

BIRLA INSTITUTE OF TECHNOLOGY AND
SCIENCE PILANI



STUDY-ORIENTED PROJECT

A REPORT

On

**Cervical Cancer Detection Using Generative
Adversarial Network**

Submitted To

Dr. L Rajya Lakshmi

Submitted By:

Archi Jain

2020B1A71380P

ACKNOWLEDGEMENT

I would like to express my gratitude to Prof. Rajya L Lakshmi, who allowed me to engage in this project and generously shared their profound insights, suggestions, and patient responses to my inquiries.

I must also convey my profound appreciation to the Computer Science Department, whose guidance, suggestions, and practical counsel have proven invaluable. Their wealth of knowledge and extensive experience have greatly contributed to my professional growth.

Additionally, I extend my thanks to the Birla Institute of Technology and Science, Pilani for facilitating this Study Oriented Project. This opportunity has afforded me the privilege of acquiring practical `experience and has consistently furnished students with this invaluable exposure over the years.

TABLE OF CONTENTS

INTRODUCTION.....	4
MODELS.....	5
Based on region.....	5
Based on regression.....	6
CERVICAL CANCER DETECTION.....	7
DATASET.....	8
GENERATIVE ADVERSARIAL NETWORK	9
CLASSIFIER.....	11
CONCLUSION AND RECCOMENDATION.....	14
FUTURE SCOPE.....	15
IMPORTANT LINKS.....	15
REFERENCES.....	16

INTRODUCTION

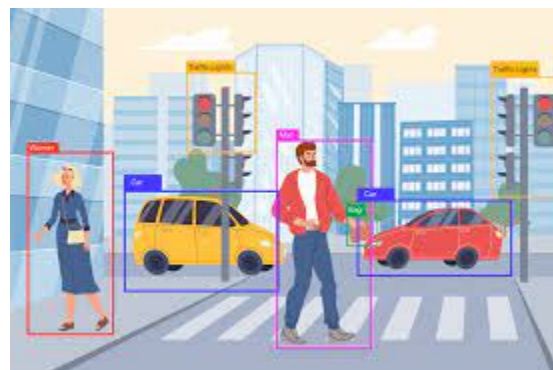
A crucial problem in computer vision is object detection, which involves locating and identifying things inside an image or a video frame. It is a fundamental part of many applications, including augmented reality, medical imaging, surveillance systems, and autonomous vehicles. Object detection has seen a revolution with the introduction of machine learning, especially deep learning, which has allowed for impressive improvements in efficiency and accuracy. In 2014, Ian Goodfellow and associates proposed a type of artificial intelligence system called Generative Adversarial Networks, or GANs. The generator and discriminator neural networks, which make up GANs, are trained concurrently via a competitive process.

The discriminator's job is to distinguish between actual and fake data, whereas the generator's goal is to make data, usually photos, that are indistinguishable from real data samples. The discriminator assesses fictitious data samples produced by the generator during training, which are created from random noise. Both networks repeatedly get better through this adversarial process: the discriminator gets better at separating real data from fake, and the generator learns to produce more convincing data.

Across a wide range of applications, such as data augmentation, style transfer, and picture production, GANs have shown impressive results. Furthermore, GANs have demonstrated success in industries like healthcare, where they can produce artificial medical pictures to train diagnostic models.

But GAN training can be difficult and unstable; it frequently experiences vanishing gradients, which impede learning, and mode collapse, in which the generator generates a restricted range of samples.

Despite these difficulties, GANs are a major development in generative modeling and have spurred a lot of interest and study in the artificial intelligence community. Their capacity to produce accurate data with little assistance from humans has great potential for a variety of real-world uses and is driving continuous advancements in machine learning.



MODELS BASED ON REGION PROPOSAL

- **R-CNN:**

- The principle of R-CNN is that it utilizes the region segmentation method of selective search to extract the region proposals in the image.
- The feature vectors will be classified using the classifier SVM to determine the classification outcomes for each region proposal.
- The model outputs exact object classifications and object bounding boxes.

- **SPP-net:**

- It is a deep neural network based on spatial pyramid pooling.
- The crop/warp procedure on the input image in the previous method can be eliminated by using the spatial pyramid pooling layer.
- Additionally, it allows input images of various sizes to flow through the convolution layer and link to the full connection layer using a feature vector of the same dimension.
- Solves the problems of object image incompleteness and deformation.

- **Fast R-CNN:**

- Unlike R-CNN, Fast R-CNN extracts the region proposal from the input image using a selective search technique and then uses ROI pooling to execute pooling on the mapped region proposal of the feature layer.
- The role of ROI pooling is just like the spatial pyramid pooling of SPP-net.

- **Faster R-CNN:**

- The region proposal network (RPN) is utilized by Faster R-CNN to address the problems of excessive processing and subpar real-time resulting from the selective search technique employed by R-CNN and Fast R-CNN.
- The technique known as the anchoring mechanism allows RPN to split the feature layer into $n \times n$ areas and produce feature regions that are centered on the region at different scales and aspect ratios.

- **R-FCN:**

- Is a full convolution neural network based on regions, having solved the problem that RoI can't share the computation.
- With the position-sensitive score maps ($k \times k \times (C+1)$ dimensional Convolutional Network).
- R-FCN could do the recognition and location simultaneously to achieve object detection.

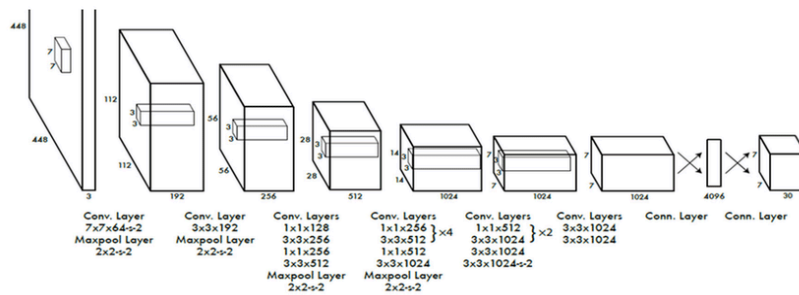
MODELS BASED ON REGRESSION

- **YOLO:**

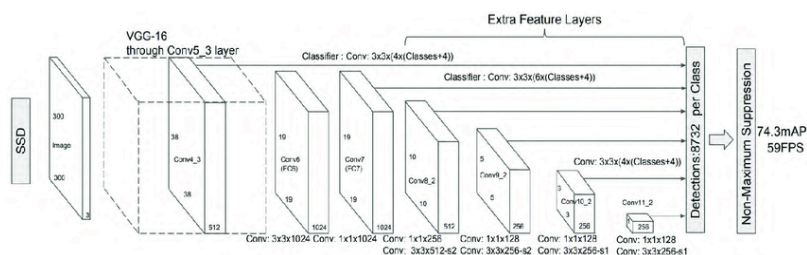
- Is a convolution neural network for real-time object detection and can accomplish end-to-end training.
- Because of the cancellation of the RoI module, YOLO won't extract the object region proposal anymore.
- YOLO divides the input image scale into 7 7 grids, each of which will produce two bounding boxes.
- To obtain the detection results, YOLO sets a threshold, filters the object proposals with low confidence, and removes the redundant object proposals.

- **SSD:**

- The design of SSD has integrated YOLO's regression idea and Faster R-CNN's anchors mechanism.
- With the anchors mechanism, SSD can extract the features of different scales and aspect ratios to guarantee detection accuracy.
- The local feature extraction method of SSD is more reasonable and effective compared with the general feature extraction method of YOLO.
- One of the disadvantages is its weak detection capacity for small objects.



YOLO



SSD

CERVICAL CANCER DETECTION

When utilizing GANs to diagnose cervical cancer, the procedure usually entails training a convolutional neural network (CNN) or other machine learning model on a dataset that contains both artificially created GAN images and actual cervical images. By extracting features from the images, the model learns to distinguish between cervical tissues that belong to different types of cancers. The dataset is enhanced by GAN-generated images, which give the model more examples to learn from and may help it generalize to previously undiscovered data. The trained algorithm evaluates fresh cervical pictures during inference and generates predictions about the existence or lack of malignant abnormalities.

This can be done with the help of the following steps:

- **Data Generation:** Generating artificial medical images that closely mimic actual cervical scans is possible with GANs. This is especially helpful when access to huge datasets is restricted due to privacy concerns or when there is a dearth of labeled data. The discriminator network evaluates the realism of the artificial images of the cervix produced by the generator network of the GAN.
- **Data Augmentation:** By producing more realistic samples, GANs can enhance already-existing datasets. This aids in the development of more reliable machine-learning models for the identification of cervical cancer. GAN-generated images can introduce differences in illumination, location, or tissue properties, which can improve the model's generalization capacity and increase the dataset's diversity.
- **Enhancing Diagnostic Accuracy:** Machine learning models that are tasked with identifying cervical cancer from cervical pictures can benefit from the addition of GAN-generated images to their training sets. The model may be able to distinguish between the different types of cancerous cervical tissues more accurately through training on a blend of actual and synthetic pictures, which could enhance diagnosis accuracy.
- **Privacy Preservation:** GANs provide a solution by creating synthetic data that maintains the essential features of authentic cervical pictures while eliminating identifiable information, which is useful in situations when acquiring genuine patient data is difficult because of privacy laws or ethical considerations. Because of this, researchers can create and assess detection algorithms without sacrificing patient confidentiality.

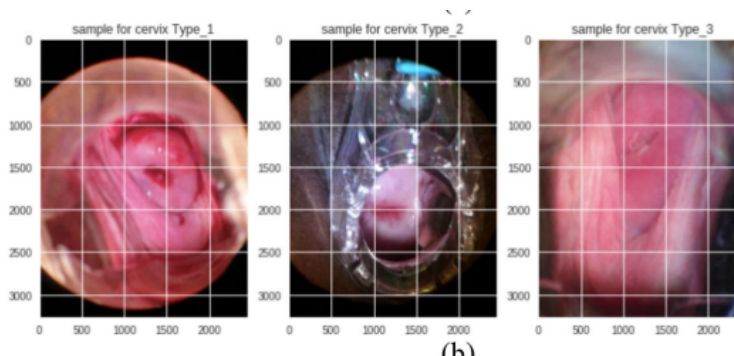
DATASET

The imported dataset consisted of 3 types of cervical cancer images named as: “Type1”, “Type2”, “Type3”. Rescaling these images is a crucial preprocessing step in machine learning, including tasks like cervical cancer detection using GANs. The process involves adjusting the size and possibly the aspect ratio of images to a standardized format suitable for training machine learning models. Here's an outline of the rescaling process:

- **Normalization:** It's usual practice to normalize the image pixel values prior to rescaling. Scaling the pixel values to lie inside a given range, usually between 0 and 1 or -1 and 1, is the usual process of normalization. By ensuring that all images have uniform intensity ranges, this phase can aid in enhancing the training procedure.
- **Choosing the Target Size:** A number of factors, including computational limitations and the machine learning model's input needs, influence the target size for rescaling. For instance, our CNN model, which detects cervical cancer, needs input photos to have a precise dimension (224 × 224 pixels).
- **Resizing:** The photos are scaled to match the intended size while maintaining their aspect ratio when the size is established.
- **Grey-Scaling:** RGB images are converted to greyscaling for ease of training. The no. of channels are reduced from 3 (RGB) to 1 (Greyscale).
- **Labeling:** Label the images generated from the GAN according to their type for further training. Select a random lot of images from the main directory and create a labeled test folder for validation and testing.

The dataset contains various directories after data pre-processing. There are a total of **23.8k** images in **18 sub-directories**. The images include grayscale images, labeled images, and also the GAN-generated images.

A separate directory is also created for the classifier training which contains both the real and the fake images.



GENERATIVE ADVERSARIAL NETWORK

A generator and a discriminator are two neural networks that compete with one another to form a GAN. The discriminator attempts to discern between actual training data and synthetic data produced by the generator. The generator generates the synthetic data. Through an iterative process, the generator learns to replicate the target distribution from an initial random data distribution. To give the generator feedback on the caliber of the samples it generates, the discriminator is trained to identify instances as genuine or fraudulent.

The main elements and procedures required in creating the generator and discriminator network are broken down as follows:

- **Input Noise Vector Shape:** The generator network will use a 100-dimensional noise vector as input to produce synthetic data. This is indicated by the input noise vector shape, which is defined as (100,).
- **Constructing the Generator Network:**
 - Model Initialization: A sequential model is used to initialize the generator and discriminator network.
 - The model is implemented using Tensorflow and Keras Layers.
 - Layers: Dense, activation, and normalization, output layers were used.
- **Model Compilation:** To construct the network, the model is invoked using a random noise sample. An outline of the network architecture is provided by printing the model's summary, which includes the layers and parameters.

```
Model: "sequential_10"
Layer (type)                Output Shape                Param #
-----
flatten_5 (Flatten)         (None, 50176)               0
dense_35 (Dense)             (None, 512)                 25690624
leaky_re_lu_25 (LeakyReLU)   (None, 512)                 0
dense_36 (Dense)             (None, 256)                 131328
leaky_re_lu_26 (LeakyReLU)   (None, 256)                 0
dense_37 (Dense)             (None, 1)                   257
Total params: 25822209 (98.50 MB)
Trainable params: 25822209 (98.50 MB)
Non-trainable params: 0 (0.00 Byte)
```

```
Model: "sequential_11"
Layer (type)                Output Shape                Param #
-----
dense_38 (Dense)             (None, 512)                 51712
leaky_re_lu_27 (LeakyReLU)   (None, 512)                 0
batch_normalization_15 (Ba  (None, 512)                 2048
tchNormalization)
dense_39 (Dense)             (None, 1024)                525312
leaky_re_lu_28 (LeakyReLU)   (None, 1024)                0
batch_normalization_16 (Ba  (None, 1024)                4096
tchNormalization)
dense_40 (Dense)             (None, 2048)                2099200
leaky_re_lu_29 (LeakyReLU)   (None, 2048)                0
batch_normalization_17 (Ba  (None, 2048)                8192
tchNormalization)
dense_41 (Dense)             (None, 50176)               102810624
reshape_3 (Reshape)          (None, 224, 224, 1)         0
Total params: 105501184 (402.46 MB)
Trainable params: 105494016 (402.43 MB)
Non-trainable params: 7168 (28.00 KB)
```

- **Activation function:** The discriminator uses a sigmoid activation to transform any value between negative and positive infinity into a value between 0 and 1. This makes it possible to

interpret the discriminator output as a probability score that indicates whether the input image is real (almost 1) or fake (nearly 0).

- **Loss Function:** When training the GAN with **binary cross-entropy loss** (it is a measurement of the discrepancy between the actual labels and the projected probabilities), it is preferable to have the output be bounded between 0 and 1, which is ensured by employing a sigmoid activation in the discriminator's last layer.
 - Discriminator loss: The discriminator is designed to produce a value of 1 for real photos and 0 for fake ones.
 - Generator Loss: The generator wants to mislead the discriminator by creating images that are identical to real photos. The binary cross-entropy loss, but with the target labels reversed (that is, 1 for false and 0 for real), is another definition of the generator loss.
 - The binary cross-entropy loss can be formulated as:

$$\text{loss} = -(y_{\text{true}} \log(y_{\text{predicted}}) + (1 - y_{\text{true}}) \log(1 - y_{\text{predicted}}))$$

where y_{true} is the true label (0 or 1) and $y_{\text{predicted}}$ is the predicted probability

Three such GANs were created for the three types of images and trained and tested for their respective datasets. Images generated from each type of GAN were labeled with their respective "type" labels for further classification.

All three models were trained for **500 epochs** in a **batch size** of **32** and we saved the generated images with a **save_interval** of **10**. All three GANs were tested individually to find the quality of generation with the help of labeled test images from the original database which were grayscale to match the channels for analysis.

- **Evaluation Metric:** MSE (**Mean Squared Error**) is a basic metric that measures the average squared difference between comparable pixels in two images and is used in image processing and quality assessment. Whereas a higher MSE number denotes greater dissimilarity, a lower MSE value shows a closer likeness between the real and predicted images. As a result, the MSE formula, which provides a numerical representation of the reconstruction error between the two sets of images, is an essential tool for assessing the integrity and correctness of generated images in relation to real images. The formula for MSE is as follows:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

Where:

- n is the total number of data points (pixels in the case of images).
- Y_i represents the actual pixel value at position i in the image.
- \hat{Y}_i represents the predicted (generated) pixel value at position i in the image.

CLASSIFIER

AlexNet is a deep convolutional neural network architecture that was a breakthrough in computer vision when it achieved state-of-the-art results on the ImageNet LSVRC-2010 competition. It was originally introduced in the paper "ImageNet Classification with Deep Convolutional Neural Networks" by Alex Krizhevsky et al.

The key aspects of AlexNet are:

- Takes 3-channel images of size 224x224 as input
- Uses max pooling and ReLU activations
- Employs 11x11, 5x5, and 3x3 convolution kernels
- Utilizes 3x3 max pooling kernels
- Leverages multiple GPUs for training

PyTorch provides a built-in implementation of AlexNet that can be loaded using **torchvision.models.alexnet()**. This allows you to easily use a pre-trained AlexNet model for transfer learning or fine-tuning your own dataset. To use AlexNet for a different number of classes, you can modify the last fully connected layer.

We train this model with our merged dataset which contains both the real and the fake image. While training it verifies that the model is on the designated device, puts it in training mode, and then uses backpropagation to update the model's parameters by looping over the training data loader. By adding up the running loss and accurate forecasts, the function determines the loss and accuracy for every batch. It calculates the average loss and accuracy after each epoch, giving crucial metrics for tracking the model's training development. This function is essential to the creation and assessment of machine learning systems since it contains the fundamental training logic required to train deep learning models efficiently.

For this task, we import **tqdm library**. This Python package offers an easy-to-use and expandable method for making progress bars for iterations and loops. It makes it simpler to keep track of the completion of repetitive tasks by enabling users to visualize task progress. When working on computationally demanding jobs that require iterating over big datasets or completing laborious processes, this library is especially helpful. tqdm creates a smart progress bar that wraps around any iterable and estimates how long the task will take to complete in addition to showing the progress. It provides several features, including the ability to manually regulate updates, display statistics in advance, and customize progress bars. tqdm is a useful tool for effectively tracking the development of Python scripts since it has a low overhead on performance.

```
-----
[INFO]: Epoch 20 of 20
Training
100% ██████████ 47/47 [05:52<00:00, 6.34s/it]
Validation
100% ██████████ 11/11 [00:30<00:00, 2.50s/it]
Training loss: 0.668, training acc: 71.251
Validation loss: 1.383, validation acc: 38.955
-----
TRAINING COMPLETE
```

We used an SVM classifier for classification. It is instantiated with a linear kernel, regularization parameter $C=1.0$, and a random seed for reproducibility. Using the `fit()` method, the SVM classifier is trained on the training features (`X_train_features`) and related labels (`y_train`). Using the `accuracy_score` function, the training accuracy is calculated by comparing the predicted labels (`y_train_pred`) with the actual training labels (`y_train`). Using the trained SVM classifier, predictions are generated on the test set features `X_test_features`. Using the `accuracy_score` function, the test accuracy is calculated by comparing the predicted labels (`y_pred`) with the actual test labels (`y_test`). To see how well the SVM classifier performs on the test set, a confusion matrix is plotted. A thorough classification report, including precision, recall, F1-score, and support for every class in the test set, is produced using the `classification_report` function.

- **CONFUSION MATRIX:**

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

- **EVALUATION METRICS:**

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

$$Specificity = \frac{TN}{TN + FP}$$

$$Sensitivity = \frac{TP}{TP + FN}$$

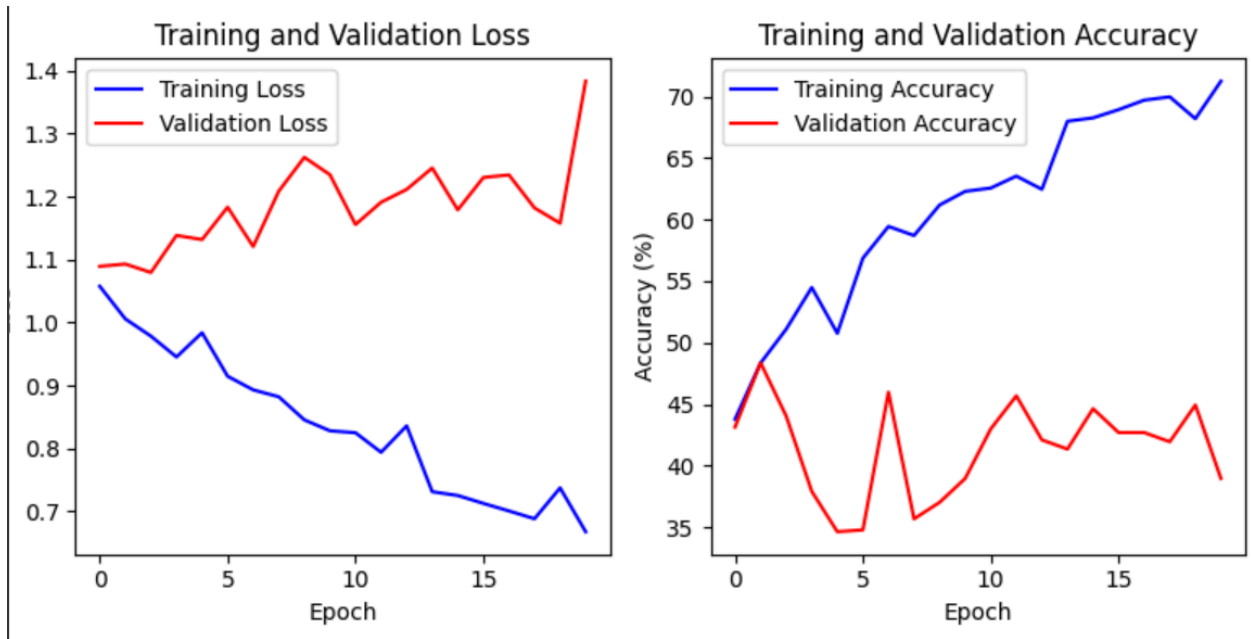
$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\ Score = 2 * \frac{Recall * Precision}{Recall + Precision}$$

$$True\ Positive\ Rate = \frac{TP}{TP + FN}$$

- **RESULT:**



The provided graphs display the training and validation loss and training and validation accuracy over a series of epochs for a machine learning model. Let's analyze each graph:

- **Training Loss (Blue Line):** This metric decreases consistently across epochs, starting from about 1.3 and approaching roughly 0.7, suggesting that the model is learning effectively from the training data.
- **Validation Loss (Red Line):** Initially, this metric also decreases, indicating that the model is generalizing well to new data. However, after about 6 epochs, the validation loss begins to fluctuate and notably spikes around epoch 16, ending much higher than it started. This behavior typically indicates overfitting, where the model learns noise or irrelevant details from the training data, which do not generalize to new data.
- **Training Accuracy (Blue Line):** Increases steadily over time, which is expected as the model continues to fit the training data better. It starts at around 40% and increases to just under 70% by the end of the observed epochs.
- **Validation Accuracy (Red Line):** This metric is much more volatile compared to training accuracy. It starts around 40%, peaks around 60% at epoch 5, and then exhibits significant fluctuations, generally trending downwards to end around 45%. The volatility and downward trend in the validation accuracy reinforce the suggestion of overfitting seen in the validation loss graph.

CONCLUSION & RECCOMENDATIONS


- **Overfitting Issue:** The primary concern here is overfitting, as evidenced by the diverging paths of training and validation metrics. The model is becoming too tailored to the training data, losing its ability to perform well on unseen data.
- **Improvement Strategies:**
 - Regularization Techniques: Implement methods like L1 or L2 regularization to penalize overly complex models.
 - Dropout: Introducing dropout layers in a neural network can help by randomly disabling neurons during training, forcing the network to learn more robust features.
 - Data Augmentation: For tasks like image and text classification, increasing the diversity of training data through augmentation can improve generalization.
 - Early Stopping: Stop the training when validation performance starts degrading, even if training performance continues to improve.
 - Hyperparameter Tuning: Adjust learning rate, batch size, or other model parameters to find a better balance between training and validation performance.

These steps should help mitigate the overfitting issue and improve the model's generalization to new, unseen data.

FUTURE SCOPE

- **CONDITIONAL GAN:** the project can also be implemented through Conditional GAN architecture. These networks are an example of a GAN design in which, in order to enhance the caliber of generated samples, the discriminator and generator are both conditioned on some extra data, such as class labels or other auxiliary data. The materials do not specifically refer to "conditional gan," but the fact that they include studies on variational inference and conditional structures in Gaussian approximations points to a relationship with the more general idea of conditional models in machine learning. A crucial concept in many probabilistic modeling techniques, including GANs, is conditioning models on a specific input. To obtain more precise information on "conditional gan," it could be advantageous to examine supplementary materials or scholarly articles that specifically tackle this subject matter about generative adversarial networks.
- Improve the quality of generated images by modifying the filter and dense layers to a better MSE.
- Test the model for some more evaluation metrics such as SSIM score.
- The code can be optimized into a single GAN performing multiclass training.

IMPORTANT LINKS

- **COLAB NOTEBOOK:**  [SOP_Cervical_Cancer.ipynb](#)
- **KAGGLE DATASET:** <https://www.kaggle.com/datasets/archijain916/cervical-cancer>

REFERENCES

1. Elakkiya, R., Subramaniaswamy, V., Vijayakumar, V., & Mahanti, A. (2021). Cervical cancer diagnostics healthcare system using hybrid object detection adversarial networks. *IEEE Journal of Biomedical and Health Informatics*, 26(4), 1464-1471.
2. Lin, Q., Chen, X., Liu, L., Cao, Y., Man, Z., Zeng, X., & Huang, X. (2022). Detecting multiple lesions of lung cancer-caused metastasis with bone scans using a self-defined object detection model based on SSD framework. *Physics in Medicine & Biology*, 67(22), 225009.
3. Yuntao Shou, Tao Meng, Wei Ai, Canhao Xie, Haiyan Liu, Yina Wang, "Object Detection in Medical Images Based on Hierarchical Transformer and Mask Mechanism", *Computational Intelligence and Neuroscience*, vol. 2022, Article ID 5863782, 12 pages, 2022. <https://doi.org/10.1155/2022/5863782>
4. Tang, C., Feng, Y., Yang, X., Zheng, C., & Zhou, Y. (2017, July). The object detection based on deep learning. In *2017 4th International Conference on Information Science and Control Engineering (ICISCE)* (pp. 723-728). IEEE.
5. Tan, L.S.L., Bhaskaran, A. & Nott, D.J. Conditionally structured variational Gaussian approximation with importance weights. *Stat Comput* **30**, 1255–1272 (2020). <https://doi.org/10.1007/s11222-020-09944-8>
6. TensorFlow Documentation. ImageDataGenerator. Accessed: December 2023. Year you accessed the page. URL: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator.
7. Pytorch Documentation. URL: <https://pytorch.org/docs/stable/index.html>