**Clip text Training.ipynb**

It is a complete workflow for performing text classification on images using the CLIP (Contrastive Language-Image Pretraining) model. CLIP is a model developed by OpenAI that can understand and generate text from images etc. In the above code, the CLIP model is used to extract text information from images and then perform text classification based on that extracted information. Here's a breakdown of the code step by step:

1. **Import Required Libraries:**

The code starts by importing necessary libraries such as os for file operations, cv2 for image processing, numpy for numerical operations, torch for PyTorch functionalities, clip for loading the CLIP model, and various components from sklearn and torchvision for metrics and data handling.

**import os**

**import cv2**

**import numpy as np**

**import torch**

**import clip**

**from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score**

**from torch.utils.data import Dataset, DataLoader**

**from torchvision import transforms**

1. **Load CLIP Model and Tokenizer:**

The code loads the CLIP model and tokenizer using the clip.load function. The chosen model variant is "ViT-B/32". The model is loaded onto the GPU if available; otherwise, it's loaded onto the CPU.

**# Load the CLIP model and tokenizer**

**device = "cuda" if torch.cuda.is\_available() else "cpu"**

**model, preprocess = clip.load("ViT-B/32", device)**

1. **Load and Preprocess Image Data for Text Extraction:**

The code defines a custom ImageTextDataset class that is inherited from torch.utils.data.Dataset. This class is used to load image data from a specified folder and preprocess it for later use. Each image is loaded using OpenCV (cv2) and stored along with its corresponding label. The \_\_getitem\_\_ method of the class performs the preprocessing by applying the specified transformations.

1. **Define Image Transformation:**

The code defines a transformation pipeline using torchvision.transforms.Compose. The transformation includes converting images to a PIL format, resizing them to a target size, converting them to tensors, and normalizing the pixel values.

**# Define the transformation to preprocess images for the CLIP model**

**target\_size = (224, 224) # Specify the target size for images**

**transform = transforms.Compose([**

**transforms.ToPILImage(),**

**transforms.Resize(target\_size),**

**transforms.ToTensor(),**

**transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])**

**])**

1. **Extract Text from Images and Perform Text Classification:**

The code specifies the folder containing the image-text dataset. It creates an instance of the ImageTextDataset class using the specified transformation. Then, it prepares a DataLoader to efficiently iterate through the dataset in batches.

**# Step 2: Extract Text from Images and Perform Text Classification**

**image\_text\_data\_folder = "text-dataset" # Change this to your dataset folder**

**image\_text\_dataset = ImageTextDataset(image\_text\_data\_folder, transform=transform)**

1. **Text Classification using CLIP:**

The code defines a linear classifier head (classifier\_head) that takes the text features extracted from the CLIP model as input and predicts the class labels. It uses a cross-entropy loss for training. The code runs through multiple epochs, iterating over batches of images and labels from the DataLoader. It extracts text features using CLIP's image encoding, computes logits using the classifier head, calculates the loss, performs backpropagation, and updates the model's parameters.

1. **Evaluate the Text Classification Model:**

After training, the code evaluates the text classification model. It sets the classifier head to evaluation mode and iterates through batches of images and labels, extracting text features, computing logits, and making predictions. The code calculates accuracy, precision, recall, and F1-score using the predicted labels and the ground truth labels.

1. **Save Text Classification Model:**

The code saves the trained text classification model (classifier head) to a file named "text\_model.pt" using torch.save.

1. **Print Results:**

Finally, the code prints the evaluation metrics (accuracy, precision, recall, and F1-score) for the text classification task.

**Clip-nature.ipynb**

It is an example of a workflow for training a fine-tuned model for nature image classification using a pre-trained CLIP (Contrastive Language-Image Pretraining) model. The CLIP model is used to extract image features, and a simple linear classifier head is added to perform the actual classification. Here's a breakdown of the code step by step:

1. **Install Required Packages:**

The code starts by installing the necessary packages: openai.clip and the CLIP repository from GitHub using pip.

1. **Import Required Libraries:**

Import various libraries including os for file operations, cv2 for image processing, numpy for numerical operations, torch for PyTorch functionalities, nn for neural network modules, optim for optimizers, and clip for loading the CLIP model. It also imports functions from sklearn for calculating classification metrics.

1. **Load and Preprocess Images for Nature Classification:**

Defines a custom ImageDataset class that loads and preprocesses images for classification. Images are resized to (224, 224) to match the input shape of the CLIP model. Labels are assigned based on class folders in the dataset. The \_\_getitem\_\_ method preprocesses images using the specified transformation.

1. **Define Image Transformation:**

Defines a transformation pipeline that converts images to tensors and normalizes their pixel values.

1. **Load the Fine-Tuned CLIP Model and Tokenizer:**

Loads the fine-tuned CLIP model using the clip.load function. The model variant is "ViT-B/32". The model is loaded onto the GPU if available; otherwise, it's loaded onto the CPU.

1. **Model Training for Nature Classification:**

Creates an instance of the ImageDataset class using the specified transformation and prepares a DataLoader to iterate through the dataset in batches. A linear classifier head is defined with the number of output classes matching the number of unique labels in the dataset. The code then trains the classifier head using a training loop that iterates over the dataset and performs backpropagation to update the classifier head's parameters.

1. **Model Evaluation for Nature Classification:**

After training, the code evaluates the nature classification model. It sets the classifier head to evaluation mode and iterates through batches of images and labels. It calculates image features using the CLIP model, computes logits using the classifier head, and predicts the class labels. It then calculates accuracy, precision, recall, and F1-score metrics using the predicted labels and the ground truth labels.

1. **Save Nature Model:**

The trained classifier head is saved to a file named "nature\_model.pt" using torch.save.

1. **Print Results:**

The code prints the evaluation metrics (accuracy, precision, recall, and F1-score) for the nature classification task.

The overall workflow involves loading images, using CLIP to extract image features, training a simple linear classifier head, evaluating the model's performance, and finally saving the trained model.

**islamophobia\_training.ipynb**

The code demonstrates a complete workflow for performing text and image classification using two different models: a Support Vector Classifier (SVC) for text classification and a Convolutional Neural Network (CNN) for nature image classification. The code consists of multiple steps, each focusing on a specific task. Let's break down the code step by step:

1. **Import Required Libraries:**

The code starts by importing various libraries, including os for file operations, cv2 for image processing, pytesseract for text extraction from images, numpy for numerical operations, and various components from sklearn, tensorflow.keras, and matplotlib for machine learning and visualization.

1. **Data Preparation:**

Specifies the paths to the folders containing the image datasets for text classification (svc\_data\_folder) and nature classification (cnn\_data\_folder).

1. **Text Detection:**

Defines a function extract\_text\_from\_image that takes an image path, reads the image, converts it to grayscale, and uses pytesseract to extract text from the image.

1. **Feature Extraction for Text Classification:**

Defines a function extract\_text\_features that iterates through image folders, extracts text using the previous function, and then converts the extracted text into numerical features using TF-IDF vectorization. Returns the text features, labels, and the vectorizer used for transformation.

1. **Model Training for Text Classification:**

Initializes a SVC model with a linear kernel. The code then calls the fit method on the model, passing the text features and labels for training.

1. **Image Preprocessing:**

Defines a function preprocess\_image that takes an image path, reads the image, resizes it to (224, 224) for CNN input, and normalizes pixel values to the range [0, 1].

1. **Load and Preprocess Images for Nature Classification:**

Defines a function load\_images that loads images from the provided folder, preprocesses them using the preprocess\_image function, and returns the preprocessed images and corresponding labels.

1. **Model Training for Nature Classification (CNN):**

Defines a sequential CNN model using TensorFlow Keras. The model consists of several convolutional and pooling layers, followed by fully connected layers. The model is compiled with the 'adam' optimizer and 'sparse\_categorical\_crossentropy' loss, and then trained on the preprocessed images and labels.

1. **Print Model Summary:**

Prints a summary of the architecture of the nature classification CNN model.

1. **Model Training and Evaluation:**

The code trains the CNN model on the nature classification dataset using the specified number of epochs and batch size.

This code demonstrates a complete pipeline, including text extraction, feature extraction, model training, and evaluation for both text and image classification tasks.