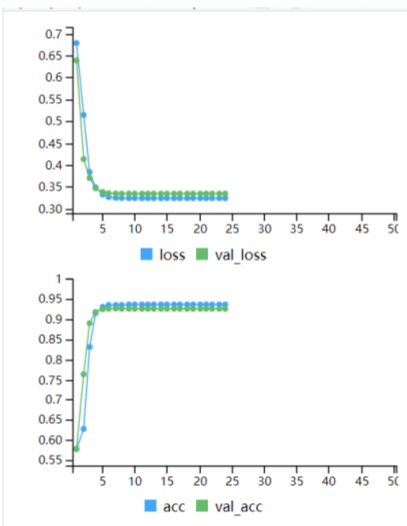
Q1
Before delving into the neural network model, we focus on preprocessing the data to achieve high accuracy—an unattainable score using the original dataset alone. We extract and sample small images from the dataset, utilizing these image snippets that potentially contain individual letters or at least a few letters. Specifically, we randomly sample some small pieces with an acceptance threshold that the small piece has more than 10% black pixels to make sure that the small piece is not whitespace. This sampling approach can be seen as a sort of bootstrapping and aggregated neural network model, i.e. bagging, which allows the neural network model to capture distinct letter shapes across various languages.

1(a)
Here, as we do not have a specific learning rate requirement, we use learning_rate_schedule_exponential_decay() and set initial_learning_rate to 0.001. Then consider following different filter numbers.

lr_initial	filter	val_loss	val_acc	test_loss	test_acc
0.001	c(8, 16)	0.6914	0.5778	0.6914	0.5774
0.001	c(16, 32)	0.5957	0.7242	0.5840	0.7480
0.001	c(8, 16, 32)	0.3534	0.8980	0.3405	0.9134
0.001	c(16, 32, 64)	0.6455	0.5784	0.6467	0.5774
0.001	c(8, 16, 32, 64)	0.6771	0.5778	0.6771	0.5774
0.001	c(16, 32, 64, 128)	0.3696	0.9032	0.3588	0.9081
0.001	c(16, 32, 64, 128, 256)	0.3342	0.9265	0.3100	0.9423
0.001	c(32, 64, 128, 256, 512)	0.5700	0.5778	0.5663	0.5774
0.001	c(16, 32, 64, 128, 256, 512)	0.3927	0.8653	0.3877	0.8556

From the above results, it is observed that filter sequences c(16, 32, 64, 128, 256) exhibit superior performance with an accuracy of 0.9423. Moving on to 1(b), we also consider filter sequences c(8, 16, 32), c(16, 32, 64, 128), and c(16, 32, 64, 128, 256, 512) as they demonstrate commendable accuracy. The following loss vs accuracy plot corresponds to the model with filter sequences c(16, 32, 64, 128, 256). There does not exist overfit and the learning rate is not high or low situation.



1(b)

lr_initial	filter	rotation	val_loss	val_acc	test_loss	test_acc
0.001	c(8, 16, 32)	TRUE	0.6605	0.5778	0.6628	0.5774
0.001	c(16, 32, 64, 128)	TRUE	0.6742	0.5778	0.6746	0.5774
0.001	c(16, 32, 64, 128, 256)	TRUE	0.6745	0.5778	0.6746	0.5774
0.001	c(16, 32, 64, 128, 256, 512)	TRUE	0.5519	0.7860	0.5365	0.7822

lr_initial	filter	flipping	loss	val_acc	test_loss	test_acc
0.001	c(8, 16, 32)	TRUE	0.6916	0.5778	0.6916	0.5774
0.001	c(16, 32, 64, 128)	TRUE	0.6352	0.5778	0.6334	0.5774
0.001	c(16, 32, 64, 128, 256)	TRUE	0.6622	0.5778	0.6630	0.5774
0.001	c(16, 32, 64, 128, 256, 512)	TRUE	0.6746	0.5778	0.6754	0.5774

Rotation and flipping do not improve the performance.

1(c)

lr_initial	filter		val_loss	val_acc	test_loss	test_acc
0.001	c(16, 32, 64, 128, 256)	batch_norm	0.0948	0.9889	0.0868	0.9948
0.001	c(16, 32, 64, 128, 256, 512)	batch_norm	0.0497	0.9918	0.0421	0.9948
0.001	c(16, 32, 64, 128, 256)	sep_conv	0.6810	0.5778	0.6811	0.5774
0.001	c(16, 32, 64, 128, 256, 512)	sep_conv	0.6810	0.5778	0.6811	0.5774
0.001	c(16, 32, 64, 128, 256)	resid_con	0.6693	0.5778	0.6692	0.5774
0.001	c(16, 32, 64, 128, 256, 512)	resid_con	0.6616	0.5778	0.6592	0.5774

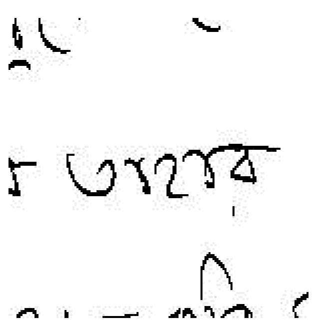
Batch normalization significantly improves the model's performance. Following that, we identified the best model with the filter sequence c(16, 32, 64, 128, 256) and incorporated batch normalization into parts d and e.

1(d)

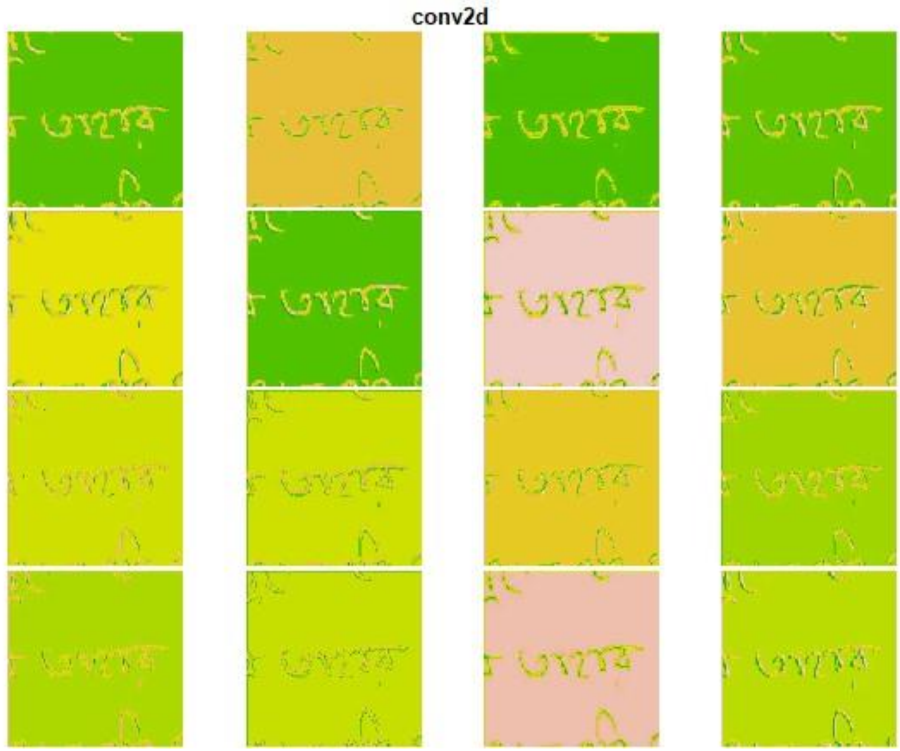
From the above result, choose a model with filter c(16,32), plus rotation to provide a visualization of the interesting activation layers. The following figure is a summary of this model.

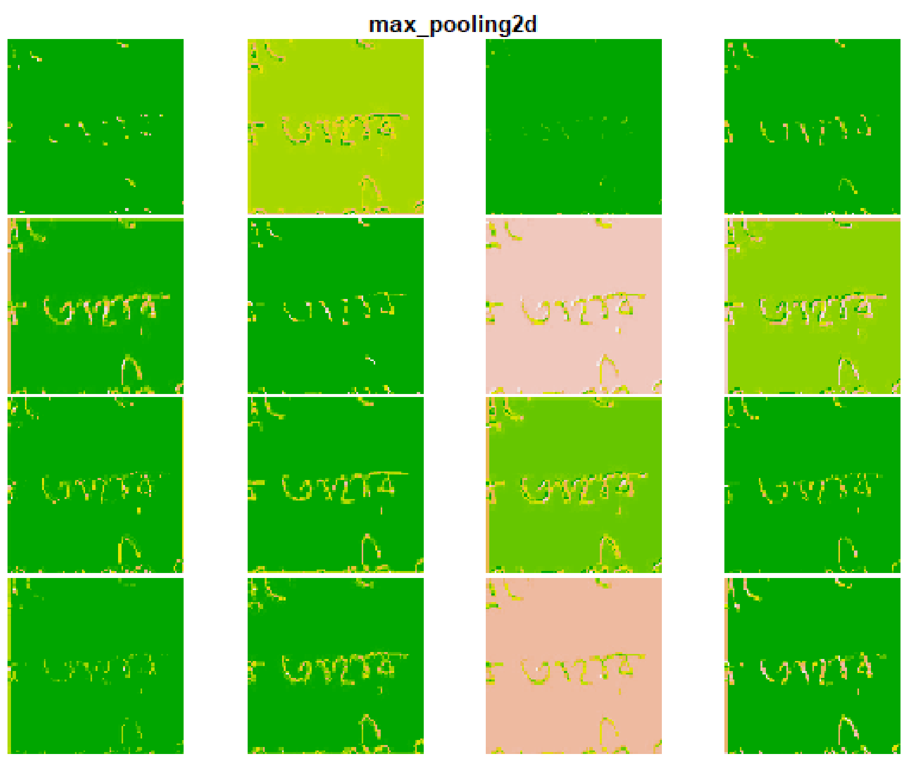
```
Model: "model"
_________________________________________________________________________________________
 Layer (type)                        Output Shape                 Param #     Trainable
=========================================================================================
 input_1 (InputLayer)                [(None, 150, 150, 1)]        0           Y
 conv2d (Conv2D)                     (None, 150, 150, 16)         144         Y
 batch_normalization (BatchNormalizat (None, 150, 150, 16)        64          Y
 ion)
 re_lu (ReLU)                        (None, 150, 150, 16)         0           Y
 max_pooling2d (MaxPooling2D)        (None, 75, 75, 16)           0           Y
 conv2d_1 (Conv2D)                   (None, 75, 75, 32)           4608        Y
 batch_normalization_1 (BatchNormaliz (None, 75, 75, 32)          128         Y
 ation)
 re_lu_1 (ReLU)                      (None, 75, 75, 32)           0           Y
 max_pooling2d_1 (MaxPooling2D)      (None, 38, 38, 32)           0           Y
 conv2d_2 (Conv2D)                   (None, 38, 38, 64)           18432       Y
 batch_normalization_2 (BatchNormaliz (None, 38, 38, 64)          256         Y
 ation)
 re_lu_2 (ReLU)                      (None, 38, 38, 64)           0           Y
 max_pooling2d_2 (MaxPooling2D)      (None, 19, 19, 64)           0           Y
 conv2d_3 (Conv2D)                   (None, 19, 19, 128)          73728       Y
 batch_normalization_3 (BatchNormaliz (None, 19, 19, 128)         512         Y
 ation)
 re_lu_3 (ReLU)                      (None, 19, 19, 128)          0           Y
 max_pooling2d_3 (MaxPooling2D)      (None, 10, 10, 128)          0           Y
 conv2d_4 (Conv2D)                   (None, 10, 10, 256)          294912      Y
 batch_normalization_4 (BatchNormaliz (None, 10, 10, 256)         1024        Y
 ation)
 re_lu_4 (ReLU)                      (None, 10, 10, 256)          0           Y
 max_pooling2d_4 (MaxPooling2D)      (None, 5, 5, 256)            0           Y
 flatten (Flatten)                   (None, 6400)                 0           Y
 dense (Dense)                       (None, 32)                   204800      Y
 batch_normalization_5 (BatchNormaliz (None, 32)                  128         Y
 ation)
 re_lu_5 (ReLU)                      (None, 32)                   0           Y
 dense_1 (Dense)                     (None, 1)                    33          Y
=========================================================================================
Total params: 598769 (2.28 MB)
Trainable params: 597713 (2.28 MB)
Non-trainable params: 1056 (4.12 KB)
_________________________________________________________________________________________
```

Displaying the test picture:

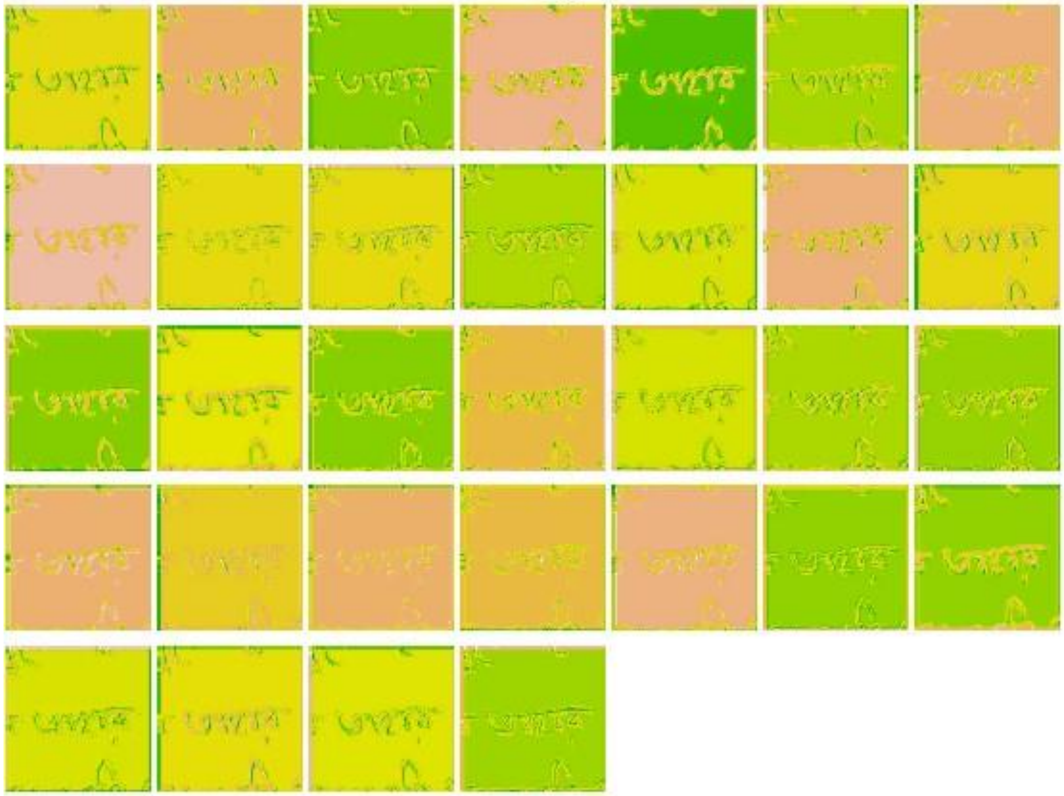


Visualizing every channel in every intermediate activation:
the features extracted by a layer become increasingly abstract with the depth of the layer. The activations of higher layers carry less and less information about the specific input being seen and more and more information about the target, which implies that our model is good.





conv2d_1



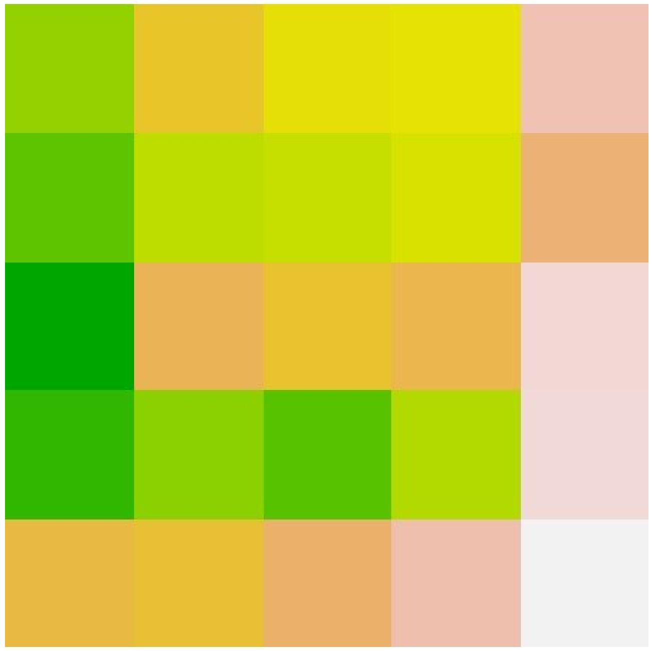
max_pooling2d_1

conv2d_4

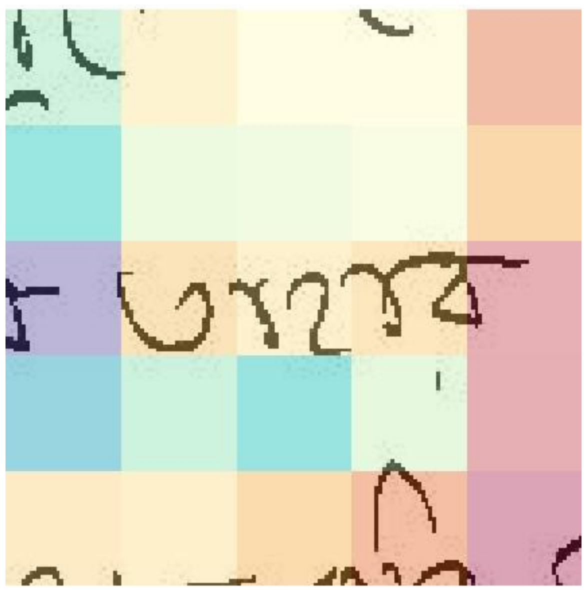
max_pooling2d_4

1(e)

Visualizing the Probability of Activations:



Visualizing Activations with Context:



Q2

2(a)

Here, as we do not have a specific learning rate requirement, we use learning_rate_schedule_exponential_decay() and set initial_learning_rate to 0.001. Then consider following different filter numbers.

lr_initial	filter	val_loss	val_acc	test_loss	test_acc
0.001	c(8, 16)	0.3078	0.8929	0.3195	0.8750
0.001	c(16, 32)	0.3207	0.9127	0.3075	0.9286
0.001	c(8, 16, 32)	0.3532	0.8770	0.3890	0.8214
0.001	c(16, 32, 64)	0.5048	0.8135	0.5017	0.8750
0.001	c(8, 16, 32, 64)	0.3330	0.8611	0.4089	0.8036
0.001	c(16, 32, 64, 128)	0.1675	0.9365	0.2483	0.9464
0.001	c(32, 64, 128, 256)	0.2123	0.9167	0.2645	0.9464
0.001	c(8, 16, 32, 64, 128)	0.2863	0.8929	0.3229	0.9107
0.001	c(8, 16, 32, 64, 128, 256)	0.2656	0.8929	0.3304	0.8929

From the above results, it is observed that filter sequences c(16, 32, 64, 128) and c(32, 64, 128, 256) exhibit superior performance with an accuracy of 0.9464. Moving on to 2(b), we also consider filter sequences c(16, 32) and c(8, 16, 32, 64, 128) as they demonstrate commendable accuracy.

2(b)

lr_initial	filter	rotation	val_loss	val_acc	test_loss	test_acc
0.001	c(16, 32)	T	0.2531	0.9286	0.2007	0.9643
0.001	c(16, 32, 64, 128)	T	0.3518	0.8810	0.2991	0.9107
0.001	c(32, 64, 128, 256)	T	0.4650	0.8571	0.4813	0.8214
0.001	c(8, 16, 32, 64, 128)	T	0.2729	0.9286	0.2159	0.9643

Here, factor = 0.2 for the rotation parameter. And rotation improve the performance for model with filter sequence c(16, 32) and c(8, 16, 32, 64, 128).

lr_initial	filter	flipping	val_loss	val_acc	test_los	test_ac
0.001	c(16, 32)	T	0.3506	0.8452	0.5536	0.7143
0.001	c(16, 32, 64, 128)	T	0.5232	0.7381	0.5771	0.6786
0.001	c(32, 64, 128, 256)	T	0.2013	0.9444	0.3463	0.8214
0.001	c(8, 16, 32, 64, 128)	T	0.3652	0.8730	0.4586	0.8214

Flipping does not improve the performance.

2(c)

lr_initial	filter		val_loss	val_acc	test_loss	test_acc
0.001	c(16, 32)	batch_norm	1.0796	0.4960	0.6892	0.5357
0.001	c(16, 32, 64, 128)	batch_norm	0.8139	0.5040	0.6911	0.4643
0.001	c(32, 64, 128, 256)	batch_norm	0.6849	0.6508	0.6694	0.3571
0.001	c(8, 16, 32, 64, 128)	batch_norm	1.1362	0.5040	0.7172	0.4643
0.001	c(16, 32)	sep_conv	0.3154	0.9444	0.2650	0.9464
0.001	c(16, 32, 64, 128)	sep_conv	0.6948	0.5040	0.6965	0.4643
0.001	c(32, 64, 128, 256)	sep_conv	0.6939	0.5040	0.6937	0.4643
0.001	c(8, 16, 32, 64, 128)	sep_conv	0.6931	0.5040	0.6938	0.4643
0.001	c(16, 32)	resid_conn	0.6021	0.7302	0.5768	0.7143
0.001	c(16, 32, 64, 128)	resid_conn	0.4415	0.8056	0.3955	0.8571
0.001	c(32, 64, 128, 256)	resid_conn	0.1950	0.9524	0.1437	0.9643
0.001	c(8, 16, 32, 64, 128)	resid_conn	0.4551	0.8294	0.4196	0.8571

The skip layers and residual connections, batch/layer normalization, and separable convolutions do not improve the performance in this situation.

2(d)

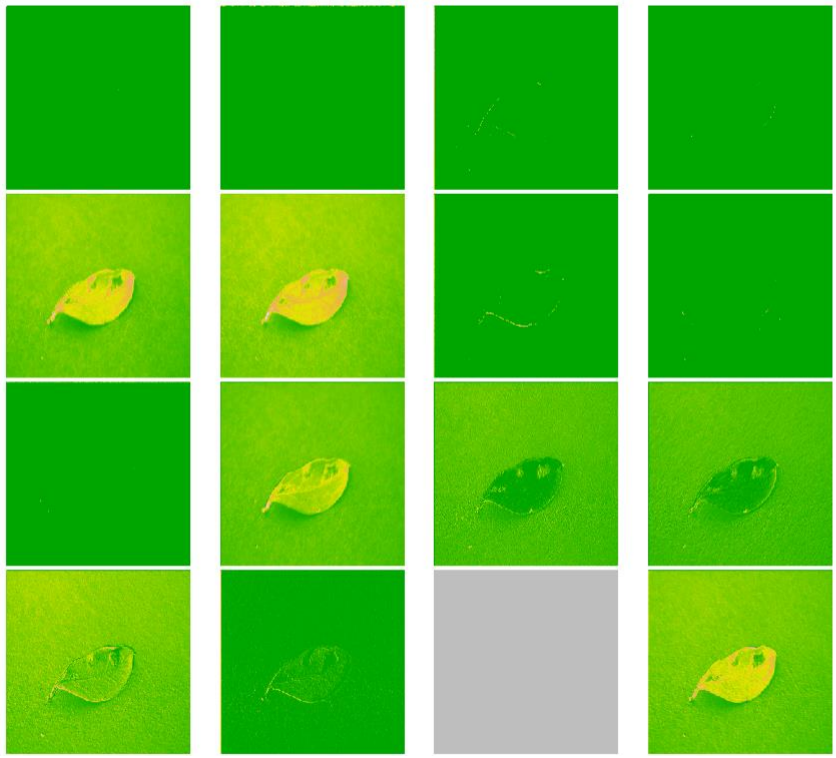
From the above result, choose a model with filter c(16,32), plus rotation to Provide a visualization of the interesting activation layers. The following figure is a summary of this model.

```
Model: "model"
_________________________________________________________________________________________
 Layer (type)                       Output Shape                 Param #      Trainable
=========================================================================================
 input_1 (InputLayer)               [(None, 256, 256, 3)]        0            Y
 random_rotation (RandomRotation)   (None, 256, 256, 3)          0            Y
 conv2d (Conv2D)                    (None, 256, 256, 16)         448          Y
 max_pooling2d (MaxPooling2D)       (None, 128, 128, 16)         0            Y
 conv2d_1 (Conv2D)                  (None, 128, 128, 32)         4640         Y
 max_pooling2d_1 (MaxPooling2D)     (None, 64, 64, 32)           0            Y
 flatten (Flatten)                  (None, 131072)               0            Y
 dense (Dense)                      (None, 32)                   4194336      Y
 dense_1 (Dense)                    (None, 1)                    33           Y
=========================================================================================
Total params: 4199457 (16.02 MB)
Trainable params: 4199457 (16.02 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________________________________
```

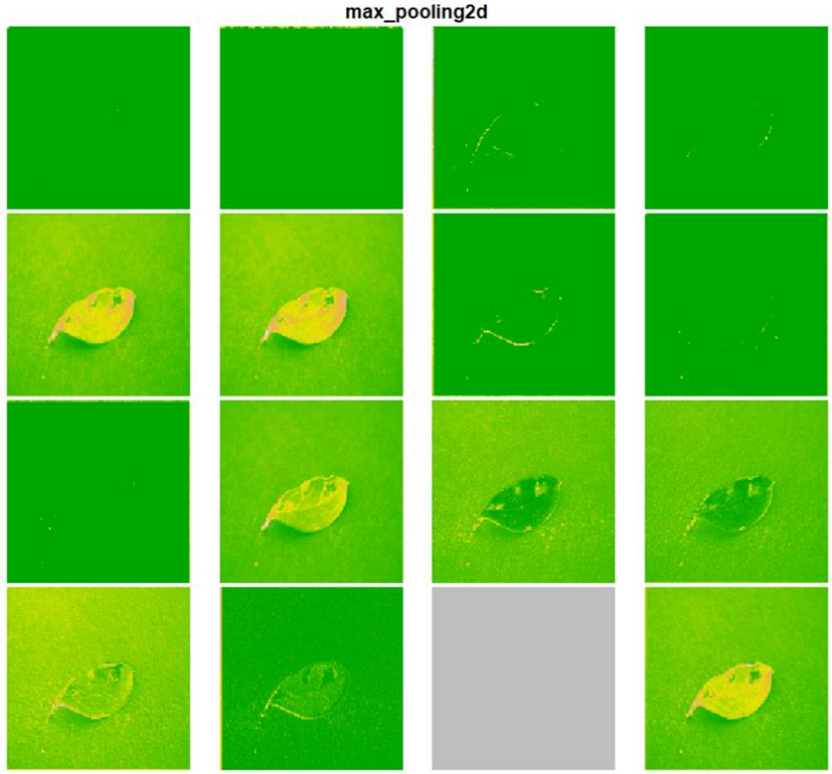
Displaying the test picture:

Visualizing every channel in every intermediate activation:

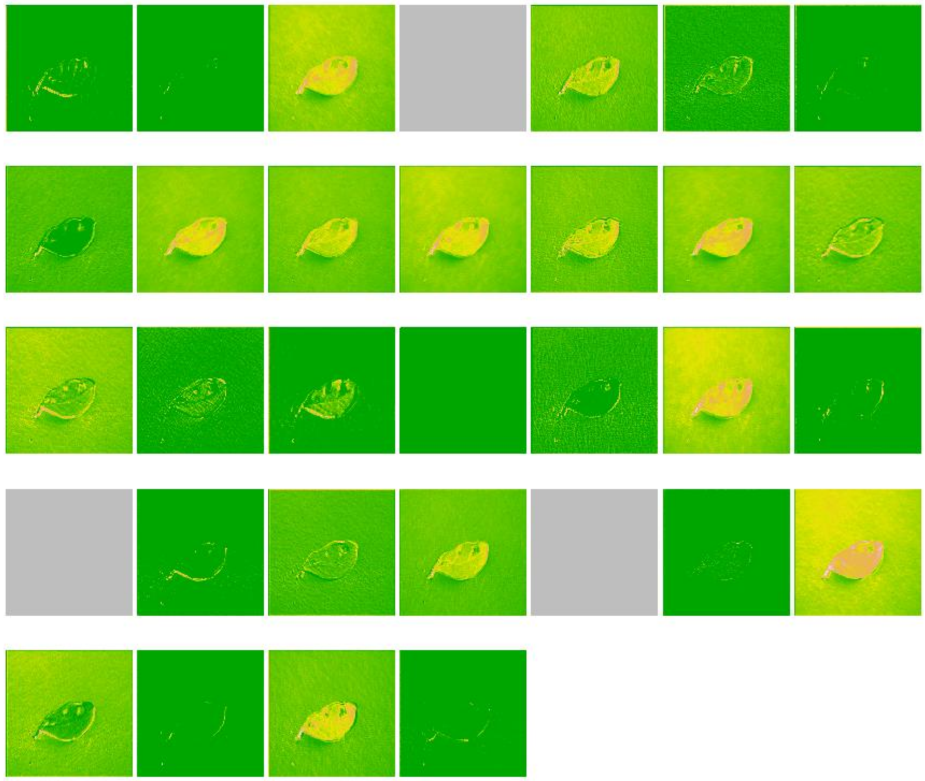
conv2d



max_pooling2d



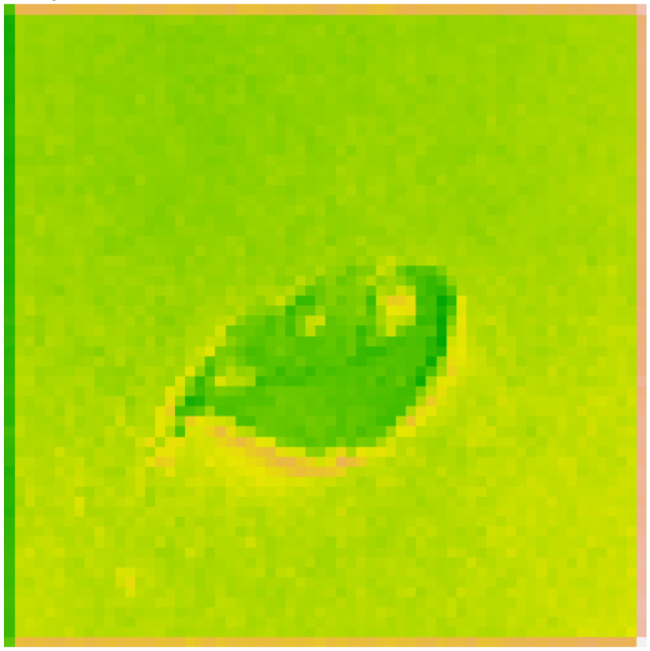
conv2d_1



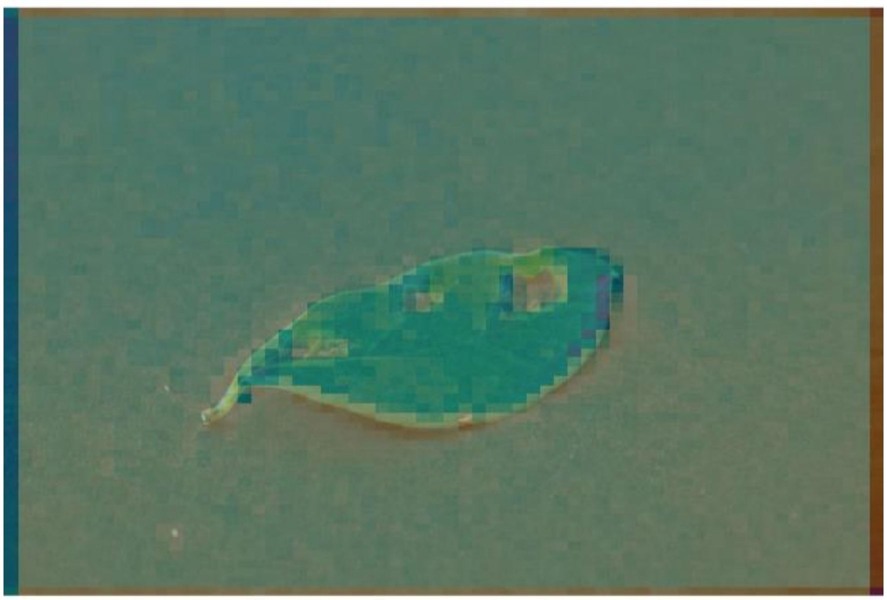
max_pooling2d_1

2(e)

Visualizing the Probability of Activations:



Visualizing Activations with Context:



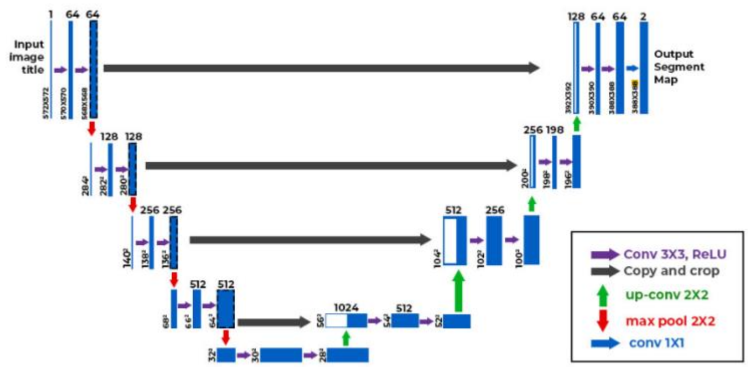
Here, it's interesting to note that the black spots on the diseased tree leaves are strongly activated. this is probably how the network can tell the difference between disease and no disease.

Q3

3(a)

Model Overview

- We utilized U-Net for the segmentation of neither (background), lung, or airway.
- Input: 128 x 128 x 1 input images (rescaled and greyscale).
    - Trained on 26*200 images (sampling 200 slices from each CT)
- Model: U-Net Architecture



- Settings:
    - Adam optimizer, learning rate = 0.001
    - Categorical cross-entropy with 3 categories (neither/lung/airway)
    - Accuracy metric
    - 10 epochs, 32 batch size

Model Training Loss/Accuracy

- Loss: 0.0345
- Accuracy: 0.9877

Model Validation Loss/Accuracy

- Loss: 0.0385
- Accuracy: 0.9862

Model Test Loss/Accuracy (20% split)

- Loss: 0.0369
- Accuracy: 0.9866

3(b)

Image 27 Loss/Accuracy

- Loss: 0.0495
- Accuracy: 0.9817

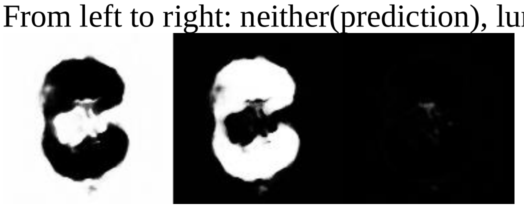
Confusion Matrix

			Ground Truth	
		Neither	Lung	Airway
	Neither	2830427	46265	562
Prediction	Lung	10528	383138	3
	Airway	1217	1469	3191

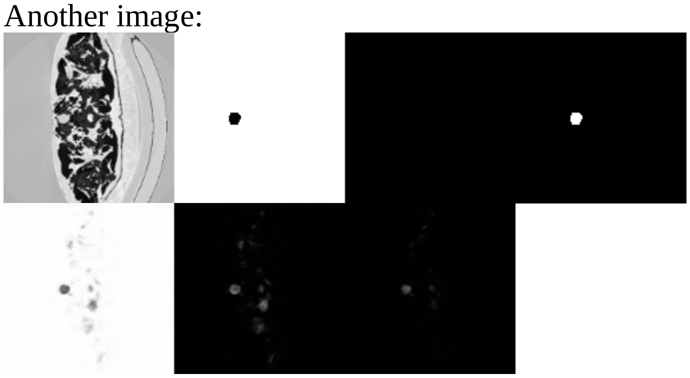
Slices of the images:

From left to right: x(data), neither(truth), lung(truth), airway(truth)

From left to right: neither(prediction), lung(prediction), airway(prediction)



Another image:



The U-Net model provided good results: high accuracy across training, validation, and test datasets. However, because of the imbalanceness of our data, classifying them as neither or lung achieves a high accuracy score. From the confusion matrix, we noticed that the prediction accuracy for the airway is not consistent with the overall 98% accuracy.

To address this issue and for future investigations, it would be nice to use alternative metrics, such as the multiclass-F1 score, and also put weights to minor categories (airway in our case) to provide more accurate results.