Comparison of Different Deep Learning Architectures on Histopathologic Cancer Detection (Final Project Group 45)

Alby Thomas
University of Michigan
albytho@umich.edu

Kartikeya Kandula
University of Michigan
kartkand@umich.edu

Abstract

With the proliferation of machine learning across every field of science, we have seen rapid advances in the effectiveness of predictive modeling. In medicine, a current roadblock in treating potential cancer patients is determining whether or not the patient possesses tumorous cells. In the current process of determining whether a patient has cancer, physicians manually go through images of scans of the patient's body which can be a time consuming task. With a machine learning based predictive model, this lengthy process could be streamlined and done at scale. Our goal is to compare popular deep learning models and apply it to this problem of identifying metastatic cancer cells in patient scans.

1. Introduction

Our problem comes from an online Kaggle competition [1, 2]. We are applying computer vision to identify metastatic cancer in small image patches taken from larger digital pathology scans. This is a binary image classification task in which we determine whether metastatic cancer is present or not in images. We will be using a slightly modified version of the PatchCamelyon (PCam) benchmark dataset. For this problem, a positive label indicates that the center 32x32 region of a patch contains at least one pixel of tumor tissue.

Our original goal was to develop a model that could get competitive results in the Kaggle competition for this problem. We quickly decided to modify the problem we are trying to address. Many of the top submissions on the Kaggle competition likely belong to Ph.D. students who have spent a considerable amount of time on this problem and it is unlikely that we would be able to get comparable scores in the given timeframe. Therefore, instead of implementing our own CNN architecture, we plan to compare the performance of popular neural networks such as ResNet50.

2. Training Data

Our dataset consists of a series of small image scans of lymph node sections taken from large pathology scans. This dataset is originally a subset of the PCam dataset. The issue, however, is that the original dataset contained duplicate images. Kaggle, however, offered a more refined version of this dataset by removing these duplicate images for us. They offered this dataset as a part of one of their competitions

This new dataset contains approximately 220,000 training images and 57,000 evaluation images. Across all the images, there is supposed to be an even split of positive and negative images. In the training data, however, there is an uneven split of images, where 60% are negative and 40% are positive.

In each of the images, there is an outer and inner region. A positive label is indicated by whether or not there is tumor tissue in the inner region. Any tumor tissue in the outer region does not impact the label.

3. Methods

3.1. Data Augmentation

There were certain issues with the dataset of images given to us from Kaggle. There are two labels for the dataset. If an image has a cancerous cell, it is given the label of 1. If it contains no cancerous cells, however, the image is given the label of 0. When taking a look at both the training and test images, we see that there is roughly an even distribution of images in the two categories. If we take a look at the training data alone, however, this is not the case. In the training data, there is roughly a 40-60 split between images with cancerous cells and images with no cancerous cells. This data imbalance could create an issue when training a neural network, as it would make a model inherently biased towards labeling an image as having no cancerous cells.

To fix this imbalance, we originally wanted to ensure that our model was looking at an equal amount of images from

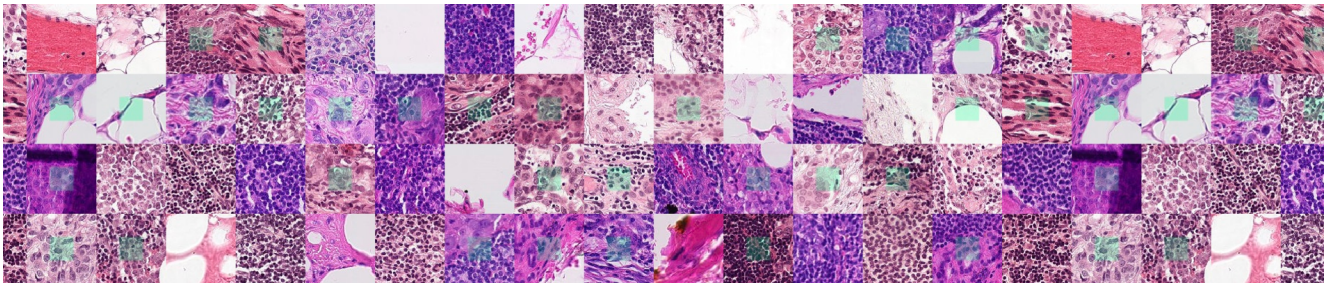

Figure 1. Sample images from the dataset featuring metastatic cancer.

the two categories. One approach to this problem was to limit the amount of images that our model had access to while training from the majority class. Although doing this would ensure that we were looking at an even amount of images between the categories, this would reduce the amount of overall training data we would have. This could potentially negatively impact the performance of our neural network as we would provide it less data to train on. A more suitable approach we devised was to perform data augmentation on the images containing cancerous cells. With this process, we go through each of the images with cancerous cells and slightly alter them and save them as a new image in the category. Doing this increases the size of the minority class until it is roughly the same as the previous majority class size. This latter process is the implementation we are working with at the moment. Currently, we change the brightness of the image. It might, however, be useful to consider rotating the image. Another approach worth looking into is going through images in the majority class and removing images from the class that are very similar to other images in the class until the classes are of equal size. An issue with this approach however is that although two images may look very similar to the human eye, the model training upon the image but find them to be very different.

We decided to create and train on two separate datasets: one with the original unaltered dataset from Kaggle and one with a balanced dataset that augmented images from the minority class as described above to create an equal amount of images in each class. We then trained models on each of these datasets and compared performances to see if the data imbalance truly harms performance of our chosen models.

3.2. Transfer Learning

Our goal was to compare of different deep learning architectures on the task of identifying metastatic tissue in histopathologic scans. To do this we used a method known as transfer learning in which a model developed for one task is repurposed and used as a starting point for a different task [3]. It is a popular method due to the the high computational cost of developing and training a model from scratch and the massive datasets required to do so. This works for many popular models such as ResNet50 because many of the layers for such architectures learn general features until they reach the final layers. Neural networks generally detect edges and corners in their early layers and shapes in their middle layers. It is the later layers that accomplish task-specific functions such as identifying complex objects. By leveraging the initial layers that identify edges, corners, and shapes, we only need to retrain the latter layers in order to repurpose a new model to new tasks.

We applied transfer learning to this problem using the Keras neural networks API running on top of Tensor Flow. Through Keras, we are able to access many well-known models such as ResNet50 and easily remove the top layers and freeze the weights of the remaining layers. After doing this, we add a series of fully connected layers to our model and finetune the model by only training these newly added layers. The output of this modified model then becomes the classification prediction for our task rather than the task that the model was originally trained for.

3.3. Optimization

We use the Adam optimization, an improvement on general stochastic gradient descent that has seen broad adoption in computer vision and natural language processing [4]. Regular gradient descent generally maintains a singular learning rate for all parameter updates in the neural network. In contrast, with the Adam optimization, individual network parameters maintain their own learning rates which are updated independently of each other. Each learning rate is adjusted based on recent magnitudes of the descent for the associated parameter.

3.4. Data Augmentation

In order to further improve the dataset, we randomly transform batches of image data with real-time data augmentation. Images are randomly rotated between 1 and 360 degrees. By doing this, we are able to artificially increase the amount of varying data that our model is exposed to.

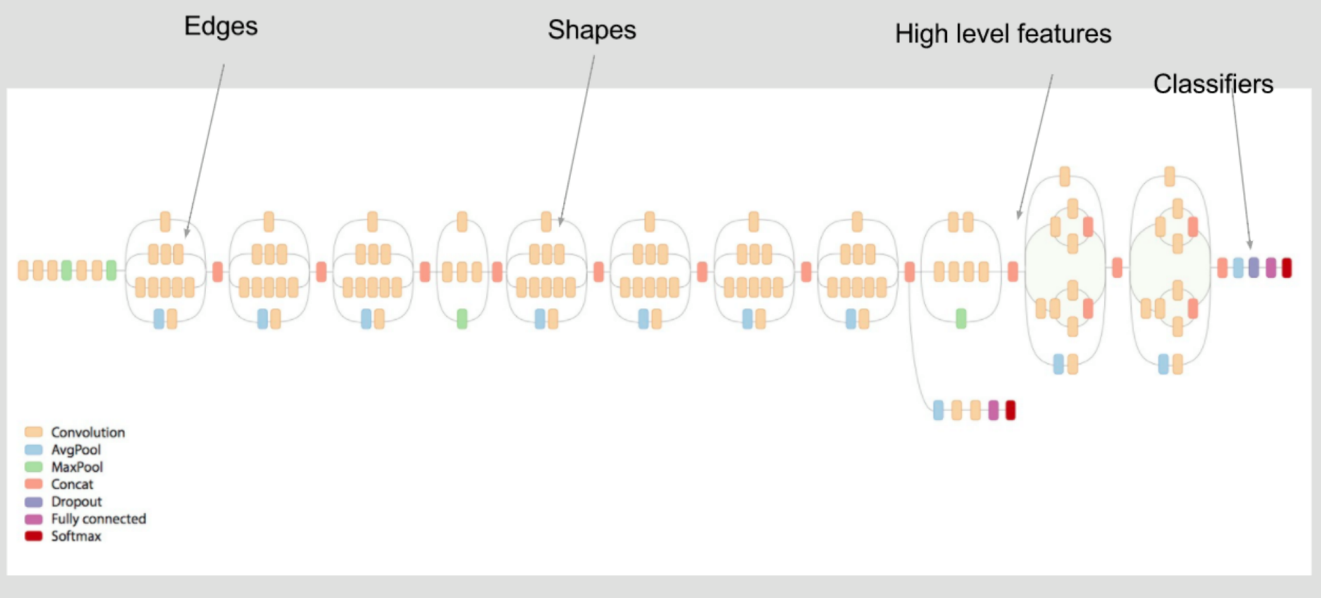

Figure 2. Architecture for Inception v3, a popular neural network model, showing where certain features are learned (TensorFlow).

4. Architectures

4.1. ResNet50

When most deep networks start converging, they run into the degradation problem. This is when the accuracy saturates and decreases quickly due to the network depth increasing. A simple solution to this would be to reduce the number of layers in the network and make it more shallow. This, however, may not work in some cases as certain problems demand complex modeling that can only be approximated with a complex neural network architecture. Another alternative solution to this problem is to use residual learning through ResNet50. A general network aims to learn the features of the input image. The ResNet50 network, instead, tries to learn the residual. This residual is simply the subtraction of the feature learned from the input of the layer. The model efficiently does this by forming a direct connection between the input of one layer and another layer. ResNet50 has been proven to be easier to train than just a simple deep convolutional neural network as well as get over the degradation problem [5].

4.2. MobileNet

While architectures like ResNet are able to yield promising results, the computation needed to train and run it may be too much for some devices. Although computers may be able to handle this computation, devices such as mobile phones may not. This is where MobileNet comes into play. This architecture was proposed by Google and made with the intent that it can be efficiently utilized by devices with small computational power. MobileNet uses depthwise separable convolutions. This type of convolution is much more efficient than the standard convolutions most architectures use. This efficient convolution method is what makes MobileNet a less computationally expensive and allows it to be accessible on more devices [6].

4.3. VGG16

VGG16 is another popular deep learning architecture architecture. What is appealing about it is its simplicity and the fact that it has won the ILSVRC competition in 2014. It only consists of 16 convolutional layers that has a 3 by 3 window. It is often used by the community as a baseline feature extractor [7].

5. Results

The results of training the different models are shown in figures 3-8. Overall, all the models seemed to perform well. Because the problem is a binary classification, guessing will give you a 50% accuracy. All the models were able to exceed this. The models were able to get an accuracy of around 80%. The ResNet50 model achieved an accuracy of 81.75%. VGG16 was able to achieve an accuracy of 81.16%. MobileNet was able to reach 77.53%.

From all this, it is evident that the model that showed the best performance was ResNet50. The model that showed the worst performance was MobileNet. Although ResNet50 did outperform MobileNet in terms of accuracy, it lagged behind when it came to training time. While ResNet50 took 45 minutes to train on a GPU, MobileNet only took 10 minutes. MobileNet took much less time to train than ResNet50

and was able to achieve an accuracy that was only about 4% lower than that of ResNet50. If a user were to not care about training time and wanted to judge a model soley off accuracy, ResNet50 would prove to be the best model. If a user were to care about the training time at all, then MobileNet could be the ideal choice as it trains in the shortest amount of time while delivering accuracies close to those of ResNet50. In this scenario, VGG16 could also be an optimal choice as it was able to get an accuracy close to that of ResNet50 in only about 14 minutes. Although this training time is slightly longer than MobileNet's, it is still significantly smaller than ResNet's training time.

These findings come from models that were trained on imbalanced data. After manipulating the training data to make it balanced and training the models on this, their accuracies only improved by roughly 1%. This shows that having an imbalanced dataset did not make a significant difference in results. This could also be due to the fact that there was only a slight class imbalance in our data. 60% of the images were negative and 40% were positive. Had the imbalance been greater, the results might have differed more.

6. Challenges

Throughout this project, we ran into some issues. One challenge we came across was with training the deep learning models. Initially, all the training was done using a CPU. This was fine when we it came to training MobileNet, as it took roughly 2 hours. It became an issue, however, when we tried training ResNet50. For this model, the total time necessary for training was roughly 20 hours. This was highly impractical to do on our end. This issue was resolved by carrying the computation over to FloydHub, an online deep learning platform, where we could use a GPU to train the models. This proved to be much faster as it shortened training time for models like ResNet50 from 20 hours to 45 minutes.

Another challenge we came across was not knowing the labels for the given test images. The test images were given simply for the sake of the kaggle competition and testing a model's accuracy on data it has never seen before. We are not meant to know which images correspond with which label. We resolved this by making our own test set. We took about 20% of the images in the training set and moved them over to the new test set. This would allow us to be able to test our models accuracy on new data as we know what label each of the images in this set belong to. This would help us figure out if we were overfitting any of the models.

This leads us to another issue we faced in this study. Although we were able create our own test set, we had programmatic issues when it came to getting our trained models to predict new images. Before passing an image into the input layer, it has to go through a few preprocessing steps. The image fist has to be flattened to a one dimensional array and then encoded. The issue came when we wanted to decode the results. Because we were training our models on a different number of classes than the architectures were originally created for, we could not simply use the decode function that came along with the model's library. Because of this, we did not know how to interpret the prediction results and were not able to successfully resolve this issue.

7. Further Work

Further investigation could be done on comparing even more than just 3 deep learning architectures. Throughout this study, hyper parameters such as number of epochs and batch size were all kept constant. It would be interesting to see how how the different models rank against one another if we were to manipulate any of these variables.

A further optimization that could be made to our solution is to run the frozen layers of the neural network on each of the images and store the resulting outputs. Because we are never modifying parameters in the earlier, frozen layers of the network, it is unnecessary and time consuming to run images through each of these layers when it could just be done once. The stored outputs could then be run through the few layers added on to the end of the network for our task-specific specialization to results in improvements in speed. However, it may prove to be expensive to store the output of the frozen layers for large datasets. Another complication is that images are rotated a random amount in real time while the network is training. If this approach were to be used, we would need to store the output from the frozen layers of the network for every possible rotation of each image.

References

[1] B. S. Veeling, J. Linmans, J. Winkens, T. Cohen, M. Welling. *"Rotation Equivariant CNNs for Digital Pathology". arXiv:1806.03962*.

[2] Ehteshami Bejnordi et al. *Diagnostic Assessment of Deep Learning Algorithms for Detection of Lymph Node Metastases in Women With Breast Cancer. JAMA: The Journal of the American Medical Association, 318(22), 21992210. doi:jama.2017.14585*.

[3] Brownlee, Jason. "A Gentle Introduction to Transfer Learning for Deep Learning" Machine Learning Mastery, 20 Dec. 2017, *https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/*.

[4] Brownlee, Jason. "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning" Machine Learning Mastery, 3 July 2017, *https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/*.

[5] Jay, Prakash. Understanding and Implementing Architectures of ResNet and ResNeXt for State-of-the-Art Image... Medium, Medium, 7 Feb. 2018, *medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cf51669e1624*.

[6] Douillard, Arthur. 3 Small But Powerful Convolutional Networks. Towards Data Science, Towards Data Science, 13 May 2018, *towardsdatascience.com/3-small-but-powerful-convolutional-networks-27ef86faa42d.*

[7] Das, Siddharth. CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and More .... Medium, Medium, 16 Nov. 2017, *medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5*.

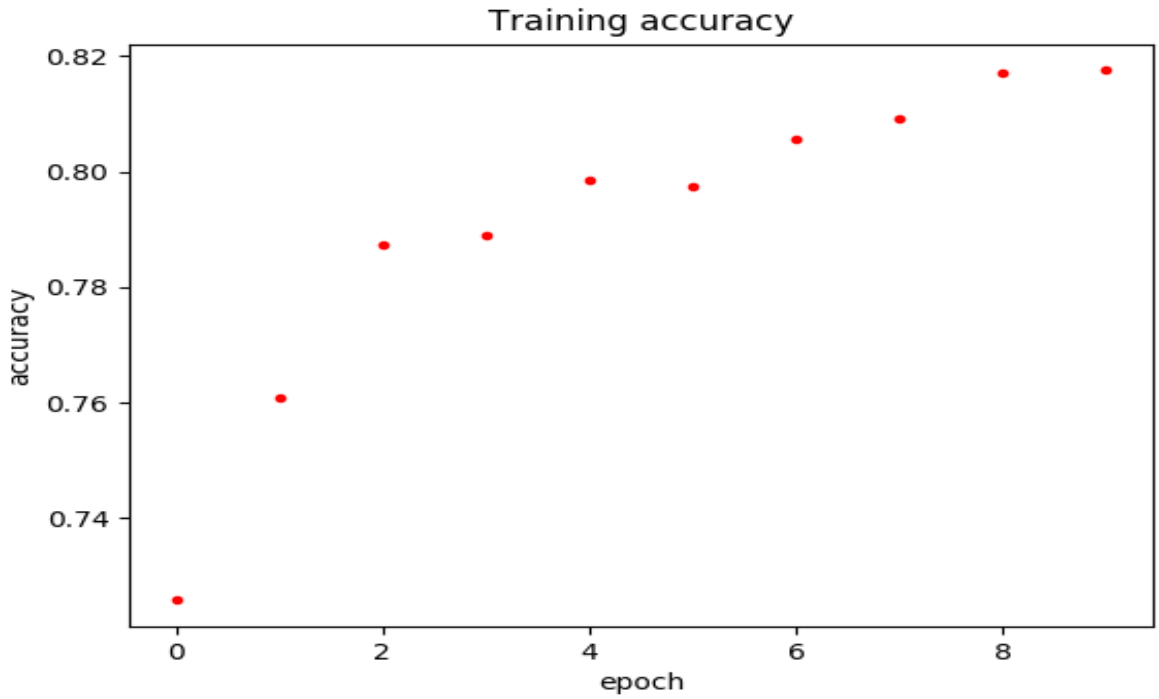

Figure 3. Accuracy vs Epoch graph for ResNet50 on imbalanced data

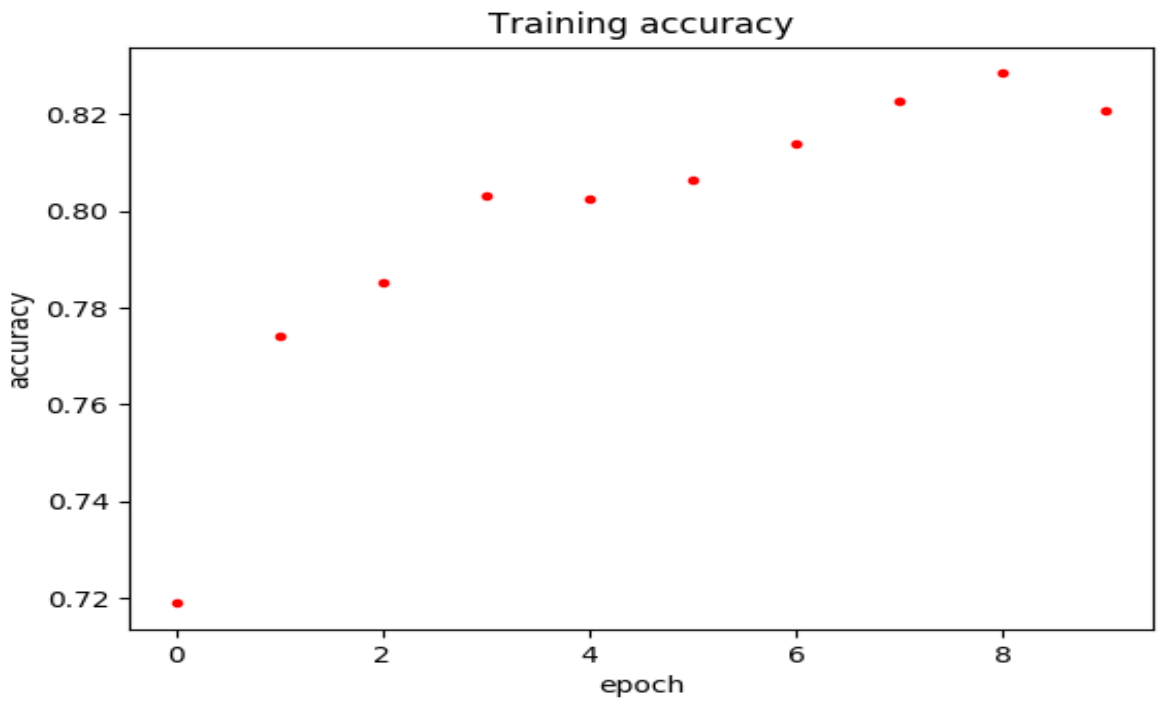

Figure 4. Accuracy vs Epoch graph for ResNet50 on balanced data

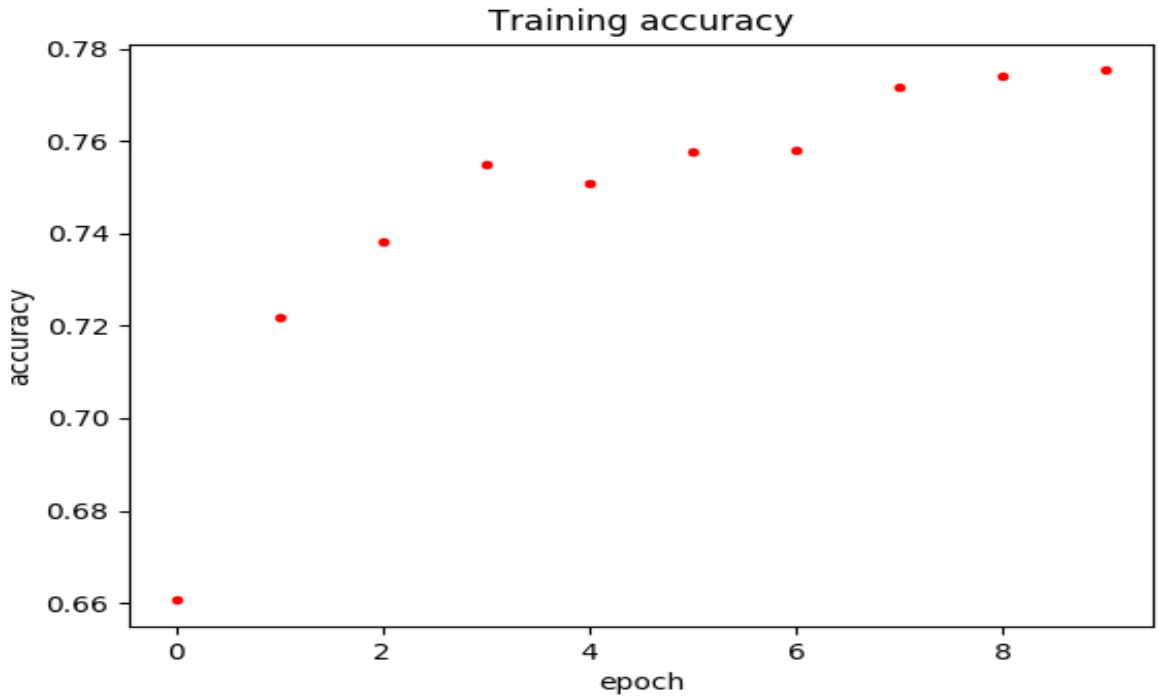

Figure 5. Accuracy vs Epoch graph for MobileNet on imbalanced data

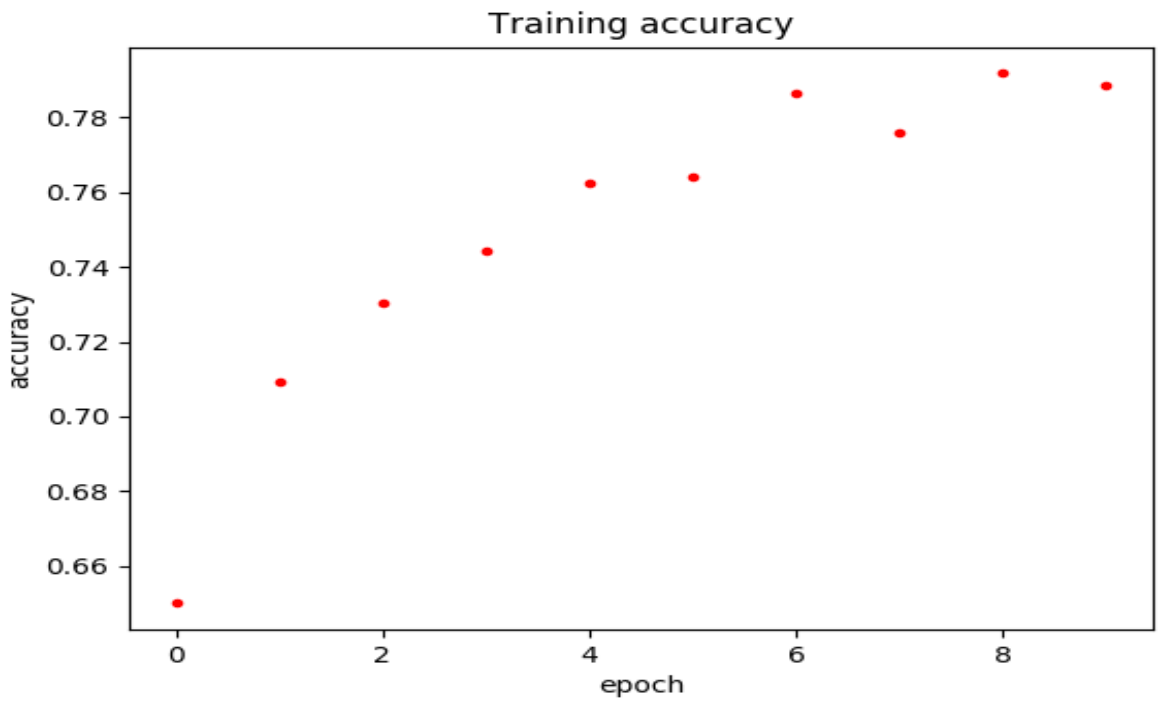

Figure 6. Accuracy vs Epoch graph for MobileNet on balanced data

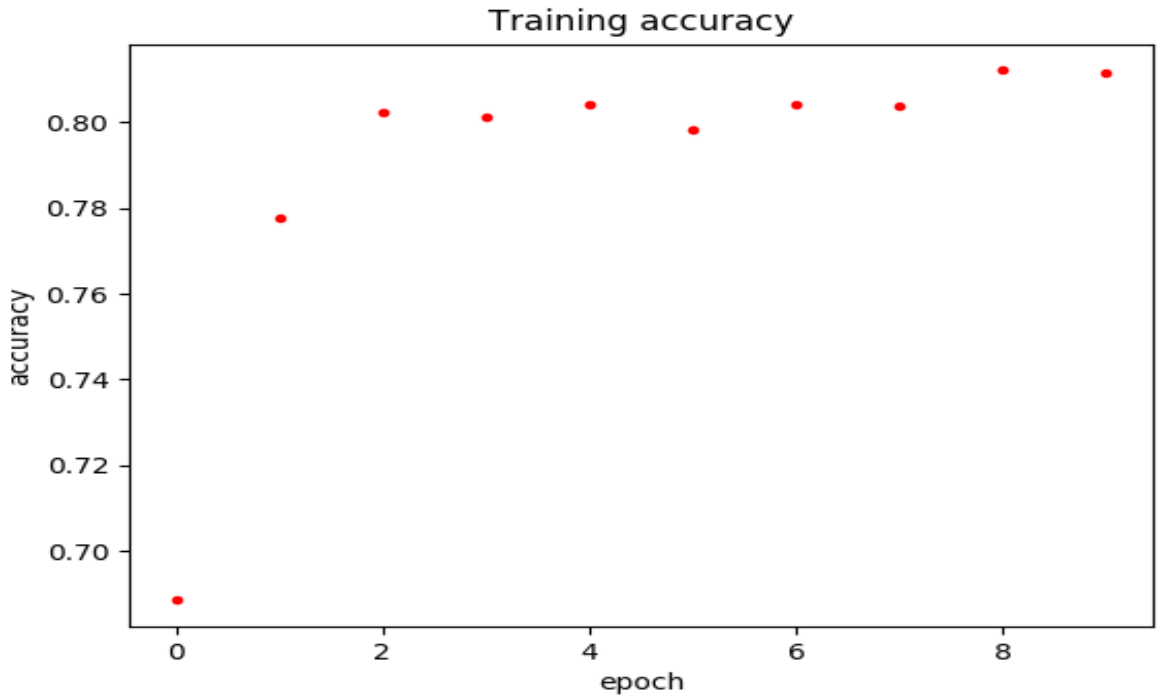

Figure 7. Accuracy vs Epoch graph for VGG16 on imbalanced data

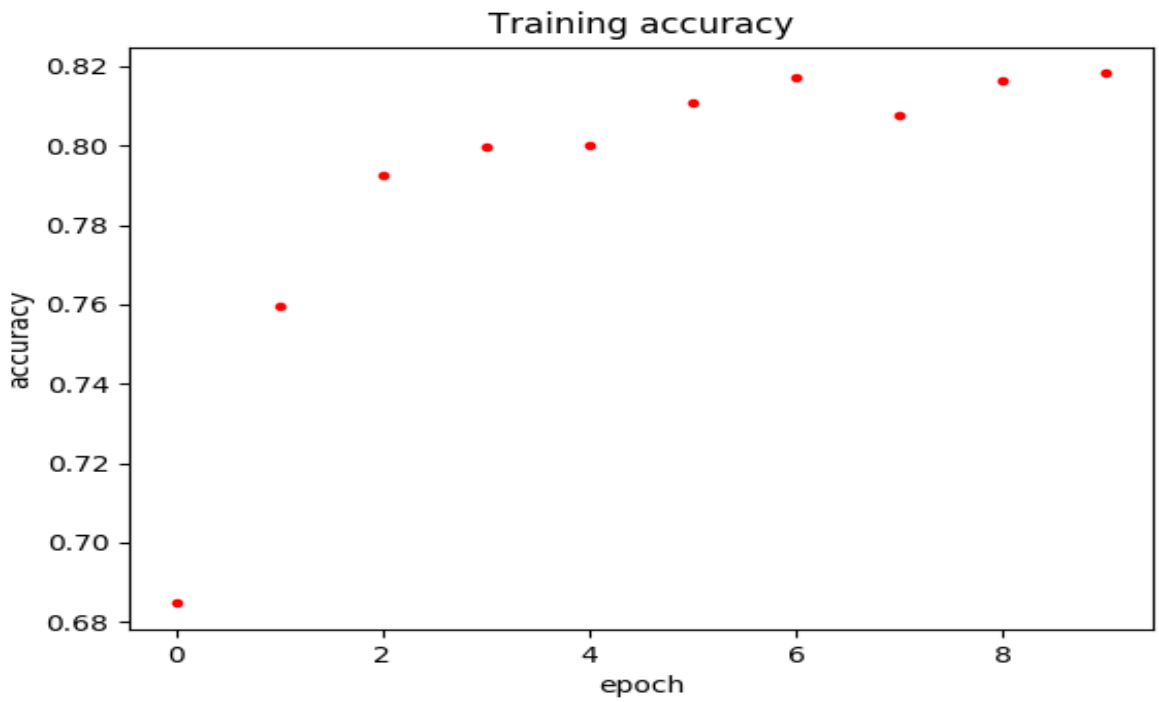

Figure 8. Accuracy vs Epoch graph for VGG16 on balanced data