MSA. In addition to this we find residual connections bypassing the MSA and MLP layers. [22]

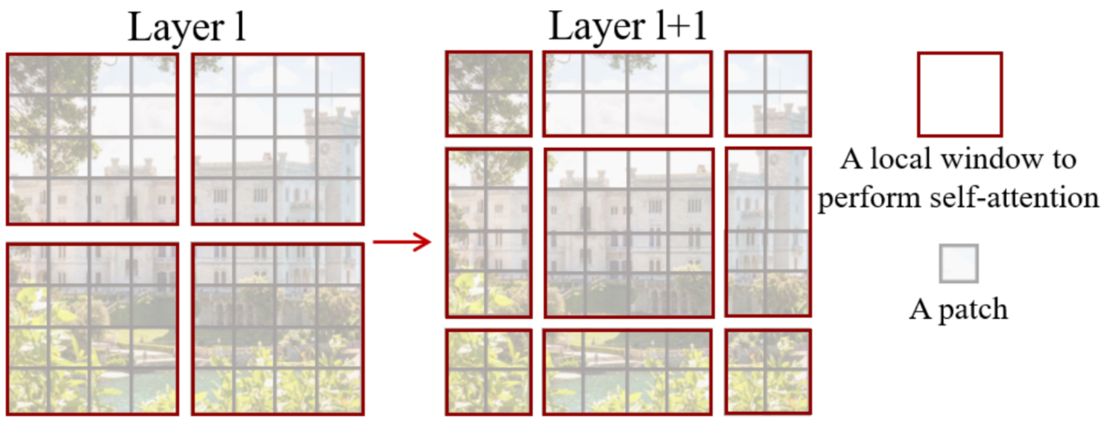


Figure 15: An image illustrating how the windows alternate from layer to layer, hence the "shifted window". Image taken from [22].

In contrast to global self-attention, which has quadratic complexity, the Swin Transformer uses window-based MSA to compute attention only within fixed-size windows. This reduces the complexity to be linear with respect to the number of patches, making it more suitable for high-resolution images, than a global approach. [22]

Window-based MSA, however, limits the network's modelling power, as self-attention is restricted to within windows. To address this, the Swin Transformer applies Shifted Window MSA after the W-MSA module. This means changing which patches constitute which window, as shown in Figure 15. Each new window consists of 4 neighbouring quarters of the old windows. This introduces important cross-connections between windows and improves network performance. As in the NLP case, the transformer does not by default have any positional information. To address this, the attention function is modified:

$$\text{Attention}(Q, K, V) = \text{SoftMax}\left(\frac{QK^T}{\sqrt{d}} + B\right)V \tag{8}$$

The difference between equation 8 and the attention equation 7 used in section 3.14.6 is the B term, which is a relative position bias. This bias is not hard coded but instead learned during training. [22]

3.15.4 Swin Transformer V2

Swin Transformer V2 introduces some improvements over the original Swin Transformer (V1) primarily aimed at scaling up model capacity for larger images. The most important of these are:

- A switch from pre-norm to post-norm was made. In V2, the order of the layer normalization layers was switched so that normalization happens after the attention and MLP layers, not before. Apparently, this reduces instability for large images [36]
- Scaled Cosine Attention. In V2, the dot product attention, described in section 3.14.6 is replaced by scaled cosine attention:

$$\text{Attention}\left(\mathbf{q}_i, \mathbf{k}_j\right) = \cos\left(\mathbf{q}_i, \mathbf{k}_j\right)/\tau + B_{ij} \tag{9}$$

Where q_i is the element from the query matrix corresponding to pixel i, k_j is an element from the key matrix corresponding to pixel j, B_{ij} is the relative position bias between pixel i and j and τ is a learnable scalar which is unique for each head and layer. Scaled cosine attention causes attention maps to be less dominated by a few pixel pairs [36].

- Log-Spaced Continuous Position Bias: In Swin Transformer V1 changing the image size would mess up the learned relative position bias. To resolve this, V2 uses a small meta network on the relative coordinates to generate bias values for arbitrary relative coordinates, making it naturally transferable to varying window sizes during fine-tuning. This network uses Log-Spaced coordinates.

Like ResNet, Swin Transformer V2 comes in many different sizes, we will be training the T, S and B versions which have 29, 50 and 88 million parameters respectively.

3.16 ConvNeXt

ConvNeXt is a family of pure CNN models that are inspired by and designed to compete with transformers in terms of accuracy and scalability for various computer vision tasks. ConvNeXt makes several key changes compared to a traditional ResNet architecture, namely: [23]

- ConvNeXt uses GELU rather than ReLU as an activation function. Typically, GELU results in slightly better performance than ReLU and is used by most state of the art transformers. ConvNeXt also reduces the number of activation functions to only one per block rather than one per layer.
- ResNet uses a large 7x7 convolution layer followed by a maxpooling as its first layer. ConvNeXt swaps this out for a “patchify layer” a 4x4 stride 4 convolutional layer inspired by a similar layer in Swin Transformer.
- ConvNeXt swaps all batch norm layers for layer norm layers and also reduces the number of normalisation layers. layer norm is a simpler operation and allows training with smaller batch sizes.
- ConvNeXt implements grouped convolutions and depth wise convolutions. Grouped convolutions divide the convolutional filters into different groups, so that each group only operates on a subset of the input channels. This reduces the number of computations and parameters, and therefore allows for wider networks. Depth wise convolutions are the extreme case of this where each group only has one channel, this is similar to self-attention in transformers, which also separates spatial and channel mixing.

ConvNeXt comes in several sizes, we will be training the T and S versions which have 29 and 50 million parameters respectively.

3.17 Improving Performance

3.17.1 Quantisation

Quantisation is a process that allows for reducing the precision of the numerical representation of tensors in order to reduce the time and memory consumption of the model.

There are several types of quantisation, including post-training static quantisation, quantisation aware training, and dynamic quantisation [37]. They can be applied both to the weights and activations (the outputs of layers).

The first step in post-training static quantisation is adding observers to the network after it has finished training. The most frequently used type of observer is the min-max observer which is a module which records the running minimum and maximum of tensors during the model's forward pass during inference. These values are then used to determine the scale and zero point for quantisation. The scale is computed as the difference between the max and min divided by the quantisation range (255 for int8), and the zero point is computed as the value that will be used to represent 0 in the original floating-point range. In order to get a good, quantised network you must therefore calibrate the network on a small representative sample of the data, in order to determine the optimal observer parameters. [37]

3.17.2 Fusing

Fusing is a process in which multiple operations are combined into a single operation for efficiency. This is particularly beneficial in the context of quantisation, where each individual operation may otherwise require dequantisation and re-quantisation. By fusing operations together, computation can be performed in the quantised domain. A common fusion in CNNs is the combination of convolution, batch normalization, and ReLU into a single operation. [37]

3.18 Data Augmentation

Data augmentation refers to the process of creating new data samples from the existing ones by applying various techniques. This is primarily done to increase the size and diversity of the training dataset. This helps improve the model's generalisation performance, making it more robust and perform well on unseen data. [38]

One type of image augmentation involves geometric transformations such as rotating, scaling, flipping, and translating. By rotating an image by different angles, resizing it using scaling factors, mirroring it horizontally or vertically, and shifting it along the x and/or y-axis, additional training data is created, reducing overtraining. The model becomes more robust and adaptable, enabling it to better identify objects regardless of their orientation, size, or location.[38]

Another type of augmentation is intensity transformations or pixel value transformations. Brightness, saturation, hue and contrast adjustment, noise injection, and colour jitter are image augmentation techniques. The model becomes more resistant to variations in pixel values and colouration that could arise from different lighting conditions, environmental factors, and colour variations.[38]

3.18.1 CutMix

CutMix is an advanced augmentation technique which has been shown to improve the performance of ResNet on ImageNet [39].

First, two training images are randomly selected. These images do not have to be similar or from the same class. A random bounding box is then cut in the first image and replaced with a bounding box from the second image. The size and location of the bounding box is chosen randomly. The labels of the two images are then mixed proportionally to the number of pixels from each image in the resulting image. The loss function is also correspondingly modified. [39]

CutMix provides a way for the model to see and learn from parts of multiple images and their labels at the same time. This can help the model to generalize better and improve its performance on unseen data. [39]

3.19 Interpretability Methods

An important part of machine learning is to interpret results and to understand what the model is looking for in the data. We try to make the black box a bit lighter. [40]

3.19.1 Grad-CAM

Grad-CAM, or Gradient-weighted Class Activation Mapping, is a technique for understanding and visualising the regions of an input image that contribute most to a particular prediction made by a network. Grad-CAM was originally introduced for CNNs but can be made to work for transformer-based models too.

Grad-CAM works by computing the gradient of the predicted class score with respect to the feature maps of the last convolutional layer in the CNN. The gradients are then globally averaged to obtain the weight for each feature map, which is then used to compute a weighted sum of the feature maps. [41] The resulting map highlights the regions in the feature maps that are most relevant to the prediction made by the network.

The output of Grad-CAM is a heatmap that can be overlaid on top of the input image, allowing us to visualise which regions of the image the network is focusing on to make a particular prediction. [41]

While Grad-CAM can give insights into which parts of an image a CNN is focusing on, it may not always reflect the true decision-making process of the model. A network could be focusing on the correct region of the image for the wrong reasons [42]. Experiments have been made that show Grad-CAM is very sensitive to small geometric transformations and can often show misleading results [42].

3.19.2 Occlusion

Occlusion is another method of figuring out what a network is looking at. It works by placing a square or circular mask over a portion of the input image, effectively hiding that region of the image from the model. The model is then asked to predict the label of the image with the masked region, and the accuracy of the prediction is recorded. By doing this for all regions of the image, we can measure which regions have the greatest effect on classification when covered up, thereby indicating which part of the image is most important to the network. It is also possible to occlude channels rather than occlude spatially. We will be utilising this when feeding the network multiple images captured from the same location but with different illumination angles in order to see which images contribute most to the network's performance. [40]

3.19.3 Maximum Class or Neuron Activation

The maximum activation approach, also known as gradient ascent, involves finding the input image that maximizes the activation of a particular neuron or class in the network. This can be done by starting with a random or noise image and then using gradient ascent to iteratively update the image in the direction that maximizes the activation of the neuron or class of interest. [43]

Due to the nature of how maxpooling layers interact with convolutions this typically leads to a lot of high frequency noise, to reduce this effect and get more sensible looking activations several regularising techniques can be performed: [43]

- Total Variation (TV) regularisation: TV regularisation encourages smoothness in the input image by penalizing large differences between neighbouring pixels. Adding TV regularisation to the gradient ascent process can help produce more visually coherent and interpretable maximally activating inputs.
- Gaussian Blur: Applying Gaussian blur to the input image during gradient ascent can help reduce high frequency noise and generate smoother Visualisations. This can be done by periodically blurring the input image at each iteration or incorporating it directly into the optimisation objective.
- L2 or L1 Penalty on Input: Adding an L2 or L1 penalty on the input image during gradient ascent can encourage the input to be close to a predefined reference image or a neutral input. This can help prevent extreme input values.
- Clipping Input Values: Clipping the input values within a certain range during the gradient ascent process can prevent extreme pixel values and ensure the resulting maximally activating inputs remain within a reasonable domain.
- Jitter: Adding random jitter (translations) to the input image during gradient ascent can encourage the network to focus on more invariant features.

3.20 Uniform Manifold Approximation and Projection, UMAP

Uniform Manifold Approximation and Projection (UMAP) is a dimensionality reduction technique that can be used for visualisation. UMAP was introduced in 2018 [44] as an alternative to the older technique t-SNE, as it is computationally more efficient.

The first step of UMAP is to create a high dimensional graph of the data. For each data point, UMAP identifies the nearest neighbours and uses these to construct a weighted graph. This graph is designed to capture the manifold on which the data lies. The weights of the edges are chosen to make this graph a fuzzy topological representation. This means that instead of having binary "is a neighbour / is not a neighbour" relationships, the graph has a spectrum of connection strengths based on the distance between points. [44]

The second step is to optimise a low-dimensional graph to be as structurally similar as possible to the high dimensional graph. The UMAP algorithm does this by minimising the cross-entropy between the two fuzzy topological representations. This step makes use of stochastic gradient descent-based methods to find the best lower-dimensional representation of the data. [44]

3.21 Tree-structured Parzen Estimator Hyperparameter Optimisation

Tree-structured Parzen Estimators (TPE) is an optimisation method used for hyperparameter tuning, i.e., finding optimal hyperparameters. [45]

TPE models the conditional probability $P(x|y)$ and the marginal probability $P(y)$, where x represents hyperparameters and y is the associated loss [45]. $P(x|y)$ is modelled differently for y less than a certain quantile q and $y > q$. q is selected such that points to one side of it are likely to yield better results than the other side, i.e. it splits the configuration space into two. TPE learns two different distributions for these two regions: $l(x)$ for good outcomes and $g(x)$ for bad outcomes. When sampling new points to try, TPE generates points from the distribution $l(x)/g(x)$. It is therefore more likely to sample points from regions of the space that have led to good outcomes, but still sometimes explores the "bad" region. It continually updates its parameters and over time, TPE will converge on the best hyperparameters. [45]

TPE has several advantages over other hyperparameter tuning techniques like grid search and random search. It uses information from past trials to inform the next set of hyperparameters to try and can quickly converge on the optimal set of hyperparameters. [45]

4 Method

4.1 Data Collection

All data was captured by us using CellaVision's DC-1 microscope with a 20x objective lens and a numerical aperture, NA, of 0.5 which implies that it has a θ_{NA} of 30° (see equation 1). A sketch of the setup can be seen in Figure 16. For some of the slides (around 65) bone marrow particles had already been segmented by an expert and we had their outlines in a low resolution preview scan of the slides. These were almost exclusively real particles, so we added several regions of things which looked vaguely like particles but were not marked as such. The coordinates of the real and fake were extracted, and a script was written to tell the microscope to go to these regions, auto focus and then capture and save the images. The remaining slides (about 50) were scanned, and we selected regions which looked like particles or might be confused with particles. These regions were then scanned with a high-resolution 100x microscope and sent to an expert for classification. After classification the images were captured with the same script as before.

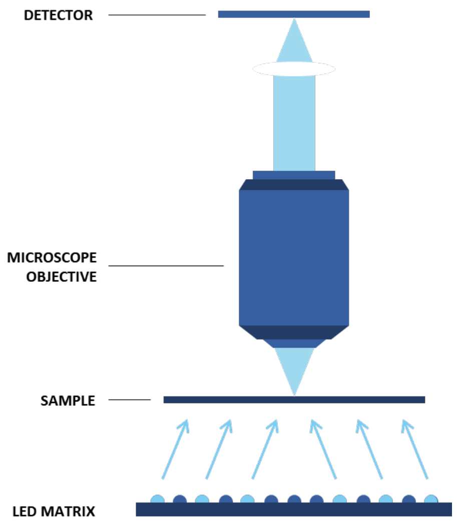


Figure 16: Setup with led matrix and microscope.

4.1.1 Autofocus

Autofocusing was done by first capturing images every 500 μm in a specific range, blurring the image with a Gaussian-filter to reduce the effect of noise, then using a Laplace filter and calculating the variance of the output. A new interval was then chosen to be between the points neighbouring the highest variance point and then the same procedure was repeated for 50 μm and finally 5 μm. The highest variance point at the 5 μm level was used to determine the focal length for capturing images of the entire particle. This procedure is based on the idea that the variance of the Laplacian is basically measuring contrast, which would be maximal at the correct focus.

4.1.2 Overlap

The area covered by an image was $960 \cdot 600$ μm which means that most of the particles were larger than a single image. We decided to capture images which covered the entire bounding box of a particle, even if they overlap. If the particle was smaller than the image surface, the particle was instead centred within the frame by our algorithm. A few examples of how images were patched can be seen in Figure 17.

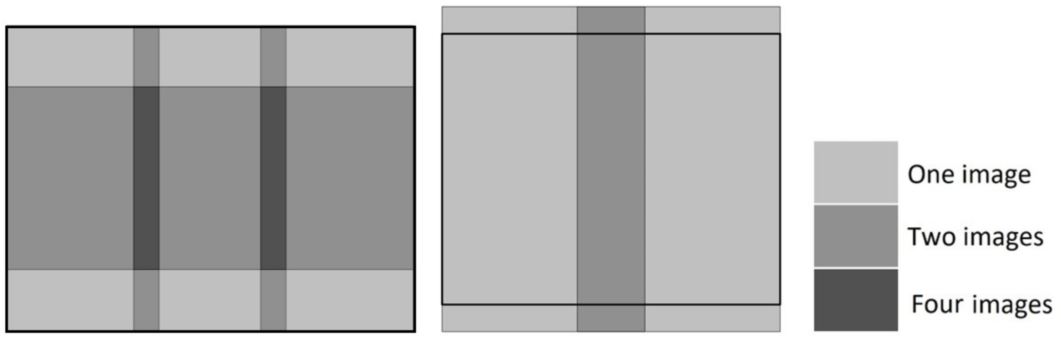
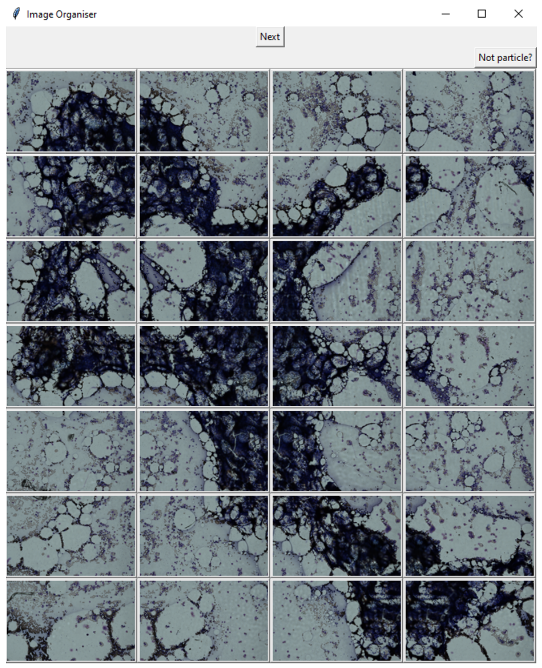
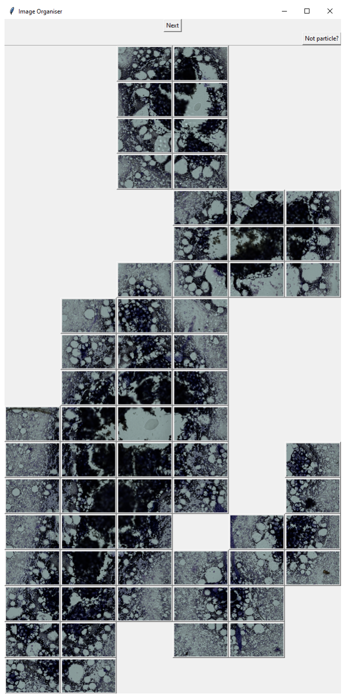


Figure 17: Example of two different overlap cases where the thick line shows the particle boundary box. The leftmost image shows an example where the bounding box for the particle is bigger than a single image. Here we fit 3 images horizontally and 2 vertically for a total of 6 images. The rightmost image shows when the image is larger than the particle bounding box in one dimension but smaller in the other, two images would be captured here.

Since our bounding boxes were rectangular and contained some margin, whereas particles are typically not rectangular, many of the images will not contain any particle. To solve this, we created a GUI which allowed us to easily remove images which only contained background. Two examples from the GUI can be seen in Figure 18.



(a) All images of a particle before background removal.



(b) A different particle after background removal

Figure 18: Example images from our background removal GUI.

4.1.3 Illumination Angles

The bright field of the objective consists of the cone defined by $\theta_{NA} = 30°$. For each image location, 32 images were captured. One image with all the LEDs within the bright field turned on and one with all the LEDs corresponding to dark field illumination angles turned on. 30 images were taken with only one led turned on at a time. The angle from which each LED illuminated the sample can be seen in Figure 19 where they are marked with red dots.

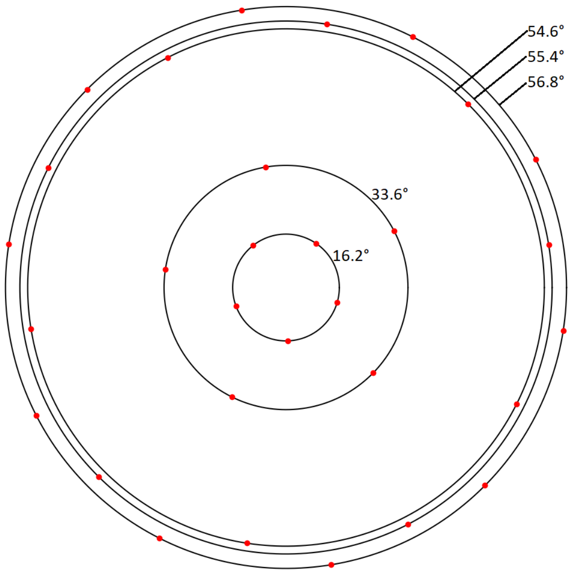


Figure 19: Positions of the point light sources marked with red dots in the circles around the optical axis.

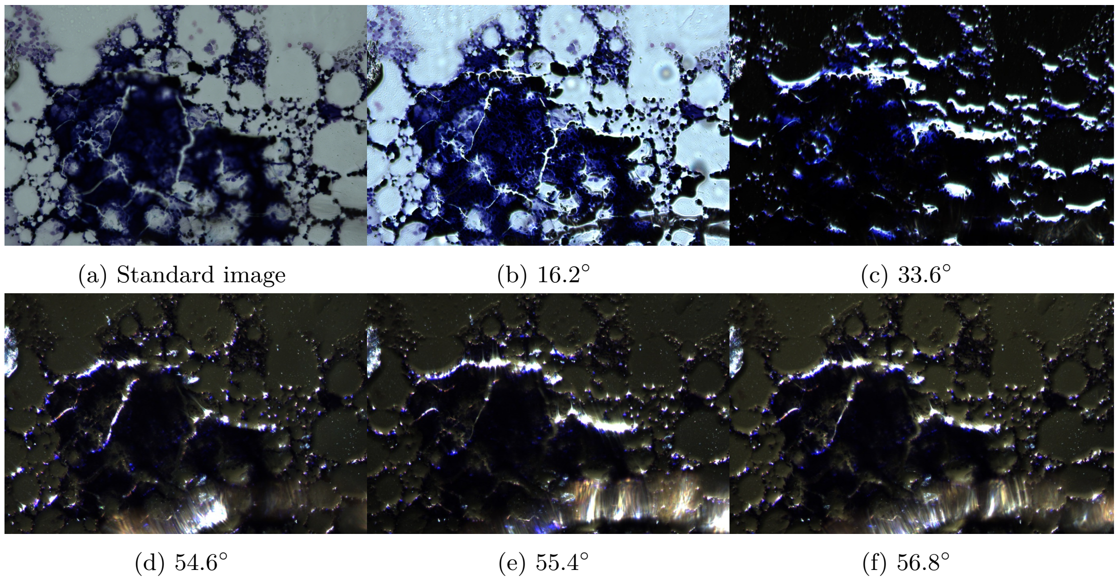


Figure 20: Examples of PLS images captured with different illumination angles.

4.1.4 Dataset

For the training set we used 66 slides with a total of 1017 real particles and 274 fake particles giving us 3741 images of real particles and 1785 of fake ones. In the validation set we had 25 slides with a total of 315 real particles and 104 fake particles giving us 1070 images of real particles and 693 fake ones. Our test set contained 377 real particles and 86 fake particles corresponding to 943 real images and 657 fake images. We tried to balance the types of coverslip, smear and stain between the datasets.

4.2 Training the Networks

Table 1: The hyperparameters which were optimised using TPE.

Hyperparameter	Types
Model type	ResNet18, ResNet34, ResNet50, SwinTV2T, SwinTV2S, SwinTV2B, ConvNeXtT, ConvNeXtS
Image type	Standard, circular darkfield, standard + circular darkfield, 30 PLS image stack
Weights frozen	Frozen, unfrozen
Augmentation	None, geometrical, image, geometrical + image, geometrical + image + CutMix
Image size	480x300, 720x450, 960x600
Pretraining	Pretrained, not pretrained

The networks were trained using a RTX3090 GPU with 24268MiB VRAM. The Tree-structured Parzen Estimators (TPE) optimisation method was run to optimise the hyperparameters shown in table 1. TPe was run for 100 iterations which means about 5% of the hyperparameter space was tested (there are 1920 possible combinations). Each iteration lasted 5 epochs. After this, we took our best model and varied each hyperparameter one at a time, training a new network for each value in order to investigate the effect of each hyperparameter. These networks were trained for 12 epochs. We used cross entropy loss as the loss function and the REX learning rate scheduling with an initial learning rate of 0.01 (see equation 2). We also investigated the effect of training on smaller training sets. Here we trained the best network of each model type (i.e., the best ResNet, best SwinT and best ConvNeXt) on 1%, 2%, 5%, 10%, 20%, 25%. 50 % and 100% of the images in the training set. These images were randomly selected.

4.3 Single PLS and occlusion experiments

To determine which PLS images might contain useful information we trained a network for each PLS image angle using our best hyperparameters, for a total of 30 networks. We then measured the average validation accuracy for each angle. We also trained a network on a stack of all 30 PLS images and used layer occlusion to figure out which layers were most important for performance. We tried occluding a specific angle and measuring the accuracy and also including everything but a specific angle and noting the accuracy.

4.4 Measuring Performance

In order to measure the time performance of the different models we ran inference on the entire validation set and measured the time. We decided to divide the time consumption into the time the data loading took and the time the actual inference took. To measure the inference time, we implemented a loop in our code to run inference 100 times in a row for each batch and then later divide the runtime by 100. In order to measure the data loading time we ran each model 100 times and computed the average runtime before subtracting the inference time consumption computed earlier.

4.5 Quantisation

We used post-training static quantisation, and convert both the weights and activation tensors from 32-bit floating-point numbers (float32) to 8-bit integers (int8), theoretically reducing the model size by 4x and making it 4x faster. We used min-max observers and calibrated on the entire training set rather than a smaller representative sample. We then fused as many operations as possible. In order to better reflect the real world application on the microscope we set the batch size to 1. In order to measure the performance we took the average of 100 inferences on the validation set on the CPU. We then divided this by the number of images to get the time per image.

4.6 Comparing the Model to Non-Expert Humans

Only half of our dataset was labelled by an expert at start of the project. After a few months of the project and we got quite good at distinguishing between particles and fakes so we tried to labelled the rest of the dataset ourselves. The rest of the dataset was later labelled by an expect and we compared our own labels to that of the bone marrow expert.

4.7 Adversarial Fake data

During training we found that our networks always misclassfied one particular fake particle in the validation set as real. We suspected that this particle was a piece of paper, so we decided to try and create extra fake data with items which might end up on a slide. This was done by placing various items like paper, hair and dust on a bone marrow slide. Images with different focus were then captured and added to the fake part of the training dataset. Note that this only effects the results in the adversarial fake data section, all other results did not contain this extra fake data. A few examples of the fake images can be seen in Figure 30 below.

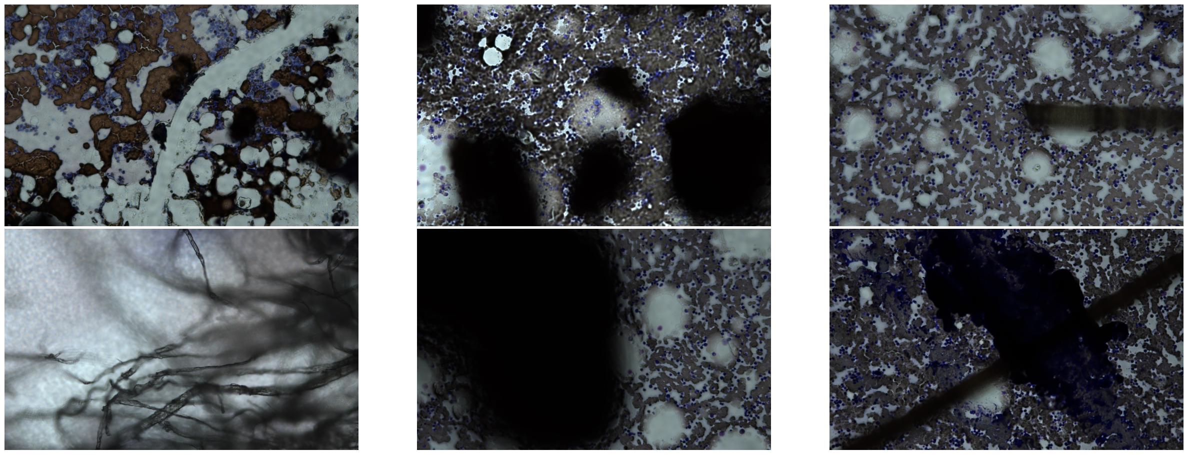


Figure 21: Examples of adversarial fake data.

4.8 Maximum activation and UMAP

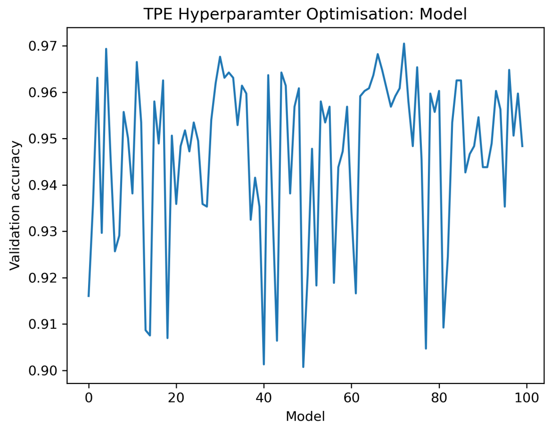
A maximum activation visualisation was done on the classification neurons for the best network in each model type (ResNet SwinT and ConvNeXt). We used a L2 regularisation of 0.01, a TV regularisation of 0.01, a Gaussian blur of 3 and random translations, rotations and scalings of 8 pixels, 5° and 10% respectively. For our UMAP visualisation we used 15 neighbours a minimum distance of 0.1, two dimensions and a euclidean metric.

5 Results

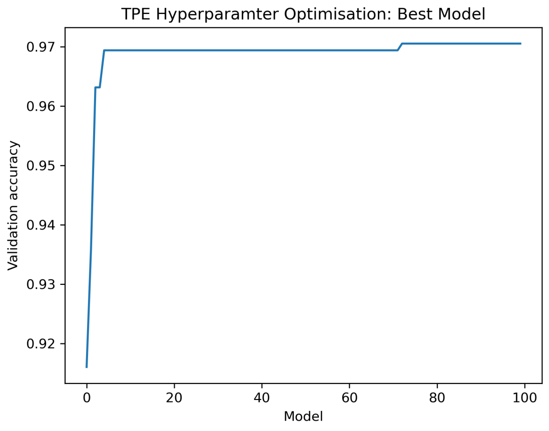
5.1 A Note on the Occasional Convergence Failure of ConvNeXt

Approximately 10-20% of the time training ConvNeXtT and ConvNeXtS simply failed, the accuracy got stuck at exactly 0.6069, corresponding to always guessing real without any improvement. We were not able to figure out why this happens, or how to resolve it, so when this happened we simply re-trained the network again with the same hyperparameters and it worked. The results shown are for the successfully trained networks.

5.2 Tree-structured Parzen Estimator Optimisation



(a)



(b)

Figure 22: TPE hyperparameter optimisation trials. To the left we see the validation set accuracy for each model and to the right we see the best found models accuracy.

The hyperparameter optimisation was run for 100 epochs and generated the best hyperparameter combination after 72 epochs, as can be seen in Figure 22b. According to the optimisation, the best results was achieved by using ConvNeXtT, standard images, unfrozen weights, image + geometric augmentation and an image size of 480 x 300 pixels.

5.3 Effect of Varying One Hyperparameter

After the hyperparameter optimisation, we varied one parameter at a time to investigate how the hyperparameters affect the validation accuracy independently of each other.

5.3.1 Model Type

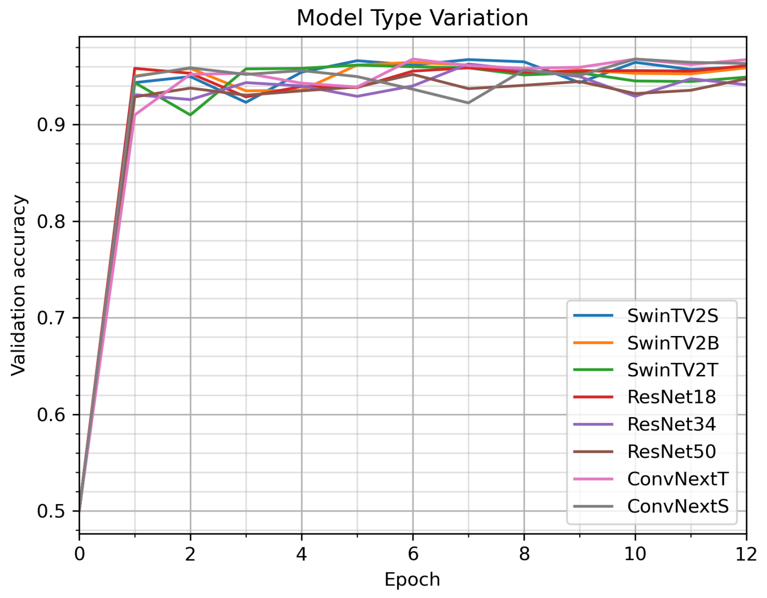


Figure 23: Validation accuracy during training for different model types. All other hyperparameters set to TPE optimum.

Table 2: Validation accuracy during variation of model type.

Model type	Validation accuracy
SwinTV2S	0.9671
SwinTV2B	0.9643
SwinTV2T	0.9614
ResNet18	0.9609
ResNet34	0.9626
ResNet50	0.9518
ConvNeXtT	**0.9677**
ConvNeXtS	**0.9677**

5.3.2 Image Type

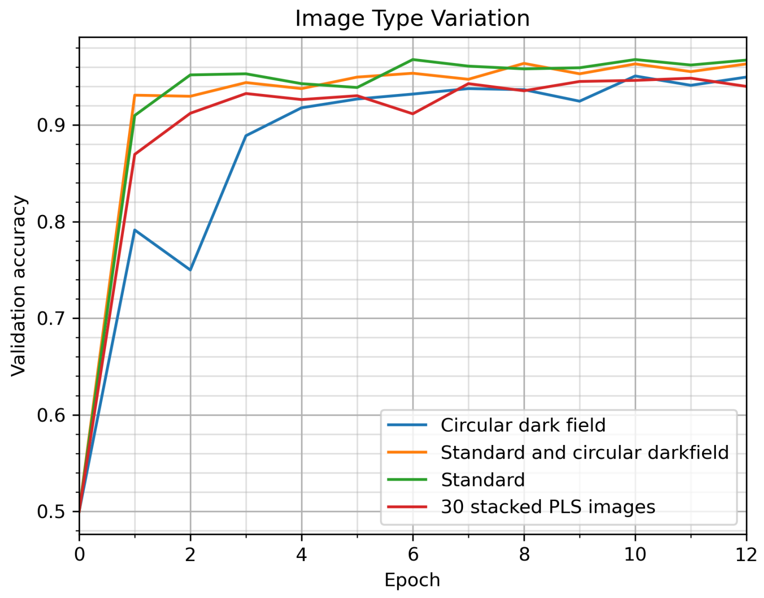


Figure 24: Validation accuracy during training for different image types. All other hyperparameters set to TPE optimum.

Table 3: Validation accuracy during variation of image type.

Image type	Validation accuracy
Standard	**0.9677**
Circular dark field	0.9507
Standard and circular dark field	0.9637
30 stacked PLS images	0.9484

5.3.3 Frozen Weights

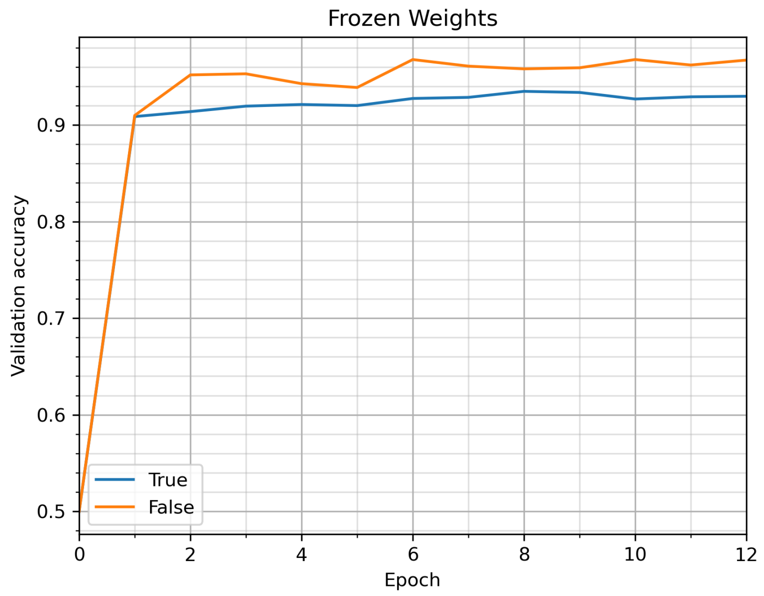


Figure 25: Validation accuracy during training while freezing all but the last layer vs unfrozen. All other hyperparameters set to TPE optimum.

Table 4: Validation accuracy during variation of frozen weights.

Frozen weights	Validation accuracy
True	0.9348
False	**0.9677**

5.3.4 Augmentation

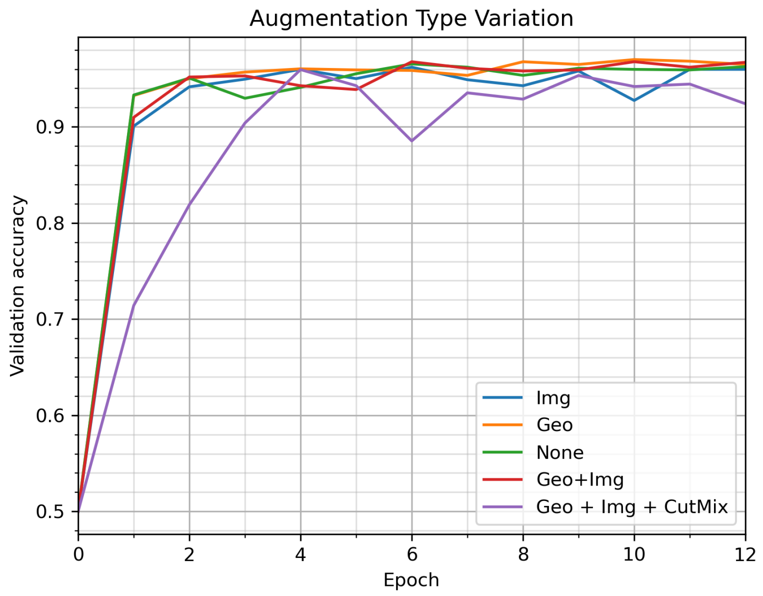


Figure 26: Validation accuracy during training for different augmentation types. Geo refers to geometrical augmentation and Img refers to image augmentation. All other hyperparameters set to TPE optimum.

Table 5: Validation accuracy during augmentation variation.

Augmentation	Validation accuracy
No augmentation	0.9654
Image augmentation	0.9620
Geometric augmentation	**0.9699**
Image and geometric augmentation	0.9677
Image, geometric and CutMix augmentation	0.9594

5.3.5 Image Size

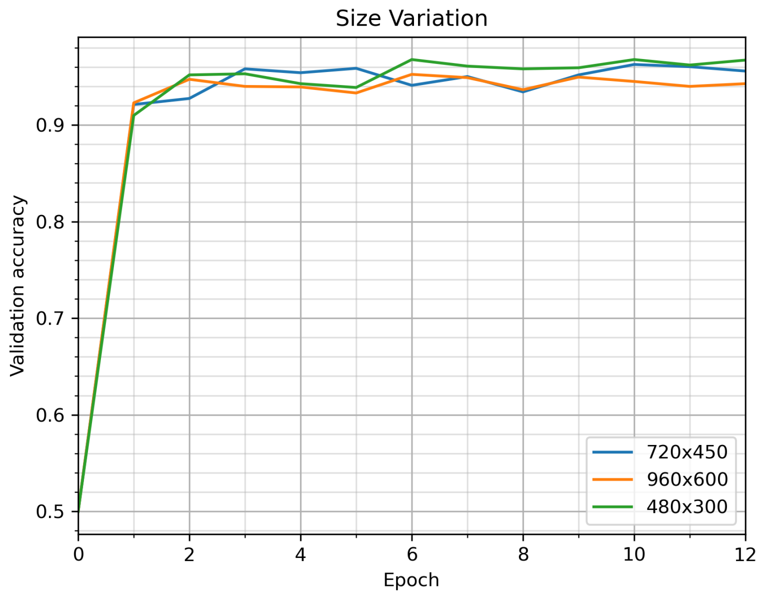


Figure 27: Validation accuracy during training for different image sizes. All other hyperparameters set to TPE optimum.

Table 6: Validation accuracy during size variation.

Image size	Validation accuracy
480x300	**0.9677**
720x450	0.9626
960x600	0.9524

5.3.6 Performance without Pretraining

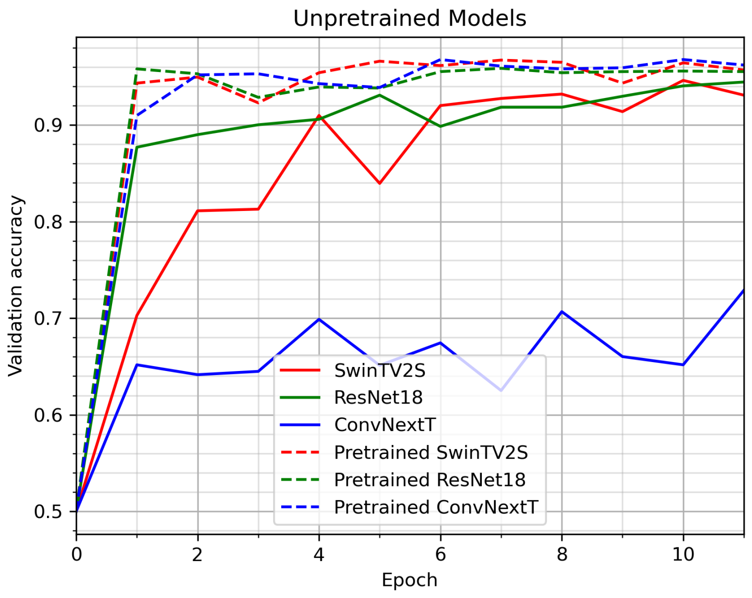


Figure 28: Validation accuracy during training for using pretrained weights vs randomly initialising them.

Table 7: Validation accuracy for non pretrained models.

Model architecture	Validation accuracy
ResNet18	0.9444
SwinTV2S	**0.9461**
ConvNeXtS	0.7289

5.3.7 Training Set Size Variation

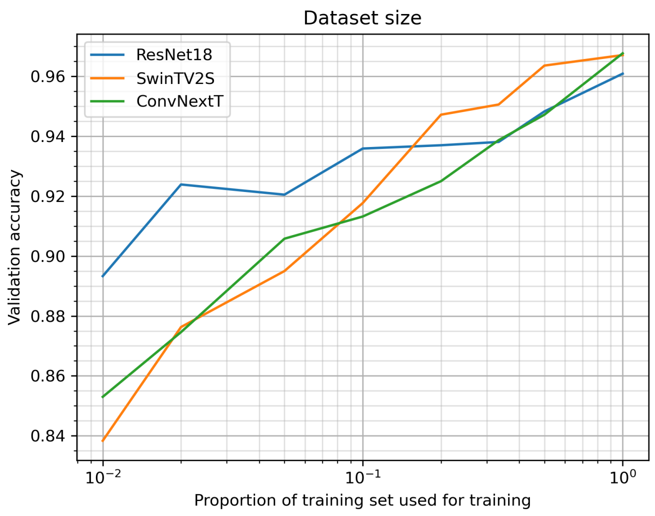


Figure 29: Validation accuracy for models trained on fractions between 1% and 100% of the training set.

5.4 Adversarial Fake data

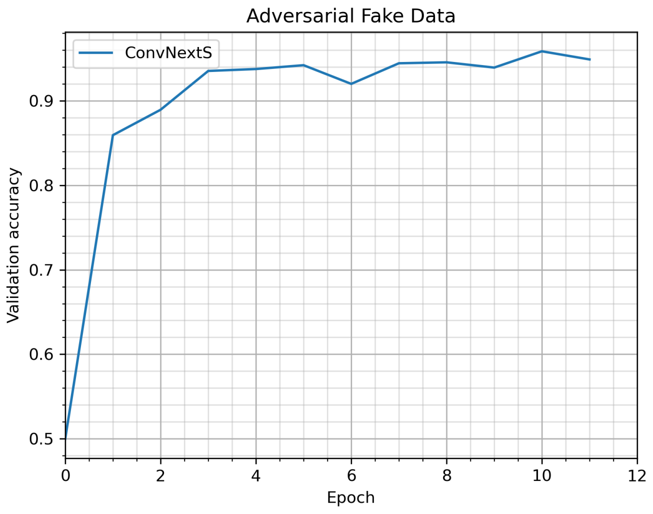


Figure 30: Validation accuracy for ConvNeXtS with adversarial fake data. It reached a maximum validation accuracy of 0.9585.

5.5 Time Consumption

5.5.1 GPU performance

Table 8: Time consumption of 3 different models, all trained on standard 480x300 images, using all augmentation types with unfrozen weights. Inference was done on a Nvidia RTX3090 and a machine equipped with a Intel Xeon Gold 6230N

Model	Data loading of validation set	Inference of validation set
ResNet18	7.9ms	1.0ms
SwinTV2S	7.4ms	7.6ms
ConvNeXtT	7.6ms	4.7ms

5.5.2 CPU Performance (Quantisation)

Table 9: CPU performance of ResNet18 for a single image

	32-bit (default)	8-bit (quantised)	Difference
Time Intel Xeon Gold 6230N	0.041s	0.0117s	3.5x speed up
Time Intel i3-6100 (microscope)	0.208s	0.0595s	3.5x speed up
Accuracy	0.961	0.959	0.002

5.6 Point Light Source Images

5.6.1 One PLS Image Networks

Table 10 shows the average validation accuracy for networks trained on one pls image in a specific angle.

Table 10: Average validation accuracy for networks trained on one PLS image at a time.

Angle	Average validation accuracy
16.2°	0.9693
33.6°	0.9503
54.6°	0.9561
55.4°	0.9524
56.8°	0.9587

5.6.2 Occlusion on a 30 PLS Image Network

During the first occlusion experiment, one point light source image was occluded at a time. This experiment generated almost the same result regardless of the occluded point light source. All tries generated an accuracy between 0.94 and 0.95 and the average validation accuracy for occlusion of light sources in a specific angle can be seen in Table 11.

Table 11: Average validation accuracy for occlusion of point light sources.

Angle	Average validation accuracy
16.2°	0.9427
33.6°	0.9476
54.6°	0.9460
55.4°	0.9484
56.8°	0.9471

For the second occlusion experiment, all point light source in a certain angle were occluded at the same time. The results can be seen in Table 12.

Table 12: Validation accuracy for occlusion of all light sources in a specific angle.

Angle	Validation accuracy
16.2°	0.9121
33.6°	0.9490
54.6°	0.9427
55.4°	0.9427
56.8°	0.9353

During the third and last occlusion experiment, all angles except one were occluded. The result can be seen in Table 13.

Table 13: Validation accuracy for occlusion of all light sources except the ones in a specific angle.

Angle	Validation accuracy
16.2°	0.8542
33.6°	0.8786
54.6°	0.8446
55.4°	0.8423
56.8°	0.8752

5.7 Visualisations

5.8 Maximum Activation

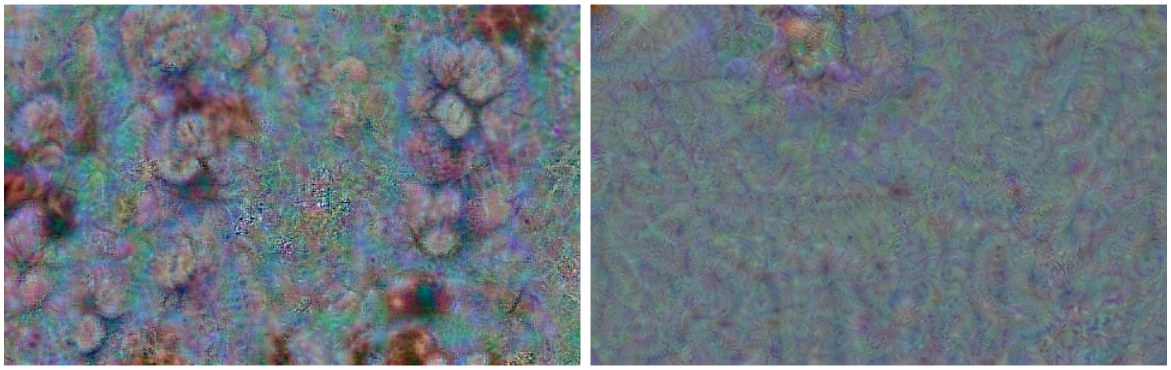

(a) Real particle (b) Fake particle

Figure 31: Maximum activations for the real and fake classes on our best ResNet18.

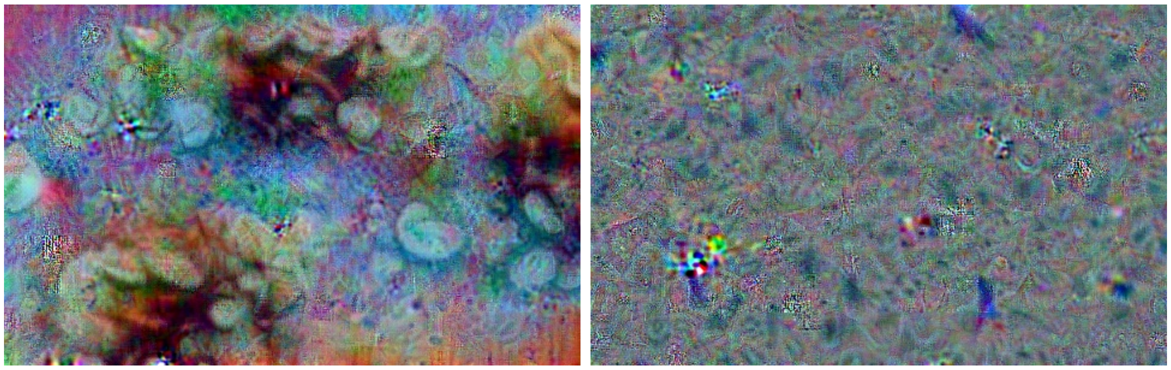
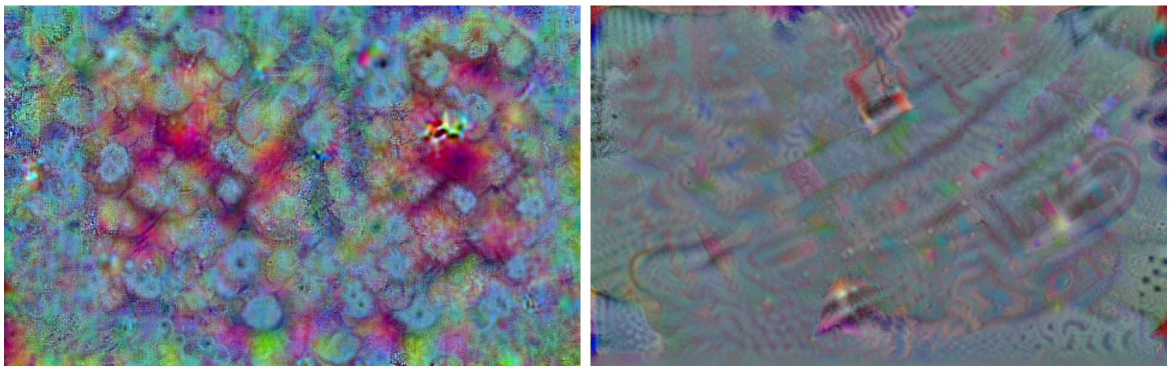

(a) Real particle (b) Fake particle

Figure 32: Maximum activations for the real and fake classes on our best SwinTV2S.



(a) Real particle (b) Fake particle

Figure 33: Maximum activations for the real and fake classes on our best ConvNeXtT.

5.8.1 UMAP

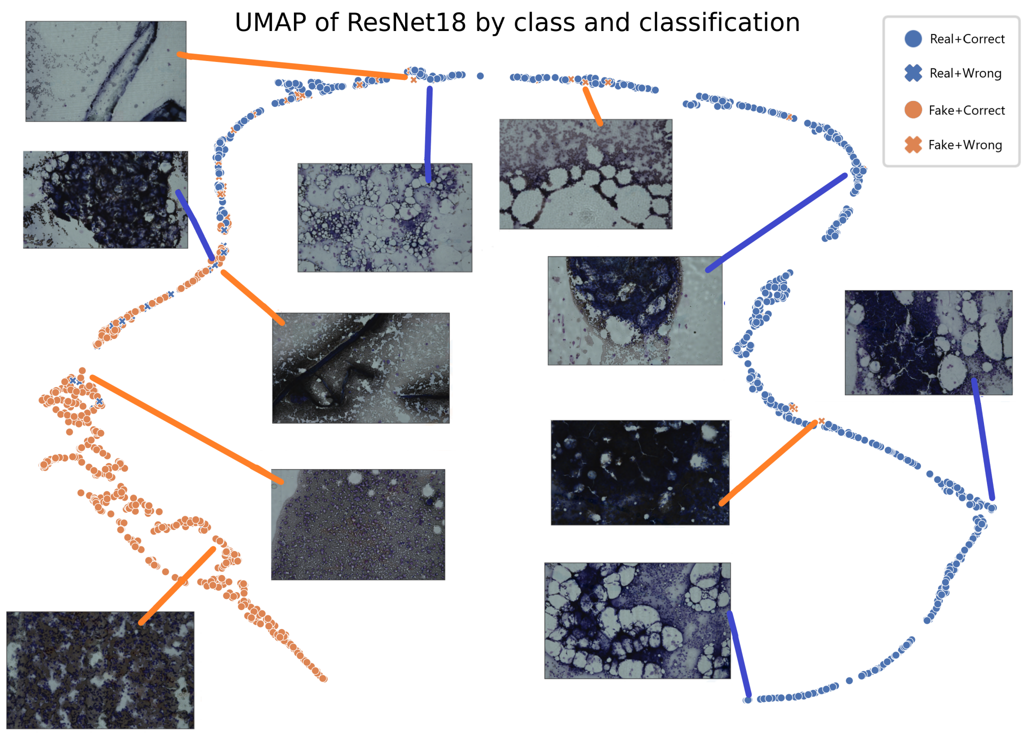


Figure 34: Figure a shows the UMAP for ResNet18 trained with image + geometric augmentation, unfrozen weights and an image size of 480x300. Blue implies the image was of a real particle while orange means fake. A dot means a successful classification while a cross means a misclassification.

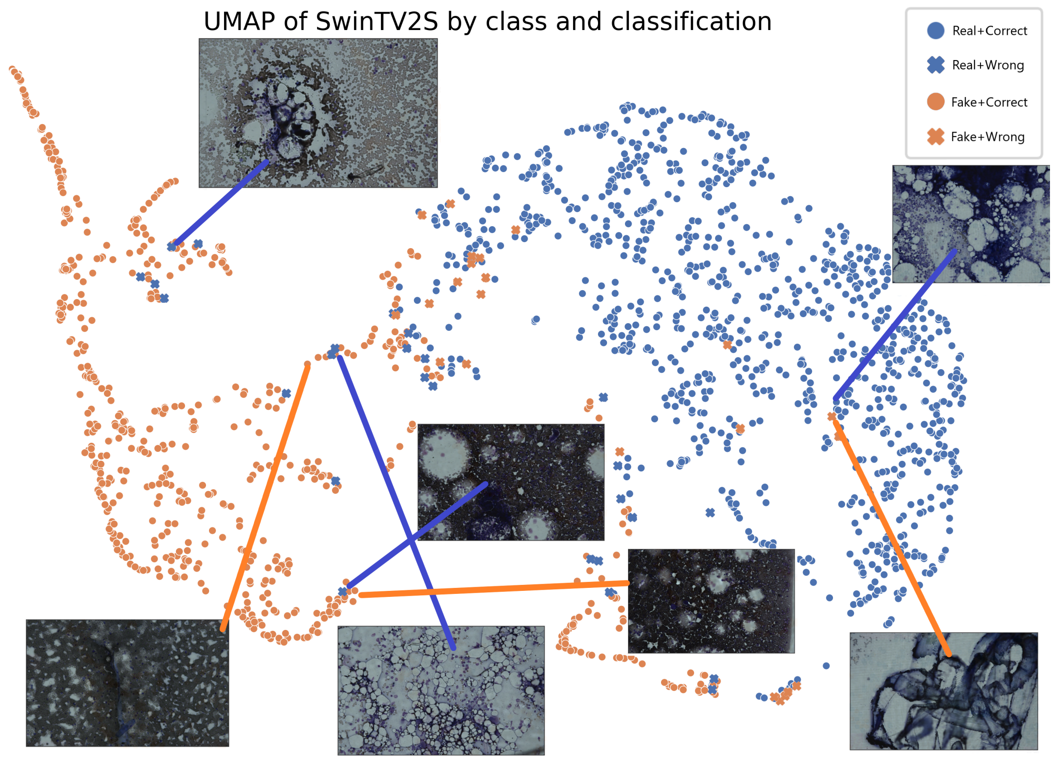


Figure 35: Figure a shows the UMAP for SwinTV2S trained with image + geometric augmentation, unfrozen weights and an image size of 480x300. Blue implies the image was of a real particle while orange means fake. A dot means a successful classification while a cross means a misclassification.

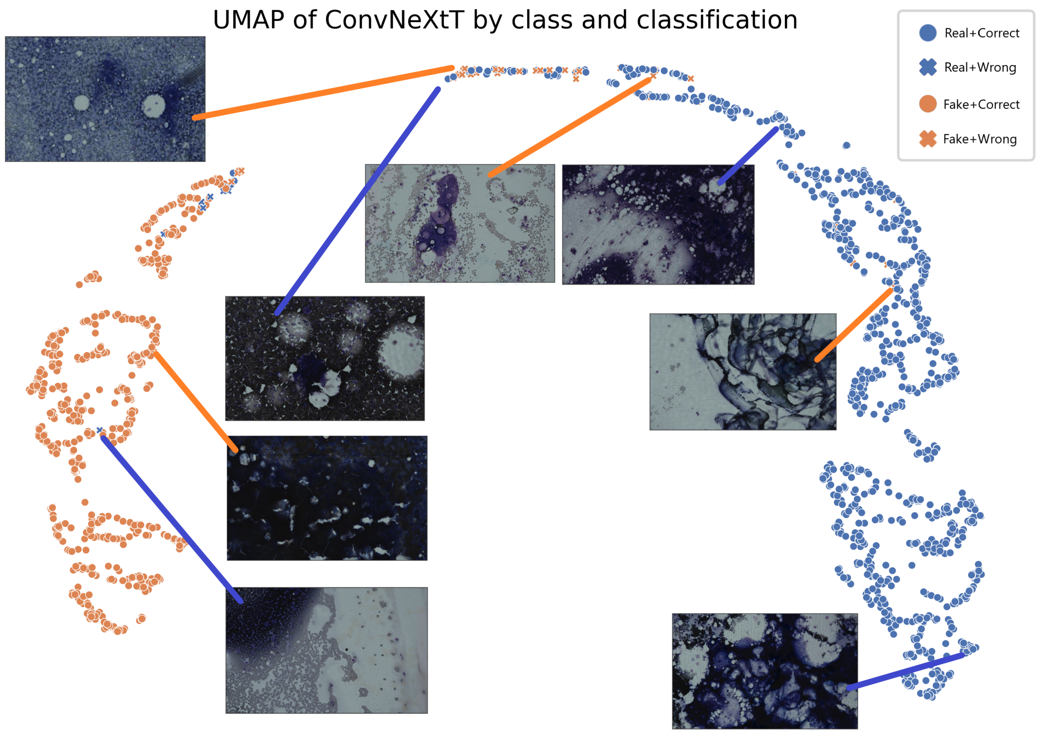
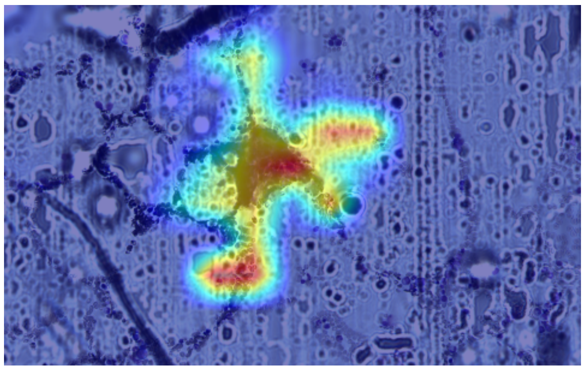


Figure 36: Figure a shows the UMAP for ConvNeXtT trained with image + geometric augmentation, unfrozen weights and an image size of 480x300. Blue implies the image was of a real particle while orange means fake. A dot means a successful classification while a cross means a misclassification.

5.8.2 Grad-CAM


(a) Portions of the image the network thinks are most real according to Grad-CAM.

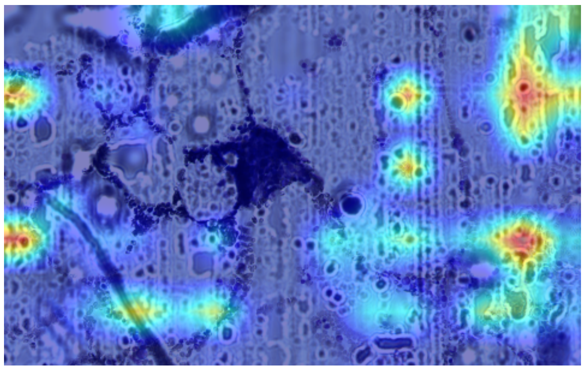
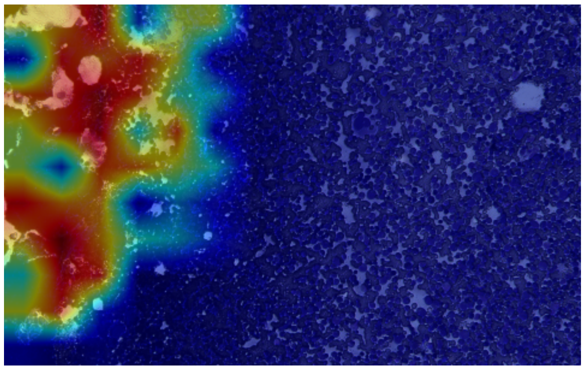

(b) Portions of the image the network thinks are most fake according to Grad-CAM.

Figure 37: Grad-CAM result where the model misclassified a fake particle as a real particle. ConvNeXtT trained with image + geometric augmentation, unfrozen weights and an image size of 480x300.


(a) Portions of the image the network thinks are most real according to Grad-CAM.

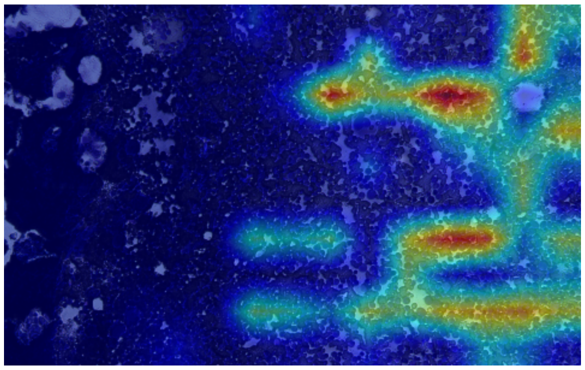
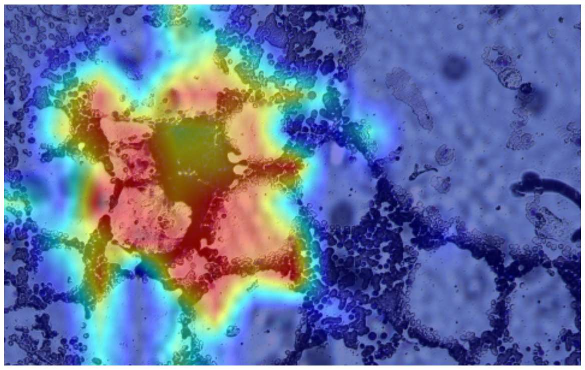

(b) Portions of the image the network thinks are most fake according to Grad-CAM.

Figure 38: Grad-CAM result where the model misclassified a real particle as a fake particle. ConvNeXtT trained with image + geometric augmentation, unfrozen weights and an image size of 480x300.


(a) Portions of the image the network thinks are most real according to Grad-CAM.

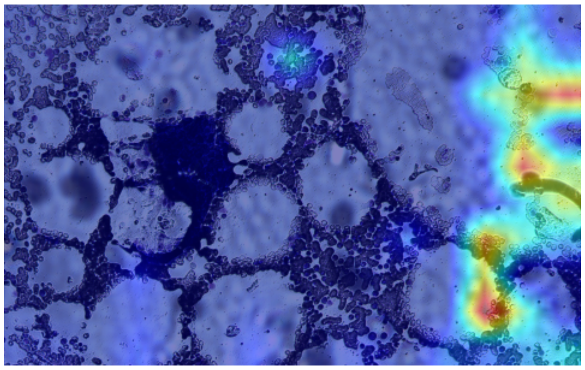
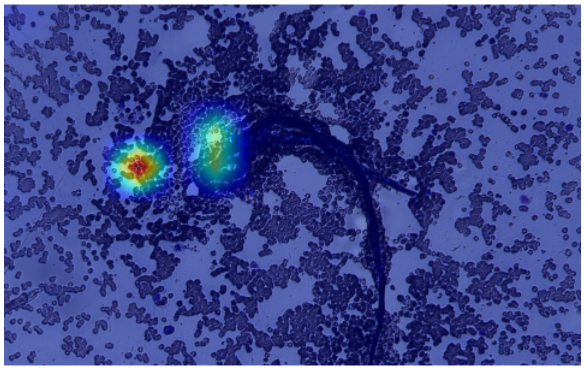

(b) Portions of the image the network thinks are most fake according to Grad-CAM.

Figure 39: Grad-CAM for a correctly classified real particle. ConvNeXtT trained with image + geometric augmentation, unfrozen weights and an image size of 480x300.


(a) Portions of the image the network thinks are most real according to Grad-CAM.

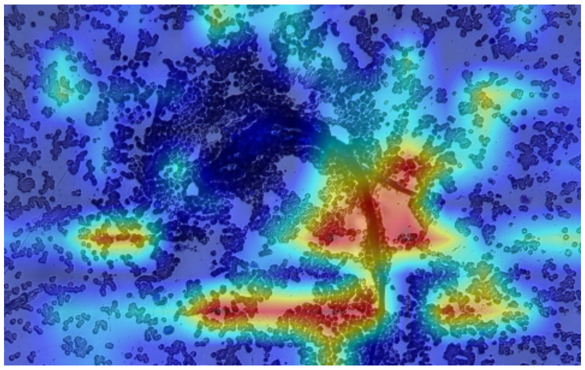

(b) Portions of the image the network thinks are most fake according to Grad-CAM.

Figure 40: Grad-CAM for a correctly classified fake particle. ConvNeXtT trained with image + geometric augmentation, unfrozen weights and an image size of 480x300.

5.9 Comparison of the Model to Non-Expert Humans

Out of 768 particles, the bone marrow expert classified 90 as unknown due to bad image quality or just too hard to determine. Out of the 678 particles classified as either fake or real, we classified 29 as unknown since it was too hard for us to determine. This implied that we classified 649 particles as either real or fake, and 614 of them were correctly classified. Our accuracy can be determined to $\frac{614}{649} = 0.9461$ if we ignore the ones that we classified as unknown. If the ones that were classified as unknown by us but not by the expert are counted as misclassified, our accuracy sinks to $\frac{614}{678} = 0.9056$.

5.10 Test set performance

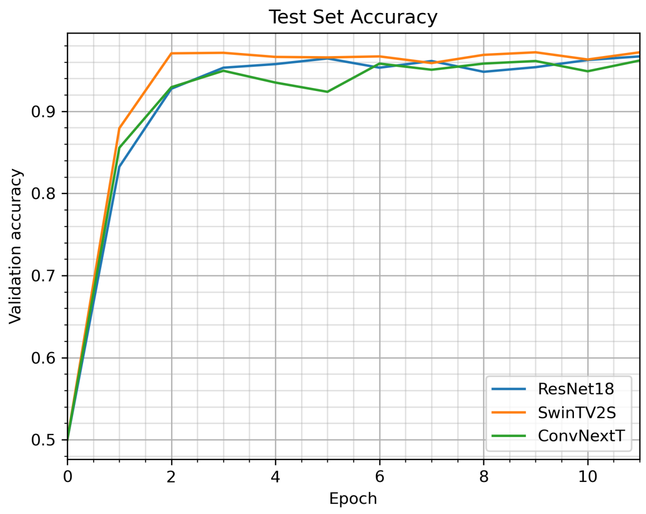


Figure 41: Performance of our models on the standard images in the test set per epoch.

Table 14: Performance of our models on the test set.

Model	Test Set Accuracy Standard Images
ResNet18 (32-bit)	0.9669
ResNet18 (8-bit)	0.9675
SwinTV2S	**0.9719**
ConvNeXtT	0.9619

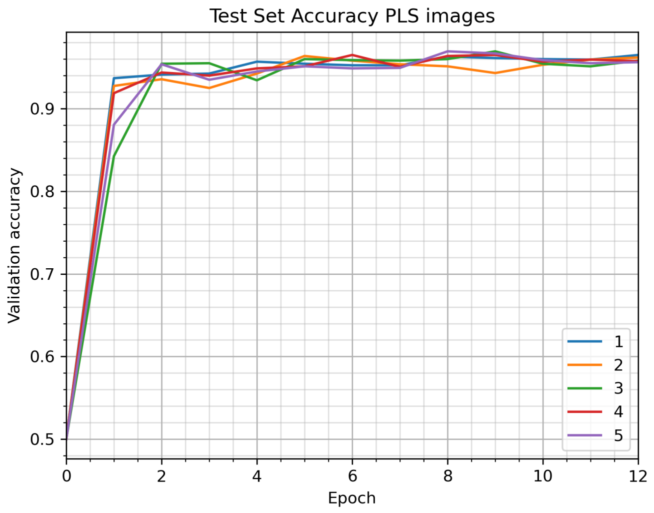


Figure 42: Performance of the single PLS image 16.2° models on the test set. The different numbers correspond to the different 16.2° point light sources.

Table 15: Performance of the single 16.2° PLS image models on the test set.

Model	Test Set Accuracy PLS images 16.2°
1	0.9650
2	0.9638
3	**0.9694**
4	0.9650
5	**0.9694**
Average	0.9665

6 Discussion

6.1 TPE Optimisation

Figure 22a and 22b show the results of the TPE hyperparameter optimisation. Figure 22a shows the accuracy for each attempt, though the validation accuracy seems fairly random between 0.90 and 0.97 with only a small, if any, improvement over time. It does not seem like TPE optimisation was a large improvement over random search in our case. In Figure 22b we can see the best model over time. The optimisation reaches close to its maximum accuracy very quickly, with almost no change after that.

6.2 Model Type Variation

Figure 23 and Table 2 show the effect of changing model type. It is quite difficult to tell the models apart, but ResNet performs slightly worse than Swin Transformer and ConvNeXt. ConvNeXtT and

ConvNeXtS performed the best in this experiment, but SwinTV2S sometimes performs better so this could just be down to chance. ConvNeXt appeared to sometimes fail completely (about 20 % of the time) with accuracy getting stuck at 60 % (which corresponds to always guessing real) while the other models never exhibited this behaviour. We think that the reason the models perform so similarly is that the underlying classification task was much easier than initially expected and that perfect performance is around 95 % as we believe the misclassification rate in our datasets to be around 5 %.

6.3 Image Types

As discussed earlier we tested several different types of images; standard images, circular dark-field images, a combination of both and a combination of 30 PLS images. Figure 24 and Table 3 shows the performance of these different image types. It appears that the standard image performs the best which contradicts the theory that the added information from other angles helps. The standard images do have an "unfair" advantage compared to the others however, the pre-training was done using 1000 image classes of normal photography which is probably more similar to standard microscopy than it is to dark-field microscopy, it could be the case that had we run this experiment on non-pretrained networks the standard image network would not have outperformed the circular dark field image network. It is also the case that feeding a model more information could degrade performance even if this information is high quality, since the model will now be required to sift through more information to find the important features. Therefore, when we feed the network 6 channel or 90 channel images, without giving the network more training data, the network will have to spread the same amount of optimisation power over a larger amount of information, potentially leading to worse performance. Additionally when we increase the number of channels we are throwing away the pretrained weights for the first layer meaning the network might not find as good first layer weights as in the 3 channel case. Furthermore we do not change the number of filters or the subsequent layers meaning that the network architectures which have been optimised for 3 channels will be ill equipped to suddenly handle 30 times more information.

6.4 Frozen Weights

Figure 25 and Table 4 show the effect of freezing all but the last layer of ConvNextS before fine-tuning. Freezing the weights is clearly a lot worse for performance and only seemed to speed up training slightly. In theory most features, at least in earlier layers, should be very similar for any image classification task as different CNNs typically learn low-level features, such as edges, corners, and colours, regardless of the specific task they are trained on. Freezing every layer but the last is obviously quite extreme, we expect that not only the early layer features but all of them will be relatively independent of the classification task. While the frozen weight model clearly performs worse at 93.5% compared to 96.8%, 93.5% is an astoundingly good score considering only the last layer could be changed. This supports the idea that learned features, at least when pretrained on a diverse dataset, are applicable to other classification tasks.

6.5 Augmentation Type Variation

Figure 26 and Table 5 show the effect of changing the type of augmentation used. As mentioned in the theory section augmentation theoretically increases the size of our dataset and could therefore be expected to improve performance unless the images are changed so much so that they no longer accurately reflect the validation distribution. In this case geometric augmentation does the best, better than image and geometric augmentation combined though the difference is small. The addition of CutMix seems to make training slower and slightly reduces performance. Perhaps CutMix is less beneficial when we only have two classes rather than 1000 or maybe the problem could be that quite a large fraction of a real particle image is “background” i.e. fake particle. It might be very disruptive if CutMix takes half of a real image that contains only background, combines it with half a fake image and labels it as 50 % real and 50 % fake. The difference between all types of augmentation is quite small, however, which might mean differences are random. This might indicate that we have more data than we “need” already and thus do not get much benefit from augmentation.

6.6 Image Size Variation

Figure 27 and Table 6 show the effect of changing the input image size. The original images were $1920x1200$ meaning the image size corresponds to how much we down sample the image before feeding it to the network. While the difference in performance between the different image sizes is small it appears that 480x300 size performs slightly better than the others. Since this is the smallest image size and therefore takes the least time and computational resources to use, it seems to be a bad idea to use larger images. This result is slightly surprising as this represents quite a large down sampling, every pixel we feed in is actually the average of 16. As a human it is much harder to classify these down sampled images but clearly the features the model is after are not affected by the noise, whereas the reduced noise from down sampling improves performance.

6.7 Training Set Size Variation

Figure 29 shows the effect of varying the training set size. Note that the x-axis is in log scale and goes from 1/100 of the training set i.e., 55 images to the entire training set. Here it is clear that ResNet18 does far better than the other models when they are not given much data. All three models seem to produce fairly straight performance vs data lines but with different starting points and different slopes. It seems like SwinTV2S has the steepest slope and overall does better than ConvNextT, only performing a tiny bit worse on the entire dataset. These performance curves support the idea that smaller networks will perform better than bigger ones when not given much data, while larger networks will shine when given large amounts of training data. The curves do not indicate that they are about to level off so we could probably expect some additional improvement if we had even more data. Note that the images from the training set to use were chosen randomly meaning they reflect the whole training set quite well. We hypothesize that scanning only one slide and capturing 55 images from it would likely not yield 89% accuracy on the validation set using ResNet18..

6.8 Non Pretrained Training

Figure 28 and Table 7 shows the performance of randomly initialising the weights of our models, rather than using the pretrained weights. Here we see that SwinTV2S and ResNet18 take much longer to train but that they eventually perform only a few percentage points (2.1 % and 1.65% respectively) worse than their pretrained counterparts. ConvNeXtT on the other hand seems to barely improve during training, reaching only 75% accuracy, a difference of 21.9%. The poor performance of ConvNeXt could in some way be related to the fact that it simply fails to train 10-20% of the time but when it fails it stays at exactly 0.60692 accuracy, while here it at least improves a tiny amount. Overall it seems like using a pretrained model definitely improves performance, especially early in the training. The effect would probably be larger when trained with smaller training sets and would probably eventually disappear given enough data. These results combined with the decent performance of the network when freezing all but the last layer illustrate that many of the features learned from a general dataset are useful also for this classification task.

6.9 Adversarial Fake data

The results of adding the adversarial fake data to the training set can be seen in Figure 30. Surprisingly training on the adversarial images did not improve performance, but instead degraded it from 96.77% to 95.85%. We originally did this because we noticed a fake particle which we believed was small paper fibers kept getting classified as real, however even after adding images of paper fiber to the training set the network still misclassfies this as real.

6.10 Time Consumption and Quantisation

In Table 8 we see the average time consumption for our models when doing inference on an image using a batch size of 16. The hardware used was a Nvidia RTX 3090 GPU and a Intel Xeon Gold 6230N. The data loading time was about 7.5ms for all models, which makes sense since they all need to load the same images. For ResNet18 loading took much longer than the actual inference. For SwinTV2S and ConvNeXtT data loading and inference times were comparable. While the inference was 7x faster for ResNet18 than SwinTV2S, including the data loading time it is only 60 % faster. We think the data loading can might optimised significantly as our data loading code is probably quite inefficient. We also believe that the data loading time will depend mostly on the CPU used while the inference will depend mostly on the GPU. We are using quite a powerful GPU so when both data loading and inference is done on a less powerful microscope CPU, inference might take a much larger portion of the total time. Since ResNet18 performs only 0.68 % worse than ConvNeXtT while inference is 4.5x faster it seems to make sense to use ResNet18 as a starting point when trying to make a computationally cheap model to be run on microscope hardware.

Table 9 shows the average inference time and accuracy of a 32-bit and a 8-bit version of ResNet18 on both the Intel Xeon Gold 6230N from before and an Intel i3-6100 which is the CPU our microscope uses. The batch size was now set to 1 as when capturing images the microscope will be capturing one at a time. Quantisation made the model run about 3.5x faster on both CPUs which is less than the theoretical 4x but obviously 4x is just an upper-bound, in practice there will be other processes which also take time which will not be sped up by quantistaion. Perhaps more importantly is that the accuracy barely degrades at all, only a 0.002 % decrease. With an inference time of 0.0595s and and accuracy of 0.959 % we believe it is very feasible to implement the 8-bit version of ResNet18 in a real world BMA analysis pipeline.

6.10.1 Maximum Activation

In figures 31a, 32a and 33a we can see clear particle like structures, dark red/purple/black regions that have similar circular features to the real particles. There is still a large amount of noise, despite all the regularisation techniques used, with high frequency patterns and vibrant colours overlayed being found across the entire image. As mentioned in the theory section (see section 3.19.3), high levels of noise seem to be a common feature of gradient ascent/maximum activation and the main difficulty in this technique is how to minimise it. Many of the arguments for why these images should be expected to be very noisy was based on CNNs, but the maximum activation for Swin Transformer seems to have a large amount of noise as well. The arguments could probably be extended to also apply to transformer networks as attention can in some sense be seen as a generalisation of CNNs with different inductive biases.

Comparing 31a, 32a and 33a qualitatively, they do look different from each other, but it is hard to pinpoint exactly what differs. Swin Transformer seems to have slightly bigger and more intense particles than the others, while in ResNet the particles seem almost translucent. Both Swin Transformer, but particularly ConvNeXt, seem to have much more vibrant colours than ResNet. These differences could be down to how the hyperparameters of the maximum activation (such as learning rate, L2 and TV coefficients or geometric jitter magnitude) interact with the different networks rather than inherent features of the networks. The method seemed to be sensitive to parameter changes with bad parameters mainly leading to images of pure noise but also changes in the vibrancy and frequencies in the images. The same parameters were used to generate the three images so it could simply be the case that the optimal parameters for each network are slightly different.

Figures 31b, 32b and 33b by contrast appear to be largely noise, although perhaps lower frequency noise than the "background" of figures 31a, 32a and 33a. The fake particle class is in some sense much more diverse than the real particle class and a large portion of it consists of "background" i.e., a single colour with smaller cells but not as much large scale structure. A single colour is obviously low frequency and the large variation image type in the fake class might mean that the network is detecting the lack of real particle more than anything else when classifying an image as fake. Again, the images from the different networks differ but it is difficult to say how. The noise in the Swin Transformer case appears to be higher frequency than the others and ConvNeXt appears to have particularly low frequency and more consistency structure in the pattern it detects.

6.11 UMAP

Figure 34 shows the UMAP for ResNet18, colour-coded by class and ticker-style-coded by successful classification. It appears that the images are all very close to being on a curve with a large empty region in the centre of the image. There seems to be a large region with almost exclusively fakes, a large region with almost only real images and then a region in between where they are mixed.

Figure 35 instead shows the UMAP for SwinTV2S, coded in the same way. This time there is no empty region, but rather the two classes form two partially intersecting regions. There appears to be one set of fake images which is better seperated from the rest and one set of fakes which intersects with the real images more. There is not as much overlap as in the ResNet18 case which is reflected in the improved performance of this network.

Figure 36 by contrast shows the UMAP for ConvNeXtT colour-coded in the same ways. Here we have a similar curve-like appearance to ResNet18 but the curve is wider. Again we have one region which is almost exclusively fake which is isolated from the rest, and one large region of reals which has some fake images at one end.

6.12 Grad-CAM

Figure 37 shows the Grad-CAM of an example where the network has misclassified a fake particle as a real particle. We can see that the model focuses on the dark-particle looking region in the centre which makes sense as this the the most particle like feature in the image. The model thinks that a few seemingly random background portions of the image are important for a would be fake classification, but that this is not enough to make that classification.

Figure 38 shows the Grad-CAM of an example where the network has incorrectly classified a real particle as a fake. We can see that the network looks at the particle when highlighting areas corresponding to real particles and that it looks at background regions when highlighting fake areas. The real area appears to be both more intense and larger than the fake area, so it is surprising that the network classifies this image as fake rather than real. Perhaps this has something to do with the flaws of Grad-CAM mentioned earlier.

Figure 39 instead shows the Grad-CAM of a correctly classified particle. The model focuses on the particle when looking for real regions which is good, and thinks the region to the right, far away from the particle is the most fake which also makes sense.

Figure 40 by contrast shows the Grad-CAM of a correctly classified fake particle. Here the network only thinks two small regions are real while large portions of the image are fake. The regions it thinks are real vs fake seem very strange to us and would definitely not be how a human would decide if a particle was real or not.

6.13 PLS Images

Table 10 shows the average validation accuracy for ConvNeXtT trained on 1 PLS image type at each angle. Note that there were 5 PLS image types for each angle apart from 56.8° which had 10, so these numbers are the average of 5 or 10 training runs. We see that 16.2°, i.e. bright-field PLS performs the best and 33.6° (the first dark-field PLS) performs the worst. 16.2° surprisingly actually performs better than the standard image. It could be the case that the “shadows” caused by only illuminating a sample from 1 angle rather than even illumination is actually helping the network detect real particles. It could for instance be the case that real particles, (which they seem to be based on our experience creating our auto-focus mechanism) are taller than fake particles, this could mean that they cast a longer shadow than fake particles. Comparing 16.2° images to standard ones, such as in Figure 20 we see that the 16.2° images are much brighter and have more visual artifacts. It seems unlikely to be the case that the artifacts actually help classification, but the large brightness discrepancy leads to an alternate hypothesis that it is actually the increased brightness which helps classification. When initially choosing the brightness of the standard images we avoided burning out pixels, i.e., too many pixels having value 255, thereby losing information. However, since particles are generally dark it might have made sense to sacrifice information about light regions in favour of more contrast in dark regions. It might be the case that increasing the brightness of the standard images would improve accuracy and beat the 16.2° images. Unfortunately doing this would mean recapturing all images which originally took 6 weeks, so we did not have time to test this.

Table 11 shows the average effect of occluding one image, i.e. replacing 3 out of the 90 channels fed into the network with zeros for each angle. We see that the performance degrades the most for 16.2°, indicating that images from this angle are the most important ones for performance.

Table 12 instead show the effect of occluding all images from one angle. Again, we see that the accuracy drops the most for 16.2° which reinforces the notion that these are the images the 30 stacked image network cares the most about. We can also notice that the performance drops the second most for 56.8°, this could be because this angle has 10 images rather than 5 so we are removing twice as many images when we occlude this angle.

Table 13 shows the effect of occluding all but one angle. Very surprisingly the 33.6° angle performs the best here, better than 16.2°. In Table 12 we saw that removing 33.6° had the least effect on the network, so it is quite strange that this result indicates that 33.6° has the most effect.

6.14 Test Set

In Figure 41 and Table 14 we see the performance of four models on the test set. Here SwinTV2S outperforms ConvNeXtT by 0.9719 to 0.9619. Interestingly the 8-bit quantised version of ResNet18 does slightly better than the 32-bit version, although the difference is small and probably due to randomness. All models do slightly better on the test set than the validation set which could either be because the test set is somehow marginally easier to classify than the validation set or because the models have now been trained on about 25% more data (i.e., the validation set).

Since the 16.2° image networks did better than the standard image networks on the validation set we decided to also test these on the test set, this time with SwinTV2S (since most of the time it does better than ConvNeXtT). The results of this can be seen in Figure 42 and Table 15. In contrast to on the validation set however, the 16.2° single PLS image networks do worse than the standard image networks, perhaps it was only chance that they did better before.

7 Conclusion

From our experiments, we can conclude that the hyperparameter that had biggest impact on performance was dataset size where the accuracy increased logarithmically with the size of the dataset. The images with standard illumination also performed better than the PLS images, probably due to the pretrained networks being biased towards standard images. However, our best model, which was a Swin Transformer V2S, achieved 97.19% accuracy on the test set. To make a model feasible to use on a real microscope hardware, our smallest model, ResNet18, was quantised from 32 bits to 8 bits. This quantised model achieved 96.75% accuracy while only taking 59 ms per image for inference on the microscope.

References

1. Wang, C.-W. *et al.* Deep learning for bone marrow cell detection and classification on whole-slide images. *Medical Image Analysis* **75,** 102270. ISSN: 1361-8415. `https://www.sciencedirect.com/science/article/pii/S1361841521003157` (2022).
2. Compston, J. Bone marrow and bone: a functional unit. *The Journal of endocrinology* **173,** 387–394 (2002).
3. Clinic, C. *Bone Marrow Biopsy* July 2022. `https://my.clevelandclinic.org/health/diagnostics/17735-bone-marrow-biopsy`.
4. Percival, M.-E., Lai, C., Estey, E. & Hourigan, C. S. Bone marrow evaluation for diagnosis and monitoring of acute myeloid leukemia. *Blood Reviews* **31,** 185–192. ISSN: 0268-960X. `https://www.sciencedirect.com/science/article/pii/S0268960X16300509` (2017).
5. Bain, B. J. Bone marrow aspiration. *Journal of Clinical Pathology* **54,** 657–663. ISSN: 0021-9746. eprint: `https://jcp.bmj.com/content/54/9/657.full.pdf`. `https://jcp.bmj.com/content/54/9/657` (2001).
6. Bharuthram, N. & Feldman, C. The diagnostic utility of bone marrow examination in an infectious disease ward. *Southern African Journal of HIV Medicine* **20,** 7. ISSN: 2078-6751. `https://sajhivmed.org.za/index.php/hivmed/article/view/974` (2019).
7. Howell, S. J. *et al.* The value of bone marrow examination in the staging of Hodgkin's lymphoma: a review of 955 cases seen in a regional cancer centre. *British Journal of Haematology* **119,** 408–411. `https://onlinelibrary.wiley.com/doi/abs/10.1046/j.1365-2141.2002.03842.x` (2002).
8. ARBER, D. A. in *Modern Surgical Pathology (Second Edition)* (eds Weidner, N., Cote, R. J., Suster, S. & Weiss, L. M.) Second Edition, 1536–1593 (W.B. Saunders, Philadelphia, 2009). ISBN: 978-1-4160-3966-2. `https://www.sciencedirect.com/science/article/pii/B9781416039662000436`.
9. Gao, P. F., Lei, G. & Huang, C. Z. Dark-Field Microscopy: Recent Advances in Accurate Analysis and Emerging Applications. *Analytical Chemistry* **93,** 4707–4726. `https://doi.org/10.1021/acs.analchem.0c04390` (2021).
10. Yeh, L.-H. *et al.* Experimental robustness of Fourier ptychography phase retrieval algorithms. *Optics Express* **23,** 33214. `https://doi.org/10.1364%5C%2Foe.23.033214` (Dec. 2015).
11. Yegnanarayana, B. *Artificial neural networks* (PHI Learning Pvt. Ltd., 2009).
12. Cburnett, W. Sept. 2006. `https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg`.
13. Thoma, M. July 2014. `https://commons.wikimedia.org/wiki/File:Perceptron-unit.svg`.
14. Ruder, S. *An overview of gradient descent optimization algorithms* 2017. arXiv: `1609.04747 [cs.LG]`.
15. Bottou, L. *et al.* Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes* **91,** 12 (1991).
16. Kingma, D. P. & Ba, J. *Adam: A Method for Stochastic Optimization* 2017. arXiv: `1412.6980 [cs.LG]`.
17. Chen, J., Wolfe, C. & Kyrillidis, A. *REX: Revisiting Budgeted Training with an Improved Schedule* 2021. arXiv: `2107.04197 [cs.LG]`.
18. Saxsena, S. *Binary Cross Entropy/Log Loss for Binary Classification* 2021.
19. Nwankpa, C., Ijomah, W., Gachagan, A. & Marshall, S. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378* (2018).
20. He, K., Zhang, X., Ren, S. & Sun, J. *Deep Residual Learning for Image Recognition* 2015. arXiv: `1512.03385 [cs.CV]`.

21. Hendrycks, D. & Gimpel, K. *Gaussian Error Linear Units (GELUs)* 2020. arXiv: `1606.08415 [cs.LG]`.
22. Liu, Z. *et al. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows* 2021. arXiv: `2103.14030 [cs.CV]`.
23. Liu, Z. *et al. A ConvNet for the 2020s* 2022. arXiv: `2201.03545 [cs.CV]`.
24. O'Shea, K. & Nash, R. *An Introduction to Convolutional Neural Networks* 2015. arXiv: `1511.08458 [cs.NE]`.
25. Plotke, M. Jan. 2013. `https://commons.wikimedia.org/wiki/Category:Convolution#/media/File:2D_Convolution_Animation.gif`.
26. Szegedy, C., Ioffe, S., Vanhoucke, V. & Alemi, A. *Inception-v4, inception-resnet and the impact of residual connections on learning* in *Proceedings of the AAAI conference on artificial intelligence* **31** (2017).
27. Ioffe, S. & Szegedy, C. *Batch normalization: Accelerating deep network training by reducing internal covariate shift* in *International conference on machine learning* (2015), 448–456.
28. Ba, J. L., Kiros, J. R. & Hinton, G. E. *Layer Normalization* 2016. arXiv: `1607.06450 [stat.ML]`.
29. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* **15,** 1929–1958 (2014).
30. Kukačka, J., Golkov, V. & Cremers, D. *Regularization for Deep Learning: A Taxonomy* 2017. arXiv: `1710.10686 [cs.LG]`.
31. Wiedemann, G., Yimam, S. M. & Biemann, C. *UHH-LT at SemEval-2020 Task 12: Fine-Tuning of Pre-Trained Transformer Networks for Offensive Language Detection* 2020. arXiv: `2004.11493 [cs.CL]`.
32. Medsker, L. R. & Jain, L. Recurrent neural networks. *Design and Applications* **5,** 64–67 (2001).
33. Staudemeyer, R. C. & Morris, E. R. Understanding LSTM–a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586* (2019).
34. Vaswani, A. *et al. Attention Is All You Need* 2017. arXiv: `1706.03762 [cs.CL]`.
35. Radford, A. *et al.* Language models are unsupervised multitask learners. *OpenAI blog* **1,** 9 (2019).
36. Liu, Z. *et al. Swin Transformer V2: Scaling Up Capacity and Resolution* 2022. arXiv: `2111.09883 [cs.CV]`.
37. PyTorch. *Quatization* 2023. `https://pytorch.org/docs/stable/quantization.html`.
38. Mikołajczyk, A. & Grochowski, M. *Data augmentation for improving deep learning in image classification problem* in *2018 international interdisciplinary PhD workshop (IIPhDW)* (2018), 117–122.
39. Yun, S. *et al. Cutmix: Regularization strategy to train strong classifiers with localizable features* in *Proceedings of the IEEE/CVF international conference on computer vision* (2019), 6023–6032.
40. Du, M., Liu, N. & Hu, X. Techniques for interpretable machine learning. *Communications of the ACM* **63,** 68–77 (2019).
41. Selvaraju, R. R. *et al.* Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *International Journal of Computer Vision* **128,** 336–359. `https://doi.org/10.1007%5C%2Fs11263-019-01228-7` (Oct. 2019).
42. Kindermans, P.-J. *et al.* The (un) reliability of saliency methods. *Explainable AI: Interpreting, explaining and visualizing deep learning,* 267–280 (2019).
43. Olah, C. *et al.* Zoom in: An introduction to circuits. *Distill* **5,** e00024–001 (2020).
44. McInnes, L., Healy, J. & Melville, J. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction* 2020. arXiv: `1802.03426 [stat.ML]`.

45. Bergstra, J., Yamins, D. & Cox, D. *Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures* in *International conference on machine learning* (2013), 115–123.