# Develop a Deep Learning Model to find the Caption or Description of an Image given an Input Image.

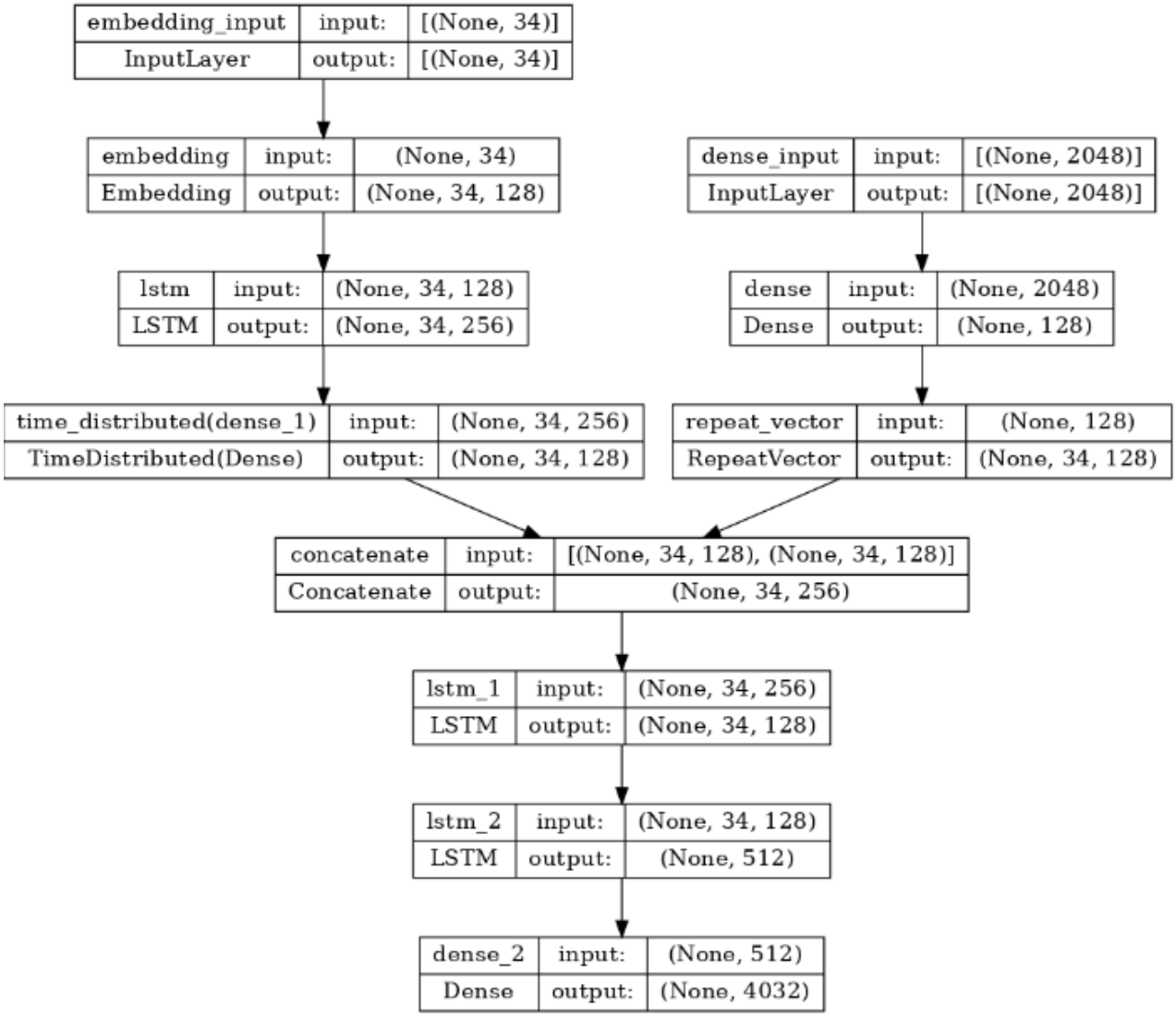
## Introduction:

An image caption description is a written caption that explains the important details of a picture. It involves generating a human readable textual description given an image, such as a photograph. For a human, it is a very simple task, but for a computer it is extremely difficult since it requires both understanding the content of an image and how to translate this understanding into natural language.

Recently, deep learning methods have displaced classical methods and are achieving state-of-the-art results for the problem of automatically generating descriptions, called “captions,” for images. In this project, we will see how deep neural network models can be used to automatically generate descriptions for images, such as photographs.

In this project, we use CNN and LSTM to generate the caption of the image. As the deep learning techniques are growing, huge datasets and computer power are helpful to build models that can generate captions for an image. This is what we are going to implement in this Python based project where we will use deep learning techniques like CNN and RNN.

## Technical Architecture:




# Pre-requisites

### Software and Tools

- Anaconda Navigator: Free and open-source distribution of Python and R for data science and machine learning applications.
- Jupyter Notebook and Spyder: Tools provided by Anaconda for development.
- Conda: Cross-platform package management system.
- Packages: Install the following packages using Anaconda prompt as administrator:

- NumPy: `pip install numpy`
- Pandas: `pip install pandas`
- Scikit-learn: `pip install scikit-learn`
- TensorFlow: `pip install tensorflow==2.3.2`
- Keras: `pip install keras==2.3.1`
- Flask: `pip install Flask`

### Deep Learning Concepts

- Convolutional Neural Network (CNN): A class of deep neural networks for visual imagery analysis.
- Flask: A Python web framework for building web applications.

## Project Objectives

By the end of this project, you will:

- Understand fundamental concepts and techniques of Convolutional Neural Network.
- Gain a broad understanding of image data.
- Learn how to pre-process/clean data using different techniques.
- Know how to build a web application using the Flask framework.

## Project Flow

### Data Collection

- Collect images of events with 5 captions each, organized into subdirectories.
- Download the dataset: Flickr8k Dataset.

### Image Pre-processing and Model Building

1. Importing Model Building Libraries
    - Import necessary libraries for model building.
2. Exploratory Data Analysis
    - Analyze and explore the dataset.
3. Initializing the Model
    - Use Keras Sequential class to define and initialize the model.
4. Configure the Learning Process
    - Compile the model with a loss function, optimizer, and metrics.

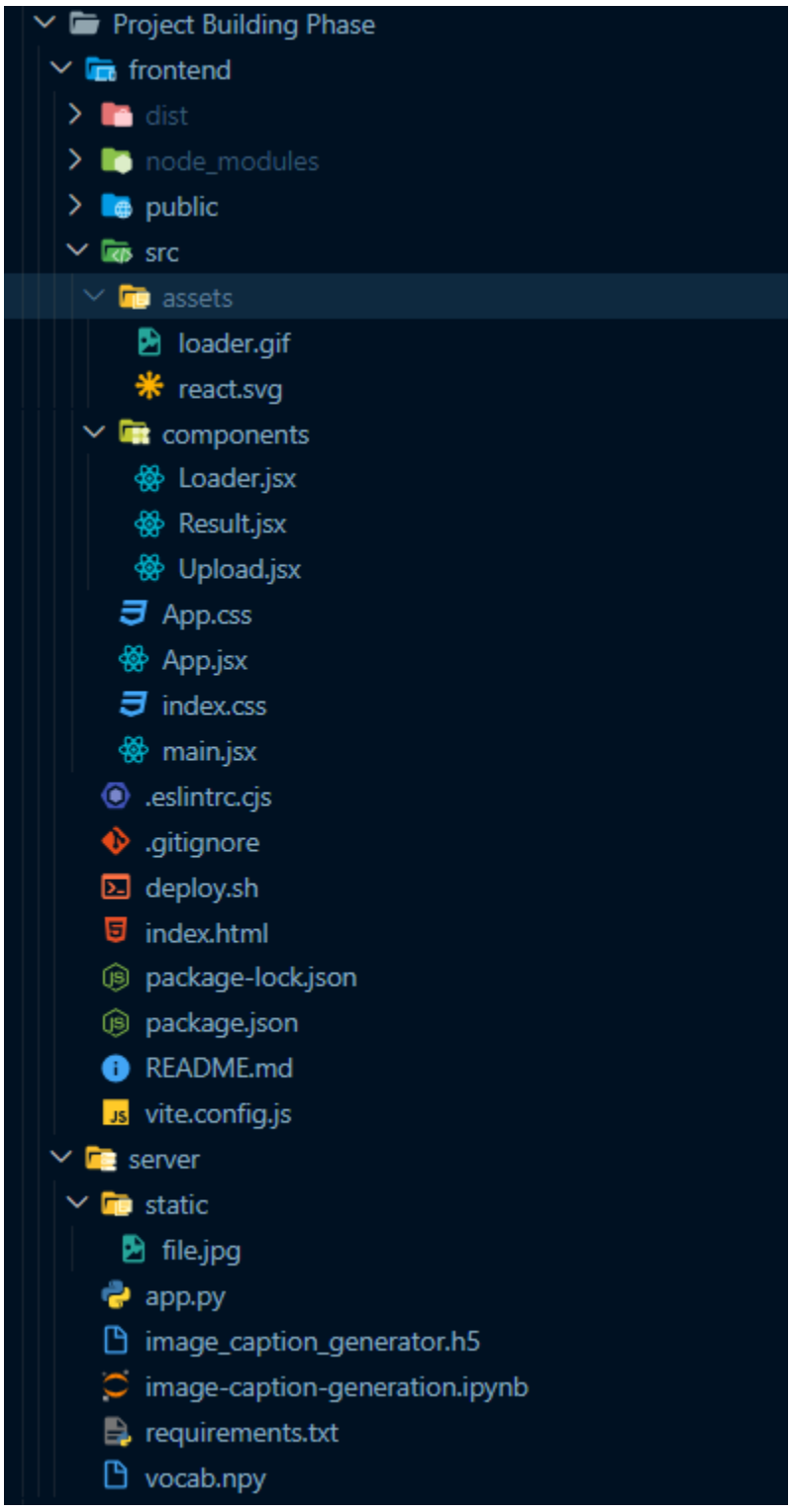
5. Training the Model
    - Train the model using image dataset.
6. Save the Model
    - Save the trained model in H5 format.
7. Test the Model
    - Evaluate the model's performance using test data.

**Application Building**

1. Create HTML Pages
    - Create HTML pages for home, introduction, and upload functionalities.
2. Python Flask Script (app.py)
    - Write Python script using Flask to run the project.

## Project Structure

- Project Folder:
    - Dataset: Contains training and testing images.
    - Flask Application:
        - Templates Folder: Contains HTML pages (index.html, prediction.html).
        - Python Script (app.py): For server-side scripting.
    - Model: Saved as model.h5.
    - Captions: Stored as tokenizer.pkl.


Project Building Phase
frontend
dist
node_modules
public
src
assets
loader.gif
react.svg
components
Loader.jsx
Result.jsx
Upload.jsx
App.css
App.jsx
index.css
main.jsx
.eslintrc.cjs
.gitignore
deploy.sh
index.html
package-lock.json
package.json
README.md
vite.config.js
server
static
file.jpg
app.py
image_caption_generator.h5
image-caption-generation.ipynb
requirements.txt
vocab.npy

# Milestones

Milestone 1: Collection of Data

- Collect images and captions based on the project structure. We have used Flickr 8k Dataset

Milestone 2: Image Pre-processing and Model Building

- Import libraries, explore data,

Importing the relevant libraries

+ Code + Markdown

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from glob import glob
from tqdm.notebook import tqdm
tqdm.pandas()
import cv2, warnings
warnings.filterwarnings('ignore')
import tensorflow as tf
from tensorflow.keras.layers import Dense, Flatten, Input, Add, Dropout, LSTM, TimeDistributed, Embedding, R
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import ModelCheckpoint
```

Loading the images

```
img_path = '/kaggle/input/flickr8k/Images/'
images = glob(img_path+'*.jpg')
images[:5]
```

```
len(images)
```

## Loading the captions

```
[ ]:
captions = open('/kaggle/input/flickr8k/captions.txt','rb').read().decode('utf-8').split('\n')
captions[:5]
```

```
[ ]:
len(captions)
```

## Visualizing images along with their captions

```
[ ]:
for i in range(5):
    plt.figure(figsize=(5,5))
    img = cv2.imread(images[i])
    img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    plt.imshow(img);
```

### Preprocessing the captions text

```
[ ]:
captions = captions[1:]
captions[:5]
```

```
[ ]:
captions[0].split(',')[1]
```

```
[ ]:
captions_dict = {}

for cap in captions:
    try:
        img_name = cap.split(',')[0]
        caption = cap.split(',')[1]
        # Each image has 5 captions
        if img_name in img_features:
            if img_name not in captions_dict:
                captions_dict[img_name] = [caption] # Storing the first caption
            else:
                captions_dict[img_name].append(caption) # Adding the remaining captions
    except:
        break
```

```
[ ]:
len(captions_dict)
```

```
[ ]:
def text_preprocess(text):
    modified_text = text.lower() # Converting text to lowercase
    modified_text = 'startofseq ' + modified_text + ' endofseq' # Appending the special tokens at the beginning and ending of text
    return modified_text
```

```
[ ]:
# Storing the preprocessed text within the captions dictionary
for key, val in captions_dict.items():
    for item in val:
        captions_dict[key][val.index(item)] = text_preprocess(item)
```

- Initialize and configure the model, train, save, and test.

## Downloading the ResNet50 inception model

[ ]:
```
inception_model = ResNet50(include_top=True)
inception_model.summary()
```

[ ]:
```
last = inception_model.layers[-2].output # Output of the penultimate layer of ResNet model
model = Model(inputs=inception_model.input,outputs=last)
model.summary()
```

## Extracting features from images

[ ]:
```
img_features = {}
count = 0

for img_path in tqdm(images):
    img = cv2.imread(img_path)
    img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    img = cv2.resize(img,(224,224)) # ResNet model requires images of dimensions (224,224,3)
    img = img.reshape(1,224,224,3) # Reshaping image to the dimensions of a single image
    features = model.predict(img).reshape(2048,) # Feature extraction from images
    img_name = img_path.split('/')[-1] # Extracting image name
    img_features[img_name] = features
    count += 1
    # Fetching the features of only 1500 images as using more than 1500 images leads to overloading memory
    if count == 1500:
        break
    if count % 50 == 0:
        print(count)
```

## Establishing the model architecture

[ ]:
```
embedding_len = 128
MAX_LEN = max_len
vocab_size = len(count_words)

# Model for image feature extraction
img_model = Sequential()
img_model.add(Dense(embedding_len,input_shape=(2048,),activation='relu'))
img_model.add(RepeatVector(MAX_LEN))

img_model.summary()

# Model for generating captions from image features
captions_model = Sequential()
captions_model.add(Embedding(input_dim=vocab_size+1,output_dim=embedding_len,input_length=MAX_LEN))
captions_model.add(LSTM(256,return_sequences=True))
captions_model.add(TimeDistributed(Dense(embedding_len)))

captions_model.summary()

# Concatenating the outputs of image and caption models
concat_output = Concatenate()([img_model.output,captions_model.output])
# First LSTM Layer
output = LSTM(units=128,return_sequences=True)(concat_output)
# Second LSTM Layer
output = LSTM(units=512,return_sequences=False)(output)
# Output Layer
output = Dense(units=vocab_size+1,activation='softmax')(output)
# Creating the final model
final_model = Model(inputs=[img_model.input,captions_model.input],outputs=output)
final_model.compile(loss='categorical_crossentropy',optimizer='RMSprop',metrics='accuracy')
final_model.summary()
```

## Model training

[ ]:
```
mc = ModelCheckpoint('image_caption_generator.h5',monitor='accuracy',verbose=1,mode='max',save_best_only=True)

final_model.fit([X,y_in],
                y_out,
                batch_size=512,
                callbacks=mc,
                epochs=200)
```

[ ]:
```
# Creating an inverse dictionary with reverse key-value pairs
inverse_dict = {val: key for key,val in count_words.items()}
```

## Saving the final trained model and the vocabulary dictionary

[ ]:
```
final_model.save('image_caption_generator.h5')
```

[ ]:
```
np.save('vocab.npy',count_words)
```

Generating sample predictions

```
[ ]:
# Custom function for extracting an image and transforming it into an appropriate format
def getImage(idx):
    test_img_path = images[idx]
    test_img = cv2.imread(test_img_path)
    test_img = cv2.cvtColor(test_img,cv2.COLOR_BGR2RGB)
    test_img = cv2.resize(test_img,(224,224))
    test_img = np.reshape(test_img,(1,224,224,3))
    return test_img
```

```
for i in range(10):
    random_no = np.random.randint(0,1501,(1,1))[0,0]
    test_feature = model.predict(getImage(random_no)).reshape(1,2048)
    test_img_path = images[random_no]
    test_img = cv2.imread(test_img_path)
    test_img = cv2.cvtColor(test_img,cv2.COLOR_BGR2RGB)
    pred_text = ['startofseq']
    count = 0
    caption = '' # Stores the predicted captions text

    while count < 25:
        count += 1
        # Encoding the captions text with numbers
        encoded = []

        for i in pred_text:
            encoded.append(count_words[i])

        encoded = [encoded]
        # Padding the encoded text sequences to maximum length
        encoded = pad_sequences(encoded,maxlen=MAX_LEN,padding='post',truncating='post')
        pred_idx = np.argmax(final_model.predict([test_feature,encoded])) # Fetching the predicted word index having the maximum probability of occurr
        sampled_word = inverse_dict[pred_idx] # Extracting the predicted word by its respective index
        # Checking for ending of the sequence
        if sampled_word == 'endofseq':
            break
        caption = caption + ' ' + sampled_word
        pred_text.append(sampled_word)

    plt.figure(figsize=(5,5))
```

## Milestone 3: Application Building

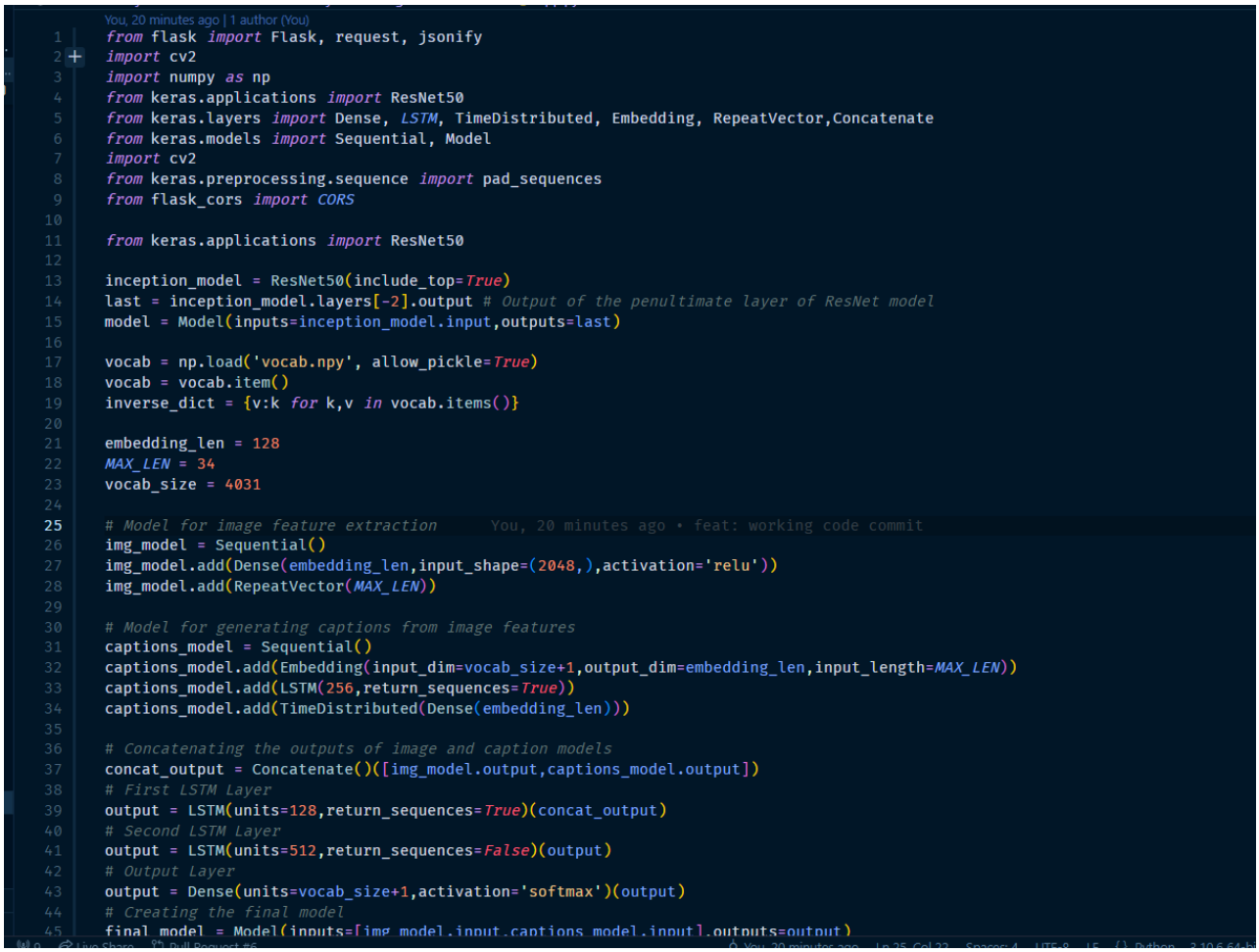
- Create HTML pages using React Framework

```
You, 1 second ago | 1 author (You)
import { useState } from "react";
import "../index.css";
import Result from "./Result";

const ImageCaptionGenerator = () => {
  const [selectedFile, setSelectedFile] = useState("");
  const [bool, setBool] = useState(false);

  const handleImageChange = (event) => {
    const img = event.target.files[0];
    setSelectedFile(img);
  };

  const handleGenerateCaption = () => {      You, now • Uncommitted changes
    if (selectedFile) setBool(true);
    else {
      window.alert("Select image first");
    }
  };

  return (
    <>
      <div>
        {!bool && (
          <>
            <input
              type="file"
              style={{ color: "black" }}
              onChange={handleImageChange}
            />
            <div>
              <button onClick={handleGenerateCaption}>Generate Caption</button>
            </div>
          </>
        )}

        {bool && <Result img={selectedFile} />}
      </div>
    </>
  );
};

export default ImageCaptionGenerator;
```


SI-GuidedProject-598189-1699954964 > Project Building Phase > frontend > src > components > Result.jsx > Result > useEffect() callback
You, 19 minutes ago | 1 author (You)
import { useEffect, useState } from "react";
import Loader from "./Loader";
import Upload from "./Upload";
const Result = (props) => {
const [preview, setPreview] = useState();
const [caption, setCaption] = useState(); // Changing caption on UI
const [bool1, setBool] = useState(false);
const fetchCaption = async () => {
const formData = new FormData();
formData.append("file", props.img);
'img' is missing in props validation
try {
const url = `http://localhost:9090/generate`;
const response = await fetch(url, {
method: "Post",
body: formData,
});
const data = await response.json();
setCaption(data.caption);
} catch (err) {
console.log(err);
}
};
useEffect(() => {
setPreview(URL.createObjectURL(props.img));
'img' is missing in props validation
You, 19 minutes ago • feat: wo
fetchCaption();
}, []);
React Hook useEffect has missing dependencies: 'fetchCaption' and 'props.img'. Either include them or remove t
const handleClick = () => {
setBool(true);
};
return (
<>
{!bool1 && (

<div className="result-window" style={{ position: "reative" }}>
<button
style={{ color: "black", marginLeft: "-31rem" }}
className="result-logout"

- Writing Flask Script



```
You, 20 minutes ago | 1 author (You)
from flask import Flask, request, jsonify
import cv2
import numpy as np
from keras.applications import ResNet50
from keras.layers import Dense, LSTM, TimeDistributed, Embedding, RepeatVector,Concatenate
from keras.models import Sequential, Model
import cv2
from keras.preprocessing.sequence import pad_sequences
from flask_cors import CORS

from keras.applications import ResNet50

inception_model = ResNet50(include_top=True)
last = inception_model.layers[-2].output # Output of the penultimate layer of ResNet model
model = Model(inputs=inception_model.input,outputs=last)

vocab = np.load('vocab.npy', allow_pickle=True)
vocab = vocab.item()
inverse_dict = {v:k for k,v in vocab.items()}

embedding_len = 128
MAX_LEN = 34
vocab_size = 4031

# Model for image feature extraction      You, 20 minutes ago • feat: working code commit
img_model = Sequential()
img_model.add(Dense(embedding_len,input_shape=(2048,),activation='relu'))
img_model.add(RepeatVector(MAX_LEN))

# Model for generating captions from image features
captions_model = Sequential()
captions_model.add(Embedding(input_dim=vocab_size+1,output_dim=embedding_len,input_length=MAX_LEN))
captions_model.add(LSTM(256,return_sequences=True))
captions_model.add(TimeDistributed(Dense(embedding_len)))

# Concatenating the outputs of image and caption models
concat_output = Concatenate()([img_model.output,captions_model.output])
# First LSTM Layer
output = LSTM(units=128,return_sequences=True)(concat_output)
# Second LSTM Layer
output = LSTM(units=512,return_sequences=False)(output)
# Output Layer
output = Dense(units=vocab_size+1,activation='softmax')(output)
# Creating the final model
final_model = Model(inputs=[img_model.input,captions_model.input],outputs=output)
```

```python
# Creating the final model
final_model = Model(inputs=[img_model.input,captions_model.input],outputs=output)
final_model.compile(loss='categorical_crossentropy',optimizer='RMSprop',metrics='accuracy')

final_model.load_weights('image_caption_generator.h5')

app = Flask(__name__)
app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 1
cors = CORS(app, resources={r"/*": {"origins": "*"}})

@app.route('/generate', methods=['POST'])
def after():
    global final_model,vocab,inverse_dict,model
    file = request.files['file']

    file.save('static/file.jpg')
    img = cv2.imread('static/file.jpg')
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img,(224,224,))                    # diff
    img = np.reshape(img,(1,224,224,3))

    test_feature = model.predict(img).reshape(1,2048)
    pred_text = ['startofseq']
    count = 0
    caption = '' # Stores the predicted captions text

    while count < 25:
        count += 1
        # Encoding the captions text with numbers
        encoded = []

        for i in pred_text:
            encoded.append(vocab[i])

        encoded = [encoded]
        # Padding the encoded text sequences to maximum length
        encoded = pad_sequences(encoded,maxlen=34,padding='post',truncating='post')
        pred_idx = np.argmax(final_model.predict([test_feature,encoded])) # Fetching the predicted word index having the maximum probability of occurrence
        sampled_word = inverse_dict[pred_idx] # Extracting the predicted word by its respective index
        # Checking for ending of the sequence
        if sampled_word == 'endofseq':
            break
        caption = caption + ' ' + sampled_word
        pred_text.append(sampled_word)

    return jsonify({'caption': caption})


if __name__ == "__main__":
    app.run(port=9090,debug=True)
```

## Milestone 4: Final Implementation

- Build and run the Flask application locally.

Go back

# Result page

**a boy is surfing**