**IMPLEMENTATION**

**MODULES:**

* Dataset
* Importing the necessary libraries
* Retrieving the images
* Splitting the dataset
* Building the model
* Apply the model and plot the graphs for accuracy and loss
* Accuracy on test set
* Saving the Trained Model

**MODULES DESCSRIPTION:**

**Dataset:**

In the first module of Prediction of Lung Cancer using Convolution Neural Networks, we developed the system to get the input dataset. Data collection process is the first real step towards the real development of a machine learning model, collecting data. This is a critical step that will cascade in how good the model will be, the more and better data that we get; the better our model will perform. There are several techniques to collect the data, like web scraping, manual interventions. Our dataset is placed in the project and it’s located in the model folder. The dataset is referred from the popular standard dataset repository kaggle where all the researchers refer it. The dataset consists of 13,125. images. The following is the URL for the dataset referred from kaggle.

<https://www.kaggle.com/datasets/jayaprakashpondy/lung-cancer>

**Importing the necessary libraries:**

We will be using Python language for this. First we will import the necessary libraries such as keras for building the main model, sklearn for splitting the training and test data, PIL for converting the images into array of numbers and other libraries such as pandas, numpy, matplotlib and tensorflow.

**Retrieving the images:**

In this module we will retrieve the images from the dataset and convert them into a format that can be used for training and testing the model. This involves reading the images, resizing them, and normalizing the pixel values. We will retrieve the images and their labels. Then resize the images to (224, 224) as all images should have same size for recognition. Then convert the images into numpy array.

**Splitting the dataset:**

In this module, the image dataset will be divided into training and testing sets. Split the dataset into Train and Test. 80% train data and 20% test data. This will be done to train the model on a subset of the data, validate the model's performance, and test the model on unseen data to evaluate its accuracy. Split the dataset into train and test. 80% train data and 20% test data.

**Inception v3** **| CNN model:**

***Architecture:***

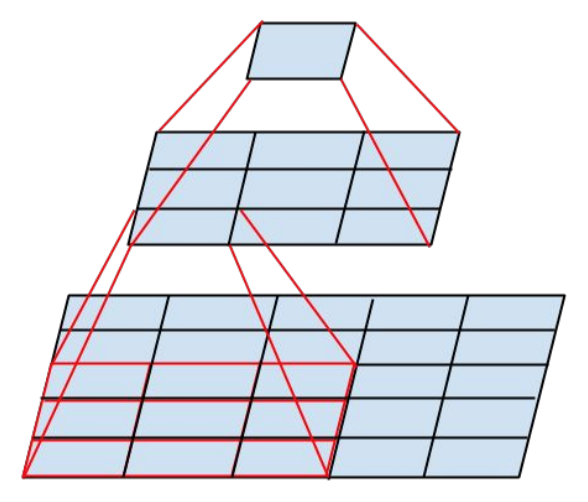
Inception v3 mainly focuses on burning less computational power by modifying the previous Inception architectures. In comparison to VGGNet, Inception Networks (GoogLeNet/Inception v1) have proved to be more computationally efficient, both in terms of the number of parameters generated by the network and the economic cost incurred (memory and other resources). If any changes are to be made to an Inception Network, care needs to be taken to make sure that the computational advantages aren’t lost. Thus, the adaptation of an Inception network for different use cases turns out to be a problem due to the uncertainty of the new network’s efficiency. In an Inception v3 model, several techniques for optimizing the network have been put suggested to loosen the constraints for easier model adaptation. The techniques include factorized convolutions, regularization, dimension reduction, and parallelized computations.

*Inception v3 Architecture*

The architecture of an Inception v3 network is progressively built, step-by-step, as explained below:

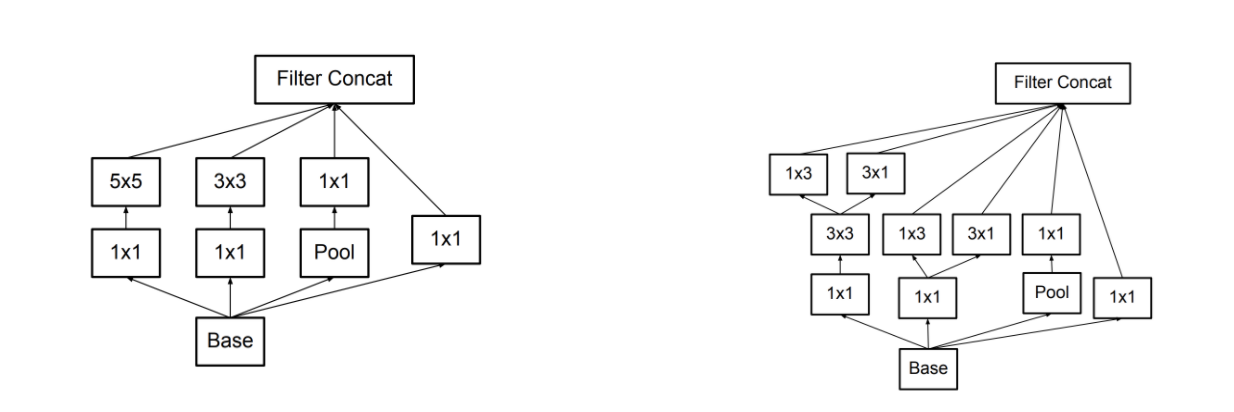
1. Factorized Convolutions: this helps to reduce the computational efficiency as it reduces the number of parameters involved in a network. It also keeps a check on the network efficiency.

2. Smaller convolutions: replacing bigger convolutions with smaller convolutions definitely leads to faster training. Say a 5 × 5 filter has 25 parameters; two 3 × 3 filters replacing a 5 × 5 convolution has only 18 (3\*3 + 3\*3) parameters instead.

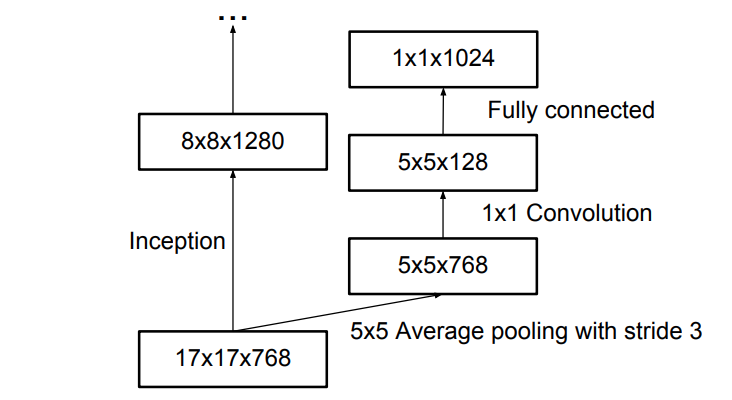


In the middle we see a 3x3 convolution, and below a fully-connected layer. Since both 3x3 convolutions can share weights among themselves, the number of computations can be reduced.

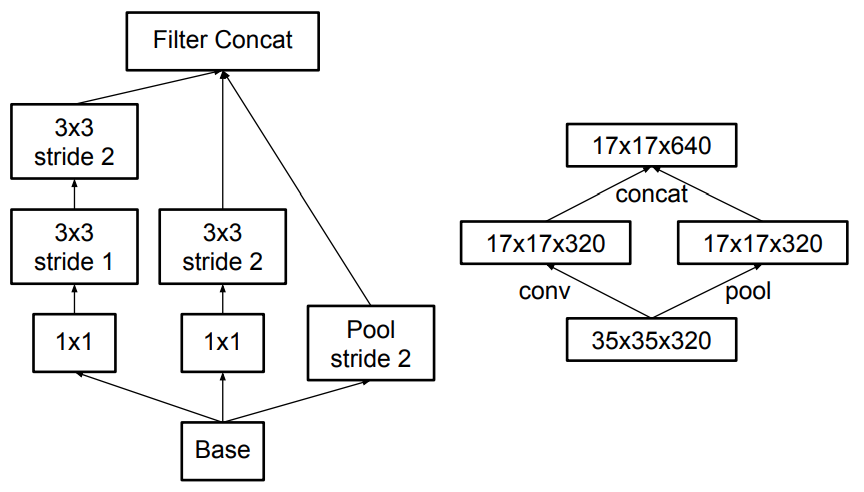
3. Asymmetric convolutions: A 3 × 3 convolution could be replaced by a 1 × 3 convolution followed by a 3 × 1 convolution. If a 3 × 3 convolution is replaced by a 2 × 2 convolution, the number of parameters would be slightly higher than the asymmetric convolution proposed.



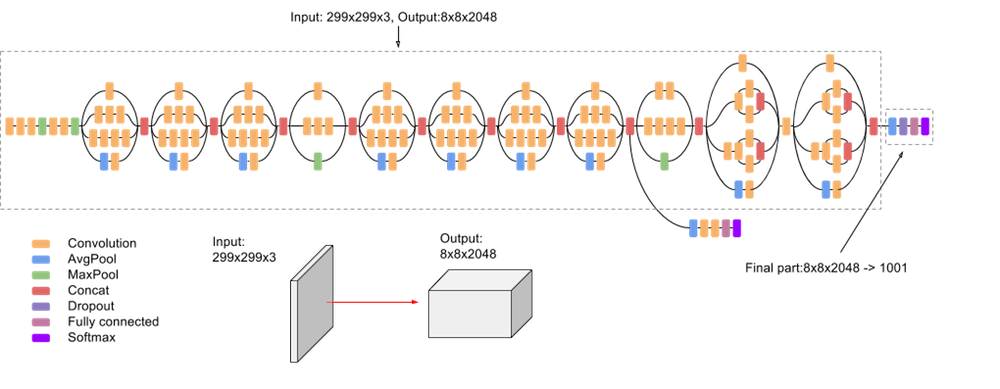
4. Auxiliary classifier: an auxiliary classifier is a small CNN inserted between layers during training, and the loss incurred is added to the main network loss. In GoogLeNet auxiliary classifiers were used for a deeper network, whereas in Inception v3 an auxiliary classifier acts as a regularizer.



5. Grid size reduction: Grid size reduction is usually done by pooling operations. However, to combat the bottlenecks of computational cost, a more efficient technique is proposed:



All the above concepts are consolidated into the final architecture.



**Building the model:**

The concept of convolutional neural networks is very successful in image recognition. The key part to understand, which distinguishes CNN from traditional neural networks, is the convolution operation. Having an image at the input, CNN scans it many times to look for certain features. This scanning (convolution) can be set with 2 main parameters: stride and padding type. As we see on below picture, process of the first convolution gives us a set of new frames, shown here in the second column (layer). Each frame contains information about one feature and its presence in scanned image. Resulting frame will have larger values in places where a feature is strongly visible and lower values where there are no or little such features. Afterwards, the process is repeated for each of obtained frames for a chosen number of times. In this project I chose a InceptionV3 Architecture model which contains only two convolution layers.

The latter layer we are convolving, the more high-level features are being searched. It works similarly to human perception. To give an example, below is a very descriptive picture with features which are searched on different CNN layers. As you can see, the application of this model is face recognition. You may ask how the model knows which features to seek. If you construct the CNN from the beginning, searched features are random. Then, during training process, weights between neurons are being adjusted and slowly CNN starts to find such features which enable to meet predefined goal, i.e. to recognize successfully images from the training set.

Between described layers there are also pooling (sub-sampling) operations which reduce dimensions of resulted frames. Furthermore, after each convolution we apply a non-linear function (called **ReLU**) to the resulted frame to introduce non-linearity to the model.

Eventually, there are also fully connected layers at the end of the network. The last set of frames obtained from convolution operations is flattened to get a one-dimensional vector of neurons. From this point we put a standard, fully-connected neural network. At the very end, for classification problems, there is a softmax layer. It transforms results of the model to probabilities of a correct guess of each class

**Apply the model and plot the graphs for accuracy and loss:**

Once the model is built, it will be applied to the validation set to evaluate its accuracy and loss. The accuracy and loss will be plotted as a function of the number of epochs to visualize the performance of the model. We will compile the model and apply it using fit function. The batch size will be 1. Then we will plot the graphs for accuracy and loss. We got average training accuracy of 94.00%.

**Accuracy on test set:**

After training and evaluating the model on the validation set, the accuracy of the model will be assessed on the test set. The accuracy on the test set will be an important metric for evaluating the model's performance. We got an accuracy of 93.00% on test set.

**Saving the Trained Model:**

Once you’re confident enough to take your trained and tested model into the production-ready environment, the first step is to save it into a .h5 or .pkl file using a library like pickle .

Make sure you have pickle installed in your environment.

Next, let’s import the module and dump the model into .h5 file.