

**CZ4042 Neural Networks and Deep Learning**

**Project 1 Report**

**Object Detection and Text Classification using Neural Networks**

Mittal Madhav (U1822408H)

**PART A : OBJECT RECOGNITION**

**INTRODUCTION**

This project aims at building a convolutional neural network to detect and classify objects into one of the 10 categories defined in our dataset. We use the CIFAR-10 dataset to achieve this.   
  
We build this neural network using TensorFlow and Keras. We try to optimize the model and thus increase the validation accuracy by experimenting training by changing several parameters, such as the channel widths, optimizers and use of dropout layers.

The goals of this project are :-

1. Designing a convolutional neural network using Keras “models” library
2. Training the network with specified parameters and plotting the training and testing plots
3. Plotting the feature maps after both the convolutional layers and the pooling layers
4. Using a grid search to find optimal combination of channels for the CNN
5. Finding the best optimizer for our network and plotting the graphs
6. [EXTRA] Rebuilding the neural network to achieve a higher accuracy

**METHODS**

1. **DATASET**

The CIFAR-10 dataset consists of 60000 images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another.

In this project, we use only one of these five training batches, and one test batch. Thus we have 10000 training images, and 2000 testing images. These images are 32 x 32 pixels RGB images.

The data is stored as a 10000x3072 [NumPy](http://numpy.scipy.org/) array of uint8s. Each row of the array stores a 32x32 colour image. The first 1024 entries contain the red channel values, the next 1024 the green, and the final 1024 the blue. The image is stored in row-major order, so that the first 32 entries of the array are the red channel values of the first row of the image.

The labels are stored as a list of 10000 numbers in the range 0-9. The number at index i indicates the label of the ith image in the array data.

A picture containing text

Description automatically generated A picture containing text

Description automatically generated

**Fig 1 : Training and test data summaries**

As it can be seen from Fig 1, the training and test have 10000 and 2000 samples respectively, and each image is a 1-D array of 3072 values. The training data has an approx. 1000 data samples from each class, whereas the test data has an exact 200 sample length for each class.

1. **MODEL**

We use the Keras library to build the convolutional neural network. A summary of the model can be found in fig 2.

Table

Description automatically generated

**Fig 2 : Model Summary**

Layer – by – layer explanation :-

* Since the data is in the form of a long 1-D array, it needs to be reshaped into a format suitable for convolution. Thus, we use the **Reshape layer**.
* **Convolution layers** are used to learn the features in an image. The first few layers of a Deep CNN are called shallow layers and they extract low-level features, whereas the latter layers are called deep layers and they extract higher-level features. Since the model we are building is shallow, thus we cannot extract any high-level features. We are only concerned with the classification part of the model.   
    
  We note that the size of features decreases after the convolution layers. This is because we choose “**Valid Padding**”. “Valid” padding doesn’t add any padding, where “Same” padding adds 0’s to bottom and right rows and columns of the image in order to preserve the size of the images.  
    
  We use the **ReLU activation function** after each convolution layer.
* These convolution layers are always followed by **pooling layers**. Pooling layers can be used for building inner activations that are (slightly) invariant to small translations of the input. They are more useful when we do not want to know where exactly the pattern is, but instead, if it exists at all or not. This is what we want since we are doing object recognition. These pooling layers reduce the breadth and height of feature map through sub sampling.   
    
  We are using **max pooling** here, which means that the layers find the max out of the window and puts that in the new feature image. Note that we use a window size of 2, with a stride of 2, so the max operation is done for a window of 4 pixels, and the final image size is half of the input.
* After the last pooling layer, the data needs to be converted back into a long 1-D array. Therefore we use the **flatten layer**.
* We pass the 1-D array to a **dense (or a full connected) layer**. Each pixel of the flattened image is connected to each neuron in the fully connected layer. These dense layers have non-linear activation function, and thus they can model and mathematical function.
* The last layer is a **Softmax layer.** Softmax’s input is the output of the last fully connected layer, and it outputs the final output of the entire neural network. This output is a probability distribution of all the label class candidates.

**Experiment and Results**

We set the following constants for the whole experiment

* Number of epochs = 1000
* Batch size = 128
* Learning rate = 0.001

**Question 1  
  
(a) Plot the (1) training cost, (2) test cost, (3) training accuracy, and (4) test accuracy against learning epochs. One plot for the costs and one plot for the accuracies.  
  
Chart, histogram

Description automatically generated**

**Fig 3 : Training and test loss vs epoch for the model**

**A picture containing shape

Description automatically generated**

**Fig 4: Training and test accuracy vs epoch for the model**

As it can be seen from figures 3 and 4, after around 400 epochs, the training loss and accuracy seemed to improve, however the test loss and accuracy stagnated. This shows that training beyond 400 epochs is causing overfitting of data.

It can also be noted that the best test accuracy for this model using the parameters mentioned in question 1 was found to be 0.531

**(b) For the first two test images, plot the feature maps at both convolution layers and pooling layers along with the test images.**

A blurry image of a person

Description automatically generated

**Fig 5 : Test images 1 and 2**

The convolution layers perform a convolution operation on the input image and the filter in order to learn the input image’s features better. The result of this convolution operation is also an image, and it is called a feature map. Looking at individual feature maps between successive convolution layers can help us give important insights into how the convolution layers are operating.

The number of feature maps are directly dependent upon the number of channels (or the width) of the filter. Here, we are using 50 and 60 channels in 1st and 2nd convolutional layer, so there will be 50 and 60 feature maps for each image respectively.

The pooling layer is used to generalize the features learnt by the convolution layers, and with a window size and stride of 2, they effectively half the input image size. This is why they appear to be more blurred out form of the convolution layer output features. The pooling may also be used to reduce the feature image size so as to reduce the total trainable parameters.

**A picture containing object, clock, monitor, screen

Description automatically generated  
Fig 6 : Convolution Layer 1 for image 1  
  
  
A picture containing clock

Description automatically generated**

**Fig 7 : Pooling Layer 1 for image 1**

A picture containing scoreboard, clock

Description automatically generated

**Fig 8 : Convolution Layer 2 for image 1**

Background pattern

Description automatically generated

**Fig 9 : Pooling Layer 1 for image 1**

A picture containing object, clock, monitor, screen

Description automatically generatedA picture containing clock

Description automatically generated

A picture containing clock

Description automatically generated

A picture containing background pattern

Description automatically generated

**Fig 10 : Conv Layer 1, Pool Layer 1, Conv Layer 2, Pool Layer 2 for image 2 respectively**

**Question 2**

**Use a grid search to find the optimal combination of the numbers of channels at the convolution layers. Use the test accuracy to determine the optimal combination. Report all 25 accuracies**

**Shape

Description automatically generated**

**Fig 11 : Test Accuracy vs Epochs for all channel combinations**

It can be seen from the graph for all channel combinations, the test accuracy starts to stagnate at around 400-500 epochs. There is only a marginal difference between the accuracy differences for all the combinations.

**Chart, bar chart

Description automatically generated  
Fig 12 : Bar Graph for max accuracies of all channel combinations**

The bar graph confirms that the variations between accuracies by using different channel combinations is not that much. However, the two selected channel lengths of 70 and 100 give us the best accuracy of 0.537 approx. **Thus we set the channel 1 to 70 and channel 2 to 100.**

**Question 3**

**Using the optimal combination found in part (2), train the network by using different optimizers. Plot the costs and accuracies against epochs for each case.**

A picture containing chart

Description automatically generated

**Fig 13 : Test accuracy vs epoch for different optimizers**

Chart, line chart, histogram

Description automatically generated

**Fig 14 : Test loss vs epoch for different optimizers**

Icon

Description automatically generated

**Fig 15 : Bar Graph for test accuracies of different optimizers**

We are now training the CNN with different optimizers with the optimal channel numbers. Momentum, RMSProp and Adam are all examples of adaptive learning rate algorithms, where learning rates are not fixed, but rather change with epochs. It must be noted that SGD Optimizer in this graph includes dropout layers in the model.

**Momentum :** SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another , which are common around local optima. In these scenarios, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum. Momentum is a method that helps accelerate SGD in the relevant direction and dampens oscillations by adding a fraction (here, 0.1). As it can been, SGD with Momentum helps us achieve a slightly better accuracy of 0.541 (as opposed to previous 0.537).

**RMSProp** : RMSprop divides the learning rate by an exponentially decaying average of squared gradients. it is a very robust optimizer which has pseudo curvature information. Additionally, it can deal with stochastic objectives very nicely, making it applicable to mini batch learning.

**Adam :** Combination of Momentum and RMSProp. In the graph, the test accuracy for Adam model remains approximately constant from epochs 100-300, after which there is a huge drop for around 50 epochs (likely due to an overflow). This drop again happens for epochs 570-620, after which it again increases and remains constant. The model before the overflow is considered.

**SGD with Dropout:** Dropout layers help prevent overfitting of a model. During training, some number of layer outputs are randomly ignored or “dropped out.” This has the effect of making the layer be treated-like it has a different number of nodes and connectivity to the prior layer. Dropout layers were added to the dense layers and are used with the SGD optimizer.

It can be seen from fig 15 that all the graphs have more or less similar testing accuracies. Momentum optimizer gives us the best accuracy of 0.541 approx.

**EXTRA PART**

Building a new, deeper convolutional neural network with regularization and dropout layers to assist training.

Table

Description automatically generated

**Fig 16 : New CNN Model Structure**

After reading about the CIFAR-10 dataset from some external websites and their implementations for object detection, I found out that an accuracy of around 50% was not enough, and a much higher classification accuracy could be aimed for. I noticed that usually the pattern was that if we increase the depth of the model with correct input shapes and regularization techniques, then the classification accuracy tends to increase.

I tested this out by adding several convolution layers. I added max pooling layers to generalize the features learnt. I added dropout layers to help prevent overfitting. However, it made the validation graph noisier. So I ended up adding batch normalization layers after every convolution layer. I also added two dense functions so that more complex functions can be learnt.

I tested with several optimizers and ended up choosing Adam as it was working the best with my model.

Shape

Description automatically generated

**Fig 17 : New CNN training and test accuracy**

As we can clearly see from the graph, the new and improved maximum testing accuracy for this model is 0.649 approx., which is around 12% more accurate than our previous best model.

**CONCLUSION**

|  |  |  |
| --- | --- | --- |
| S.No | Model | Accuracy |
|  |  |  |
| 1 | SGD | 53.1% |
| 2 | Momentum | 54.1% |
| 3 | RMSProp | 48.1% |
| 4 | Adam | 47.5% |
| 5 | SGD with Dropouts | 53.5% |
| 6 | [EXTRA] Custom CNN with Adam, dropouts, and regularization | 64.9% |

**PART B : TEXT CLASSIFICATION**

**Introduction**

This project aims at building CNNs and RNNs to detect and classify objects into one of the 15 categories defined in our dataset. The classification is done on character and word levels.

The dataset used in this project contains the first paragraphs collected from Wikipage entries and the corresponding labels about their category. The training dataset contains 5600 entries and test dataset contains 700 entries. The label of an entry is one of the 15 categories such as people, company, schools, etc.

The goals of this project are :-

1. Design a Character CNN Classifier that receives character ids and classifies the input
2. Design a Word CNN Classifier that receives word ids and classifies the input.
3. Design a Character RNN Classifier that receives character ids and classify the input.
4. Design a word RNN classifier that receives word ids and classify the input.
5. Compare the test accuracies and the running times of the networks implemented in parts (1) – (4).
6. For RNN networks implemented in (3) and (4), perform the following experiments :
   1. Replace the GRU layer with (i) a vanilla RNN layer and (ii) a LSTM layer
   2. Increase the number of RNN layers to 2 layers
   3. Add gradient clipping to RNN training with clipping threshold = 2.

**Methods**

**1) Creating the CNN and RNN Models**

The CNN models are used to learn the spatial information of the characters and words with respect to each other, whereas the RNN models are used to learn the temporal information. The models were thus created at character level and word levels. The model structures are displayed in the figure below.

Table

Description automatically generated Table

Description automatically generated

**Table

Description automatically generated Table

Description automatically generated**

**Fig 18 : Model summaries for character and word CNN and RNN respectively**

As it can be seen from the figures, the CNN models *mainly* consist of four layers – two convolution layers and two max pooling layers. As stated in Part-A of the experiment, Convolution layers are used to learn the features in an image, and the pooling layers can be used for building inner activations that are (slightly) invariant to small translations of the input. The RNN models *mainly* consist of a single RNN layer.

The word-level models have an additional Embedding layer of size 20 which is able to learn how to represent words as dense vectors by projecting the words into a continuous vector space. This allows the model to learn relationships between words.

It must be noted that while we can use SparseCategoricalCrossEntropy to calculate the loss for all the CNN and RNN models, we need to use different optimizers. The CNN models were giving accurate results with an SGD optimizer, however, the RNN accuracy and loss plots seemed to stagnate with the SGD optimizer. Thus, the Adam optimizer was used with the RNN models.

For all the models we calculate the loss using SparseCategoricalCrossEntropy and use the SGD optimizer with a learning rate of 0.01 and a batch size of 128. We train each model for 250 epochs and keep track of the train and test set losses and accuracies.

**2) Adding dropouts**

Dropout layers help prevent overfitting of a model. During training, some number of layer outputs are randomly ignored or “dropped out.” This has the effect of making the layer be treated-like it has a different number of nodes and connectivity to the prior layer. Dropout probabilities of 0, 0.4 and 0.8 are used as experimental values.

**3) Using different built-in Keras RNN layers**

There are three built-in layers in Keras – Vanilla RNN, GRU and LSTM. GRU layer was chosen for the RNN models in the first five questions, and then GRU and LSTM layers were chosen as experimental values to find accuracies of combinations of different parameters.

**4) Using different number of RNN layers**

Increasing the number of hidden layers increases the ability of the neural network to learn more complex functions and thus more complex characteristics of the dataset. Only 1 RNN layer was chosen for the RNN models in the first five questions, and then 1 and 2 layers were chosen as experimental values to find accuracies of combinations of different parameters.

**5) Using gradient clipping**

Gradient clipping is a technique to prevent exploding and vanishing gradients in very deep networks, usually in recurrent neural networks. It prevents any gradient to have norm greater than the set threshold and thus the gradients are clipped. Gradient clipping is only used in the last question, with a clipping threshold value of 2.

We set the following constants for the whole experiment

* Number of epochs = 250
* Batch size = 128
* Learning rate = 0.001

**Experiment and Results**

**Question 1  
  
(a) Design a Character CNN Classifier that receives character ids and classifies the input. Plot the entropy cost on the training data and the accuracy on the testing data against training epochs.**

**A picture containing chart

Description automatically generated**

**Fig 19 : Entropy cost plot on training data for Char CNN**

**Chart

Description automatically generated**

**Fig 20 : Accuracy plot on validation data for Char CNN**

As it can be seen from the figures, character CNN has a validation accuracy of 0.694 and a training loss of 1.242. The validation accuracy seems to stagnate after 150 epochs.

**Question 2  
  
(a) Design a Word CNN Classifier that receives character ids and classifies the input. Plot the entropy cost on the training data and the accuracy on the testing data against training epochs.**

Shape

Description automatically generated

**Fig 21 : Entropy cost plot on training data for Word CNN**

A picture containing background pattern

Description automatically generated

**Fig 22 : Accuracy plot on validation data for Word CNN**

As it can be seen from the figures, Word CNN has a validation accuracy of 0.344 and a training loss of 1.729. The validation accuracy shoots up only after 200 epochs. The model has not converged at 250 epochs, and thus has a very high loss of 1.729, and a very low accuracy of 0.344. Had the model been trained for more epochs, it might have had achieved a greater (or at least similar) accuracy than Char CNN.

**Question 3  
  
(a) Design a Character RNN Classifier that receives character ids and classifies the input. Plot the entropy cost on the training data and the accuracy on the testing data against training epochs.**

A picture containing shape

Description automatically generated

**Fig 23 : Entropy cost plot on training data for Char RNN**

A picture containing shape

Description automatically generated

**Fig 24 : Accuracy plot on validation data for Word RNN**

As it can be seen from the figures, Char RNN has a validation accuracy of 0.714 and a training loss of 0.353. The validation accuracy seems to stagnate after just 25 epochs.

**Question 4  
  
(a) Design a Word RNN Classifier that receives character ids and classifies the input. Plot the entropy cost on the training data and the accuracy on the testing data against training epochs.**

Shape

Description automatically generated

**Fig 23 : Entropy cost plot on training data for Word RNN**

A picture containing graphical user interface

Description automatically generated

**Fig 24 : Accuracy plot on validation data for Word RNN**

As it can be seen from the figures, Word RNN has a validation accuracy of 0.864 and a training loss of 1.728e-06. The validation accuracy seems to stagnate after just 25 epochs.

**Question 5  
  
(a) Compare the test accuracies and the running times of the networks implemented in parts (1) – (4).**

**A picture containing chart

Description automatically generated**

**Fig 25 : Comparison of running times of different models**

As it can see from the figure, the RNN models take much more time to train as compared to the CNN models. Additionally, the Word models take slightly more time to train than the character models.

**A picture containing shape

Description automatically generated**

**Fig 26 : Comparison of validation accuracies of different models**

As it can be seen, for the character models, both RNN and CNN have similar accuracies. However, for word models, the CNN models do not seem to be able to converge within 250 epochs, whereas the RNN model converges on epoch 50. Thus, RNN word model has better accuracy than RNN character model.

**(b) Experiment with adding dropout to the layers of networks in parts (1) – (4), and report the test accuracies. Compare and comment on the accuracies of the networks with/without dropout.**

A picture containing chart

Description automatically generated

**Fig 27 : Comparison of validation accuracies of Char-CNN models with different dropout rates**

An increase in dropout rate seems to reduce the validation accuracy for the Character CNN models.

A picture containing chart

Description automatically generated

**Fig 28 : Comparison of validation accuracies of Word-CNN models with different dropout rates**

For Word CNN, we can see that for a dropout rate of 0.4, it starts to converge at around 220 epochs, however the model has not converged fully yet. For a dropout of 0.0, the model has not converged at 250 epochs, and validation accuracy is still increasing. For a dropout of 0.8, the model’s testing accuracy seems to stagnate.

A picture containing chart

Description automatically generated

**Fig 29 : Comparison of validation accuracies of Char-RNN models with different dropout rates**

An increase in dropout rate seems to reduce the validation accuracy for the Character RNN models.

A picture containing text

Description automatically generated

**Fig 30 : Comparison of validation accuracies of Word-RNN models with different dropout rates**

An increase in dropout rate seems to reduce the validation accuracy for the Word RNN models.

Thus, on an average, higher dropout rates tend to decrease the validation accuracy and make the training noisier for our CNN and RNN models. This might be because the models were not overfitting and thus the dropout layer just slowed down the training process.

**Question 6  
  
(a) For RNN networks implemented in (3) and (4), perform the following experiments with the aim of improving performances, compare the accuracies and report your findings:**

**a. Replace the GRU layer with (i) a vanilla RNN layer and (ii) a LSTM layer**

**b. Increase the number of RNN layers to 2 layers**

**c. Add gradient clipping to RNN training with clipping threshold = 2.**

**A picture containing calendar

Description automatically generated**

**Fig 31 : Graphs for all combinations of parameters for Word RNN**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S. No | Type of Layer | Number of Layers | Clipping Threshold | Accuracy |
| 1 (a) | GRU | 1 | None | **0.821** |
| (b) |  |  | 2 | **0.828** |
| (c) |  | 2 | None | **0.870** |
| (d) |  |  | 2 | **0.847** |
| 2 (a) | Vanilla RNN | 1 | None | **0.835** |
| (b) |  |  | 2 | **0.845** |
| (c) |  | 2 | None | **0.889** |
| (d) |  |  | 2 | **0.881** |
| 3 (a) | LTSM | 1 | None | **0.871** |
| (b) |  |  | 2 | **0.821** |
| (c) |  | 2 | None | **0.847** |
| (d) |  |  | 2 | **0.862** |

**Table 1 : Comparison of validation accuracies of all combinations of parameters for Word RNN**

As we can see from the above graph and table, the validation accuracies for all of combinations of Word RNN parameters range from 82.1% - 88.9%. The maximum test accuracy was achieved by **the Vanilla RNN model with 2 layers and no clipping**, and it was **88.9%**.

It can be noted that for GRU and Vanilla RNN models, the accuracy increased with an increase in number of layers, but decreased when we introduced clipping. For LTSM layer, the accuracy decreased when we introduced the second layer, but increased when we introduced clipping.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S. No | Type of Layer | Number of Layers | Clipping Threshold | Accuracy |
| 1 (a) | GRU | 1 | None | **0.831** |
| (b) |  |  | 2 | **0.834** |
| (c) |  | 2 | None | **0.544** |
| (d) |  |  | 2 | **0.835** |
| 2 (a) | Vanilla RNN | 1 | None | **0.829** |
| (b) |  |  | 2 | **0.828** |
| (c) |  | 2 | None | **0.824** |
| (d) |  |  | 2 | **0.821** |
| 3 (a) | LTSM | 1 | None | **0.860** |
| (b) |  |  | 2 | **0.824** |
| (c) |  | 2 | None | **0.824** |
| (d) |  |  | 2 | **0.826** |

**Table 2 : Comparison of validation accuracies of all combinations of parameters for Char RNN**

As it can be seen from the table, the validation accuracies for all combinations of character RNN parameters range from a value of 82.1% - 86.0%. The maximum test accuracy was achieved by **the LTSM RNN model with 1 layers and no clipping**, and it was **86.0%**

It can be noted that when there are two GRU layers, then the model seems to have the lowest accuracy of 54.4%. But when we include clipping, it increases the validation accuracy to 83.5%. This could be because the neural network may be prone to exploding gradients, that is, the gradient is becoming too large and the error gradients are accumulating, resulting in an unstable network. This is prevented by the clipping gradient.