COMP 551 Applied Machine Learning
Project Report: Image Classifier
Team name: Neural Network For Harambe

Taanvi Chhetri
taanvi.chhetri@mail.mcgill.ca

Arun Rawlani
arun.rawlani@mail.mcgill.ca

Yuanhang Yang
yuanhang.yang@mail.mcgill.ca

***Abstract*— We demonstrate the working of different methods, namely Logistic Regression, Feedforward Neural Network, Convolutional Neural Network, in classifying images from a small subset of ImageNet database. We comment on the effectiveness of all employed methods. Our tests find that Inception v3 CNN has the highest validation accuracy of 88.00% among these methods, and scored 88.00% on Kaggle competition.**

I. INTRODUCTION

This paper demonstrates our work in classifying images from a small subset of ImageNet database, with only 26,334 training images from 40 categories.A baseline logistic regression classifier and a fully-connected feedforward neural network were used by our team. Furthermore, we used library for an implementation of convolutional neural network using TF-learn. Finally, we used transfer learning on the pre-trained Inception-v3 model which yielded the highest accuracy of 88% on Kaggle. Our highest accuracies for logistic regression is 23.2%, as shown by Kaggle. For feedforward NN, validation on reserved training data showed an accuracy of 29.56%. Discussion of these three approaches is provided in Section VI.

II. RELATED WORK

A. Logistic regression

Logistic regression is very challenging to perform on large-scale data sets, especially when the feature space is of high dimension: defining the hyper-planes to split up a large number of data points can naturally be hard. An interesting way around it was presented in a paper [1] talking about optimizing multiclass logistic regression (MLR) by using stochastic gradient descent, and the authors also noted a way to reduce the cost by taking advantage of the structure of the gradient matrix. But since logistic regression is not the main topic on the project, we decided to focus our energy on other approaches and simply use regular logistic regression techniques.

B. Feedforward Neural Networks

For the last few years, Artificial Neural Networks (ANN) has been an area of a lot of research, especially in terms of its image classification abilities. This is due to the imperative impact it has in a myriad of fields including Cancer and Alzheimer's disease research. The inherent structure of ANN allows the algorithm to process data that is generally characterized to be noisy in high dimension, complex, imprecise, or error-prone[2]. The reason that ANNs are very powerful against classification problems is that, they are able to deal with many technical problems, one of the main being the "Curse of Dimensionality"[3]. Implementation of an ANN can be done in many different ways[4]. We follow the backpropagation method in this paper which is the most common method of training multi-layer networks[5].

C. Convolutional Neural Networks

Simple Neural Networks cannot capture the hierarchical structure of an image. This makes it difficult for the neural network to be unaffected by transformations and deformations in images. Consequently, we have to incorporate the concept of translational invariance in our neural network and using convolutions is one efficient way to do it. [6]

Recently, convolutional neural networks have achieved unprecedented levels of accuracy for image classification on large benchmark data sets such as ImageNet. [7]

Domain adaption of learned classifiers is also an effective strategy that can be employed to avoid the large data sets and high computational resources required to train deep convolutional neural networks from scratch. The notion of transfer learning has been researched as part of domain adaption. It was employed by Blitzer et al. who adapted sentiment classifiers originally trained on book reviews to reviews for kitchen appliances[8]. Similar work has been done for adapting visual classifiers to new image data which has yielded positive results [9]. Consequently, domain adaption has emerged as an effective strategy to use models trained on large data sets and then transfer the model's object knowledge to classify on novel domains.

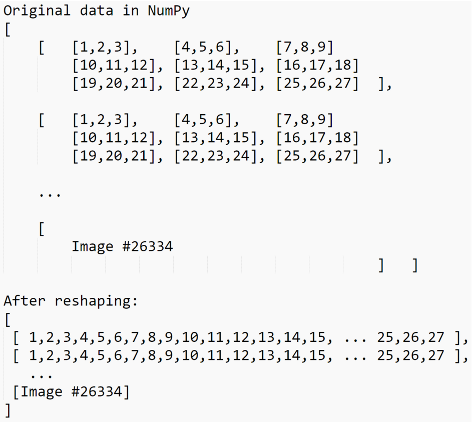
III. PROBLEM REPRESENTATION

The data provided to us was an array of images, encoded into an NumPy array of dimension $26344 * 3 * 64 * 64$, where each value represents a pixel's RGB values for each of the 26,344 images (64 * 64 pixels). Data pre-processing and feature design were handled differently for the three techniques.

A. Logistic regression

For the logistic regression learner, the large NumPy array is not directly usable because it expects 1- or 2-dimensional arrays as input. We reshaped the training and testing image data from a 4-dimensional array into a 2-dimensional array.

This is done by simply concatenating all rows of an image matrix into a single row, hence representing each image with a 1D array. An illustration is provided in figure 1.



```
Original data in NumPy
[
    [   [1,2,3],    [4,5,6],    [7,8,9]
        [10,11,12], [13,14,15], [16,17,18]
        [19,20,21], [22,23,24], [25,26,27]  ],

    [   [1,2,3],    [4,5,6],    [7,8,9]
        [10,11,12], [13,14,15], [16,17,18]
        [19,20,21], [22,23,24], [25,26,27]  ],

    ...

    [
        Image #26334
                                            ]   ]

After reshaping:
[
 [ 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15, ... 25,26,27 ],
 [ 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15, ... 25,26,27 ],
   ...
 [Image #26334]
]
```

Fig. 1. Illustration of data reshaping for logistic regression. Showing 3x3 pixels images as example, but real data is 64x64

Due to time constraints, we didn't focus on feature selection for logistic regression: even with only 75% training data (about 19,750 images), the running time of our script on the best virtual machine we could afford was about 11 hours, making cross-validation over different feature combinations impractical. Besides, we spent more time on neural networks methods because we expect them to suit the occasion better than logistic regression. For these reasons, we simply used all information given in the data set, rather than handpicking features to use.

B. Feedforward Neural Network

In order to optimize the algorithm the data was reconstructed to create new data structures. The reshaping of each image is similar to what was done in Logistic Regression, as shown in figure 1. However, while storing the information, each image is stored as a column vector and not a row vector.

A list of 2 entry tuples was created to contain information about the image and the class it belongs to. The list looks like: [(x,y)].

1) **x**: Each x represents 1 image. Each image is represented by pixels. To structurally denote this, we created a numpy ndarray of size 'n' (changes depending on training or validation). Each element of this array is a numpy ndarray of size 12288 (64*64*3) which is the number of pixels in each image.
2) **y**: y is an array of numbers where each element is a number between 0-39. This represents the class of the image. The size of y also depends on whether we are dealing with the training or validation set.

C. Convolutional Neural Networks

1) TFLearn: For the implementation of Convolutional Neural Networks using TFLearn, we first zero-centered the data and then normalized it. Even though the relative scales of image pixels are approximately equal, normalizing the values can still help us make the data consistent in the scenario where the images on the ImageNet were taken from different sources with varied RGB scaling.

Data augmentation was done by adding random left and right flips to the images. Furthermore, random rotations were added to the images to further augment the data on which the model can be trained on.

2) Inception v3: For fine-tuning of Inception-v3 model, the images were initially re-sized to a size of 299 * 299, by inserting padding, as the pre-training on inception-v3 has been performed using that size. However, with no significant increase in accuracy the images were restored to their original size of 64 * 64 as the computation with smaller images was relatively faster. Yet, additional images were added from ImageNet data set for categories producing the highest number of misclassifications, which were then resized to 64 * 64 to be consistent with the previous data set.

IV. ALGORITHM SELECTION AND IMPLEMENTATION

A. Logistic regression

Scikit-learn's $linear\_model.LogisticRegression$ model was used to perform the logistic regression learning. Our work was mostly around pre-processing the image data (to fit the model's requirement the input data's dimension had to be at most 2 - see section III-A), and producing testing/validation results (confusion matrix). For validation, we reserved 25% of the data for validation, and arrived at the results presented in section V-A.

B. Feedforward Neural Networks

Based on related readings and previous exposure to Machine Learning we know that often the process of choosing optimal hyper-parameters is empirical and heuristic. We decided to optimize the FNN with a simple, naive cross-validation process to prevent an exhaustive search of optimal parameters. We tried a few arbitrary combinations, picked the ones with best empirical results, and "locally" refined the values based on the results.

Initially, we compared the difference between a single hidden layer and multiple hidden layers other hyper-parameters *ceteris paribus*. In this we discovered that increasing the number of layers doesn't improve accuracy, so we only used a single hidden layer FNN. With more computational power, we definitely could look into improvements with multi-layer FNNs.

Using our naive cross validator, we set a few possible values for each of the hyper-parameters, and tested their effect on accuracy and performance. To start out tried to find a subset of parameters that gave us a better result than the random predictor - 2.5%.[10]

Our algorithm is a simple Feedforward and Backpropagation supervised learning classifier. Understanding and implementing the appropriate mathematical functions played an important role in optimizing our model.

We chose to work with sigmoid neurons as they output a value between 0 and 1. This allows us to modify the weights and biases, during each epoch, to optimize the classification. Modifications to the weights and biases were done by the following equations:

- **Activation function**: $\sigma(wx + b)$
- **Sigmoid function**: $\sigma(z) \equiv \frac{1}{1+e^{-z}}$

A way to test our model's efficiency we need to minimize the input functions. In order to do so, the gradient descent must decay to the global minima. For efficiency reasons we implemented the stochastic gradient descent method.

Stochastic gradient works by randomly picking out m samples from the training set thus creating a batch. We estimate the gradient $\bigtriangledown C$ by computing $\triangle C_x$ for these batches. This helps us optimize the training process. The weights and biases are computed and updated by the following equations [10]:

- $w_k -> w'_k = w_k - \frac{\eta}{m}\Sigma_j \frac{\delta C_x j}{\delta w_k}$
- $b_k -> b'_k = b_k - \frac{\eta}{m}\Sigma_j \frac{\delta C_x j}{\delta b_k}$

Other related equations [10] can be found in the appendix.

To determine which cost function to use we implemented the Feedforward algorithm on two cost functions:

- Mean Square Error cost function
- Cross Entropy cost function - This is what we decided to implement.

Backpropagation helps us compute the error and the cost of the gradient function. It is an efficient and popular algorithm to do the same. To further optimize the results we incorporated L2 regularization.

C. Convolutional Neural Networks

Due to the high number of dimensions involved, the fully-connected neuron structure of regular neural networks does not scale well with images of high pixel density. Consequently, CNN consists of specialized layers where not all neurons are connected to each other.

1) TFLearn: The TFLearn Convolutional Neural Network created was trained on the augmented data using random flips and rotations. The baseline network structure starts off with a 64 * 64 pixel image as the input where it is followed by a convolutional input layer with 64 feature maps with a size of 3 * 3 and uses a rectifier linear activation function. This forms the 3D spatial arrangement of filters, where max pooling layer performs a down-sampling operation along the spatial dimensions resulting in a smaller volume. The pooling layer is followed by 2 additional convolutional layers with 128 feature maps followed by a dropout layer with p = 0.5. It ends with a fully connected layer with a softmax activation function and 40 output units making the predictions. The CNN architecture can be seen in Fig 7 in the Appendix.

The network is then trained with the learning rate initialized to 0.001 and uses cross entropy as the loss function. For optimization, it uses ADAM to compute the adaptive learning rate which has shown significant success with sparse gradients that lead it to converge faster. [11]

The dropout layer has proven to be an efficient regularization method, by reducing the variance in the model. [12] With p=0.5, we randomly delete 50% of the units during forward and backpropagation iterations, that will allow the network to learn a more robust set of features. The technique helps us to decrease co-dependence of units on each other in the network.

2) Inception: Training a deep convolutional network from scratch requires enormous amount of computational power and data. However, transfer learning reduces overhead by simply training the last layer of a pre-trained CNN where the weights from previous hidden layers are locked and only the weights for the final layer are retrained for the specific data set at hand as seen in Figure 9.

Googles Inception-v3 and VGG-16 were few of the pre-trained models that generalized well when used for transfer learning. However, the Inception v3 has been pre-trained on the ILSVRC 2014 data set and has produced the best results on the Image Net challenge. [13]. Consequently, it was the clear choice for the pre-trained model used for our data set.

The optimal set of hyper-parameters was chosen by doing cross-validation on different hyper-parameter combinations that have yielded successful results. [14] These included selecting values for the number of training steps, the initial learning rate and the train batch size. The combinations that produced significant differences in accuracy are listed in Table V in the Appendix. The optimal combination of hyper-parameter values is shown in the Testing and Validation section.

V. Testing and Validation

A. Logistic regression

For logistic regression learner, 25% of training data was reserved for validation. Upon the model trained with 75% of training data, we achieved an accuracy of 23.2% on the testing data, according to Kaggle scoring. The 25% validation data showed us the confusion matrix in figure 2.

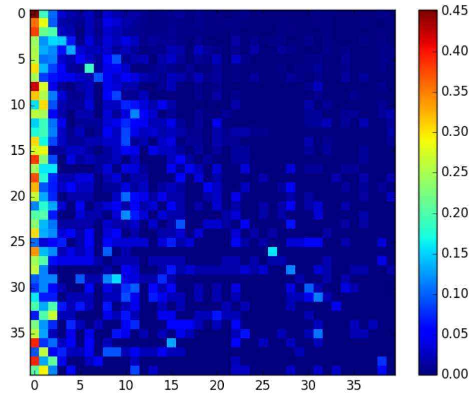


Fig. 2. Confusion matrix obtained from reserved validation data.

As soon as seeing the accuracy is only 23.2%, one would naturally assume the confusion matrix won't show the nice, distinct diagonal indicating strong classification results. This is indeed the case here. Two observations were made on this confusion matrix. Firstly, a faint diagonal from top-left to bottom-right could still be spotted, although much less convincing than desired. Secondly, the warmer colors pile up in the leftmost three columns, implying the categories most often confused for are the first three. We believe this is due to their sheer volumes: compared to the later categories, some of which having only hundreds of images, the first three categories contain approximately 8000, 4000 and 2000 images, hence the classifier is more inclined to them. Another piece of evidence for this theory is the fact that, even the diagonal appears faint in general, its much more distinctive in the top-left corner of the matrix than the rest.

B. *Feedforward Neural Network*

After testing various hyper parameters, we obtained an optimal accuracy of 29.56%. Table I below lists the hyper parameters used to obtain this result.

TABLE I

HYPER-PARAMETERS USED IN THE FNN ALGORITHM

Epochs	Learn Rate	Lambda	No. of Neurons	Batch Size
30	0.2	5.0	100	10

The naive cross validator was used to find these hyperparameters as discussed in Algorithm Implementation.

Based on empirical tests we found that changing the learning rate had a significant effect on our results. Initially, we decided to test the various learning rates on the entire data set. Figure 5 in the appendix showcases this relationship. We realized that a learning rate as low as 0.01 saturated the gradient descent after the first few epochs. Infact, as seen Figure 6. the accuracy with this hyper parameter never changed. This means that the model wasn't training. With a 0.5 learning rate the model does seem to perform better. In fact the line of best fit generated for 0.5's data shows an increasing exponential trend in accuracy 6. Seeing such results we decided to perform more tests with different hyper parameters but on a smaller subset of data to save on computation time.

As we can see from Figure 3 below, even though the data set sizes were smaller, the trend is quite similar. In fact after seeing this performance we realized that 0.2 is better than 0.5. It starts off at a lower cost and reaches the global minima at approx. the same instance as 0.5.

It should be noted that although the cost functions do change quite dramatically, the accuracy did not increase beyond 29%. We could have improved the accuracy and tested out various other parameters; however, the time and computational cost were too high to do the same.

C. *Convolutional Neural Networks*

The data was split into train, validation and test sets with percentages of 80%, 10% and 10%, respectively. The split was stratified to ensure that the ratios of different category images remains consistent across all three sets.

1) TFLearn: Initially, with the number of epochs set to 20, the model gave an accuracy of 55%. Increasing the number of epochs to 50 increased the accuracy to 65.1%. A further increase in epochs did not yield a significant increase in the accuracy of the model. Additionally, setting dropout layer to 30% instead of the recommended 50% increased the accuracy to 67.3%.

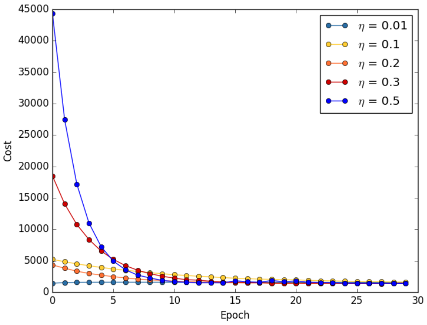


Fig. 3.

Visualizing cost of FNN for 5 different learning rates. Training = 5000; Validation = 400

TABLE II

ACCURACIES OF THE DIFFERENT CNN MODELS

Model	Accuracy
TFLearn	67.3%
Inception-v3	88%

2) Inception: With default hyper-parameter settings, the retrained model produced an accuracy of 84.06%. Table III

TABLE III

INITIAL COMBINATION OF HYPER-PARAMETERS

learn rate	train steps	train batch size
0.01	4000	100

Finding the right hyper-parameters values, using cross-validation, for retraining the last layer played a huge part in increasing the accuracy of the inception-v3 model from 84.06% to 88.00%. Using a larger initial learning rate of 0.05 helps the model to explore in the early stages of the algorithm and allow larger changes in weight values, which helps explains the gain in accuracy. Table IV shows the hyper-parameters that gave the highest accuracy and best score on Kaggle. Fig 4 and Fig 8 (Appendix) shows the cross entropy and accuracy of the model while it is being trained.

TABLE IV

COMBINATION OF HYPER-PARAMETERS WITH HIGHEST ACCURACY

learn rate	train steps	train batch size
0.05	8000	100

With the best running model, we had 343 misclassified images from a test set of 2600 images. The missclassifications were prevalent in classes 19 and 23, and is elaborated further in the Discussion section.

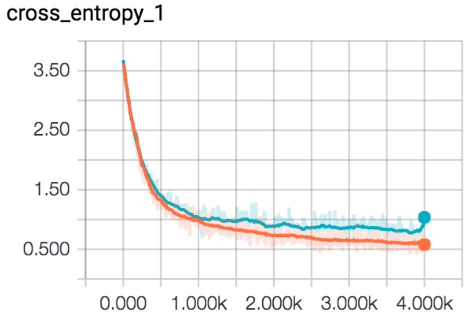


Fig. 4. Cross entropy with respect to number of training steps Orange=test Blue=validation

VI. DISCUSSION

A. Logistic regression

To recap, we never planned to count on logistic regression to give us the decisive results on Kaggle, due to its long running time and unsatisfactory performance. For this reason, we did the logistic regression part in ”fast-forward” mode, focusing on generating *some* results rather than exhaustively optimizing them.

The pro of this method is its simplicity and straightforwardness. Since we were allowed to use external libraries (scikit-learn, by our choice), the only heavy-lifting for us are data pre-processing and validation.

The cons of this method are easy to spot: first of all, high-dimensional feature space and large data set is a known bad combination for running time of logistic regression. With a provisioned virtual machine with 4 cores and 28 GB of RAM, our logistic regression learner still required approximately 11 hours to learn against 75% of the training data. Second of all, with as many as 40 categories and an imbalanced data set (see section V-A), the prediction results are less than satisfactory, even though higher than a random predictor.

We believe one way to improve the results is to balance the data set, giving each category approximately the same number of image instances. But we don’t think the running time issue can be easily resolved without some optimization tricks, such as the ones discussed in [1]. Another possible approach is the apply SIFT feature extraction to feed the learner with more ”meaningful” image data, rather than raw data as-is.

B. Feedforward Neural Network

One interesting aspect of our FNN implementation is its running time: it is just fast enough to produce results in a reasonable time frame, but too slow to allow extensive cross-validation. In general, on a virtual machine with 8 cores and 30 GB of RAM, an epoch takes 3-5 minutes depending on hyper-parameters, and accuracy improvement usually slows down after just a few epochs, so we can expect relatively ”close to peak” results within an hour, an improvement from logistic regression (see section VI-A). However, this rate isn’t enough for us to scan a broad range of hyper-parameters in cross-validation, so we had to pick our hyper-parameters in a partially arbitrary manner (see section IV-B).

The nature of neural network differs from statistical model (such as logistic regression) in that it has multiple ports for input data to come in, unlike statistical models with a single input channel. This makes neural networks more versatile in dealing with higher-dimensional feature spaces, an apparent advantage over logistic regression. But the higher level of sophistication also leads to less than optimal training time, making cross-validation difficult.

Our naive cross-validation was a negotiation with the reality that running time doesn’t permit too many cross-validation iterations. To an extent, our approach is an acceptable way around the situation, allowing us to finish the task. On the other hand, it leaves the possibility of improving the results by exhaustive searching of better parameters.

C. Convolutional Neural Networks

A weakness of the transfer learning strategy is that the trained classifier will not be able to beat the performance of a deep CNN which has been trained from scratch. However, when training a deep CNN from scratch, we can use the weights from a pre-trained model as initial values and then retrain these weights using the custom data set. This strategy will facilitate in converging faster than initializing the weight vectors to random values.

To further improve accuracy, the Inception-v3 should have been retrained on images of size 299 * 299 instead of 64 * 64. This is because the pre-trained weights were obtained using ImageNet images of size 299 * 299 and using a new size to train the final layer can lead to possible discrepancies in the new weights.

High number of misclassifications were noted for certain classes, most notably for 19 and 23 which corresponded to zucchini and cucumber datasets. A major reason for misclassification is the visual similarity of the two objects with the only significant differentiating factor being the striations on the zucchini skin. The striations are not visible in an image with lower pixel density, which can be improved by using images of higher resolution for future training datasets.

VII. STATEMENT OF CONTRIBUTIONS

1) Chhetri: Manual feedforward neural network implementation; report writing.
2) Rawlani: CNN network with TF-Learn implementation; inception-v3 network for retraining and predicting setup; report writing.
3) Yang: Logistic regression implementation; feedforward neural network debugging and cross-validation; report writing.

We hereby state that all the work presented in this report is that of the authors.

REFERENCES

[1] P. Xie, J. K. Kim, and E. P. Xing, “Large scale distributed multiclass logistic regression,” 2014.

[2] D. B. N. P. M.SEETHA, I.V.MURALIKRISHNA, "Artificial neural networks and other methods of image classification," 2008.

[3] Y. Nan, Q. Feng, and S. Zuolei, "Image classification by feature dimension reduction and graph based ranking," 2013.

[4] H. J. S. H. Seema Singh, Surabhi B R, "Neural network based methods for image classification-application and analysis," 2014.

[5] J. D. PAOLA and R. A. SCHOWENGERDT, "A review and analysis of backpropagation neural networks for classification of remotely-sensed multi-spectral imagery," *International Journal of Remote Sensing*, vol. 16, no. 16, pp. 3033–3058, 1995. [Online]. Available: http://dx.doi.org/10.1080/01431169508954607

[6] E. Kauderer-Abrams, "Quantifying translation-invariance in convolutional neural networks," 2016.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," pp. 1097–1105, 2012. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[8] X. Glorot, A. Bordes, and Y. Bengio, "Domain adaptation for large-scale sentiment classification: A deep learning approach," 2011.

[9] M. F. Kate Saenko, Brian Kulis and T. Darrell, "Adapting visual category models to new domains," 2010.

[10] "Michael a. neilsen," 2015.

[11] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[12] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *CoRR*, vol. abs/1512.00567, 2015. [Online]. Available: http://arxiv.org/abs/1512.00567

[14] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *CoRR*, vol. abs/1609.04836, 2016. [Online]. Available: http://arxiv.org/abs/1609.04836

[15] E. Kauderer-Abrams, "Quantifying translation-invariance in convolutional neural networks," 2016.

VIII. APPENDIX

Stochastic Gradient Descent Equations

Let C denote the function with m parameters.
So $m = v_1 .... v_m$

- $\triangle C \;\approx\; \bigtriangledown C + \;\triangle v$
  Where $\bigtriangledown C$ is the gradient vector
- We can find the change in v by:
  $\triangle v = -\eta \bigtriangledown C$
  where η is the learning rate

TABLE V

HYPER PARAMETERS VALUES USED FOR INCEPTION V3

learn rate	train steps	train accuracy	validation accuracy	test accuracy (%)
0.01	3700	93.2%	78.3%	85.3%
0.01	4000	96.7%	78.7%	85.4%
0.01	5500	96.5%	76.7%	85.8%
0.01	8000	95.5%	83.6%	86.3%
0.01	12000	97.5%	83.3%	86.7%
0.05	3700	92.7%	68.7%	86.9%
0.05	4000	95.7%	79.9%	87.1%
0.05	5500	96.3%	81.1%	86.7%
0.05	**8000**	**97.4%**	**83.9%**	**87.9%**
0.05	12000	98.0%	83.6%	87.4%

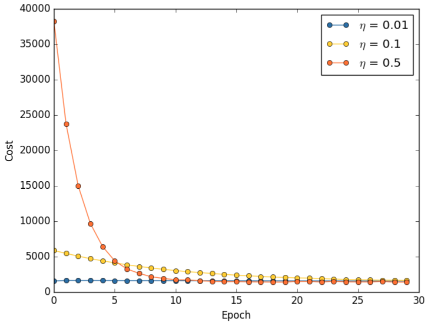


Fig. 5. Visualizing cost of FNN for 3 different learning rates. Training = 21,075; Validation = 5269

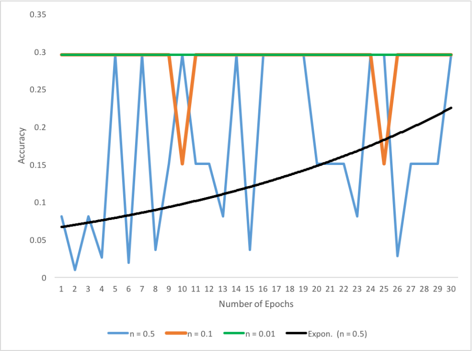


Fig. 6. Visualizing the accuracy of FNN for 3 different learning rates. Training = 5000; Validation = 400

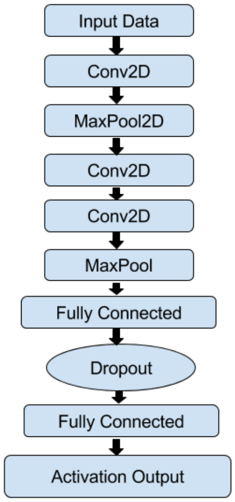


Fig. 7. The architecture of TFLearn Convolutional Neural Network

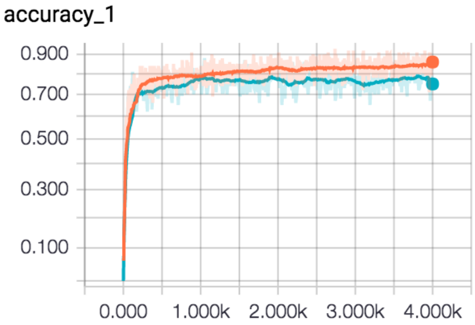


Fig. 8. Accuracy with respect to number of training steps. Orange=test Blue=validation

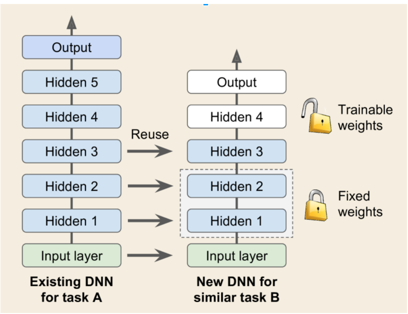


Fig. 9. Visualizing the concept of transfer learning for pre-trained models.