CNN based Season Classification System for Images

COMS 4995 : Deep Learning for Computer Vision

Dhananjay Shrouty

UNI - ds3521

04.27.2017

INTRODUCTION

The system is designed to provide a method for tagging images based on the season they depict. As of now, the system can predict and classify images based on four seasons - Fall, Spring, Rain and Winter. In designing the system, I have made use of various state-of-the-arts pre-trained models and added my model on top of it to perform a softmax based prediction on whether which season gets the highest probability. The system does fine tuning of the layers in order to achieve better results than a pre-trained model. In the report, I provide a comparison of the various models based on how well they performed on the dataset.

DATA COLLECTION

The data was collected using Python to scrape images from Google. I wrote a Python script which downloaded 400 images based on a search term. The number 400 seems apt because I noticed that the images tend to diverge from the relevance to the search term as one goes into later pages of Google Image search. 400 Images for a single class seem very less to train a model on. So, I used multiple search terms based on location of seasons at various places such as “Fall New York”, “Spring California”, “Rain London”, “Winter Michigan” etc. This helped me extend my dataset manifold. In the end, I was able to scrape a total of approximately 10000 images which I then segregated into train, validation and test sets.

PROBLEMS WITH RAW DATA

Various problems can arise with raw data. The first is the format. The libraries which I used relied on JPG and PNG images. So, I had to filter out all the other file formats such as PHP, GIF so as the dataset I give as input to the system is in the correct format. Second problem which I faced was corrupt files. It happened that various JPG and PNG files which I was using as input to the system were corrupted (maybe because of not being to download effectively or being them as corrupt in the first place). For dealing with the second problem, I wrote a Python script using Pillow which helped me identify the corrupt images and remove them. I found nearly 100 corrupt images in my dataset of 10000 images. These findings suggest that data cleaning is important especially if using raw images as it can lead to many problems later in the data pipeline. Cleaning the data beforehand and in an automated way helps save a lot of time which can be used for better development and experimentation of different models.

THE DATASET

Below are some sample images from the dataset which were used either for training or testing purposes

1. Fall
2. Rain
3. Spring
4. Winter

Some sample images from the dataset are shown below:

FALL :

RAINY :

SPRING :

WINTER :

MODELS

I have implemented fine-tuning of the following models (used pretrained model weights from imagenet) and below are the findings based on their performance. In order to suit the models to my problem, I designed the last layer of dense as one containing 1024 units and then using softmax function to get the probability of the classes and then choosing the best class from among them. The class_type that was used had to be changed from Binary to categorical to perform a one vs all classification.

1. VGG16 - I used VGG16(16 layers) as a baseline model although it is far from a baseline model. VGG16 actually performs nicely for a small dataset. With just ~7000 training images and ~1600 cross-validation images, it achieves an accuracy of 88.89% (Cross Validation Accuracy) and 87.16% (Test Accuracy)**.** The following graphs give an overview of how the loss decreases and how learning curves help us determine when the model is starting to overfit. I also found out that the convergence of loss function takes place faster when we use a sophisticated

gradient learning algorithm such as “Adam” or “RMSProp” compared to Stochastic Gradient Descent with line search.

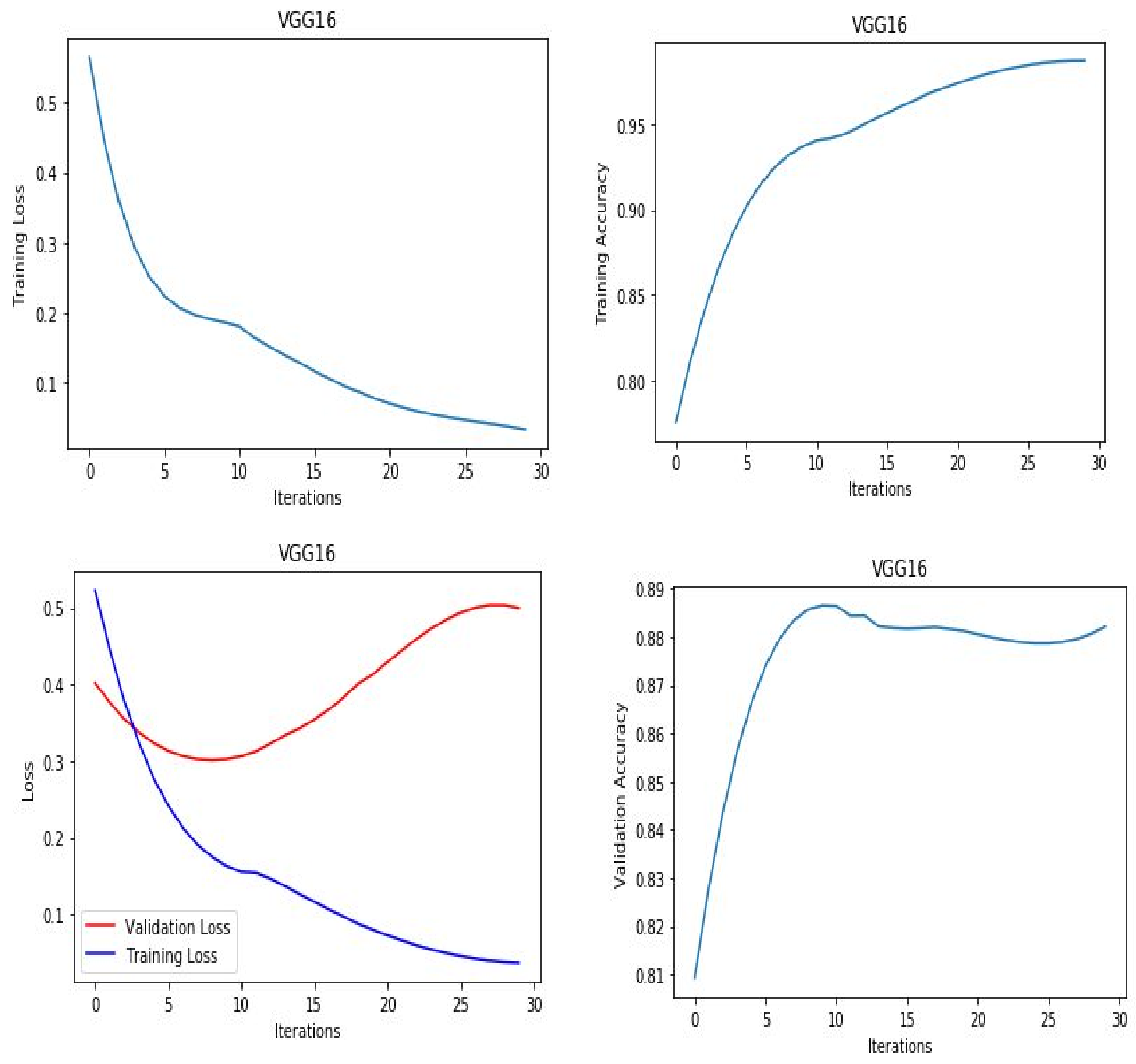


Figure. Graphs for VGG16 Training and Cross Validation

2. VGG19 - VGG19 is a slightly more complex model than VGG16 in that it has 3 more layers compared to VGG16 and thus the speculation I had before running this was that it should give slightly better results when trained on the dataset. With the dataset containing ~7000 training images and ~1600 cross-validation images, it achieves an accuracy of 89.82% (Cross Validation Accuracy) and 88.10% (Test accuracy). The following graphs give an overview of how the loss decreases and learning curves help us determine when the model is starting to overfit. Again,

here Adam and RMSProp result in faster convergence (Although Adam performs slightly faster)

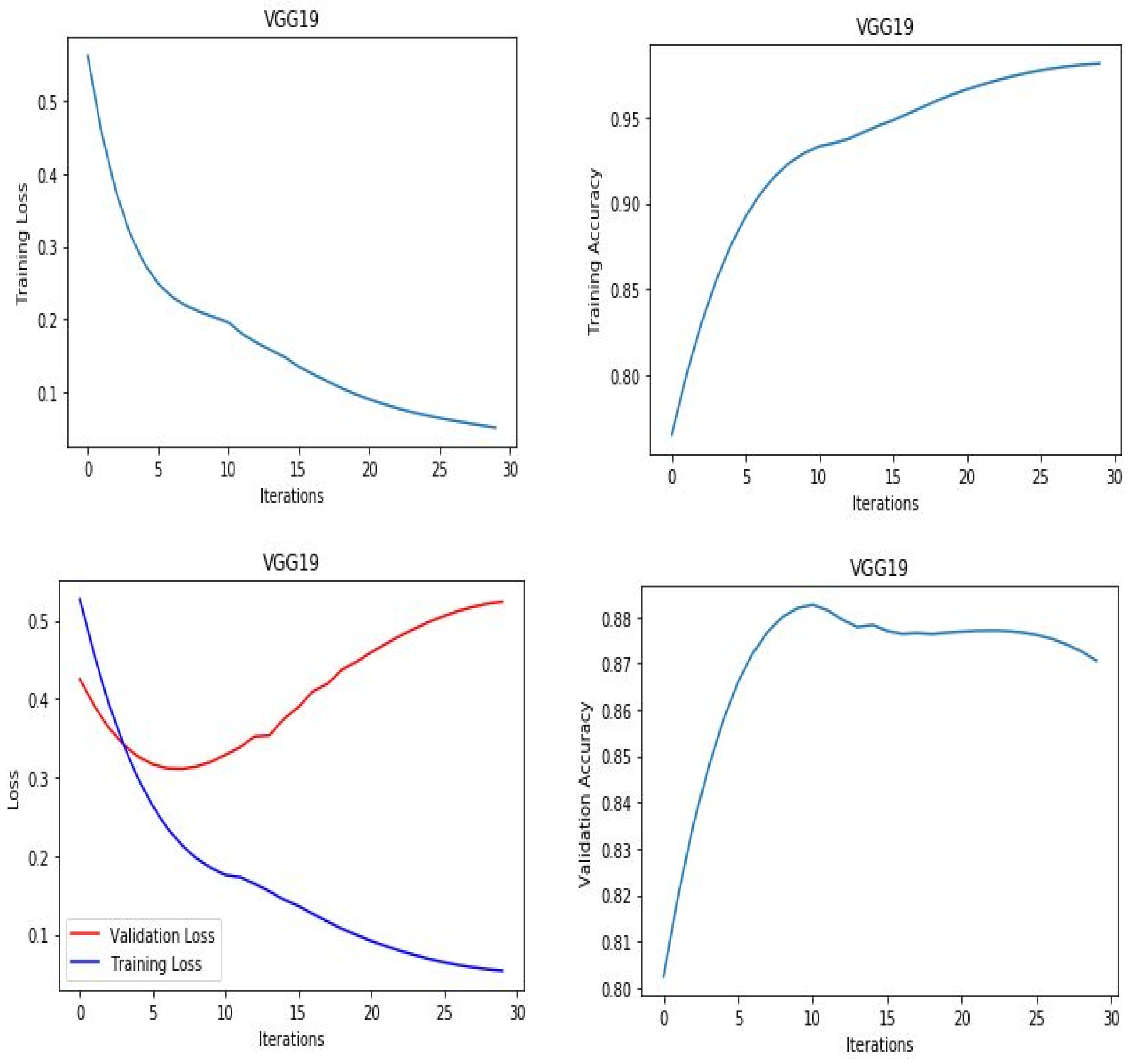


Figure. Graphs for VGG19 Training and Cross-Validation

3. InceptionV3 - The Inception Model is a fairly complex model from Google Research and contains more than 300 layers. I observed that if only some of the last layers are unfreezed then the accuracy does not improve much over the epochs. I tried freezing all except first 50 layers. I fine-tuned the model and discovered an interested finding. InceptionV3 is supposed to perform better than VGG16 or VGG19, but for my case even over 100 epochs it gave an inferior performance comapared to VGG16 and VGG19. One of the reasons could be that

InceptionV3 comprises of numerous layers and thus has numerous parameters (much more compared to VGG16 and VGG19). The weights possibly don't train very well on small datasets and the InceptionV3's performance is heavily dependent on the amount of data. With the dataset containing ~6000 training images, it achieves an accuracy of 86.35% (Cross Validation Accuracy) and 86.07% (Test Accuracy). The following graphs give an overview of how the loss decreases and learning curves help us determine when the model is starting to overfit. Again, here Adam and RMSProp result in faster convergence (Although Adam performs slightly faster).

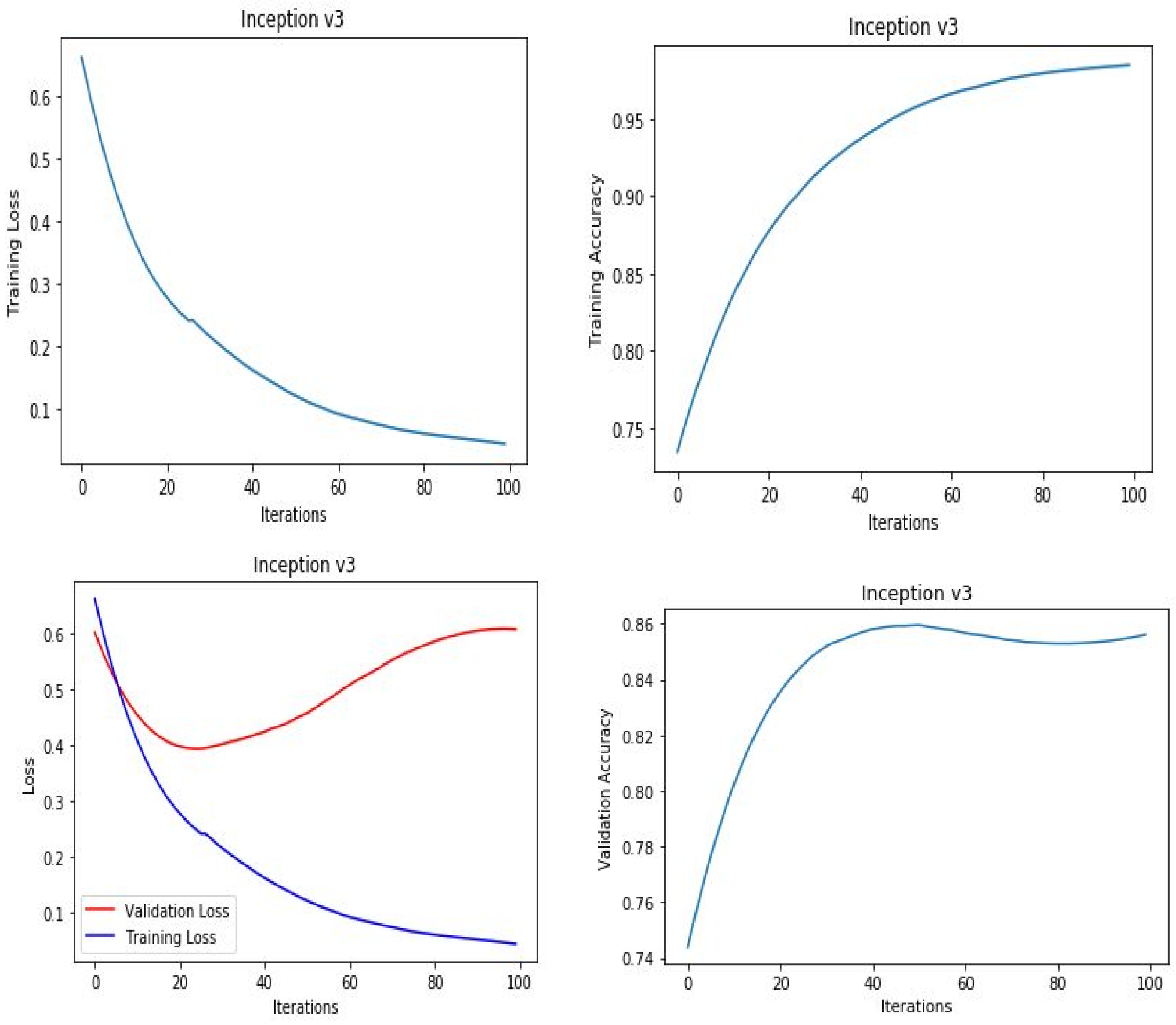
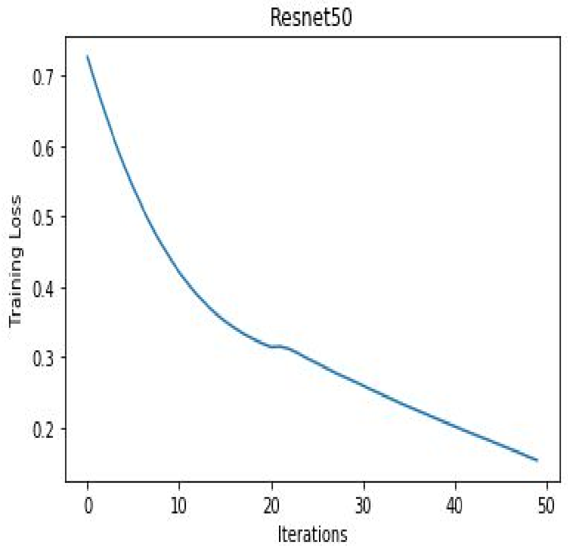
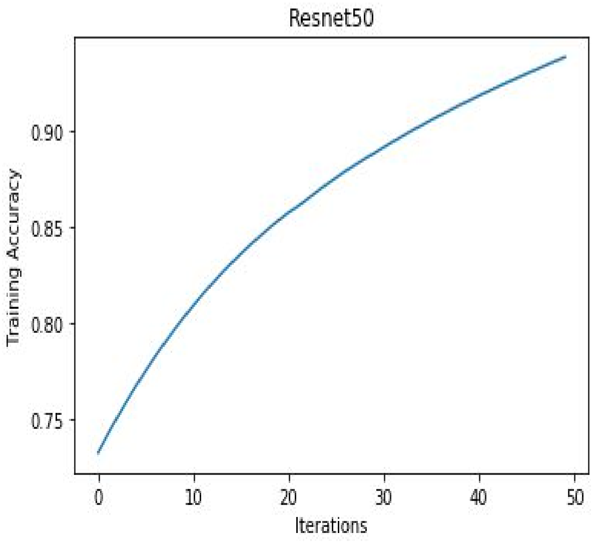


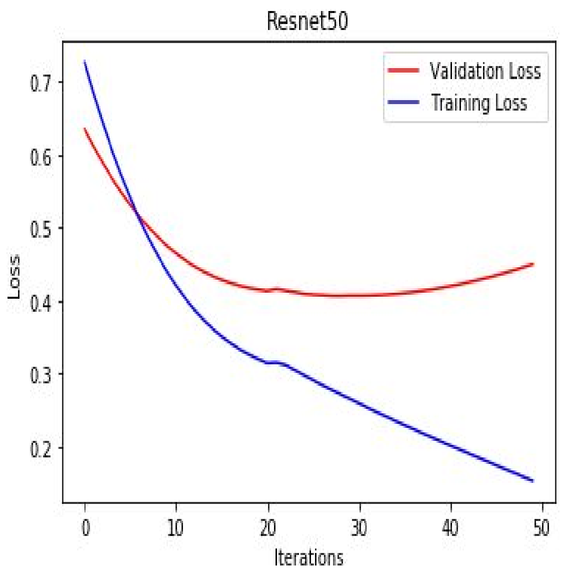
Figure. Graphs for InceptionV3 Training and Cross-Validation

4. Resnet - The Resnet Model is yet another complex model developed by Microsoft

Researchers and contains more than 100 layers. This also required unfreezing most of the layers which are not at the start of the network (only first 20 layers were freezed before fine tuning). With the dataset containing ~7000 training images and ~1600 cross validation images, it achieves an accuracy of 85.33% (Cross Validation Accuracy) and 86.32% (Test Accuracy). The Resnet performance showed similar tendencies as shown by InceptionV3 that it requires a very large dataset to perform better than VGG16 and VGG19 models. The reason is similar that Resnet has many weights to learn and they are learned significantly better when a huge dataset is fed to the system as training data. The following graphs give an overview of how the loss decreases and learning curves help us determine when the model is starting to overfit. Again, here Adam and RMSProp result in faster convergence (Although Adam performs slightly faster).







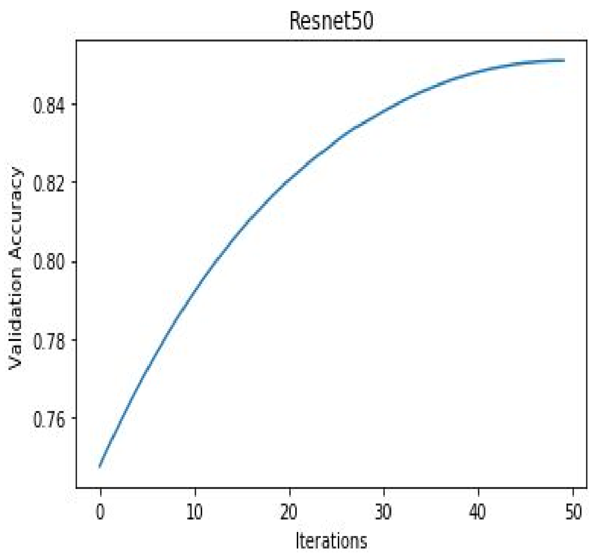


Figure. Graphs for Resnet50 training and validation

CONCLUSION

This project summarizes how different models would behave on a non-curated real-world dataset. It also mentions how data can be cleaned to feed the most well-defined and relevant images to the model so that it does not crash. It also summarizes a comparison of how different models such as VGG16, VGG19, Resnet50 and InceptionV3 perform on real world data and what caveats should be kept in mind when applying these models in a real-world data-critical system. Below is a table depicting the end results for all the models. I found out that complex models are more data-thirsty i.e. their performance is heavily dependent on the amount of data they are provided. When the amount of data is low, comparatively simpler models such as VGG16 and VGG19 would perform slightly better (both in terms of accuracy and lower training time).

	CV Accuracy	Test Accuracy	CV Loss	Test Loss	Training Accuracy	Training Loss
VGG16 (30 epochs)	88.89%	87.16%	0.3023	0.5556	98.10%	0.0534
VGG19 (30 epochs)	89.82%	88.10%	0.2970	0.5225	98.26%	0.0496

Inception V3 (100 epochs)	86.35%	86.07%	0.3867	0.6407	98.47%	0.0476
Resnet50 (50 epochs)	85.33%	86.32%	0.3967	0.4331	93.80%	0.1542

Table. Results for all the models trained and tested

The entire code of the problem can be found at my github portfolio - www.github.com/dhananjay014

The dataset can be downloaded from my drive (only accessible using LionMail) - https://drive.google.com/open?id=0BxR0HkfYA2_ATTV6SnVvS1R3d00

REFERENCES

1. Simonyan,K. & Zisserman,A. (2015), *Very Deep Convolutional Networks For Large-Scale Image Recognition,* ICLR
2. Szegedy, C. et al (2015), *Going Deeper with Convolutions,* CVPR
3. Szegefy, C. et al (2015), *Rethinking the Inception Architecture for Computer Vision, CVPR*
4. He, K. et al (2015), *Deep Residual Learning for Image Recognition*, Microsoft Research
5. Zhou, B. et al,(2014) *Learning Deep Features for Scene Recognition using Places Database,* NIPS, MIT Press
6. CS231n - Convolutional Neural Networks for Visual Recognition, Stanford University
7. www.stackoverflow.com
8. Google Images - images.google.com
9. Keras Python Library- https://keras.io
10. Tensorflow - www.tensorflow.org