CAPSTONE PROJECT: SECOND PROGRESS REPORT

Table of Contents

1. Data Preprocessing
    a. Patch Generation
    b. Limitations
    c. Train, Validation and Test Strategy
2. Modelling Approaches and Results
    a. Hard Coded Features
    b. Basic ConvNet Architecture
    c. ResNet inspired Network Architecture
3. Future Work

1. Data Preprocessing

1.1 Patch Generation

We were able to perform the automatic center detection mentioned as a next step in the first report. For that purpose, we used the Hough Circle Transform[1] function implemented in OpenCV library. Manually adjusting the parameters of the Hough Circle Transform function, we were able to identify circle centers and corresponding radius on the petri-dish images. As several circles were identified, we found the mean circle and removed the petri-dish border.

Fig. 1. Circles identified with Hough Circle Transform

For 'Serial' type images, we also removed the numbers used to annotate the different bacterial culture concentrations since they are meant to facilitate the analysis of samples by humans and can add bias to our classifier.

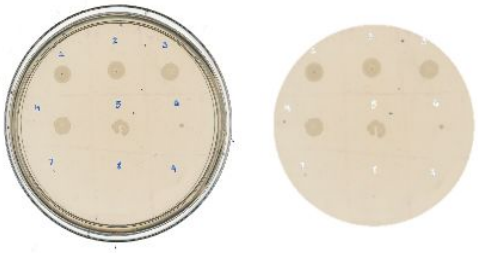


Fig. 2. Removal of petri-dish border and background numbers on 'Serial' image

Image Annotation

We initially used Simple Thresholding[2] on a grayscale version of the pre-processed images to create masks that would help us identify pixels corresponding to bacterial colonies areas. This way darker pixels above a certain threshold were labelled as “positive” regions - i.e. containing bacteria colonies - while the background was hidden or discarded. Since certain samples of images are brighter than others, we manually adjusted the threshold value to ensure that no portion of bacteria colonies were left out.

After highlighting the darker patches within each image, we noticed that some masked images had portions of the background that were erroneously highlighted. To fix this, we took the groups of pixel that were highlighted by our mask, measured the amount of pixels contained within each cluster - defined as a group of pixels connected to each other - and programmatically removed the smallest clusters within each picture. This second step improved the quality of our “masking” and as a result we could automatically isolate the bacterial colonies within a picture (Fig. 4).

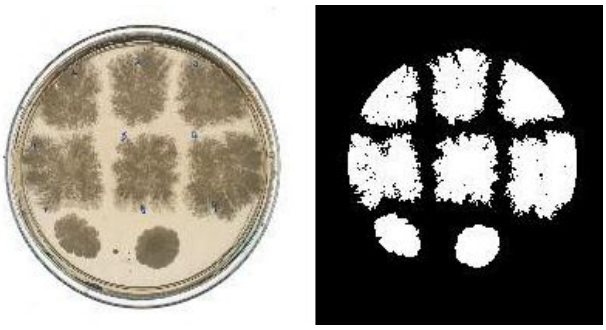


Fig. 3. Example of image annotation / positive masks

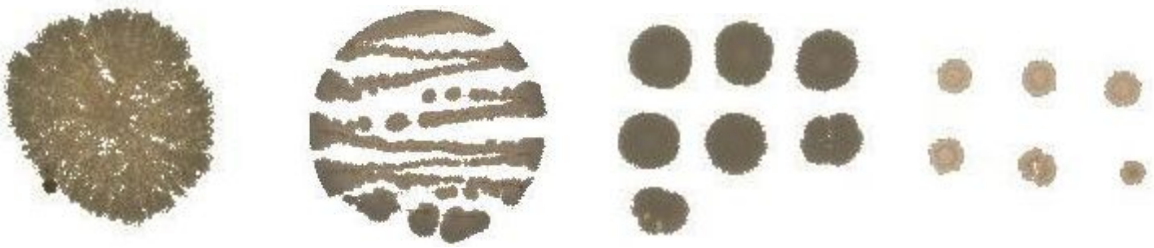


Fig. 4. Images with only positive pixels

Patch Generator

After removing the background from each image, we created a function to automatically generate the patches to be used for training. This can be thought of as a sliding window of size (n, n), where n is the desired length/width of the patches. This window traverses the image left-to-right capturing a patch each time it moves, with the number of pixels covered at each movement defined by its “lateral stride”. For example, when the lateral stride is set to 1, the

sliding window will move 1 pixel to the right at each iteration and so on. A separate “vertical stride” parameter can be also set to regulate the amount of pixels that the sliding window moves down each time it completes a row. As a consequence, using stride values smaller than the patch dimension resulted in overlapping regions, which allowed us to augment the size of our dataset if needed.

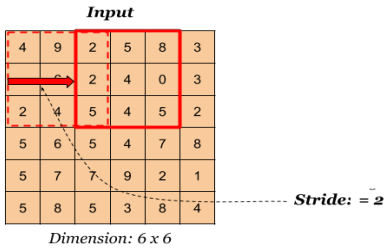
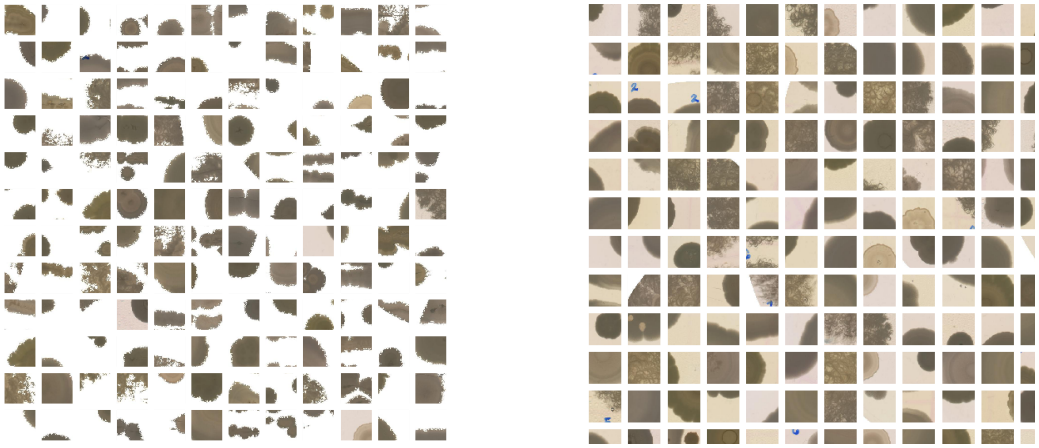


Fig. 5. Example of horizontal ‘stride’.

In addition, the function has the ability to generate patches with or without background, which enabled us to experiment with different combinations of training data.



(a) (b)

Fig. 6. (a): patches with only positive pixels. (b): patches including dish background

Training Classes

Unlike what we had at the time of our first report, we ended up using 8 different classes as shown in the table below. We made this decision after hearing the feedback from the project sponsors, who clarified which samples might belong to the same species and should be considered as the same class for the scope of this project.

Class ID	Class Name	Sample / Location
1	Bacillus megaterium	P1C1
2	Bacillus aryabhattai	P1C2+P1C3
3	Bacillus paranthracis	P1C4+P1C7
4	Lysinibacillus boronitolerans	P1C5
5	Bacillus mycoides	P1C6
6	Bacillus wiedmannii - 1	P1C8
7	Bacillus bingmayongensis	P1C9
8	Bacillus wiedmannii - 2	P1C10

Table 1. Mapping of training classes. background

1.2 Limitations of Sampling Approach

Classes	1	2	3	4	5	6	7	8
Weights	23	14	3.8	7.7	26	17.8	44	2.65
Normalised	0.1655272	0.1007557	0.027348	0.0554156	0.1871177	0.1281036	0.3166607	0.0190716
Control Patches	155	261	963	471	141	205	83	1376

Table. 2.Counts of Patches generated on Control Set and

After extracting and calculating the total number of patches for each bacteria class, we found out the number of positive patch samples is not equally distributed among classes. As shown in table 1, Class 7 has least patches. The imbalance is due to the image processing, which uses similar size patches for every colony. In the case of colonies such as P1C9, their small size relative to other bacteria's will result in fewer patches. In order for our performance metric not to be affected by the unbalanced class distribution, we decided to reweigh our loss function to be inversely proportional to the occurence of these classes during training our models.

1.3 Setting Train and Validation set

For training our model, we used patches from Serial data as our Training set and those from Control data as a validation set. We thought this was necessary due to limited availability of data, even though that could compromise the performance of our model. Since 'Serial' and 'Control' type images both correspond to cultures of the same bacterial species, this approach seemed reasonable at first.

1.3.1 Information Leakage with random sampling:

Initially, we generated the training and validation datasets by generating overlapping patches across different image regions and randomly assigning each patch to one of the two sets. However, after getting validation accuracy above 90% with a simple baseline model,we realized that we had overlapping patches from the same image section distributed in both training and validation, which explained the overly-optimistic result.

In addition, we realized that for Control and Streak samples there are often only one full image for each class, which probably caused our model to overfit.

1.3.2 Problem with Streak Data:

'Streak' dishes have very different visual growing patterns as compared to 'Serial' or 'Control' and the individual colonies are hardly distinguishable at the naked eye. One the other hand, the outline of each individual colony is easily identifiable on Control and Serial dishes. Unsurprisingly, training our model solely on 'Streak' data and validating on either 'Control' or 'Serial' leads to worse results (with prediction accuracy around 55% on balanced validation set) than training on Serial and validating on Control or vice-versa.

Thus, we are currently using **'Serial'** data for **training** and **'Control'** data for **validating** the model. The only limitation we foresaw with this approach was that 'Serial' images correspond to petri-dishes with several colonies growing in the same petri-dish, competing for resources, which might have an effect on the growing patterns as compared to 'Control' images, which correspond to petri-dishes with a single colony on the petri-dish. Therefore, training on Control/Serial and validating on the other might not capture all the features that are unique to that bacteria grown under specific conditions, potentially leading to poorer performance. However, we have no way to test this effect due to limited availability of data.

2. Modelling Approach

2.1 Hard Coded Features

In the literature of image processing, most research is focusing on how to extract useful features before deep learning came to our view. Here we adopt two global features: Hu moments[5] and Haralick textures[6]. The former captures information about the shape of the colony, while Haralick textures encodes texture information. Based on our colony images, shape and texture are two important features to help us identify the species. Part of our codes are referenced by the article [7].

After extracting the features, we implement seven basic machine learning models including: linear regression, linear discriminant analysis, k-nearest neighbors, decision trees, random forests, gaussian naive bayes and support vector machine. We got the results on the validation set as follows:

	LR	LDA	KNN	CART	RF	NB	SVM
64*64 patches	54.51%	61.30%	67.49%	59.75%	73.34%	29.46%	49.50%
128*128 patches	51.64%	60.28%	66.37%	62.41%	68.06%	40.31%	39.87%

Table. 3. Balanced CV accuracy scores in training sets across all models

We observed that our best model, **random forest** performed poorly on the validation data, and achieved the balanced accuracy score of 38.72% and 49.48% for patch size of **64*64** and **128*128** respectively. This may be attributed to inherent differences in the population of Control and Serial datasets.

2.2 Basic ConvNet Architecture

Since the images we were trying to classify are small patches that do not contain many unique patterns, we were motivated to explore the effectiveness of simple convolutional neural network based architectures to solve this task. We restricted ourselves to two different patch sizes 64x64 and 128x128 and followed the same train and validation approach. We have used dropout for regularisation and Adam optimizer for updating model weights.

Model Structure

```
Net(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv2): Conv2d(6, 12, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv2_drop): Dropout2d(p=0.5, inplace=False)
  (conv3): Conv2d(12, 12, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv3_drop): Dropout2d(p=0.5, inplace=False)
  (conv4): Conv2d(12, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv4_drop): Dropout2d(p=0.5, inplace=False)
  (fc1): Linear(in_features=1024, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=10, bias=True)
)
```

Fig. 7: Architecture of a Basic ConvNet Model on 128*128 patches

Prediction Result

After training on serial data and validating on control data for 10 epochs, we obtained the prediction accuracy shown in table 4. Considering the imbalanced distribution of classes, we used the inverse of class size to reweight the loss function and the accuracy. From the training and validation curves we observe that even though train and validation loss go down together, our model clearly overfits on the training data.

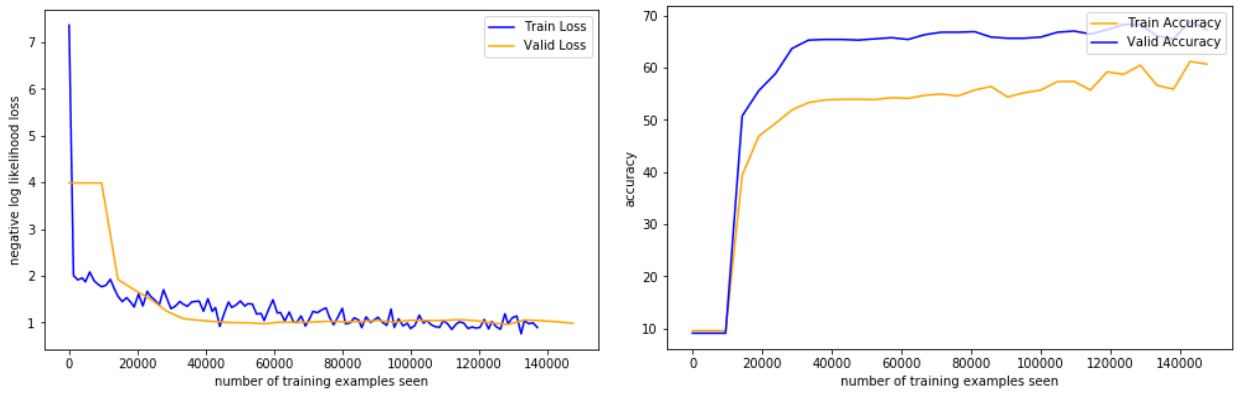


Fig. 8: Train & Validation Curves for Basic ConvNet Model on 128*128 patches

	Size 128 * 128	Size 64 * 64
Train Set (Serial)	0.767	0.745
Validation Set (Control)	0.378	0.467

Table. 4. Balanced Accuracy Scores for different patch sizes

We see that our models have a much worse performance on the validation set than on the training set. On the other hand, the model built on patches of size 64 fared better than its counterpart built on 128x128 patches.

		Predicted class									
		C1	C2-3	C4-7	C5	C6	C8	C9	C10	Total / support	Recall
Actual class	C1	91	54	0	0	65	17	0	1	228	0.40
	C2-3	56	262	8	0	38	38	0	7	409	0.64
	C4-7	0	19	574	600	7	5	0	21	1226	0.47
	C5	0	4	166	420	3	0	0	8	601	0.70
	C6	8	25	15	4	122	76	0	1	251	0.49
	C8	3	15	1	0	49	285	0	0	353	0.81
	C9	0	0	0	0	0	0	87	1	88	0.99
	C10	0	18	26	42	11	0	0	850	947	0.90
	Precision	0.58	0.66	0.73	0.39	0.41	0.68	1.00	0.96		

Fig. 9: Confusion Matrix of validation set (64*64)

		Predicted class									
		C1	C2-3	C4-7	C5	C6	C8	C9	C10	Total / support	Recall
Actual class	C1	10	4	0	0	31	4	0	0	49	0.20
	C2-3	9	49	0	1	11	11	0	3	84	0.58
	C4-7	0	2	102	189	0	2	0	1	296	0.34
	C5	0	0	52	89	0	0	0	2	143	0.62
	C6	1	4	1	2	28	18	0	0	54	0.52
	C8	0	8	1	0	15	52	0	0	76	0.68
	C9	0	0	0	0	0	0	11	0	11	1.00
	C10	0	0	7	8	11	0	0	228	254	0.90
	Precision	0.50	0.73	0.63	0.31	0.29	0.60	1.00	0.97		

Fig. 10: Confusion Matrix of validation set (128*128)

If we focus on fig 10, the confusion matrix of model on 128*128 patch shows that it cannot clearly separate between the following pairs (3,4), (5,6) and (1, 5).

2.3 ResNet inspired Network Architecture

We also used a Resent18 network, implementing the original structure described in the Resnet paper[3], except:

1. We used input size 128x128 pixels instead of 224x224 and due to this smaller input size, we then used a kernel size of 3x3 on the first convolutional layer (instead of 7x7 as original), in order to get better results.
2. We implemented the residual block as proposed in what is commonly known as the 'ResnetV2' paper[4].

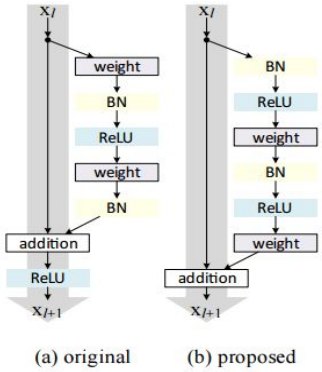


Fig. 11[4]. (a): original Residual block, (b): the residual block we used

After trying different patch sizes (**224x224**, **128x128**, **64x64**), plus different strides and patch configurations (patches with positives pixels only vs patches including petri-dish background, patches including or not, the petri-dish border, amount of pixels for a patch to be considered positive, etc.) we got the best results using the following setup to generate training patches:

Patch size: (128, 128)	**(patch width, patch height)**
Stride: (60, 60)	**(horizontal stride, vertical stride)**
Min pos. Pixels: 1250	**(min amt of positive pixels in a patch to be considered positive)**
Include dish border: False	**(False -> no petri-dish border in training patches)**
Positive pixels only: False	**(False -> include petri dish background)**

For validation, we used the following, then down-sampled to get a balanced validation set:
Validation stride = (22, 22)
Val min pos. pixels = 1024

We used a low number for the minimum amount of positive pixels for a validation patch to be considered positive, as we wanted our model to be able to correctly classify patches even containing a very small positive region. If we increase this parameter, the accuracy increases but we wouldn't be challenging our model.

Validation patches were generated from original images. This way, we test our model performance when pre-processing may not be feasible. As an example, image annotation may not be feasible on testing data, as discussed during meetings with project sponsors.

For training of the model, we used the following training setup:

- Used class weights on training loss function to balance loss, as we had imbalanced data
- **L2 regularization:** 5e-5
- **Optimizer:** RMSprop, momentum = 0.9
- **Learning rate:** 1e-3 decay on each training epoch: 0.01
- Trained 50 epochs (about 12 minutes on Google Colab's GPU)
- Reduced learning rate by 0.85 on plateau of val loss, using patience of 3 epochs
- Early stop if val accuracy didn't improve after **35 epochs**
- Save best validation accuracy model.

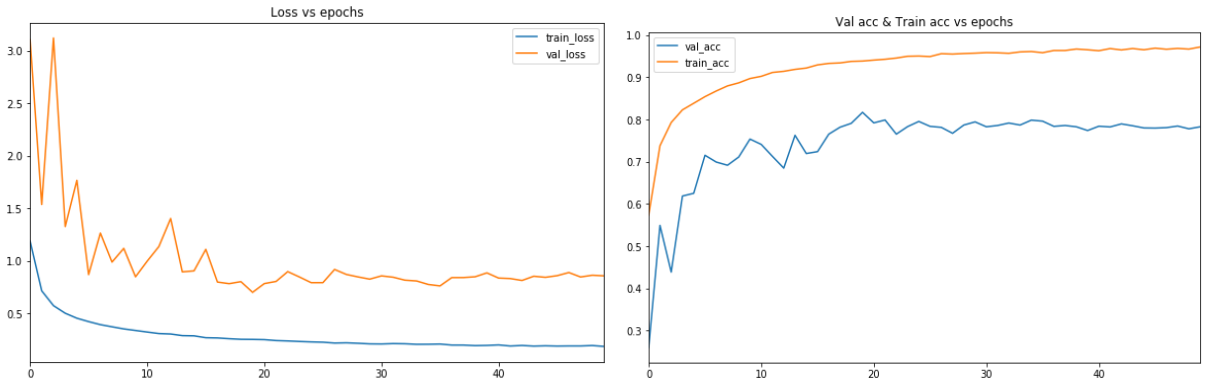


Fig. 12: Train & Validation Curves for Basic ConvNet Model on 128*128 patches

Here we got a validation accuracy of **80.7%**. Complete results are summarized as follows:

		Predicted classes									
		C1	**C2-3**	**C4-7**	**C5**	**C6**	**C8**	**C9**	**C10**	**Total / support**	**Recall**
Actual classes	**C1**	263	65	0	0	0	45	0	0	373	0.71
	C2-3	114	236	0	0	0	23	0	0	373	0.63
	C4-7	0	2	333	33	0	5	0	0	373	0.89
	C5	0	0	227	146	0	0	0	0	373	0.39
	C6	0	10	8	9	343	3	0	0	373	0.92
	C8	2	2	23	0	0	346	0	0	373	0.93
	C9	0	0	0	0	0	0	373	0	373	1.00
	C10	4	0	0	0	1	0	0	368	373	0.99
	Precision	0.69	0.75	0.56	0.78	1.00	0.82	1.00	1.00		

Fig. 13: Confusion Matrix for ResNet18 model

3. Future Work

As Next steps, we will try deeper networks (e.g. Renet34, Resnet50) and try to tune training parameters like learning rate, decay, L2 regularization, etc. We will also try to use additional data augmentation techniques like rotations and flips, which may help us to improve the results.

Unlike our initial trials at the time of the first report, we decided not to experiment further with pre-trained models. We had initially evaluated some pre-trained models like Resnet50 and VGG16, but we soon realized that model complexity resulted in strong overfitting of the training set while validation accuracy was comparable to taking a random guess. We believe the reason might be that the available pre-trained models were trained on very different sets of images, and the features they learned were not suitable for classifying bacteria species..

We are hoping to gain access to additional data in order to test our models with completely new samples. The project sponsors are currently working to gather additional data and classifying it for this project.

Fig. 14: Example of still unlabeled testing data.

References

[1] Hough Circle Transform function implemented in OpenCV library:
https://docs.opencv.org/master/d3/de5/tutorial_js_houghcircles.html
[2] Simple Thresholding function implemented in OpenCV library:
https://docs.opencv.org/master/d7/d4d/tutorial_py_thresholding.html
[3] K. He, X. Zhang, S. Ren, J. Sun. Deep Residual Learning for Image Recognition, 2015.
Available at: https://arxiv.org/abs/1512.03385
[4] K. He, X. Zhang, S. Ren, J. Sun. Identity Mappings in Deep Residual Networks, 2016.
Available at: https://arxiv.org/abs/1603.05027
[5] Hu, Ming-Kuei. "Visual pattern recognition by moment invariants." IRE transactions on information theory 8.2 (1962): 179-187.
[6] An introduction to haralick texture :
http://wiki.awf.forst.uni-goettingen.de/wiki/index.php/Haralick_Texture
[7] An tutorial on image classification:
https://gogul.dev/software/image-classification-python#feature-extraction

Contribution to the report

Akhil Punia (ap3774):

1. Experimented with multiple configurations of basic convnet architectures
2. Worked on editing and compiling report

Andres Rios (dar2196):

1. Experimented with resnet inspired network and transfer learning models
2. Contributed to pre-processing and developed patch generator module

Carlo Provinciali (cp2984):

1. Contributed to data preprocessing, information leakage and future work section
2. Contributed to report consolidation, editing and proofreading

Paridhi Singh(ps3060):

1. Contributed to data preprocessing section of the report
2. Developed initial model attempts

Xinyuan Cao (xc2461):

1. Experimented with Hard coded features for basic machine learning model
2. Contributed to future work section of the report

Zhejin Dong (zd2221):

1. Experimented with multiple configurations of basic convnet architecture
2. Developed image preprocessing and contributed to patch generator module