Project Impasta

Bharadwaj Sannapaneni
Electrical and Computer Engineering
University of Florida
Gainesville, United States
sannapanenib@ufl.edu

Mukhil Azhagan Mallaiyan Sathiaseelan
Electrical and Computer Engineering
University of Florida
Gainesville, United States
mukhil.mallaiyan@ufl.edu

***Abstract*—Convolutional Neural Networks (convnets) are widely used in performing Image classification. Although they were dormant until recently, use of convolutional neural networks has increased with the increase in the amount of data available through internet, social media and other means, especially as images. In this paper we describe popular architectures and modules that are being used to build the convolutional neural network that can classify and identify the presence of pasta in an image. The data required for the network is obtained and prepared for training. Our aim is to explore different neural network architectures and study them for the purpose of classification and use pre-trained architectures to perform transfer learning for our specific purpose , while minimizing the training time for new problem scenarios.**

Keywords*— *Data Preparation, CNN, Deep Learning, Computer Vision, Impasta

I. Introduction

Image classification is a fundamental task in computer vision. Classifying the image is a basic task in performing object detection, image segmentation etc., Traditionally image classification involves extracting the image features manually and inputting them to a classifier. This process is both arduous and less accurate. The accuracy of the system depends a lot in implementing feature extractor and managing its intricacies, which is a strenuous task [1].

However, due to the enormous amount of image data that is being available through mobile phones, internet and social media, and due to the improvements in the computational power, employing Convolutional Neural Networks (CNN) for image classification is an efficient and feasible choice. CNN is a specialized Neural Network that works particularly well on image data. It has the ability to learn the features hierarchically and perform an accurate classification[2, 3]. CNNs have gained their popularity when Krizhevsky et al., used a deep convolutional neural network in ImageNet Large Scale Visual Recognition Challenge(ILSVRC) 2012 and achieved a winning accuracy by classifying ImageNet data of 1.2 million images into 1000 classes. In this project we aim to build various convolutional neural networks which can classify an image into Pasta image or not with varying performances, we try to both create generalized networks and also pave the way to try to modify pre-trained networks such as VGGnet and Alexnet and use them for our purpose by carefully fine-tuning its critical layers.

II. Convolutional Neural Network

CNNs are typically made of alternating convolutional and pooling layers followed by one or more fully connected layers in the end. Although this is a general structure, different changes in the architectures were proposed in improving the classification accuracy. A typical CNN architecture is shown in *Fig 1*. An input car image is feeded directly to the network, after a series of Convolutional and Pooling layers, the representation is fed to fully connected layer which outputs the label.

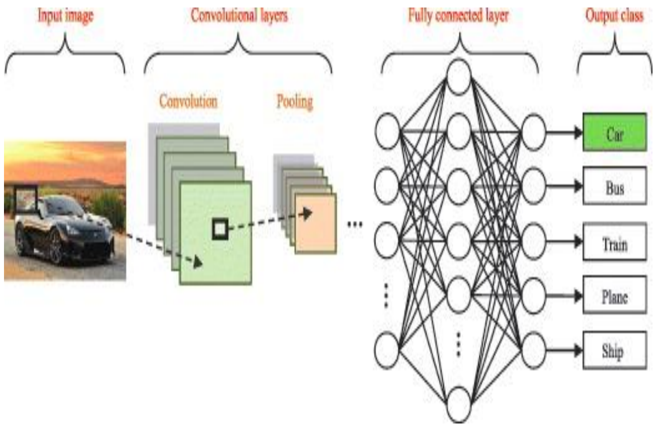


Figure 1: A typical Convolutional Neural Network ([4])

Convolutional layers works as feature extractors, they learn to represent the features of inputs. Inputs of each layer are convolved with the weights producing a feature map. The result is passed through a activation function which is nonlinear. Neurons in a feature map has same weights while different feature maps can have different weights [5]. Suppose if X_j^l is the lth layer's jth feature map in a convolutional layer then,

$$X_{\square}^{l} = f\left(\textstyle\sum_{i \epsilon M_j} X_i^{l-1} * W_{ij}^{l} + b_j^{l}\right) \qquad (1)\ (\textit{from [4]})$$

where, W^l is the weight matrix of lth layer feature map, $b_{\square}^l$ is the bias, f is the nonlinear activation function and $*$ refers to the convolution operation. Rectified Linear Unit (ReLU) is the typically used nonlinear activation function, however Sigmoid functions are also used in few cases.

Pooling layers downsamples the feature maps spatial resolution thereby achieving spatial invariance. These are usually followed by the convolutional layers. There are different types of pooling which includes average-pooling, stochastic pooling and max-pooling. However max-pooling is being used in most of the recent architectures. Max-pooling is given by equation

$$Y_{k,i,j} = \max\ (X_{p,q}) \qquad (2)$$

where $Y_{\square\square\square}$ is the output of pooling of kth feature map, $X_{p,q}$ is the element at location (p,q) in the pooling region. Thus max-pooling layer selects the maximum element in the receptive field.

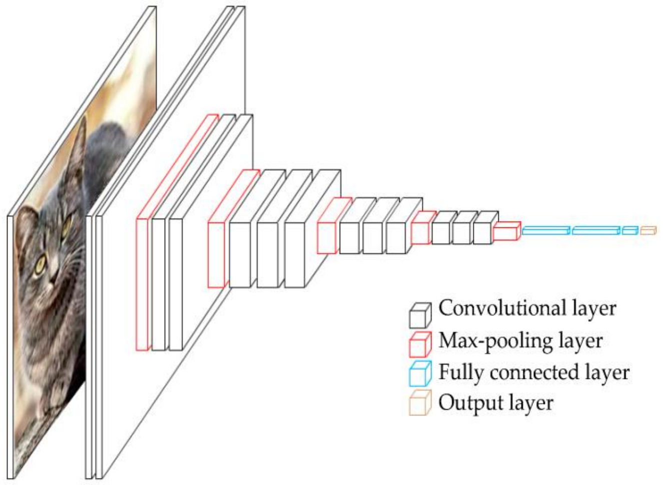


Fig 2: A general overview of a CNN

Convolutional and Pooling layers are stacked alternatively for the required depth, followed by a fully connected layer. In this paper, we will also perform modifications in higher feature convolution layers to retrain an already trained network, called transfer learning. A typical fully connected layer uses softmax function for classification. The output of this layer is given by

$$X^{l}_{\square} = f(W^{l}\, X^{l-1} + b^{l}\ )([4]) \qquad (3)$$

where W^{l} is the weight matrix of lth layer, X^{l-1} is the input feature map, b^{l} is the bias.

The CNN is trained using learning algorithms to obtain the desired results. Backpropagation algorithm is the popular and widely used algorithm for training CNN [6]. It updates and adjusts the weights and biases. The goal is to minimize the error function. Usually Mean Square Error(MSE) is used as a metric. Weights and biases are updated layer by layer as given

$$W \ \ = W \ \ - \eta \frac{\partial E(W,b)}{\partial W} \qquad (4)$$

$$b \ \ = b \ \ - \eta \frac{\partial E(W,b)}{\partial b} \qquad (5)$$

where η is the learning rate, W is weight and b is the bias.

III. Related Work

CNNs were used to perform individual character recognition in 1998 by LeCun et al., This work has emphasized the importance and advantages of CNNs. *Figure 3* shows the architecture used for this CNN. LeCun et al., also introduced a Modified National Institute of Standards and Technology (MNIST) data set which has around seventy thousand handwritten digits. The dataset provided is extensively used for many computer vision tasks and problems.

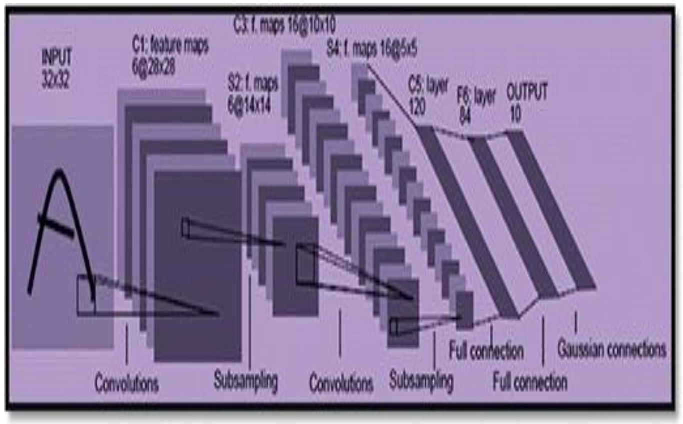


Figure.3 CNN architecture used by LeCun et al.,

Many CNN architectures have been developed since early 21st century. AlexNet [7] proposed by Krizhevsky et al., is among the popular CNN architectures. AlexNet is similar but deeper than LeNet. The Deep Neural network created by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton achieved a top 5 test error rate of 15.4% on ImageNet dataset. The network has 5 convolutional layer, pooling layers, dropout layers and three fully connected layers. The success of AlexNet has motivated many other deep convolutional neural network architectures including ZF Net [8], VGG Net [9], GoogleNet [10] and ResNet [11].

ZF Net built by Matthew Zeiler et al., has achieved a winning error rate of 11.2% in 2013 ILSVRC competition. Zeiler et al., fine tuned the AlexNet and gave a good intuition about the architecture. VGG Net is created by Karen Simonyan et al., which achieved 7.3% error rate. This architecture has 19 layers. GoogleNet is a 22 layer architecture which won 2014 ILSVRC with top 5 error rate of 6.7%. ResNet architecture created by Microsoft has won ILSVRC 2015 with an top-5 error rate of 3.6%. This is one of the deepest architecture with 152 layers.

IV. Methods and Architectures

In this project our aim is to classify the image data into Pasta Images and Non-Pasta images. The required data is obtained from ImageNet. We have obtained a data around 2000 images. Our aim is to minimize the amount of training images used and yet achieve comparable results. Obtained data is modified, augmented and prepared of the training with Neural Network.We would also like to show how to reduce training time by modifying previously trained networks to an problem statement and achieve comparable results

As mentioned the data obtained will be trained on different architectures including AlexNet, VGG Net and our own generalized network. The results from these different architectures will be analyzed and presented. Figure *4* represents the block diagram of our work in this project. The initial block diagram was modified for lack of time and to reduce deviation from subject matter. Instead of working on a website to deploy the project , we have decided to work on a mode of Transfer Learning for our problem.

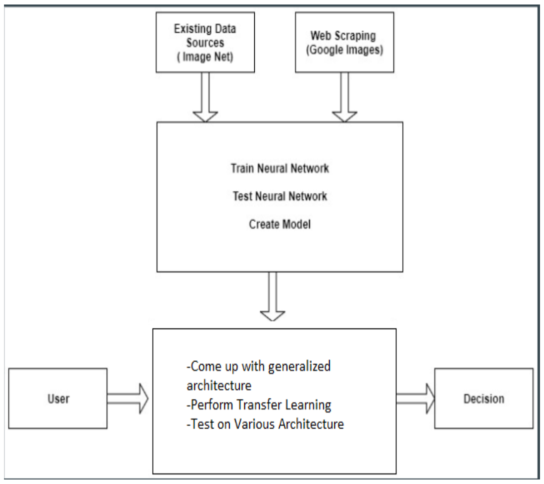

Existing Data Sources (Image Net)
Web Scraping (Google Images)
Train Neural Network
Test Neural Network
Create Model
User
-Come up with generalized architecture
-Perform Transfer Learning
-Test on Various Architecture
Decision

Fig4. Block diagram of Project Impasta

A. AlexNet

This architecture developed by Krizhevsky et al., has five convolutional layers, three fully connected layers and a softmax. Figure 5 shows the architecture used. In our project we will train the model with a momentum of 0.9 and weight decay of 0.0005. Although we start with these parameters we will vary and experiment different hyperparameter values.

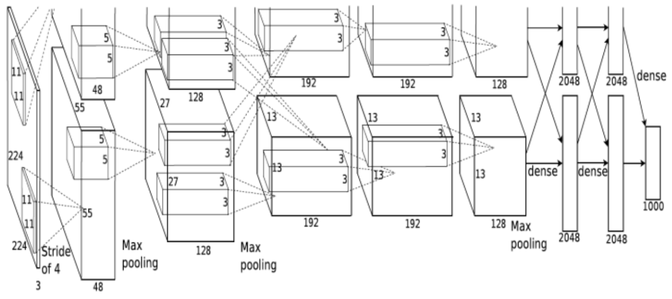


Fig5. AlexNet Architecture (from [7])

B. Generalized network

The architecture was developed by us to aid in generalizing the network for various problems , this was done by incorporating ideas from the various networks . It features 15 layers overall , including activation layers and drop out layers. 3 Convolution layers along with max pooling and activation for each conv layers , followed by a fully connected network supported by a sigmoid activation layer.

The problem of food classification is different from classification of cats and dogs or cars and bikes, these have distinct differences between them. While food classification has a problem of having a lot more features and thus will need a varied architecture and lot more training samples .

We propose a method to train our network in about 30 minutes with about 1000 images of pasta and 1000 other images. Please visit the code for more details[].

C. VGG Net

VGG Net built by Karen Simonyan et al., has 19 layers and has the architecture as shown in the Fig 6. This network uses 3 x 3 filters which is very different from its ancestors AlexNet or ZF Net. It uses combination of 3 x 3 convolutional layers which gives the advantages of having small filters while being as effective as a large filter. Combination of two 3 x 3 layers is as effective as a 5 x 5 layer and the combination of three 3 x 3 gives a field of 7 x 7 layer. This architecture increases the depth of the network while keeping the spatial dimensions less. This aspect of the paper proved that the network depth plays an important part in the performance of the network. We can say that VGG is a Homogenous network, which uses only 3 x3 layers for convolutions and 2 x 2 layers for pooling. 3 x 3 convolutions are done with a stride of 1 and padding of 1. 2 x 2 max pooling is done with a stride of 2 and with no pad.

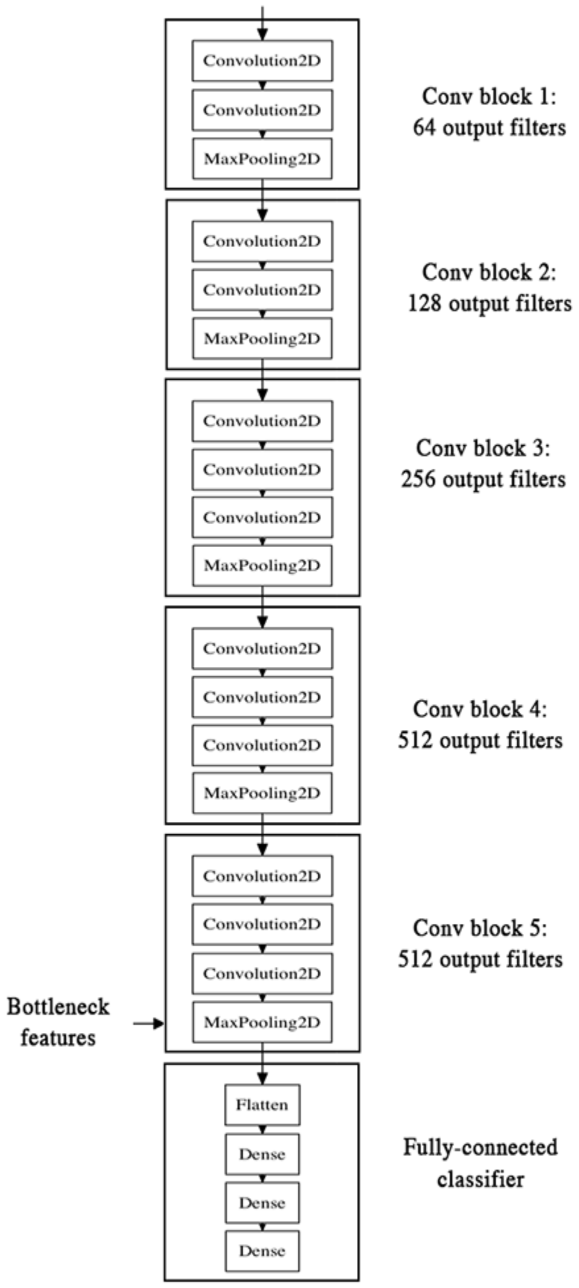


Fig 6. VGG Net Architecture

An important drawback of the VGG model is that it shows high accuracy for these images as they belong to the imagenet dataset ,and perform poorly when tested with images from other sources , to overcome this we train the network with a varied dataset especially for pasta and non pasta images , This will reduce overtraining and fit general datasets well. These features however act as a bottleneck for training other images and datasets , thus these form a bottleneck and are thus called bottleneck features

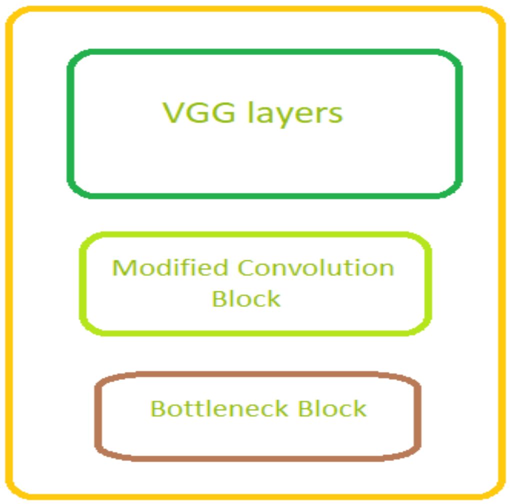


Figure 7: Fine-Tuned Architecture

This VGG model is retained and we attempt transfer learning on VGG to train with our own dataset of pasta and non pasta . For this we employ a method of freezing the lower layers and activating and training the fully connected layer and the last convolutional block. To achieve this , we are training the network slowly with more exploration space , with a slower learning rate , and we also use and SGD optimization function. Finding the balance is important because we want to also maintain the correlation between the frozen layers and the fine-tuned model on top. The bottom layers only identify general features and not much of image specific information , and the VGG having trained on Imagenet , it has all the information for various features possible , this is an advantage that has been leveraged to this approach of transfer learning.

V. EXPERIMENT DETAILS

The training is done using two frameworks. We are using Python Caffe to build and train one of our Convolutional Neural Networks , While we are making use of keras running on a tensorflow backend to train our generalized model and VGG and to achieve transfer learning. Although one of our laptop is running 4GB Nvidia GTX 1050-Ti which can help for faster computations we still need more computational power as it is taking very long time, while still making use of CuDNN. As mentioned earlier around 2000 images of data is downloaded and prepared for training.

Accuracy and error rates will be used as performance metrics.Visualization will be provided using class confusion matrices and the plots of accuracy and loss.

Our CNN architecture will produce true or false result stating whether the given image is Pasta or not. We can improve this by adding or modifying softmax layer to classify into more classes. For example we can detect the type of pasta. There are various Kaggle Contests on Image classification we can participate with some modifications in architecture. We are also planning to build a web interface and make it available on internet. Interface will let users to upload an image and perform classification.

VI. RESULTS

A. *Discussion on Generalized model*

As mentioned in Section III B. the network focuses on speed and generalization.

Over 50 epochs it was noticed that the generalized model was able to classify with reasonable accuracy the images , while slowly decreasing in loss as seen in, This serves the purpose of training all the neurons with the same amount of loss and to not focus training over a specialized area. The classification of food has much more distinguishable features , from color to shape to texture and consistency . This has to be captured . Thus the model performs to capture most of them to a accuracy of about 70 percent . This minor tradeoff in accuracy gives a boost in classifying complex images with many features.

Table 1 shows the class confusion matrix of test results from out generalized network. Positive column indicates that the image is classified as pasta and the Negative column indicates that images are classified as Non-Pasta. Total 160 images, 80 pasta and 80 Non-pasta images are used for testing the networks. This network classified 50 Non-Pasta images correctly as Non-Pasta and the rest 30 as Pasta. 68 of the Pasta Images are classified correctly as Pasta while the rest 12 are classified incorrectly. The overall accuracy of the this network over the test data is 85%. It has to be noted that these images were not from the imagenet database. While vanilla VGG net performed poorly at 56% on images taken at random. This is a good increase in accuracy as well as reinforces the importance of the fine tuning

TABLE 1: GENERALIZED MODEL CLASS CONFUSION MATRIX

Class Confusion Matrix	ImPasta	Pasta
ImPasta	50	30
Pasta	12	68

Figure 8 shows cases of failure of the generalized network , these are due to images that resemble pasta . The worst performance was in the middle image , where it does not resemble pasta , but it was still classified as a pasta . This could be a corner case . While the other cases are explainable . The image to the left was noticed to have an out of focus image that had poorly defined features , in addition to the mix of various vegetables in it. This caused an prediction accuracy of 34 percent ,and was rounded off to be non pasta. While the image on the right is a form of rice and resembles pasta to a very high extent both in color and form . This could be a reason for such wrongful classification.

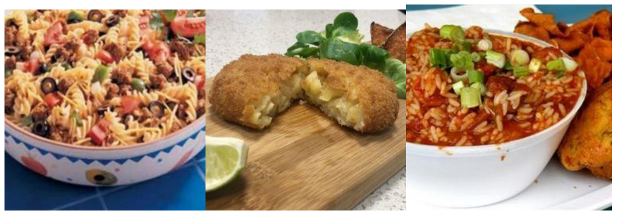


Fig 8: Cases of Failure: Classified as not pasta(left), classified as pasta(middle and right).

B. *Discussion on VGG net*

VGG net was initially trained on a dataset from Imagenet , which has an extremely large dataset and also contains various categories, This network would have just learnt various features of images with wide amount of variations. This is a good network for transfer learning and also for generalization .

This network is used as a base and we just include a fully connected layer on top of it to utilize what it has already learnt. Fig 10 and 11 show the performance of using pretrained VGG weights and when trained on images for the final layers alone, while this gives a high amount of accuracy , it was noticed to display high amounts of overtraining. This image worked and trained well on our training dataset , but when tested on a few images that were obtained from elsewhere it performed poorly. This is clearly due to using already trained features that poorly correlate to the given problem .

We sort to a fine-tuning strategy as discussed in Section III and Fig 7 . Where the final convolutional block is retrained for the given problem .

Over 50 iterations and using an SGD optimizer of a learning rate of 1e-4 ,and a momentum of 0.9 . This slowly varying the problem specific features of the final convolutional block enables better performance on pasta images specifically . This use of a pre-trained network to successfully apply it to another similar problem is called Transfer learning . Apart from this , we not only want to achieve accuracy , but also want faster training on images the network has seldom seen . The network runs in 25 minutes and works on the given problem.

Table 2 shows the class confusion matrix of test results from Fine Tuned VGG Network. The same images 160 used for generalized network are used again for fine tuned VGG network. This network classified 54 Non-Pasta images correctly as Non-Pasta and the rest 26 as Non-Pasta. 67 of the Pasta images are classified correctly as Pasta while the rest 13 are classified incorrectly. We can say that the overall accuracy of trained and Fine Tuned VGG network is 83.5%, while its negative identification accuracy increases to 67.5%.

TABLE2: FINE TUNED VGG MODEL CLASS CONFUSION MATRIX

Class Confusion Matrix	ImPasta	Pasta
ImPasta	54	26
Pasta	13	67

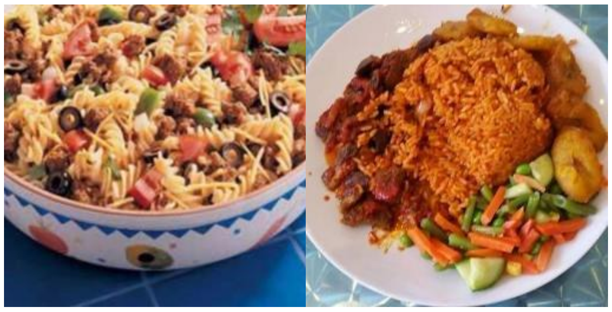
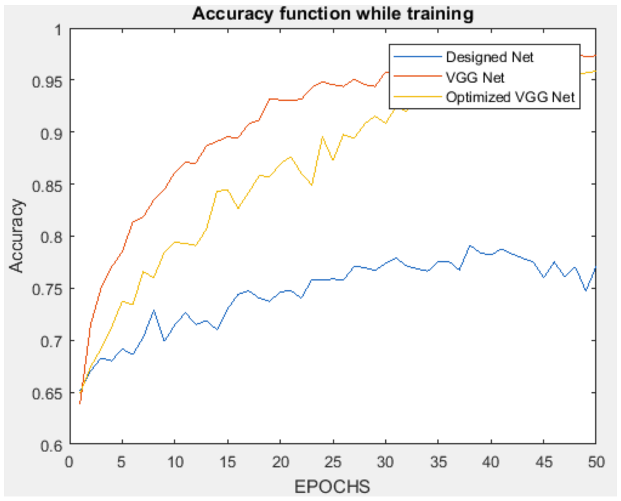


Fig 9: Cases of Failure Classified as not pasta(left), Classified as pasta(right) on Fine-tuned VGG.

The images above provide examples of where the network fails , the image to the left has shown consistency mistakes for both the generalized network and also the fine-tuned VGG . the image on the left however is a non-pasta image of rice , that is classified as pasta . This is due to the similarity in color and texture of the images to pasta. This can be further improved by more dataset , but that would be against the problem formulation for this project .



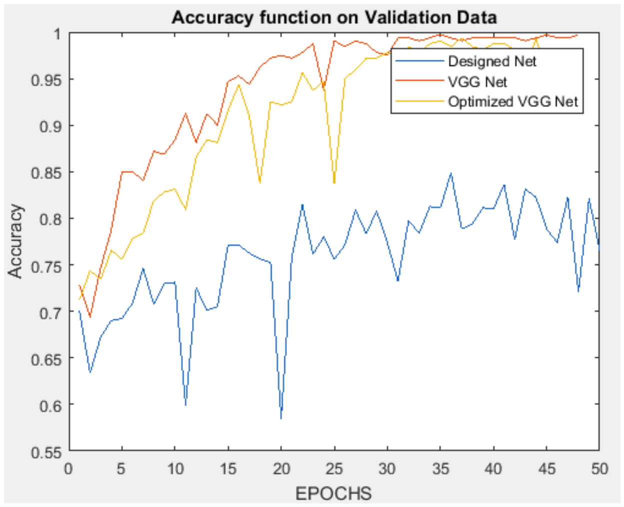


Fig 10: A comparison of Accuracy of VGG , fine-tuned VGG and our generalized model.

Both VGG and optimized VGG give more than 95 percent accuracy as seen from Fig 10., but this time taken for training is very expensive in both computation and space. , We wish to introduce our general model that is able to run in 20 to 30 minutes and produces results of about 80 percent as can be seen from the graph . This general model will work for classification of various such images as well .

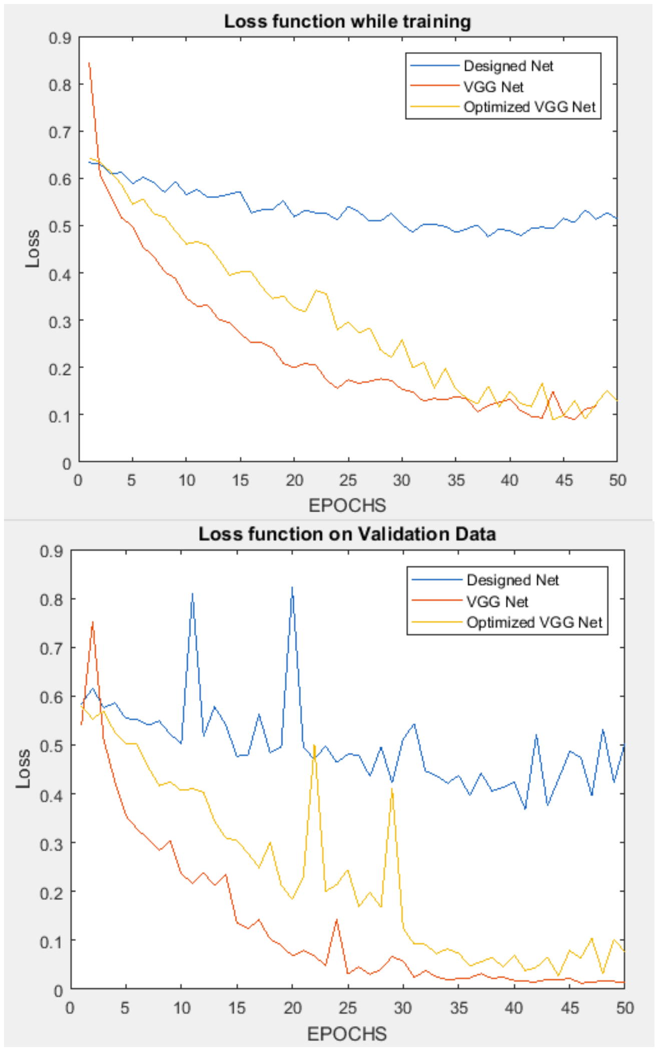
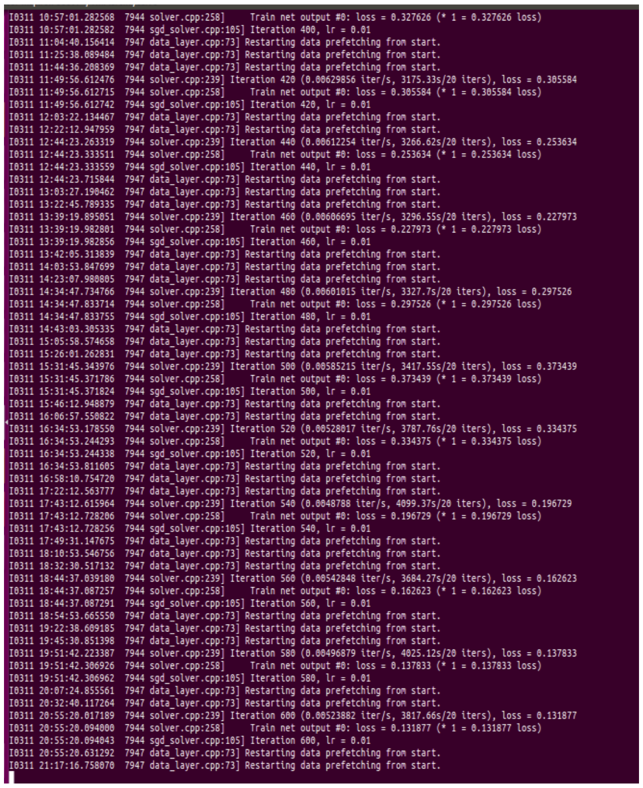


Fig11 : A comparison of loss across Generalized model , VGG and VGG fine-tuned(optimized)

C. Discussion on Alex Net

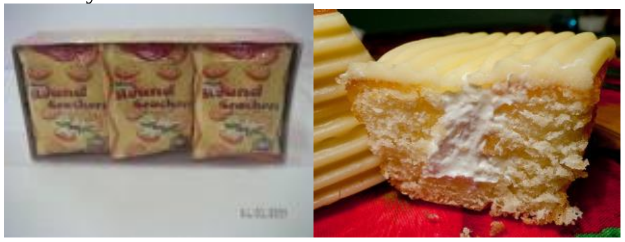
The downloaded data of 5000 images from ImageNet is modified for training. We renamed the downloaded images and labeled them accordingly. We have also created '.txt' files for inputting them to the Python Caffe. Data is also converted to LMDB format. Mean image is computed for the Caffe usage.



```
I0311 10:57:01.282568  7944 solver.cpp:258]     Train net output #0: loss = 0.327626 (* 1 = 0.327626 loss)
I0311 10:57:01.282582  7944 sgd_solver.cpp:105] Iteration 400, lr = 0.01
I0311 11:04:40.156414  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 11:25:38.089484  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 11:44:36.208369  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 11:49:56.612476  7944 solver.cpp:239] Iteration 420 (0.00629856 iter/s, 3175.33s/20 iters), loss = 0.305584
I0311 11:49:56.612715  7944 solver.cpp:258]     Train net output #0: loss = 0.305584 (* 1 = 0.305584 loss)
I0311 11:49:56.612742  7944 sgd_solver.cpp:105] Iteration 420, lr = 0.01
I0311 12:03:22.134467  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 12:22:12.947959  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 12:44:23.263319  7944 solver.cpp:239] Iteration 440 (0.00612254 iter/s, 3266.62s/20 iters), loss = 0.253634
I0311 12:44:23.333511  7944 solver.cpp:258]     Train net output #0: loss = 0.253634 (* 1 = 0.253634 loss)
I0311 12:44:23.333559  7944 sgd_solver.cpp:105] Iteration 440, lr = 0.01
I0311 12:44:23.715844  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 13:03:27.190462  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 13:22:45.789335  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 13:39:19.895051  7944 solver.cpp:239] Iteration 460 (0.00606695 iter/s, 3296.55s/20 iters), loss = 0.227973
I0311 13:39:19.982801  7944 solver.cpp:258]     Train net output #0: loss = 0.227973 (* 1 = 0.227973 loss)
I0311 13:39:19.982856  7944 sgd_solver.cpp:105] Iteration 460, lr = 0.01
I0311 13:42:05.313839  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 14:03:53.847699  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 14:23:07.980805  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 14:34:47.734766  7944 solver.cpp:239] Iteration 480 (0.00601015 iter/s, 3327.7s/20 iters), loss = 0.297526
I0311 14:34:47.833714  7944 solver.cpp:258]     Train net output #0: loss = 0.297526 (* 1 = 0.297526 loss)
I0311 14:34:47.833755  7944 sgd_solver.cpp:105] Iteration 480, lr = 0.01
I0311 14:43:03.305335  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 15:05:58.574658  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 15:26:01.262831  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 15:31:45.343976  7944 solver.cpp:239] Iteration 500 (0.00585215 iter/s, 3417.55s/20 iters), loss = 0.373439
I0311 15:31:45.371786  7944 solver.cpp:258]     Train net output #0: loss = 0.373439 (* 1 = 0.373439 loss)
I0311 15:31:45.371824  7944 sgd_solver.cpp:105] Iteration 500, lr = 0.01
I0311 15:46:12.948879  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 16:06:57.550822  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 16:34:53.178550  7944 solver.cpp:239] Iteration 520 (0.00528017 iter/s, 3787.76s/20 iters), loss = 0.334375
I0311 16:34:53.244293  7944 solver.cpp:258]     Train net output #0: loss = 0.334375 (* 1 = 0.334375 loss)
I0311 16:34:53.244338  7944 sgd_solver.cpp:105] Iteration 520, lr = 0.01
I0311 16:34:53.811605  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 16:58:10.754720  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 17:22:12.563777  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 17:43:12.615964  7944 solver.cpp:239] Iteration 540 (0.0048788 iter/s, 4099.37s/20 iters), loss = 0.196729
I0311 17:43:12.728206  7944 solver.cpp:258]     Train net output #0: loss = 0.196729 (* 1 = 0.196729 loss)
I0311 17:43:12.728256  7944 sgd_solver.cpp:105] Iteration 540, lr = 0.01
I0311 17:49:31.147675  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 18:10:53.546756  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 18:32:30.517132  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 18:44:37.039180  7944 solver.cpp:239] Iteration 560 (0.00542848 iter/s, 3684.27s/20 iters), loss = 0.162623
I0311 18:44:37.087257  7944 solver.cpp:258]     Train net output #0: loss = 0.162623 (* 1 = 0.162623 loss)
I0311 18:44:37.087291  7944 sgd_solver.cpp:105] Iteration 560, lr = 0.01
I0311 18:54:53.665550  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 19:22:38.609185  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 19:45:30.851398  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 19:51:42.223387  7944 solver.cpp:239] Iteration 580 (0.00496879 iter/s, 4025.12s/20 iters), loss = 0.137833
I0311 19:51:42.306926  7944 solver.cpp:258]     Train net output #0: loss = 0.137833 (* 1 = 0.137833 loss)
I0311 19:51:42.306962  7944 sgd_solver.cpp:105] Iteration 580, lr = 0.01
I0311 20:07:24.855561  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 20:32:40.117264  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 20:55:20.017189  7944 solver.cpp:239] Iteration 600 (0.00523882 iter/s, 3817.66s/20 iters), loss = 0.131877
I0311 20:55:20.094000  7944 solver.cpp:258]     Train net output #0: loss = 0.131877 (* 1 = 0.131877 loss)
I0311 20:55:20.094043  7944 sgd_solver.cpp:105] Iteration 600, lr = 0.01
I0311 20:55:20.631292  7947 data_layer.cpp:73] Restarting data prefetching from start.
I0311 21:17:16.758070  7947 data_layer.cpp:73] Restarting data prefetching from start.
```

Fig. 12: Screenshot of AlexNet Getting Trained

As mentioned AlexNet was trained using Caffe. The network was trained for two days on a computer that does not have GPU. The network was trained for 846 iterations and it is terminated manually as it has reached a decent training accuracy. The trained network was tested on the same 160 image set that was used previously. The network has achieved an accuracy of 68.75 % on the test data. The *Figure 13* shows the images that AlexNet misclassified. The first row of images are the Non-Pasta images but are classified as Pasta and the second row shows the Pasta Images that are classified as Non-Pasta. The reduced accuracy of the AlexNet is probably because that the trained Neural network lost its generalization. The test accuracy has reduced significantly from the training accuracy. It fell down from 86.24% to 68.75 %.



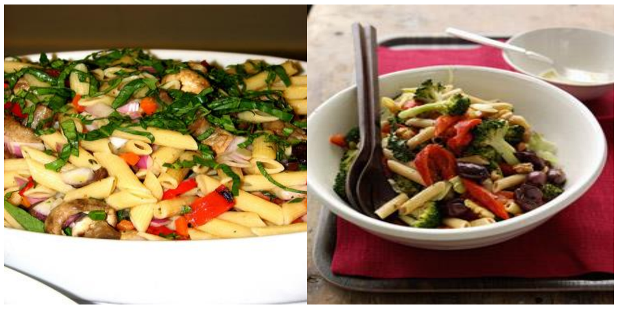


Fig 13: Cases of Failure, Classified wrongly as pasta(top), Classified as not pasta(bottom)

Figure 14 shows the loss function of the AlexNet while it was getting trained. X axis shows the number of Epochs and the Y axis shows the loss of the network.

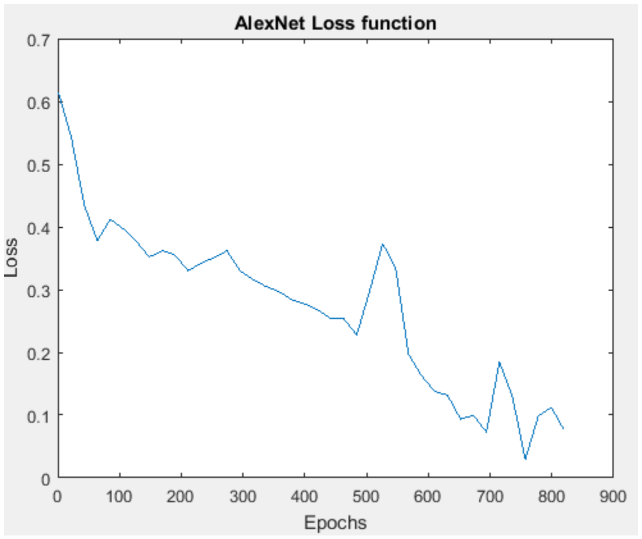


Fig 14: Graph of AlexNet Loss function vs Number of Epochs

VII. Conclusion

We were able to train the models successfully and achieve a decent accuracy in all cases. The main goal of this project has been to understand and work with various Convolutional Neural Network architectures used for image classification problem and come up with a generalized network for any image classification. A secondary goal was to achieve transfer learning using a popular pretrained network. While still maintaining importance for a speedy training process . Except for Alex net , which was an experiment to force it to work on a smaller number of images still took a long time, the other cases of VGG and fine-tuned VGG was able to perform very well with 25 minutes of training. As expected the results of fine tuned VGG are better than the normal VGG better and also performs well on generalization, and AlexNet performs better than general CNN, even on a short training period. The major drawback is with the computational resources available in case of Alexet which is exactly the problem we overcame incase of the fine-tuned VGG.

Table 3 was our initial Schedule and milestones to cover We have hit most of our deadlines. Some of the problems we faced are the change in Google's terms and conditions for downloading images from images.google.com, that affected our plan to write a web crawler to download images, We decided to stick to images on Imagenet and manually download about 160 images for test scenarios. Another problem was the lack of computational resources , which stopped more experimentation on various architectures , to overcome this we had to be smart and modify VGG and bringdown the training time. Training on MNIST was redundant , it was a common problem and we did not want to work on that much . We adopted the problem of training the network on food images , pasta especially to experiment and present results on the performance of CNNs on various textures and high frequency variations in images. Food is always a mixture of various smaller problems and we wanted to generalize our network for this case and we achieved it with reasonable accuracy.

TABLE 3: MILESTONES

Date	**Weekly Schedule**		
	Goal	***Week***	***Miss/Hit***
2-5-18	Project Proposal	0	Hit
2-15-18	Read Article and Papers	1	Hit
2-22-18	Obtain Images	2	Hit
3-1-18	Finalize Architecture	3	Hit
3-9-18	Train MNIST	4	Changed for redundancy
3-12-18	Midterm	5	Hit
3-30-18	Train Model	5,6,7	Hit
4-9-18	Website	8	Changed to accomodate Transfer Learning
4-23-18	Final Submission	10	Completed

VIII. Future Work

Future work will be in the form of coming up with better accuracy for the models. As shown, the classification of food is a complex problem and requires capturing of various important features . Given much more powerful computational resources , it is possible to further finetune the VGG model , it is also possible to train AlexNet to a further extent. One could also do an analysis on the effect of modifying the optimization and training algorithm on our generalized network to suit the problem of food classification. The major identifying feature for our network seems to the color of the image , and a few distinguishing shapes , further analysis can be done using [12] and find the corner cases and better train the network.

REFERENCES

[1] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.

[2] A.S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, CNN Features Off-the-Shelf: An Astounding Baseline for Recognition, 512-519, 2014.

[3] H. Kataoka, K. Iwata and Y. Satoh, DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition, Computer Science, 50(1): 815-830, 2013.

[4] Rawat, Waseem, and Zenghui Wang. "Deep convolutional neural networks for image classification: A comprehensive review." Neural computation 29.9 (2017): 2352-2449.

[5] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436–444.

[6] Srinivas, S., Sarvadevabhatla, R. K., Mopuri, K. R., Prabhu, N., Kruthiventi, S. S., & Babu, R. V. (2016). *A taxonomy of deep convolutional neural nets for computer vision*. arXiv 1601.06615.

[7] Krizhevsky, A., Sutskever, I., and Hinton, G.E. Imagenet classification with deep convolutional neural networks. In NIPS, 2012

[8] Zeiler, M. D. and Fergus, R. Visualizing and understanding convolutional networks. CoRR, abs/1311.2901, 2013. Published in Proc. ECCV, 2014

[9] K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, Computer Science, 2014.

[10] Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015

[11] Canziani, Alfredo, Adam Paszke, and Eugenio Culurciello. "An analysis of deep neural network models for practical applications."arXiv preprint arXIv:1605.07678 (2016).

[12] Kexin Pei, Yinzhi Cao, Junfeng Yang, Suman Jana,"DeepXplore: Automated Whitebox Testing of Deep Learning Systems",arXiv preprint arXiv:1705.06640.