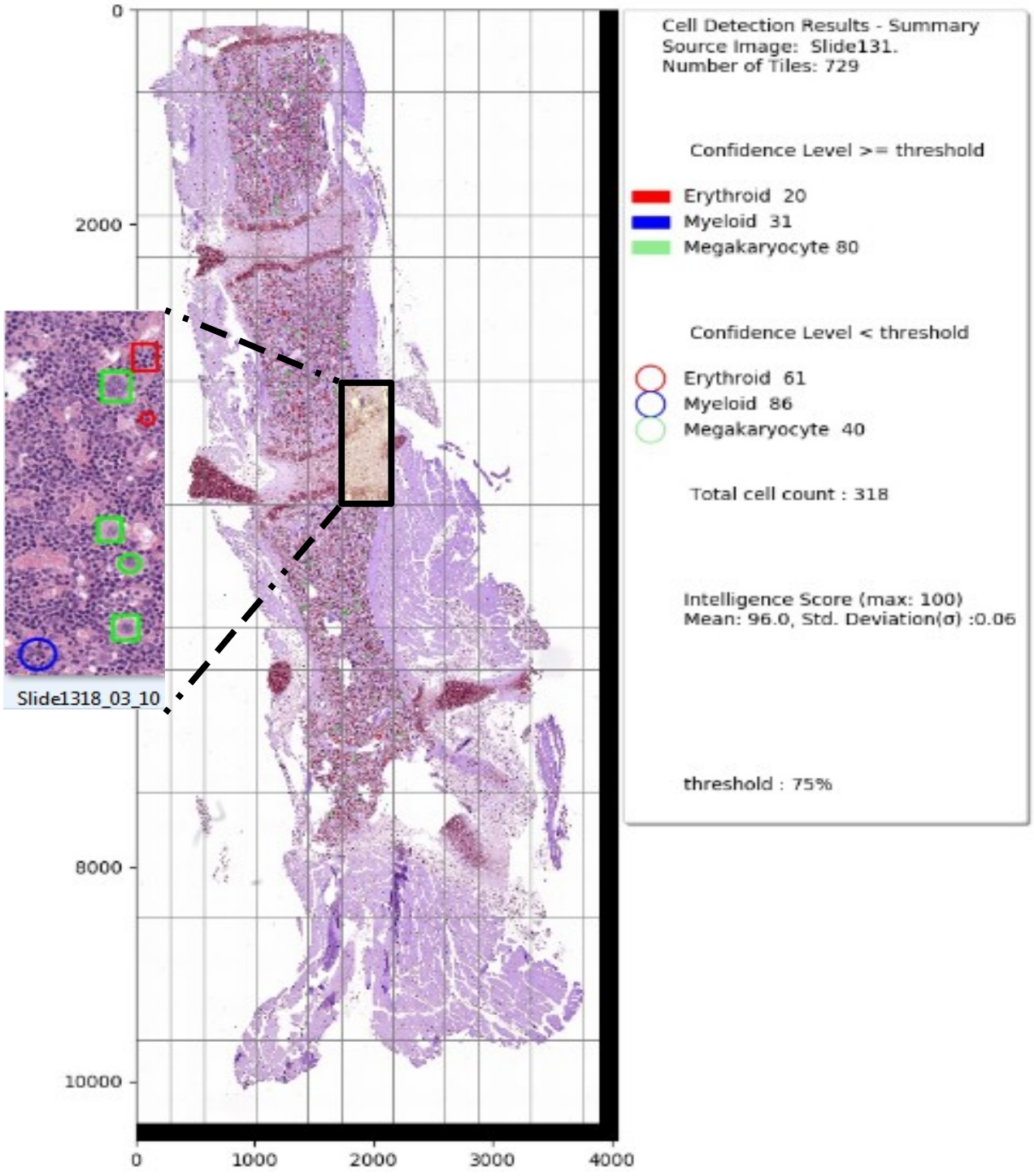
Automatic detection and quantification of Bone Marrow cells



A simple approach leveraging a trained mature Neural Network

a.k.a

Transfer Learning

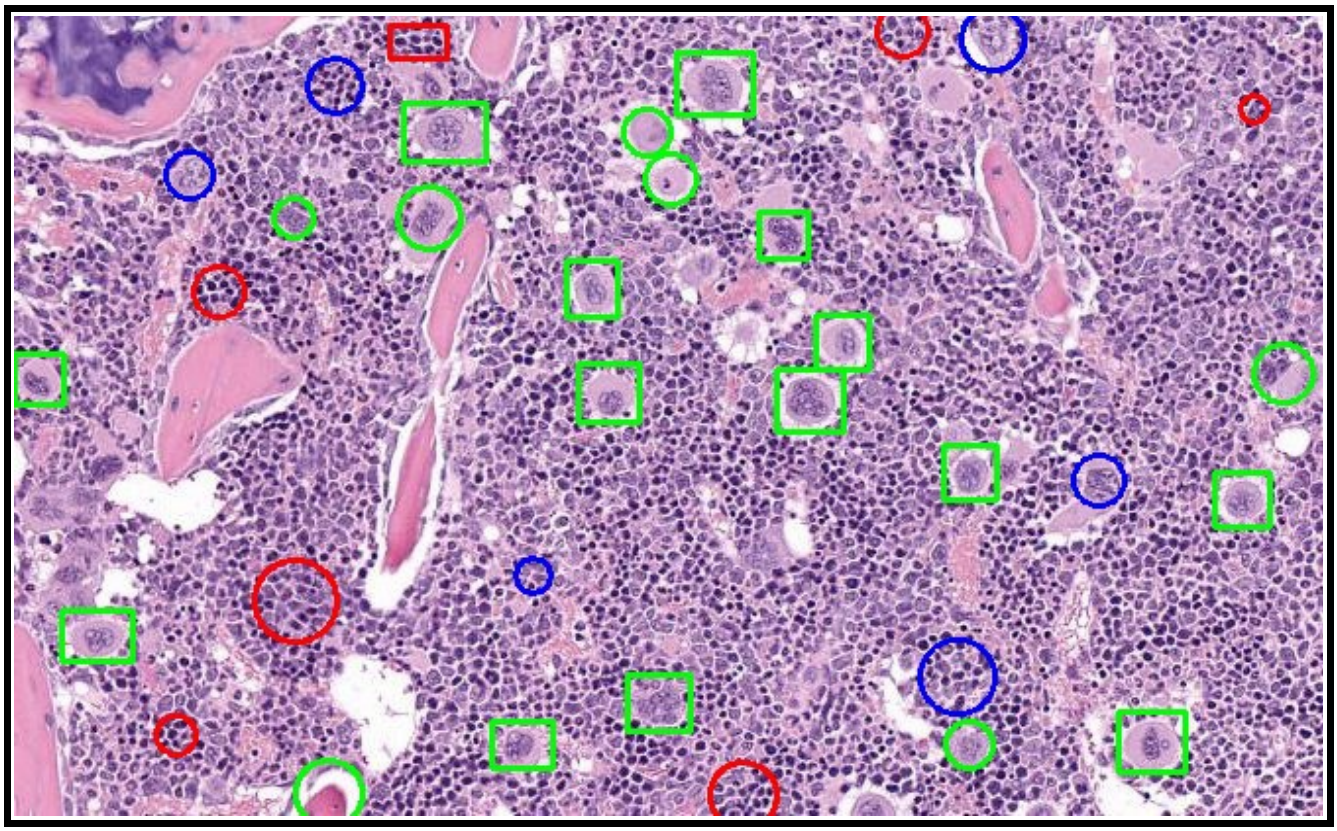


Table of Contents

Overview of Solution Architecture and Results....4
- Training a pre-trained reputed Neural Network (NN) to detect blood stem cells....4
- Detection & Counting of blood stem cells in images....5
- 20 Slide Summary : Totals, Intelligent Scores & hard numbers....6
- Runtime details....7
- Why stop at 0.06 average error value ?....7
- Solution Output Organization....8

Identify hematopoietic cell lineages from bone marrow images....9
- Challenge summary....9
- Challenge requirements....9
- My approach....9

Solution facets....10
Summary of Achievements....11
- Annotated cell types detected and labeled with Intelligent Score with ~ 96% accuracy as shown for one training slide below:....11
- 20 Slide Summary : Totals, Intelligent Scores & hard numbers....12
- Summary of details per Slide - Tiles in Slide Worksheets....13
- Summary of Tiles (labeled and annotated) per Slide - Visualization....13
- Summary for each slide – Hard Numbers plus Visualization....14
- Intelligent score (IS) with a must include Standard Deviation....15
- Completely open-source based through and throughout....15
- Super flexible – parameter driven....15

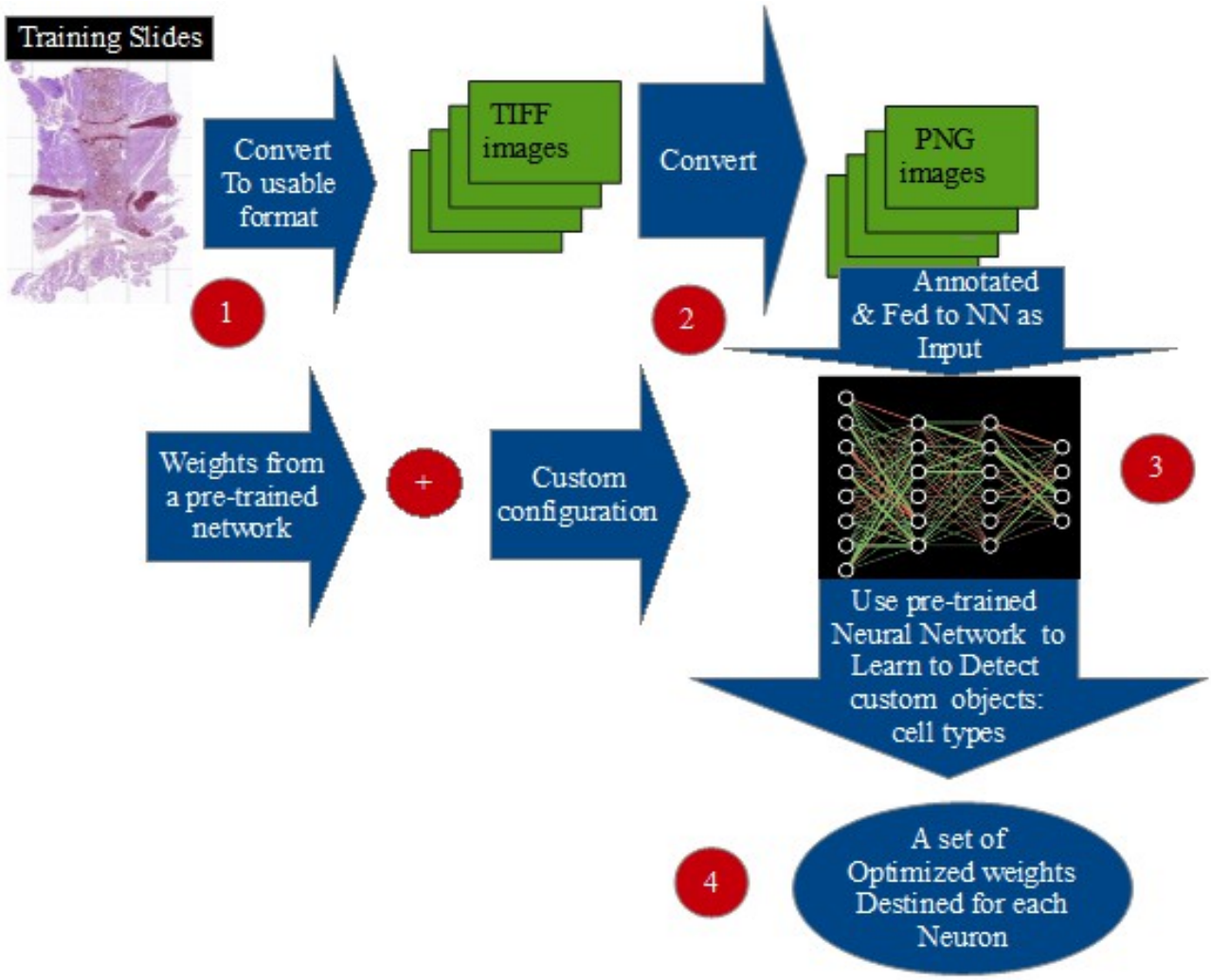
Platform Choice....16
- Poor first choice – Windows 7 Laptop without GPU....16
- Good free cloud-option – Google Colab with punishing restrictions....16
- Conclusion – best to get a dedicated CPU/GPU/RAM/Storage combo....16

AI/ML for designing a Neural Network to detect blood cells....17
- Data Preparation....17
  - Choice of bio-formats for image conversion....17
  - Tile size determination....17
- Training....18
  - Annotate and label training images....18
  - Transfer Learning/Training for custom Object Detection....19
  - Object detection and Labeling cell types....20
  - Training algorithm choice – YOLO /darknet....21
  - Intelligent Score -explanation....22

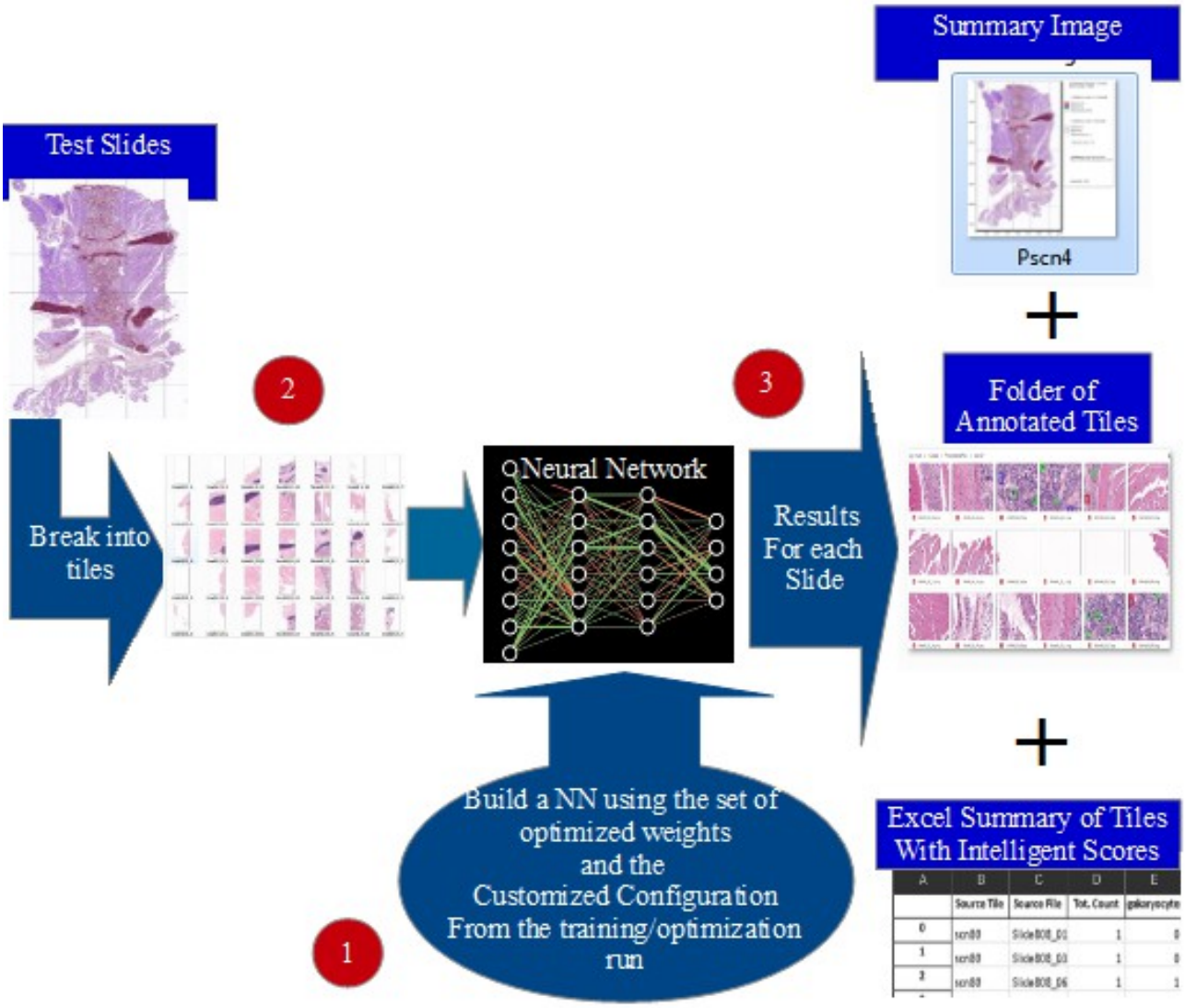
Key observations (O), ideas (I) and errors (E)....24
Implementation Details....26
Further thoughts....29

Overview of Solution Architecture and Results

Training a pre-trained reputed Neural Network (NN) to detect blood stem cells



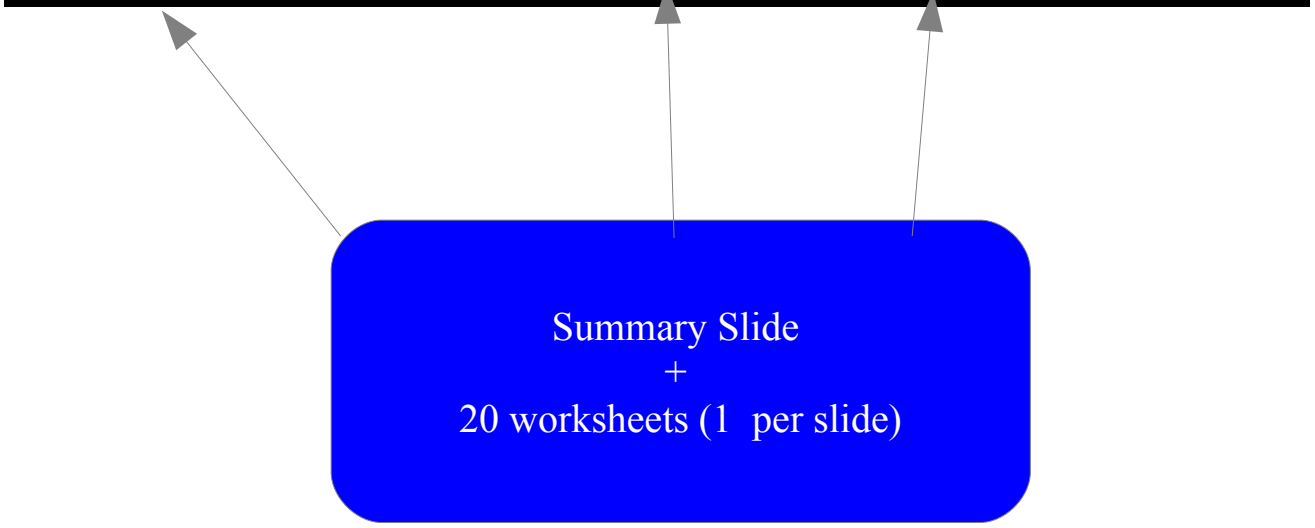
Detection & Counting of blood stem cells in images



20 Slide Summary : Totals, Intelligent Scores & hard numbers

	Source File	Num Tiles	Total Cells	Intelligent Score	Stdev	Time (seconds)
0	Slide115X.scn	361	107	96.0	0.06	42.17
1	Slide77X.scn	961	110	90.0	0.07	96.25
2	Slide75X.scn	900	190	94.0	0.06	98.18
3	Slide73X.scn	784	174	90.0	0.08	80.24
4	Slide81X.scn	900	225	90.0	0.07	88.1
5	Slide131X.scn	729	318	96.0	0.06	74.07
6	Slide12X.scn	841	217	90.0	0.08	82.32
7	Slide5X.scn	650	193	92.0	0.08	63.79
8	Slide10X.scn	625	58	90.0	0.08	62.24
9	Slide120X.scn	812	167	93.0	0.08	82.33
10	Slide123X.scn	529	175	89.0	0.08	53.37
11	Slide142X.scn	462	157	91.0	0.08	47.02
12	Slide16X.scn	729	115	90.0	0.06	69.42
13	Slide20X.scn	576	133	93.0	0.08	56.75
14	Slide4X.scn	702	141	92.0	0.08	68.05
15	Slide79X.scn	625	68	91.0	0.09	64.63
16	Slide80X.scn	676	106	91.0	0.08	71.68
17	Slide84X.scn	625	70	85.0	0.09	66.61
18	Slide9X.scn	676	117	88.0	0.08	66.16
19	Slide90X.scn	756	145	93.0	0.07	79.19
		Total cells detected:	2986		Total time:	1412.57
20	Parameters :					
21	confidenceLevel	confidenceLowLimit	factor	smallImageSize	threshold	
22	0.75	0.05	2	256	0.001	

Summary | scn16 | scn4 | scn123 | scn79 | scn12 | scn131 | scn90 | scn73 | scn80 | scn115



Runtime details

20 Slides processed in ~ 24 minutes.

Training the Neural Network ~ 48 hours of GPU time

To arrive at an average error value = 0.06
Over 20,000 iterations.

Why stop at 0.06 average error value ?

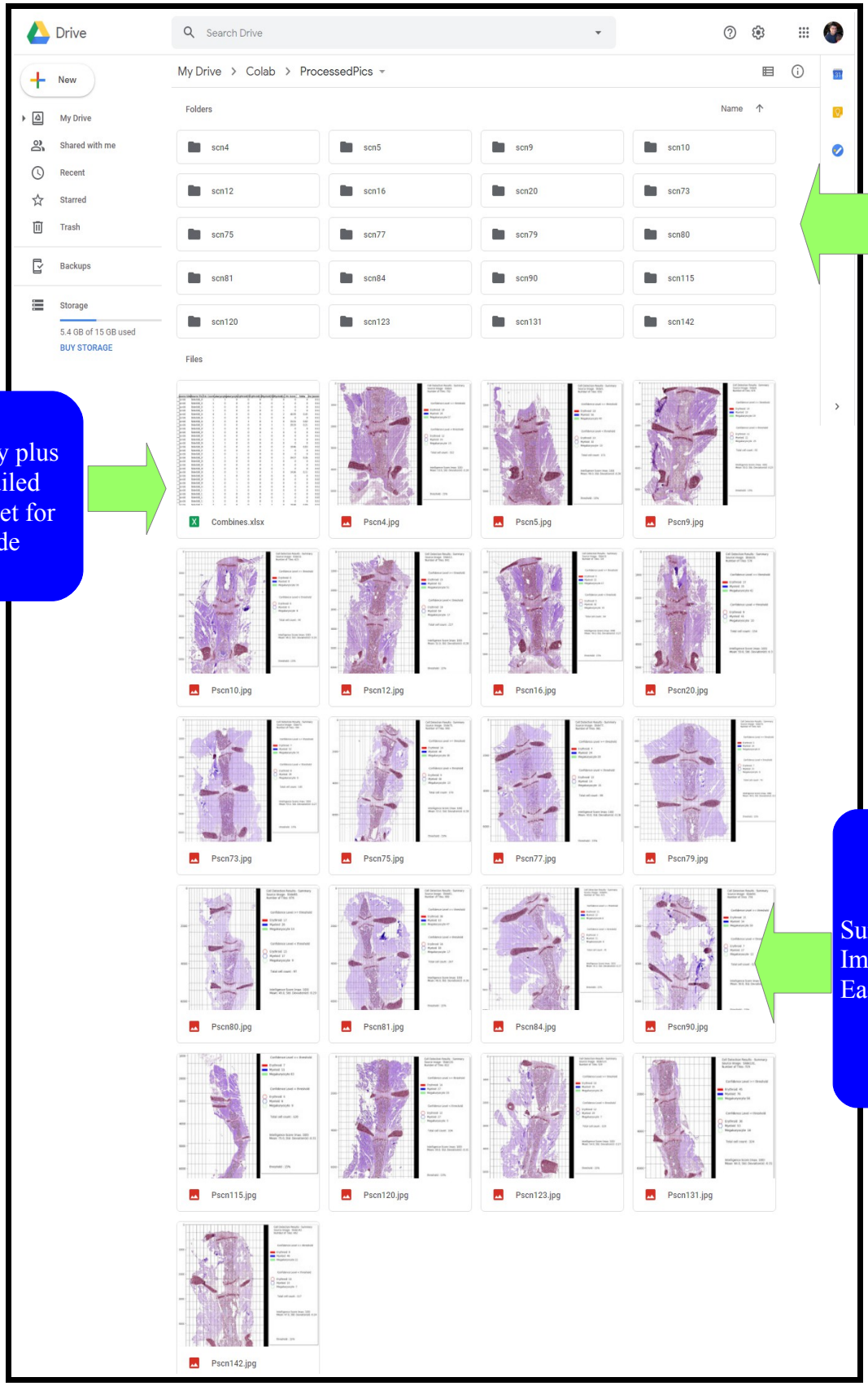
2 reasons:

1. In my estimate it is likely better than a human expert - only 6 errors per 100 instances over an extensive run.

2. Further error reduction is in the realm of dedicated computing resources.

But, I am most interested to see the results with a 0.01 average error. That training needs dedicated resources and perhaps 50,000 iterations. I ran slightly above 30,000 iterations.

Solution Output Organization



Folders With Annotated Tiles for Each Slide

Summary plus One detailed Worksheet for Each Slide

Summary Image for Each Slide

Identify hematopoietic cell lineages from bone marrow images

Challenge summary

Red blood cells, white blood cells and platelets all have fixed life expectancies and are replenished continuously. Blood forming stem cells, also known as hematopoeitic stem cells, residing mostly in the bone marrow, are the progenitors of such blood cells.

For drugs involved in cancer treament, hematotoxicity is dose-limiting as the bone marrow is affected by drug induced toxicity. Thus, identifying population shifts in the bone marrow caused by cancer drugs is of paramount importance to better understand and estimate the effect of drugs.

The challenge is to automate the identification and quantification of the three hematopoeitic cell lineages in sections of sternal bone marrow images from mice and rats – namely megakaryocytes, erythroid and myeloid cells.

Challenge requirements

1. Detailed solution description clearly demonstrating how it addresses the technical requirements.
2. Learn from the hand-annotated images suitably. Use the other scanned Leica .scn format images as necessary to improve learning/training.
3. Formulate an algorithm, preferably Artificial Intelligence (AI) and Neural Network(NN) based that can learn to identify the 3 hematopoietic cell types.
4. Clearly and unambiguously identify the three major hematopoietic cell types – namely Erythroid, Myeloid and Megakaytocytes in the 20 test images provided and submit the results. Output of the analysis for the 20 test images should be images with predicted boundaries of the labels (3 cell types) marked for easy visualization in either tiff or jpeg format.
5. The solution must be based on open-source tools.
6. Choose a suitable package for handling Leica .scn images and test it as Astrazeneca is leaving it to the Solver to determine best approach on this topic.

My approach

I have taken the approach that the results are to be used for comparative purposes to assess the effect of oncology treatments over time. Thus I have made the results not only data-driven but also paid attention to visuals. The visuals assist in calibrating the detection results to the point where the data can be trusted 100%.

From that point onwards, Summary slides with the Counts and Intelligent Scores must deliver all the requisite information, and the visual details like tiles will be used on demand.

Solution facets

I have broken the problem into 5 key areas:

1. *Data Preparation*

This is the challenge of converting .scn images to an equivalent set of .jpg or .png images that are easier to work with. I used bio-format, one of the suggested tools for image conversion.

2. *Machine Learning/AI for Training and Detection*

I used YOLOv3 in conjunction with darknet to build and train a Neural Network using the concept of Transfer Learning.

3. *Smart Platform Choice*

Both areas described above are memory/storage/processor hogs and thus this presents a third equally intriguing challenge. And while there is an abundance of cloud based platforms that can be spun up in no time, they are not economical. Vmware Virtual workstation and Google colab are two platforms I chose to use for this solution very effectively.

4. *Results-Data Organization and Presentation*

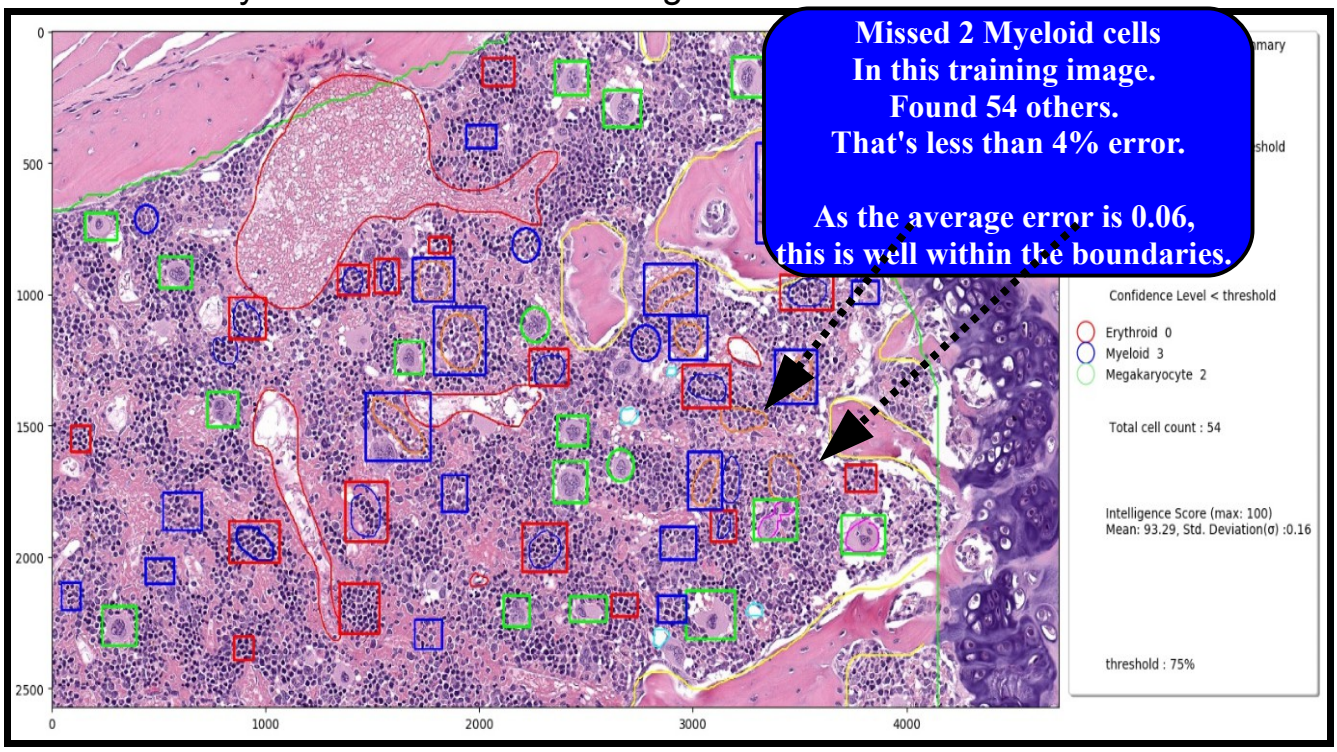
Simple organization of the data is key to interpreting the results of this exercise. I am using a combination of jpg images and excel worksheets to organize and present all my results optimally. It's detailed through out this document.

5. *Automation*

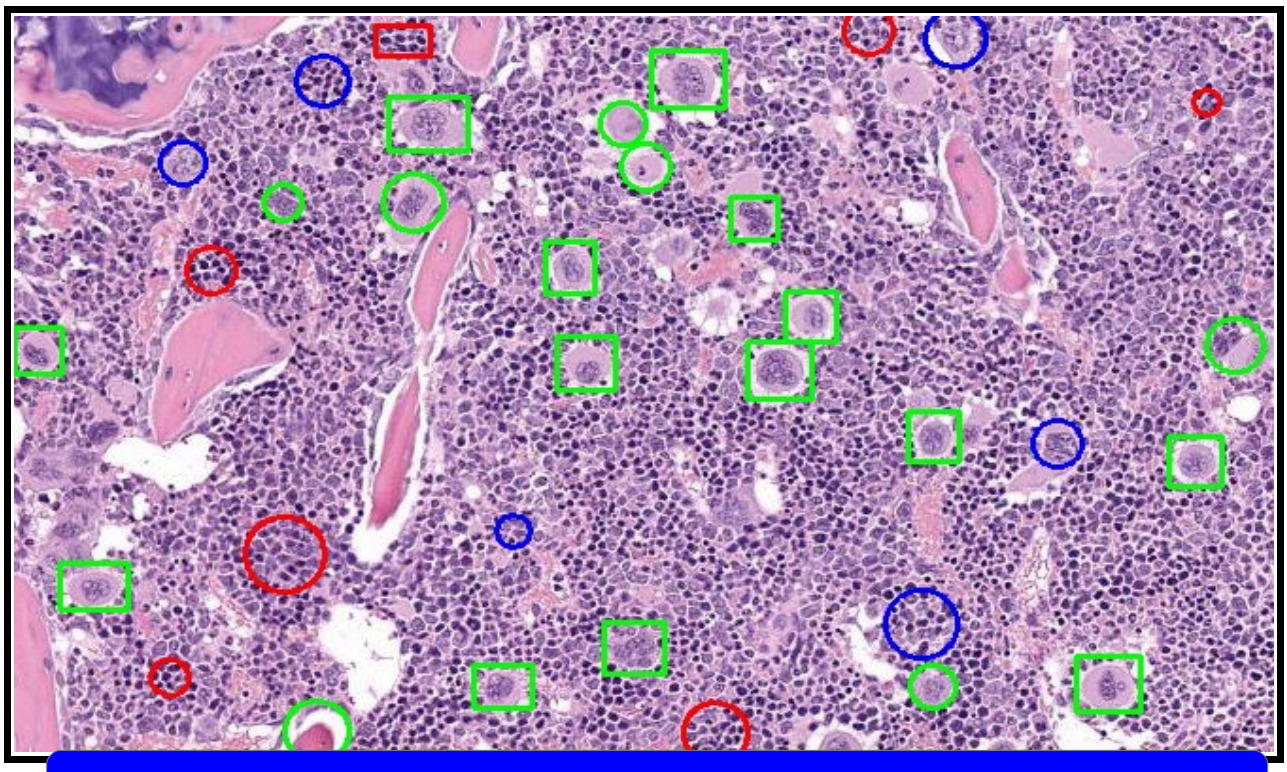
99% of the steps in my solution are automated. As a result the 20 huge images are transformed and analyzed automatically requiring just 1-2 manual steps.

Summary of Achievements

Annotated cell types detected and labeled with Intelligent Score with ~ 96% accuracy as shown for one training slide below:



Another example of cell-type detection.



Note: Rectangles indicate detection above probability-threshold

20 Slide Summary : Totals, Intelligent Scores & hard numbers

	Source File	Num Tiles	Total Cells	Intelligent Score	Stdev	Time (seconds)
0	Slide115X.scn	361	107	96.0	0.06	42.17
1	Slide77X.scn	961	110	90.0	0.07	96.25
2	Slide75X.scn	900	190	94.0	0.06	98.18
3	Slide73X.scn	784	174	90.0	0.08	80.24
4	Slide81X.scn	900	225	90.0	0.07	88.1
5	Slide131X.scn	729	318	96.0	0.06	74.07
6	Slide12X.scn	841	217	90.0	0.08	82.32
7	Slide5X.scn	650	193	92.0	0.08	63.79
8	Slide10X.scn	625	58	90.0	0.08	62.24
9	Slide120X.scn	812	167	93.0	0.08	82.33
10	Slide123X.scn	529	175	89.0	0.08	53.37
11	Slide142X.scn	462	157	91.0	0.08	47.02
12	Slide16X.scn	729	115	90.0	0.06	69.42
13	Slide20X.scn	576	133	93.0	0.08	56.75
14	Slide4X.scn	702	141	92.0	0.08	68.05
15	Slide79X.scn	625	68	91.0	0.09	64.63
16	Slide80X.scn	676	106	91.0	0.08	71.68
17	Slide84X.scn	625	70	85.0	0.09	66.61
18	Slide9X.scn	676	117	88.0	0.08	66.16
19	Slide90X.scn	756	145	93.0	0.07	79.19
		Total cells detected:	2986		Total time:	1412.57
20	Parameters :					
21	confidenceLevel	confidenceLowLimit	factor	smallImageSize	threshold	
22	0.75	0.05	2	256	0.001	

Summary | scn16 | scn4 | scn123 | scn79 | scn12 | scn131 | scn90 | scn73 | scn80 | scn115

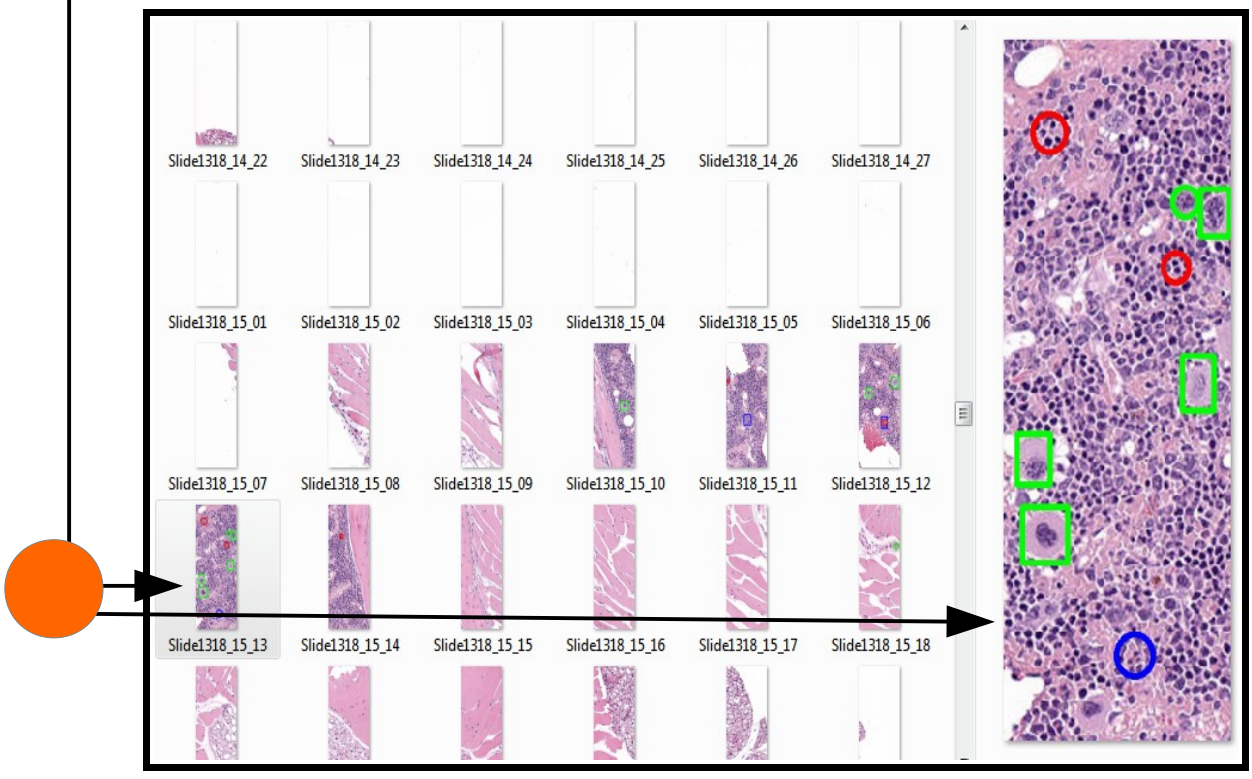
Notice that each Intelligent Score is presented with a standard deviation.

A low standard deviation indicates high data quality Compared to a high standard deviation value.

Summary of details per Slide - Tiles in Slide Worksheets

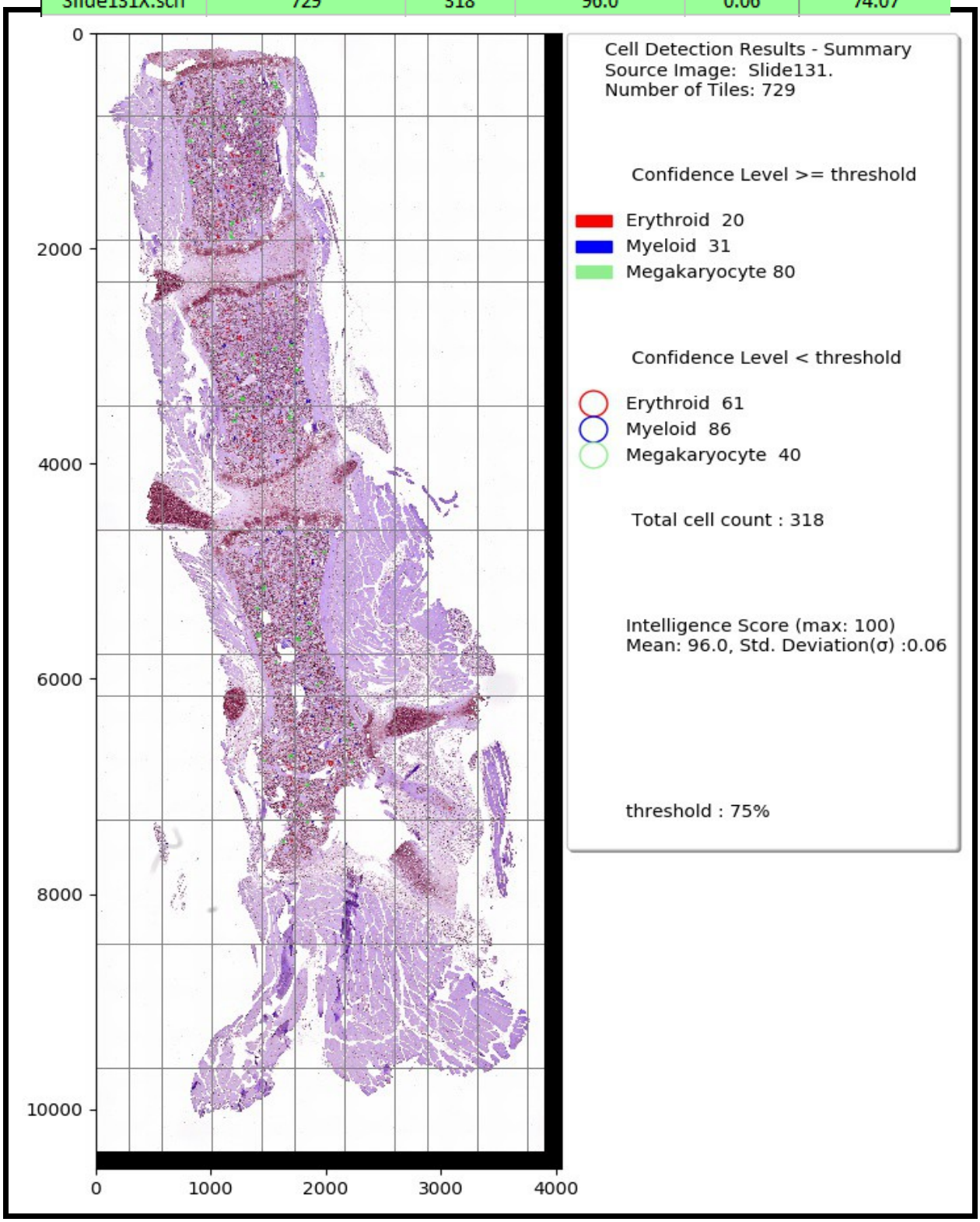
C	D	E	F	G	H	I	J	K	L	M
Source Tile	Tot. Count	Megakaryocytes(H)	Megakaryocytes(L)	Erythroid(H)	Erythroid(L)	Myeloid(H)	Myeloid(L)	Int. Score	Stdev	Time (seconds)
Slide1318_01_05	2	0	1	0	1	0	0	31.65	0.26	0.07
Slide1318_04_11	8	1	1	0	3	0	3	42.42	0.31	0.08
Slide1318_08_12	8	3	0	0	2	0	3	65.03	0.33	0.08
Slide1318_15_13	8	4	1	0	2	0	1	68.49	0.36	0.07

Summary of Tiles (labeled and annotated) per Slide - Visualization



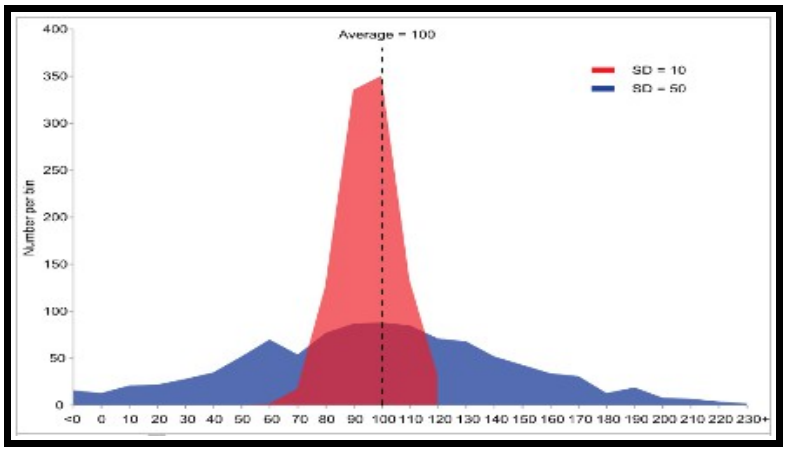
Summary for each slide – Hard Numbers plus Visualization

Source File	Num Tiles	Total Cells	Intelligent Score	Stdev	Time (seconds)
Slide131X.scn	729	318	96.0	0.06	74.07



Intelligent score (IS) with a must include Standard Deviation

The IS score ascribed to each image is based on confidence level of each of the hematopoietic cell-types. IS also includes a standard deviation (σ) number as this denotes the spread of the confidence



level around a mean. A mean without a stdev doesn't make sense. This picture above illustrates my point – without a stdev figure you don't know how close the data is distributed around the mean. More details on this topic later.

Completely open-source based through and throughout

Complete list of software packages used in the document later.

Super flexible – parameter driven

python3 ImageSlicerPlayLinux.py ***0.75 0.22 2 512***

This is the way the main module is invoked. This is the Module that calls all the other modules for analysis and reporting. As you can see there are only 4 configurable parameters

1. **upperConfidence** – above this confidence, cells are annotated with a rectangle, below this confidence cells are annotated with a circle.
2. **lowerConfidence** – below this value, they are discarded,
3. **magnification factor** - based on the tile size the tile may need magnification to be on par with the detection training algorithm. For example if you trained to look for a lawn mower and a car in an image, you need to magnify the image if the car is the size of an ant.
4. **tile size** – needs to be chosen in conjunction with the magnification factor such that objects in the image are "visible" to the detector Neural Network algorithm. A happy balance between the two need to be struck as smaller tiles → more tiles → higher processing overhead and perhaps sub-optimal detection.

Platform Choice

Poor first choice – Windows 7 Laptop without GPU

Initially I attempted to train a Neural Network on a Windows 7 laptop without any GPU leveraging VMWARE Virtual Workstation. It took 4-5 days to execute 400 steps of optimization with only only 1 object to detect. I needed at least 10,000 steps and train the NN for 3 different classes. It quickly became obvious that I needed a better computing platform. My Windows 7 laptop has a 1.5GHz Intel Celeron CPU with 8G of RAM.

Good free cloud-option – Google Colab with punishing restrictions

While it is easy to spin-up a system with limitless cpu/gpu/ram in the cloud; and there are numerous vendors who provide these options, those options are expensive. Thus I started looking for more affordable options failing which I would have to consider buying my own computer with a fast GPU and at least 16G of RAM. Storage was not a concern as SSD hard drives are cheap.

Colab provided by Google had all the answers – meaning CPU/GPU/Storage.

But the limitations are:

-13G of RAM

-12 hours usage limits

This is inconvenient.

Conclusion – best to get a dedicated CPU/GPU/RAM/Storage combo

After a couple of months of battling colab my recommendation is that a better option is to rent resources in the cloud e.g. in AWS. But this is not free and thus a bit of prep is needed.

AI/ML for designing a Neural Network to detect blood cells

Data Preparation

Choice of bio-formats for image conversion

This is because I wanted pliable file formats. After spending a lot of time with the various tools that have been suggested I decided to use **bio-formats** as they had a relatively easy python interface. I used bio-format to convert from SCN → TIFF → PNG. I worked with png format images.

Tile size determination

This pertains to decomposing large images into tiles, which are fed to the trained Neural Network. The size of the tiles is very important as the objects in the tiles need to coincide roughly with the size of the objects the NN has been trained on. This is easily settled with visuals and raw numbers – the optimal size I arrived at for the test slides is a 256 x 256 sized tile.

As this is an input parameter to the main python program, this can be easily altered to handle other image Slides needing different magnifications and thus differently sized tiles.

Training

Annotate and label training images

Tool used: LabelImg (opensource)

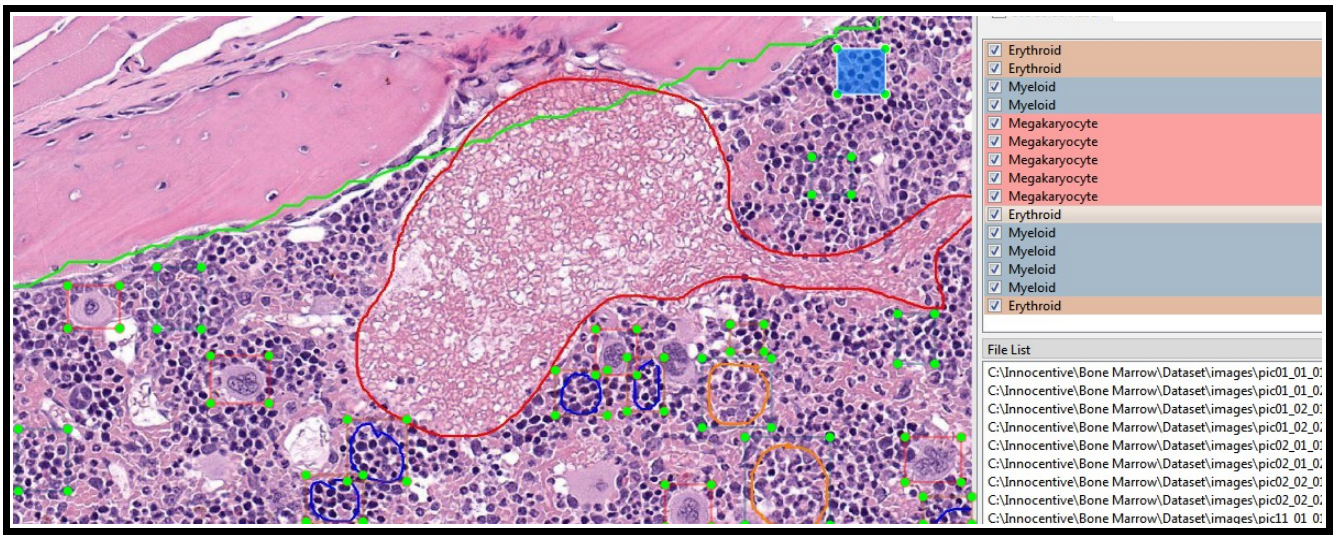
This step creates annotations on the images themselves as shown and captures 5 essential elements per object that are identified in the image namely:

1. class-type - meaning Erythroid/Megakaryocyte/Myeloid

2. CenterX, 3. CenterY – of the cell or body of cells

4. Width, 5. Height – of the rectangle enclosing the cell/body of cells

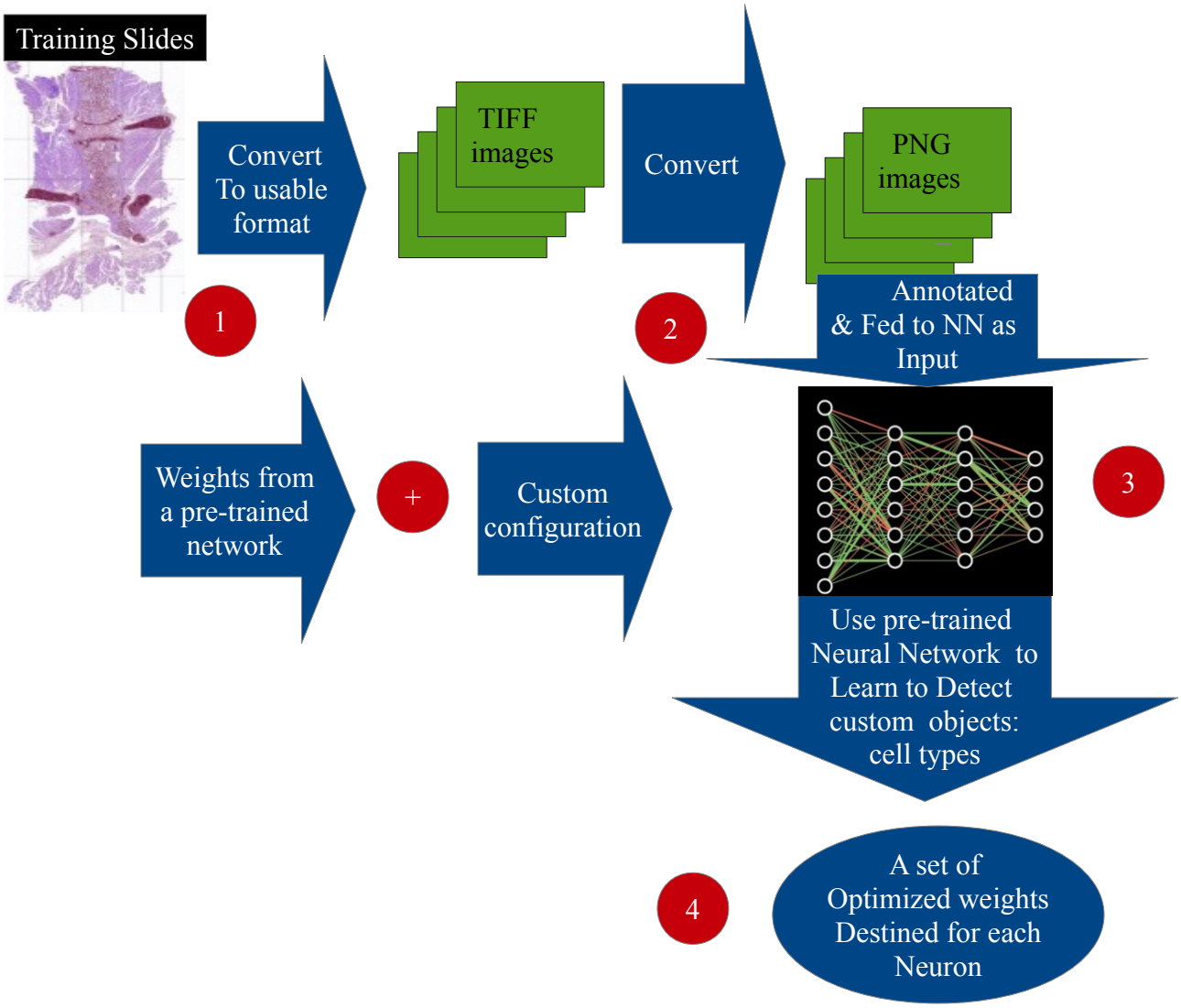
These 5 attributes for each of the objects identified in each image are stored in a file that form the label file for the specific image. Thus each training image has a correspondingly named label file that carries the 5 data points for each of the objects (cell-types in our case) identified in the image. This is a labor intensive manual process. And it has to be done carefully for each of the training images.



Transfer Learning/Training for custom Object Detection

For this solution, I leverage an existing Neural Network (NN) that has already been pre-trained on multiple image classification tasks over extensive time. I chose a Neural Network with a reputation for speed and accuracy - Yolo (You only look once). I trained this NN to learn to detect custom objects, in this case the 3 blood cell types, using the training images as input to the NN.

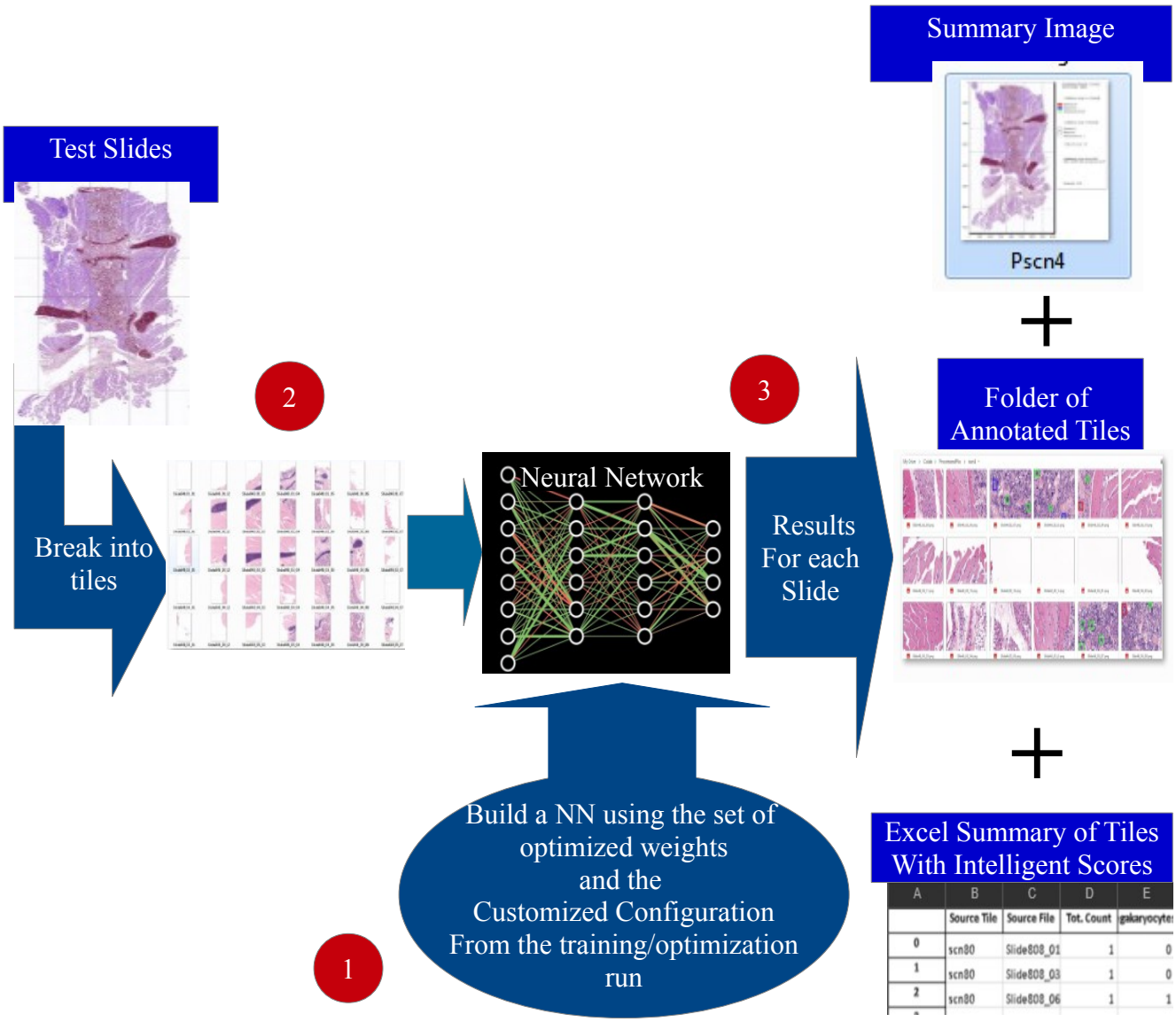
Once satisfied with the optimization (error below a certain threshold), a set of weights is obtained which paired with the configuration files, define the newly trained NN.



Object detection and Labeling cell types

Leveraging the set of weights we obtained from the previous exercise,

first build a NN with the same configuration as the previous trained NN meaning using the same configuration files (same Input, Output and Hidden layers). Load the weights obtained from the Learning/training/optimization exercise into this Neural Network (this defines the behavior of each Neuron). This picture explains it.



Training algorithm choice – YOLO /darknet

1. My algorithm is based on YOLOv3 (You only look once) which has been extensively trained and has a reputation for speed and accuracy. This is an extremely good base to start from as the extensive pre-training and the accuracy means that we can automatically leverage a very mature and fine-tuned NN. Moreover the NN is highly configurable and optimized for the specific task at hand. While the original YOLOv3 algorithm is trained for detecting nearly 100 objects, I used the transfer-learning method to train it for our specific purpose, namely, to detect the hemapoietic cells quite successfully.

2. Overall a very robust, solid and yet lightweight solution. This makes it portable and even usable in other platforms. All you have to do is to transport the configuration of the NN and the weights found by optimization and voila – you have a replica network in another platform, ready for detection.

3. While there are a few other ML platforms like Tensorflow, I chose darknet as it struck a right balance between simplicity and configurability. It made details available to the user via configuration parameters while keeping things relatively simple from a usability point of view. With darknet, the outcome of optimization of the NN was simple – a set of weights that in conjunction with the configuration file defines a Neural Network. I used this Neural Network to solve this challenge.

4. I used darknet/yolo in Unix; however the same can be also used in a Windows environment. I liked this flexibility as I used the Windows platform to verify multiple smaller modules during development.

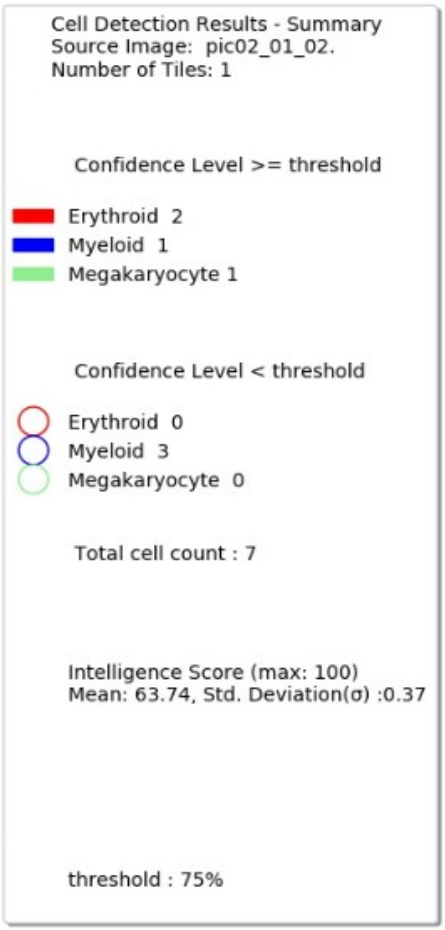
5. Darknet has a lot of support available online. Every question I had had some bearing in the internet either directly or indirectly – asked by someone already. This proved to be a huge advantage.

Intelligent Score -explanation

A word or two about the Intelligent Scoring method used in this solution – and I have chosen some low numbers to drive home my point. Here goes:

1. The Intelligent score is a ***combination*** of the ***mean*** and ***stdev*** (standard deviation)of the confidence levels associated with each cell-type identification for all cells above a threshold. Thus for this result fragment from detecting multiple tiles for one Slide:

```
0 .Restypes & confidence  b'Myeloid' 0.9906082153320312 pic02_01_02.png
1 .Restypes & confidence  b'Megakaryocyte' 0.9711456894874573 pic02_01_02.png
2 .Restypes & confidence  b'Erythroid' 0.9423680305480957 pic02_01_02.png
3 .Restypes & confidence  b'Erythroid' 0.7970065474510193 pic02_01_02.png
4 .Restypes & confidence  b'Myeloid' 0.3794688880443573 pic02_01_02.png
5 .Restypes & confidence  b'Myeloid' 0.29751867055892944 pic02_01_02.png
6 .Restypes & confidence  b'Myeloid' 0.08387193083763123 pic02_01_02.png
```

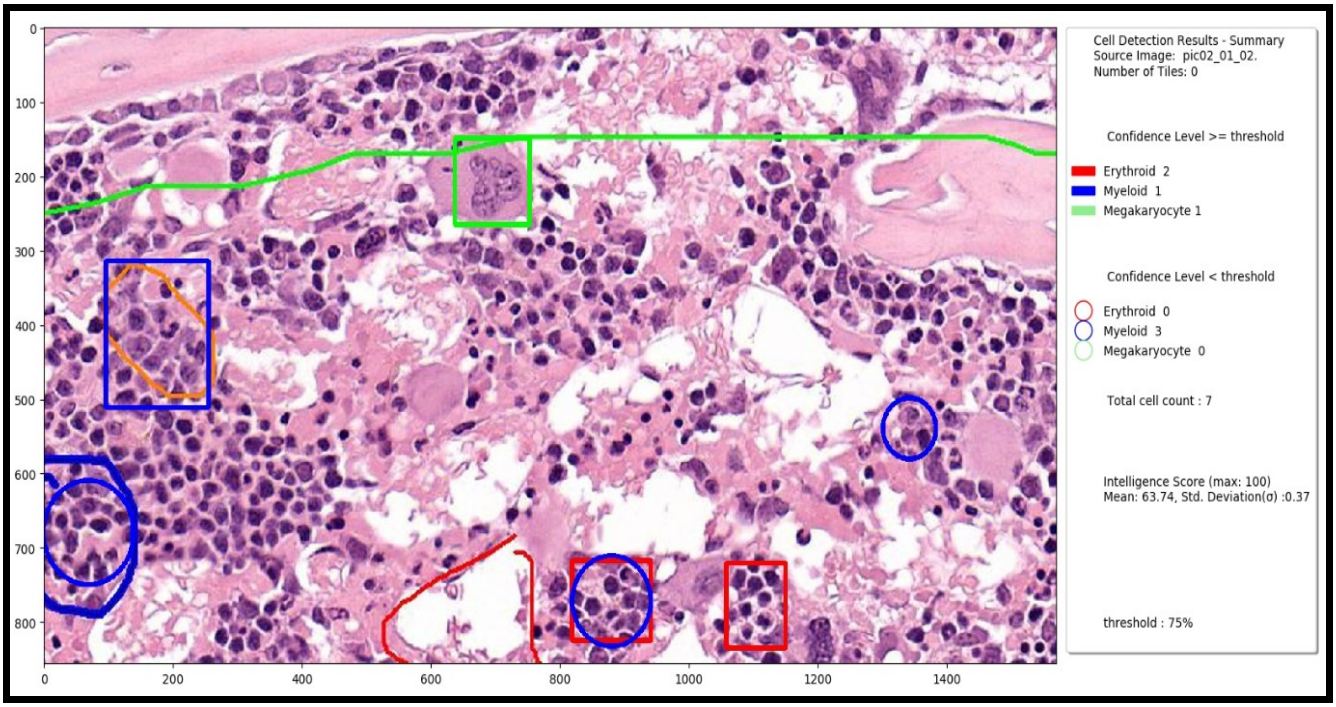


This corresponding data above represents the Legend to the left :

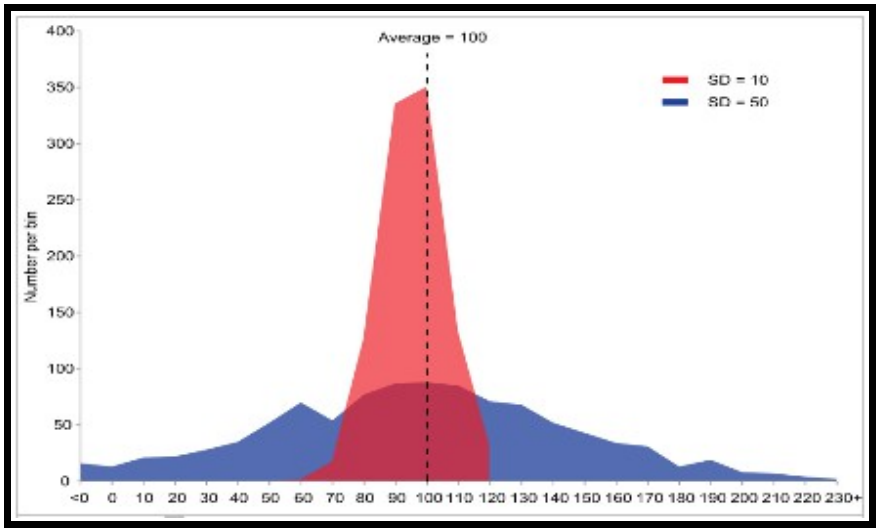
Notice that only a few low confidence numbers can bring down the mean substantially. This makes the standard deviation larger as now the data spread is wider. And thus the standard deviation is most important to consider along with the mean and it shows how tight the data sample is around the mean.

2. The threshold of 75% is not a “bad” or a “low” threshold number. It just goes to show that at 75% thresholding my algorithm finds and identifies cells with a very low stdev. If you look at my results, the stdev is at or below 0.09. This emphasizes the strength of the algorithms I have used.

Cells with confidence lower than the threshold are annotated with a circle. All cells with confidence levels of threshold and above are annotated with a rectangle.



Intelligent Score (IS) has a max achievable value of 100. As each detected cell has a corresponding confidence value in percentage, this solution leverages the confidence value to calculate the IS. While many combinations are possible, I decided to keep it simple and use a threshold. Any confidence value exceeding that threshold is added to a set and the mean and standard deviation of this set of scores represents the overall IS of the slide. And then I multiply the mean by 100.



As this picture clearly shows, a mean doesn't mean much without a standard deviation as this represents the quality of the overall data. Standard deviation is an excellent indicator of the spread of the values around a mean.

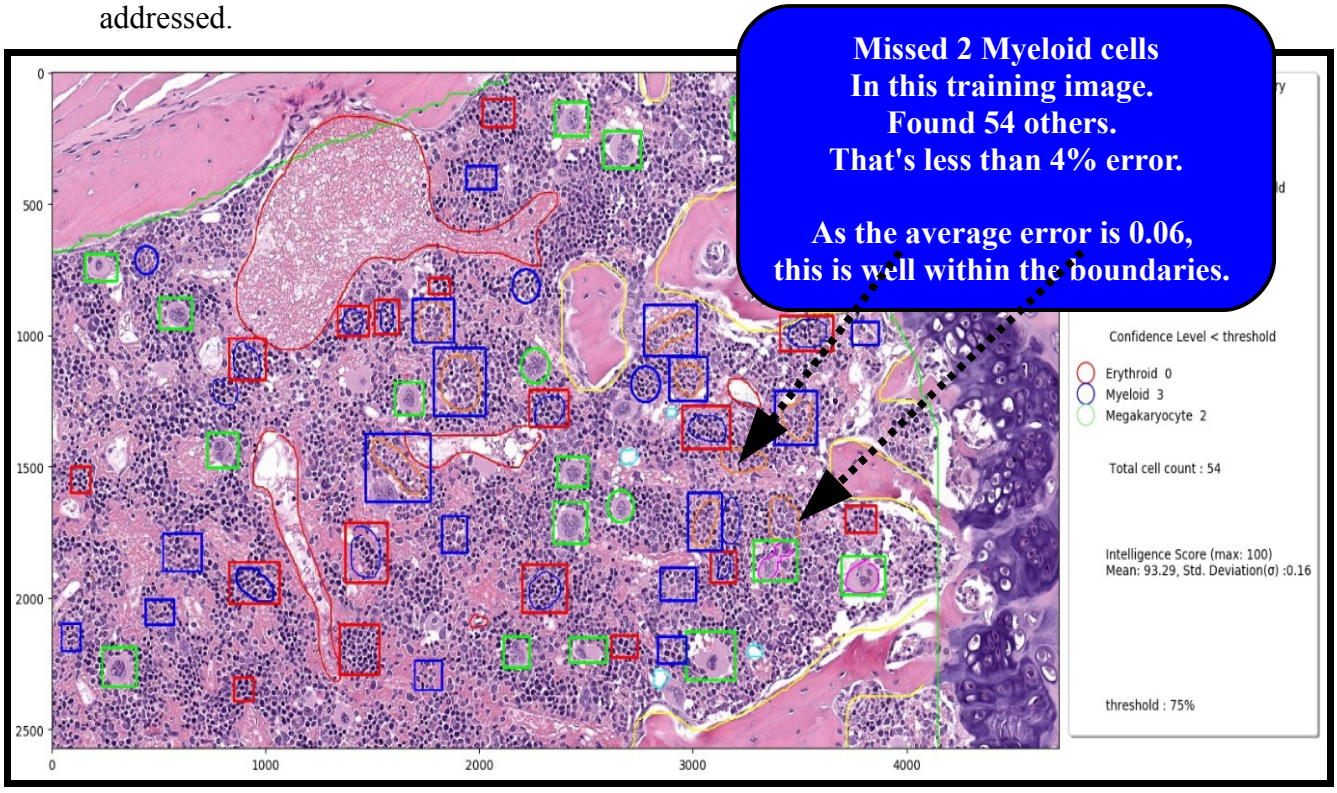
Key observations (O), ideas (I) and errors (E)

1. **O: Yolo** is an excellent algorithm for customized object detection. While there are numerous configuration parameters involved, it is fairly easy to learn as the internet contains helpful notes. It gives you the same controls as Tensorflow while at the same time hiding some of the nitty-gritty details. For example, for a given number of objects it guides you with the size of the NN you are encouraged to use via configuration files. And its easy portability across platforms is another very attractive attribute. Yolo has no problem detecting small objects if you care to follow the correct steps to train it well.

2. **O: Python-opencv** was a very useful set of tools which because of its cross-platform capability was used a lot for this exercise. I would test modules on my Windows 7 laptop and VMWare Virtual Ubuntu platform (on the same laptop) before throwing it out on google colab.

3. **O**: The **darknet** environment provided by J Redmon is an excellent platform for custom object detection as it comes with a set of absolutely simple and elegant tools for that objective.

4. **E: More Myeloid cells annotations** in the training images will certainly help to make my solution better and more effective and will vastly improve detection. As you can see below I missed 2 Myeloid cells and I am sure with additional training images this deficiency can be addressed.



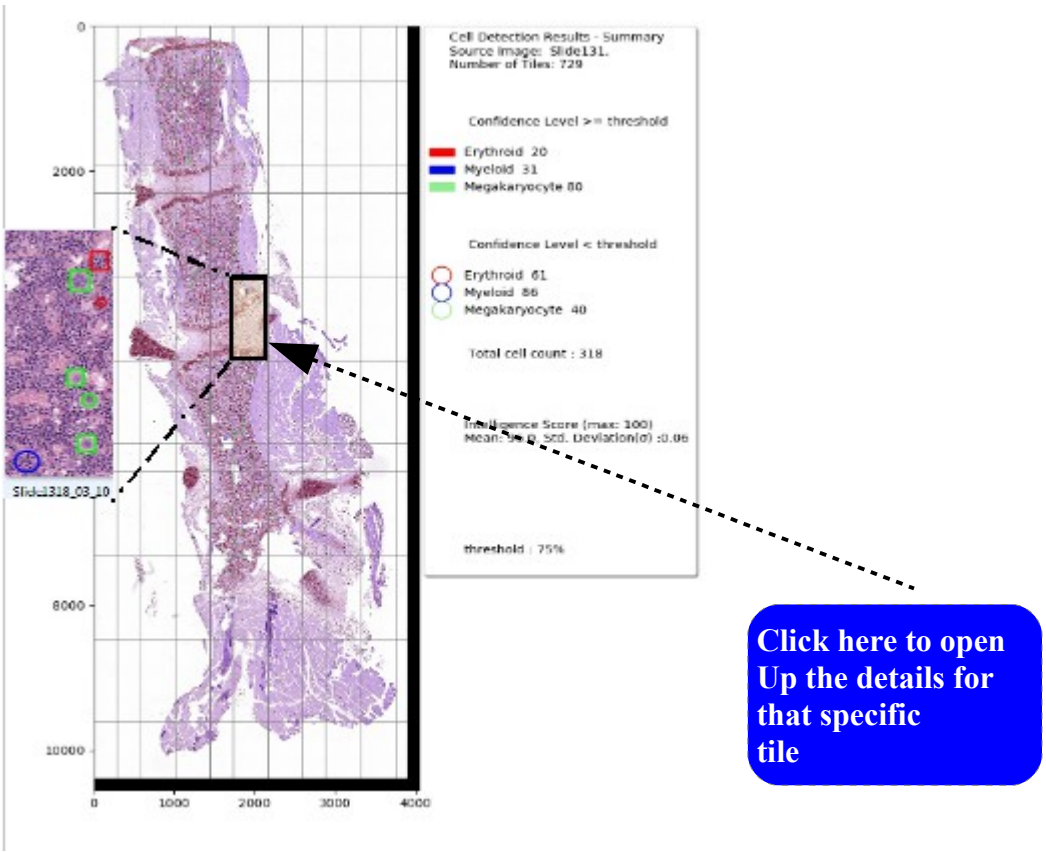
5. **I** : I relied heavily on **google colab** for training as a GPU is the ideal platform for such a computationally heavy exercise. Instead of this it is best to procure an i7 desktop with a fast

Nvidia GPU for self reliance. The google environment is too disruptive and wasted a lot of time.

6. **I**: I used tiling to break large images into smaller image tiles, which I then coalesced to form a the bigger image with all the annotations after object detection and counting. I store the annotated small tiles in a folder named after the parent Slide.

   For the future, this can be made interactive where the small tiles can be accessed easy access via clicking the large image. It is fairly easy to do and will provide more clarity to what you are after. In that case each small tile/image can be provided with its own legend. I **deliberately didn't add a legend to each tile** this time as this would make the tile look extremely crowded.

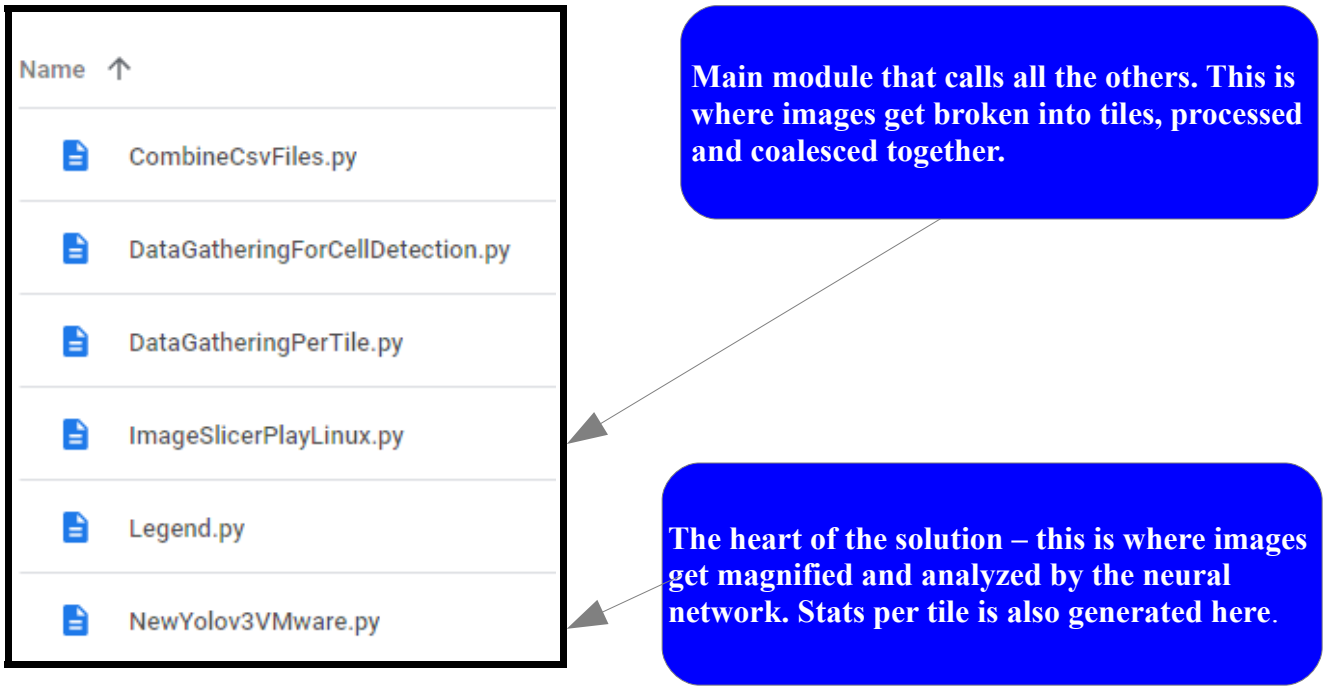
   However, of that is desired, it is but a one line code change and easily done. This is what I am talking about:



Implementation Details

A. 6 python modules

Names are self-explanatory.



B. Jupyter notebooks to run on google colab to

1. convert SCN images to TIFF to PNG
2. Train the Neural Network using Transfer learning
3. Run the trained Neural Network to detect blood cells in the PNG images

C. Open-source packages used

- LabelImg – to annotate images and produce label.txt files containing object details for each image file.

- opencv – to harness Video/Image processing libraries(sudo apt-get install python-opencv) (https://docs.opencv.org/3.4/d2/de6/tutorial_py_setup_in_ubuntu.html)

- darknet – for leveraging a training and detection platform. Darknet is an opensource NN network written in C and CUDA. It is fast, easy to install and is capable of running both in CPU and GPU environments. Below are the attributes of darknet. `@misc{darknet13)`

```
  author =    {Joseph Redmon},
  title =     {Darknet: Open Source Neural Networks in C},
  howpublished = {\url{http://pjreddie.com/darknet/}},
  year = {2013--2016}
}
```

- dos2unix – for handling .txt and .data and .names files' suitability for use in unix after being shipped from Windows (Vmware workstation in Windows)

- Bio-format: (bfformats-python) - an open-source software tool for reading and writing proprietary microscopic image data. I use it here to read .scn images and convert them into tiff (Tagged Image File Format) and then to png (Portable Network Graphics) format.

- Ubuntu – a linux platform https://ubuntu.com/

- Vmware Virtual Workstation – used to prototype and test python code before throwing it on colab.

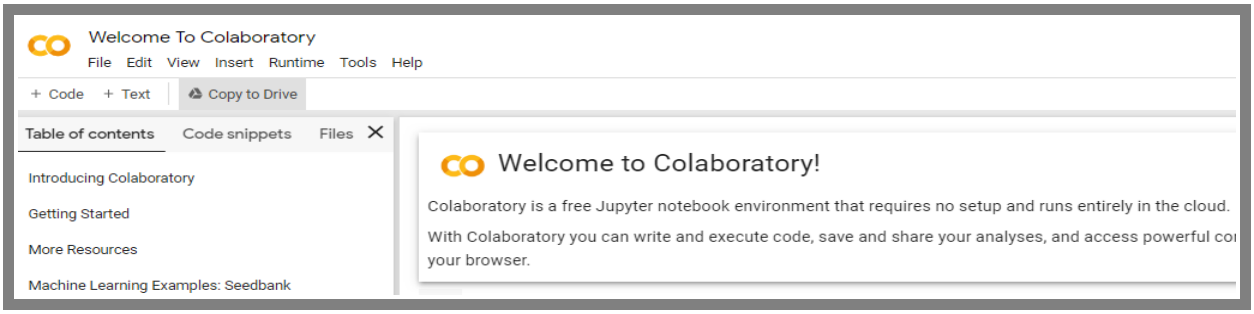
- PIL – Python Imaging library

- Python

- Jupyter notebook in conjunction with google colab

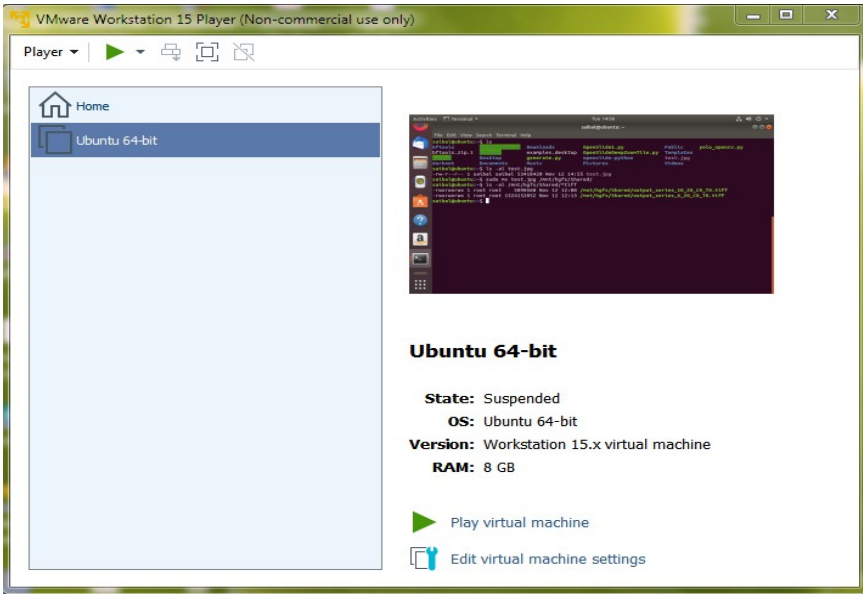
D. Primary Environment: Google colab/google drive

A free (12- hour usage limit) environment – proved to be very useful with 13G of RAM and GPU provided for computing. It was a much faster environment to use than the laptop obviously. However the handicap was the jupyter programming environment which meant there was a fixed setup time for each usage session.



E. Secondary Environment : VMWare Virtual Desktop/ubuntu

Every change I made in colab I pre-tested in the VMWare/ubuntu environment on my Windows laptop. This proved to be a very useful environment for testing proof-of-concept ideas.



Further thoughts

1. More training will improve results and so will additional annotations of Myeloid cells types. With inputs from just 5 annotated slides, the accuracy of the learning algorithm is quite impressive. Increasing the training input by 5-10 additional annotated slides will significantly improve the detection mechanism.

2. A dedicated desktop with i7 cpu, 16G RAM and at least a Nvidia 1080Ti GPU would have been ideal for this exercise as this would limit the time I wasted struggling with Google colab. For Universities or a Research Environment that's the path I would choose. And this is achievable within $2k.

3. Overall, this approach will automate the study of oncology therapeutics vastly. If a lineup of drugs are being tested in parallel, this system will pave the way to study the effect of the different drug lines, compare them and assist in selecting the ones based on impact on one or other cell types. Drug testing and elimination of unsuitable candidates will become very efficient and rapid.

4. Automation that I have implemented is already significant but there is room for even more automation which will speed-up cell detection further.