Assignment 2 – Group 42

Tutors

- Kelvin Hsu
- Anthony Tompkins

Group Members

- Nagib Shah – nsha9343
- Girishkumar Dhotarkar – gdho0835
- Raghuveer Sripada – rsri0030

Contribution:

Student Name	Student Unique Key	Contribution
Nagib Shah	**Nsha9343**	**33.33%**
Girishkumar Dhotarkar	**Gdho0835**	**33.33%**
Raghuveer Sripada	**Rsri0030**	**33.33%**

Abstract

Image recognition is the latest buzz word in the field of computer vision. Although there have been several key and industry disruptive advancements in this space such as Convolutional Neural Nets and Residual Nets, image recognition and classification still remains a constant challenge. In this report we have explored several methods and techniques for image recognition/classification alongside some empirical comparison between them. Image recognition and object detection has a wide range of use case in almost all industries. In our report we have approached this problem from a chronological perspective when it comes to advancements and technologies available at our disposal, ranging from Bags of Visual Words, SIFT, SVMs (classical approach) to more modern Convolutional Neural Networks, Residual Networks, Recurrent Neural Networks. This report aims to provide a deep insight into the wide range of activities, tasks, methods, procedures that are involved with image classifications and closes out with a detailed comparison of the various methods from empirical, practical, and theoretical perspective.

Introductions

The CIFAR-10 dataset in question consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. Problem is to predict test images based on training images with maximum accuracy, precision. Overall the image recognition can be applied in all manners of disciplines and cross industries from biostatistics, facial recognition, image tagging and countless more use cases. As such image recognition, object detection, and classification is an important branch of computer vision and neural networks in general.

In this report, various techniques have been utilised to analyse the cifar10 image set and classify the images accordingly. Image recognition involves a lot of information operation, requiring high processing speed, GPU and CPU, alongside recognition accuracy as well as precision. We have applied various validation metrics throughout the experiments to prove the feasibility and effectiveness of the optimization method. We have opted to explore technology platforms such as Scikit Learn, Keras, Tensorflow, as well as Scikit Image libraries to carry out all experiments outlined in this report. We have attempted to approach this problem from the simplest to the most modern and widely practiced and accepted methodologies available. Following are various methods we have used for CIFAR-10 dataset image recognition.

- Sift/Daisy Features & SVM (Classical approach)
- Convolution Neural Network (CNN)
- Recurrent Neural Networks – (RNN - LSTM & GRU)
- Residual Neural Network – (ResNet)

Previous Work

As per previous work done on image classifications, it has been proven that neural networks methods are best for classification of images. Following are few top 3 neural networks and their CIFAR-10 classification success rates.

Fractional Max-Pooling (Result 96.53%)
Convolutional networks almost always incorporate some form of spatial pooling, and very often it is alpha times alpha max-pooling with alpha=2. Max-pooling act on the hidden layers of the network, reducing their size by an integer multiplicative factor alpha. The amazing by-product of discarding 75% of your data is that you build into the network a degree of invariance with respect to translations and elastic distortions. However, if you simply alternate convolutional layers with max-pooling layers, performance is limited due to the rapid reduction in spatial size, and the disjoint nature of the pooling regions. Form of fractional max-pooling reduces overfitting on a variety of datasets: for instance, improvement on the state-of-the art for CIFAR-100 without even using dropout [18].

Spatially-sparse convolutional neural networks (Result 93.72%)
Convolutional neural networks (CNNs) perform well on problems such as handwriting recognition and image classification. However, the performance of the networks is often limited by budget and time constraints, particularly when trying to train deep networks. Motivated by the problem of online handwriting recognition, we developed a CNN for processing spatially-sparse inputs; a character drawn with a one-pixel wide pen on a high resolution grid looks like a sparse matrix. Taking advantage of the sparsity allowed us more efficiently to train and test large, deep CNNs. On the CASIA-OLHWDB1.1 dataset containing 3755 character classes we get a test error of 3.82%. Although pictures are not sparse, they can be thought of as sparse by adding padding. Applying a deep convolutional network using sparsity has resulted in a substantial reduction in test error on the CIFAR small picture datasets: 6.28% on CIFAR-10 and 24.30% for CIFAR-100 [18]

Scalable Bayesian Optimization Using Deep Neural Networks (Result 93.17%)
Bayesian optimization is an effective methodology for the global optimization of functions with expensive evaluations. It relies on querying a distribution over functions defined by a relatively cheap surrogate model. An accurate model for this distribution over functions is critical to the effectiveness of the approach and is typically fit using Gaussian processes (GPs). However, since GPs scale cubically with the number of observations, it has been challenging to handle objectives whose optimization requires many evaluations, and as such, massively parallelizing the optimization. Exploration the use of neural networks as an alternative to GPs to model distributions over functions. It shows that performing adaptive basis function regression with a neural network as the parametric form performs competitively with state-of-the-art GP-based approaches, but scales linearly with the number of data rather than cubically. This allows achieving a previously intractable degree of parallelism, which we apply to large scale Hyperparameter optimization, rapidly finding competitive models on benchmark object recognition tasks using convolutional networks, and image caption generation using neural language models [18]

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, 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.

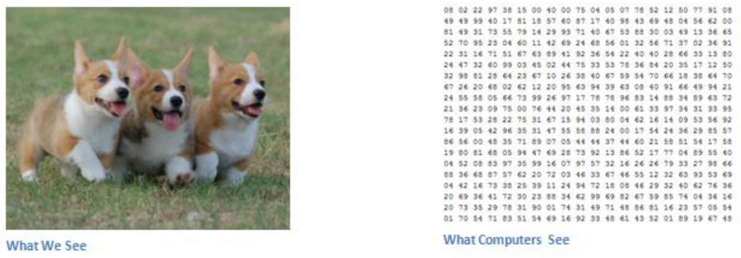


Figure 1: Computer vision vs our vision [6]

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 neighbourhood 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 are 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 it 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. The key concepts of neural nets are the inputs, neurons, the weights, bias terms, the output/activation function, and the optimization function (gradient descent/backpropagation).

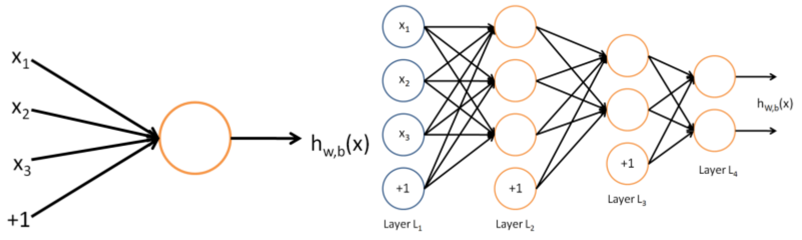


Figure 2: Stanford University depiction of a generic feed forward multi-layer neural network.

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 the **Rectified Linear Function (ReLU)** in all our cases. Recent research suggests that ReLU activation 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) -

$$h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{n} W_i x_i + b)$$

$$f(z) = max(0, x)$$

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 functions for the network is outlined below.

$$f'([z_1, z_2, z_3 \ldots]) = [f'(z_1), f'(z_2), f'(z_3), \ldots]$$

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x^{(i)},y^{(i)})\right] + \lambda W_{ij}^{(l)}$$

$$\frac{\partial}{\partial b_{ij}^{(l)}} J(W,b) = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial b_{ij}^{(l)}} J(W,b;x^{(i)},y^{(i)})$$

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.

2. **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.

3. **Recurrent Neural Network (RNN – LSTM & GRU)**

   Humans don’t start their thinking from scratch every second. As you read this essay, you understand each word based on your understanding of previous words. You don’t throw everything away and start thinking from scratch again. Your thoughts have persistence. Traditional neural networks can’t do this, and it seems like a major shortcoming. For example, imagine you want to classify what kind of event is happening at every point in a movie. It’s unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones. Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.

4. **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.

Method 1: Sift/Daisy Features & SVM Details

For this experiment we have opted to take a classical approach for image classification. Before the advent of CNNs and ResNets bulk majority of the image classifications involved manual curation of features before passing the features into a classifier. For our experiment we have opted to utilise Daisy (part of the Scikit image library) to extract the feature descriptors of key points identified from the images and then utilising an old school SVM to classify the images. We have taken 2 slight variations to this approach.

- Daisy Feature + BoVW (Bag of Visual Words) + SVM (Radial & Linear)
- Daisy Feature + SVM Linear

Theory

Daisy similar to SIFT is able to detect and generate feature descriptors for image matching from a grayscale image. According to the authors of the algorithm the algorithm utilises gaussian weighting and circularly symmetrical kernel alongside Histogram of Gradients to generate local image descriptors [4]. The algorithm requires specification of circle radius, number of circles/rings, and the step. Given that we have a small 32x32 image space we have opted to proceed with a step of 4, with a circle radius of 5 to maximise the number of feature descriptors learnt from the images. Daisy is suitable for BoVW feature generation and also allows for fast dense extraction of features (Scikit image documentation).

BoVW [5] is a technique for generation of a vocabulary or a codebook from key points of interest from an image. In typical sense we can use a bag of words to carry out a text classification. However, since we are dealing with an image we have to first develop a codebook so that we can generate a bag of words. In our approach we have attempted to generate the codebook utilising a k-means clustering algorithm to develop the codebook. Later the codebook can be utilised to generate the bag of visual words for any given image. A rudimentary way to explain this process is that k-means clustering allows us to develop a dictionary of words such as wheels in the case of cars and then look for wheels within a newly extracted daisy feature. An intuitive way to think of this process is that we end up learning the key features that identify an object from other objects (e.g. cars have wheels but not people).

Once we have the vocabulary generated the next step is to develop a bag of visual words for a given image. In simple terms, this is simply a process of looking at how many times a specific visual word occurred in a given image. This is done by generating a histogram of occurrences of words within an image. Generating the BoVW also acts as a means of dimensionality reduction through the process of Daisy, K-Means, & Histogram generation as it effectively shrinks the entire image set to a set of visual words only.

Once we have to features identified and generated (BoVW or Daisy) we can utilise any number of classifiers to define the decision boundary. However out of all the pool of classifiers at our disposal we have opted to utilise SVM for several key reasons. The primary reason behind our use of SVM is that it is able to define non-linear decision boundaries. The second reason is that SVMs have regularisation parameters which allow us to penalise the algorithm for overfitting on the training data. Thirdly, we have the choice of kernels at our disposal which allows us to engineer various different types of kernels that best suits our dataset. For our exercise we have opted to utilise both the radial and linear kernel.

Implementation

The implementation of our model can be described in the following steps –

- Convert the images into grayscale this immediately drops the colour channels and thereby we carry out an unintended dimensionality reduction during this step
- Apply a Daisy algorithm on each and every image within the image set and extract some Daisy feature descriptors
- Once all the features are extracted we can now apply a K-means clustering on the features to learn/develop our vocabulary/codebook. To identify the best K we have opted to apply MiniBatchKmeans (for performance gains) for a range of clusters from 20-500 clusters
- Application of elbow evaluation to lock in the optimum number of clusters that best explains the feature space. The diagram below indicates the result of the grid search for best K over

the feature space. Based on the information we have opted to lock in 450 as the optimum K as it yields the lowest k-means score

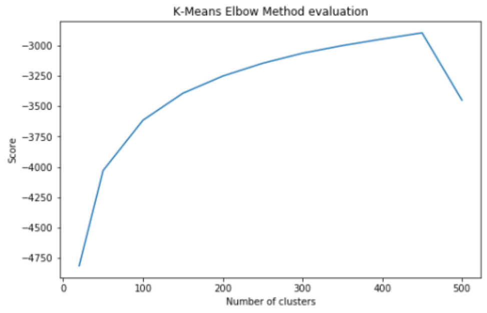


Figure 3: Hunt for Best K

- Generate the BoVW from the codebook learnt (using the best K). We effectively generate a histogram of occurrences from the Daisy features for each image. The result of this operation is a reduction of the original 1024 (32x32) features to 450 features. A simple way to think of this is we end up representing each image with 450 key features (BoVW) and their occurrences
- Train SVM Radial & SVM Linear using the BoVW learnt from the images
- Alternatively, we have also trained an SVM Linear directly on the features extracted from the Daisy extracted features. However, the features in its raw state are of high dimensionality and in many aspect, this is the opposite of dimensionality reduction

Parameter Optimisations and Generalisation

SVMs have one key hyper parameter that can help us train the model to its full potential. It is the cost parameter. If a clean boundary cannot be defined the cost parameter allows us to specify the acceptable C parameter (cost) that is treated by the model as the acceptable misclassification limit. In our problem space we have opted to do a grid search to find the best C parameter by training and testing the model over either a 10-fold cross validation or a simple train and test split (SVM Radial). It is worth noting here that the SVM method is generally a binary classifier and for that reason we have utilised OVR (One vs Rest) approach to our 10-class classification problem.

BoVW & SVM Radial –

SVM Radials do not scale well to any sample size above 10000. However, in our case we have a sample space of 50K and thereby a radial kernel is more computationally intensive and time consuming. To work within our CPU boundary, we have opted to utilise a simple train and test split to carry out our grid search.

C values	0.1, 1, 1.5, 2, 5
Method	Train and test split (80/20)
Time taken	1000+ seconds per test cycle
Best C parameter	5
Best F1-micro (avg)	0.51 (approx.)

BoVW & SVM Linear –

Scikit utilises liblinear rather than libsvm for LinearSVC classifier. Simply put this utilises a linear kernel and is able to scale well to large dataset as well as high dimensionality however, due to its linear kernel it is able to understand less if the data is not linearly separable. Due to its high speed and performance we have opted to utilise a 10-fold validation process to find the best C and train this model.

C values	1, 2, 3, 5, 10, 100
Method	10-fold cross validation
Time taken	40+ seconds per test cycle
Best C parameter	10
Best F1-micro (avg)	0.41 (approx.)

Daisy Features & SVM Linear –
For this instance, we have designed an experiment by training the SVC Linear without any pre-processing or BoVW. We simply extract of feature from the full training set and flatten the feature dimension to change the shape of the dataset from 50000x32x32 to 50000x2176. It is worth noting that although this almost doubles the feature space, given the Linear SVCs performance on large high dimensional dataset it is still possible to train this model over the full 10-fold cross validation.

C values	0.1,1,3,5,10
Method	10-fold cross validation
Time taken	180+ seconds per test cycle
Best C parameter	10
Best F1-micro (avg)	0.58 (approx.)

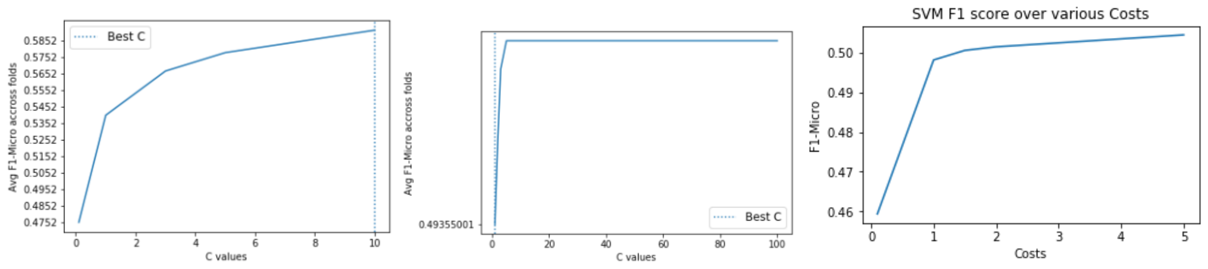


Figure 4: F1 Micro score over the c values
(From the left – SVC Linear with Daisy, SVC Radial with BoVW, SVC Linear with BoVW)

Method 2: Convolutional Neural Network (CNN) Details

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, colours, 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.

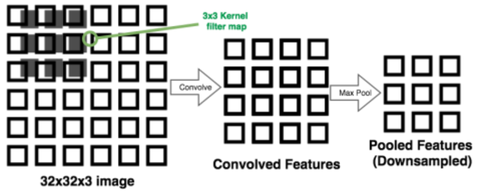


Figure 5 – Convolution and Max Pooling

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.

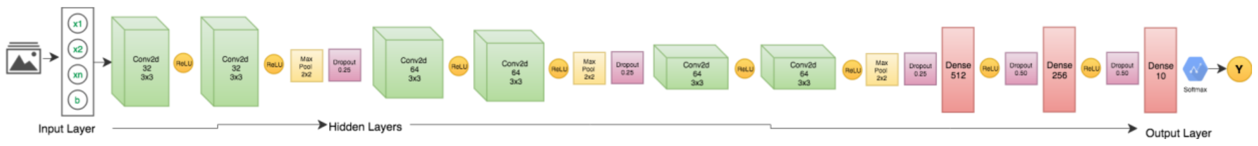


Figure 6: CNN Architecture

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.

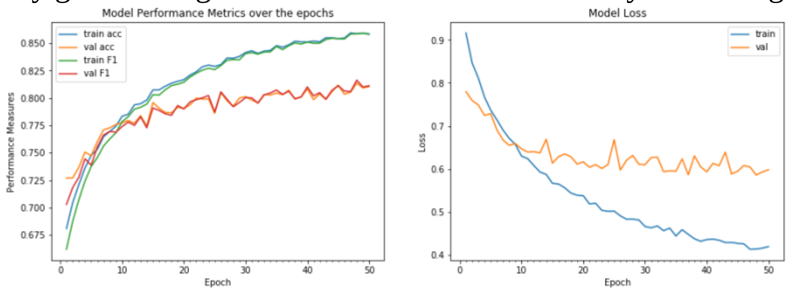


Figure 7: CNN model performance on test split and loss curve

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.

<u>Method 3: Recurrent Neural Network (RNN - GRU / LSTM) Details</u>

Theory

At times, we only need to look at most recent information to predict a current task. E.g. consider a model trying to predict the caption based on picture. In such cases we are predicting the caption based on current image/and to an extent based on the most recent image and we do not need any further context. In such cases, where the gap between the relevant information and the place that

it's needed is small, RNNs can be modelled to learn use the past information. But there are also cases where we need more contexts. Consider trying to predict the last word in the text "The cats ate a lot of food ……. and ***are*** feeling sleepy.". Recent information suggests that the next word after ***and*** is probably a verb, but if we want to narrow down which verb it is (singular/plural), we need the context of the noun which in this case is ***cats/cat***, from further back. It's entirely possible for the gap between the relevant information and the point where it is needed to become very large.
Unfortunately, as that gap grows, RNNs become unable to learn to connect the information. This can be overcome by modifying RNNs to branch out a new model call **Long Short Term Memory (LSTM)** and **Gated Recurrent Unit (GRU). Difference between them is as below -**

- The GRU unit controls the flow of information like the LSTM unit, but without having to use a memory unit. It exposes the full hidden content without any control
- The performance of GRU is on par with LSTM, but computationally more efficient
- LSTMs remember longer sequences than GRUs and outperform them in tasks requiring modelling long-distance relations

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

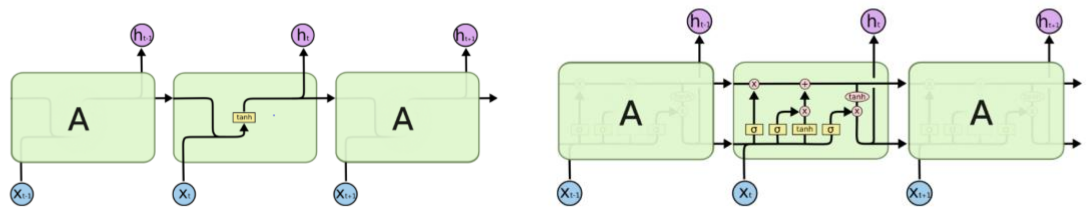


Figure 8: LSTM Neural Network [8]

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way. GRU is a more customized version of LSTM.

Implementation:

Design for Assignment-2 CIFAR-10 dataset

- ✓ time_steps=32 # timesteps to unroll
- ✓ n_inputs=32 # rows of 32 pixels (an CIFAR10 image is 32x32)
- ✓ n_units=512 # hidden LSTM units
- ✓ num_classes=10 # classes/labels (0-9)
- ✓ batch_size=500 # Size of each batch
- ✓ n_epochs=5
- ✓ 2 stacked LSTM/GRU units in sequence

We have arrived at this hyper-parameter combination after a trial and error method using binary search on batch size, epoch and number of hidden units. The highest accuracy that we have got with these parameters for both LSTM and GRU is ~52%, but this model has taken ~4 hours to train. Hence we have run the model with parameters for which the model runs faster.

Observations

The model evaluation parameters for LSTM and GRU is much below that of CNN or RESNet, that have a more static approach to the problem. Here we have passed the input dataset mimicking a sequential order but in actuality it's not. So the model is not seeing the whole image and is only seeing a part of image at a mimicked timestamp. Hence this is resulting in a poor accuracy.

Method 4: Residual Neural Network (ResNet) Details

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.

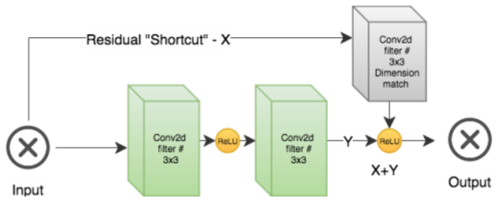


Figure 9: Shortcut connections in ResNets

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.

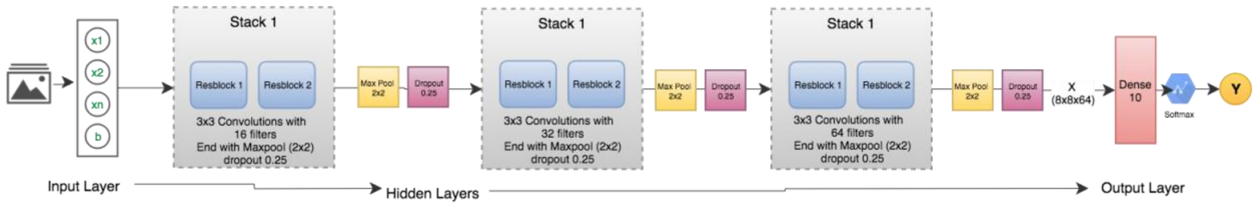


Figure 10: ResNet Architecture

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	3x3 kernel, padding same, stride of 1. We also double the

	convolutions	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.

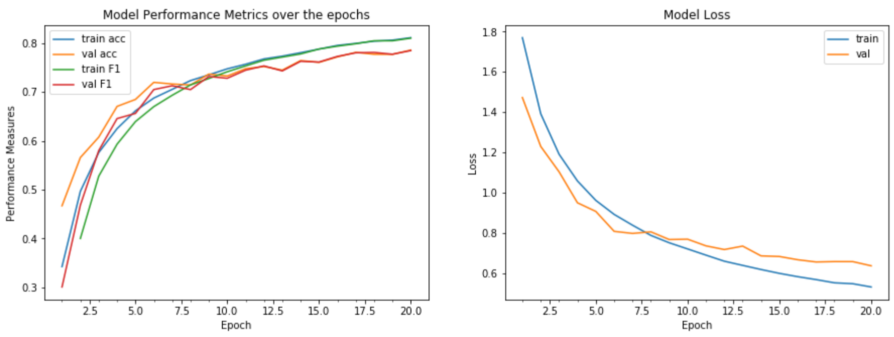


Figure 11: ResNet performance over test split alongside loss curve

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.

Experiments & Discussions

Post processing and optimisation of each models we have predicted on the hold out dataset (unseen data) and compared performance statistics and metrics. The hold out dataset was never utilised during the training and evaluation of any of the models. This way we are able to empirically prove each models' generalisation capability. The experiment yielded the following interesting result –

Accuracy Statistics Comparison:

We have plotted performance metrics for all models to run an empirical comparison. Figure below indicates various models' key performance metrics (avg.). Each parameter is indicated by circle and size of circle indicates a high performance for each model (bigger circle indicates better performance). From comparison below clearly, **CNN is best performing model**, but ResNet also come close to best performing. CNN took more time to train as compare to all other model although we have run a total of 50 epochs. However, ResNet has potential to become best performing model if it is trained deeper for much longer. RNN is best for time sequence image classification but not for multi-class image classifier.

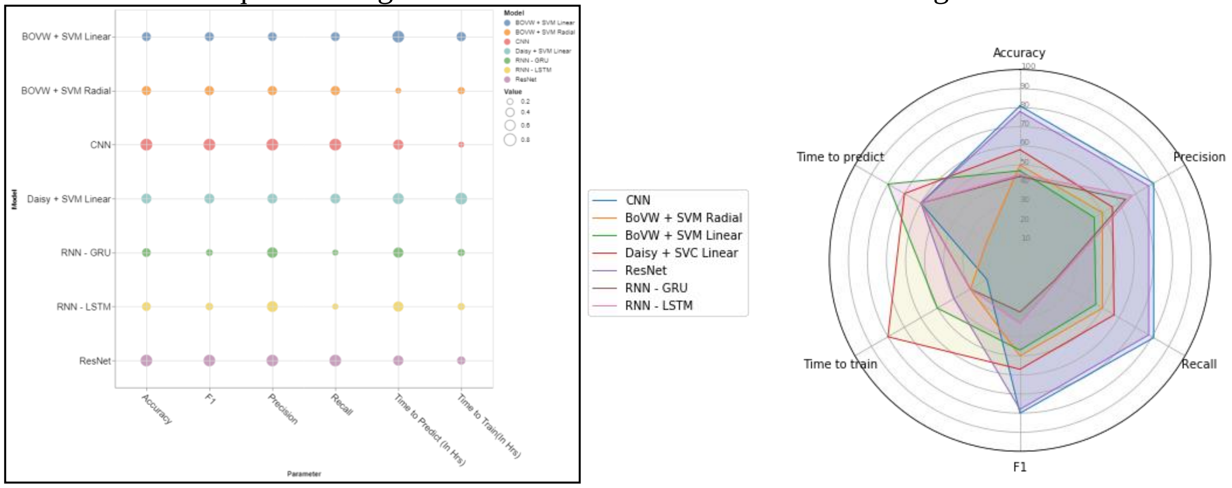


Figure 12: Performance Stats of Model (left bubble chart, right spider chart)

The figure above provides an easy to digest summarization of our findings after exploring a wide range and breadth of image classification models. The SVC Linear is the clear winner when it comes to training time as the LinearSVC library is able to scale extremely well to large sample sizes and large feature space. However, it loses out in all other critical categories (Accuracy, Precision, Recall, F1) when compared to the more advanced counter parts (the artificial neural nets).

The two standout winners are CNN and ResNets as they are able to outperform every other model in all key metrics (Precision, Recall, F1, Accuracy) with a standout figure in the the upper range of .80. However, training these models impose a large challenge when it comes to hardware resource and training time. Another key callout with artificial neural nets is the explainability factor. Although CNN and ResNets are a form of supervised learning, the classification boundaries cannot be explained due to the complex nature of their setup (fully connected layers and multiple layers of convolutions). We know the model generates predictions based on low level features, but these features are unknown and cannot be explained in human readable format.

Receiver Operating Characteristic (ROC) – Area under Curve – AUC:

Out of all models following models are top performing models in terms of accuracy statistics, on reviewing AUC and Confusion matrix of both models it has been identified that CNN model is getting confused to identify images of "Cat" and "Dog". ResNet Model is getting confused for "Bird" & "Frog" with other images. Training both models for longer time (More echoes) may help to get better result.

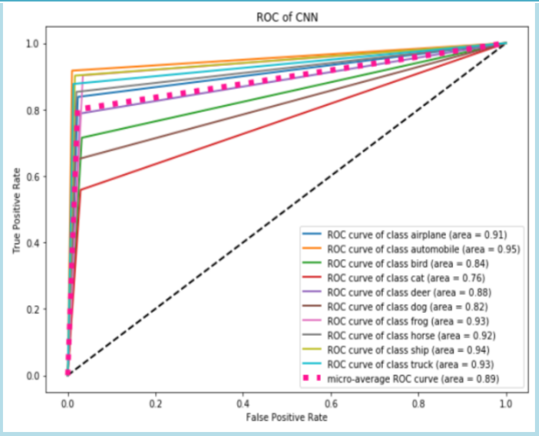
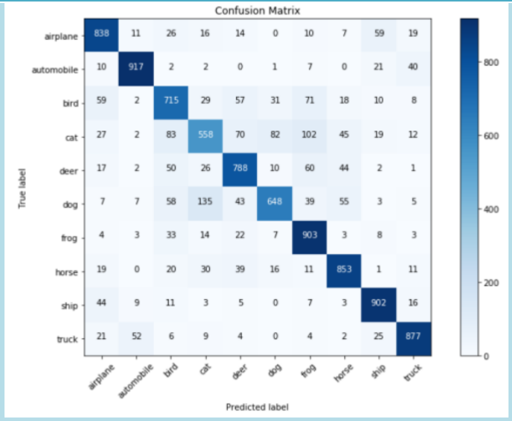
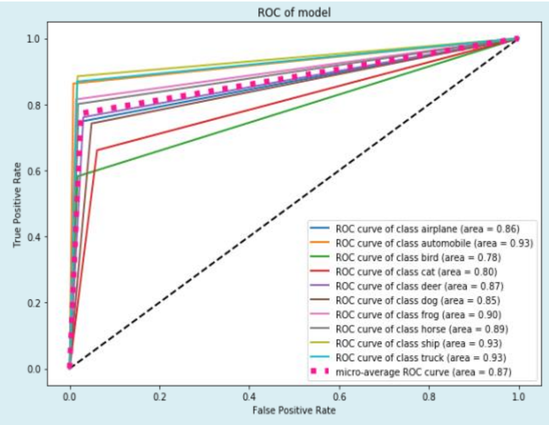
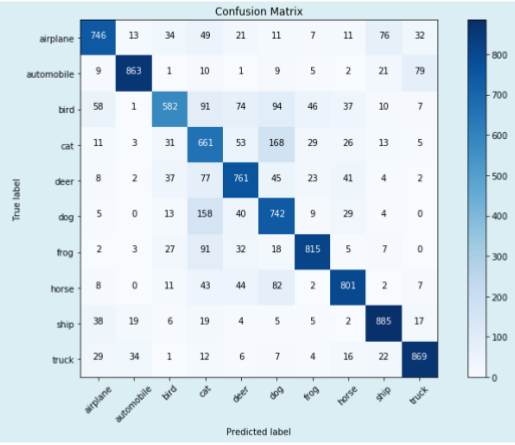
Model	ROC – AUC	Confusion Matrix
CNN		
ResNet		

Figure 13: Best Performing Models ROC and CM

Hardware / Software Used:

- **Machine** : **MacBook Pro (13-inch, 2017, Four Thunderbolt 3 ports)**
- **OS** : **MacOS High Sierra Version 10.13.3**
- **RAM** : **16GB of 2133MHz LPDDR3 memory**
- **Processor** : **3.5 GHz Intel Core i7 processor**

It is possible for CNNs and ResNets to perform even more if we are able to fine tune and experiment with the hyper parameters (no of layers, convolutions, kernels, nodes in dense layers etc.). However, with our limited CPU resources this imposes a great challenge.

Conclusions

The experiments and exercises conducted with the various models ranging from the classical to more modern methods have provided a very comprehensive overview of each model, their setup, benefits and drawbacks. Although we have explored only a tiny fraction of what is available in the field of computer vision, the findings outlined in this report can provide a great insight into some of the most popular methods in practice today. Our findings, clearly indicates great success with Neural Nets especially with Convolutional Neural Networks and Residual Networks when it comes to image classification, but one should always consider all drawbacks both technical and practical when opting their design and implementation choices.

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. DAISY: An Efficient Dense Descriptor Applied to Wide Baseline Stereo Engin Tola, Vincent Lepetit, Pascal Fua IEEE Transactions on Pattern Analysis and Machine Intelligence Vol. 32, Nr. 5, pp. 815 - 830, May 2010
5. Visual categorization with bags of keywords G Csurka, C Dance, L Fan, J Willamowski, C Bray Workshop on statistical learning in computer vision, ECCV 1 (1-22), 1-2
6. A beginners guide to understanding conv nets - https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/
7. Image Classification in Python using BoVW - Ian London https://ianlondon.github.io/blog/how-to-sift-opencv/
8. Bag of Visual Words for Image Classification & recognition https://kushalvyas.github.io/BOV.html
9. Convolutional Neural Network for Image Classification https://cs231n.github.io/convolutional-networks/
10. An overview of Resnets and its variants https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035
11. Resnet example with Keras (Examples) https://github.com/keras-team/keras/blob/master/examples/cifar10_resnet.py
12. ConvNet for Cifar10 by Praneet Kaur http://parneetk.github.io/blog/cnn-cifar10/
13. Normalised RGB http://aishack.in/tutorials/normalized-rgb/
14. Bags of Visual Words Model http://iplab.dmi.unict.it/furnari/teaching/2017-2018/social-media-management/lab2_bag_of_visual_words/index.html
15. Daisy Feature Extraction http://scikit-image.org/docs/dev/auto_examples/features_detection/plot_daisy.html#sphx-glr-auto-examples-features-detection-plot-daisy-py
16. RNN SLTM/GRU http://karpathy.github.io/2015/05/21/rnn-effectiveness/
17. RNN LSTM: http://colah.github.io/posts/2015-08-Understanding-LSTMs/
18. Previous Work http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#43494641522d3130

Appendix

Instructions to run code:

1. Install tensorflow - "pip install tensorflow". Ensure the latest version of tensorflow is installed.
2. Install Keras - "pip install keras". Ensure keras v2.1.3 is installed.
3. Install scipy, numpy and scikit-learn libraries via pip.
4. Install Skimage library
5. Install altair "pip install altair" & Install vega3 (required for altair) - "pip install vega3"
6. Download the cifar10 dataset (cifar-10-python.tar.gz) from https://www.cs.toronto.edu/~kriz/cifar.html and place in "root notebook directory/data" directory & unzip the archive.
7. Keras has been configured to save the model checkpoints in the file system directly. Please ensure the following directory structures are created before running any models
    - root notebook directory/files
    - root notebook directory/files/keras
    - root notebook directory/files/keras/checkpoints
    - root notebook directory/files/keras/checkpoints/resnet_checkpoints
    - root notebook directory/files/keras/checkpoints/tensorboardLogs

    NOTE: tensorboard log command is commented out. If you wish to review tensorboard logs simply uncomment the lines
8. Sequentially go through each cell and execute the code. It is recommended to look for any instances where you can limit the total number of epochs, iterations etc. to ensure a quick runtime.
9. If memory is an issue on the runtime instance, then simply restart the kernel and execute the desired model post the data load cells (simply load the data and run your desired model instead of all models)
10. Keras has been configured to save the best model based on F1-score values. Some sample code has been provided in the appropriate cells (post save) in the event any of the saved models are required to be loaded.
    NOTE: There are several CPU intensive models in this notebook and training all of them will require a lot of time (in excess of 6+ hours)

Tips for speed up

1. Reduce the total number of epochs carried out by the neural network training
2. Reduce the total number of C values when carrying out grid search
3. The gridsearchcv call in this notebook spins off 3 jobs in parallel. If you have additional CPUs at your disposal then recommend increasing parallel job numbers.
4. If you just need to validate the code then simply set the epochs to 1. This will ensure the code executes within 5-10 minutes (depending on the hardware configuration)