2021

TERM PROJECT

TEAM

ABDULLAH KHAN | YUVRAJ RAGHUVANSHI | ASHISH GUPTA

INDIAN INSTITUTE OF TECHNOLOGY PALAKKAD | Deep Learning: CS 5007

INTRODUCTION

This is the report for Term Project for Deep Learning. The project was developed in a semester long fashion. There are three tasks which constitute this project. The first task is multi-label image classification. The second task is multi-modal image classification while the third task is Image Captioning. For all the tasks, Pascal50s dataset was used to train and test. This report contains the detailed overview of implementation and justification for design approach used to do different tasks.

CONTRIBUTIONS

The project was done in a team of three – Ashish Kumar Gupta, Abdullah Khan and Yuvraj Raghuvanshi under the well guided mentorship of Ms. Shikha Mallick and supervision of Dr. Mrinal Kanti Das. The design aspects of each problem were discussed but the final code implementation was done individually by dividing the tasks among ourselves. My work included Task-3, Abdullah’s work was on Task-1 while Yuvraj did the Task-2.

CONTENTS

Task-1

Problem Statement

The first task is to classify the images into 20 classes(person, bird, cat, cow, dog, horse, sheep, aeroplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa, tv/monitor). As the dataset is for image description tasks, you won't find the image labels off-the-shelf, but need to scrape it using the 50 sentence descriptions associated with each image. There might be cases where the exact class names are not present in the description. Think of an approach to overcome this issue. Also, some images may contain more than two objects belonging to different classes. So, basically treat this as a multi-label classification problem. Next, using the images and the scraped labels, perform simple image classification using deep convolutional neural networks.

Multi-label Classification

The task of image classification isn't new or unusual, however, the two main pillars of intrigue which make this task so interesting are the absence of explicit labels for the input dataset and the inherent distinction between multi-class classification and multi-label classification. The difference between multi-label classification and multi-class classification is the cardinality of the output of the classification task. In multi-label classification, as the name suggests, an input image can have multiple correct labels due to the presence of various objects in the image. The labels of an input are not mutually exclusive of each other as in case of multi-class classification.

Label Extraction

The next challenge was to find such labels for each of the input images. Since the dataset was originally used for the task of image description, we had to scrape for labels from the image descriptions provided. These descriptions were obviously human written and we as humans usually communicate our thoughts using quite a varied vocabulary. However, this problem was easily accounted for by also scraping for the synonyms of the 20 labeled classes. The main challenge, however, remains untackled and that is to scrape credible labels from these descriptions. Since the dataset was originally used for image description, naturally the descriptions captured a complete scene rather than the objects that we so desire to classify. Without semantically analysing these descriptions, we cannot even tell what the main focus of them is, let alone accurately extract our objects of interest. This leads to a situation where we might get a label corresponding to an object which isn't even present in the input or a label being absent due to it having less significance towards the complete scene, which these descriptions captured. Such an error was rather reduced by using 50 sentences, lowering the probability of the absence case from happening. Since the main focus of the task was image classification, we stopped at this point satisfied by the results we were able to achieve.

Evaluation Metrics

Evaluation metrics play an important role of determining the performance of our model, however, we simply cannot use accuracy as our evaluation metric since in case of multi-label classification, there may be predictions which are neither completely wrong nor right. A prediction containing a subset of the actual classes should be considered better than a prediction that contains none of them, i.e., predicting two of the three labels correctly is better than predicting no labels at all.

Thus, for this reason we choose to use Area under ROC curve, Recall and Precision. If T denotes the true set of labels for a given sample, and P the predicted set of labels, then the following metrics can be defined on that sample as shown below:

- Precision $= |T \cap P| / |P|$
- Recall $= |T \cap P| / |T|$

ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. However in case of multi-label classification, in order to measure a multi-class classifier we have to average out the classes somehow. There are two different methods of doing this called micro-averaging and macro-averaging. In micro-averaging all TPs, TNs, FPs and FNs for each class are summed up and then the average is taken. In macro-averaging we just take the average of the precision and recall of the system on different sets.

Training Procedure

The dataset was divided into two parts, train and test for the final model evaluation in the ratio of 80 : 20. For hyper-parameter tuning, we used a cross validation with 5-fold split randomly. Using a stratified strategy isn't that simple in case of multi-label classification. The parameters we optimised by hyper-parameter testing were: Learning rate, size of dense layer at the end of the classifier, batch_size, dropout percentage. The final values of these parameters are summarized in the following table:

Parameter	Optimal Value
Learning Rate	0.001
FNN size	128
Batch size	32
dropout	0.75

Table 1

For this task we performed a comparative analysis of the following two models; the first one being a custom CNN model with a general architecture following pattern:

$$((\text{Conv2d} + \text{BatchNorm})*\text{nConvs} + \text{Pooling})$$

Following is the summary of the model we developed:

```
_________________________________________________________________

Layer (type)              Output Shape            Param #

=================================================================

Conv_0 (Conv2D)            (None, 250, 250, 64)     1792

_________________________________________________________________

BN_0 (BatchNormalization)    (None, 250, 250, 64)     256

_________________________________________________________________

Conv_1 (Conv2D)            (None, 250, 250, 64)     36928

_________________________________________________________________

BN_1 (BatchNormalization)    (None, 250, 250, 64)     256

_________________________________________________________________

MP_0 (MaxPooling2D)         (None, 125, 125, 64)     0

_________________________________________________________________

Conv_2 (Conv2D)            (None, 125, 125, 64)     36928

_________________________________________________________________

BN_2 (BatchNormalization)    (None, 125, 125, 64)     256

_________________________________________________________________

Conv_3 (Conv2D)            (None, 125, 125, 64)     36928

_________________________________________________________________

BN_3 (BatchNormalization)    (None, 125, 125, 64)     256

_________________________________________________________________

flatten (Flatten)          (None, 1000000)         0

_________________________________________________________________

dropout (Dropout)          (None, 1000000)         0

_________________________________________________________________

dense (Dense)              (None, 128)            128000128

_________________________________________________________________

dense_1 (Dense)            (None, 20)             2580

=================================================================

Total params: 128,116,308

Trainable params: 128,115,796
```

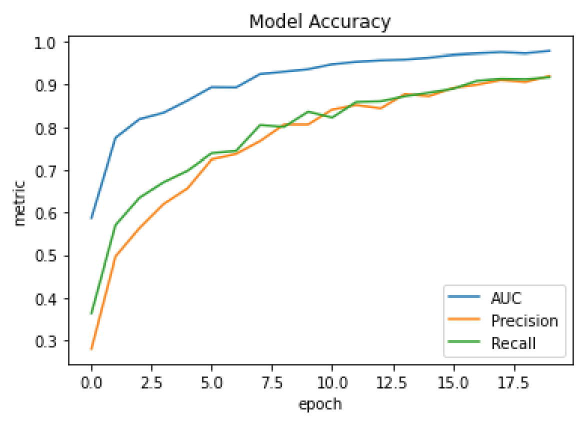
```
Non-trainable params: 512

_________________________________________________________________
```

However, we soon realised that the data we have is far insufficient to train a model from scratch which could not only recognize every object, but also multiple objects in the same image. Thus we then settled on a pretrained model. We used **vgg16** as our model of choice. Following is the summary of the pretrained model. We have not shown the standard vgg configuration due to space constraints.

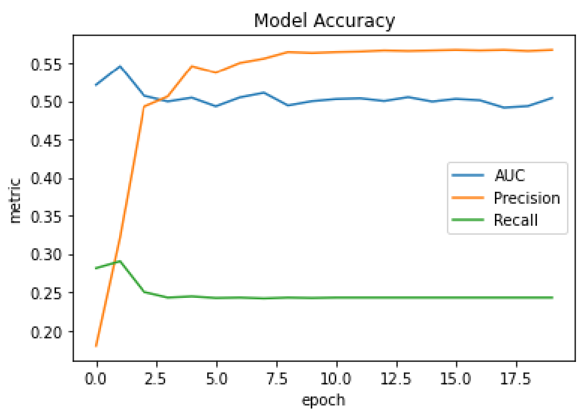
```
_________________________________________________________________

Layer (type)              Output Shape           Param #

=================================================================

Vgg16 (PreTrain)           (None, 125, 125, 64)    14714688

_________________________________________________________________

flatten (Flatten)          (None, 25088)          0

_________________________________________________________________

dropout_1 (Dropout)        (None, 25088)          0

_________________________________________________________________

dense_2 (Dense)            (None, 128)            3211392

_________________________________________________________________

dense_3 (Dense)            (None, 20)             2580

=================================================================

Total params: 17,928,660

Trainable params: 3,213,972

Non-trainable params: 14,714,688

_________________________________________________________________
```

Results

Following graphs show the training history of both the models for the finalized hyper parameters mentioned before. We can clearly see that the Pretrained model vastly outperforms the custom model as expected.



Pretrained VGG16



Custom CNN

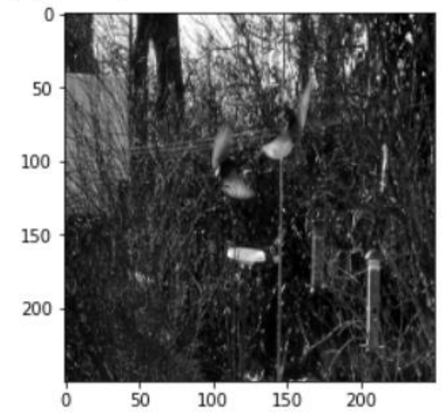
These experiments were repeated 5 times and the final metrics achieved by both the models is summarized below

Model	AUC ROC	Precision	Recall
VGG16	98.69±0.01	94.03±0.02	96.21±0.02
Custom CNN	50.70 ±0.01	56.25±0.01	24.29±0.03

Output

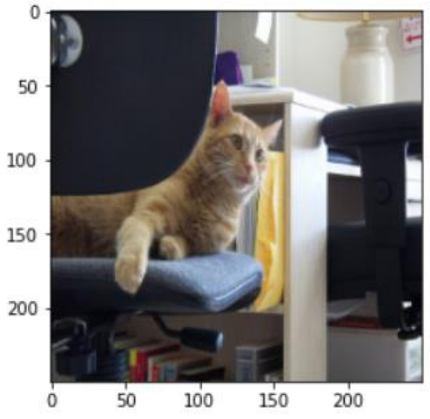
Following are the outcomes of the model on the test set:

```
['plant']
```



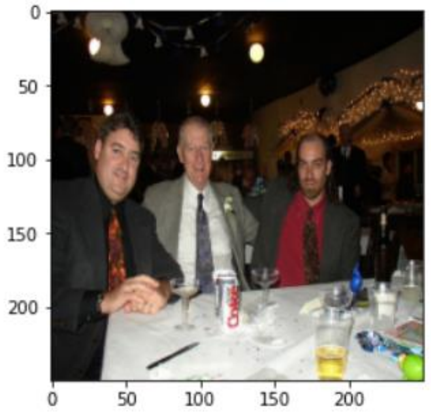
Here the input image is of some sort of forest/ shrubs.Even though there are some obstructions, our model was able to predict that there were plants in this image and no other recognizable object which as we can see is true.

```
['cat']
```



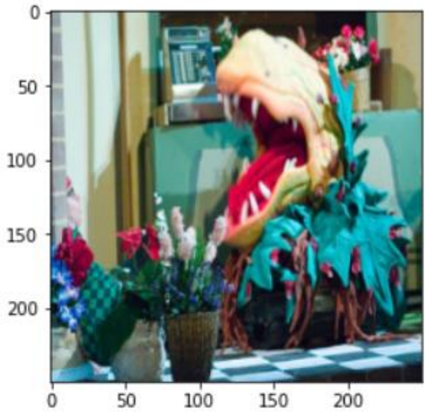
Here the main object we recognize is the cat sitting on the chair which the model also predicts. We can also see chairs and tables in the picture but those objects are cropped by the borders and the cat is clearly the defining object in this image.

```
['person', 'table']
```



Here is an outcome where the model was able to recognize various objects present in the image and gets it correct also.

```
['person']
```



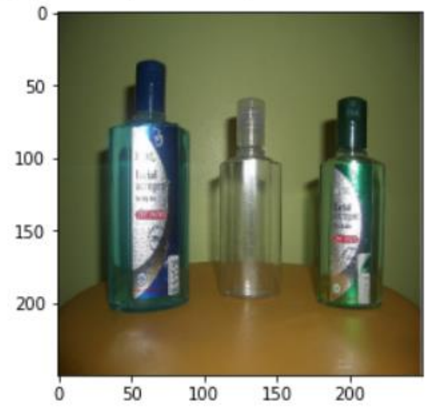
This image contains a variety of objects; some from the labelled classes while others not, however, our classifier was not able to identify these objects and rather gave us the prediction: person.

We shall later discuss the reason for this in our challenges faced section.

In our inference function we have added a confidence parameter which sets the minimum amount of confidence above which we select a label, since the model outputs the probabilities of each label being present in the image. Following example shows the advantage of such an approach:

```
show_result(40, 0.02)
```

```
['person', 'table']
```



The model classifies this image as nothing using the default confidence level i.e., 0.5, however, when we lower our confidence level to 0.02, the model is able to identify the presence of the table in the image.

Challenges Faced

- The labels extracted from the dataset made it biased towards one of the classes. The 'person' class was present in the labels twice as compared to the second most populous label. We tried to balance this by using a weighted loss function, but the metrics didn't change much.
- The credibility of the labels extracted remains a cause for concern and is responsible for relative poor performance compared to other tasks.
- The dataset was quite small, we tried data augmentation, but due to little to no improvement, we didn't use it in final evaluation.

Task-2

Introduction

Typically, image classification in deep learning involves using convolutional neural networks(CNNs) to extract features from images and then a classifier model (usually a feedforward neural network) is trained on those features. In contrast, multimodal image classification involves combining the features extracted from training image data with some metadata that might be present that could aid in the classification. Often, as in our case, this metadata is in the form of some text associated with the image, such as a description of the image.

The goal of multimodal image classification is to find this suitable joint feature representation combining image and text. In our case, the dataset we use is the Pascal50S dataset, that consists of 1000 images belonging to 20 different classes, and each image comes with 50 textual descriptions of itself.

Literature Review

Up until recently, computer vision and natural language understanding (NLU) have progressed parallely and independently of each other, and both fields have seen significant breakthroughs in the recent decade, the some of the most important ones being AlexNet in 2012 for computer vision and Transformers and the attention model introduced in Vaswani et. al., 2017[1] for natural language processing.

Other recent studies such as Goodfellow et. al., 2017[2] have indicated that representations of images and text learned from very deep neural networks with exceptionally large datasets for some task T1 are transferable to another task T2 with a new dataset with remarkable result. This approach is called transfer learning and has shown significant promise in this decade.

The approach we take for our task of Multimodal image classification on the Pascal50S dataset is inspired by Miller et. al., 2020[3]. We use a similar model that extracts the features of an image and its related text using pretrained models trained on large datasets and groups the textual

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit,J., Jones, L., Gomez,A.N., . . . Polosukhin, I. (2017). Attention is all you need. url: https://arxiv.org/abs/1706.03762

[2] Goodfellow, I., Yoshua Bengio, & Courville, A. (2017). Transfer learning and domain adaptation. In T. Dietterich (Ed.),Deep learning (pp. 526-531). Cambridge, MA: MIT Press.

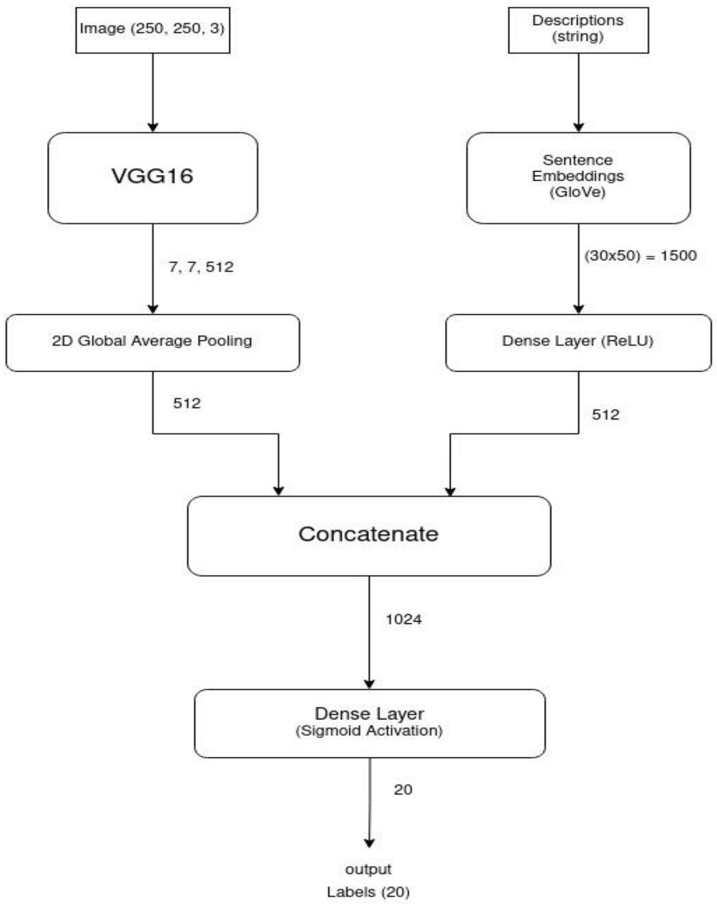
[3] Miller, Stuart J.; Howard, Justin; Adams, Paul; Schwan, Mel; and Slater, Robert (2020) "Multi-Modal Classification Using Images and Text," *SMU Data Science Review*: Vol. 3 : No. 3 , Article 6.

URL: https://scholar.smu.edu/datasciencereview/vol3/iss3/6 features and image features separately and concatenates them into one feature vector. This method of concatenation instead of some mathematical operation or free shuffling and mixing makes the model resilient to missing or slightly misleading data, as the model can easily learn to ignore the part of the feature that corresponds to missing data. The Pascal50S dataset doesn't have any missing data, so this hypothesis has not been tested in our model, but it has been tested in the original paper and this method of combining the individual features of image and text has been shown to be effective.

Model

As mentioned previously, we use a model that concatenates features extracted separately from image and text and use it as our final feature to train a feedforward network on which serves as the classifier.

Here is a visual illustration of the model we have used for Task 2:



The image (resized to 250, 250 along each of its 3 channels) is fed to a VGG16 model initialized with the pretrained imagenet weights and a feature representation of the image of shape (7, 7, 512) is produced. This is passed through a 2-dimensional Global Average Pooling layer to produce the final image feature representation. The choice of 2-dimensional global average pooling layer instead of a flatten layer has been made to reduce the number of dimensions in the final representation, to keep the training time and the number of trainable parameters low.

The feature representation for the captions, that is the sentence embeddings, are generated using a slightly different approach than in the paper referenced. We use the pretrained GloVe word embeddings to generate a sentence embedding. What we do is simply concatenate the word embeddings of each word in the order they appear in the sentence to create the embedding of the sentence as a whole. We have chosen the GloVe 50d word embeddings to do so, which simply means that each word in the sentence is represented by a vector of length 50. The sequence/sentence length has been fixed to 30, which is taken to be close to the average caption length in the dataset. Therefore, our sentence embedding thus generated is of length $50 \times 30 \ = 1500$. We use a linear densely connected layer to encode this vector into a vector of reduced dimensions (512, 1) and facilitate faster training.

Note that the sentence embeddings we generate are simply a concatenation of individual word embeddings of the sentence. This representation may fail to capture the fact that words in a sentence may have different levels of emphasis (or weights, in mathematical terms) in the semantic meaning of the overall sentence. So, a better way would have definitely been to use a model that takes this into account and generates sentence embeddings that capture this semantic feature of language. One such model is a Universal Sentence Encoder (USE) such as Google’s BERT which is a transformers based model that encodes sentences into embedding vectors that take into account the contextual relationship between words (via the attention mechanism) that can be used for transfer learning. However, this model was not chosen for the project simply because of the fact that we, the group members, did not fully understand how the transformers architecture really works, and thus we felt it would be dishonest to use such powerful pretrained models in our project without understanding the details behind them. Therefore, we chose to go with GloVe, which we understood the details of and could therefore reason about freely with respect to results of the model.

Once the features for images and text are extracted by methods described above, we prepare a joint representation by simply concatenating both feature vectors, which is then fed to a feedforward classifier network with sigmoid activations in the final layer, in which the output of each unit represents the probability of the corresponding class being present in the image (as is the norm for multi-label classification).

The choice of concatenating the two separate features was made instead of performing some mathematical operation on both for the same reason as given in the referenced paper. Grouping the features separately and concatenating them makes the model resilient to the occasional samples where either the image or the text is missing or corrupted/nonsensical. However, since our dataset has no such samples, this hypothesis has not been tested in our case, but the referenced paper tests it.

This concludes the details about the model used for training.

Training Details

Each sample in the dataset consisted of three parts, the image, one sentence that describes the image, and the target labels. There were a total of 1000 images in the dataset, and 50 captions for each image. Therefore, the total size of the training dataset was 50,000 samples.

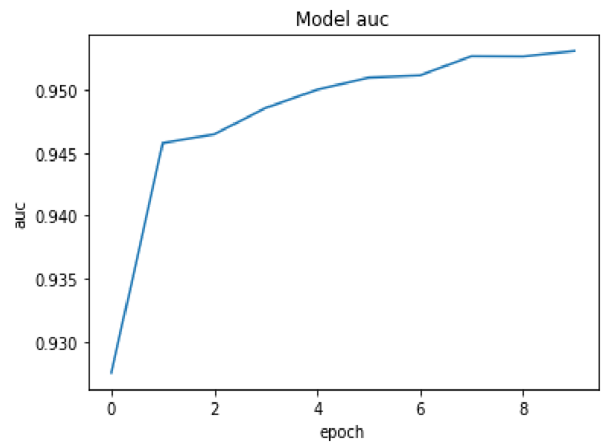
10% of this dataset was split for testing, and the model was trained on 45,000 samples.

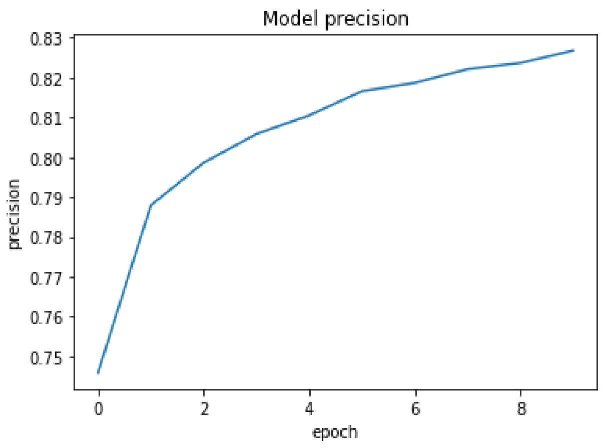
Ideally, the hyperparameters would have been chosen via cross validation like they were done for task 1. However, things aren't always ideal, and the training time for this task, at about 5 minutes per epoch was significantly larger than that of the model for task 1 which was at about 6 seconds for each epoch.

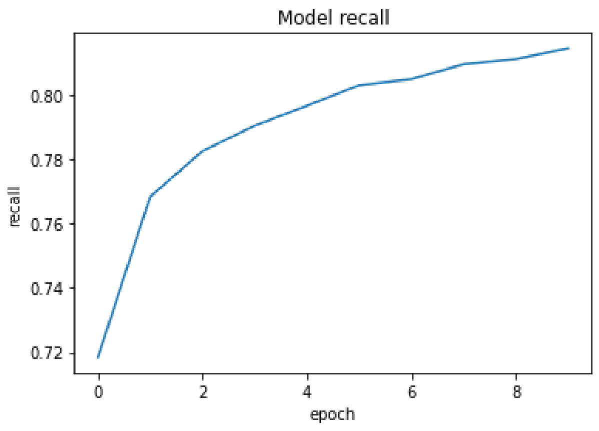
Therefore, in the interest of time, we chose to start with the same hyperparameters that we obtained from cross-validation performed in task 1, and tweaked them manually based on training with a 10% validation split of the training dataset. Once, we obtained the final hyperparameters, we removed the validation split and trained the model on all the 45,000 training samples using those hyperparameters.

The loss used for the training was binary-crossentropy loss and the metrics used to gauge the performance were the same as those in task 1, namely, AUC, precision, and recall. As mentioned already, accuracy is not a particularly good metric for multi-label classification, so it's better to use metrics such as AUC, precision, recall, f1 score, hamming loss etc.

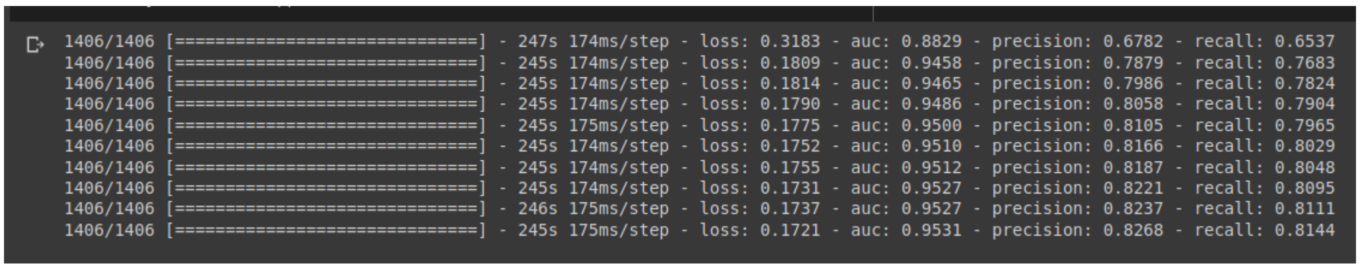
Here are the plots on the training metrics:







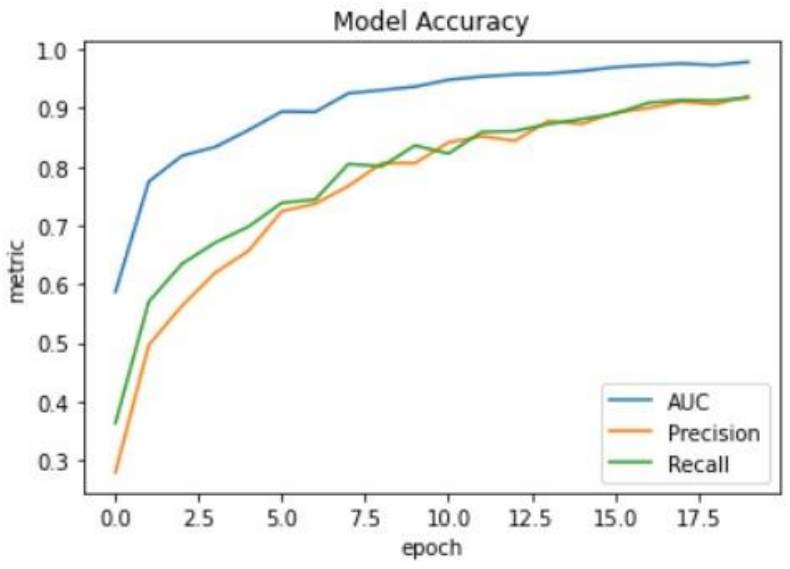
And here is the training history:



```
1406/1406 [==============================] - 247s 174ms/step - loss: 0.3183 - auc: 0.8829 - precision: 0.6782 - recall: 0.6537
1406/1406 [==============================] - 245s 174ms/step - loss: 0.1809 - auc: 0.9458 - precision: 0.7879 - recall: 0.7683
1406/1406 [==============================] - 245s 174ms/step - loss: 0.1814 - auc: 0.9465 - precision: 0.7986 - recall: 0.7824
1406/1406 [==============================] - 245s 174ms/step - loss: 0.1790 - auc: 0.9486 - precision: 0.8058 - recall: 0.7904
1406/1406 [==============================] - 245s 175ms/step - loss: 0.1775 - auc: 0.9500 - precision: 0.8105 - recall: 0.7965
1406/1406 [==============================] - 245s 174ms/step - loss: 0.1752 - auc: 0.9510 - precision: 0.8166 - recall: 0.8029
1406/1406 [==============================] - 245s 174ms/step - loss: 0.1755 - auc: 0.9512 - precision: 0.8187 - recall: 0.8048
1406/1406 [==============================] - 245s 174ms/step - loss: 0.1731 - auc: 0.9527 - precision: 0.8221 - recall: 0.8095
1406/1406 [==============================] - 246s 175ms/step - loss: 0.1737 - auc: 0.9527 - precision: 0.8237 - recall: 0.8111
1406/1406 [==============================] - 245s 175ms/step - loss: 0.1721 - auc: 0.9531 - precision: 0.8268 - recall: 0.8144
```

If we compare this to the training history and plots of task 1, we will surprisingly see that the training results of task 1 slightly outperform those of task 2! The model of task 1 had nearly 100% auc, precision and recall, whereas the task 2 model has about 95% auc, and around 82% precision and recall!

But shouldn't multimodal image classification perform better than simple image classification? It should, and as we will see soon, it does, and greatly so.



Training results for task 1

The discrepancy we see in the training results, as we will find out when we see the test results, is simply the result of the task 1 model overfitting the training dataset, because of its small size. The dataset of task 2 is 50 times larger in contrast, so the model does not overfit as much, and so the training performance is slightly low, but the test performance, which we will see next, is much better than that of the model used for task 1, as it should be.

Testing

The model was tested on the 10% of the initial dataset, consisting of images that it had not seen in the training set.

The performance metrics on the test set were the following:

- Test AUC: 0.858
- Test Precision: 0.782
- Test Recall: 0.654

Let's compare this to the best test performance metrics of task 1, which were the following:

- Test AUC: 0.703
- Test Precision: 0.685
- Test Recall: 0.461

These were, however, for the 20% test split, so it may not be a fair comparison.

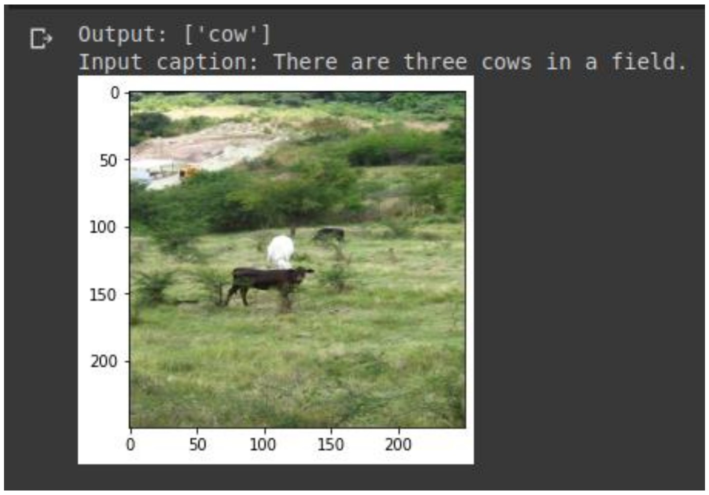
Instead, let's look at the results we obtained when tested on 10% of the dataset and trained on 90% of it:

- Test AUC: 0.753
- Test Precision: 0.718
- Test Recall 0.560

It is clearly evident that the task 2 model outperforms the task 1 model by a significant margin in all metrics tested. This is to be expected of course, as in an image classification task, it could be argued that textual descriptions may convey more information than the images themselves, especially if the labels are present in the text directly, as in our case. Thus, multimodal classification is expected to perform much better, even with the non-ideal sentence embeddings we have chosen.

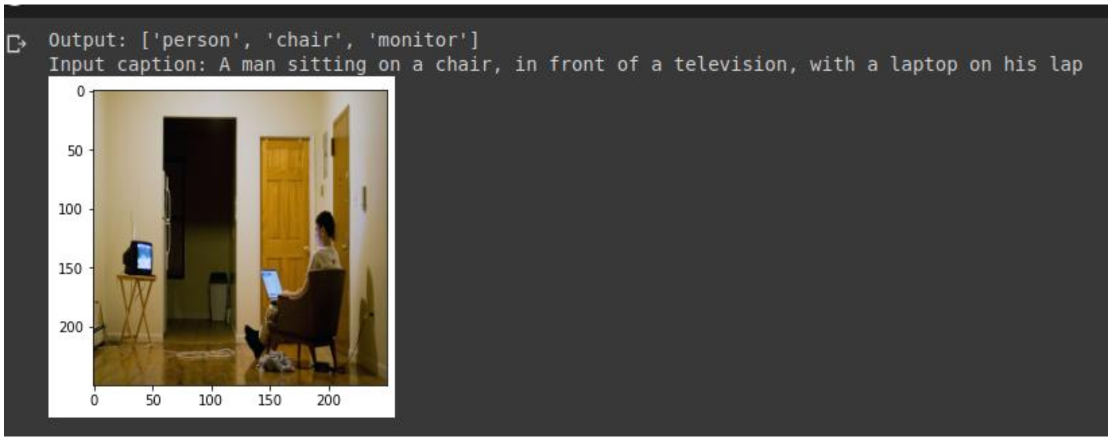
Some results produced by the model

To close the report for task 2, let's look at some results produced by the final model on the test dataset:



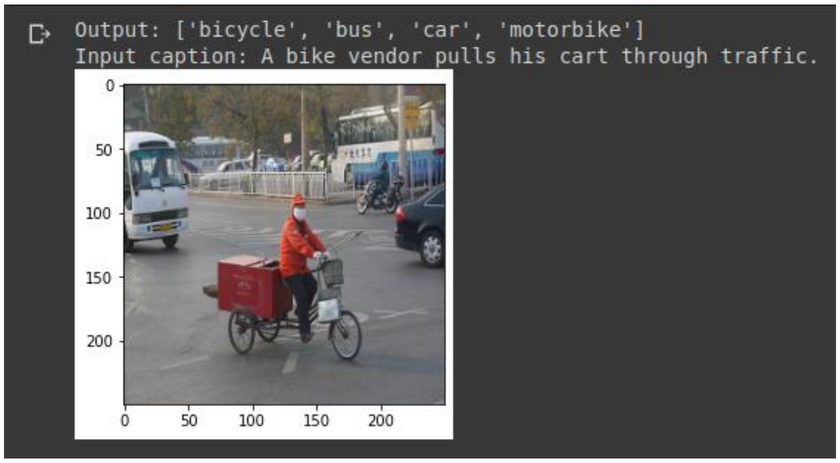
```
Output: ['cow']
Input caption: There are three cows in a field.
```

The model successfully figures out that there are cows present in the image. This is an especially significant result because in task 1 the cow class was one of the classes the model was facing the most difficulty in identifying.



```
Output: ['person', 'chair', 'monitor']
Input caption: A man sitting on a chair, in front of a television, with a laptop on his lap
```

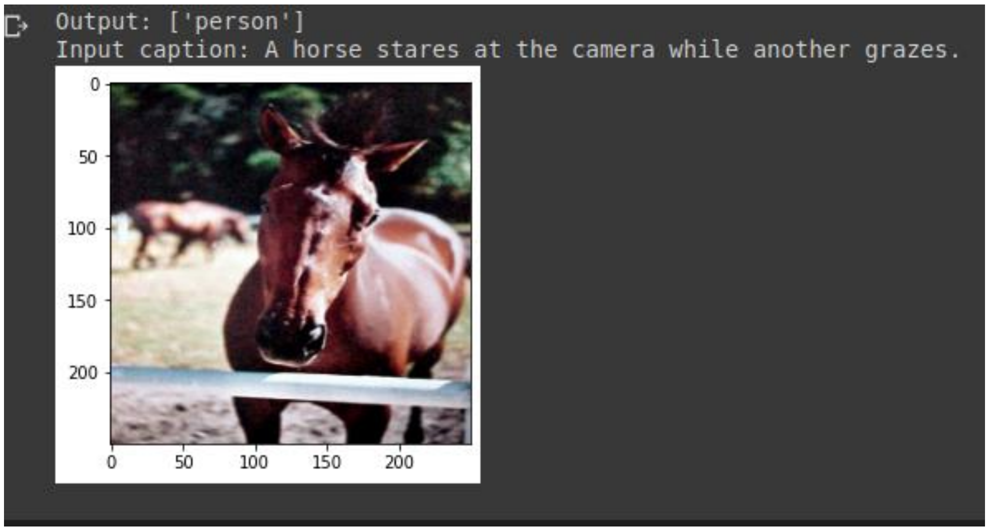
Again, the model performed reasonably well on this image, identifying all but the table in the image. The model seems to miss some classes in images at times, which is also to be expected by the recall score of 65%, which indicates that the model will miss 35% of the classes present in an image on average.



```
Output: ['bicycle', 'bus', 'car', 'motorbike']
Input caption: A bike vendor pulls his cart through traffic.
```

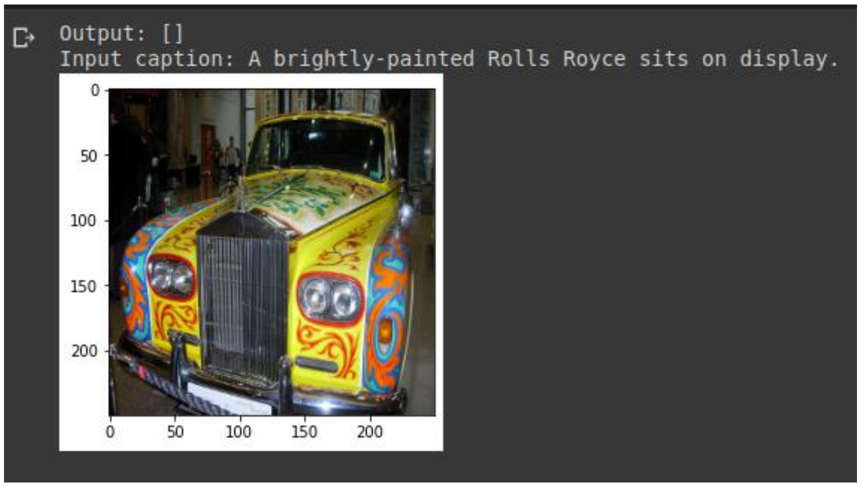
This is again, a remarkable result, as the model identifies 4 out of 5 labels present in the image successfully, missing only the person class. This is even though the captions do not mention any of the objects except for the bike, therefore the model used the image to identify the classes and perhaps the word “traffic” in the sentence helped it look for all vehicles that are in the image.

Now, of course, we’d be lying if we claimed that all results of the model were remarkably good. There were some terrible results as well, some of which are shown below:



```
Output: ['person']
Input caption: A horse stares at the camera while another grazes.
```

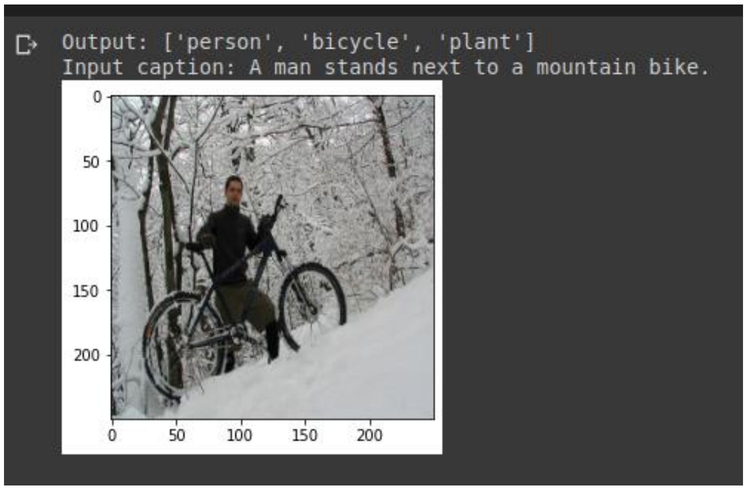
Here, we see the model fails and misclassifies a horse as a human. The model might have gotten confused due to the fact that the horse is looking directly into the camera with a straight face, just as a human would, but regardless, the model should be able to classify this correctly because the description contains the word horse in it. So, it may come down to simply the imbalance in the data. We saw in task 1 that the person class heavily outweighs any other class in the dataset in terms of frequency, and therefore most misclassification errors the model makes will have to do with the person class being involved.



```
Output: []
Input caption: A brightly-painted Rolls Royce sits on display.
```

In this example, the model fails to identify anything. It should ideally identify the car, but it is perhaps failing to do so because the car is covered in uniquely bright and vibrant colors, something the model may have never encountered in the training dataset.

Finally, we will close our discussion of task 2 with a perfect result from the model:



```
Output: ['person', 'bicycle', 'plant']
Input caption: A man stands next to a mountain bike.
```

Here, we see the model predicts all the correct class labels for the image and doesn't predict any wrong labels. This is an example of perfect classification, with 100% precision recall.

Thus, ends our discussion of Task 2- Multimodal Classification

TASK-3

Problem Statement

Using a combination of networks like RNN, CNN and probably FNN build a model for image description. The image and associated sentences can be used for training the model.

Overview and Motivation

When humans are asked to describe a picture, it is a very easy task for them. Even five-year-old kids can describe a picture very easily. But when it comes to Artificial Intelligence, this was one of the most daunting tasks, until recently. The PhD thesis of Andrej Karapathy [4] has led to recent advancement in this problem. The model presented in this Report closely follows the design semantics of his study.

Image captioning has various uses. It can be used as an aid to visually impaired people, better description of CCTV footages for security, better google image search etc.

Data Preparation

Data

The data used in the task is taken from PASCAL50S dataset. The dataset used contains 1000 images with each containing 50 descriptions. There are mainly 20 different objects in the dataset.

Preprocessing

Train-Test Split: Out of the 1000 images, 950 were randomly chosen for train and remaining 50 for test. The split was done in this fashion because in image captioning one requires a good amount of dataset to train a model which produces acceptable results on unseen data. Different splits were tried and finally in order to have good test results the 950-50 split was chosen.

Images: Since the dimensions of all the images in the dataset wasn't same, all the images were first converted to 250x250x3.

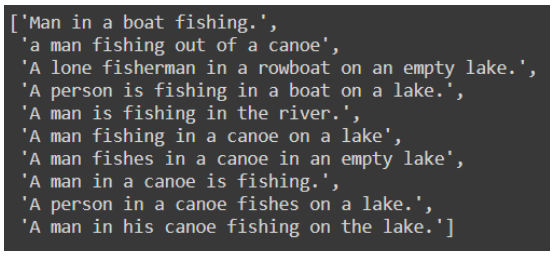
Text Descriptions: There were 50 descriptions for each image making a total of 50K descriptions in the whole dataset and 47.5K descriptions in train part. The descriptions in the dataset included upper-case letters, punctuation marks, alpha-numeric words etc. All these were preprocessed – upper cases converted to lower case so that the two words with same

[4] https://cs.stanford.edu/people/karpathy/deepimagesent/

meaning aren't treated differently, punctuation marks and alpha numeric words were removed and finally it was observed that the article 'a' had a very high frequency which were affecting the predicted captions too much, so all the single letter words like 'a', 'I', etc. were removed. In order to RNN (which we have used) to identify start and end of sentence, two tokens 'Start_Seq' and 'End_Seq' were added.

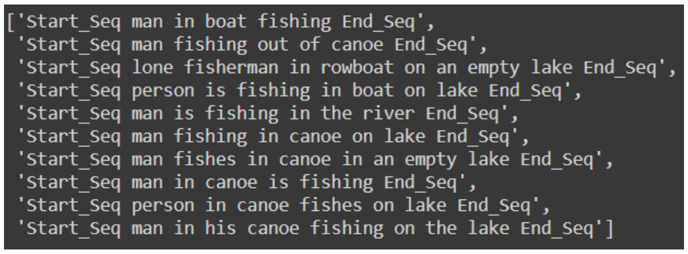
A typical text transformation looked like this:

Original



```
['Man in a boat fishing.',
 'a man fishing out of a canoe',
 'A lone fisherman in a rowboat on an empty lake.',
 'A person is fishing in a boat on a lake.',
 'A man is fishing in the river.',
 'A man fishing in a canoe on a lake',
 'A man fishes in a canoe in an empty lake',
 'A man in a canoe is fishing.',
 'A person in a canoe fishes on a lake.',
 'A man in his canoe fishing on the lake.']
```

Transformed



```
['Start_Seq man in boat fishing End_Seq',
 'Start_Seq man fishing out of canoe End_Seq',
 'Start_Seq lone fisherman in rowboat on an empty lake End_Seq',
 'Start_Seq person is fishing in boat on lake End_Seq',
 'Start_Seq man is fishing in the river End_Seq',
 'Start_Seq man fishing in canoe on lake End_Seq',
 'Start_Seq man fishes in canoe in an empty lake End_Seq',
 'Start_Seq man in canoe is fishing End_Seq',
 'Start_Seq person in canoe fishes on lake End_Seq',
 'Start_Seq man in his canoe fishing on the lake End_Seq']
```

A vocabulary consisting of all the words was made. However, all those words which occurred less than or equal to 5 times were removed. This was done in order to make the model more robust to outliers and make less mistakes. It was found that the vocabulary consisted of 2628 such words.

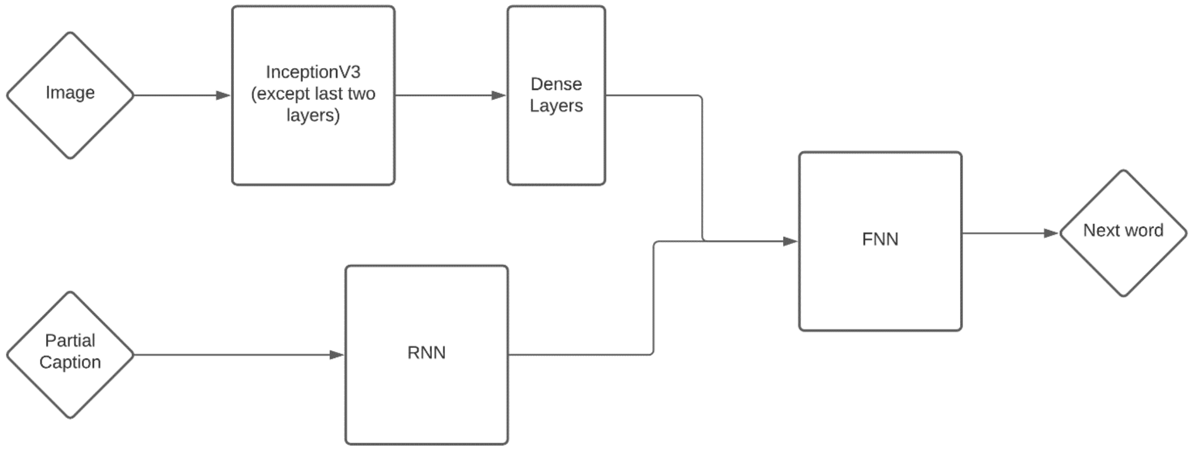
Several other things – like maximum length of a description and word to index and index to word dictionaries were added to facilitate the code.

Encoding Images

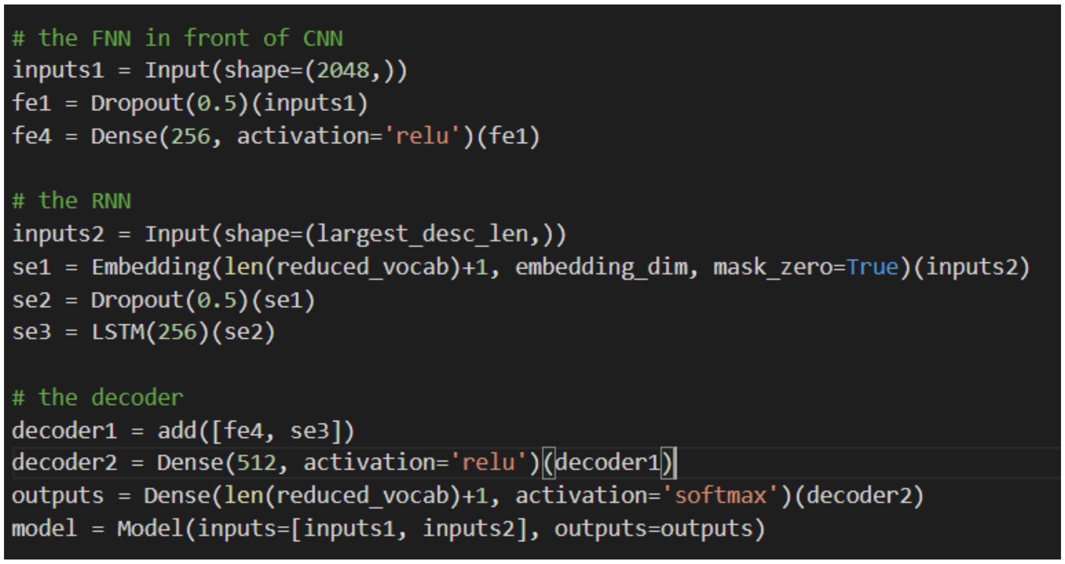
Instead of adding our own convolutional layers, we relied on using pretrained models and weights to get the image encodings. By image encodings we mean the output of CNN layers of the pretrained layer which is further used by our model to train. The pretrained model which has been used is InceptionV3. The resized images (250x250x3) were first preprocessed using InceptionV3 preprocess function, which is recommended to use before supplying input to the model, and then the output of model's 2nd last layer was used as encodings. We have used the pretrained weights and not our own. One of the primary reasons for that the no. of images that we have in our data is very less and in order to have a CNN model we need to train it on large no. of images. So, instead of training our own model, it was found that pretrained models were giving much better results.

Other models such as VGG16 was also tried but it was found that Bleu Score performance of InceptionV3 was better. One of the primary reasons which we came up with was VGG16 did not perform well for the difficult tasks as it is a simple stack of convolutional and max-pooling layers followed by one another and finally fully connected layers. In simpler words, it is not able to extract very complex features. On the other hand, Inception nets have inception modules that consist of 1X1 filters also known as pointwise convolutions followed by convolutional layers with different filter sizes applied simultaneously. This allows Inception nets to learn more complex features.

Model



This is how the model looks like. The image is first passed through InceptionV3 to get encodings as stated previously. Then these encodings pass through some layers of FNN and then their output is passed to another FNN but this output is combined with the output of RNN part. The RNN part takes the partial caption as input, has an embedding, dropout and LSTM layer which provides the output of this part. The output of this part and the output of above part is combined together which forms an input to FNN part. So, here the InceptionV3 and Dense layers in front of it acts like encoder and the second FNN part acts like decoder which uses output of RNN part to decode the probabilities of next word.



```
# the FNN in front of CNN
inputs1 = Input(shape=(2048,))
fe1 = Dropout(0.5)(inputs1)
fe4 = Dense(256, activation='relu')(fe1)

# the RNN
inputs2 = Input(shape=(largest_desc_len,))
se1 = Embedding(len(reduced_vocab)+1, embedding_dim, mask_zero=True)(inputs2)
se2 = Dropout(0.5)(se1)
se3 = LSTM(256)(se2)

# the decoder
decoder1 = add([fe4, se3])
decoder2 = Dense(512, activation='relu')(decoder1)
outputs = Dense(len(reduced_vocab)+1, activation='softmax')(decoder2)
model = Model(inputs=[inputs1, inputs2], outputs=outputs)
```

This is the complete architecture of the model. The FNN in front of CNN gets input the image encodings received from InceptionV3. The dropout used is 0.5 in order to reduce overfitting. There were many other things which were tried like more no. of layers, different no. of units and activation function {relu and leaky relu}, but this worked best for us.

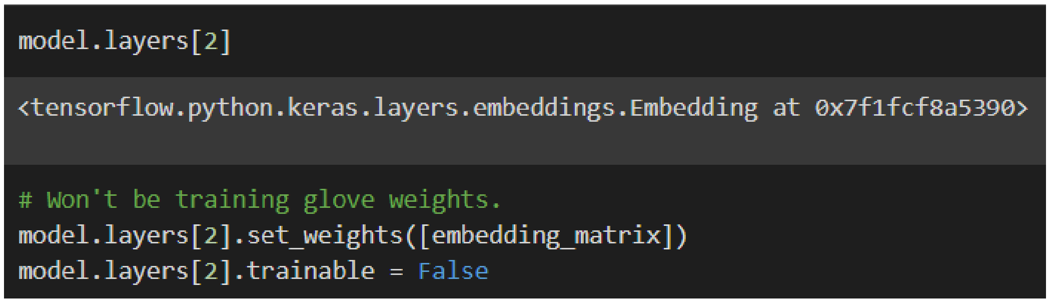
Generator

In order to meet RAM requirements, instead of making a full dataset carrying image encodings and partial sentences since we usually train our model in batches of data points, we use a generator. The generator which we have designed closely follows the generators which are already present in Keras with some differences.

Our dataset contains 50 captions per image, so if we naively select five images per batch, the batch created will have around 5*50*(max length of a caption) points. Also, all 50 descriptions will be given to the model at once which is clearly a poor way to train because then the model will only try to remember the last image which it saw. So, to overcome this ordeal, instead of giving all descriptions at once the generator will use only n number of descriptions which will be input from the user. In order to train on all descriptions, we kept a memory system in our generator in form of a dictionary which maintains the count of which all descriptions have already been used and would give out the next n descriptions. The output of the generator is X and Y, X is a tuple because our model accepts two inputs – image encodings and partial sentences. The output Y is a one-hot encoded vector of length equal to vocabulary size plus one. It specifies the probability of next word. The generator outputs the batch and when called again will resume from where it stopped earlier.

Word Embeddings

We have mapped every word to 300-d long vector. For that we have used pretrained Glove Model. Then we formed a matrix corresponding to each word in our vocabulary and the 300-d vectors which we used in our embedding layer. Note that we are not training this layer but using the pretrained weights of Glove.



```
model.layers[2]
```

```
<tensorflow.python.keras.layers.embeddings.Embedding at 0x7f1fcf8a5390>
```

```
# Won't be training glove weights.
model.layers[2].set_weights([embedding_matrix])
model.layers[2].trainable = False
```

Training

The optimizer used for training the model was Adam with Categorical cross entropy as the loss function. The training was done in steps. First the model was trained on default values of initial learning rate and small no. of images per batch. Then once done with that, we increased the no. of images per batch and reduced the learning rate by a factor of 10. A significant decrease in loss was observed. We followed this approach for two times and after that considered our model trained.

While training we did try other hyper-parameters such as no. of captions to be used per image in a batch, no. of images per batch and different learning rate. However, we didn't perform any cross validation, the final parameters we selected was based on mere observation of captions on unknown data and Bleu Score.

Note that all code related details can be found in the code itself. This report only contains the high-level overview.

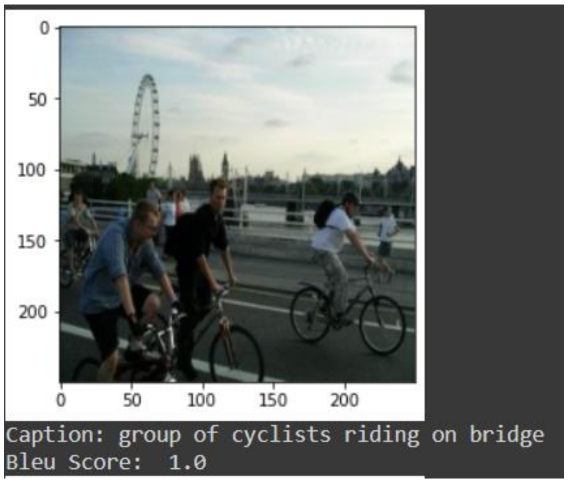
Testing out the model

In order to have a final function to generate a model, greedy method was used to find the caption. The method feeds into image encoding and start symbol ('Start_Seq') to the model, and greedily selects the highest probable next word and feeds it back to the model until end symbol ('End_Seq') is met.

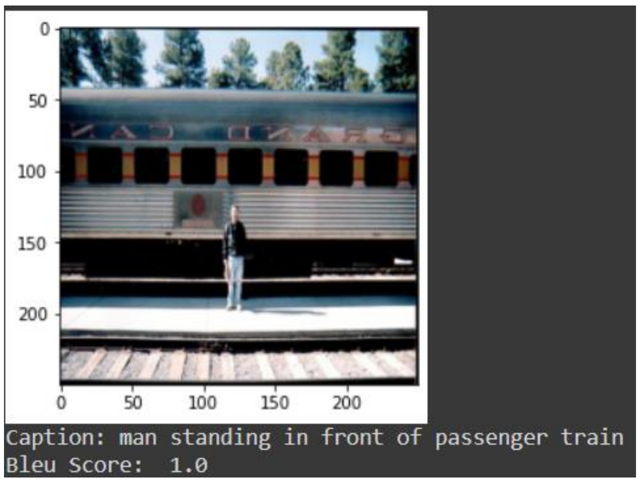
Evaluation metric: We used Bleu Score as out evaluation metric for images present in our dataset. Bleu score was used because it is generally used in translation models and it kinds of tell how close the prediction is from the actual results.

On Train data

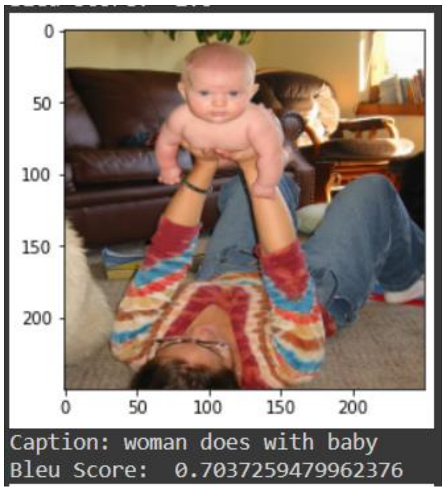
Some results on train data:



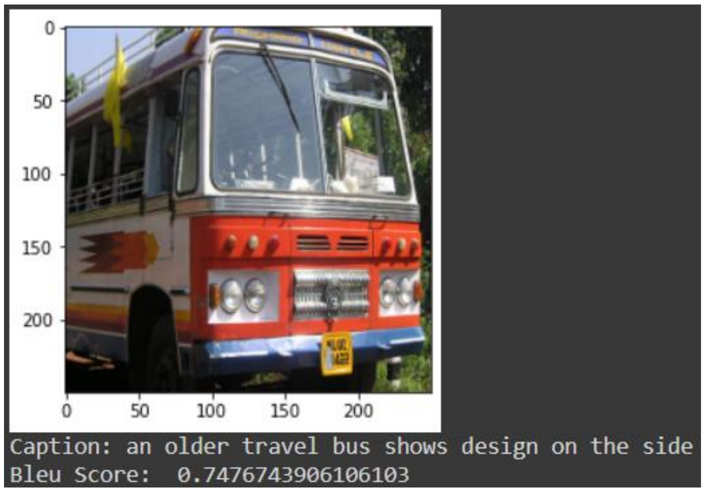
```
Caption: group of cyclists riding on bridge
Bleu Score:  1.0
```



```
Caption: man standing in front of passenger train
Bleu Score:  1.0
```



```
Caption: woman does with baby
Bleu Score:  0.7037259479962376
```

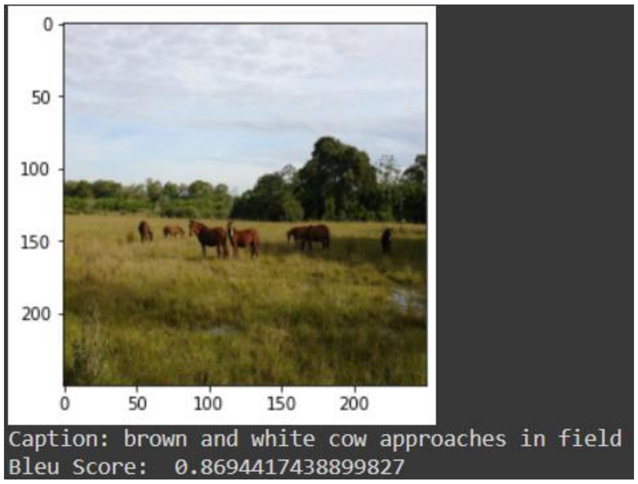


```
Caption: an older travel bus shows design on the side
Bleu Score:  0.7476743906106103
```

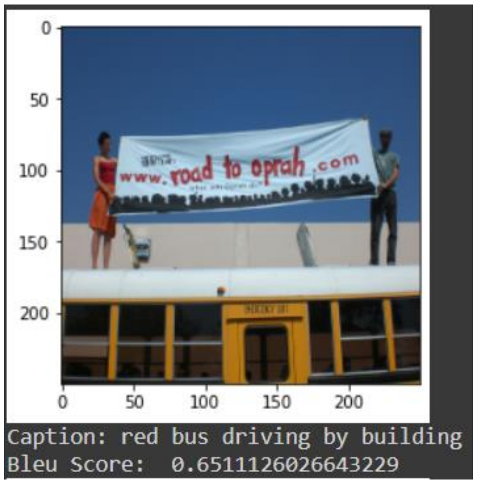
As can be seen the captions generated on train data are very much detailed with very good bleu score. This slightly indicates over-fitting.

On test data

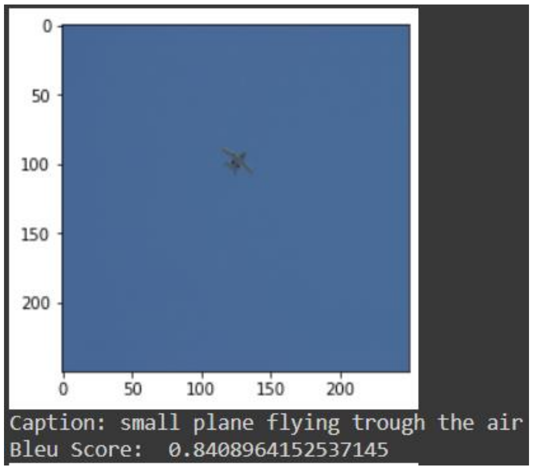
Some results on test data:



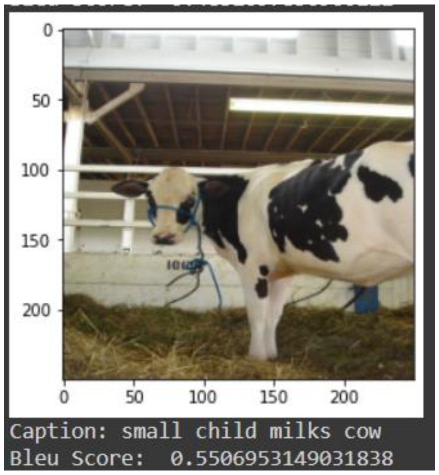
```
Caption: brown and white cow approaches in field
Bleu Score:  0.8694417438899827
```



```
Caption: red bus driving by building
Bleu Score:  0.6511126026643229
```



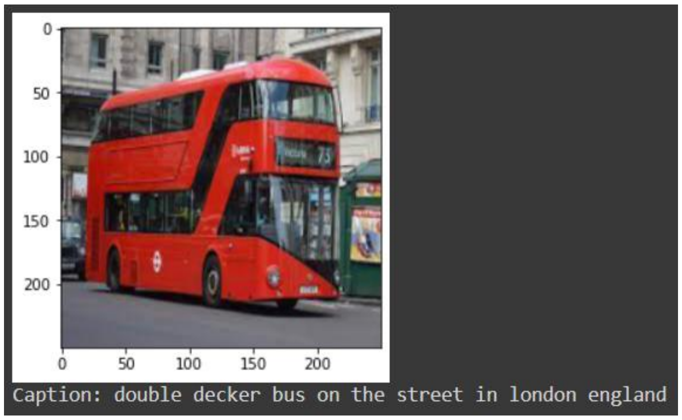
```
Caption: small plane flying trough the air
Bleu Score:  0.8408964152537145
```



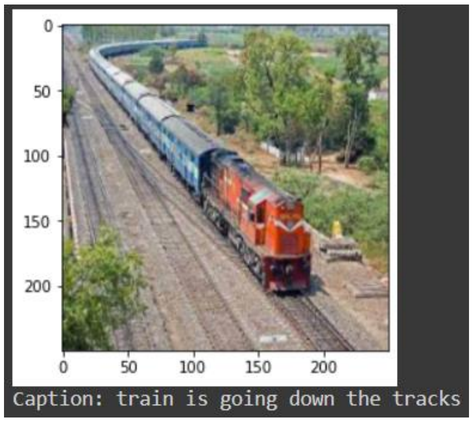
```
Caption: small child milks cow
Bleu Score:  0.5506953149031838
```

As can be seen the model is able to identify major objects from the unknown images but it is not able to clearly identify what action is going on. One of the primary reasons for that is a small training data. The Bleu score as can be seen is considerably low as compared to train scores.

Random Internet Images:

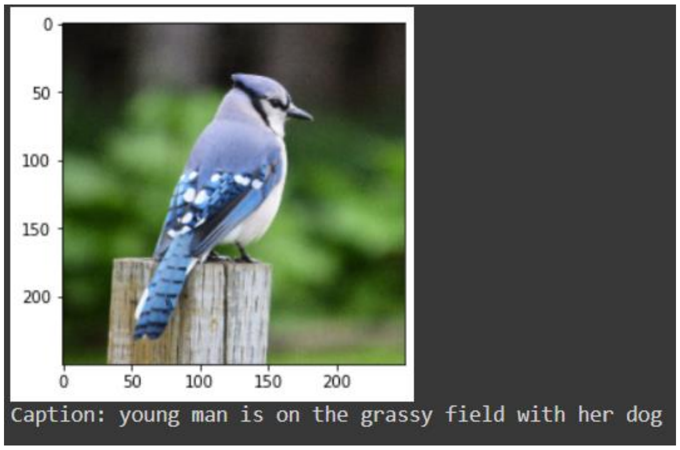


```
Caption: double decker bus on the street in london england
```

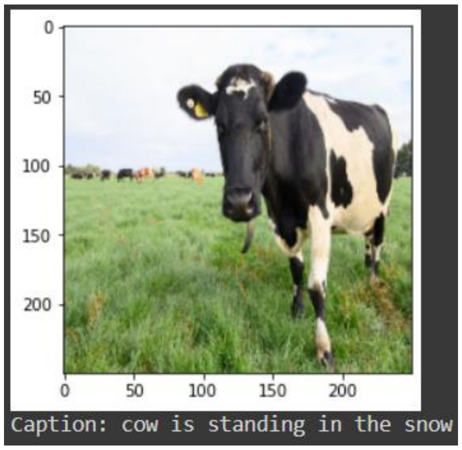


```
Caption: train is going down the tracks
```

On some of the images like these the captions generated were pretty good. While on some, stated below, the captions generated were completely ridiculous.



```
Caption: young man is on the grassy field with her dog
```



```
Caption: cow is standing in the snow
```

As can be seen, in the first image the model was not at all able to figure out the bird and gave out a strange caption. In the second image the model was able to figure out the cow, but it couldn't figure out the grass and called it snow. The reasons for this can be that in the train dataset there were less images of birds and cows and probably that's why it is not able to figure these out clearly.

For more results and details of the code you can look up the notebook submitted.