MINI PROJECT REPORT

On

Video Captioning

Session 2020-21

Submitted by
Kaustubh Sisodia (181500318)
Aman Vikrant Garg (181500080)
Shivam Dubey(181500667)
Sudheer Kumar(181500731)

Department of Computer Engineering & Applications

Institute of Engineering & Technology

GLA University
Mathura- 281406, INDIA

Student Information

Name: Kaustubh Sisodia Aman Garg Shivam Dubey Sudheer Rana	University Roll. No.-181500318 181500080 181500667 181500731
Mobile:7455082308	Email: kaustubh.sisodia_cs18@gla.ac.in

Information about Industry/Organization:

Organization	GLA University ,Mathura
Contact Person	Mr. Vaibhav Diwan 9754435581

Project Information:

Title Of Project	Video Captioning
Role & Responsibility	
Technical Details	Hardware Requirements: Main Processor Core I3 Hard-disk Capacity 1 G.B RAM 2 GB Clock Speed 2.8 Hz Keyboard 104 Key Software Requirements: Operating System Window 10 Language Python 3.6.6 Jupyter Notebook
Training Implementation Details	Fully Implemented
Training Period	Start Date:07/02/2021 End Date:20/04/2021

Summary of the Project Work

The project entitled **Video Captioning** was completed successfully. The system has been developed with much care and free of errors and at the same time it is efficient and less time consuming. The purpose of this project was to study the **Convolution Neural Network** , **Recurrent Neural Network** and **Transfer Learning** with the help of that identify the caption of a Video .

This project helped us in gaining valuable information and practical knowledge on several topics like creating CNN models and Deep Learning using Python, Tensorflow & Keras, usage of functions of Keras module . The entire code is error free. Also the project helped us understanding about the development phases of a project and software development life cycle. We learned how to test different features of a project.

This project has given us great satisfaction in having designed an model which can be implemented to anyone by simple modifications. There is a scope for further development in our project to a great extent. A number of features can be added to this model in future like providing face detection and other functionality.

ACKNOWLEDGEMENT

The project work in this report is an outcome of continuous work over a period and drew intellectual support from various sources. I would like to articulate our profound gratitude and to all those people who extended their wholehearted co-operation and have helped me in completing this project successfully.

I am thankful to Mr. Sharad Gupta for teaching and assisting me in making the project successful. I would also like to thank our parents & other fellow mates for guiding and encouraging me throughout the duration of the project.

Kaustubh Sisodia (181500318)
Aman Vikrant Garg (181500080)
Shivam Dubey (181500667)
Sudheer Kumar (181500731)

Department of Computer Engineering and Applications

GLA University, Mathura

17 km Stone NH#2, Mathura-Delhi Road, P.O. Chaumuha

Mathura, 281406

DECLARATION

I hereby declare that the project work entitled "Video Captioning" submitted to the GLA University Mathura, is a record of an original work done by me under the guidance of Mr. Vaibhav Diwan.

Signature of Candidate:

Kaustubh Sisodia (181500318)
Aman Vikrant Garg (181500080)
Shivam Dubey(181500667)
Sudheer Kumar (181500731)
Course: Computer Science and Engineering
Year: III
Semester:6

ABSTRACT

Mini Project is the requirement for all engineering students in order to complete their Bachelor of Engineering degree at the GLA University, Mathura. Mini Project is a very important programme, since it complements both the academic and professional aspects of the engineering education . Exposing the students to the practical experience and actual working environment shall open the avenues for developing their skills and capabilities, as well as enhancing their intellectual and emotional persona. The Mini Project also can provide strong linkages between university-industries that shall pave opportunities for "smart partnerships" and industrially driven research. The outcomes of the EIT that are mainly based on the assessment covering the company's and university's evaluation will provide the feedback for student’s performance after 75% completion of their engineering study. The remarks from the companies on the students will very much helpful for the university to have a continuous quality improvement especially on curriculum practiced.

Table of Content

Introduction To Deep Learning

Deep learning is a branch of machine learning which is completely based on artificial neural networks, as neural network is going to mimic the human brain so deep learning is also a kind of mimic of human brain. In deep learning, we don't need to explicitly program everything. The concept of deep learning is not new. It has been around for a couple of years now. It's on hype nowadays because earlier we did not have that much processing power and a lot of data. As in the last 20 years, the processing power increases exponentially, deep learning and machine learning came in the picture.

A formal definition of deep learning is- neurons ,In human brain approximately 100 billion neurons all together this is a picture of an individual neuron and each neuron is connected through thousand of their neighbours.

The question here is how do we recreate these neurons in a computer. So, we create an artificial structure called an artificial neural net where we have nodes or neurons. We have some neurons for input value and some for output value and in between, there may be lots of neurons interconnected in the hidden layer.

Installation of Jupyter Notebook on window

Instructions tested with Windows 10 64-bit and Continuum's Anaconda 5.20

Install GNU on Windows

1.Download and install GNU on Windows (Gow) from https://github.com/bmatzelle/gow/releases/download/v0.8.0/Gow-0.8.0.exe. select The default options when prompted during the installation of Gow.

Install Anaconda and Jupyter Notebook

1.Downloads and install Anaconda from https://repo.anaconda.com/archive/Anaconda3-2019.07-Windows-x86_64.exe.

Select the default options when prompted during the installation of Anaconda

1. Open "Anaconda Propt" by finding it in the window (start) menu.
2. Type the command in red to verified Anaconda was installed.

```
> python --version
Python 3.7.3
```

3. Type the command in red to update Anaconda

```
> conda update --all --yes
```

Start Jupyter Notebook

1. Type the command in red to start jupyter Notebook

```
> jupyter notebook
```

System and tools

Operating System

The programming work was carried out on one computer which ran the window 10 system. The model was tested on two computers which ran window 10 and Ubuntu.

Python

Python is a widely used general-purpose, high level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

Python is a programming language that lets you work quickly and integrate systems more efficiently.

There are two major Python versions- Python 2 and Python 3. Both are quite different.

Beginning with Python programming:

1) Finding an Interpreter:

Before we start Python programming, we need to have an interpreter to interpret and run our programs. There are certain online interpreters like https://ide.geeksforgeeks.org/, http://ideone.com/ or http://codepad.org/ that can be used to start Python without installing an interpreter.

Windows:There are many interpreters available freely to run Python scripts like IDLE (Integrated Development Environment) which is installed when you install the python software from.

Jupyter Notebook

The Jupyter Notebook is an interactive computing environment that enables users to author notebook documents that include: - Live code - Interactive widgets - Plots - Narrative text - Equations - Images - Video

These documents provide a complete and self-contained record of a computation that can be converted to various formats and shared with others using email, Dropbox, version control systems (like git/GitHub) or nbviewer.jupyter.org.

Keras Module

What is Keras?

Keras is an Open Source Neural Network library written in Python that runs on top of Theano or Tensorflow. It is designed to be modular, fast and easy to use. It was developed by François Chollet, a Google engineer.

Keras doesn't handle low-level computation. Instead, it uses another library to do it, called the "Backend. So Keras is high-level API wrapper for the low-level API, capable of running on top of TensorFlow, CNTK, or Theano.

Keras High-Level API handles the way we make models, defining layers, or set up multiple input-output models. In this level, Keras also compiles our model with loss and optimizer functions, training process with fit function. Keras doesn't handle Low-Level API such as making the computational graph, making tensors or other variables because it has been handled by the "backend" engine.

What is Backend?

Backend is a term in Keras that performs all low-level computation such as tensor products, convolutions and many other things with the help of other libraries such as Tensorflow or Theano. So, the "backend engine" will perform the computation and development of the models. Tensorflow is the default "backend engine" but we can change it in the configuration.

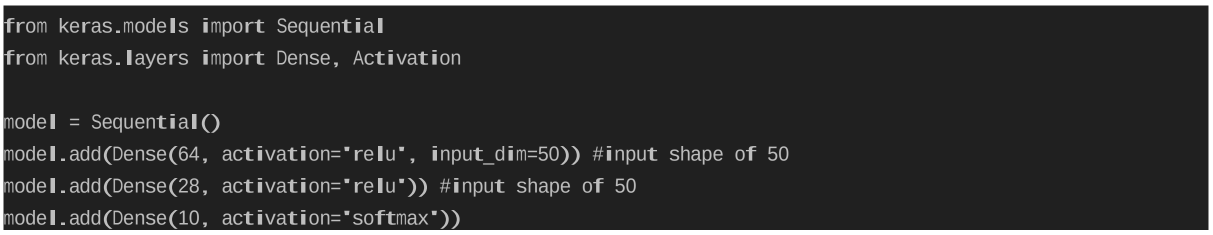
Theano is an open source project that was developed by the MILA group at the University of Montreal, Quebec, Canada. It was the first widely used Framework. It is a Python library that helps in multi-dimensional arrays for mathematical operations using Numpy or Scipy. Theano can use GPUs for faster computation, it also can automatically build symbolic graphs for computing gradients. On its website, Theano claims that it can recognize numerically unstable expressions and compute them with more stable algorithms, this is very useful for our unstable expressions.

On the other hand, Tensorflow is the rising star in deep learning framework. Developed by Google's Brain team it is the most popular deep learning tool. With a lot of features, and researchers contribute to help develop this framework for deep learning purposes.

Advantages of Keras

Fast Deployment and Easy to understand

Keras is very quick to make a network model. If you want to make a simple network model with a few lines, Keras can help you with that. Look at the example below:



```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential()
model.add(Dense(64, activation="relu", input_dim=50)) #input shape of 50
model.add(Dense(28, activation="relu")) #input shape of 50
model.add(Dense(10, activation="softmax"))
```

Because of friendly the API, we can easily understand the process. Writing the code with a simple function and no need to set multiple parameters.

Large Community Support:

There are lots of AI communities that use Keras for their Deep Learning framework. Many of them publish their codes as well tutorial to the general public.

Have multiple Backends

You can choose Tensorflow, CNTK, and Theano as your backend with Keras. You can choose a different backend for different projects depending on your needs. Each backend has its own unique advantage.

Cross-Platform and Easy Model Deployment

With a variety of supported devices and platforms, you can deploy Keras on any device like

iOS with CoreML

Android with Tensorflow Android,

Web browser with .js support

Cloud engine

Raspberry Pi

Multi GPUs Support

You can train Keras with on a single GPU or use multiple GPUs at once. Because Keras has a built-in support for data parallelism so it can process large volumes of data and speed up the time needed to train it.

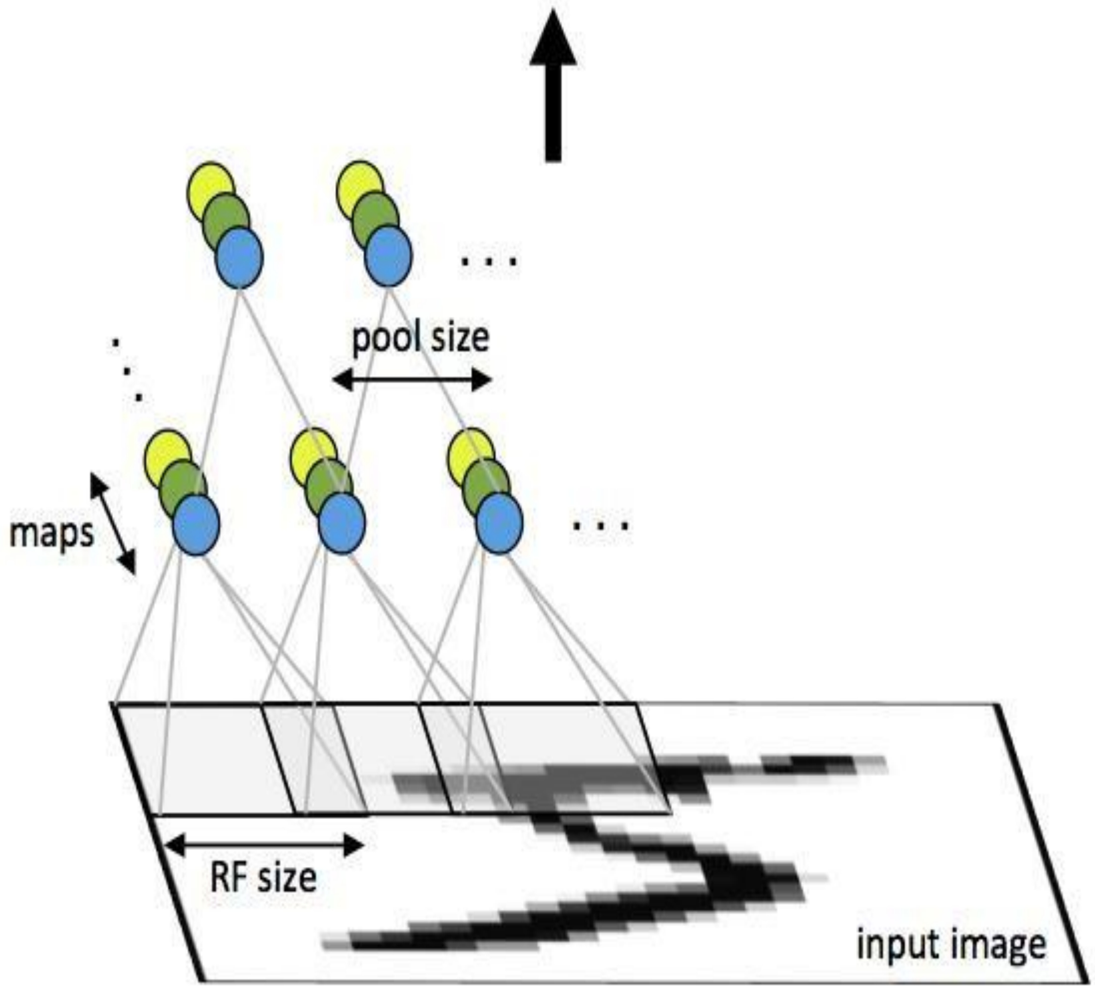
Introduction to Convolutional neural network

Overview

A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network. The architecture of a CNN is designed to take advantage of the 2D structure of an input image (or other 2D input such as a speech signal). This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNNs is that they are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units. In this article we will discuss the architecture of a CNN and the back propagation algorithm to compute the gradient with respect to the parameters of the model in order to use gradient based optimization. See the respective tutorials on convolution and pooling for more details on those specific operations.

Architecture

A CNN consists of a number of convolutional and subsampling layers optionally followed by fully connected layers. The input to a convolutional layer is a m x m x r image where m is the height and width of the image and r is the number of channels, e.g. an RGB image has r=3. The convolutional layer will have k filters (or kernels) of size n x n x q where n is smaller than the dimension of the image and q can either be the same as the number of channels r or smaller and may vary for each kernel. The size of the filters gives rise to the locally connected structure which are each convolved with the image to produce k feature maps of size m−n+1. Each map is then subsampled typically with mean or max pooling over p x p contiguous regions where p ranges between 2 for small images (e.g. MNIST) and is usually not more than 5 for larger inputs. Either before or after the subsampling layer an additive bias and sigmoidal nonlinearity is applied to each feature map. The figure below illustrates a full layer in a CNN consisting of convolutional and subsampling sublayers. Units of the same color have tied weights.

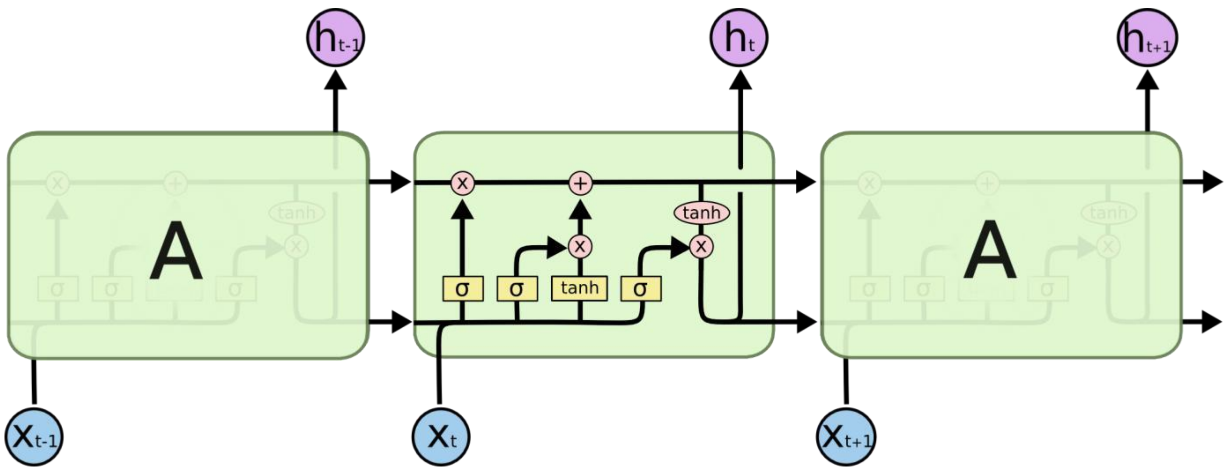


First layer of a convolutional neural network with pooling. Units of the same color have tied weights and units of different color represent different filter maps.

After the convolutional layers there may be any number of fully connected layers. The densely connected layers are identical to the layers in a standard multilayer neural network.

Introduction to Long Short Term Memory

Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work.1 They work tremendously well on a large variety of problems, and are now widely used. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!
All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



The Core Idea Behind LSTMs

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It’s very easy for information to just flow along it unchanged.The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through!”An LSTM has three of these gates, to protect and control the cell state.

Step-by-Step LSTM Walk Through

The first step in our LSTM is to decide what information we’re going to throw away from the cell state. This decision is made by a sigmoid layer called the “forget gate layer.” It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. A 1 represents “completely keep this” while a 0 represents “completely get rid of this.”

Let’s go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.

The next step is to decide what new information we’re going to store in the cell state. This has two parts. First, a sigmoid layer called the “input gate layer” decides which values we’ll update. Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state. In the next step, we’ll combine these two to create an update to the state.

In the example of our language model, we’d want to add the gender of the new subject to the cell state, to replace the old one we’re forgetting.

It’s now time to update the old cell state, C_{t-1}, into the new cell state C_t. The previous steps already decided what to do, we just need to actually do it.

We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t*\tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.

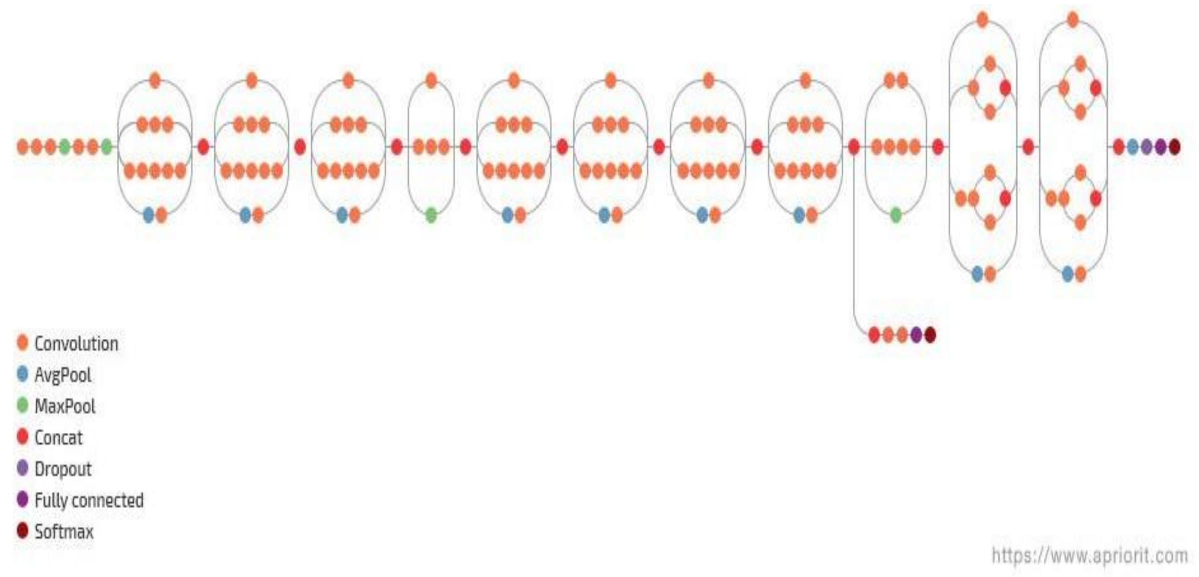
In the case of the language model, this is where we’d actually drop the information about the old subject’s gender and add the new information, as we decided in the previous steps.

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through \(\tanh\) (to push the values to be between \(-1\) and \(1\)) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.

InceptionV3 Model Structure

The Inception network was an important milestone in the development of CNN classifiers. Prior to its inception (pun intended), most popular CNNs just stacked convolution layers deeper and deeper, hoping to get better performance.



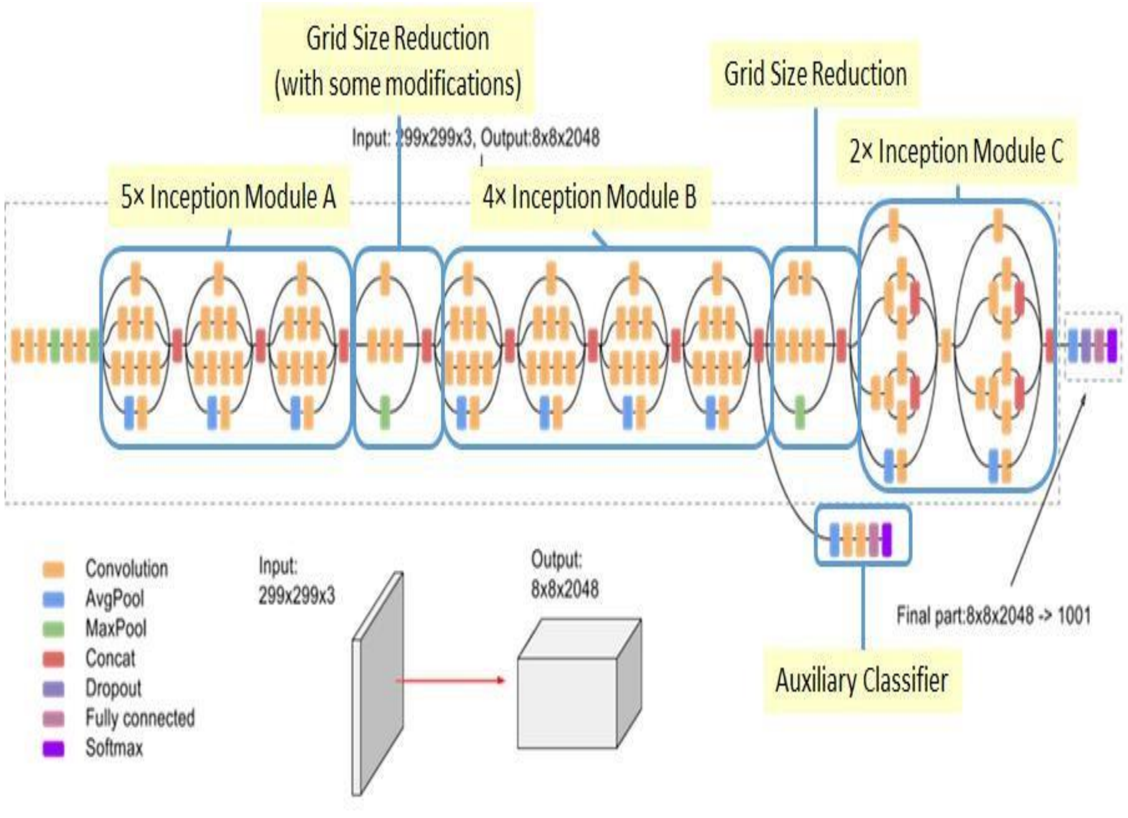
This section focuses on experimental setup for mammals' classification model using Inception-v3 on Tensor Flow
framework. Here classification model is separated into following four stages: image preprocessing, training,
verification followed by testing.

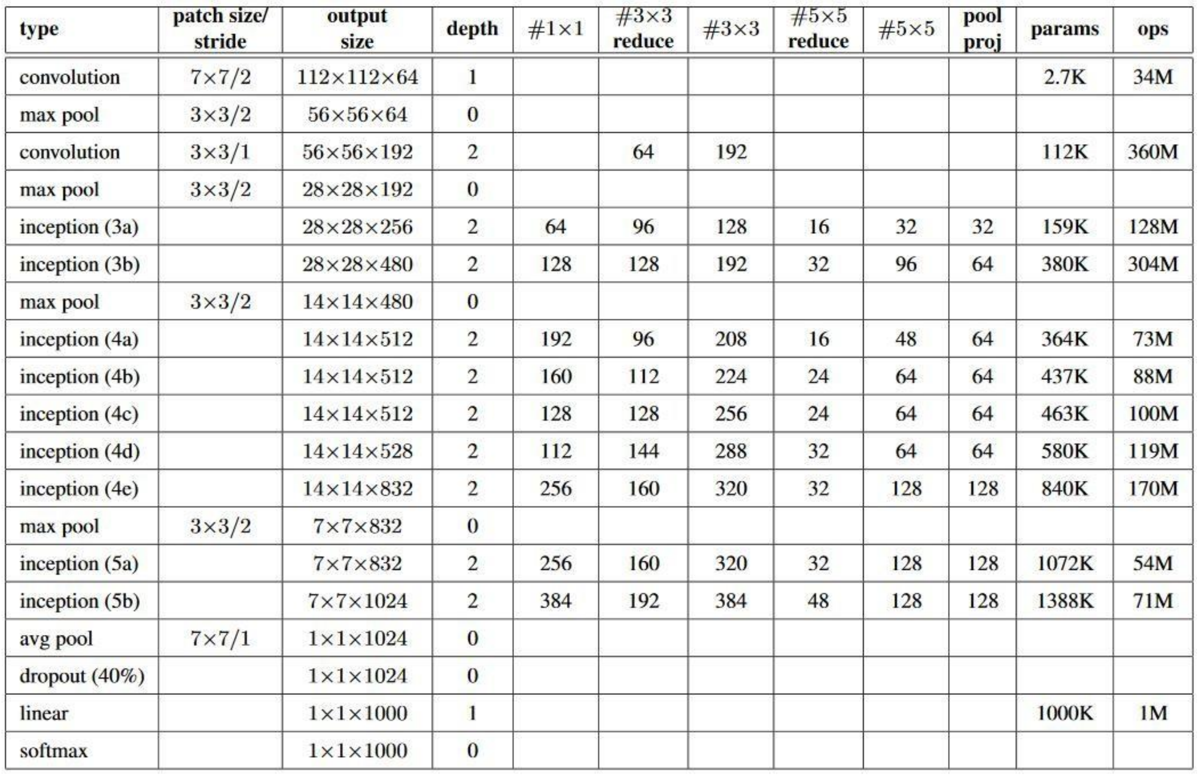
A. Image Preprocessing

Image pre-processing is the name for operations on images at the lowest level of abstraction whose aim is an
improvement of the image data that suppress undesired distortions or enhances some image features important for
further processing. It does not increase image information content. Its methods use the considerable redundancy in
images. Neighbouring pixels corresponding to one object in real images have the same or similar brightness value and
if a distorted pixel can be picked out from the image, it can be restored as an average value of neighbouring pixels.

B. Training Process

While training the model we use approximately 2000 image dataset, around 400 images per mammal, every image is
used multiple times through training process. Computing the layers behind the layer just before the final output layer
which performs the grouping for each image takes a substantial time. As the lower layers of the network are not being
changed their outputs can be stored and used again.
C. Verification and testing process
By testing, we mean evaluating the system in several conditions and observing its behavior, as stated above we are not
just providing single image as input to the inception model instead multiple images multiple times watching for defects.
By verification, we mean producing a compelling argument that the system will not misbehave under a very broad
range of circumstances so the accuracy of model will not be varied.

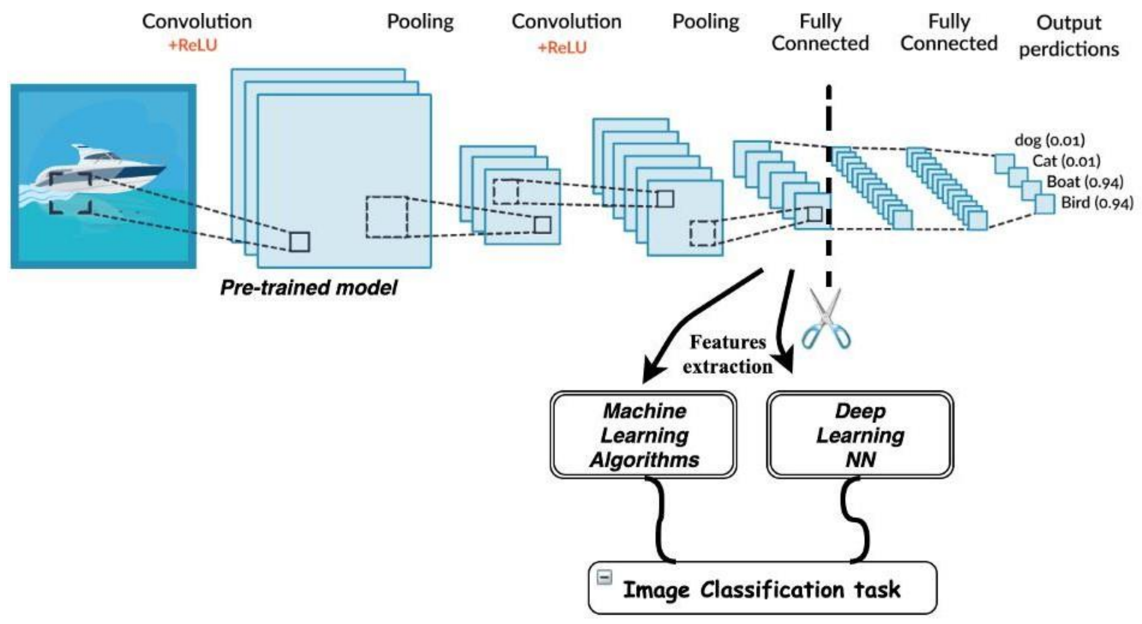




type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Brief Description Of Transfer Learning

The first thing to remember here is that, transfer learning, is not a new concept which is very specific to deep learning. There is a stark difference between the traditional approach of building and training machine learning models, and using a methodology following transfer learning principles.



Traditional learning is isolated and occurs purely based on specific tasks, datasets and training separate isolated models on them. No knowledge is retained which can be transferred from one model to another. In transfer learning, you can leverage knowledge (features, weights etc) from previously trained models for training newer models and even tackle problems like having less data for the newer task!

Let’s understand the preceding explanation with the help of an example. Let’s assume our task is to identify objects in images within a restricted domain of a restaurant. Let’s mark this task in its defined scope as T1. Given the dataset for this task, we train a model and tune it to perform well (generalize) on unseen data points from the same domain (restaurant). Traditional supervised ML algorithms break down when we do not have sufficient training examples for the required tasks in given domains. Suppose, we now must detect objects from images in a park or a café (say, task T2). Ideally, we should be able to apply the model trained for T1, but in reality, we face performance degradation and models that do not generalize well. This happens for a variety of reasons, which we can liberally and collectively term as the model’s bias towards training data and domain.

Transfer learning should enable us to utilize knowledge from previously learned tasks and apply them to newer, related ones. If we have significantly more data for task T1, we may utilize its learning, and generalize this knowledge (features, weights) for task T2 (which has significantly less data). In the case of problems in the computer vision domain, certain low-level features, such as edges, shapes, corners and intensity, can be shared across tasks, and thus enable knowledge transfer among tasks! Also, as we have depicted in the earlier figure, knowledge from an existing task acts as an additional input when learning a new target task.

Project Description of Video Captioning

```
import os
import string
import glob
from tensorflow.keras.applications import MobileNet
import tensorflow.keras.applications.mobilenet

from tensorflow.keras.applications.inception_v3 import InceptionV3
import tensorflow.keras.applications.inception_v3



from tqdm import tqdm
import tensorflow.keras.preprocessing.image
import pickle
from time import time
import numpy as np
from PIL import Image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Embedding, TimeDistributed, Dense, RepeatVector,\
                    Activation, Flatten, Reshape, concatenate, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras import Input, layers
from tensorflow.keras import optimizers

from tensorflow.keras.models import Model

from tensorflow.keras.layers import add
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
```

```
START = "startseq"
STOP = "endseq"
EPOCHS = 10
USE_INCEPTION = True

def hms_string(sec_elapsed):
    h = int(sec_elapsed / (60 * 60))
    m = int((sec_elapsed % (60 * 60)) / 60)
    s = sec_elapsed % 60
    return f"{h}:{m:>02}:{s:>05.2f}"

null_punct = str.maketrans('', '', string.punctuation)
lookup = dict()

with open( os.path.join(root_captioning,'Flickr8k_text','Flickr8k.token.txt'), 'r') as fp:
    max_length = 0
    for line in fp.read().split('\n'):
        tok = line.split()
        if len(line) >= 2:
            id = tok[0].split('.')[0]
            desc = tok[1:]
      # Cleanup description
            desc = [word.lower() for word in desc]
            desc = [w.translate(null_punct) for w in desc]
            desc = [word for word in desc if len(word)>1]
            desc = [word for word in desc if word.isalpha()]
            max_length = max(max_length,len(desc))
            if id not in lookup:
                lookup[id] = list()
            lookup[id].append(' '.join(desc))
lex = set()
for key in lookup:
    [lex.update(d.split()) for d in lookup[key]]
```

```
print(len(lookup)) # How many unique words
print(len(lex)) # The dictionary
print(max_length) # Maximum length of a caption (in words)

img = glob.glob(os.path.join(root_captioning,'Flicker8k_Dataset', '*.jpg'))

train_descriptions = {k:v for k,v in lookup.items() if f'{k}.jpg' in train_images}
for n,v in train_descriptions.items():
    for d in range(len(v)):
        v[d] = f'{START} {v[d]} {STOP}'

if USE_INCEPTION:
    encode_model = InceptionV3(weights='imagenet')
    encode_model = Model(encode_model.input, encode_model.layers[-2].output)
    WIDTH = 299
    HEIGHT = 299
    OUTPUT_DIM = 2048
    preprocess_input = tensorflow.keras.applications.inception_v3.preprocess_input
else:
    encode_model = MobileNet(weights='imagenet',include_top=False)
    WIDTH = 224
    HEIGHT = 224
    OUTPUT_DIM = 50176
    preprocess_input = tensorflow.keras.applications.mobilenet.preprocess_input



    encode_model.save_weights("image_caption_model.h5")
    print("Saved model to disk")
```

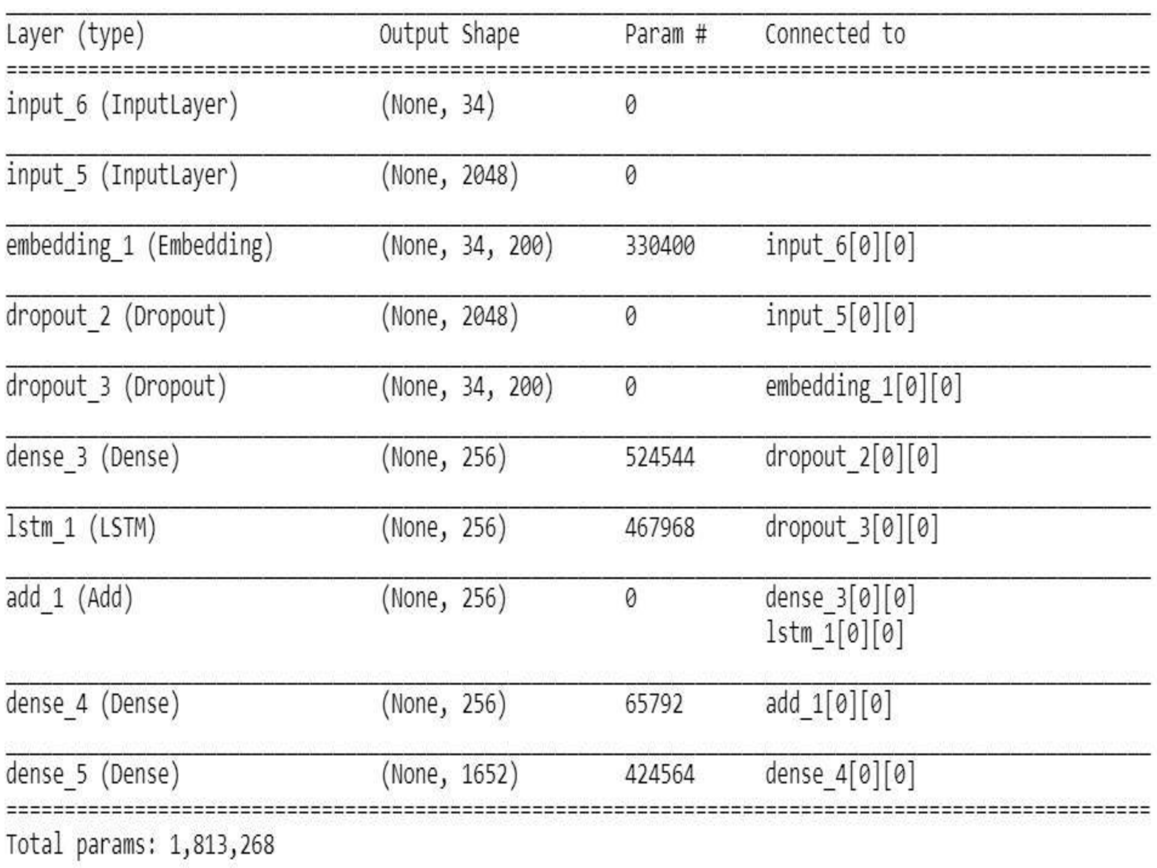
```
 def encodeImage(img):
   # Resize all images to a standard size (specified bythe image encoding network)
   img = img.resize((WIDTH, HEIGHT), Image.ANTIALIAS)
 # Convert a PIL image to a numpy array
   x = tensorflow.keras.preprocessing.image.img_to_array(img)
 # Expand to 2D array
   x = np.expand_dims(x, axis=0)
 # Perform any preprocessing needed by InceptionV3 or others
   x = preprocess_input(x)
 # Call InceptionV3 (or other) to extract the smaller feature set for the image.
   x = encode_model.predict(x) # Get the encoding vector for the image
 # Shape to correct form to be accepted by LSTM captioning network.
   x = np.reshape(x, OUTPUT_DIM )
   return x
train_path = os.path.join(root_captioning,"data",f'train{OUTPUT_DIM}.pkl')
if not os.path.exists(train_path):
   start = time()
   encoding_train = {}
   for id in tqdm(train_img):
      image_path = os.path.join(root_captioning,'Flicker8k_Dataset', id)
      img = tensorflow.keras.preprocessing.image.load_img(image_path,
target_size=(HEIGHT, WIDTH))
      encoding_train[id] = encodeImage(img)
   with open(train_path, "wb") as fp:
      pickle.dump(encoding_train, fp)
   print(f"\nGenerating training set took: {hms_string(time()-start)}")
else:
   with open(train_path, "rb") as fp:
      encoding_train = pickle.load(fp)
```

```
test_path = os.path.join(root_captioning,"data",f'test{OUTPUT_DIM}.pkl')
if not os.path.exists(test_path):
    start = time()
    encoding_test = {}
    for id in tqdm(test_img):
        image_path = os.path.join(root_captioning,'Flicker8k_Dataset', id)
        img = tensorflow.keras.preprocessing.image.load_img(image_path,
target_size=(HEIGHT, WIDTH))
        encoding_test[id] = encodeImage(img)
    with open(test_path, "wb") as fp:
        pickle.dump(encoding_test, fp)
    print(f"\nGenerating testing set took: {hms_string(time()-start)}")
else:
    with open(test_path, "rb") as fp:
        encoding_test = pickle.load(fp)
all_train_captions = []
for key, val in train_descriptions.items():
    for cap in val:
        all_train_captions.append(cap)
len(all_train_captions)

word_count_threshold = 10
word_counts = {}
nsents = 0
for sent in all_train_captions:
    nsents += 1
    for w in sent.split(' '):
        word_counts[w] = word_counts.get(w, 0) + 1

vocab = [w for w in word_counts if word_counts[w] >= word_count_threshold]
print('preprocessed words %d ==> %d' % (len(word_counts), len(vocab)))
```

```
def data_generator(descriptions, photos, wordtoidx, max_length, num_photos_per_batch):
    # x1 - Training data for photos
  # x2 - The caption that goes with each photo
  # y - The predicted rest of the caption
    x1, x2, y = [], [], []
    n=0
    while True:
        for key, desc_list in descriptions.items():
            n+=1
            photo = photos[key+'.jpg']
      # Each photo has 5 descriptions
            for desc in desc_list:
            # Convert each word into a list of sequences.
                seq = [wordtoidx[word] for word in desc.split(' ') if word in wordtoidx]
        # Generate a training case for every possible sequence and outcome
                for i in range(1, len(seq)):
                    in_seq, out_seq = seq[:i], seq[i]
                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                    out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                    x1.append(photo)
                    x2.append(in_seq)
                    y.append(out_seq)
            if n==num_photos_per_batch:
                yield ([np.array(x1), np.array(x2)], np.array(y))
                x1, x2, y = [], [], []
                n=0
```

```
glove_dir = os.path.join(root_captioning,'glove.6B')
embeddings_index = {}
f = open(os.path.join(glove_dir, 'glove.6B.200d.txt'), encoding="utf-8")

for line in tqdm(f):
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs

f.close()
print(f'Found {len(embeddings_index)} word vectors.')



embedding_dim = 200

# Get 200-dim dense vector for each of the 10000 words in out vocabulary
embedding_matrix = np.zeros((vocab_size, embedding_dim))

for word, i in wordtoidx.items():
    #if i < max_words:
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # Words not found in the embedding index will be all zeros
        embedding_matrix[i] = embedding_vecto
```

```
inputs1 = Input(shape=(OUTPUT_DIM,))
fe1 = Dropout(0.5)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, embedding_dim, mask_zero=True)(inputs2)
se2 = Dropout(0.5)(se1)
se3 = LSTM(256)(se2)
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)
caption_model = Model(inputs=[inputs1, inputs2], outputs=outputs)
```



```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_6 (InputLayer)             (None, 34)            0
____________________________________________________________________________________________________
input_5 (InputLayer)             (None, 2048)          0
____________________________________________________________________________________________________
embedding_1 (Embedding)          (None, 34, 200)       330400      input_6[0][0]
____________________________________________________________________________________________________
dropout_2 (Dropout)              (None, 2048)          0           input_5[0][0]
____________________________________________________________________________________________________
dropout_3 (Dropout)              (None, 34, 200)       0           embedding_1[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 256)           524544      dropout_2[0][0]
____________________________________________________________________________________________________
lstm_1 (LSTM)                    (None, 256)           467968      dropout_3[0][0]
____________________________________________________________________________________________________
add_1 (Add)                      (None, 256)           0           dense_3[0][0]
                                                                   lstm_1[0][0]
____________________________________________________________________________________________________
dense_4 (Dense)                  (None, 256)           65792       add_1[0][0]
____________________________________________________________________________________________________
dense_5 (Dense)                  (None, 1652)          424564      dense_4[0][0]
====================================================================================================
Total params: 1,813,268
```

```
caption_model.layers[2].set_weights([embedding_matrix])
caption_model.layers[2].trainable = False
caption_model.compile(loss='categorical_crossentropy', optimizer='adam')

EPOCHS = 3
model_path = os.path.join(root_captioning,"data",f'caption-model.hdf5')
if not os.path.exists(model_path):
    for i in tqdm(range(EPOCHS*2)):
        generator = data_generator(train_descriptions, encoding_train, wordtoidx,
max_length, number_pics_per_bath)
        caption_model.fit_generator(generator, epochs=1, steps_per_epoch=steps,
verbose=1)

    caption_model.optimizer.lr = 1e-4
    number_pics_per_bath = 6
    steps = len(train_descriptions)//number_pics_per_bath

    for i in range(EPOCHS):
        generator = data_generator(train_descriptions, encoding_train, wordtoidx,
max_length, number_pics_per_bath)
        caption_model.fit_generator(generator, epochs=1, steps_per_epoch=steps,
verbose=1)
    caption_model.save_weights(model_path)
    #print(f"\Training took: {hms_string(time()-start)}")
else:
    caption_model.load_weights(model_path)
```

Git Hub Project Link-
https://github.com/naman-echo/Mini_project2_Video_captioning

Conclusion

Difference between results (say accuracy score) of test dataset and training dataset is 0.1%
As we see, the difference between scores of test dataset and training dataset is very low (0.1%), hence, the conclusion is that our final model (XGBoost) is good enough, and it doesn't have any overfitting or underfitting.
Although XGBoost (with n_estimators=20 and max_depth = 10) is good enough, there may be a chance to improve this model further, by say, increasing the number of estimators and trying out some more hyperparameters.
As we see above, Ensemble also has given good results, we can try Ensemble with some more models and with some more hyperparameters to improve the results further.
This experiment was limited to Machine Learning algorithms. You can try Deep Learning techniques (say CNN, etc.) to improve the results further.

References

https://github.com/zalandoresearch/fashion-mnist#why-we-made-fashion-mnist

https://www.pyimagesearch.com/2019/02/11/fashion-mnist-with-keras-and-deep-learning/

https://research.zalando.com/welcome/mission/research-projects/fashion-mnist/