Assignment 2 - Group: (write your group number)

Tutors: (list all your tutors)

Group members: Firstname1 Lastname1 (unikey1), Firstname2 Lastname2 (unikey2), Firstname3 Lastname3 (unikey3)

Abstract

Abstract text goes here, justified and in italics. The abstract would normally be one paragraph long.

Introduction

This template should be used as a starting point for your report.

Previous Work

**Methods Used**

Image Classifications are complex tasks that require extraction of features and key attributes from images before they can be pumped into a model for classification. Often times, these feature extraction processes impose a massive challenge as images come in all manners of sizes (dimensions), lighting conditions, intensities of pixels, angles, scale etc. This makes it difficult for a computer to understand the key features before a classification can be carried out.

Although in humans understanding & processing images comes as a second nature, the in the field of computer vision the computer simply gets a large array of numbers to process. The figure below depicts a clear picture of this challenge.

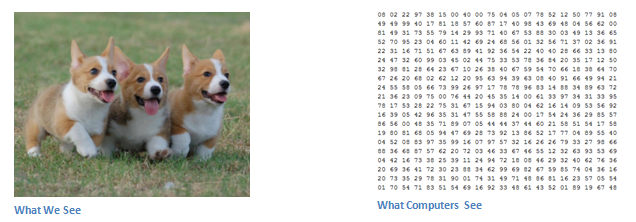


Figure 1: Computer vision vs our vision (<https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/>)

Our chosen dataset (Cifar10) is such a problem where we have thousands of 32x32 images containing 10 distinct classes of objects. We have opted to approach this classification task from a chronological technology/technique/method in the field of computer vision.

Below is a list of methods that we have attempted in chronological order

1. **Sift/Daisy Feature Extraction Followed by Classification**

Before the advent of more modern techniques such as convolutional neural networks the field of computer vision utilized several scale invariant algorithms (Sift, Surf) to carry out manual feature curation/extraction before pumping the said features into a classification algorithm. Although a lot of algorithms have been developed to extract features this is still a manual, unreliable, and inconstant methodology that hardly yielded good performance.

1. **Convolutional Neural Network (CNN)**

A more modern approach in the field of computer vision where we utilize an artificial neural network alongside a technique/methodology of convolutions to understand features and predict the image classes after passing it through a multi-layer neural network. This method yields a high degree of accuracy and able to understand more complex distributions of the data.

1. **Recurrent Neural Network (RNN – LSTM & GRU)**
2. **Residual Neural Network (ResNet)**

Another variation of a multi-layer neural network that still utilizes convolutions in the heart but with a slight twist called “short-cuts” that addresses the vanishing gradient problem in a deep neural network and thereby allows build and construction of deeper networks. ResNets are able to understand even more complex features simply via the convent of a deeper network and ability to train deeper for longer.

**Background on SIFT (Scale Invariant Feature Transformation)**

There are several techniques to extract features from images such has Histogram of Gradients (HoG), Binarizing and blurring, corner detection (Corner Harris and corner peak) but by far the most widely used method is SIFT keypoint detector.

Sift is effective due to the fact that it is able to detect/match features between images even if the scale, orientation, viewpoint, and illumination are different between images. The SIFT algorithm takes a grayscale image and generates interest points (keypoints) from the image where the local gradient orientation histograms of the image intensities are collected and statistically summarized to produce a keypoint descriptor of the local image structure (Prof. Tony Lindeberg, 2012, Scholarpedia, 7-5:10491). Typically, these statistics are gathered from a surrounding neighborhood of each keypoint.

We have opted to utilize a variation of SIFT (since it is a proprietary algorithm) called “Daisy feature extractor” which is available in the scikit image library. Once these descriptors from each image captured they can be utilized for image classification tasks as well as image matching.

**Background on Multi-Layer Neural Network**

The key advantage of a multi-layer neural net is that is able to predict and model on any distribution of data and able to create non-linear decision boundaries. The figure below depicts a simple artificial neural net and also a multilayer neural net.

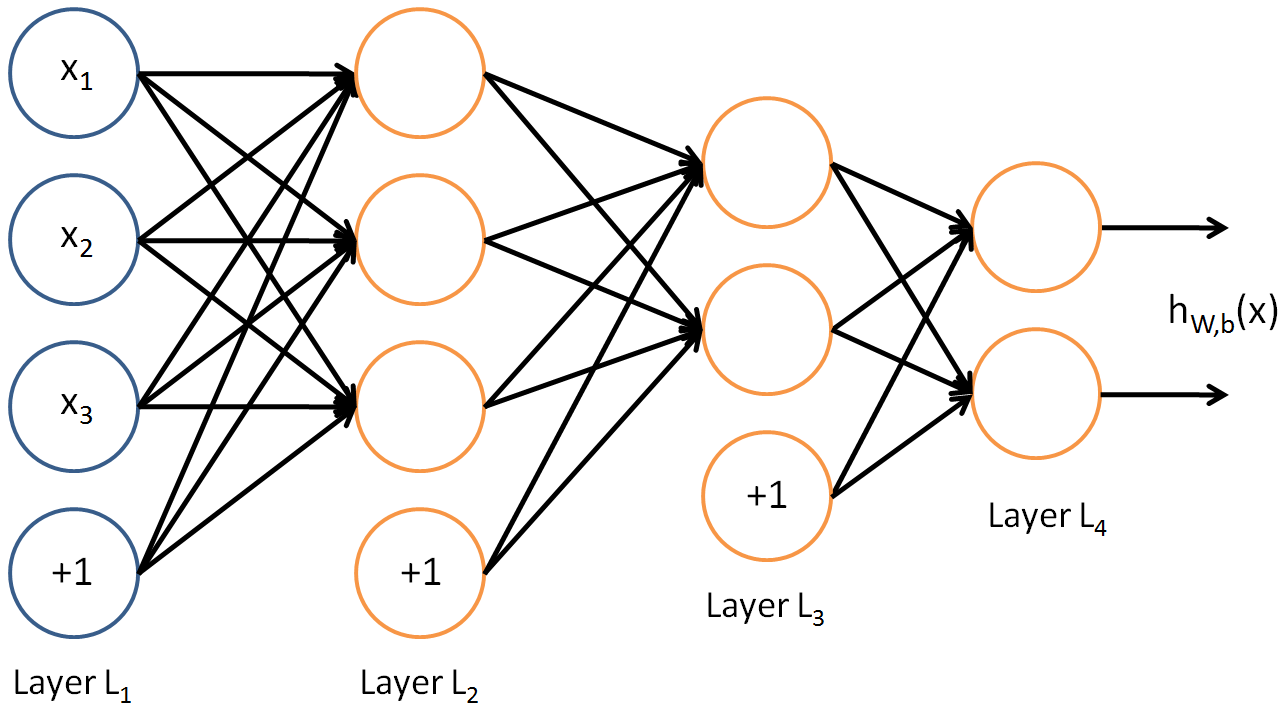
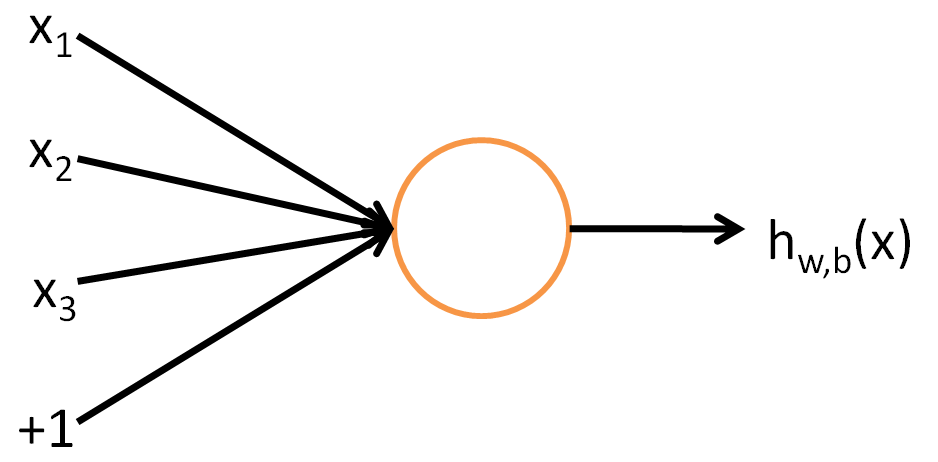
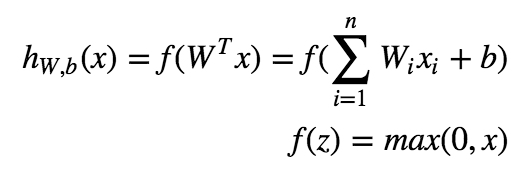


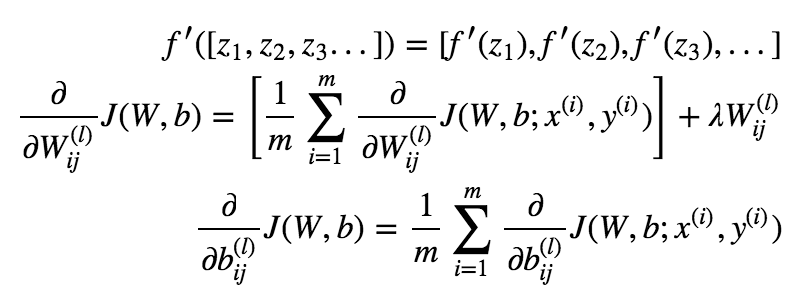
Figure 2: Stanford University depiction of a generic feedforward multi-layer neural network.

The key concepts of a neural nets are the inputs, neurons, the weights and bias terms, the output/activation function, and the optimization function (gradient descent/backpropagation). Each circle depicted in the image above is a single neuron or a computation unit which takes in the input (x1, x2, x3, and the bias unit) and outputs some new x via an activation function. The output from one neuron is then passed onto (feed forward) to the next layer of neurons which in turn carries out a similar exercise of applying an activation function. Often layers are fully connected to each other and these layers are called dense layers.

There are several activation functions that are in practice but for our exercise we have opted to utilize a Rectified Linear Function (ReLU) in all our cases. Recent research suggests that ReLU activations perform better in deep neural networks when compared to its counterparts (UFLDF Tutorial on multilayer neural network – Stanford University). The formula that runs through each of the neurons in our case can be summarized below (where n is the number of x inputs and f is the activation function) -

****

Optimizations in a neural network is carried out with gradient descent however, there are several layers within a deep neural network therefore we utilized backpropagation to optimize the weights within the network. Backpropagation simply put, allows the easy calculation of the partial derivatives of the cost functions for each layer. The chain rule allows easy calculation of the derivative of the overall cost function ((UFLDF Tutorial on multilayer neural network – Stanford University). The formula of the overall cost function for the network is outlined below.



**Convolutional Neural Network (CNN)**  
CNNs are by far the most popular method for image classification tasks. The core idea of CNN are the convolutions which learns various high and then low-level features within an image and thereby able to utilize them for image classifications and identification. In generalized terms, convolutions allow the network to learn edges, orientations, colors, blotches, blobs and allows neurons to activate when similar edges, orientations etc. are identified within another image.

**Theory**

Key concepts of a convolution are the kernel, number of filters, and pooling. Typically, multiple layers of convolutions are applied on a sample set before classifications are made. Often convolution layers are followed by a pooling layer to allow down sampling of results to ease computation load during training. In general term we have a 3x3 kernel (sliding window) that scans the images with “same” padding and a stride of 1 and using a 3x3 filter convolve with the entire image and thereby obtain feature activation values (convolved features).

In a more technical term, each step the convolution simply attempts to take a 3x3 patch of the image and apply the filter (randomly instantiated by Keras/TensorFlow) to look for a particular pattern/feature. The end result is effectively k number of features learnt.

Convolution layers are almost always followed by a pooling layer. This serves several key purposes. Firstly, the pooling ensures a down sampling is done and thereby reduces the dimension of the convolved feature. This is done by taking a statistical aggregation or summary of a contiguous area within the feature dimension. In our case we have carried out a max pool over a 2x2 region after we have learnt the convolved features. Secondly, pooling also ensures the features extracted post pooling are translation invariant, in other words if the same feature appears transformed (e.g. flipped, rotated) it will still be identified. Thirdly, due to the reduced dimension the network also has to do fewer calculations and operations when the pooled features are passed into the dense layers.

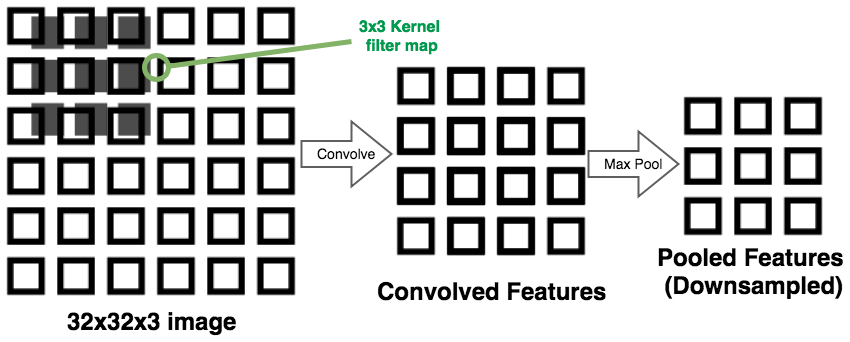


Figure 3 – Convolution and Max Pooling

**Implementation:**

The rest of a ConvNet follows the exact same principle as a multi-layer neural network. In our specific case we have opted to add in multiple dense layers followed by an output layer with 10 neurons with a SoftMax activation since cifar10 is a 10-class classification problem. The final architecture of our CNN is outlined below.

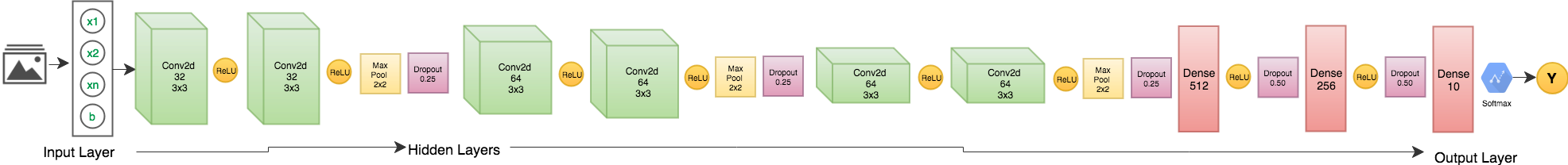


Figure 4: CNN Architecture

Figure 4 above depicts the 3 cycles of convolutions (in green) with each cycle followed by a max pooling (2x2) and dropouts at 0.25 to add in some regularization effect in the network. Dropouts simply drops off or ignores neurons below a certain probability threshold (0.25) and introduces the concept of learning less and thereby prevent overfitting of the model. We have opted to start with a 32 filter 3x3 kernel convolution and in each cycle double the filter size (32, 64, 128) and at each step carry out pooling to down sample X. This results in output shapes of dimensions 30x30x32, 13x13x64, 4x4x128 feature space. The clear effect is the down sampling of the original image but in gradual steps increasing the depth and thereby allowing the network to learn more complex low-level features.

We have also placed 3 dense layers post the convolutions in gradual decrease of 512, 256, and finally close out the output layer with a 10 unit dense layer followed by a SoftMax activation to cater to the 10 class classifications. The dense layer has additional dropouts of 0.50 to introduce yet more bias to the network and thereby allow the network to generalise to unseen data.

Finally, some technical details involving initialisation and optimisation,

|  |  |  |
| --- | --- | --- |
| Options | Method | Description |
| Weight Initialisation | Glorot\_uniform | Keras by default utilizes the Xavier uniform initializer to initialize the weights where it draws samples from a uniform distribution within a limit. The limit is set based on the number of input units and the number if output units within the tensor. |
| Optimisers | Adam | According to the authors of the optimiser the algorithm is described as “*first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments*” - Adam: A Method for Stochastic Optimization by Diederik Kingma, Jimmy Ba. In Simple terms it is a memory efficient stochastic gradient descent algorithm that is suitable for problem space where there are large data and parameter numbers and also suitable to noisy and sparse gradients. |

**Parameter Optimisations and Generalisation**

ConvNets and in general neural networks have several hyperparameters to tune. These hyper parameters include the learning rate, number of layers, number of dense layers, number of convolutions, the kernel size, the number of filters etc. Hunt for the optimum setting therefore, is a very time and resource intensive task as neural networks take a long time to train especially since we are dealing with images. Although in this particular case the image dimensions are small (32x32x3) it still requires millions of calculations at each iteration. Couple that with the gradient descent calculations this becomes a very CPU intensive task. Generally, when training neural networks and especially when it involves images, GPUs are preferred to allow for quick training and prototyping.

In our case we are bound by CPUs only and therefore we have opted for a leaner approach by training a relatively shallow network and have decided to limit the total number of epochs. As a good starting point we have taken an existing architecture (described in previous section) that has worked well for this dataset in the past instead of starting from ground zero. The summary of the hyper parameters can be seen in the table below.

|  |  |  |
| --- | --- | --- |
| Hyperparameter | Setting | Details |
| Convolutions | 6 convolutions | 32, 64, 128 features with 3x3 kernel, same padding and stride of 1 |
| Dropouts | 5 dropouts | 0.25 post conv and .50 with dense layers |
| Pooling | 3 Max pools | Max pool with 2x2 pool size |
| Dense layers | 3 dense layers | Gradual decrease of the dense layers (512, 256, 10) |
| Activations | ReLU, SoftMax | All ReLU activations except the final output layer using a SoftMax |

We have also decided to test the model’s generalisation capabilities by splitting the training dataset into 40K images for training and 10K for evaluation of the model as 10-fold is not a typical choice when it comes to evaluating neural networks. At each epoch the network updates the weights (based on backpropagation) and we allow the model to predict on the evaluation dataset as well as the training set.

In addition to training and validation dataset accuracy we have also designed to capture the F1-score to allow for a more detailed analysis of the models’ generalisation capability across all 10 classes. Furthermore, we have also captured the loss statistics at each epoch.

The diagram below (figure 5) indicates after 50 epochs the F1 and accuracy on training data at 85% (approx.) with the evaluation dataset stabilising at 80% (approx.). A similar divergence can be seen when looking at the evaluation and train dataset losses. The training dataset loss is clearly seen to be dropping but the evaluation set loss is stagnant at 0.6. All this indicates that the model is not properly generalising to unseen data and we are slowly overfitting the model.

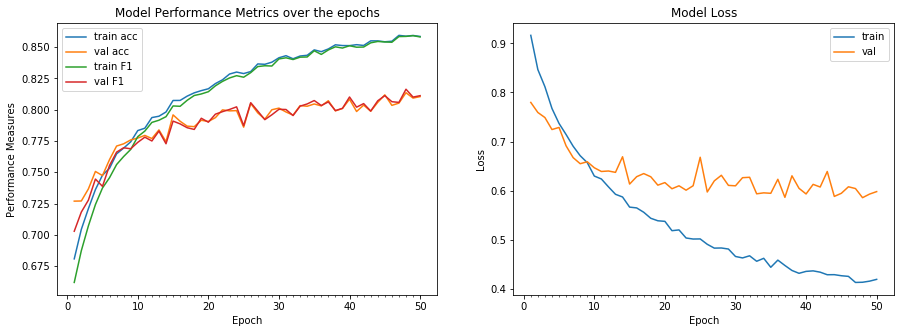


Figure 5: CNN model performance on test split and loss curve

Further, tweaks can be made to the network to now counter this divergence effect by introducing more dropouts and perhaps by playing with varying degree and number of convolutions. However, since training ConvNets are so expensive when it comes to CPU, time and resources that we have opted to close out the exploration and parameter optimisation. Provided some GPU access further refinements to the architecture can be made and also longer epoch numbers can be introduced. The CNN in its current state took approx. 2.8 hours to train for a total of 50 epochs.

**Residual Neural Network (ResNet)**

ResNets brought about a revolution in the field of computer vision when it first came out in 2015. Unlike the deep convolutional neural networks, Residual Neural Networks allow training of much deeper networks and thereby is embodied with great representational capability not only to images but in a wide range of activities such as object detection and face recognition (an overview of ResNets – Towards Machine Learning).

**Theory**

One of the biggest problems with deep neural networks is the vanishing gradient problem. Vanishing gradient is a serious drawback of a deep neural network where the repeated multiplication during backpropagation makes the gradient extremely small (almost towards zero). The end result is that the model learns very small amounts with large number of epochs and with an eventual increase in overall cost within the network (cost all of a sudden start to climb as opposed to descending). ResNets attempt to solve this vanishing gradient problem through a concept known as “Shortcut” or “Skip” connections.

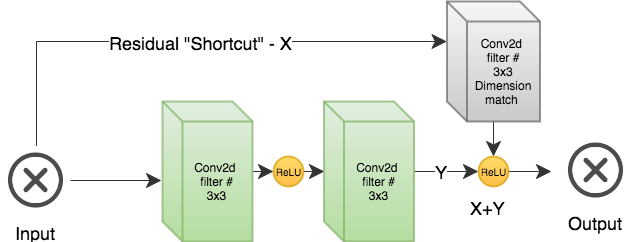


Figure 6: Shortcut connections in ResNets

The figure above depicts the core concept of a ResBlock that we have implemented which involves passing the X directly to a ReLU/Activation function and thereby skipping the two convolution layers. As we downsample X it is also important to make sure the dimensions match up during the shortcut connection so that the output Y and the residual X can be added up (X + Y) before applying an activation function. For this reason, ResNets typically involve using an intermediate convolution layer (depicted in gray) to match the dimensionality of the output from the prior convolution layers. By passing in the residuals via the skip connections the network solves the problem of vanishing gradients. Other than the vanishing gradient solution ResNets also improves and encourages feature reuse and thereby achieves high parameter efficiency. Typically, ResNets involve stacking these residual blocks together to produce output.

**Implementation**

Outside of this new concept of res blocks and shortcut connections the rest of the architecture of a ResNet is almost the same as a CNN. We still utilize convolutions followed by max pool and dropouts to introduce bias (regularization) to the network. The authors of ResNet carried out their implementation by stacking the res blocks together. In our case we have opted to utilize 3 stacks in total with each stack containing two residual blocks.

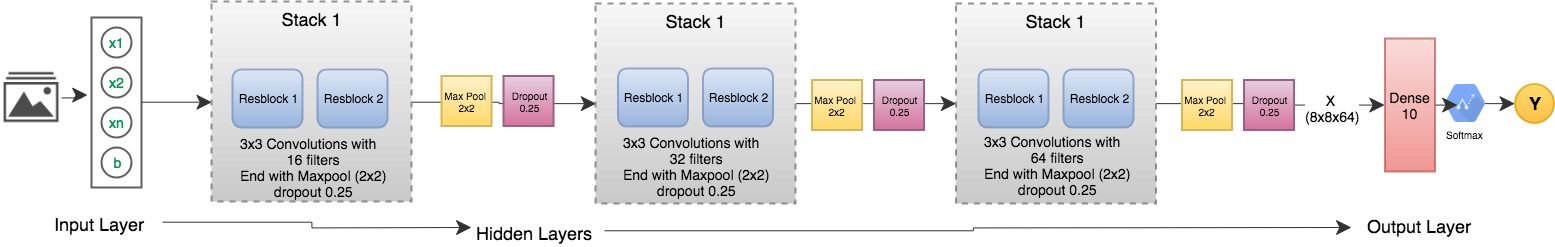
****

Figure 7: ResNet Architecture

The diagram above illustrates our chosen architecture which contains 3 stacks of 2x2 convolutions. At the end of each stack we carry out max pooling to downsample X as well as introduce dropouts of 0.25 to ensure the model gets to generalize. It is also worth noting here that to ensure the shortcuts can be created, the convolutions within a resblock always uses the same kernel, filter numbers, with same padding. This ensures the dimensions are always the same. Another callout is the use of an intermediate convolution applied to X before applying the final ReLU within a res block. This is done to ensure that the output dimension (Y) matches the (X) so that we can carry out the addition (X+Y).

Similar to our ConvNet we have opted to start with 16 filters of 3x3 kernel convolution and double the filter size (32, 64) and at stack within the network. This results in output shapes of dimensions 32x32x16, 16x16x32, 8x8x64 feature space. The clear effect of downsampling at each stack and the subsequent depth increase allows the network to learn more complex low-level features similar to the ConvNet. The final output layer (post 3 stacks) is a single dense layer with 10 neurons followed by a SoftMax activation.

The beauty of the ResNet is the fact that we can build deep networks without the fear of the vanishing gradient problem. Ideally, the network is meant to include additional resblocks and perhaps we can trial with multiple dense layer. However, even with this shallow setup, our network still contains a total of 14 convolutions. We have opted to design a shallow network to allow the network to train on a CPU with a minimal number of epochs.

Similar to the CNN our implementation of ResNet utilizes Adam optimizer alongside the Xavier Uniform Initializer for the weights initialization.

**Parameter Optimisations and Generalisation**

ResNets just like CNNs have several parameters to train. The same exact challenges for optimisation on a CPU remains. The unique situation with ResNet however, comes in the form of number of stacks and the number of resblocks that yields better results. The original authors of ResNet utilised Batch Normalisation over dropouts but in our case we have opted to keep the design similar to the CNN for a more direct comparison and carried out our dropouts after each stack. The following table outlines the various hyper parameters that are available with ResNets and our chosen setup.

|  |  |  |
| --- | --- | --- |
| Hyperparameter | Setting | Details |
| Stacks | 3 stacks | Stacks containing the residual blocks. |
| Res blocks | 6 resblocks | 2 resblocks per stack |
| Shortcuts | - | Total of two shortcuts in each stack via the two resblocks |
| Convolutions | 14 convolutions | 3x3 kernel, padding same, stride of 1. We also double the filter numbers in each stack. (16, 32, 64 filter convolutions) |
| Dropouts | 3 dropouts | 0.25 end of each stack |
| Pooling | 3 Max pools | Max pool with 2x2 pool size before the dropouts are carried out. |
| Dense layers | 1 dense layer | 10 neuron dense layer followed by a SoftMax Activation |
| Activations | ReLU, SoftMax | All ReLU activations except the final output layer using a SoftMax |

ResNets at the end of the day is another form of multi-layer neural network and shares many if its drawbacks. Similar to our CNN training we have opted to train this model based on an 80/20 split of the training dataset. The 20% dataset (10K images) is kept in order to evaluate the models’ generalisation capability. Furthermore, we have reused the same metrics from CNN to empirically assess the models’ performance on the test split (F1, Precision, Recall etc.). In addition, the training losses (categorical cross entropy) is also captured at each iteration.

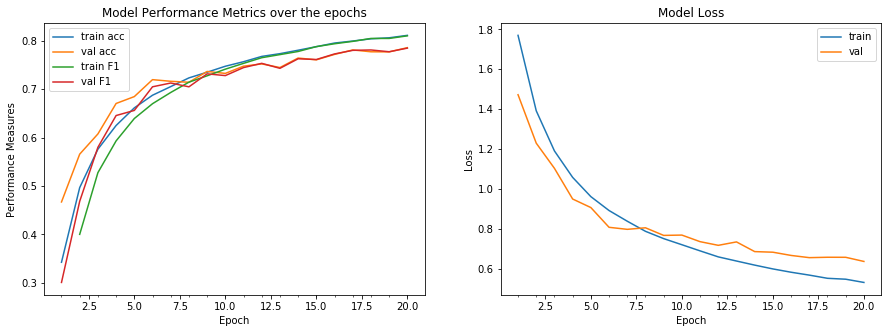


Figure 8: ResNet performance over test split alongside loss curve

The figure above depicts the models’ performance over 20 epochs. Please note, that we have opted to leave the epoch sizes as low as 20 and we have also altered the architecture of the ResNet to allow the training to be conducted within a reasonable time frame. Even with shallow network, 20 epochs took approx. 1.6 hours to complete. Both the loss curve as well as the F1-score across all epochs on the training and test/evaluation set seems to be tightly grouped together indicating a good fit and the adequate generalization capability of our architecture. After 20 epochs the F1 on training data is at 0.80 (approx.) with the F1 on validation set closely following at 0.78 (approx.). Unlike our CNN, there is minimalistic divergence on the errors and F1, accuracy metrics.

Typically, several iterations and prototyping are required to understand the different effects of the various hyper parameters to fine tune the model. To save time and resources we have opted to adopt an existing architecture for this from the Keras website with some minor tweaks relating to allow the network to train quicker on a CPU (reduced number of stacks, resblocks etc.). As with any image classification using deep learning, GPUs are desired as opposed to CPUs.

**Sift/Daisy Features & SVM**

**Experiments**

**Conclusions**

**References**

1. Multi Layer Neural Network – Stanford University - <http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/>
2. Pooling Overview – Stanford University - <http://ufldl.stanford.edu/tutorial/supervised/Pooling/>
3. Feature extraction using convolutions – Stanford University - <http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution/>
4. Gardner RM, Golubjatnikov OK, Laub RM, Jacobson JT, Evans RS. Computer-critiqued blood ordering using the HELP system. Comput Biomed Res 1990;23:514-28.