CAPSTONE PROJECT REPORT

Automatic Image Captioning

Team 14 – Subhaprada Gollakota, Harish P, Narendar Paladi

Mentor – Brahmani Nutakki

February 2023

Contents

INTRODUCTION

Automatic Image Captioning is the process of generating a textual description of an image automatically. The goal of this task is to translate an image into a natural language description, capturing the essence of the image.

In this report, we will discuss an approach to automatic image captioning using an image set, a pre-trained CNN model, and an LSTM (Long Short-Term Memory) network.

LITERATURE REVIEW

The research on automatic image captioning started around 2014 with the publication of [1]. This paper introduced a new model called "Deep Visual-Semantic Alignments" (DVSA) that could generate captions for images using a deep neural network.

After the publication of this paper, several other research works emerged. One prominent research paper in this area is in [2] - The proposed model consists of two main components: a deep convolutional neural network (CNN) for extracting visual features from the input image, and a long short-term memory (LSTM) network for generating a sequence of words that form the image description. This paper introduced a novel approach to aligning image regions with corresponding words in the caption. The paper proposes a neural network model for generating descriptions of images, which involves learning deep visual-semantic alignments between images and their corresponding descriptions.

[3] proposes an end-to-end neural network model for generating natural language captions for images, where the model takes an image as input and produces a sentence that describes the image. The proposed model also uses CNN and LSTM for generating the image caption. The key differentiator is the use of an attention mechanism, which allows the model to selectively focus on different parts of the image when generating the caption. Specifically, the attention mechanism weights the features extracted by the CNN based on their relevance to the words being generated by the LSTM network. This allows the model to attend to different parts of the image at different time steps, enabling it to generate more informative and accurate captions.

BASIC ARCHITECTURE

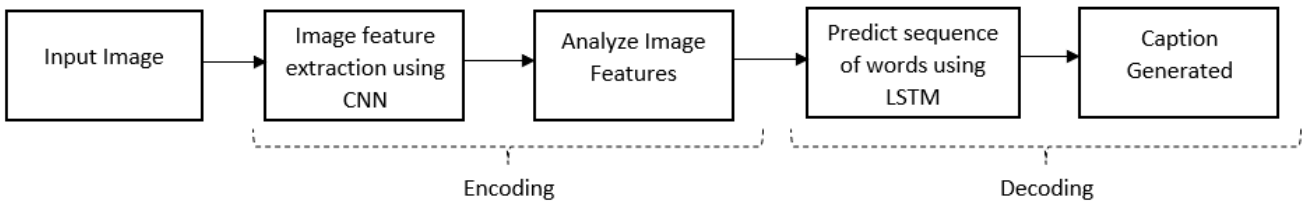


Figure. 1 Basic Architecture of Auto-Image captioning

The image is passed through the pre-trained model, and a feature vector is extracted from the final layer of the network. The feature vector is then passed to an LSTM network, which generates a caption by predicting the next word in the sequence given the previous words. Once the model is trained, it can be used to generate captions for new images.

PRE-TRAINED MODEL CONSIDERATIONS

One important aspect of automatic image captioning is the choice of image feature extractor, which is typically a pretrained deep neural network that is used to extract relevant features from images. Two of the most commonly used image feature extractors for automatic image captioning are ResNet50[4] and VGG19[5].

VGG19: VGG19 is a 19-layer convolutional neural network that was trained on the ImageNet dataset. It is known for its ability to extract fine-grained features from images, which can be useful for image captioning tasks that require a high degree of detail in the captions. However, VGG19 is computationally intensive and may not be suitable for real-time applications.

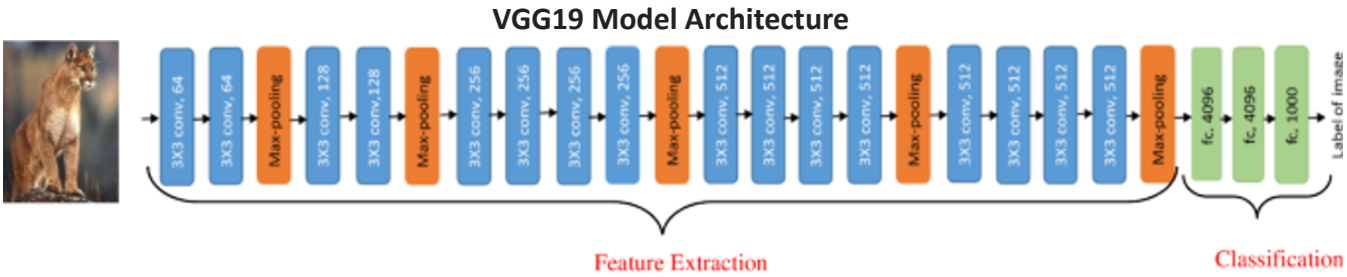


Figure. 2 VGG19 Model Architecture

Resnet50: ResNet50 is a deep residual network that was trained on the ImageNet dataset and is known for its ability to achieve high accuracy on image classification tasks. It has been shown to perform well on image captioning tasks due to its ability to extract high-level features from images. One advantage of using ResNet50 is that it is computationally efficient, making it well-suited for real-time applications.

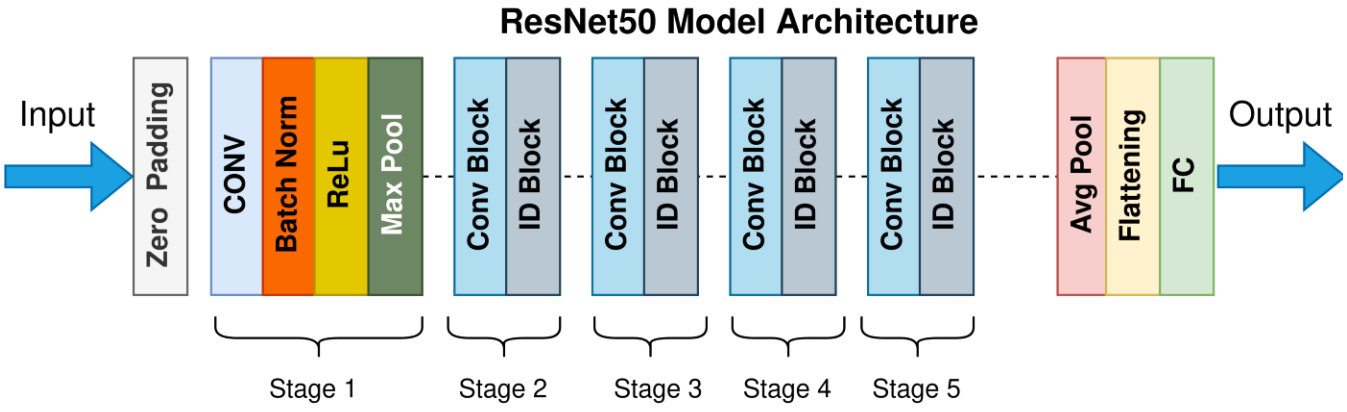


Figure. 3 ResNet50 Model Architecture

ResNet50 is well-suited for real-time applications and can generate captions that are semantically and syntactically correct, while VGG19 is better suited for applications that require a high degree of detail in the captions.

DATASET

Flickr30K: The Flickr30K image set is a collection of 30,000 images, each annotated with five captions describing the scene depicted in the image. The captions in the Flickr30K dataset are written in natural language and provide a rich source of textual information that can be leveraged to train a machine learning model for image captioning.
Flickr8K: Flickr8k_Dataset contains a total of 8092 images in JPEG format with different shapes and sizes. Flickr8k_text contains text files describing the images. Flickr8k.token.txt contains 5 captions for each image.

For our capstone project, we have generated two models – one with Flickr8K dataset and VGG19 pretrained model and the other with Flickr30K dataset and Resnet50 pretrained model. For the 8k model, we had downloaded the dataset but that was extremely memory intensive. So, we used deeplake[6] instead for the 30K dataset. We have included our findings from both the models and their outputs.

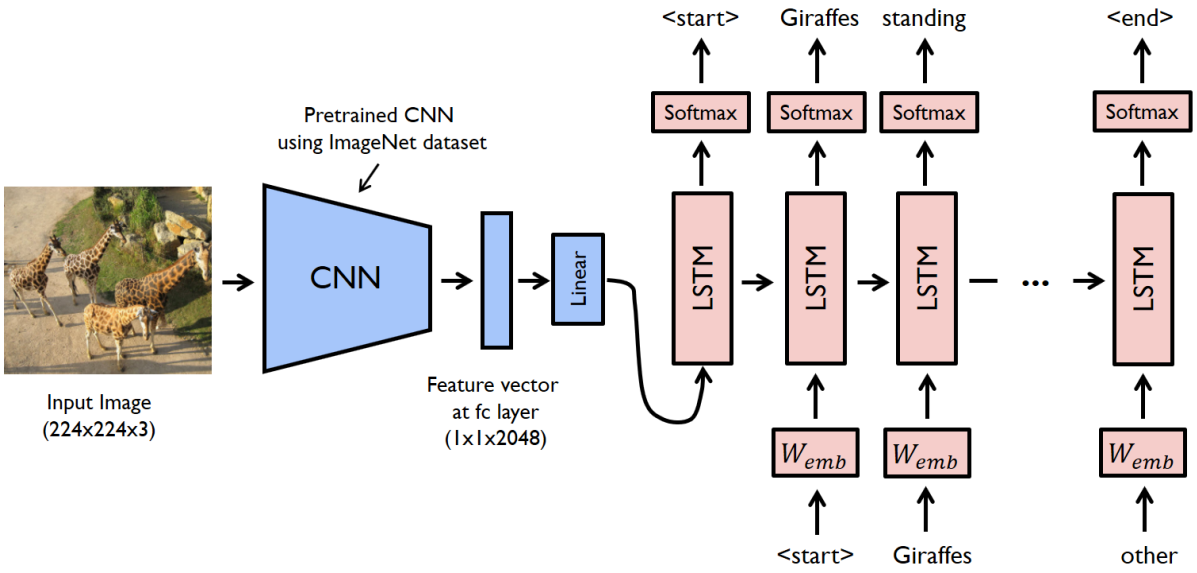


Figure. 4 Combined CNN-LSTM Model Architecture

FEATURE EXTRACTION

Feature extraction using pre-trained models works by using a pre-trained neural network to extract high level features from an input image. The pre-trained model has already been trained on a large dataset such as ImageNet and has learned to extract important features from images. The process of feature extraction works as follows:

1. The input image is passed through the pre-trained model which consists of several layers of convolutional, pooling and activation layers.
2. Each layer extracts a different set of features from the input image
3. The later layers extract higher-level features
4. The final layer produces a fixed length feature vector which represents the high-level features

IMAGE PRE-PROCESSING

Pre-processed images are passed through the pre-trained model to extract the features. Image pre-processing is an important step as it prepares the input image in the format expected by the pre-trained model. For example, the input image may have different sizes against what is expected. The pre-processing includes resizing the image to a consistent size, converting the image into a tensor and normalizing the pixel values of the image. Furthermore, by performing image preprocessing, we can ensure that the final model will result in more accurate feature extraction as well as reduce the effect of lighting conditions. Thus, ensuring to improve the overall performance of the Automatic image captioning system.

The VGG-19 model takes a color image as input, a 3-channel image is created by assigning (red, green, blue) channels, respectively. All regional images are cropped from the 3-channel image and scaled to 224×224×3. Keras application VGG19 has a method preprocess_input to take care of this.

The original ResNet50 model was trained with images of size 224 x 224 pixels and a number of preprocessing operations like the subtraction of the mean pixel value in the training set for all training images. We need to pre-process the images you want to predict in the same way. Keras application resnet50 has a method preprocess_input to take care of this. This reduction in size greatly enhances the performance as it has less data to work on.

LANGUAGE MODEL

In the field of automatic image captioning, the choice of language model is just as important as the choice of image feature extractor. Language models are used to generate captions based on the image features extracted by the image feature extractor. One commonly used language model for automatic image captioning is the Long Short-Term Memory (LSTM) model[7]. LSTMs are a type of Recurrent Neural Network (RNN) that are designed to capture long-term dependencies in sequential data. In the context of automatic image captioning, LSTMs can be trained to generate captions by encoding the image features and decoding them into captions.

CAPTIONS PREPROCESSING

Machines are not familiar with complex English words, so, to process model's data they need a simple numerical representation. Every word of the vocabulary must be mapped with a separate unique index value. An in-built tokenizer function is present in the Keras library to create tokens from our vocabulary. In order to pad the captions, we first determined the maximum length of a caption in the dataset. Then, we padded all other captions to that length. Each preprocessed caption is stored in a dictionary where the key is the image name and value is a dictionary. Since each image has multiple captions, the dictionary stores {image name, list of captions}. During preprocessing, we even created a vocabulary and returned it to the main function. The length of vocabulary will be used during building the model.

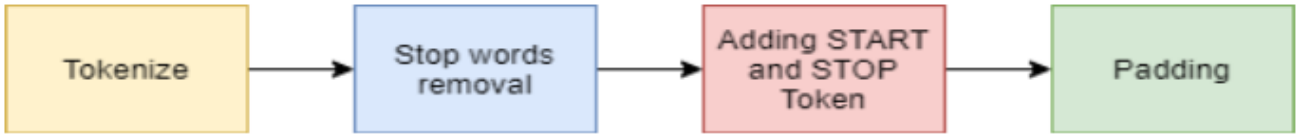


Figure. 5 Caption pre-processing

One-hot encoding does not capture the relationships between different words. Therefore, it does not convey information about the context.

Glove (Global Vectors for Word Representation) is another commonly used representation for captions. GloVe[8] is a pre-trained word embedding model that represents words as vectors in a high-dimensional space. Glove has been shown to perform well on a variety of NLP tasks, including automatic image captioning.

To start with GloVe, first we have to download the pre-trained model hosted at [9]. Unzip it and there are five text files representing the word and its corresponding vectors in various dimensions like 50 dimensions, 100 dimensions etc. We are using 50 dimensions here. We could try out other dimensions too and see how the performance would improve. To create the word embeddings, we need to create a dictionary holding each word and its respective vector.

Another thing we can do with GloVe vectors is find the most similar words to a given word using euclidean distance to measure how far apart the two words are. We could rank all words by closeness to a given word. The embedding matrix generated will serve as input to the decoder part of the model.

DATA GENERATOR

Training the model involves 30K or 8K images (based on the dataset we are using), feature vector and captions vector for each of the images, which is huge data. We will be using a generator method that will yield in batches of input and output sequences to avoid session crashes.

X1(feature vector)	X2(Text sequence)	y(word to predict)
feature	startseq,	two
feature	startseq, two	dogs
feature	startseq, two, dogs	playing
feature	startseq, two, dogs, playing	with
feature	startseq, two, dogs, playing, with	toy
feature	startseq, two, dogs, playing, with, toy	endseq

Figure. 6 Example of sequence generation

MODEL DEFINITION

- Feature Extractor – The feature extracted from the image has a size of 4096, with a dense layer, we will reduce the dimensions to 256 nodes.
- Sequence Processor – An embedding layer will handle the textual input, followed by the LSTM layer.
- Decoder – By merging the output from the above two layers, we will process by the dense layer to make the final prediction. The final layer will contain the number of nodes equal to our vocabulary size.

Visual representation of the final model is:

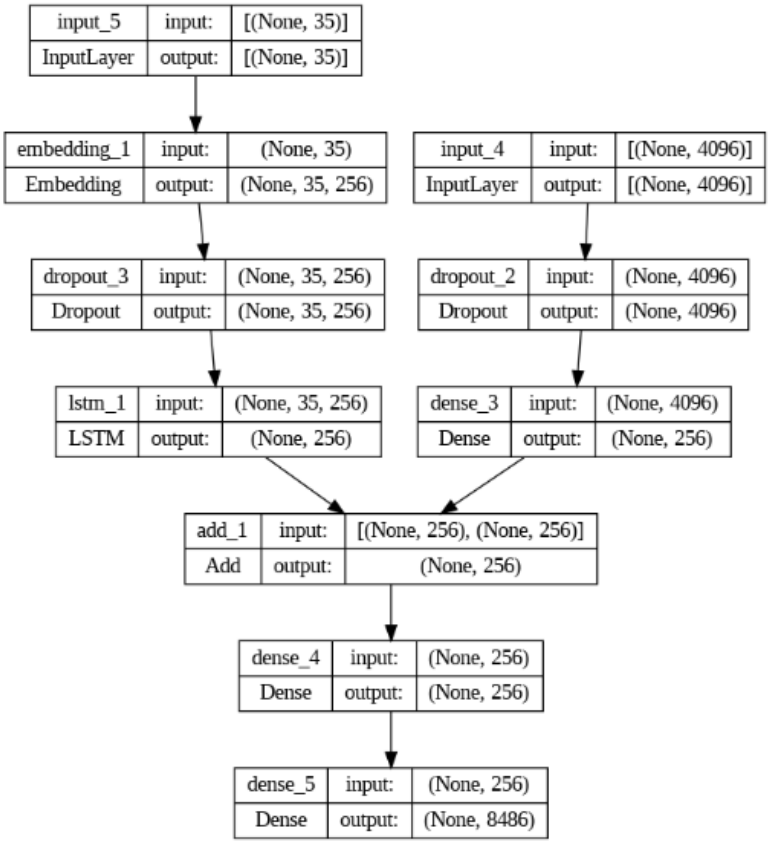


Figure.7 8K-VGG19-LSTM Model

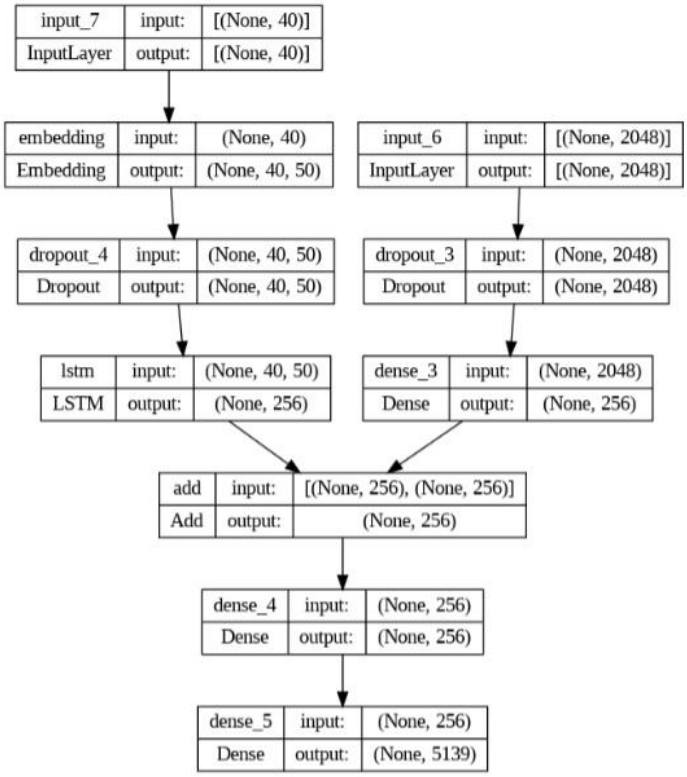


Figure.8 30K-ResNet50-LSTM Model

TRAINING THE MODEL

The train and test data have been split up in a 90:10 ratio. To train the model, we will be using the training images by generating the input and output sequences in batches and fitting them to the model using model.fit_generator. Training needs to be done over multiple epochs. Based on the dataset selected, splitting up the training over multiple epochs is not enough, based on the system capability. For training 30K data, we had to save the model after a few epochs and re-train it on another batch of data and repeat it over till the data is present.

TESTING THE MODEL

To test the model, we will load the saved model and generate predictions. The predictions contain the maximum length of index values so we will use the same pickle file to get the words from their index values.

TEST RESULTS

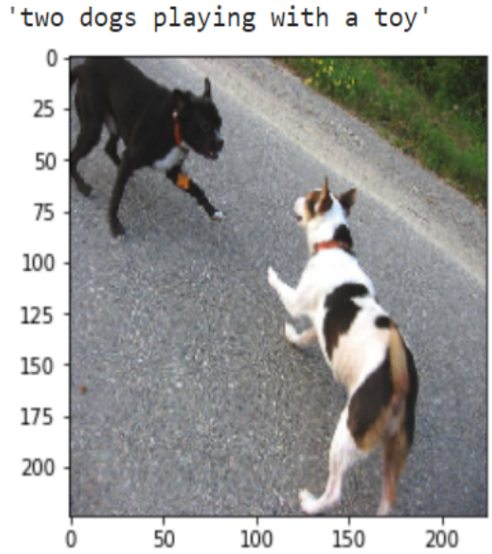


Figure.9 Test Results of 8k-VGG19 model

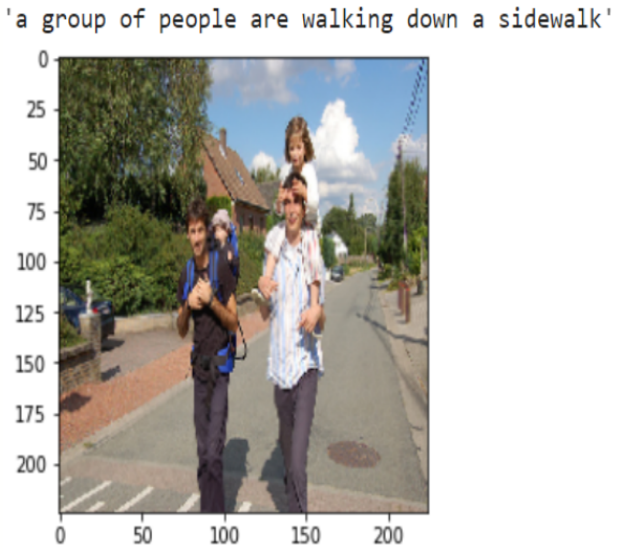


Figure.10 Test Results of 30K-ResNet50 model

BLEU SCORES

The BLEU (Bilingual Evaluation Understudy)[10] score is a commonly used evaluation metric for machine translation and image captioning tasks. It measures the degree of overlap between the generated captions and a set of reference captions. The reference captions are typically human-generated captions that describe the same image.

To calculate the Bleu score, we need to provide the reference and candidate sentences in the form of tokens.

Below are the snapshots of the bleu score generated for the 8k and 30k based models:

```
[54] from nltk.translate.bleu_score import corpus_bleu
     # validate with test data
     actual, predicted = list(), list()

     for key in tqdm(test):
         # get actual caption
         captions = mapping[key]
         # predict the caption for image
         y_pred = predict_caption(model, features[key], tokenizer, max_length)
         # split into words
         actual_captions = [caption.split() for caption in captions]
         y_pred = y_pred.split()
         # append to the list
         actual.append(actual_captions)
         predicted.append(y_pred)

     # calcuate BLEU score
     print("BLEU-1: %f" % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
     print("BLEU-2: %f" % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
```

```
100% 810/810 [08:51<00:00, 1.29it/s]
BLEU-1: 0.518395
BLEU-2: 0.297478
```

Figure.11 8K-VGG19-LSTM Model BLEU scores

```
for i in range(0,100):
    # get actual caption
    #captions = mapping[key]
    caption_0 = ds.caption_0[i].data()["value"]
    caption_1 = ds.caption_1[i].data()["value"]
    caption_2 = ds.caption_2[i].data()["value"]
    caption_3 = ds.caption_3[i].data()["value"]
    caption_4 = ds.caption_4[i].data()["value"]
    #print(img_name)
    captions = []
    captions.append(caption_0)
    captions.append(caption_1)
    captions.append(caption_2)
    captions.append(caption_3)
    captions.append(caption_4)

    # predict the caption for image
    #y_pred = predict_caption(model, features[key], tokenizer, max_length)
    y_pred=runModel_bluescore(i)
    # split into words
    actual_captions = [caption.split() for caption in captions]
    y_pred = y_pred.split()
    # append to the list
    actual.append(actual_captions)
    predicted.append(y_pred)

# calcuate BLEU score
print("BLEU-1: %f" % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
print("BLEU-2: %f" % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
```

```
BLEU-1: 0.553406
BLEU-2: 0.359980
```

Figure.12 30K-ResNet50-LSTM Model BLEU scores

Bleu Scores are between 0 and 1. A score of 0.6 or 0.7 is considered the best you can achieve. Even two humans would likely come up with different sentence variants for a problem, and would rarely achieve a perfect match.

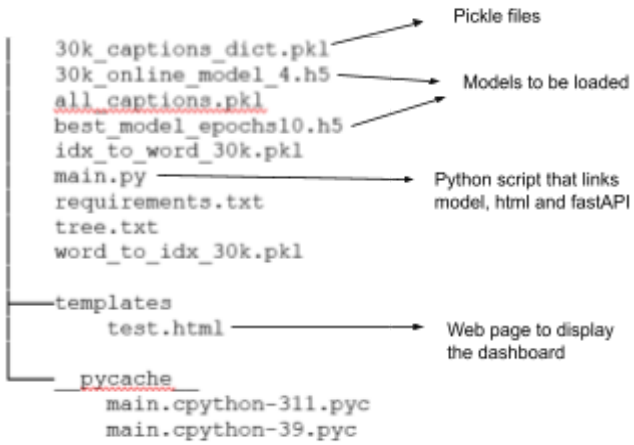
While the BLEU score is widely used as an evaluation metric for machine translation and image captioning tasks, it has some limitations. For example, it may not accurately capture the quality of the generated captions in terms of grammaticality, and semantic accuracy. A high BLEU score does not guarantee that the generated captions are semantically and grammatically correct, or that they accurately convey the meaning of the image. It can be worthwhile considering other evaluation metrics like ROUGE(Recall-Oriented Understudy for Gisting Evaluation)[11], METEOR(Metric for Evaluation of Translation with Explicit Ordering)[12] and CIDEr(Consensus-based Image Description Evaluation)[13]. Human evaluations are considered as the most accurate way to assess the quality of the generated captions.

DEPLOYMENT

The next step after developing a machine learning project is to make it deployable. Here, FastAPI is used to deploy Machine Learning models. FastApi is a Python web framework based on the OpenAPI Specification and JavaScript Object Notation Schema for developing RESTful APIs.- FastAPI allows web applications to be written efficiently in clean and modern Python code. It is easy to use as it also has a built-in web based interactive and documentation system that can help to test API endpoints and explore their functionality. It fully supports asynchronous programming and can be run with Uvicorn. Uvicorn is an ASGI (async server gateway interface) compatible web server. It's (simplified) the binding element that handles the web connections from the browser or api client and then allows FastAPI to serve the actual request

Figure.13 Uvicorn-FastAPI-Model Interaction Diagram

We start by creating the below folder structure after the basic installations.



Folder structure:

Figure.14 Deployment Folder structure

The requirements.txt file contains all the packages that need to be installed. Create a project folder in the working directory. Install the requirements by using following command:

```
>pip install -r requirements.txt
```

Most of the action happens in the main.py. To start with, import all the required libraries.

```
from fastapi import FastAPI, File, UploadFile
from tensorflow.keras.applications import vgg19
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.applications.vgg19 import preprocess_input, decode_predictions
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
from keras.models import Model
from keras.models import load_model
from pickle import load
import numpy as np
import io
from io import BytesIO
from PIL import Image
import pydantic
```

Once all the libraries are successfully imported, we create a FastAPI app instance and define a route for serving our HTML template.

```
app = FastAPI(title='Automatic Image Captioning')
templates = Jinja2Templates(directory='templates/')
```

In the index method, we simply return a basic HTML form that allows users to upload a file to the '/test/' endpoint. We have defined the method and decorated it with @app.get which means that our API supports the GET method.

```
@app.get("/test",tags=['HTML Connect'])
async def index(request:Request):
    return templates.TemplateResponse("test.html", context={"request": request})
```

Then, we define the '/test' endpoint which handles the file upload and image processing. This endpoint takes a file parameter of type bytes. It is read using io.BytesIO and encoded using base64.b64encode.

```
#app.post("/test",tags=['HTML Connect'])
def get_caption(request:Request, file: bytes = File(...)):
    try:
        image_html = io.BytesIO(file)
        image_html = base64.b64encode(image_html.getvalue()).decode("utf-8")

        image = Image.open(io.BytesIO(file))
        image = np.asarray(image.resize((224,224)))[...,:3]
        image_resnet50 = extract_features_resnet50(image)
        caption = generate_desc(trained_model,tokenizer,image_resnet50,40)
        caption = caption[10:-8]
        response_resnet50 = Response(caption, media_type='text/html')

        image_VGG19 = extract_features_VGG19(image)
        caption_VGG19 = generate_desc_VGG19(trained_model_VGG19,tokenizer_VGG19,image_VGG19,35)
        caption_VGG19 = caption_VGG19[8:-6]
        response_vgg19 = Response(caption_VGG19, media_type='text/html')

        return templates.TemplateResponse("test.html", context={"request": request,"result":caption, "result_VGG19":caption_VGG19,'pic':image_html,
        'model': "ResNet50", 'model_vgg19' : "VGG19"})
```

Reading file and encoding

Image pre-processing and feature extraction

Generate caption and fill the Response object

Here we are deploying both our models to predict captions for the same image.

```python
def get_model():
    # load the ResNet50 model
    model = load_model('30k_online_model_4.h5')
    return model

def get_model_VGG19():
    # load the VGG19 model
    model = load_model('best_model_epochs10.h5')
    return model
```

Once the code is ready, start the application using Uvicorn by executing the command:

Uvicorn main:app --reload

```
C:\Users\Harish Pattubala\Desktop\AIML\CapstoneProject\Deployment>uvicorn main:app --reload
←[32mINFO←[0m:     Will watch for changes in these directories: ['C:\\Users\\Harish Pattubala\\
Desktop\\AIML\\CapstoneProject\\Deployment']
←[32mINFO←[0m:     Uvicorn running on ←[1mhttp://127.0.0.1:8000←[0m (Press CTRL+C to quit)
←[32mINFO←[0m:     Started reloader process [←[36m←[1m17776←[0m] using ←[36m←[1mStatReload←[0m
2023-01-28 16:21:42.944591: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlo
w binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU
 instructions in performance-critical operations:  AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
←[32mINFO←[0m:     Started server process [←[36m18272←[0m]
←[32mINFO←[0m:     Waiting for application startup.
←[32mINFO←[0m:     Application startup complete.
```

http://127.0.0.1:8000/docs is the link to access the interactive system to test the endpoints. Below are the screenshots of our testing the endpoints.

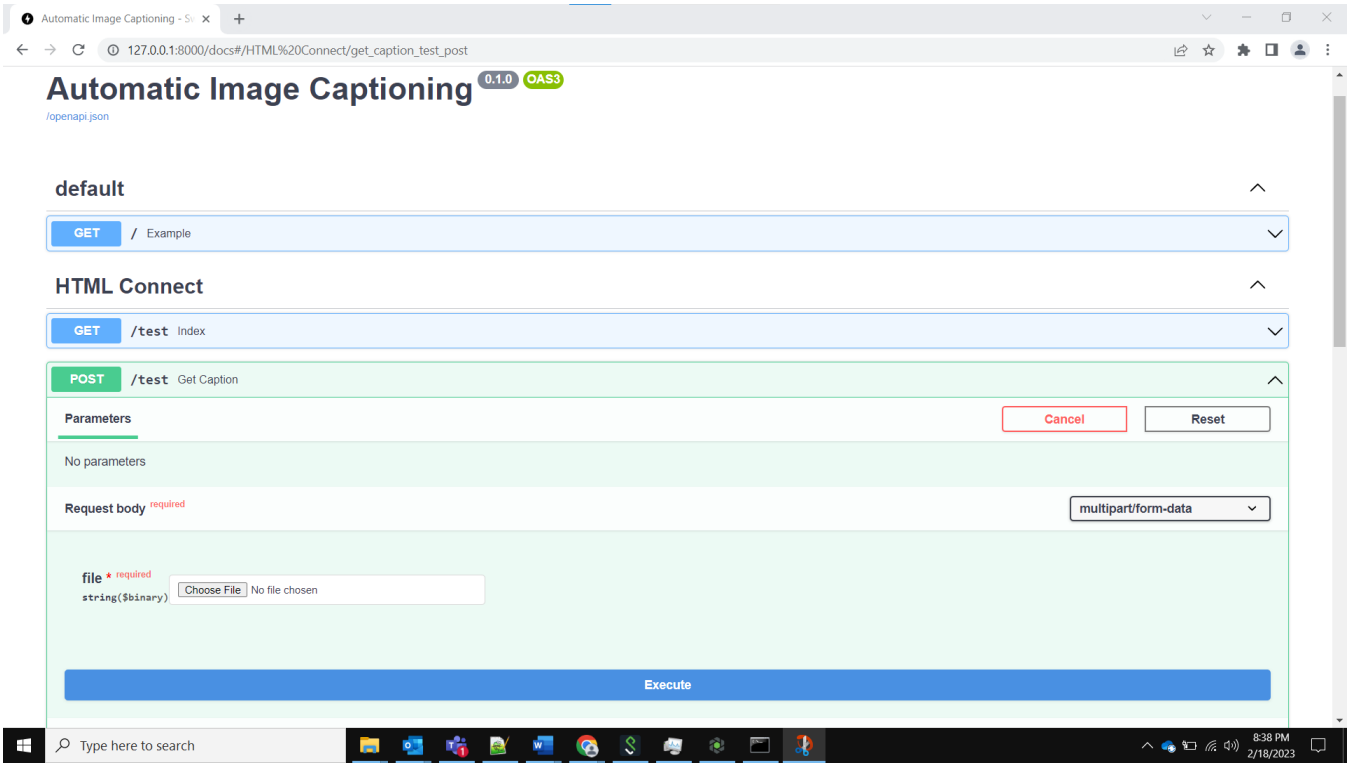


Figure.15 Initial page of the Dashboard

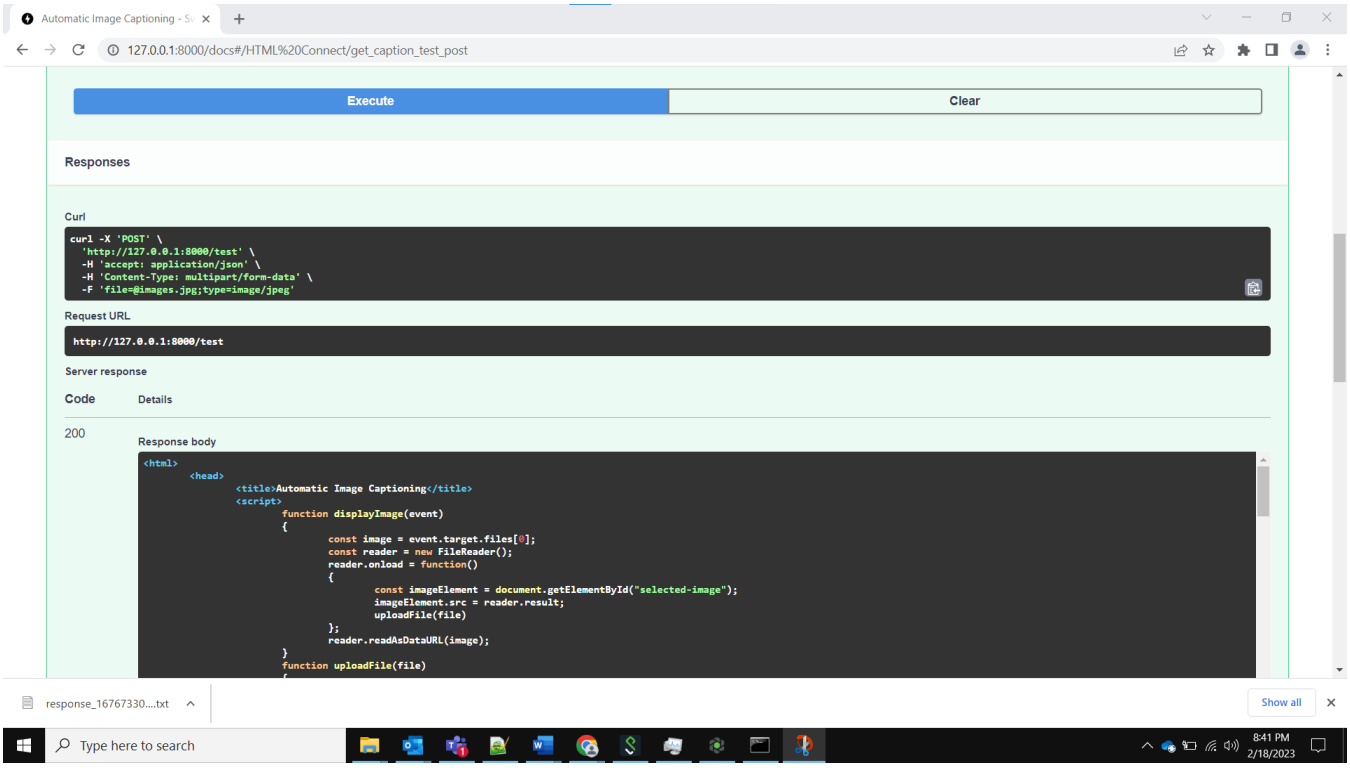


Figure.16 Result page of the Dashboard

Post successful deployment of our model as an API using FastAPI, we have created a HTML template which will allow users to interact with API and get the image data from the user and post the predicted captions on the screen.

```
<html>
    <head>
        <title>Automatic Image Captioning</title>
        <script>
            function displayImage(event)
            {
            function uploadFile(file)
            {
        </script>
        <style>
    </head>
<body style="background-color:powderblue;" align="center">
<h1 align="center" style="color:green;font-family:verdana;">Team 14</h1>
<h1 align="center" style="color:blue;">AUTOMATIC IMAGE CAPTIONING</h1>
<p align="center" style="font-size:20px;"><b>(Predicts captions for the gi
<hr>
<br/><br/>
<form action="/test" method="post" enctype="multipart/form-data">
    <input type="file" accept="image/*" name="file" id ="fileInput" onchan
    <tr><td><input type="submit" value="Generate Caption"></td></tr>
    <br/><br/>

</form>
```

The resultant html is as below:

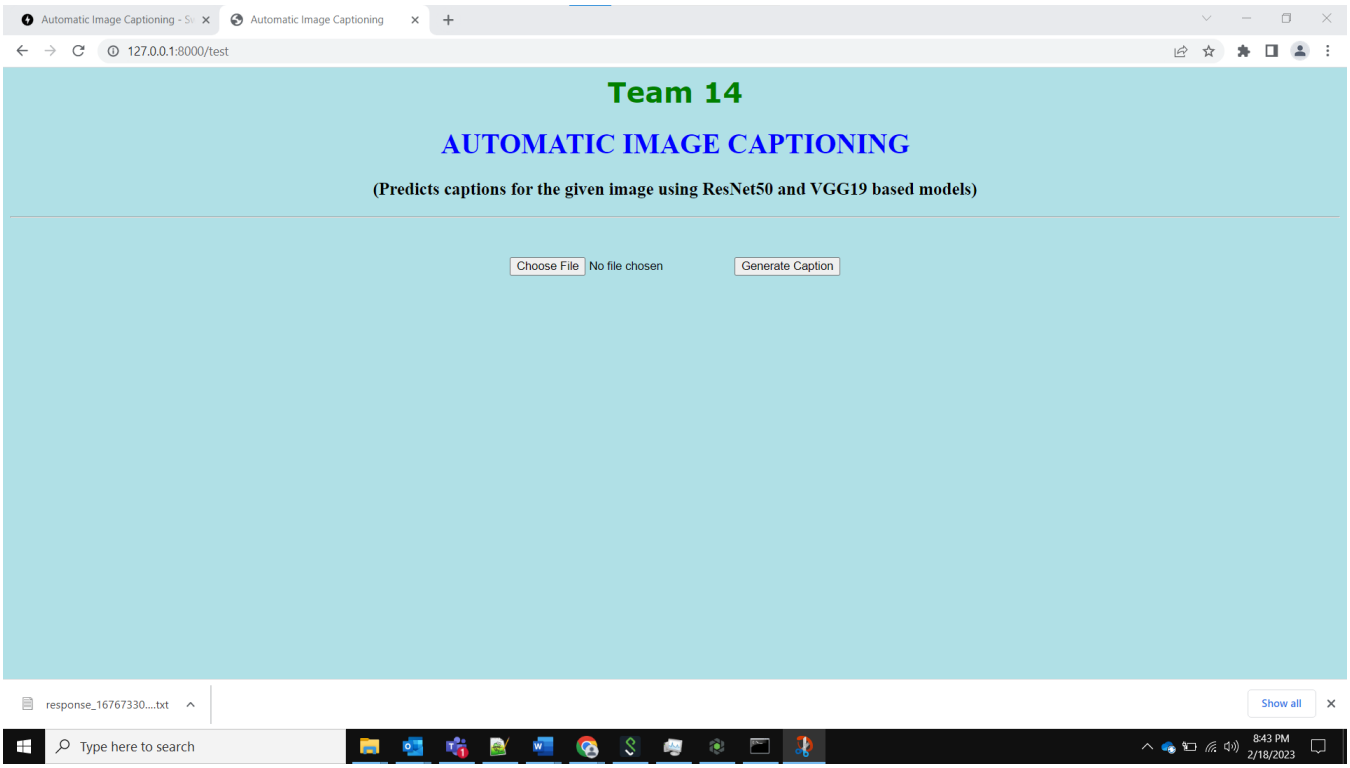


Figure.17 Dashboard

DEPLOYMENT RESULTS

Few snapshots of the final deployed dashboard as follows:

Figure.18 Dashboard results -1

Figure.19 Dashboard results -2

The Key point to note is that as our model is trained on data, it cannot predict the words that are out of its vocabulary. The current dataset that our model is trained on, has many dog images and the model predicts non-dog images as dogs.

Here is a small comparison of the captions generated by 8k-VGG19 model and 30K-RestNet50 model:

Image	**Caption generated by 30K-Resnet50 model**	**Caption generated by 8K-VGG19 model**
 Figure.20 Dashboard results -3	A soccer player is running in the air	Two men are playing soccer on the field
 Figure.21 Dashboard results -4	A group of young girls are playing with soccer ball	Two children are playing in the snow
 Figure.22 Dashboard results -5	The dog is bearing the snow	A white dog is running through the snow
 Figure.23 Dashboard results -6	A brown and white dog is jumping in the grass	dog is jumping through the grass
 Figure.24 Dashboard results -7	A yellow and white dog is jumping through the field	bird is flying through the water

The prediction of dogs is much better in comparison with other images. Similarly, the model is trained on a lot of soccer images, so the prediction is much better in that case too.

Apart from these, we have also handled exceptions in the code. As an example, here is an image that the model couldn't pre-process:

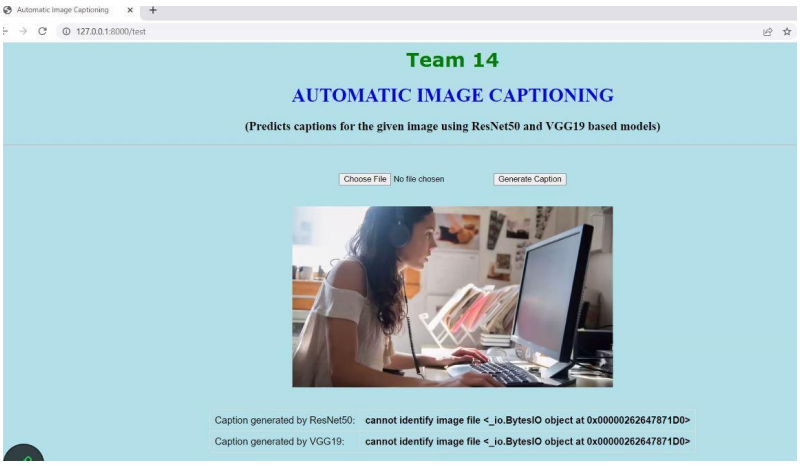


Figure.25 Dashboard results - Exception

APPLICABILITY IN THE REAL WORLD

This is a rapidly growing area of research that has numerous real-world applications like:

Accessibility: Image captions can help make images more accessible to visually impaired individuals who use text-to-speech software to navigate the web.

Figure.26 Use case -1

Search Engine Optimization (SEO): Image captions can improve the visibility of images in search engines, making it easier for users to find relevant images.

Figure.27 Use case -2

Social Media: Image captions can provide additional context for images shared on social media platforms, making them more engaging and shareable.

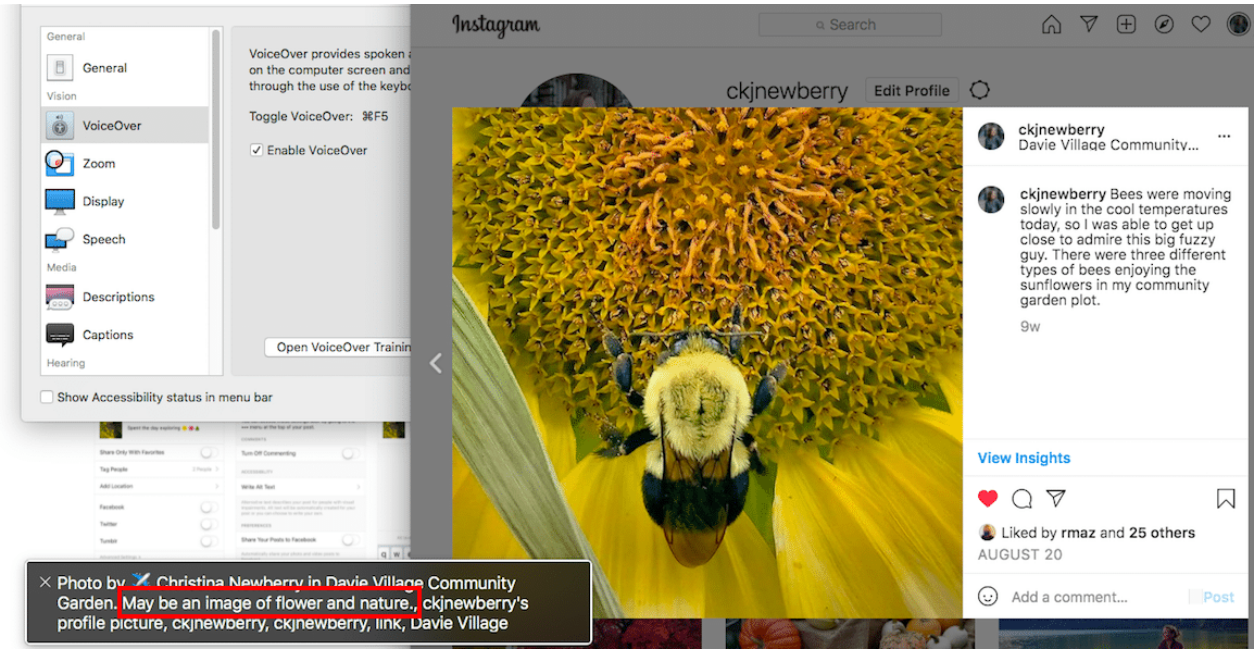


Figure.28 Use case -3

E-commerce: Image captions can provide additional product information for online shoppers, making it easier for them to find what they are looking for.

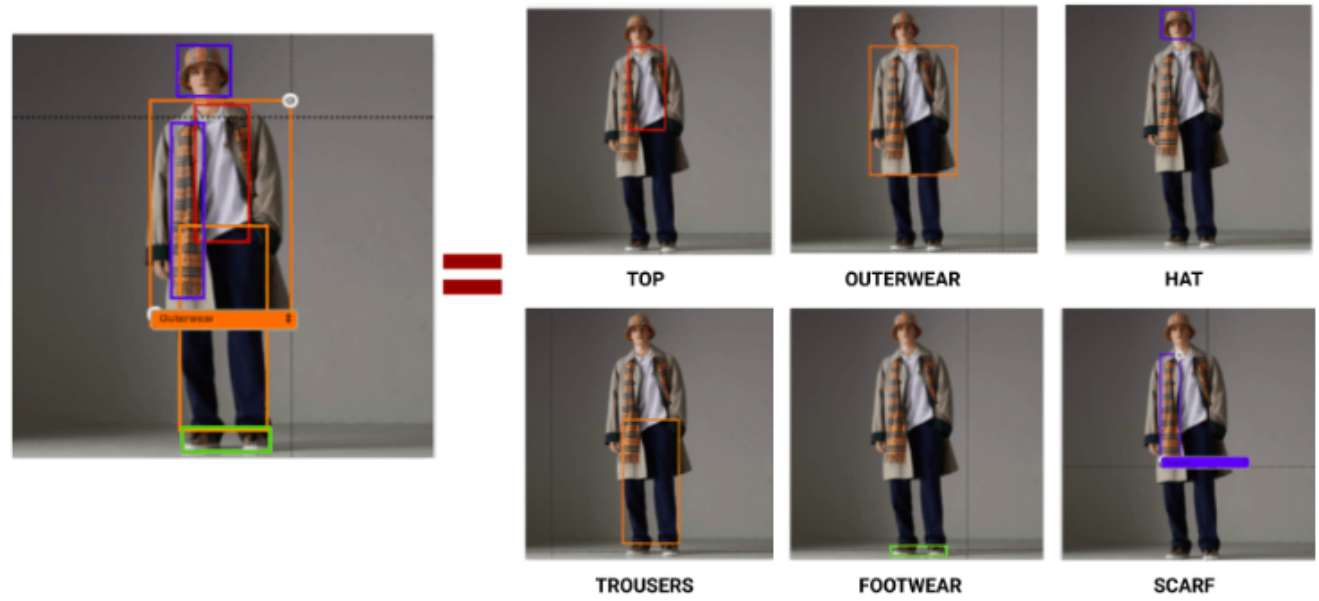


Figure.29 Use case -4

WHAT NEXT?

The project was really helpful in understanding how CNNs and LSTM work

EXTENSION TO THE CURRENT PROJECT

There are several potential extensions of the project on automatic image captioning, each of which has the potential to result in more accurate and diverse captions.

One possibility is to explore the use of transformers, which are a type of neural network architecture that has been shown to outperform traditional RNNs and LSTMs in terms of both accuracy and speed.

Another possibility is to experiment with image search based on the caption provided. As a basic version, we could use the same model to generate captions to a large set of images and store them along with the images in a database. When a user performs a search, we pre-process the user input (the same way we did for captions early on). The images that have the closest caption to the user input is returned as a search result. However, the potential issues we could be up for are synonymy(User's search string may have 'big' but our captions may have only 'large', causing the model to not return accurate images as it doesn't understand the synonyms) and polysemy(User's search string may have words that have multiple meanings. For example, bank might mean different in different contexts)

Other natural extensions could be to explore:

- Multi-modal models that incorporate other forms of data like audio, video, text etc.,
- Attention mechanisms that allow models to selectively focus on different areas of the image

IMPROVE ACCURACY OF EXISTING MODEL

Increasing the amount of data that a model is trained on, brings in a huge bump in the performance as shown below:

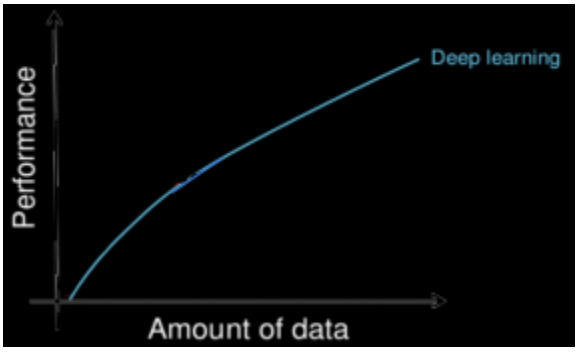


Figure.30 Data vs Performance graph

Improve Performance using data:

Using a larger dataset for training will improve accuracy. Like:

- Imagenet is the biggest image dataset with over 14 million images spread across 20,000 different classes. These have pre-built libraries to directly be used in model training.
- Cifar contains 80 million tiny images dataset. Cifar-10 contains 10 object classes, Cifar-100 contains 100 object classes. These datasets have an implementation in deep learning libraries.

There are few other similar datasets that we can choose and train the models based on our requirement.

Improve Performance using layers:
Adding a few more hidden layers is said to increase accuracy. Usually implementing three numbers of hidden layers gives the optimal performance in terms of time complexity and accuracy. On the other hand when the number of hidden layers cross the optimal number of hidden layers, time complexity increases in orders of magnitude as compared to the accuracy gain.

Improve Performance With Algorithm Tuning:
A quick way to get insight into the learning behavior of your model is to evaluate it on the training and a validation dataset each epoch, and plot the results. Observe the results for overfitting or underfitting and take measures to avoid it.

CONCLUSION

Current research in the field of automatic image captioning focuses on developing models that generate captions that are not only semantically and syntactically correct but also diverse, concise, and culturally and socially sensitive. Some models may generate captions that are culturally or socially insensitive, which can be harmful or offensive to some individuals. Generating captions that are diverse and inclusive is a major challenge in the field of automatic image captioning. This includes captions that accurately reflect the diverse appearances, cultures, and experiences of individuals in the images.

Evaluating the quality of generated captions is another major challenge in the field of automatic image captioning. This includes developing metrics that accurately capture the semantic and syntactic correctness of captions as well as the diversity and inclusivity of captions. Researchers are exploring new ways to evaluate the quality of generated captions.

REFERENCES

1. ImageNet Classification with Deep Convolutional Neural Networks(https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks) (Krizhevsky, Alex and Sutskever, Ilya and Hinton, Geoffrey E, 2012)
2. Neural Image Caption Generation with Visual-Semantic Alignments (https://cs.stanford.edu/people/karpathy/cvpr2015.pdf) (Andrej Karpathy and Fei-Fei 2015)
3. Show and Tell: A Neural Image Caption Generator (https://arxiv.org/abs/1411.4555) (Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan, 2015)

4. Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), 770-778.(https://openaccess.thecvf.com/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf) (He, K., Zhang, X., Ren, S., & Sun, J., 2016)
5. Very Deep Convolutional Networks for Large-Scale Image Recognition (https://arxiv.org/abs/1409.1556v6) (Karen Simonyan, Andrew Zisserman, 2014)
6. Deep Lake: a Lakehouse for Deep Learning (https://arxiv.org/abs/2209.10785) (Sasun Hambardzumyan, Abhinav Tuli, Levon Ghukasyan, Fariz Rahman, Hrant Topchyan, David Isayan, Mark McQuade, Mikayel Harutyunyan, Tatevik Hakobyan, Ivo Stranic, Davit Buniatyan, 2022)
7. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555. (https://arxiv.org/pdf/1412.3555.pdf)(Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. ,2014)
8. GloVe: Global Vectors for Word Representation(https://aclanthology.org/D14-1162) (Pennington et al., EMNLP 2014)
9. GloVe: Global Vectors for Word Representation (https://nlp.stanford.edu/projects/glove/) (Pennington et al., EMNLP 2014)
10. BLEU: a Method for Automatic Evaluation of Machine Translation (https://aclanthology.org/P02-1040.pdf) (Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu)
11. ROUGE: A Package for Automatic Evaluation of Summaries (https://aclanthology.org/W04-1013/) (Chin-Yew Lin, 2004)
12. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments (https://www.cs.cmu.edu/~alavie/METEOR/pdf/Banerjee-Lavie-2005-METEOR.pdf)(Satanjeev Banerjee, Alon Lavie, 2005)
13. CIDEr: Consensus-based Image Description Evaluation (https://arxiv.org/abs/1411.5726)(Ramakrishna Vedantam, C. Lawrence Zitnick, Devi Parikh, 2015)