

REPORT

MID-PROJECT REVIEW

Akshara Nair - MT22008 | Ayush Agarwal - MT22095 | Medha - MT22110
Mohit Gupta - MT22112 | Nidhi Verma - MT22044 | Pratik Chauhan - MT22118

Problem formulation:

With a massive amount of content being published on an individual's social media, there exists an emergent need to monitor if the posts being posted by an individual are indicative of the existence of early symptoms of depression. **An early and effective analysis and prediction through pre-learned representations of depression by visual and textual data forms the basis of the problem of the proposed research based project.** The aim of the proposed research based project is development and deployment of Machine Learning and Deep Learning models to analyze multimodal data to predict depression. For the evaluation of baseline models, we have scraped the data as images and text from platforms where user based content is generated such as Reddit (Text) and Shutterstock (Images) and evaluate preliminary results from the best suited models for the purpose. We further look forward to finding the best suited models for classification of an image or text based input as depressed or not depressed and combine the predictions of the specific modalities to provide efficient results on Social Media Posts of an individual.

Literature review:

Physical activities and geographic movement patterns were classified using built-in phone sensors, such as the acceleration and Global Positioning System (GPS) sensors, respectively. A portion of the characteristics were chosen. With an accuracy of 87.2%, the SVM classifier was used to differentiate between the three different severity levels of depression (absence, moderate, and extreme) [1]. Information gathered by mobile phone sensors, including utilization of applications, brightness, acceleration, rotation, and orientation. They were combined by the creator to create higher-level feature vectors. The subjects' stress levels could be predicted using the fusions of these feature vectors [2]. Researchers looked at how the facial cues of depressed and normal individuals changed in the same circumstance. (while displaying positive, neutral, and negative pictures). To measure the facial cue changes on the face, they used a person-specific active appearance model to detect 68-point landmarks. Statistical features are extracted from distances between feature points of the eyes, eyebrows, and corners of the mouth to feed the SVM classifier. The classifier achieved 78% test accuracy [3]. They identified 68-point landmarks using a person-specific active appearance model to quantify the facial cue changes on the face. To provide the SVM classifier with statistical features, the distances between the feature points

of the eyes, brows, and corners of the lips are measured. The classifier's test success score was 78% [3]. Due to the following factors, some scholars have recently attempted to combine various modalities: First, when the convergence of modalities is carried out, the input of each modality can be better understood. Second, each modality has benefits of its own. Consequently, a combination can produce a superior result. Third, the elements have compatibility traits. In order to identify psychomotor retardation, Williamson et al. [4] used feature sets drawn from facial movements and acoustic verbal signals. For dimensionality reduction, they used principal component analysis, and to categorize the combination of primary feature vectors, they used the Gaussian mixture model. The majority of the works covered in this section use ML-based techniques rather than deep learning techniques (Support Vector Machines, Gaussian Mixture Models, Random Forest, etc.). This could be due to a lack of training material availability. Deep learning requires more training data than ML, whereas ML-based methods can be taught on less data. Another explanation might be that supervised machine learning is more effective when there is a known connection between the inputs and labels. To increase the accuracy of the model, many features can be extracted and then assessed. **Depression Detection by Analyzing Social Media Posts of Users :** This study suggests a data-analytic strategy for identifying depression in any human. The data used in this suggested model is gathered from user posts on Twitter and Facebook, two well-known social media websites. The entries are vectorized using the SVM, and their positivity, negativity, or neutrality is ascertained using Naive Bayes. It takes the user's username and examines their social media posts to gauge their degree of susceptibility to depression. The machine learning model is taught to categorize the six ranges of depression criteria. (Considered Normal, Mild, Moderate, Borderline, Severe, and Extreme). The model's assessed accuracy is 74.00% [5].

References

- [1]: Masud MT et al (2020) Unobtrusive monitoring of behavior and movement patterns to detect clinical depression severity level via smartphone. J Biomed Inform 103:103371
- [2]: Fukazawa Y et al (2019) Predicting anxiety state using smartphone-based passive sensing. J Biomed Inform 93:103151
- [3]: Wang Q, Yang H, Yu Y (2018) Facial expression video analysis for depression detection in Chinese patients. J Vis Commun Image Represent 57:228–233
- [4]: Williamson JR, Quatieri TF, Helfer BS, Ciccarelli G, Mehta DD (2014) Vocal and facial biomarkers of depression based on motor incoordination and timing, 65–72
- [5]: N. A. Asad, M. A. Mahmud Pranto, S. Afreen and M. M. Islam, "Depression Detection by Analyzing Social Media Posts of User," 2019 IEEE International Conference on Signal Processing, Information,

Communication & Systems (SPICSCON), Dhaka, Bangladesh, 2019, pp. 13-17, doi: 10.1109/SPICSCON48833.2019.9065101.

METHODOLOGY AND RESULTS :-

1. Scraping Process

- Image Scraping:

We are scraping images from **shutterstock**. For that we are using selenium, and BeautifulSoup. We have created a method which takes 4 parameters, searchTerm, searchPage, image_type, directory_path to store the images.

When we get the 'lxml' object of the web page, then we find all the **** tags, having class name '**mui-117n00y-thumbnail**'. Then for each image, we are taking the src value and using the urllib library to retrieve the image and storing it in the given directory. Using this, we are scraping almost 2500 images containing both depression and not_depression images. Then after scraping images from the web, we need to preprocess all images. For preprocessing all the images, we apply:

- 1. Reduce the dimension of each image to (128, 128)**
- 2. Convert the image into grayscale.**
- 3. Remove shutterstock watermark.**

Then we split the images into TRAIN and TEST in the ratio of 80:20. The images data folder contains two folders named test and train, which further contains two folders one for each class i.e. depression and not_depression. The specific format is used to load the data efficiently from Image Data Generators for training the Deep Learning Models through Libraries.

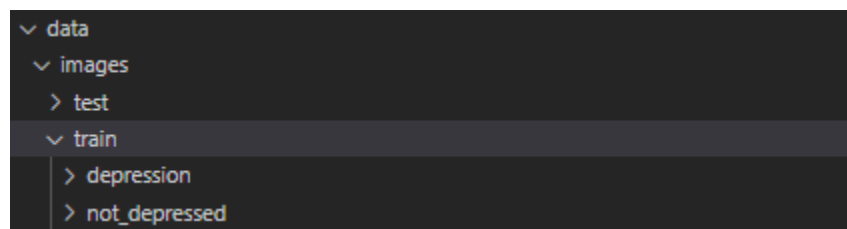
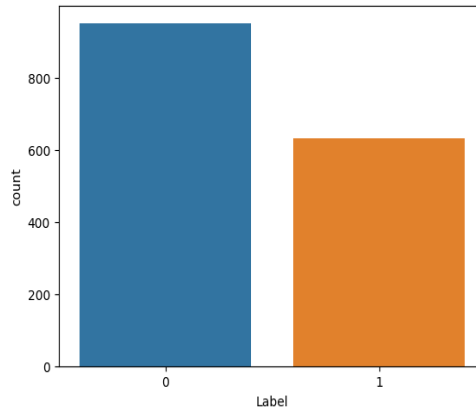


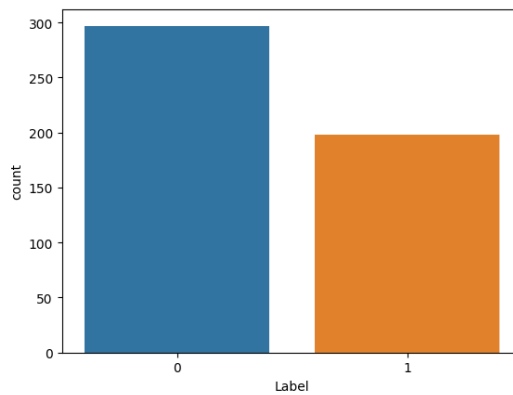
Image Based Classification

A. Data Distribution : The data distribution is balanced across the categories. The relative ratio of the difference amongst the number of samples in the dataset is followed across the data split in all of the three sets namely, the data being used for Training, Validation and Testing and hence ensures that the results are not biased due to imbalanced data distribution. The Training and Validation split is 80:20 from all the images currently in the Training Dataset.

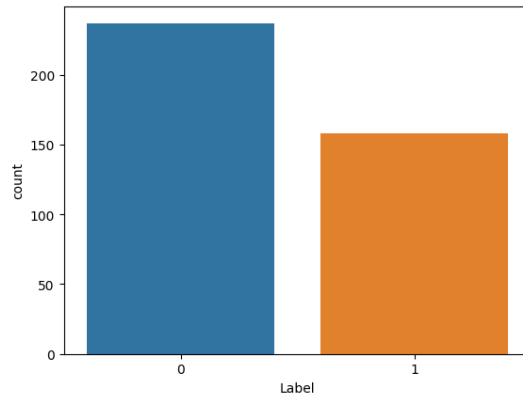
1. For Training Data



2. For Test Data



3. For Validation Data

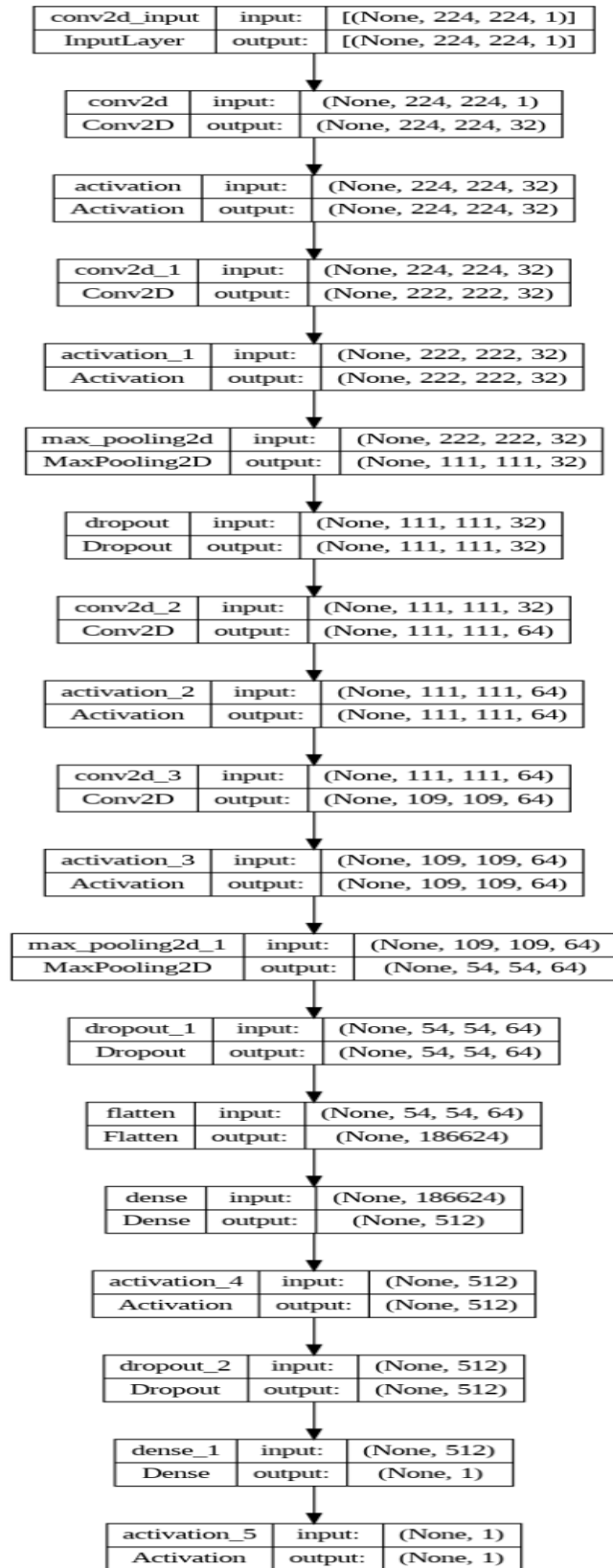


B. Data Visualization : Visualizing the first five images of each batch (BATCH - SIZE = 64) - The first five images of each batch with the corresponding labels are as depicted below.

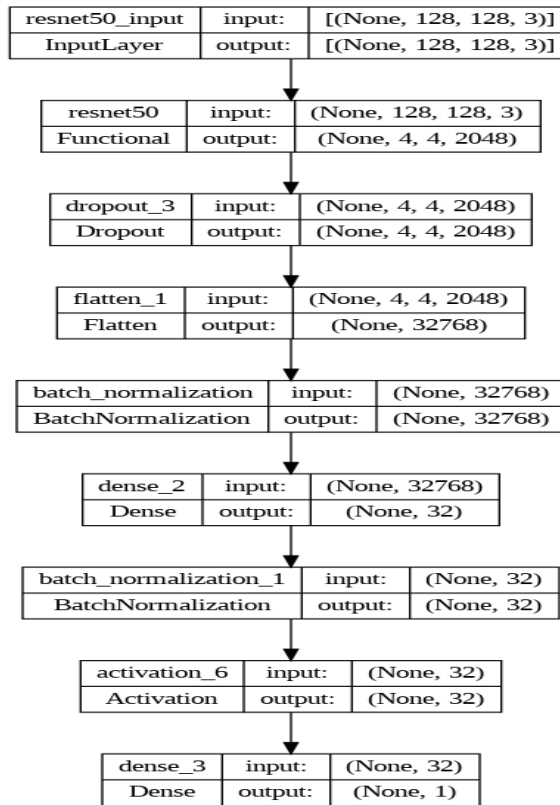


C. Model Architectures for Image Classification :

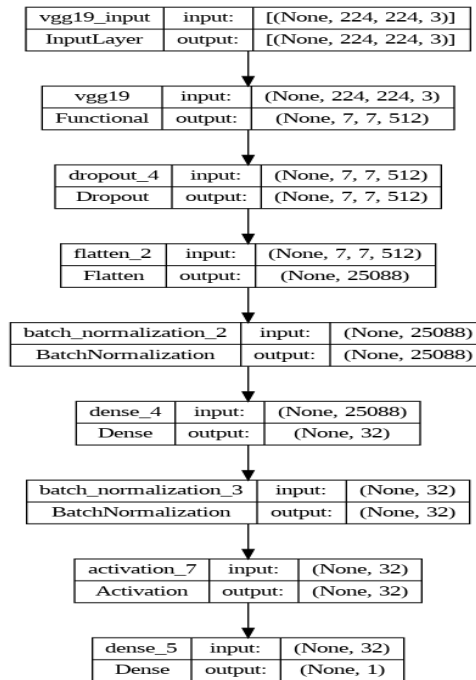
1. Deep CNN



2. ResNet50(Pre-Trained and Fine Tuned)



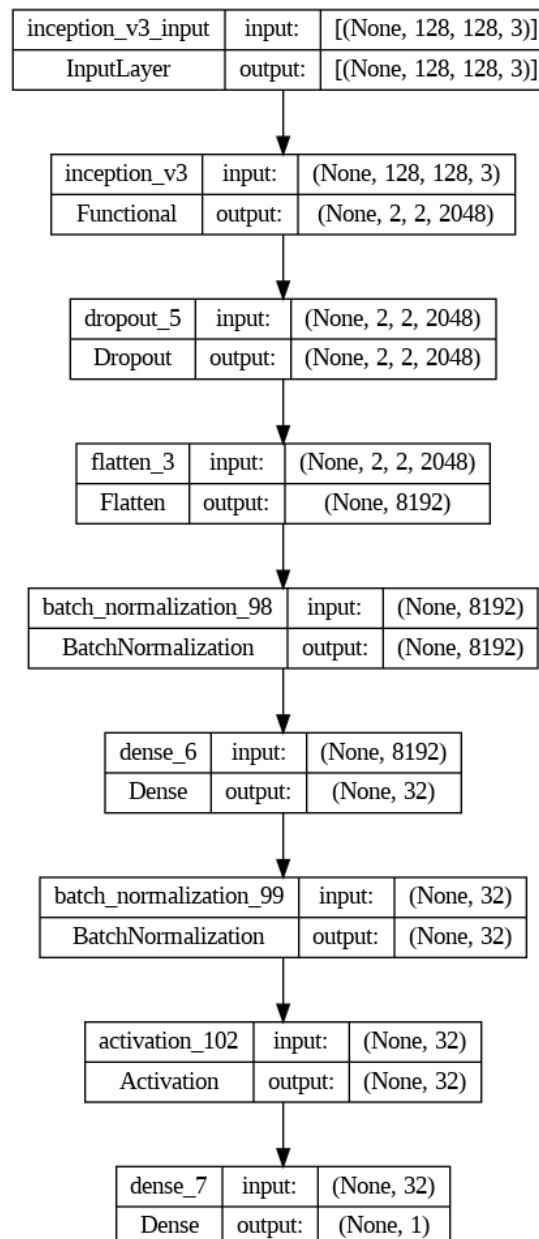
3. VGG19(Pre-Trained and Fine Tuned)



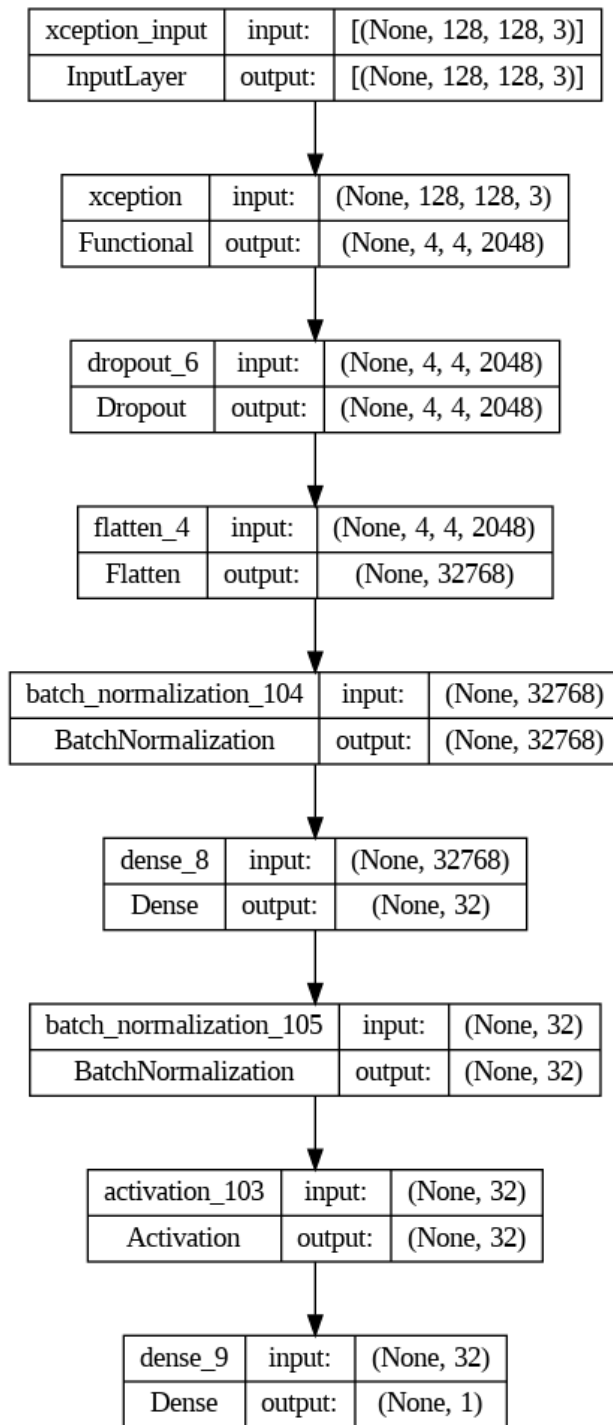
MID-PROJECT REVIEW

The following improvements are proposed to find out the best Deep Learning Model for Visual or Image based input. As a part of the Mid-Project Review, the initial imbalanced dataset is evaluated for various distinct model architectures for the task of Image Classification. The models introduced and checked for various distinct parameters, to improve the performance of the overall classifiers is as follows: -

4. InceptionV3 (Pre-Trained and Fine Tuned)

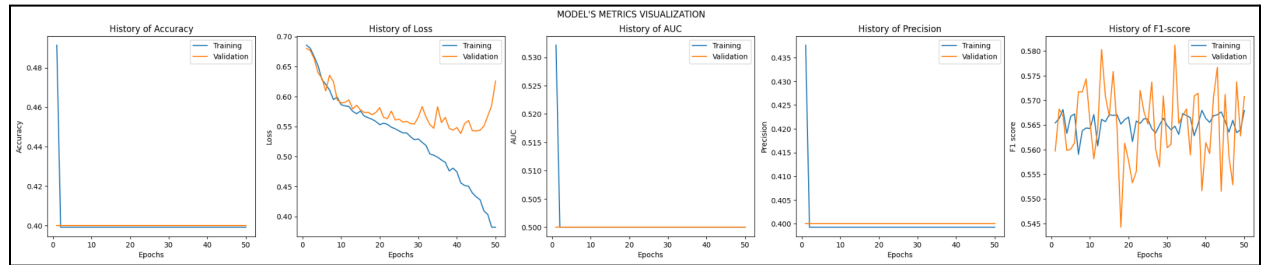


5. Exception

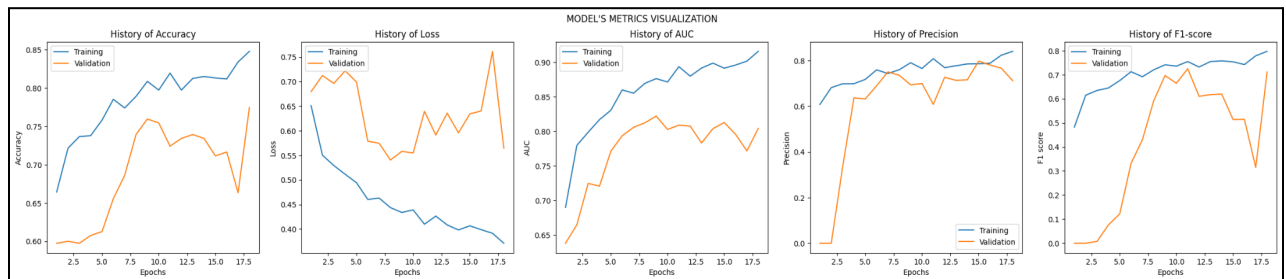


D. Results

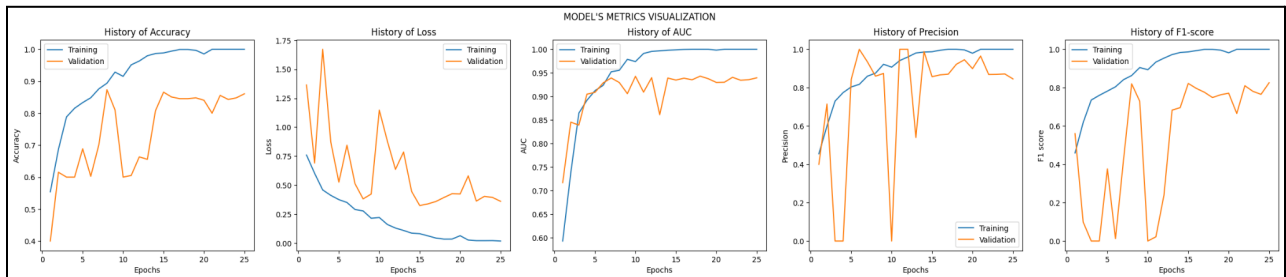
1. Deep CNN: Results



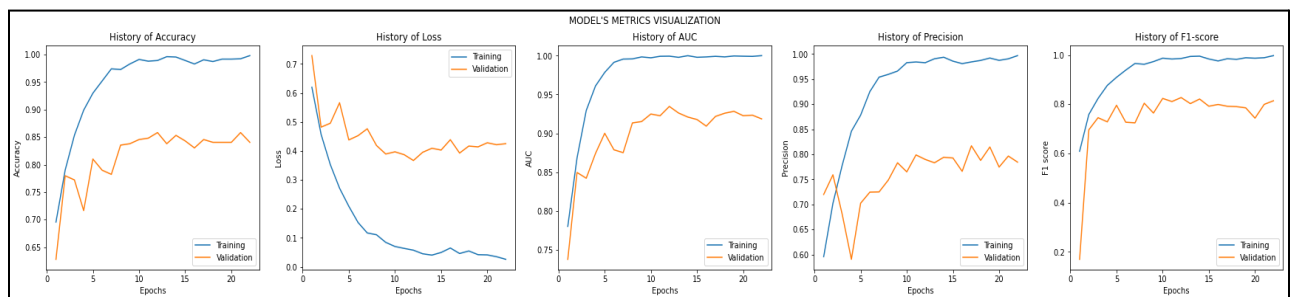
2. Model Architecture - ResNet50(Pre-Trained on ImageNet): Results



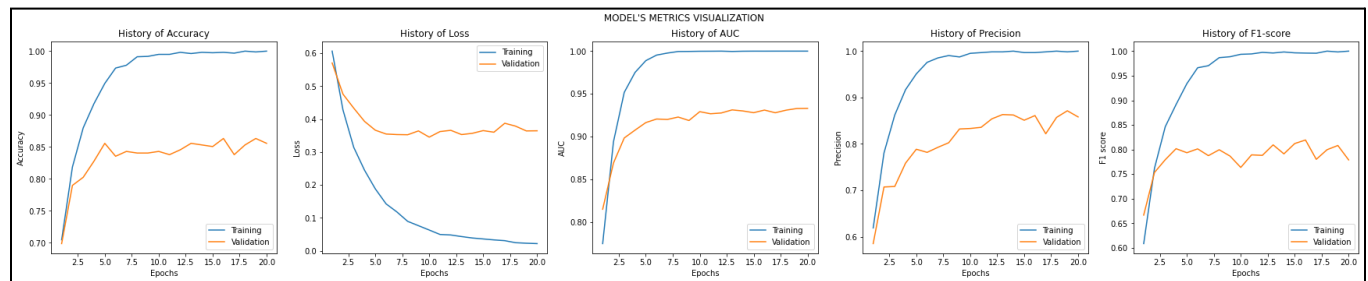
3. Model Architecture - VGG19: Results



4. Model Architecture - InceptionV3: Results



5. Model Architecture - Exception: Results



The Training and Validation Accuracies for the specified models, being implemented are as below :-

1. **Deep CNN** - Training Accuracy - 39% and Validation Accuracy - 40%.
2. **ResNet50** - Training Accuracy - 84.78% and Validation Accuracy - 77.47%.
3. **VGG19** - Training Accuracy - 100% and Validation Accuracy - 86.08%.
4. **InceptionV3** - Training Accuracy - 99% and Validation Accuracy - 89%
5. **Xception** - Training Accuracy -99.56% and Validation Accuracy - 83 %

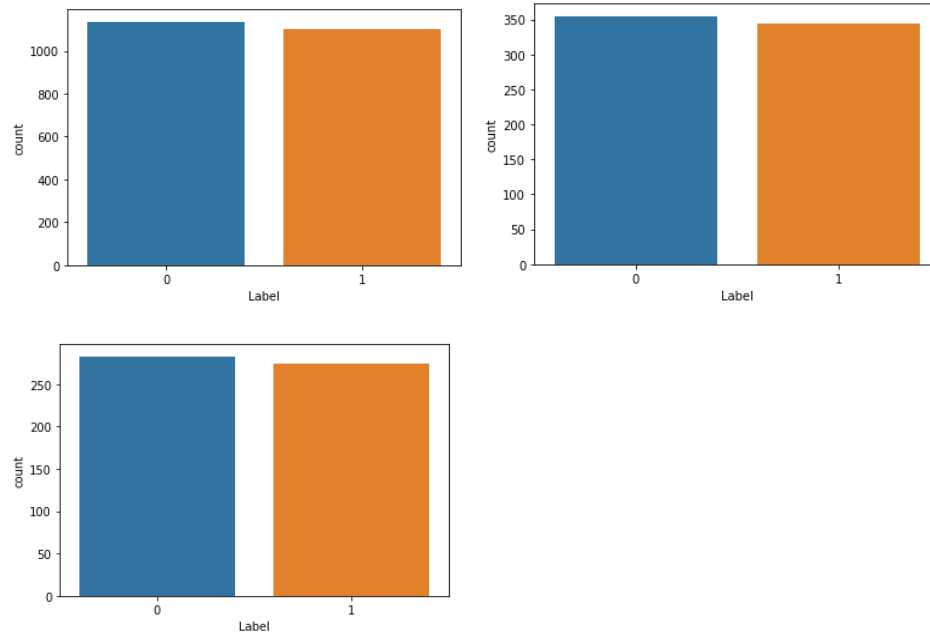
As a part of the Mid-Project Analysis, the evaluation metrics as Loss, Accuracy, Precision, Recall and F-1 Score are being evaluated on the Test Set and the metrics reported through various experimentations being carried out are reported as follows: -

1. Xception Architecture - Loss: 0.4644 - Accuracy: 0.8505 - Precision: 0.8523 - Recall: 0.7576 - AUC: 0.9201 - F1-Score: 0.7909
2. **Inception Architecture - Loss: 0.4024 - Accuracy: 0.8586 - Precision: 0.8019 - Recall: 0.8586 - AUC: 0.9257 - F1-Score: 0.8162**
3. ResNet Architecture - Loss: 0.6255 - Accuracy: 0.7253 - Precision: 0.7719 , Recall: 0.4444 - AUC: 0.7937 - F1-Score: 0.5467
4. VGG16 Architecture -
5. Deep Convolutional Neural Network - Loss: 0.5337 - Accuracy: 0.4000 - Precision: 0.4000 - Recall: 1.0000 - AUC: 0.5000 F1- Score : 0.5650

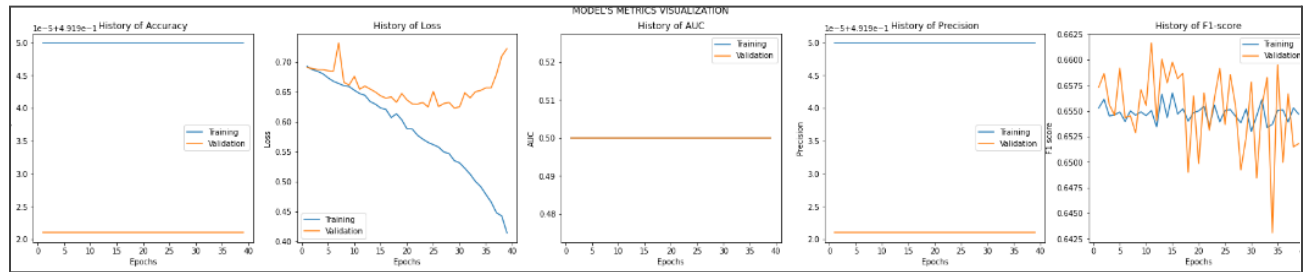
Conclusion - As observed from the test set, the best behavior of the predictions under the test set is highly accurate when Inception Architecture is deployed to serve the predictions.

The performance of the model, further deprecated by scraping the data from the web as the images were irrelevant for the keyword. At the training time, though the data was sufficient and balanced, it was not the relevant data to be explored or used to solve the specified task of predicting depression.

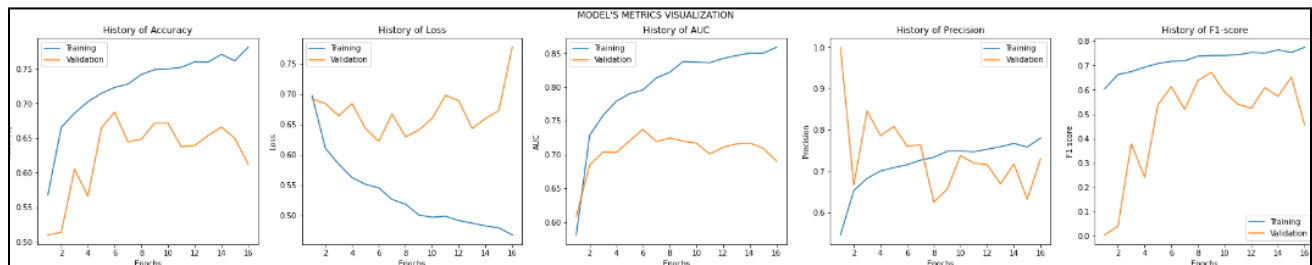
The updated Dataset is balanced, which can be viewed as follows for the Train, Validation and Test Datasets.



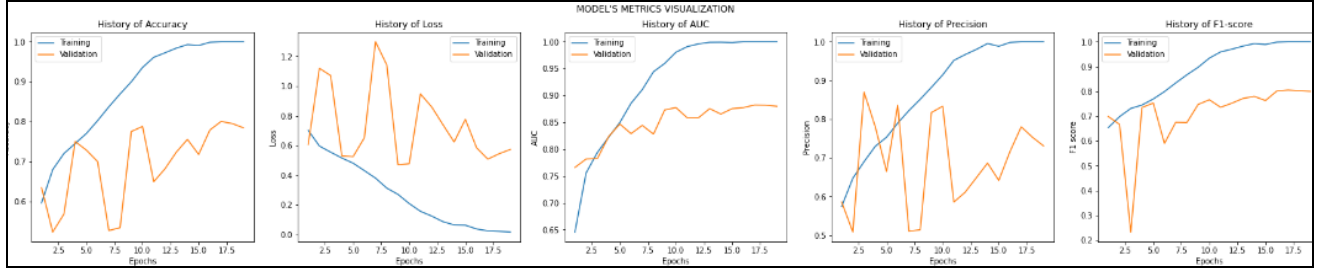
1. Deep Convolutional Neural Network



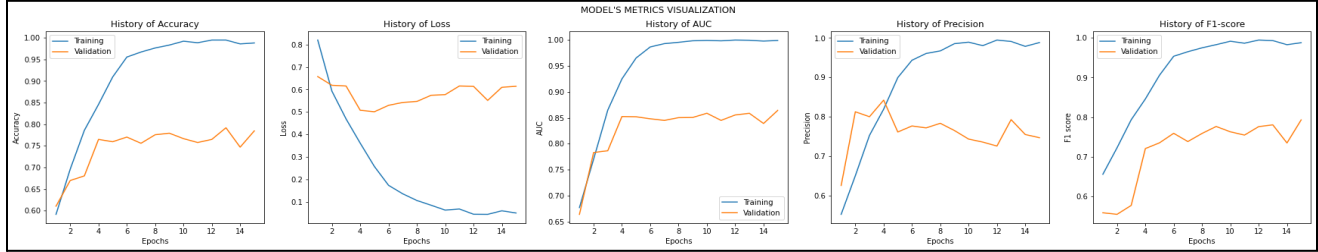
2. ResNet Architecture



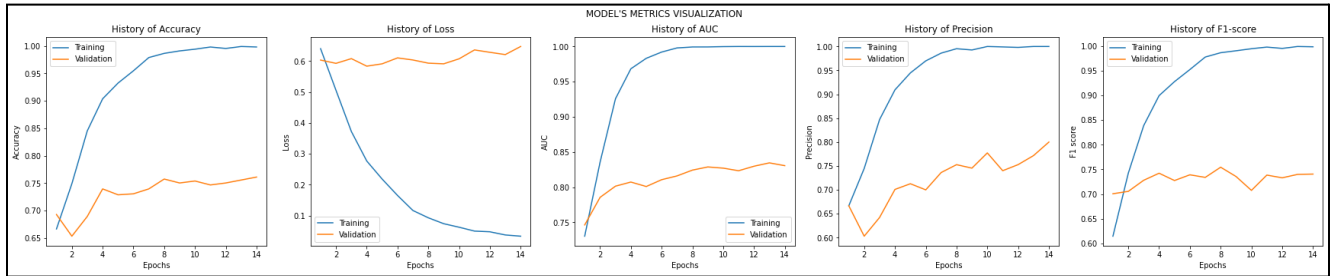
3. VGG16 Architecture



4. InceptionV3 Architecture



5. Expection Architecture



GRAD-CAM Visualisations -

Grad-CAM (Gradient-weighted Class Activation Mapping) is a popular technique in computer vision used for visualizing and understanding the regions of an image that a convolutional neural network (CNN) focuses on when making a particular prediction. Specifically, Grad-CAM produces a heatmap that highlights the most important regions of an image for a given classification decision made by a CNN.

Grad-CAM works by computing the gradient of the score for a target class (i.e., the class with the highest probability) with respect to the feature maps in the final convolutional layer of a CNN. These gradients are then used to weight the feature maps, effectively measuring how important each feature map is for the target class. Finally, the weighted feature maps are averaged to produce the heatmap, which highlights the most relevant regions of the image for the target class.

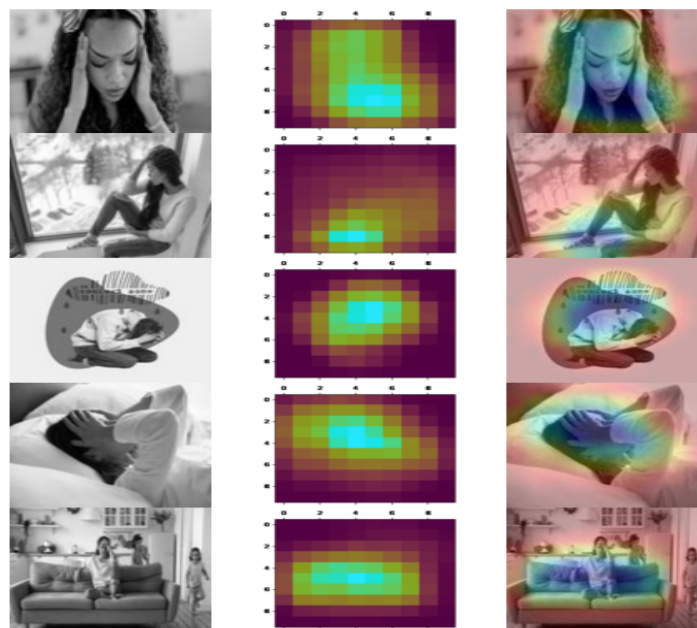
GRAD-CAM supports a variety of tasks in computer vision, including image classification, object detection, and image segmentation. By visualizing the regions of an image that a CNN focuses on, GRAD-CAM provides insights into how a model makes its predictions and helps identify potential areas for improvement. It is being used for interpretability and explainability of deep learning models.

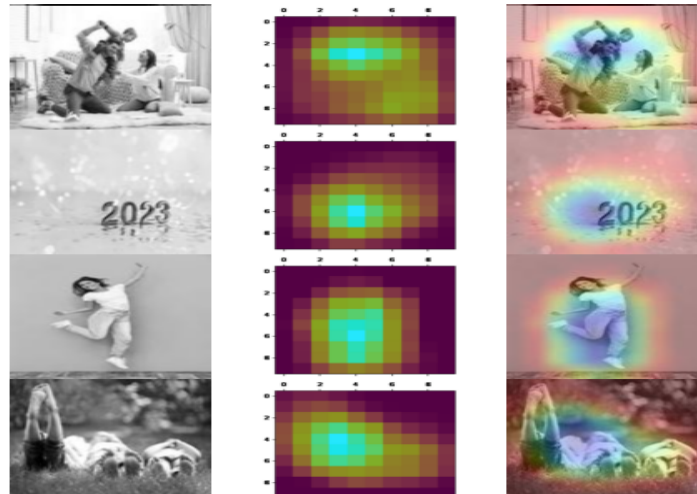
GRAD-CAM Visualization for Depressed and Not-Depressed Images.

The portion highlighted in the images for depressed images usually learns to infer the results from certain specific portions of the image, commonly observed as the facial expressions of the individual as well as the posture of a person identified. In general the pre-trained model is perfectly able to identify the distinction between facial features of a happy or a depressed person as it is easy to learn a relevant representation for the same on the basis of a general observation as the high points of the face as eyes, end of lips, when high and raised, infer a happy expression and is able to indicate the model to learn the specificity.

The posture of an individual as bent legs or arms are indicative of a depressed person and hence are being focused on by the CNN and pre-trained models to learn relevant representations. The GRAD-CAM represents the maximum activations learnt by the CNN, which ultimately lead to the classification of images to a certain category.

In summary, GRAD-CAM is a technique used to visualize the most relevant regions of an input image for a particular classification task, based on the activation of the final convolutional layer of a CNN.





Text Based Classification

A. Text Scraping: - For scraping text data, we are using the reddit api '**praw**'. What we are doing is, we first find all the top posts having the subreddit 'depression anxiety', then fetching the title, and the body of the post and storing it in a pandas dataframe. Then after we are saving the dataframe in the .csv file. We have extracted around 12000 text posts containing both depressed and not_depressed posts.

B. Text Preprocessing-

These preprocessing techniques are:

1. **Whitespace removal:** The `re.sub()` method is used to remove multiple whitespace characters with a single space character.
2. **URL removal:** The `re.sub()` method is used to remove URLs from the input text data.
3. **User mention removal:** The `re.sub()` method is used to remove user mentions, which start with the "@" character.
4. **Number removal:** The `re.sub()` method is used to remove all digits from the input text data.
5. **Emoji removal:** The `re.sub()` method is used to remove all emojis from the input text data. **Lowercasing and tokenization:** The input text data is converted to lowercase and tokenized using the `word_tokenize()` method from the NLTK library.

6. **Stopword removal:** Stop words, which are common words such as "the" and "and" that do not provide much meaning, are removed using list comprehension and the stopwords corpus from the NLTK library.
7. **Punctuation removal:** Punctuation marks are removed using list comprehension and the `isalnum()` method.
8. **Spelling checking:** The `Speller()` method from the `enchant` library is used to check the spelling of each word in the input text data.
9. **Lemmatization:** The `WordNetLemmatizer()` method from the NLTK library is used to lemmatize each word in the input text data, reducing them to their base form.

Overall, these preprocessing techniques help to clean and normalize the input text data, making it easier to analyze and extract meaning from it.

C. **Text Models:** - Firstly, we created the tokenizer which uniquely labeled our top 1000 words and rest as OOV words. Now, we pass this to our model, which consists of three layers:

1. **Embeddings Layer:** Embedding class used to generate 32-dimensional embeddings of every input sample sentence present in the dataset.
2. **Bi-LSTM Layer:** An RNN type, modeling the sequential dependencies between words and phrases in both directions of the sequence.
3. **Dense Layer.**

```

08 2 model.summary()

Model: "sequential_4"

Layer (type)                 Output Shape              Param #
=====
embedding_4 (Embedding)      (None, 764, 32)          724896

bidirectional_4 (Bidirectio  (None, 64)                16640
nal)

dense_4 (Dense)              (None, 1)                 65
=====
Total params: 741,601
Trainable params: 741,601
Non-trainable params: 0

```

D. Results on few samples :


```
9 #bracket represent probability of class 1(depressed). if prob > 0.5, its classified as depressed, else not.
1/1 [=====] - 1s 665ms/step
thing puts smile face brother hes best -> not depressed (0.002152541186660528)
anxiety literally ga kill someone anyone help pls mens sleep drinking water help -> depressed (0.9907008409500122)
I am so depressed kill -> depressed (0.9696369767189026)
life is beautiful -> not depressed (0.01041035819798708)
lets play in the sun -> not depressed (0.033363226801157)
i am very much scared of medha -> depressed (0.7482262849807739)
```

The Training and Test Accuracies for the specified model being implemented are as below : -

Training Accuracy - 0.9914 and Test Accuracy - 0.9291

Mid Project Review

Data Collection:

We scrapped more data with the prompt ‘suicidal’ and added them as samples for ‘depressed’ class. This was done to create a balanced dataset, since scrapping done using the earlier tags had resulted in imbalance.

Adding ‘suicidal’ - tagged sentences as depressed seems reasonable since suicide is an extremely negative sentiment as suicidal thoughts hint of a depressive mind. However, their could be non-suicidal sentences yet hinting at depression, also observed in our dataset.

Ex: ‘I will never be able to live a prosperous life’.

Hence, we did not scrape and add non-suicidal samples as non-depressed.

Experiments:

1. Non-Contextual Embeddings + SVM:

Non-contextualized embeddings are a type of word embedding that represents each word as a fixed vector, regardless of its context. These embeddings are generated by training a model on a large corpus of text data and using statistical techniques to encode each word as a dense, low-dimensional vector.

Non-contextualized embeddings such as Word2Vec and GloVe are widely used in natural language processing tasks, such as language modeling, sentiment analysis, and text classification.

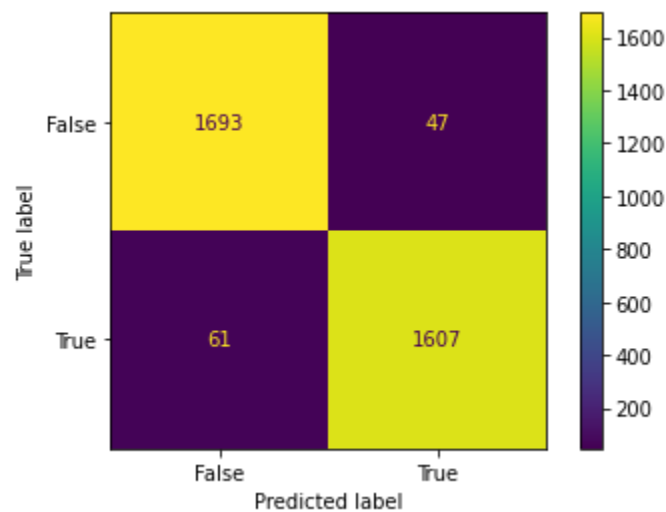
FastText is a type of non-contextualized word embedding algorithm developed by Facebook's AI research team. It is based on the idea that words are composed of character n-grams, or sequences of one or more characters, and that these n-grams can provide useful information about the meaning of a word. FastText generates embeddings by training a neural network on a large corpus of text data, using these character n-grams as features.

One advantage of non-contextualized embeddings is that they can be generated quickly and efficiently, as they do not require the context of the word to be taken into account. This makes them well-suited for tasks that require fast and scalable processing of large amounts of text data.

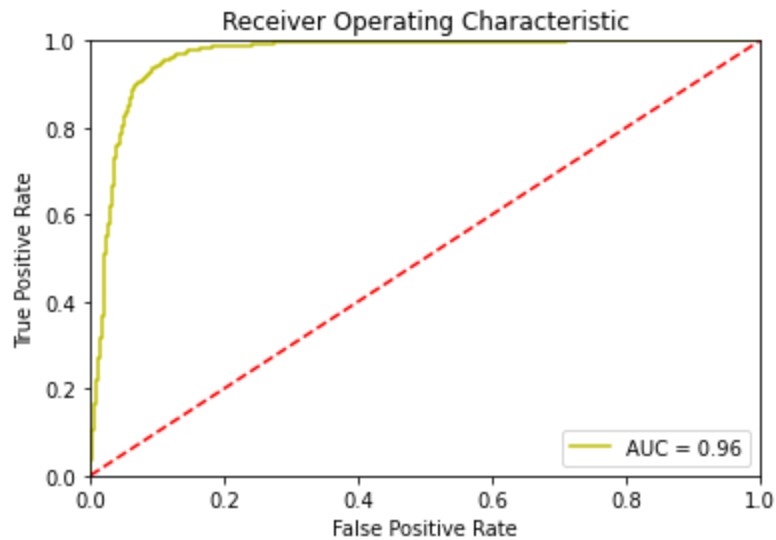
However, they may not capture the full range of meaning of a word, as the same word can have different meanings in different contexts. This limitation can be addressed by using contextualized embeddings, which take into account the context in which a word appears.

In our project, FastText embeddings are trained using Support Vector classifier and the evaluation results are as follows:-

1. Accuracy Score for test_set : **0.9683098591549296**



- 2.



3.

4. Precision Score for the data **0.971584038694075**

5. F1 Score for the data **0.9590288315629741**

2. Contextual Embeddings + SVM:

Contextualized embeddings are a type of word embedding that captures the meaning of a word in its surrounding context. Unlike traditional word embeddings, which represent each word as a fixed vector regardless of its context, contextualized embeddings are dynamically generated based on the context in which a word appears.

This allows them to better capture the nuances and subtleties of language and can be particularly useful in tasks that require a deeper understanding of language, such as natural language processing and machine translation. bert-base-nli-mean-tokens is a pre-trained contextualized embedding model based on the

Bidirectional Encoder Representations from Transformers (BERT) architecture. This model is trained on a large corpus of text data, using a masked language modeling and next sentence prediction objective. The bert-base-nli-mean-tokens model generates contextualized embeddings by encoding each word in a sentence as a vector, and then taking the mean of all these vectors to obtain a sentence-level embedding.

This makes it useful for tasks that require a single representation of an entire sentence, such as sentence classification, semantic similarity, and question answering.

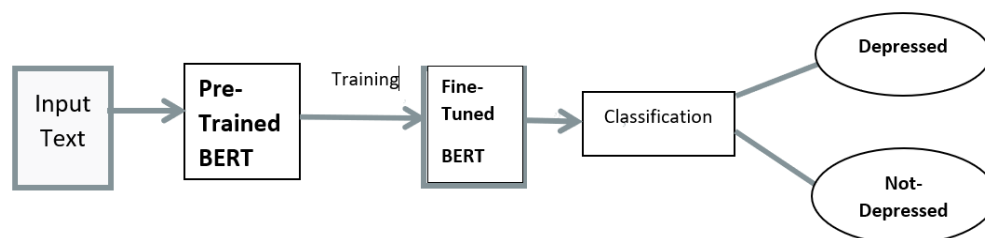
One advantage of contextualized embeddings is that they can capture the full range of meaning of a word, taking into account its context and allowing for a more nuanced understanding of language. The bert-base-nli-mean-tokens model, in particular, has shown to perform well on a range of natural language processing tasks, achieving state-of-the-art results on benchmark datasets.

However, contextualized embeddings can be computationally expensive to generate, and may not be as interpretable as non-contextualized embeddings.

Additionally, since contextualized embeddings are generated based on the context in which a word appears, they may not be suitable for tasks that require a fixed representation of a word, such as information retrieval.

3. Contextual BERT Embeddings +Fine-Tuning:

BERT (Bidirectional Encoder Representations from Transformers): In our project we are using a pre-trained BERT model from Hugging Face to classify our text whether it belong from depressed category or not-depressed category.



One of the main features of BERT is its ability to understand the context of words in a sentence.

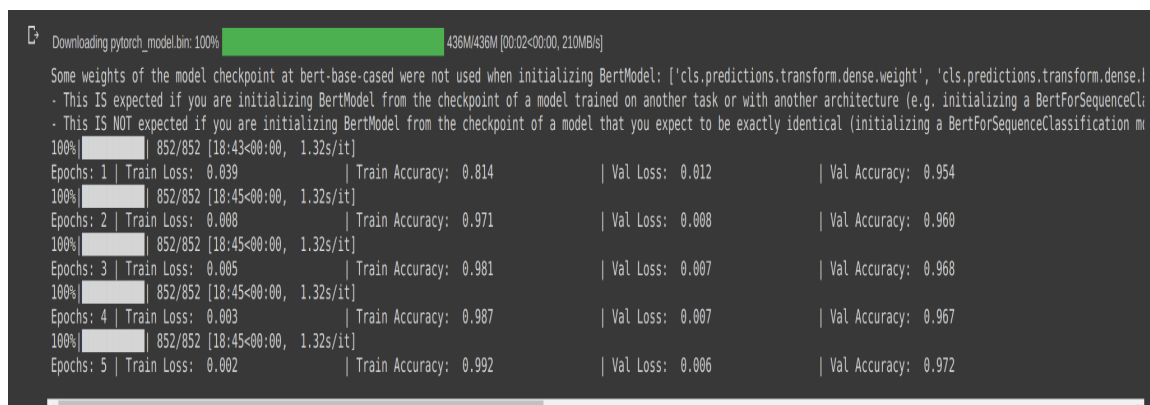
At a high level, BERT is composed of multiple transformer blocks, which consist of two sub-components: the multi-head self-attention mechanism and the feedforward neural network. The self-attention mechanism allows the model to attend to different parts of the input sequence while the feedforward neural network applies non-linear transformations to the attended features.

The input to BERT is a sequence of tokens, which are first converted into dense vector representations using an embedding layer. These embeddings are then processed by multiple transformer blocks in a specific order. During training, BERT learns to generate contextualized embeddings by taking into account the surrounding tokens in the input sequence.

Fine-tuning BERT: The process of fine-tuning BERT involves several steps. First, the pre-trained BERT model is loaded, and the output layer is replaced with a new layer that is specific to the task being performed.

The fine-tuning process involves feeding the pre-trained BERT model with the labeled sentences from the dataset and training it to predict the correct label for each sentence. During this process, the model's weights are updated to minimize the difference between its predicted labels and the true labels from the dataset.

Once the model has been fine-tuned, it can be used to classify new, unseen sentences into their respective categories.



```

C:\> Downloading pytorch_model.bin: 100% 436M/436M [00:02<00:00, 210MB/s]

Some weights of the model checkpoint at bert-base-cased were not used when initializing BertModel: ['cls.predictions.transform.dense.weight', 'cls.predictions.transform.dense.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model)
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model)

100%|██████████| 852/852 [18:43<00:00, 1.32s/it]
Epochs: 1 | Train Loss: 0.039 | Train Accuracy: 0.814 | Val Loss: 0.012 | Val Accuracy: 0.954
100%|██████████| 852/852 [18:45<00:00, 1.32s/it]
Epochs: 2 | Train Loss: 0.008 | Train Accuracy: 0.971 | Val Loss: 0.008 | Val Accuracy: 0.960
100%|██████████| 852/852 [18:45<00:00, 1.32s/it]
Epochs: 3 | Train Loss: 0.005 | Train Accuracy: 0.981 | Val Loss: 0.007 | Val Accuracy: 0.968
100%|██████████| 852/852 [18:45<00:00, 1.32s/it]
Epochs: 4 | Train Loss: 0.003 | Train Accuracy: 0.987 | Val Loss: 0.007 | Val Accuracy: 0.967
100%|██████████| 852/852 [18:45<00:00, 1.32s/it]
Epochs: 5 | Train Loss: 0.002 | Train Accuracy: 0.992 | Val Loss: 0.006 | Val Accuracy: 0.972
  
```

Observation : As we see after training the model , we get validation Accuracy of 97.2 %, Validation loss = 0.006.

Source: [Text Classification with BERT in PyTorch | by Ruben Winastwan | Towards Data Science](#)

Conclusion:

Finetuning the large language model allows the model to adapt to specific characteristics of the data and the task and therefore results in better performance than other two types of pre-trained embeddings.

One possible explanation for having such high accuracy scores is because LLM's like BERT are trained on very large datasets thereby learning very rich semantics about the language. Moreover, the finetuning makes it more tuned to our subtask. Also, using 'depression', 'suicide' for one class and 'happy', 'joy' etc

for the other non-depressive class are highly polarised aspects ; thereby the model must be learning very different vectors in the feature space, accurately.