**Team:** Harshwardhan Yadav, Prekshi Vyas, Ivan Zhao

#### **Computer Vision (Traditional Pipeline)**

**Data preprocessing.** Describe in detail your data preprocessing, including:

* What software packages you used

| **Software Package** | **Purpose** |
| --- | --- |
| Pickle | Loading the dataset |
| Numpy | Performing operations on the images |
| Matplotlib | Visualization |
| PIL (Python Imaging Library) | Converting numpy arrays to images. |
| OpenCV | Applying Gaussian Blurring to images |
| Scikit-Image Exposure | Applying Histogram Equalization |
| Pytorch | Creating / Manipulating Image Tensors  Image Transformations (Rotation, Flipping, Brightness |

* How you set up the training/validation/test set:

The CIFAR-10 dataset contains 6 batches including a “test\_batch”, each containing 10,000 images of randomly distributed classes. There are 10 classes in total.

To create our dataset split we do the following:

1. Training set : To create the training set we are combining 4 training batches and we observed that the combined set has reasonably balanced distribution.
2. Validation set: One of the 6 batches has been saved into the validation set.
3. Test set: The batch called “test\_batch” has been assigned to the test set.

* Any data preprocessing you used.

We used the below techniques to preprocess our data:

1. Normalization: To scale the pixel values in range [0, 1].
2. Gaussian Blurring: To reduce noise and smoothen images.
3. Histogram Equalization: We tried applying to the images but not all the images were enhanced and some rather became more distorted. So, we dropped it’s application.
4. We also tried sharpening the images but unnecessary portions of the image got sharpened, again distorting the images further.
5. We also tried denoising using fastNIMeansDenoisingColored, but that takes the portions of the image that belong to the object of interest as noise. Hence, we only stick to the Gaussian blurring to remove noise. Gaussian blurring did not distort the image’s object of interest.

**Machine learning models.** Describe in detail how you trained your machine learning models (namely, softmax regression), including:

* What software packages you used
* PyTorch
* torch.nn: PyTorch module to define the SoftmaxRegression class and the loss function (nn.NLLLoss()).
* torch.optim: To define the optimizer for updating the model parameters during the training loop.
* torch.Tensor: To perform tensor operations, such as matrix multiplication and logarithmic softmax.
* Values of key hyperparameters that you chose.

a. Learning Rates:

- Explored 5 different learning rates: [0.001, 0.01, 0.04, 0.08, 0.1]

b. Epochs:

- Conducted experiments over 3 epochs for each model. [5, 25, 50]

c. Optimizers:

- Consider 2 optimizers: [Adam, SGD].

Additionally we investigated following two scenarios:

1. Dataset shift applied to the training set. These include random rotations (up to 20 degrees), horizontal and vertical flips, and color jitter with a brightness variation of 0.2, to introduce diversity in the training data.

2. No dataset shift applied to the training set.

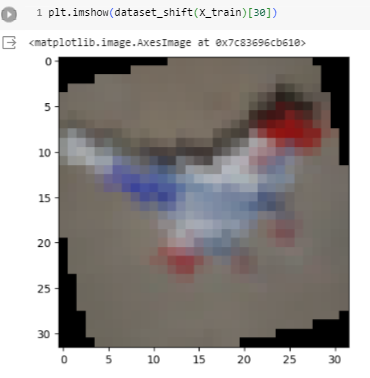
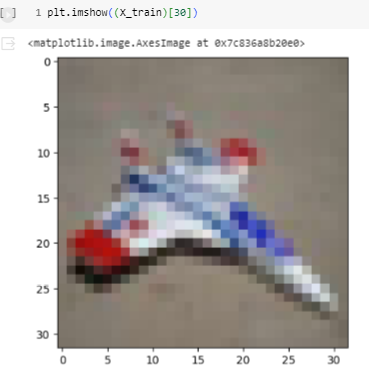
| **Hyperparameter** | **Values** |
| --- | --- |
| Learning rate | [0.001, 0.01, 0.04, 0.08, 0.1] |
| Optimizers | SGD and Adam |
| Epochs | [5, 25, 50] |

**Some examples of images before and after preprocessing (Normalization + Gaussian blurring):**

A collage of images of birds

Description automatically generated

**A sample before and after Dataset shift:**



* For hyperparameters that you varied, how did you vary them

**Epochs**: the number of epochs we selected were 5, 25 and 50 because the loss and the accuracy of the model didn’t show any significant improvement beyond ~25 epochs. Due to this, we can see that the model tends to converge pretty quickly and adding more epochs does not justify the improvement in performance of the model.

**Learning Rates:** We tried different learning rates such as 0.001, 0.01, 0.04, 0.08 and 0.1 for all the models to cover a standard range which would give us information on what learning rates give a smooth curve and which end up providing fluctuations.

**Optimizer:** We used two optimizers Adam and SGD to assess model performance.

**Results and discussion.**

Provide tables and plots showing your results, including:

* Training/validation/test results
* Results on how test performance varies with hyperparameter choices

Since we have 5 different values for learning rate, 3 epochs, 2 optimizers and 2 different kinds of data (**X\_train → no dataset shift applied and X\_train\_ds → dataset shift applied**) , the total number of models trained: 5×3×2×2=60.

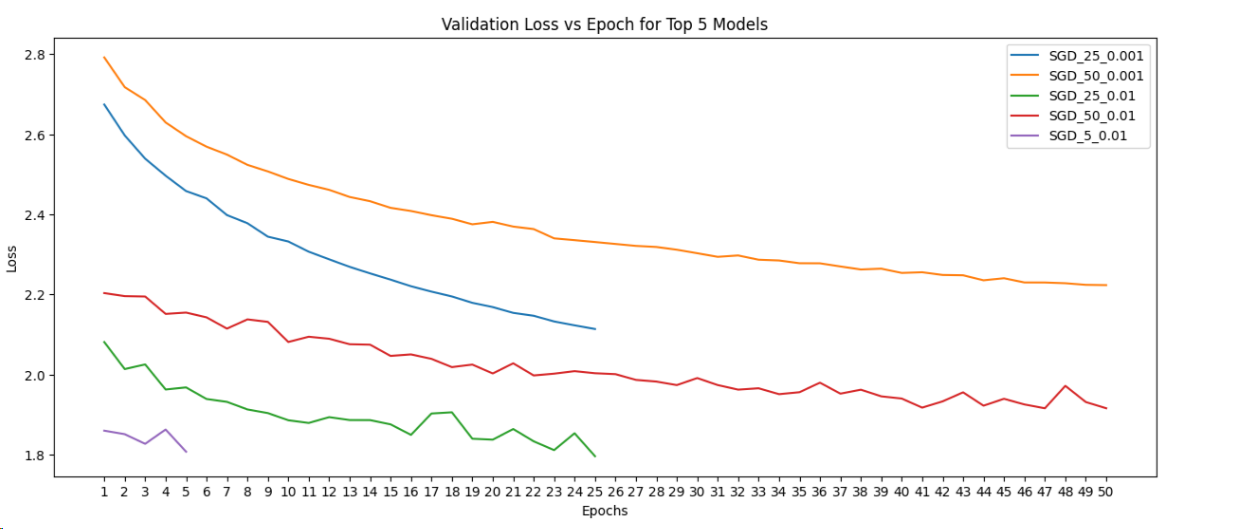
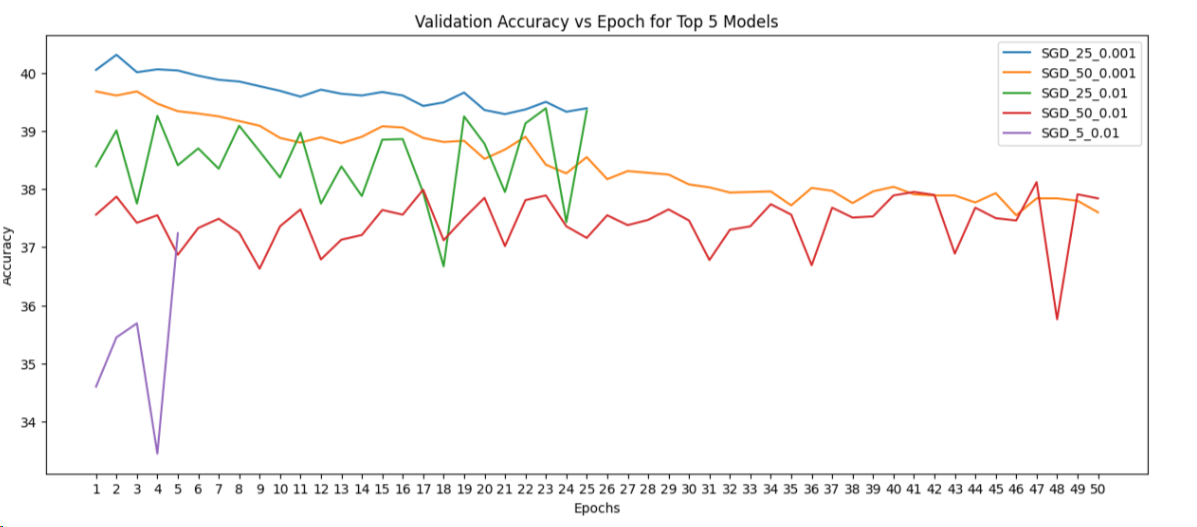
For each model, we have evaluated its performance on validation sets to observe how well it learned from the data and performance on the test set to see how well it generalized to unseen data.

Based on the weighted f1 scores calculated on validation dataset, we have chosen the **top 5 models** to be:

* **Table:**

| SoftMax Model Label | Dataset | Optimizer | Epochs | Learning rate (lr) | F1-Score | Rank (Based on F1-Score) |
| --- | --- | --- | --- | --- | --- | --- |
| SGD\_25\_0.001 | X\_train | SGD | 25 | 0.001 | 0.3953 | 1 |
| SGD\_50\_0.001 | X\_train | SGD | 50 | 0.001 | 0.3856 | 2 |
| SGD\_25\_0.01 | X\_train | SGD | 25 | 0.01 | 0.3822 | 3 |
| SGD\_50\_0.01 | X\_train | SGD | 50 | 0.01 | 0.3706 | 4 |
| SGD\_5\_0.01 | X\_train | SGD | 5 | 0.01 | 0.3545 | 5 |

**Graphs for Validation accuracies and losses for these 5 models**

****

**Classification Report for the top 5 models by Rank (scores rounded to nearest integers):**

A screenshot of a computer

Description automatically generated

**1. SGD\_25\_0.001**

**2. SGD\_50\_0.001**

A screenshot of a computer

Description automatically generated

**3. SGD\_25\_0.01**A screenshot of a computer screen

Description automatically generated

A screenshot of a computer screen

Description automatically generated

**4. SGD\_50\_0.01:**

A screenshot of a computer

Description automatically generated

**5. SGD\_5\_0.01:**

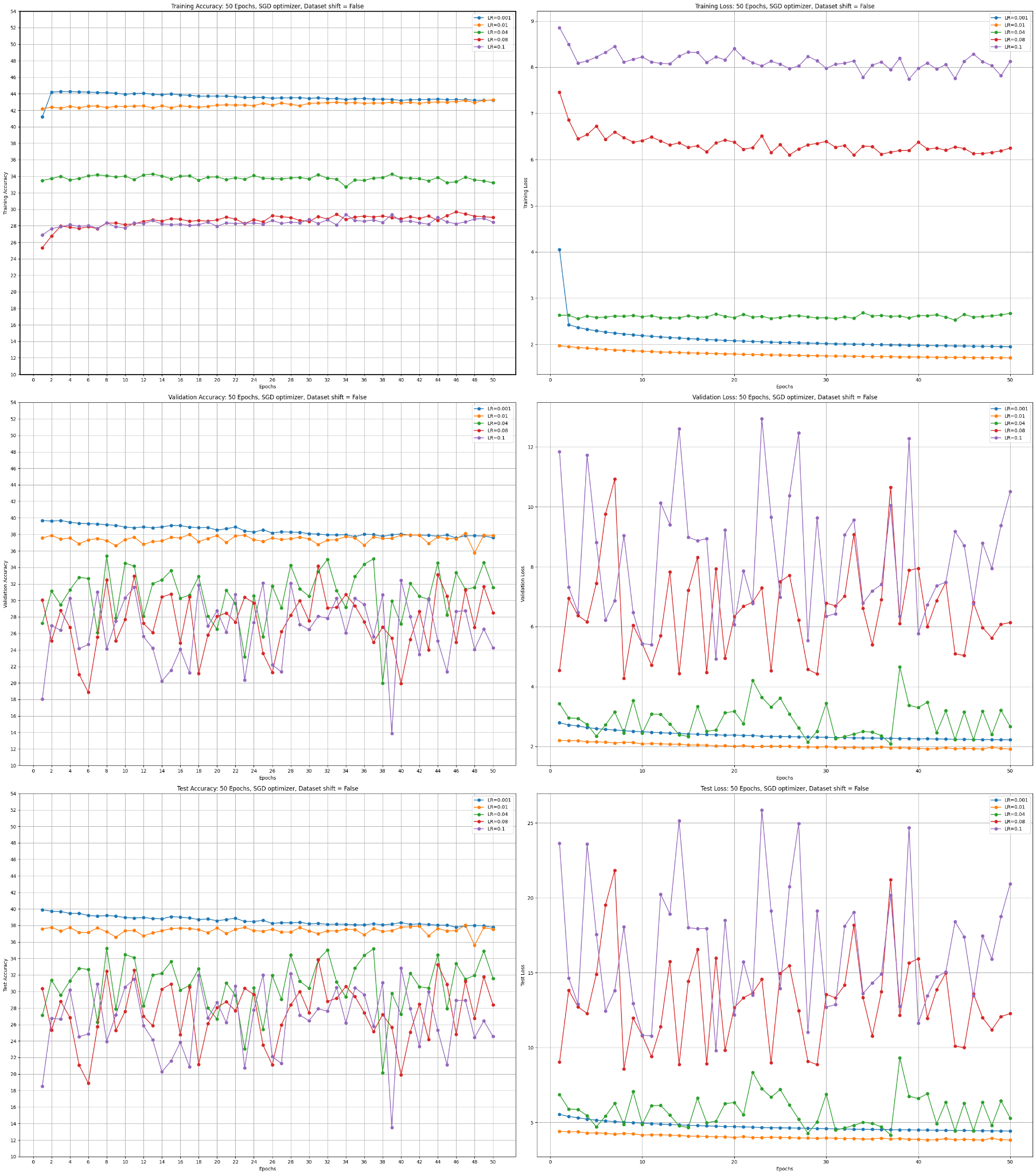
**Plots for Training and Validation Accuracies and Losses**

1. **SGD\_25 (i.e model with optimizer SGD, epochs=25) and different learning rates.**



From the validation plots we can infer that learning rates of 0.001 is the best learning rate for 25 epochs. The learning rate of 0.01 has almost similar performance as learning rate 0.001 for 25 epochs but it has some fluctuations on validation accuracy. Other learning rates 0.04, 0.08 and 0.1 are too high for the model.

**2. SGD\_50 (i.e model with optimizer SGD, epochs=50) and different learning rates:**

****

From the validation plots we can infer that learning rates of 0.001 is the best learning rate overall. The learning rate of 0.01 has almost similar performance as learning rate 0.001 but it has some fluctuations on validation accuracy. Other learning rates 0.04, 0.08 and 0.1 are too high for the model.

**3. SGD\_5 (i.e model with optimizer SGD, epochs=5) and different learning rates.**



From the validation plots we can infer that the learning rate of 0.01 is the best learning rate for 5 epochs. The learning rate of 0.001 has lesser accuracy despite no fluctuations. Other learning rates 0.04, 0.08 and 0.1 have lesser accuracy and higher loss.

But as epochs increase, the learning rate of 0.001 proves to perform better as we saw in the above 2 sections.

**Plots for Testing Accuracies and Losses**

1. **5 epochs SGD Optimizer and Dataset shift = False**



From the test plots we can infer that the learning rate of 0.01 is the best learning rate for 5 epochs (but not overall as we will see below) The learning rate of 0.001 has lesser accuracy despite no fluctuations. Other learning rates 0.04, 0.08 and 0.1 have lesser accuracy and higher loss. This is consistent with the validation plots.

1. **25 epochs SGD Optimizer and Dataset shift = False** A screenshot of a graph

   Description automatically generated

From the test plots we can infer that learning rates of 0.001 is the best learning rate overall. The learning rate of 0.01 has almost similar performance as learning rate 0.001 for 25 epochs but it has some fluctuations on test accuracy. Other learning rates 0.04, 0.08 and 0.1 are also too high for the model. This is consistent with the validation plots for 25 epochs.

1. **50 epochs SGD Optimizer and Dataset shift = False**

A screenshot of a graph

Description automatically generated

From the test plots we can infer that the learning rate of 0.001 is the best learning rate for 50 epochs (but not as good as 25 epochs as we saw above). The learning rate of 0.01 has almost similar performance as learning rate 0.001 but it has some fluctuations on test accuracy. Other learning rates 0.04, 0.08 and 0.1 are also too high for the model. This is consistent with the validation plots results for 50 epochs.

**Other observations:**

Models where we applied data set shifts and/or Adam optimizer was used performed poorly as compared to ones where no dataset shifts were applied and SGD optimizer was used.

#### A screenshot of a computer screen Description automatically generated

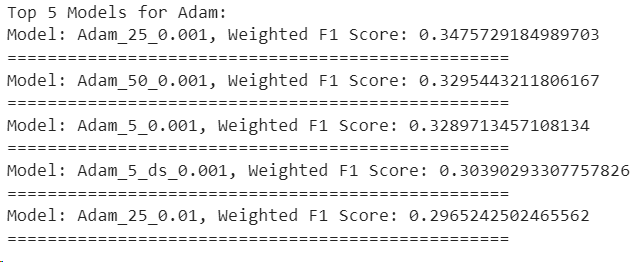
**Test Accuracy and Loss plot for few of the poor models viz. Adam\_50\_ds\_0.1, Adam\_50\_ds\_0.08** (Adam Optimizer with dataset shift applied and 50 epochs with learning rates 0.1 and 0.08)

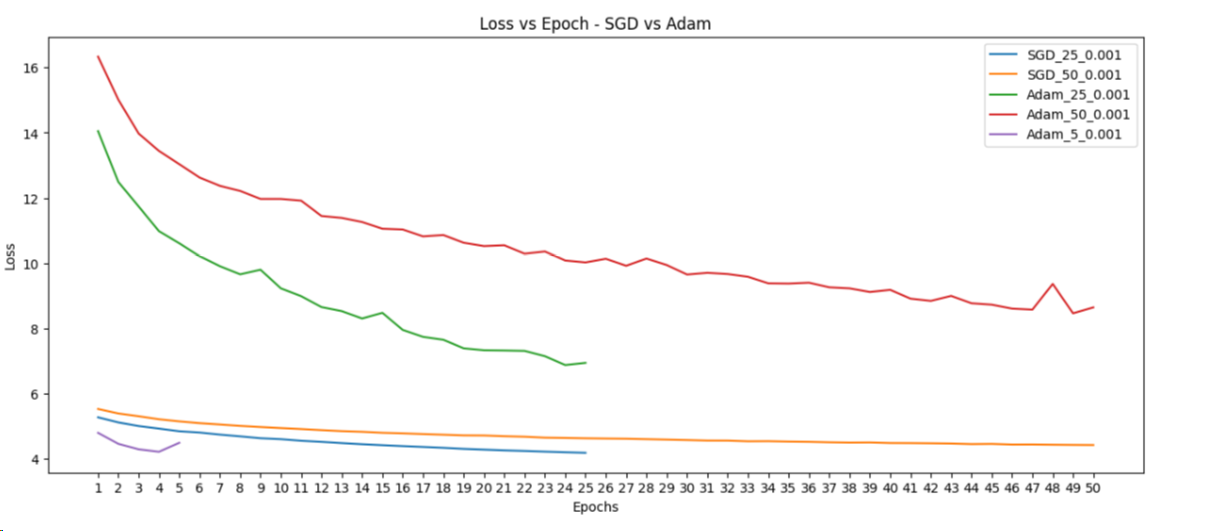
A screenshot of a graph

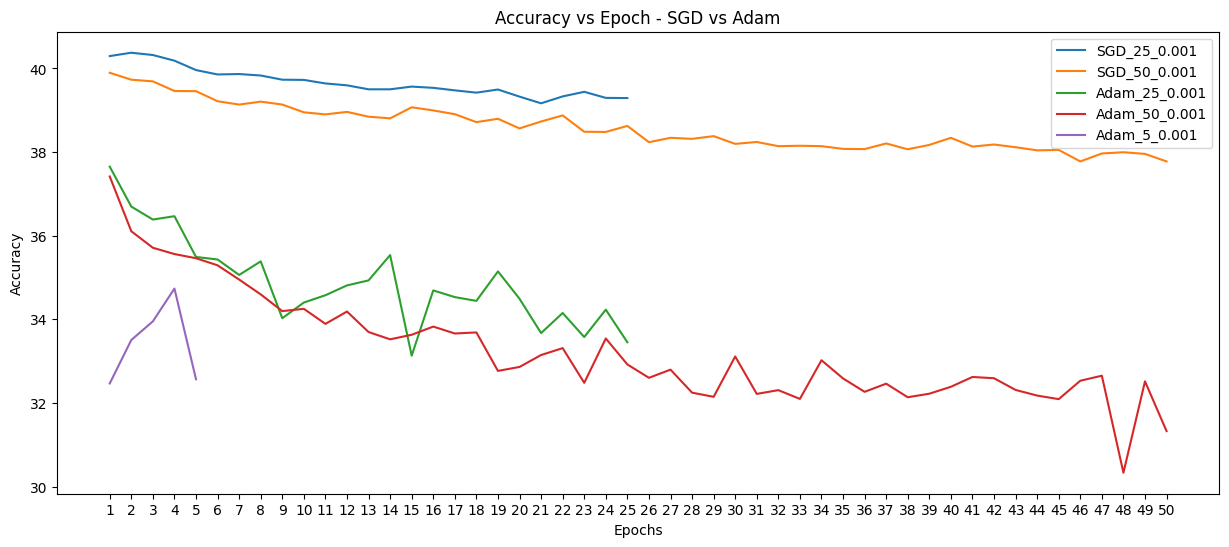
Description automatically generated

We can infer from these graphs that except learning rate = 0.001, all other learning rates are not stable. Even with learning rate = 0.001 the accuracy is less when using Adam optimizer and Dataset shifts.

**Comparing some best models from Adam and SGD Optimizer:**







We can infer from these graphs that SGD outperforms Adam for the considered learning rates.

The trends in our results are as follows:

1. The models tend to learn better when no dataset shifts are applied.
2. The Softmax Regression model is able to provide ~40% accuracy with a smaller number of epochs as well.
3. The learning rate 0.001 is ideal to reach the results faster in a smaller number of epochs (#25)
4. The SGD optimizer outperforms the Adam optimizer as can be seen from the classification reports and the accuracy curves in the results section above. One possible reason can be that Adam is more sensitive to learning rate values. For future considerations we can try with smaller learning rates to get better performance with Adam.

One possible reason why softmax regression may not yield high accuracy is because the model is only a single layer. This could mean that the model has low complexity which leads to a decrease in performance. In addition, the model takes in a flattened image such that the 3 dimensional image is turned into a 1 dimensional vector. As such, the spatial structure of an image is lost in the input to our model. By ignoring the location associations and spatial neighborhoods, the model may be losing meaningful features, which hinders performance. However, our model is still better than random. Since there are 10 classes, a random guess is expected to be correct 10% of the time. With an accuracy of 40% we see that our model is significantly better than random.

#### **Natural Language Processing (Traditional Pipeline)**

**Data preprocessing.** Describe in detail your data preprocessing, including:

* What software packages you used

| **Software Package** | **Purpose** |
| --- | --- |
| re, string | Working with regex and string operations. |
| Kaggle, zipfile | Loading the Kaggle dataset and unzipping. |
| Pandas | Reading the data from csv and storing it in pandas data frame. |
| nltk | For data preprocessing tasks such as tokenization, lemmatization and stop words removal. |
| bs4 (Beautiful Soup) | Cleaning html tags from the data. |
| sklearn.model\_selection.train\_test\_split | For creating training, validation and test datasets. |

* How you set up the training/validation/test sets

Using the train\_test\_split function we have first split the data into 75% training and 25% “combined” sets. The “combined” set has been further split into test and validation sets.

Hence, the training set has 75% of 50,000 samples i.e., 37,500 samples; validation set and test set each contain 6,250 samples.

* Any data preprocessing you used

The following functions have been used to pre-process the data:

1. Cleaning: Removing punctuation marks, non- alphanumeric characters, more than 1 consecutive spaces, trailing or leading spaces.
2. Converting to lowercase.
3. Removing html tags (alternatives considered were regex and lxml library but not as robust).
4. Removing stop words.
5. Applying Lemmatization (alternative considered was stemming but lemmatization provides more meaningful results).
6. Checks for imbalance in data: since data is balanced no methods performed to address this issue.
7. One hot encoding for the target values (sentiment = 1 if positive and 0 if negative).

* Describe your feature engineering in detail

**Dataset shifts:** We train the models on two formats of the dataset. The first includes short sentences and the second includes long sentences. The short sentences are formed by setting the length of each document in the dataset to be (M – (1/3) S), where M is the mean length of documents and S is the standard deviation of the length of documents in the training dataset. Whereas, the long sentences are formed by setting the length of each document in the dataset to be (M + (1/3) S). With this, the short documents are 89 words each and the long documents are 150 words each.

For feature engineering, we used the TF-IDF vectorization to analyze the words that are important in specific documents but not very common in the entire dataset. We used sklearn’s TfidfVectorizer vectorizer for this with the minimum document frequency and maximum document frequency set to 0 and 1, respectively, to take into account all the words. N-gram range used was (1, 3) i.e., unigrams, bigrams and trigrams used to represent documents in the feature space. But this gave a feature size of 5807919 for each document. Since, our model is weak and also the feature dimension is too large we did not consider TF-IDF. Also, it takes considerable time and memory to train on this feature extraction.

Due to this we test two different methods of feature extraction:

1. Glove Embeddings with averaging: - Here, we take the 50-dimensional glove word vectors trained on 6 billion tokens to represent each word. Then, to find the representation of the document we average the embeddings of all the words to create an embedding of 50-dimensional vector for a single document. Hence, if the document has m words each of which is a 50-dimensional word vector, the document would also be represented by a 50-dimensional vector in the feature space which is obtained by the averaging of the m 50-dimensional word vectors. Hence, the training data after this method of feature extraction becomes a matrix of size (37500, 50), where 37500 is the number of documents and 50 is the feature representation of each document.
2. Glove Embeddings with concatenation + PCA: - Here, we represent each document by concatenating the word embeddings of each word in the document. For example, if the document has m words, then the document would be represented by a (m\*50) dimensional word vector in the feature space that is formed by the concatenation of the word vectors of m words. This gave a training dataset of (37500, 4450) for short sentences whereas (37500, 7500) for long sentences.  
   Also, since this would cause a very high dimensional feature space representation of the document, we apply PCA to reduce the dimension of each document’s feature vector (originally 4450/7500-dimensional) to 500-dimensional vector.  
   The test accuracy with and without PCA applied on the dataset was 0.71 and 0.73. We can see that the model still provides similar/good enough accuracy with the dataset after PCA is applied and the model is trained significantly faster as compared to when trained on a dataset without PCA. Hence, we decided to apply PCA when we represent each document as a concatenation of the word embeddings.

For applying the above steps on a document, we add special tokens called “<PAD>” tokens to the document to reach the desired number of words. The embedding of the <PAD> token is set to a zero vector. Also, if the document contains words that are not seen in the Glove embeddings, we also replace those words with the <PAD> token.

Hence, we create four combinations of datasets to train the models on, they are as below:

1. X\_SA: - Original dataset X after considering **S**hort sentences + **A**veraging of word embeddings to represent each document. Hence, the training dataset is **X\_train\_SA**, validation dataset is **X\_val\_SA** and testing dataset is **X\_test\_SA**.
2. X\_SCR: - Original dataset X after considering **S**hort sentences + (PCA **R**eduction after **C**oncatenation of words embeddings) to represent each document. Hence, the training dataset is **X\_train\_SCR**, validation dataset is **X\_val\_SCR** and testing dataset is **X\_test\_SCR**.
3. X\_LA: - Original dataset X after considering **L**ong sentences + **A**veraging of word embeddings to represent each document. Hence, the training dataset is **X\_train\_LA**, validation dataset is **X\_val\_LA** and testing dataset is **X\_test\_LA**.
4. X\_LCR: - Original dataset X after considering **L**ong sentences + (PCA **R**eduction after **C**oncatenation of word embeddings) to represent each document. Hence, the training dataset is **X\_train\_LCR**, validation dataset is **X\_val\_LCR** and testing dataset is **X\_test\_LCR**.

**Machine learning models.** Describe in detail how you trained your machine learning models (namely, softmax regression), including:

* What software packages you used

We used the following software packages for the machine learning models:

1. PyTorch: To create the AdaBoost classifier with Logistic regression as a weak classifier.
2. SkLearn: To compare sklearn’s Logistic Regression model with the created AdaBoost classifier.
3. XgBoost: To compare the standard XgBoost library’s XGBClassifier with the created AdaBoost classifier.

* What machine learning models you trained

We trained three models initially to compare their relative performance. These include the three models defined above. Upon comparing the created AdaBoost, Logistic Regression and XgBoost, we found that the three models do not differ much in their capacity to learn the distribution of the training data. Also, the validation and test set also form a similar distribution to the train set. Below, we show the classification report for the AdaBoost, XgBoost and Logistic Regression all trained and tested on the X\_SCR dataset with a learning rate of 0.01 and T (# of weak learners) = 25 for AdaBoost and XgBoost and max\_iter = 1000 for Logistic Regression:  
A black background with white text

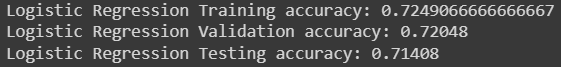
Description automatically generated  
A screenshot of a computer screen

Description automatically generated

A black background with white text

Description automatically generated  
A screenshot of a computer

Description automatically generated



A screenshot of a computer

Description automatically generated

From this comparison, we can see that the AdaBoost model and Logistic Regression model provide similar performance on the dataset and outperform the standard XGBClassifier. Hence, we have considered AdaBoost model architecture as the main traditional network pipeline for the given NLP Task. The main reason for selection of AdaBoost over Logistic Regression of sklearn is that for certain parameter choices, AdaBoost gives a boost of approx. 1 - 1.5% in accuracy over sklearn’s Logistic Regression.

* Values of key hyperparameters that you chose

The key hyperparameters of our AdaBoost model includes the following: -

1. **T**: # of weak learners. We choose the number of weak learners to be 5, 25 and 50.
2. **lr**: the learning rate of the weak learner. We choose learning rates to be 0.1, 0.01 and 0.001.
3. **lambda**: the regularization parameter for the weak learner. We keep the regularization parameter to be zero.
4. **optimizer**: the optimizer used while training. We train and test the model on two different types of optimizers namely Adam and SGD+momentum. The decay rates for the first and second momentum in Adam are 0.9 and 0.999 with epsilon = 1e-8. In SGD, the momentum we use is 0.9. Again, for both optimizers the weight decay is set to zero i.e., we do not use any regularization in AdaBoost.

* For hyperparameters that you varied, how did you vary them

The hyperparameters were varied with the following reasoning in mind: -

**T**: the number of weak learners we selected were 5, 25 and 50 because the loss and the accuracy of the model tends to stagnate after 15-20 weak learners. Due to this, we can see that the model tends to converge pretty quickly and adding more learners does not justify the improvement in performance of the model. This can be seen below when the model is trained on 100 weak learners:

1. Loss on validation set while training  
   A graph of a number of weak learners

   Description automatically generated
2. Accuracy on validation set while training  
   A graph with blue lines

   Description automatically generated

**lr**: We tried training with different learning rates such as 0.5, 0.1, 0.05, 0.01 and 0.001 on the same dataset for all the three models. The performance usually stayed the same with the accuracy being capped at around 73%, the difference was just the number of weak learners in AdaBoost and XgBoost, and the epochs in Logistic Regression is reduced for the model to achieve the said accuracy with a higher learning rate. Finally, for the AdaBoost model we then selected variations in three learning rates i.e., 0.1, 0.01 and 0.001. We show that the best learning rate turned out to be 0.1 with Adam optimizer in the results section.

**lambda**: We tried training the model with and without the regularization term with L2 regularization. With the regularization term we tried different values such as 10, 1000, 10000 and 100000. But when trained with regularization the model turned out to give worse results on the test set as compared to when trained without regularization. Hence, this would mean that the model does not have great learning capacity for the task and it does not overfit the training data. Thus, we decided to drop the regularization to improve the accuracy of the model.

**optimizer**: We have selected two different optimizers to train the model. We selected SGD+momentum as our primary optimizer but that learns too slowly and does not converge quickly when compared to XgBoost and LogisticRegression. Then we tried with Adam and that seemed to learn quickly and provide at par results with both these standard methods. Hence, in our analysis we include two different optimizers, Adam and SGD+momentum. We will show this difference of optimizers in the results section.

* What subsets of features did you consider

We have described the subset of features included in detail when we described the feature engineering in detail above. But, as a summary, we have considered using four different combinations of feature selection. The first one includes averaging the embeddings of all the words in a document to represent the document with the document length set to 89 words. The second one includes concatenating the embeddings of all the words in a document and then applying PCA for reduction to a 500-dimensional vector to represent the document with the document length set to 89 words. The third one includes averaging the embeddings of all the words in a document to represent the document with the document length set to 150 words. The fourth one includes concatenating the embeddings of all the words in a document and then applying PCA for reduction to a 500-dimensional vector to represent the document with the document length set to 150 words.

**Results and discussion.** Provide tables and plots showing your results, including:

We select three different choices of the hyperparameter **T**, three different choices of hyperparameter **lr** and two different choices of **optimizer** as shown above.

Thus, we have 4 datasets to train on, 3 **T** values, 3 **lr** values and 2 **optimizers**. This way we train 4 \* 3 \* 3 \* 2 = 72 different models.

**Out of these 72 different models, we select the best 5 models based on the F1-score of the models. We also include an SGD model to show that there is quite a difference between SGD and Adam optimizer.** Below we provide the results for these five best models and provide our reasoning for the results obtained:

* Training/validation/test results:  
  Now, we firstly train on the training datasets of all the 72 model combinations and then select the top 5 models out of these when tested on the corresponding validation datasets based on the F1-score of these 5 models on validation dataset. The top 5 models + an SGD model selected this way include the following:  
  **Table:**

| Model | Dataset | Optimizer | T | Learning rate (lr) | F1-Score | Rank (Based on F1-Score) |
| --- | --- | --- | --- | --- | --- | --- |
| AdaBoost | X\_train\_LA | Adam | 25 | 0.1 | 0.7498 | 1 |
| X\_train\_LA | Adam | 50 | 0.1 | 0.7494 | 2 |
| X\_train\_LA | Adam | 5 | 0.1 | 0.7484 | 3 |
| X\_train\_LA | Adam | 25 | 0.01 | 0.7416 | 4 |
| X\_train\_SA | Adam | 50 | 0.1 | 0.7394 | 5 |
| X\_train\_SA | SGD | 50 | 0.001 | 0.4524 | 51 |

The cost of these 5 best models per epochs (number of weak learners) while training on validation set is as below:  
  
A graph with lines and text

Description automatically generated with medium confidence

When Adam’s cost is compared to SGD:  
  
A graph with red lines

Description automatically generated  
The accuracy on validation data of these 5 best models per epochs (number of weak learners) while training is as below:

A graph with different colored lines

Description automatically generated

When ADAM accuracy is compared to SGD:  
A graph showing a graph

Description automatically generated  
  
Next, we provide the **classification report** of these models on the test set:  
1) Rank – 1 model(adam\_25\_0.1):  
A screenshot of a graph

Description automatically generated

2) Rank – 2 model(adam\_50\_0.1):  
A screenshot of a computer

Description automatically generated

3) Rank – 3 model(adam\_5\_0.1):

A screenshot of a computer

Description automatically generated  
  
4) Rank – 4 model(adam\_25\_0.01):  
A screenshot of a computer

Description automatically generated  
  
5) Rank - 5 model(adam\_50\_0.1):  
A screenshot of a computer

Description automatically generated

6) The SGD model(sgd\_50\_0.001):  
A screenshot of a computer

Description automatically generated  
  
  
Next we discuss these results below.

* Results on how test performance varies with hyperparameter choices:  
  From the accuracy curves, classification reports and the table above we can see that the model learns quicker and also gives slightly better accuracy with a higher learning rate of 0.1. For this a comparison can be made between the rank-1 and rank-4 model where only the learning rate of the models is different (rank-4 model has a lower learning rate of 0.01).  
  Also, we can clearly see from the results that the SGD optimizer is not very useful as it almost does not learn anything about the data and tries to predict everything with a probability of 0.5. Hence, the Adam optimizer must be the one that should be selected. With the Adam optimizer the f1-score tends to be above 0.73 for the top 10 models. Also, the number of weak learners is not required to be high, we can even do good estimates even with 5 learners when compared to the models that use 25 and 50 weak learners. But, according to the results the best f1-score is obtained when we use 25 weak learners.  
  Here, we have not implemented the early stopping approach. For example, we can get slightly better accuracy than the final rank-1 model when we stop the model early at around 13 weak learners as can be seen in the accuracy and cost curves above (curves that are without SGD).   
  Also, we believe that since the learning task of NLP is hard the model is able to learn the best with Adam and a higher learning rate(usually adam is also able to learn quickly with lower learning rates and performs worse for higher learning rates but the hardness of the task utilizes the higher learning rate to provide good results) but not with SGD even with a high learning rate.
* Results on how test performance varies with choice of features:  
  We can clearly see from the table above that the dataset used is X\_train\_LA that gives the best models. We also see that the dataset X\_train\_SA is the second-best dataset.  
  But the datasets X\_train\_LCR and X\_train\_SCR do not give any results in the top 5 models (Again we have discussed the creation of these datasets and feature selection done to create them in the section above where we explain the feature selection in detail).  
  These results of feature selection make sense. This is because the model is AdaBoost and it has less capability to capture the structure of the sentence. Hence, if we give high dimensional data to it to learn from (as in LCR or SCR that is 500-dimensional), the model simply does not learn much as it has low capacity. On the other hand, when we give averaged embeddings(50-dimensional) then the model is able to learn faster and give better results as it has to deal with the data in a compact version that it can learn better.  
  Moreover, we see that the model performs better when we use long sentences as compared to short sentences. This also makes sense since long sentences tend to cover more words and hence feature selection from more words should in theory give better accuracy as compared to the model learnt with short sentences.  
  Hence, we can see that the test performance tends to be better when the model learns on long sentences (due to more context to learn from) and shorter dimensional representation of these sentences (due to the low model capacity for this task).
* Results on how test performance varies with choice of machine learning model:  
  The performance variation of the models was explained in the Machine Learning models section above where we explain the machine learning models trained. The reason we mentioned the selection of AdaBoost over simple linear regression was an improvement in accuracy of 1-1.5% on the test set.  
    
  Now, the rank-1 model’s classification report from the table above is as below:  
  A screenshot of a graph

  Description automatically generated  
  F1-Score of rank1-model: 0.7498  
    
  Now, the Logistic Regression model’s classification report with the same hyperparameters as the rank-1 model can be given as below:  
  A screenshot of a computer

  Description automatically generated  
  F1-Score of Logistic Regression model is: 0.7401  
    
  Hence, we can see from the above results that AdaBoost provides a little boost to the performance as compared to the Logistic Regression model although the Logistic Regression model is also very powerful. Hence, we use AdaBoost in our training pipeline as the primary model.

In addition, provide discussion of the trends in your results.

The trends in our results are as follows:

1. The models tend to learn better when longer sentences are provided.
2. The distinction between longer contexts from longer sentences and shorter contexts from shorter sentences is made clear by representing each document via a fixed length feature embedding for both the longer sentences and shorter sentences. This shows that longer sentences provide more information and also that the feature selection methods used are also able to capture these longer contexts for the model.
3. Secondly, AdaBoost and Logistic Regression models provide similar results with minor improvement provided by AdaBoost to Logistic Regression. On the other hand, these both perform better as compared to the standard XGBClassifier.
4. The AdaBoost model is able to provide good accuracy with a smaller number of weak learners as well. Hence, the number of weak learners less than or equal to 25 is ideal.
5. The learning rate is kept high to be 0.1 to reach the results faster in a smaller number of weak learners.
6. Also, the Adam optimizer outperforms the SGD optimizer by a large margin as can be seen from the classification reports and the accuracy curves in the results section above.
7. The models also do learn the structure of the data as we obtain general accuracy of >70% for the different hyperparameter choices which is better than the random accuracy of 50%.