Breast Cancer Detection

GROUP NO. 3

1. Minita Joshee [23]
2. Amaan Nizam [46]
3. Abhiram Pillai [52]
4. Anne Pinto [53]

PROJECT GUIDE : DR. SATISHKUMAR CHAVAN

DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION
DON BOSCO INSTITUTE OF TECHNOLOGY

Objectives

- To classify the tumors as Benign and Malignant tumors.
- To detect location of the tumour.
- To automate the process of detection of abnormal tissues.
- To evaluate the performance of various Deep Learning approaches for detection and segmentation of tumours

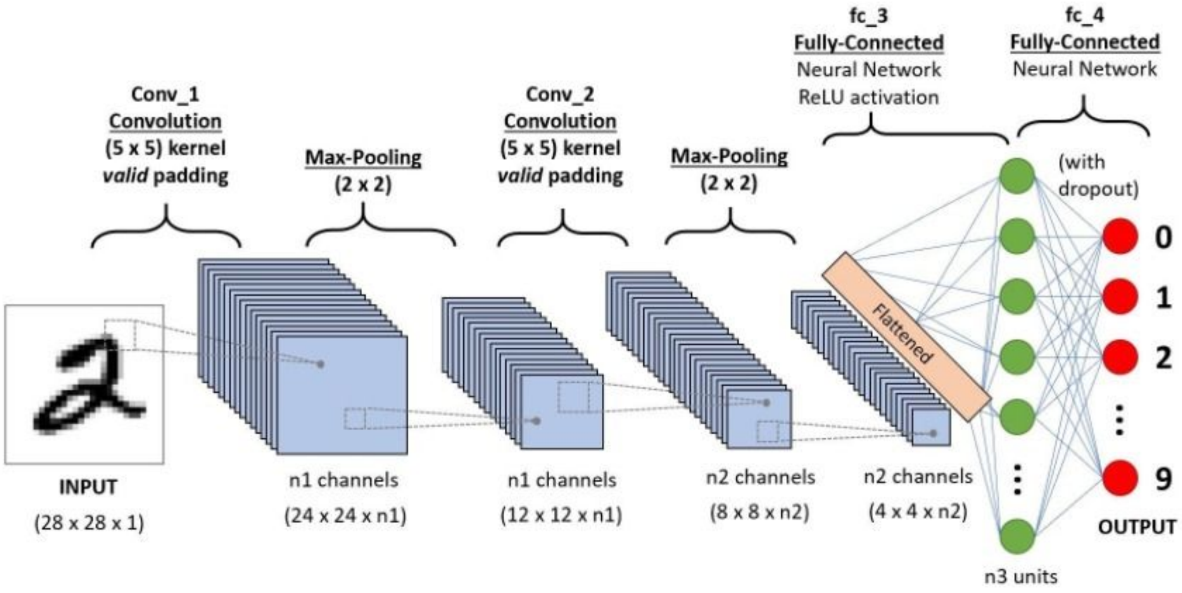
Outcomes

- Software tool which automatically detects cancerous tissues.
- Software will be able to identify location of tumour and detect volume of tumour.
- It will act an an assisting tool to radiologists to classify or choose the abnormal mammogram and prioritise based on level of concern.
- Learning how to write a Technical paper using LaTeX

Problem Statement

To develop an automated detection and segmentation of tumours using mammogram in Cranial-Caudal and Medial-lateral oblique (CC and MLO) views using Deep Learning Techniques

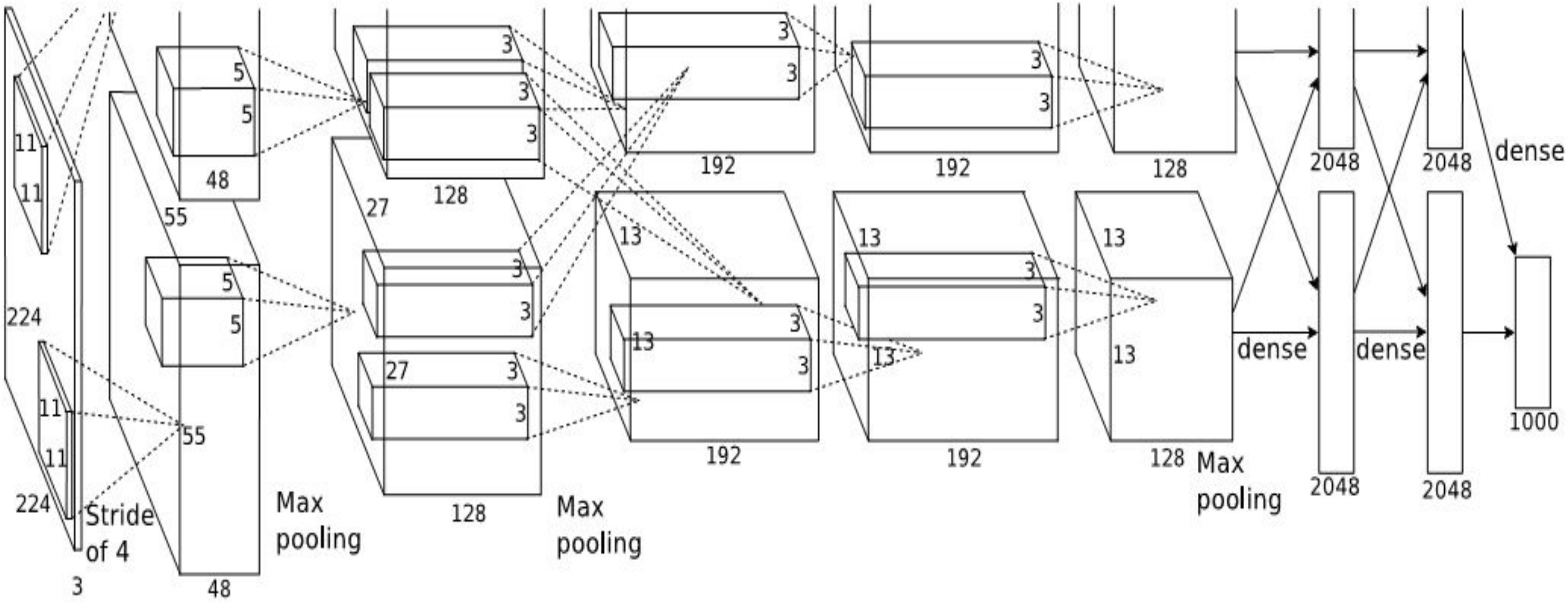
CNN


Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
(with
dropout)
Flattened
0
1
2
9
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units
OUTPUT

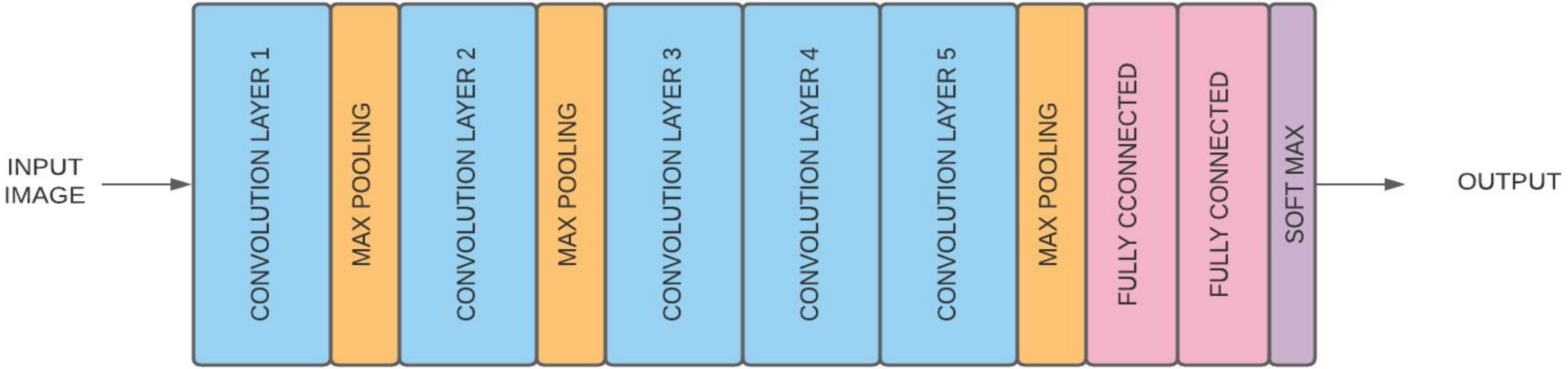
Training Using Alexnet

- 8 layers deep neural network
- A total of 60 million parameters
- 5 Convolutional layers and 3 Fully Connected (dense) layers
- Dataset consisting of 15 million images,over 22,000 categories with variable resolution images so scaled down to fixed resolution 256x256
- Why ReLu Non-linearity?

Architecture of Alexnet



Architecture



Results of Alexnet

+ Code + Text RAM Disk Editing

```
steps_per_epoch=int(8000/32),
epochs=15,
validation_data=test_set,
validation_steps=200,
#callbacks=callback
)
```

```
Found 20000 images belonging to 2 classes.
Found 5000 images belonging to 2 classes.
Epoch 1/5
/usr/local/lib/python3.6/dist-packages/keras_preprocessing/image/image_data_generator.py:720: UserWarning: This ImageDataGenerator specifies `featurewi
  warnings.warn('This ImageDataGenerator specifies '
/usr/local/lib/python3.6/dist-packages/keras_preprocessing/image/image_data_generator.py:728: UserWarning: This ImageDataGenerator specifies `featurewi
  warnings.warn('This ImageDataGenerator specifies '
250/250 [==============================] - 109s 437ms/step - loss: 0.7304 - accuracy: 0.4997 - val_loss: 0.6961 - val_accuracy: 0.5000
Epoch 2/5
250/250 [==============================] - 106s 425ms/step - loss: 0.7112 - accuracy: 0.5010 - val_loss: 0.6937 - val_accuracy: 0.4994
Epoch 3/5
250/250 [==============================] - 108s 430ms/step - loss: 0.7098 - accuracy: 0.5027 - val_loss: 0.6944 - val_accuracy: 0.4991
Epoch 4/5
250/250 [==============================] - 108s 431ms/step - loss: 0.7147 - accuracy: 0.4934 - val_loss: 0.6932 - val_accuracy: 0.5081
Epoch 5/5
250/250 [==============================] - 108s 431ms/step - loss: 0.7129 - accuracy: 0.4993 - val_loss: 0.6949 - val_accuracy: 0.4969
```

```
[ ] import os
    model_name = 'DogCat.h5'
    save_dir = 'pretrained_model' # pretrained_model # pred_model
    # Save model and weights
    if not os.path.isdir(save_dir):
        os.makedirs(save_dir)
```

ENG 07:50:29 26-10-2020 15

Training using VGG-16

- VGGNet consists of 16 convolutional layers with only 3x3 kernels.
- **Input**: VGG takes in a 224x224 pixel RGB image.
- **Convolutional Layers**: The convolutional layers in VGG use a very small receptive field (3x3, the smallest possible size that still captures left/right and up/down).
- **Fully-Connected Layers**: VGG has three fully-connected layers: 4096 channels each and the last one has 1000 channels
- **Hidden Layers**: All of VGG's hidden layers use ReLU (a huge innovation from AlexNet that cut training time).

Architecture

Results of VGG-16

```
Epoch 1/10
100/100 [==============================] - 33s 333ms/step - loss: 0.6756 - acc: 0.6855 - val_loss: 0.3504 - val_acc: 0.8335
Epoch 2/10
100/100 [==============================] - 33s 332ms/step - loss: 0.4837 - acc: 0.7655 - val_loss: 0.4827 - val_acc: 0.7800
Epoch 3/10
100/100 [==============================] - 33s 328ms/step - loss: 0.4147 - acc: 0.8110 - val_loss: 0.2349 - val_acc: 0.8980
Epoch 4/10
100/100 [==============================] - 33s 330ms/step - loss: 0.4037 - acc: 0.8160 - val_loss: 0.2329 - val_acc: 0.9030
Epoch 5/10
100/100 [==============================] - 33s 327ms/step - loss: 0.4103 - acc: 0.8205 - val_loss: 0.2368 - val_acc: 0.8960
Epoch 6/10
100/100 [==============================] - 33s 326ms/step - loss: 0.3732 - acc: 0.8255 - val_loss: 0.2103 - val_acc: 0.9070
Epoch 7/10
100/100 [==============================] - 33s 332ms/step - loss: 0.3564 - acc: 0.8420 - val_loss: 0.2249 - val_acc: 0.9030
Epoch 8/10
100/100 [==============================] - 33s 329ms/step - loss: 0.3532 - acc: 0.8455 - val_loss: 0.2329 - val_acc: 0.9050
Epoch 9/10
100/100 [==============================] - 33s 329ms/step - loss: 0.3583 - acc: 0.8440 - val_loss: 0.2276 - val_acc: 0.9015
Epoch 10/10
100/100 [==============================] - 33s 331ms/step - loss: 0.3418 - acc: 0.8475 - val_loss: 0.2320 - val_acc: 0.9045
```

Training using GoogleNet

- 22 layers deep.
- The architecture was designed to keep computational efficiency in mind. The idea behind that the architecture can be run on individual devices even with low computational resources.
- Uses many different methods such as 1x1 convolution and global average pooling that enables it to create a deeper architecture.

Architecture

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

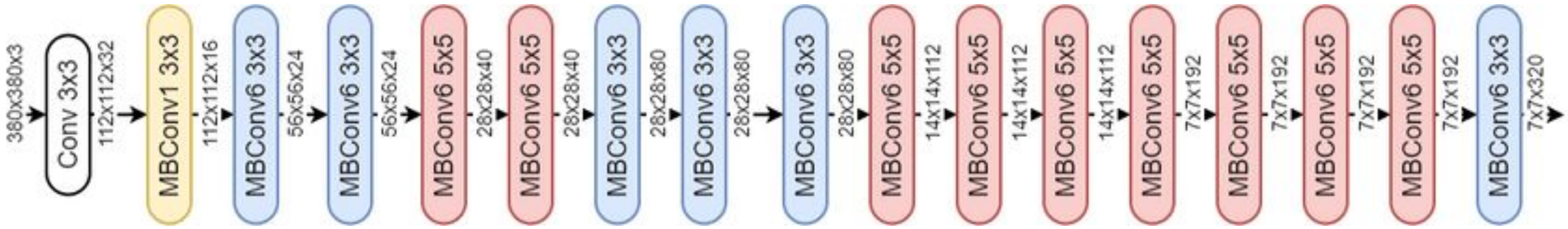
Results of GoogleNet

```
Epoch 1/10
100/100 [==============================] - 28s 278ms/step - loss: 1.1381 - acc: 0.8415 - val_loss: 0.1191 - val_acc: 0.9595
Epoch 2/10
100/100 [==============================] - 26s 256ms/step - loss: 0.4630 - acc: 0.8785 - val_loss: 0.3551 - val_acc: 0.9185
Epoch 3/10
100/100 [==============================] - 27s 267ms/step - loss: 0.3491 - acc: 0.9020 - val_loss: 0.1194 - val_acc: 0.9565
Epoch 4/10
100/100 [==============================] - 26s 256ms/step - loss: 0.3000 - acc: 0.9140 - val_loss: 0.1213 - val_acc: 0.9640
Epoch 5/10
100/100 [==============================] - 26s 265ms/step - loss: 0.3183 - acc: 0.9070 - val_loss: 0.1598 - val_acc: 0.9550
Epoch 6/10
100/100 [==============================] - 26s 256ms/step - loss: 0.3174 - acc: 0.9205 - val_loss: 0.1801 - val_acc: 0.9575
Epoch 7/10
100/100 [==============================] - 27s 266ms/step - loss: 0.3334 - acc: 0.9105 - val_loss: 0.1175 - val_acc: 0.9580
Epoch 8/10
100/100 [==============================] - 26s 257ms/step - loss: 0.3147 - acc: 0.9080 - val_loss: 0.2037 - val_acc: 0.9440
Epoch 9/10
100/100 [==============================] - 26s 263ms/step - loss: 0.2794 - acc: 0.9185 - val_loss: 0.2248 - val_acc: 0.9445
Epoch 10/10
100/100 [==============================] - 26s 256ms/step - loss: 0.3136 - acc: 0.9145 - val_loss: 0.1320 - val_acc: 0.9620
```

EfficientNet

- Alexnet ⟶ VGG-16⟶ Googlenet.
- Process of scaling up CNNs is not well understood; many ways of doing this, mostly arbitrarily chosen.
- Three types of scaling.- width,depth,resolution.
- Efficientnet uses compound scaling.
- Saturation happens if only one type of scaling is done.

EfficientNet



EfficientNet

```
Epoch 1/10
100/100 [==============================] - 32s 324ms/step - loss: 0.3151 - accuracy: 0.9055 - val_loss: 0.0549 - val_accuracy: 0.9670
Epoch 2/10
100/100 [==============================] - 30s 304ms/step - loss: 0.3398 - accuracy: 0.9160 - val_loss: 0.0211 - val_accuracy: 0.9670
Epoch 3/10
100/100 [==============================] - 30s 303ms/step - loss: 0.4132 - accuracy: 0.9070 - val_loss: 0.0252 - val_accuracy: 0.9780
Epoch 4/10
100/100 [==============================] - 30s 303ms/step - loss: 0.3304 - accuracy: 0.9170 - val_loss: 0.5167 - val_accuracy: 0.9810
Epoch 5/10
100/100 [==============================] - 31s 309ms/step - loss: 0.3719 - accuracy: 0.9195 - val_loss: 0.1717 - val_accuracy: 0.9780
Epoch 6/10
100/100 [==============================] - 30s 299ms/step - loss: 0.3617 - accuracy: 0.9145 - val_loss: 3.9468e-06 - val_accuracy: 0.9760
Epoch 7/10
100/100 [==============================] - 31s 307ms/step - loss: 0.3225 - accuracy: 0.9245 - val_loss: 0.0853 - val_accuracy: 0.9710
Epoch 8/10
100/100 [==============================] - 30s 299ms/step - loss: 0.3143 - accuracy: 0.9265 - val_loss: 1.0649e-05 - val_accuracy: 0.9780
Epoch 9/10
100/100 [==============================] - 30s 297ms/step - loss: 0.3283 - accuracy: 0.9225 - val_loss: 0.3379 - val_accuracy: 0.9570
Epoch 10/10
100/100 [==============================] - 30s 298ms/step - loss: 0.3275 - accuracy: 0.9220 - val_loss: 0.4117 - val_accuracy: 0.9790
```

Comparing The Four Models

Name of Model	Number of Layers	Accuracy (%)	Validation Accuracy (%)	Loss (%)	Validation Loss (%)
Alex Net	8	49.93	49.69	71.29	69.49
VGG-16	16	84.75	90.45	34.18	23.2
Efficient Net	17	92.9	97.9	32.75	41.17
Google Net	22	91.45	96.2	31.36	13.2

Plan For Remaining Work

- Working on the MIAS dataset with the basic CNN and then proceeding with other models also.
- Trying to increase the accuracy and lower the losses for the MIAS dataset.

References

[1] Krizhevsky Alex, Sutskever, I., H. Geoffrey E.,"Alex Net," Advance Neural Inf Process. Syst, vol. 25, pp.1-9, 2012.

[2] J. Fieres, J. Schemmel, and K. Meier, "Training convolutional networks of threshold neurons suited for low-power hardware implementation," In Neural Networks IJCNN'06. International Joint Conference on IEEE, pp. 21-28, 2012.

[2] M. Lin, Q. Chen, and S. Yan, "Network in network," vol.1312, pp.4400, 2013.

[3] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell, "An overview of machine learning. In Machine learning." Springer Berlin Heidelberg , vol.3, pp.23, 2013.

[4] C. Nebauer, "Evaluation of convolutional neural networks for visual recognition," IEEE Transactions on Neural Networks, vol.4, pp. 685-696, 2014.

[5] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed ,Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, Google Inc. University of North Carolina, Chapel Hill, University of Michigan, Ann Arbor Magic Leap Inc. , "Going Deeper with Convolutions", 2014.

References

[6] Karen Simonyan & Andrew Zisserman Visual Geometry Group, Department of Engineering Science, University of Oxford, “Very Deep Convolutional Networks For Large-Scale Image Recognition”, 2014.

[7] S. C. Dharmadhikari, M. Ingle, and P. Kulkarni, “Empirical studies on machine learning based text classification algorithms.” Advanced Computing 2, vol.6, pp.161, 2015.

[7] S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," 2017 International Conference on Engineering and Technology (ICET), Antalya, pp. 1-6, 2017.

[10]Tan, Mingxing, and Quoc V. Le. "Efficientnet: Rethinking model scaling for convolutional neural networks.", vol.1905, pp.11946, 2019.