INDIAN INSTITUTE OF TECHNOLOGY PALAKKAD

CS 5007: Deep Learning

Project: Pascal-VOC 2008

Team 7:

- Yogesh R, 111801047
- Subhash S, 111801042
- Neeraj P, 111801027

Mentor: Rimmon Bhosale

Table of Contents

1 Problem Statement

1.1 Task 1

The images need to be classified into 20 classes (person, bird, cat, cow, dog, horse, sheep, airplane, 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. This is a multi-label classification problem. Perform a simple image classification using deep convolutional neural networks.

1.2 Task 2

Perform a multi-modal image classification by combining representation of text and images

1.3 Task 3

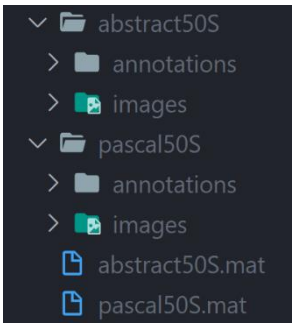
Build a model for image description. The image and associated sentences can be used for training the model.

2 Data Extraction

2.1 Initial Preprocessing

The dataset is downloaded and extracted from https://filebox.ece.vt.edu/~vrama91/CIDEr_miscellanous/cider_datasets.tarB

The annotations were in the form ".mat", which was converted into ".txt", and the files were rearranged into the following structure as shown.



The duplicate entries of annotation were removed.

2.2 Cache generation

We can observe that the initial preprocessing and label extraction are quite time-consuming steps and we are only interested in the result of these to solve our upcoming tasks, so we cache the important result from these processes.

The abstract, and pascal images, annotation, and target (from label extraction) are stored as ".npy" files and upload to google drive, so they could be easily retrieved later without running initial preprocessing and label extraction again. Additionally, resized images of dimensions (224, 224) are also cached and will be using for upcoming tasks.

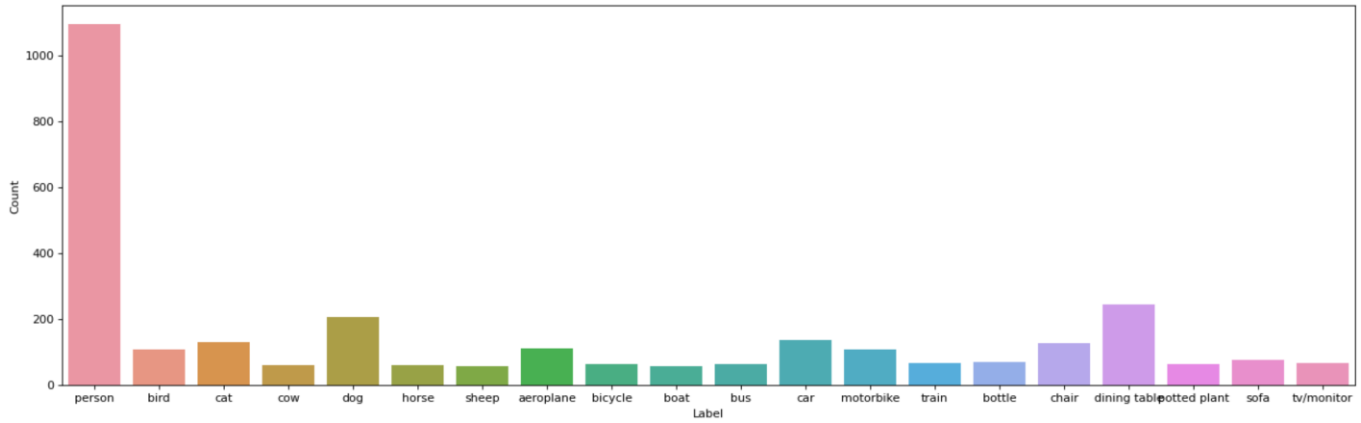
3 Common Attributes of Task 1 and Task 2

3.1 Preprocessing Images

1. All the images were resized to 224 x 224 x 3 and the alpha channel was removed from the abstract dataset.
2. All the images were saved in the cache as NumPy arrays from 0-255.
3. Further preprocessing was performed using InceptionV3's preprocessor function from TensorFlow to match the expected inputs of the InceptionV3 model.
4. For image caption, we obtain the features from VGG16 bypassing the resized images.

3.2 Data imbalance

We combine the images and annotation of abstract and pascal datasets. We observe that the data is highly imbalanced as shown below



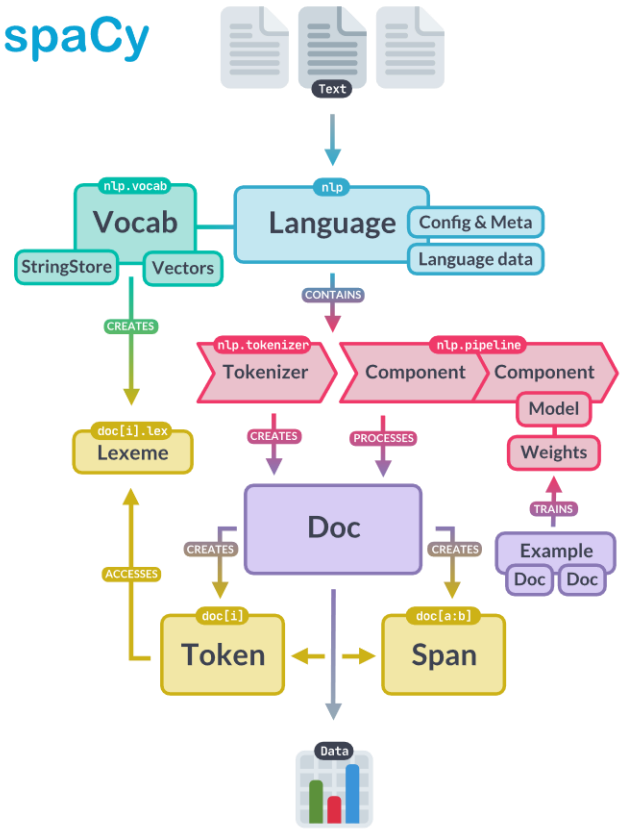
3.3 Loss Function

Initially, we trained on binary cross-entropy but realized that due to an imbalanced dataset, this did not give good results. The soft-f1 loss function was used instead which considered the imbalance in the dataset. This link explains why the soft-f1 loss can be preferred during multi-label classification with an imbalanced dataset. The class_weight parameter in model.fit() function helps in weighing the loss function depending on the number of occurrences of each class in the dataset. Combining the class_weight parameter with the soft-f1 loss helped in training the model to a good f1-score in a short time. The class_weight parameter is only applied while training and not while validating. This explains the fact that the training loss is very high when compared to the validation loss i.e it helps the model "pay more attention" to examples from an under-represented class. For reference, see this link.

4 Task 1

4.1 Label Extraction

The spaCy NLP library was used to achieve label extraction. spaCy converts a text into a doc object after a pipeline of NLP processing as shown.



The label extraction is done in three stages:

1. Finding the exact match of the label in the lemmatized words of the text. E.g., if a cat exists in the list of lemmatized words, then we assume the image has a cat in it.
2. Finding the lemmatized words that closely match the labels. E.g., if the words like a parrot or an eagle are found, we assume the image as a bird.
3. Named Entity Recognizer.

   Some annotations have person in it but are justified using only the proper names of the people in that image. E.g., “Mike and Sam are playing chess in the room”. In this sentence, people are there but it’s not explicitly mentioned. We use spaCy’s named entity recognizer to tackle this problem. As shown in the image below, we can see that spaCy managed to find out that the names are people

Mike PERSON and Sam PERSON are playing chess in the room

4.2 Image Classification

4.2.1 Model Creation

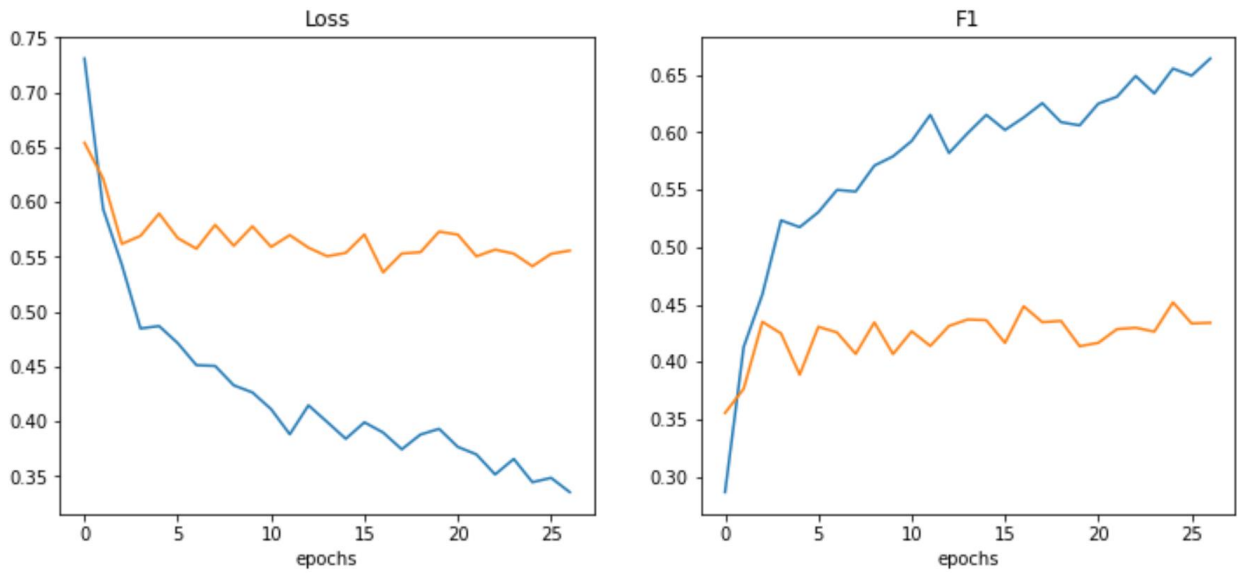
We will be using the InceptionV3 pre-trained CNN model as our base model. The final model architecture is as follows.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
inception_v3 (Functional)    (None, 2048)              21802784
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               1049088
_________________________________________________________________
dense_4 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 20)                10260
=================================================================
Total params: 23,124,788
Trainable params: 1,322,004
Non-trainable params: 21,802,784
```

The images are preprocessed using the inception input preprocessor, split into train and test and the model is compiled using the Adam optimizer and f1 loss function. The popular callbacks such as Early Stopping and Model Checkpoint are used to prevent overfitted and save progress respectively.

The training is done for 27 epochs (early stopping). The following are the result of the training.

4.2.2 Training History



4.2.3 Classification Report

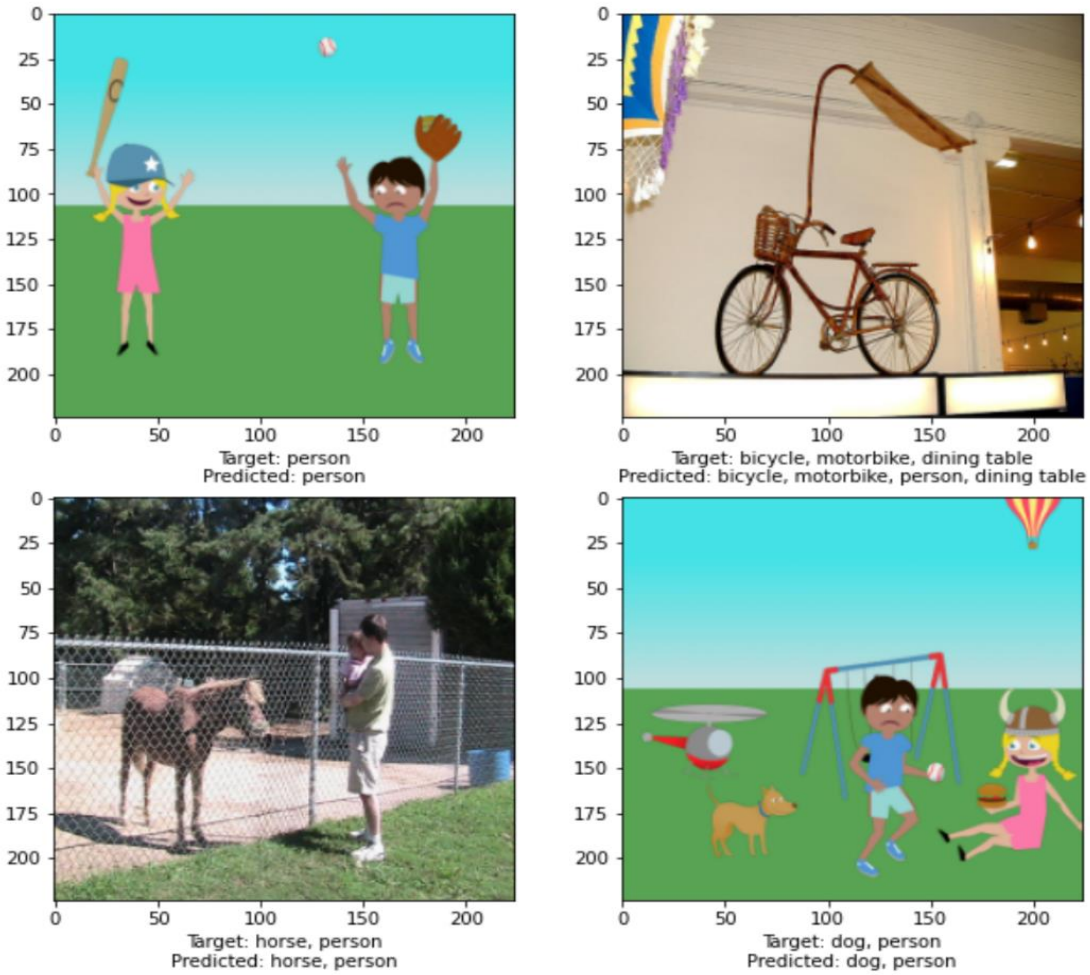
Test Dataset

	precision	recall	f1-score	support
0	1.00	0.77	0.87	300
1	0.33	0.47	0.39	15
2	0.38	0.62	0.47	13
3	0.83	0.77	0.80	13
4	0.41	0.32	0.36	60
5	0.88	1.00	0.93	7
6	0.77	1.00	0.87	10
7	0.42	1.00	0.59	11
8	0.92	0.80	0.86	15
9	0.70	1.00	0.82	7
10	0.67	0.91	0.77	11
11	0.65	0.65	0.65	31
12	0.85	0.89	0.87	19
13	0.82	0.60	0.69	15
14	0.17	0.67	0.27	3
15	0.43	0.75	0.55	16
16	0.50	0.50	0.50	44
17	0.38	0.43	0.40	7
18	0.58	0.44	0.50	16
19	0.67	0.44	0.53	9
micro avg	0.73	0.69	0.71	622
macro avg	0.62	0.70	0.63	622
weighted avg	0.78	0.69	0.72	622
samples avg	0.77	0.74	0.71	622

Full Dataset

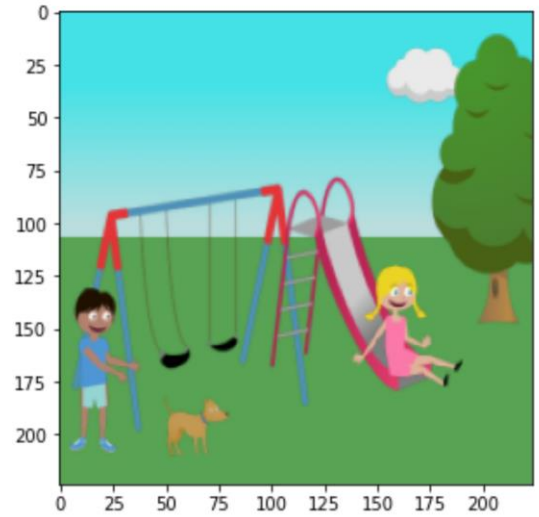
	precision	recall	f1-score	support
0	1.00	0.73	0.84	1500
1	0.50	0.83	0.63	65
2	0.50	0.86	0.63	76
3	0.88	0.90	0.89	58
4	0.68	0.50	0.58	280
5	0.83	0.94	0.88	53
6	0.77	0.98	0.86	45
7	0.69	0.96	0.81	80
8	0.92	0.85	0.88	67
9	0.91	0.98	0.95	54
10	0.78	0.98	0.87	50
11	0.72	0.65	0.68	150
12	0.90	0.98	0.94	99
13	0.88	0.70	0.78	82
14	0.49	0.94	0.64	35
15	0.58	0.80	0.68	91
16	0.70	0.66	0.68	259
17	0.63	0.85	0.72	46
18	0.88	0.62	0.73	105
19	0.85	0.85	0.85	66
micro avg	0.82	0.74	0.78	3261
macro avg	0.76	0.83	0.78	3261
weighted avg	0.86	0.74	0.78	3261
samples avg	0.86	0.78	0.78	3261

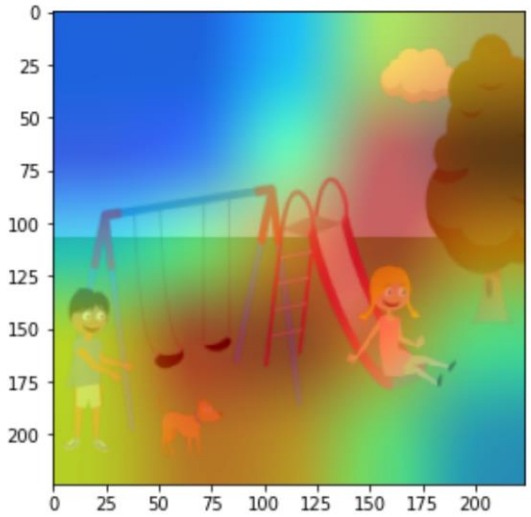
4.2.4 Prediction from the test dataset



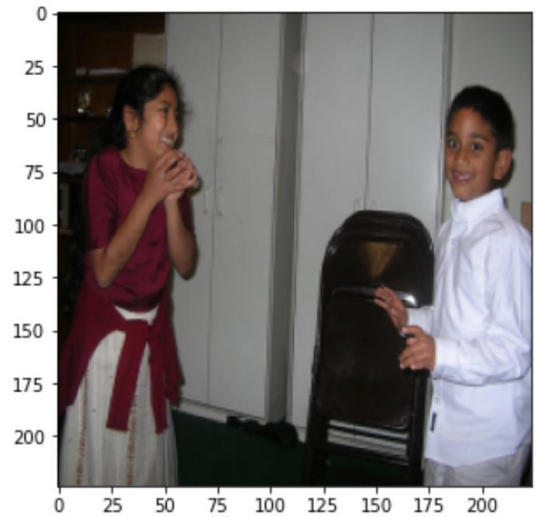
4.2.5 Activation Maps

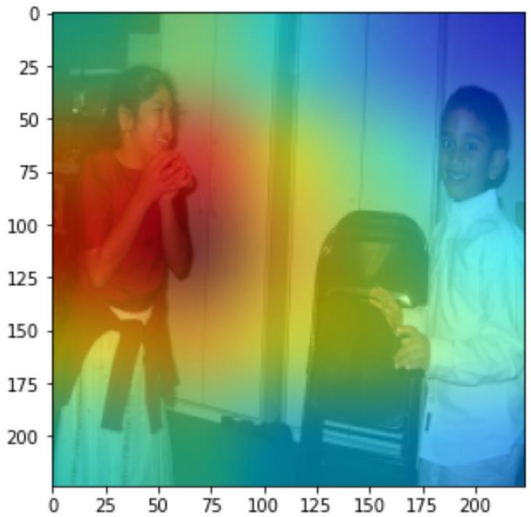
Targets = {'person', 'dog'}





Targets = {'person', 'chair'}





5 Task 2

In this task, we are required to find a combined representation of the image and its corresponding descriptions.

5.1 Idea

The convolution neural network (CNN) will encode the images whereas a custom text model will encode the descriptions and a combined feature vector will represent the image and the text.

5.2 Model Architecture

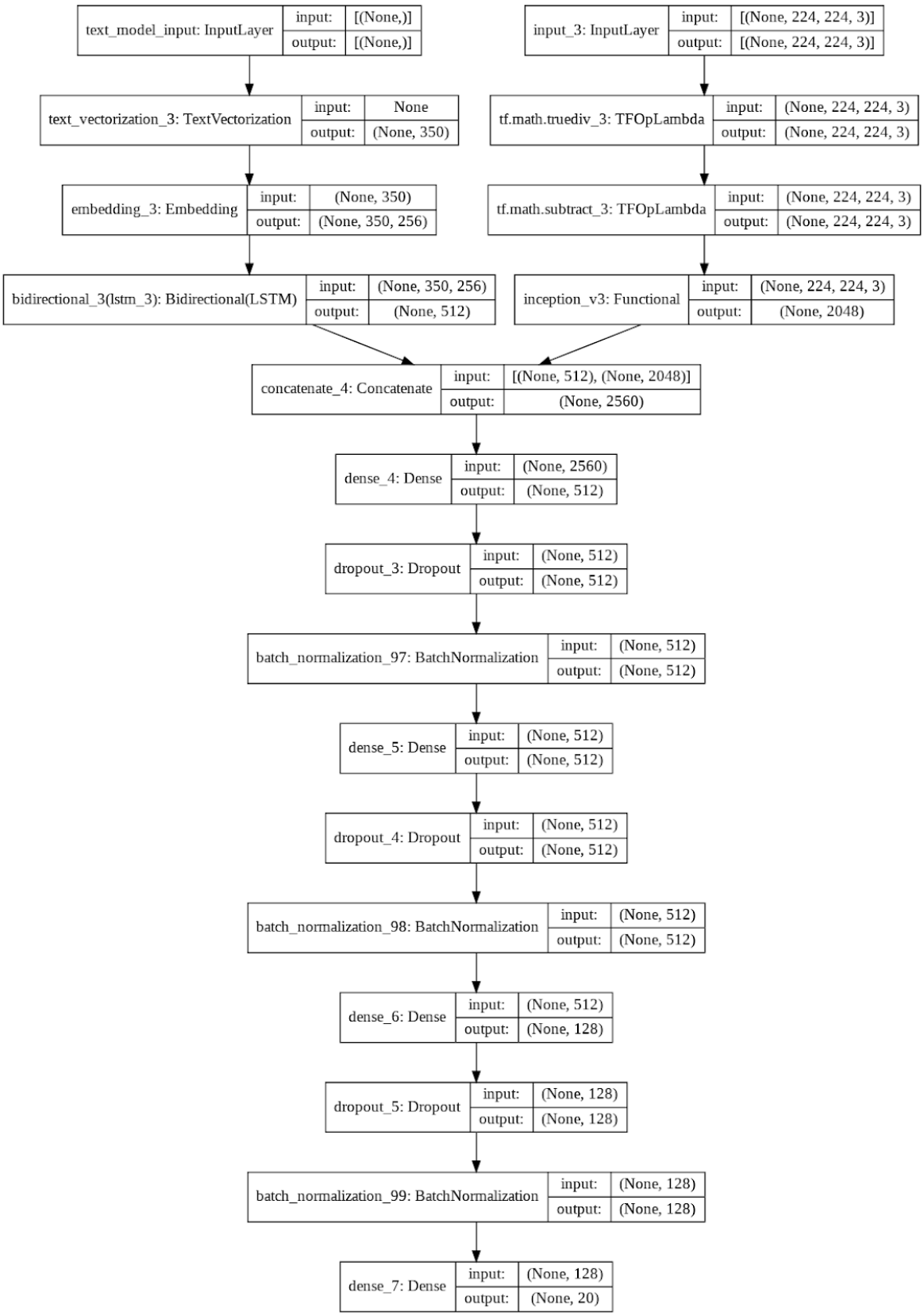
The images are passed through InceptionV3 (a pre-trained CNN model) whereas the descriptions are tokenized, encoded into dense vectors, and are passed through an LSTM layer.

The outputs from the CNN and the text models are concatenated as a single feature vector and are passed through a set of Dense, Dropout, and Batch Normalization layers followed by 20 sigmoid units to represent the 20 output classes.

The below model explains the model architecture. The top-left part of the image is the feature extractor for the descriptions and the top-right part of the image is the feature extractor for the images which also consists of the preprocessing layers (tf.math.truediv_3 and tf.math.subtract_3) which normalize the images along the 3 channels. The bottom half of the image consists of the fully connected layers which are the feature classifiers.

Since this is a multi-label classification problem, we use the sigmoid activation function for the last layer since we want an independent prediction for each class in the image. We use the soft-f1 loss function along with the class_weights parameter in the model.fit which accounts for the imbalance in the dataset.

5.2.1 Model Plot



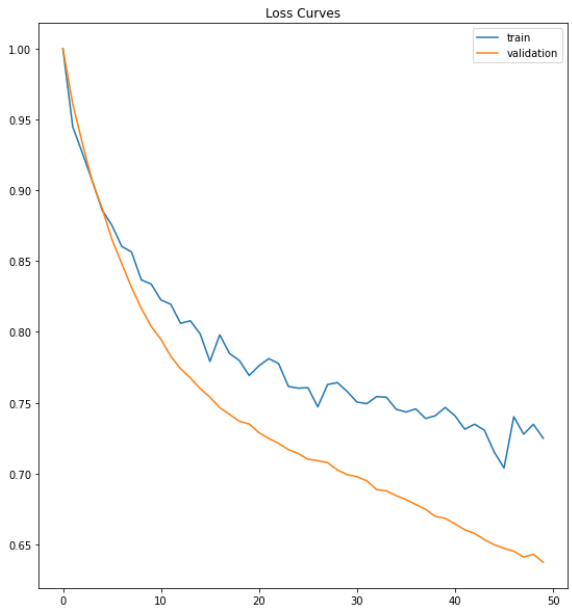
5.2.2 Model Summary

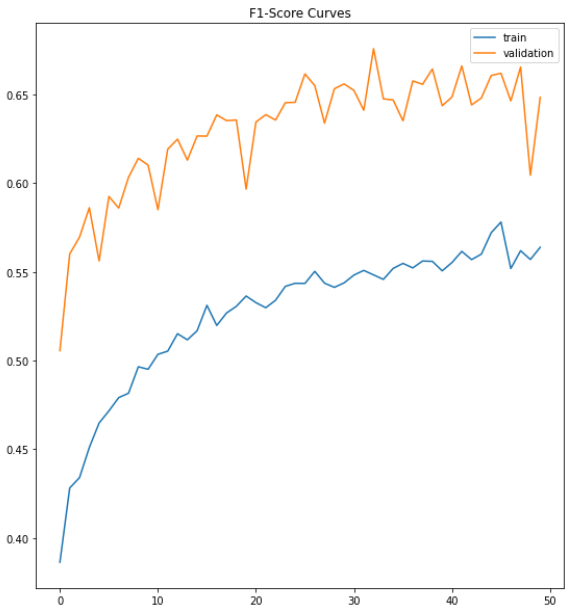
Layer (type)	Output Shape	Param #	Connected to
text_model_input (InputLayer)	[(None,)]	0	
input_5 (InputLayer)	[(None, 224, 224, 3)	0	
text_vectorization_4 (TextVecto	(None, 350)	0	text_model_input[0][0]
tf.math.truediv_4 (TFOpLambda)	(None, 224, 224, 3)	0	input_5[0][0]
embedding_4 (Embedding)	(None, 350, 256)	512000	text_vectorization_4[0][0]
tf.math.subtract_4 (TFOpLambda)	(None, 224, 224, 3)	0	tf.math.truediv_4[0][0]
bidirectional_4 (Bidirectional)	(None, 512)	1050624	embedding_4[0][0]
inception_v3 (Functional)	(None, 2048)	21802784	tf.math.subtract_4[0][0]
concatenate_7 (Concatenate)	(None, 2560)	0	bidirectional_4[0][0] inception_v3[0][0]
dense_8 (Dense)	(None, 512)	1311232	concatenate_7[0][0]
dropout_6 (Dropout)	(None, 512)	0	dense_8[0][0]
batch_normalization_194 (BatchN	(None, 512)	2048	dropout_6[0][0]
dense_9 (Dense)	(None, 512)	262656	batch_normalization_194[0][0]
dropout_7 (Dropout)	(None, 512)	0	dense_9[0][0]
batch_normalization_195 (BatchN	(None, 512)	2048	dropout_7[0][0]
dense_10 (Dense)	(None, 128)	65664	batch_normalization_195[0][0]
dropout_8 (Dropout)	(None, 128)	0	dense_10[0][0]
batch_normalization_196 (BatchN	(None, 128)	512	dropout_8[0][0]
dense_11 (Dense)	(None, 20)	2580	batch_normalization_196[0][0]

Total params: 25,012,148
Trainable params: 3,207,060
Non-trainable params: 21,805,088

Note: Although the above images show the preprocessing layer, these layers would not be visible while running the ipynb because the features are extracted for all images and cached to reduce training time but they are an accurate representation of what our model does.

5.3 Training History





Why is my training loss much higher than my testing loss?

A Keras model has two modes: training and testing. Regularization mechanisms, such as Dropout and L1/L2 weight regularization, are turned off at testing time. They are reflected in the training time loss but not in the test time loss.

The left image consists of training loss and validation loss whereas the right image consists of training f1-score and validation f1-score. We can see that validation metrics are better than that of the training metrics, this can be explained by the use of Dropout. The Dropout regularization is applied only while training as mentioned in the Keras FAQ section which means the validation data uses all the neurons whereas the training data drops some of the neurons, justifying the results obtained.

5.4 Classification Report

Test Dataset	Full Dataset

	precision	recall	f1-score	support
0	0.62	1.00	0.77	932
1	0.90	0.66	0.76	136
2	0.98	0.86	0.91	106
3	0.81	0.84	0.82	85
4	0.61	0.61	0.61	132
5	1.00	0.78	0.87	89
6	0.93	0.83	0.88	84
7	0.92	0.77	0.84	71
8	0.87	0.76	0.81	96
9	1.00	0.77	0.87	104
10	0.76	0.88	0.82	86
11	0.65	0.72	0.68	186
12	0.96	0.81	0.88	140
13	0.98	0.90	0.94	92
14	0.50	0.77	0.61	48
15	0.58	0.51	0.54	194
16	0.62	0.75	0.68	271
17	0.87	0.43	0.58	79
18	0.89	0.82	0.85	109
19	0.79	0.80	0.80	105
micro avg	0.71	0.81	0.76	3145
macro avg	0.81	0.76	0.78	3145
weighted avg	0.74	0.81	0.76	3145
samples avg	0.73	0.85	0.75	3145

	precision	recall	f1-score	support
0	0.73	1.00	0.85	5482
1	0.95	0.71	0.81	533
2	0.98	0.69	0.81	652
3	0.91	0.90	0.90	280
4	0.69	0.71	0.70	1035
5	1.00	0.88	0.94	309
6	0.98	0.92	0.95	282
7	0.96	0.84	0.90	548
8	0.84	0.92	0.88	313
9	1.00	0.89	0.94	290
10	0.84	0.92	0.88	315
11	0.73	0.75	0.74	657
12	0.99	0.90	0.94	547
13	0.97	0.87	0.92	318
14	0.87	0.85	0.86	338
15	0.69	0.66	0.67	621
16	0.70	0.83	0.76	1221
17	0.95	0.66	0.78	319
18	0.88	0.88	0.88	381
19	0.91	0.90	0.90	346
micro avg	0.79	0.88	0.83	14787
macro avg	0.88	0.83	0.85	14787
weighted avg	0.81	0.88	0.83	14787
samples avg	0.82	0.91	0.83	14787

5.5 Predictions from the test dataset

Example 1

1. A tattooed man sits on a motor scooter while another man looks on.
2. A man with tattoos is riding a yellow moped.
3. A guy standing over a yellow motor scooter.
4. A man examines the handlebars of a motorbike.
5. A man sitting on a yellow and black motorbike.

```
Actual: ['person' 'motorbike']
Predicted: ['person' 'motorbike']
```

Example 2

1. The family is posing at a birthday party.
2. Kenneth is having a birthday party with his family.
3. A family is celebrating a birthday with a cake.
4. A birthday party with balloons, cake, and family.
5. A group of people have a birthday party

```
Actual: ['person' 'dining table']
Predicted: ['person' 'dining table']
```

6 Task 3

6.1 Idea

This is a supervised learning task. We are given two datasets i.e., Pascal50S and Abstract50S which contain images along with their corresponding captions. Each image has 50 captions. The basic idea is to somehow use the image and predict the caption.

Hence, for extracting the features from an image, we will use a Convolutional neural network and then process the vectorized image to get the result.

As we have to generate a caption, the idea is to generate it word by word. So for generating the next word, we will use all the previous words along with the vectorized image to predict the next word. This is the final idea.

6.2 Feature Extraction from the images

I used the VGG16 pre-trained model whose weights are adjusted for the ImageNet dataset. The reason for using a pre-trained model is that it has its weights already adjusted to the ImageNet dataset which has 1000 categories and a large number of images. This model encodes the images into vectors of dimension 512.

I also tried using InceptionV3 but it did not give me a better accuracy than VGG16.

6.3 Preprocessing the Text

For the preprocessing part, we add ‘<start>’ at the beginning and ‘<end>’ at the end of each sentence and then use the tokenizer in TensorFlow to encode each sentence and to find the vocabulary. The tokenizer also eliminates the punctuation marks and also converts all text to lowercase.

For example,

```
<start> jenny and mike are playing baseball together. <end>
```

6.4 The problem in Creating the Input Dataset for the model

The most challenging part while implementing the above idea is that we have to vectorize the image and then pass it to the model along with the sequence until now. This will consume a lot of memory as we have 1500 images and each image has 50 captions such that the maximum caption length is approximately 50 and also the vocabulary of the captions i.e., the number of unique words in the captions, is too large.

The vocabulary size matters because the output for the model will be one hot encoded and hence, the size of the vocabulary matters.

So, to generate an appropriate dataset for the model, we will have to repeat an image for the number of captions times the number of words in the caption which will approximately be 1500(number of images) x 50(number of captions per image) x 50(number of words per caption). Also, for all these inputs the true output will be one hot encoded and hence, the memory required to store that will also be huge.

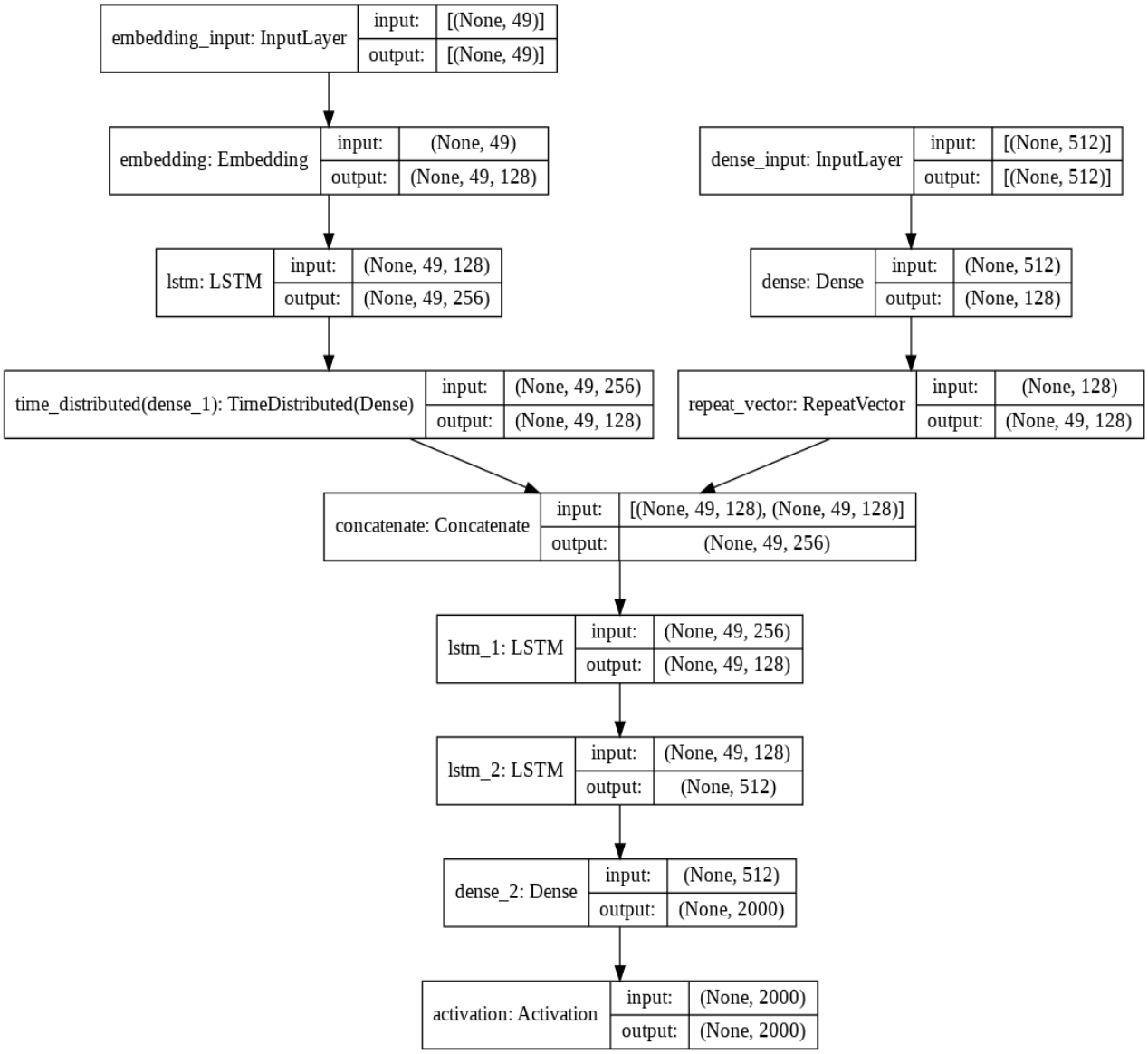
6.5 Generating the Dataset

Due to the memory issue, I created a generator to generate the dataset batch by batch. I also used indices of the image and then passed them to the generator. So only at the time of generation, the indices are mapped to the vectorized image. Inside the generator itself, the encoding of the sequence of words until now gets padded and the current word gets one-hot encoded.

6.6 Model Architecture

The model takes two inputs: the vectorized image and the encoded sequence of words seen until now. After concatenating the inputs, a sequence of LSTM layers is used to predict the next word. In the last layer “SoftMax” activation is used to predict the probabilities of all the words.

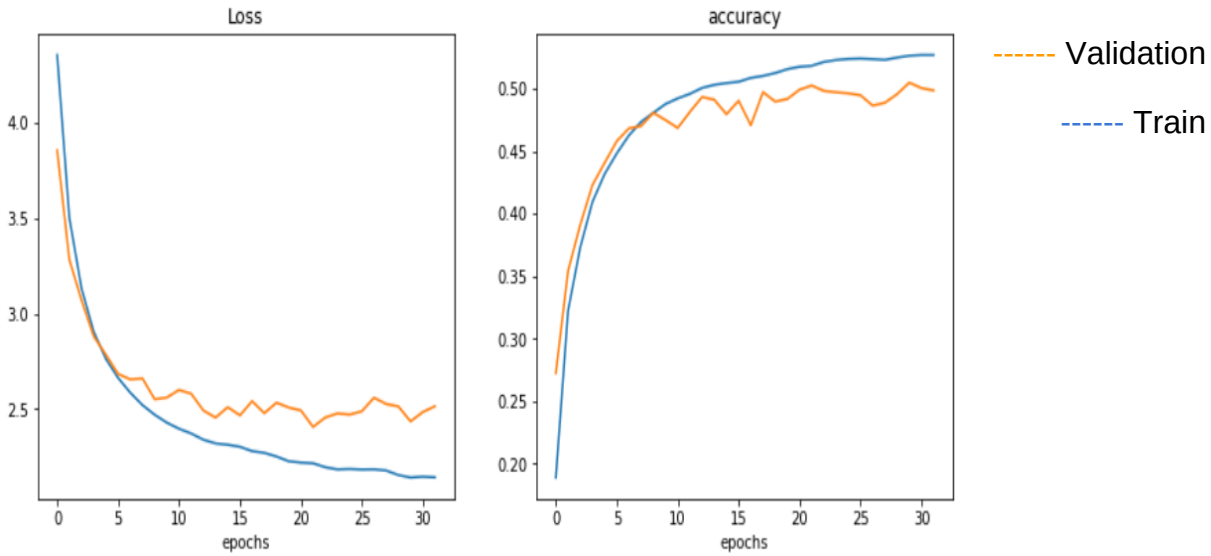
6.6.1 Model Plot



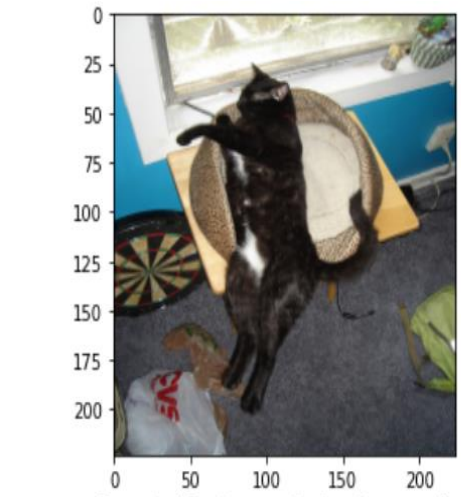
6.6.2 Model Summary

```
Model: "model"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
embedding_input (InputLayer) [(None, 49)]              0
_________________________________________________________________________________________________
dense_input (InputLayer)     [(None, 512)]             0
_________________________________________________________________________________________________
embedding (Embedding)        (None, 49, 128)           256000      embedding_input[0][0]
_________________________________________________________________________________________________
dense (Dense)                (None, 128)               65664       dense_input[0][0]
_________________________________________________________________________________________________
lstm (LSTM)                  (None, 49, 256)           394240      embedding[0][0]
_________________________________________________________________________________________________
repeat_vector (RepeatVector) (None, 49, 128)           0           dense[0][0]
_________________________________________________________________________________________________
time_distributed (TimeDistribut (None, 49, 128)        32896       lstm[0][0]
_________________________________________________________________________________________________
concatenate (Concatenate)    (None, 49, 256)           0           repeat_vector[0][0]
                                                                   time_distributed[0][0]
_________________________________________________________________________________________________
lstm_1 (LSTM)                (None, 49, 128)           197120      concatenate[0][0]
_________________________________________________________________________________________________
lstm_2 (LSTM)                (None, 512)               1312768     lstm_1[0][0]
_________________________________________________________________________________________________
dense_2 (Dense)              (None, 2000)              1026000     lstm_2[0][0]
_________________________________________________________________________________________________
activation (Activation)      (None, 2000)              0           dense_2[0][0]
=================================================================================================
Total params: 3,284,688
Trainable params: 3,284,688
Non-trainable params: 0
_________________________________________________________________________________________________
```

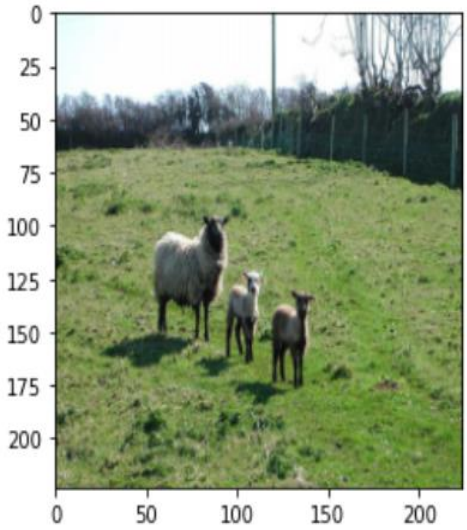
6.7 Training History



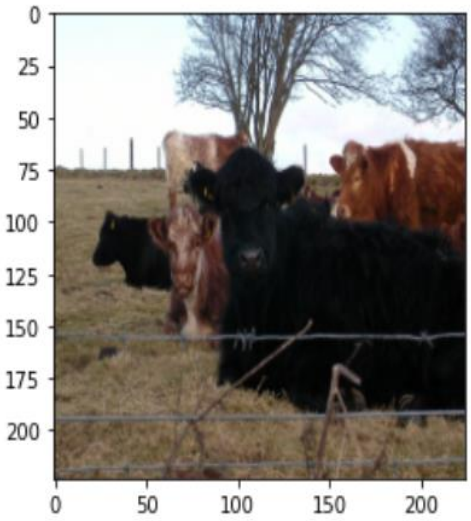
6.8 Predictions from the test data



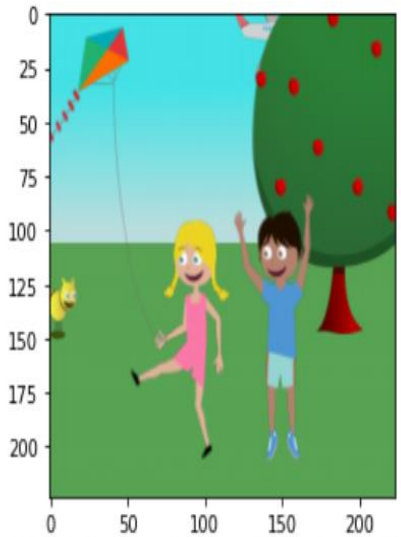
Generated Caption: a cat laying down on a bed
Original Caption:a black cat laying stretched out in bed by the window.



Generated Caption: a sheep and two lambs stand in a field
Original Caption:an adult sheep looks on with its young.



Generated Caption: a group of cows are laying in a pasture
Original Caption:some cows are sitting in the grass.



Generated Caption: jenny is flying a kite and mike is flying a kite
Original Caption:jenny is flying a kite while mike looks on with excitement.