PackageX Assignment 2

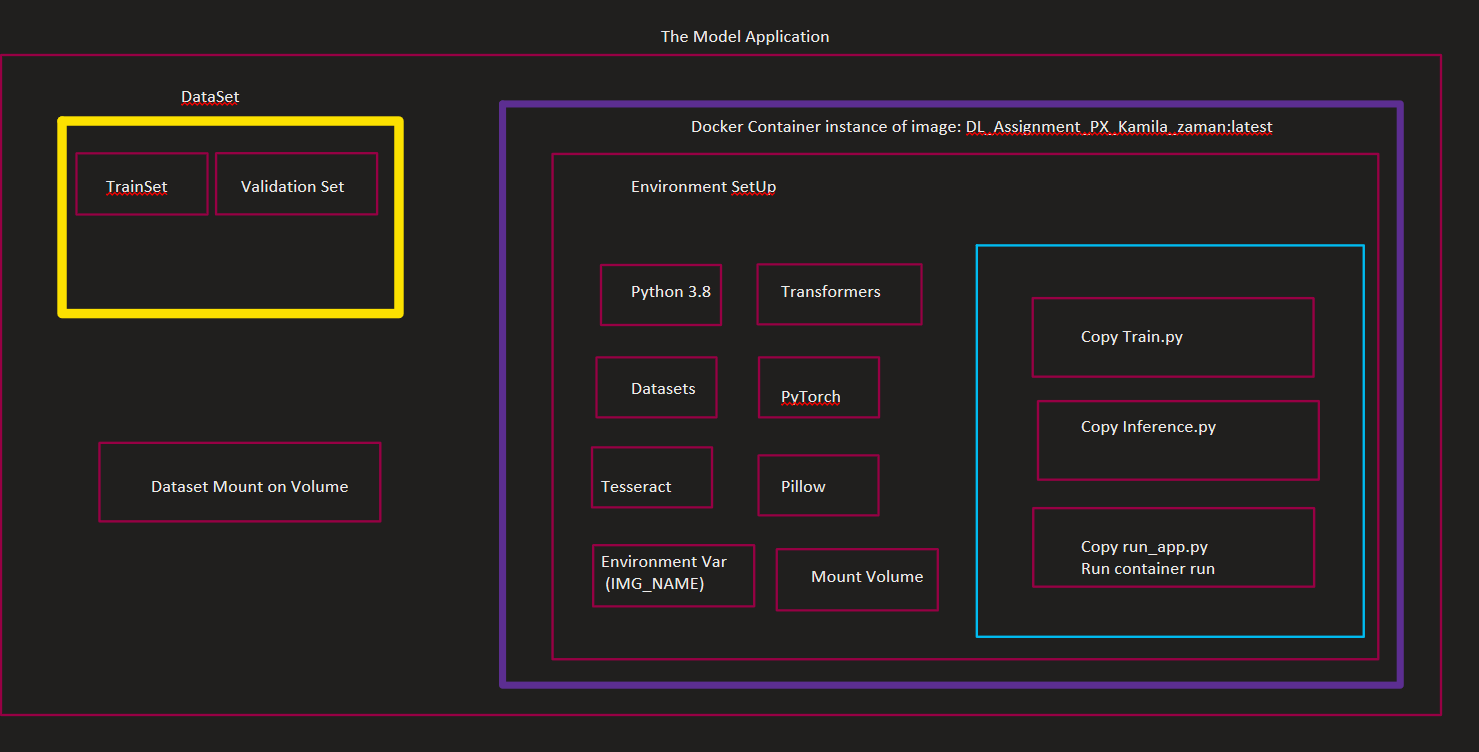
Kamila Zaman – 14th December 2022

# Problem Statement

*“Document classification using LayoutLMv3 model architecture”*

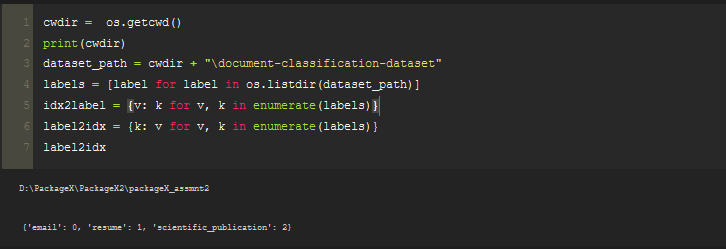
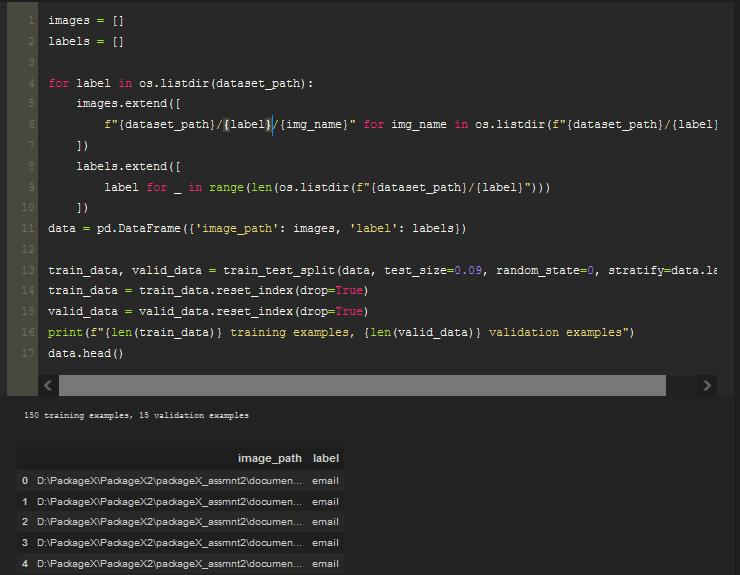
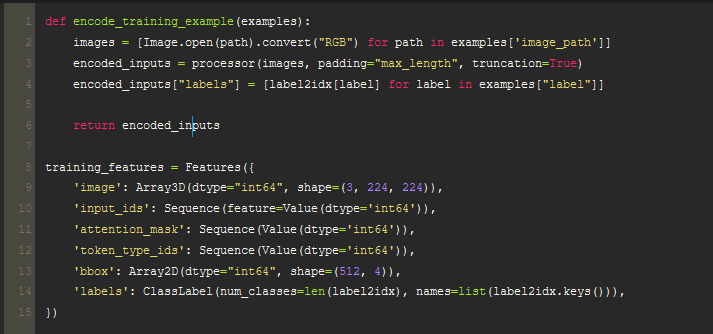
**Approach Sketch:**

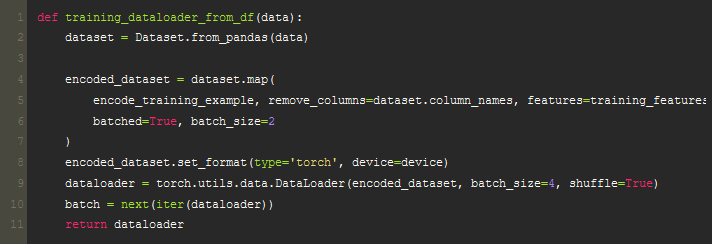
1. Identify required packages and their compatible version
2. Using python 3.8.9 – most stable with transformers and pytorch for LayoutLMV models as per online community
3. Create Docker Image from defined dockerfile with all required modules and components
4. Run Docker container of created image with dataset mount for image directory
5. Attach running container to VSCode IDE using Dev-Containers extension for code development and testing
6. Develop Solution
7. Create ReadMe.md
8. Specify commands for running complete project independently
9. Upload to Github

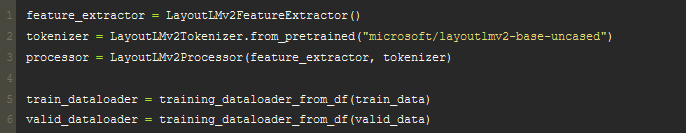


**Task Break Down:**

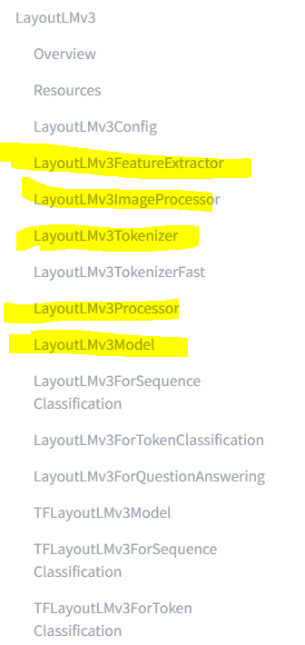
# Data

* 1. **Download dataset**
  2. **Understand**
     1. Contains 3 classes of document types
     2. Format: .png files
  3. **Summarize**
     1. Classes: email(55 items), resume (55 items) and scientific publication (55 items)
  4. **Scan for potential issue for pre-processing (data cleaning/resampling required or not)**
     1. Since data is equally distributed for each class, we do not have to worry about class under or over representation hence no resampling required
     2. Quality of data is good; all documents are already sorted and clean
  5. **Read Data** 
     1. Directory wise reading of files into a pandas dataframe
     2. Enumerate Folders with class number
     3. Class labels: Email -> 0 , Resume -> 1, Scientific Publication -> 2
  6. **Pre-Process** 
     1. Label generation required since currently images sorted manually folder wise. Read and assign label by folder to all images and create data frame
  7. **Decide and create train/ test split**
     1. I will go by the industry standard of doing a 80% training and 20% testing. I will not do a validation set as we have small dataset and we will estimate model performance on test/validation set.
     2. 150 sample for training, 15 samples for test set
  8. **Data Encoding** 
     1. Apply required encoding on the images for the layoutLMV3 model specifications
     2. Data frame containing image and label will be created

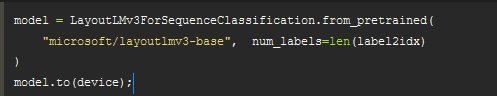
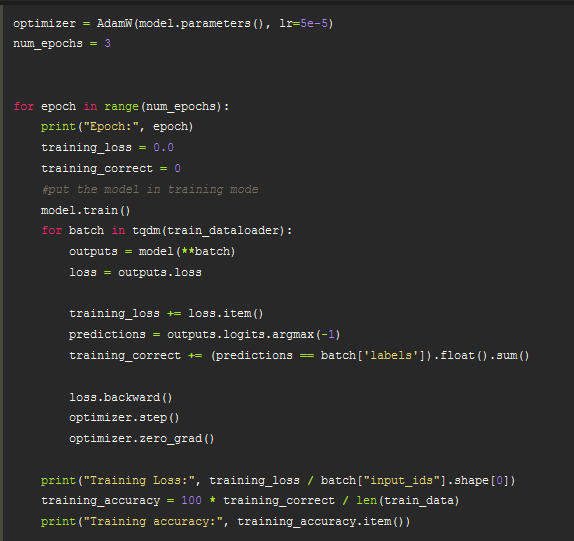
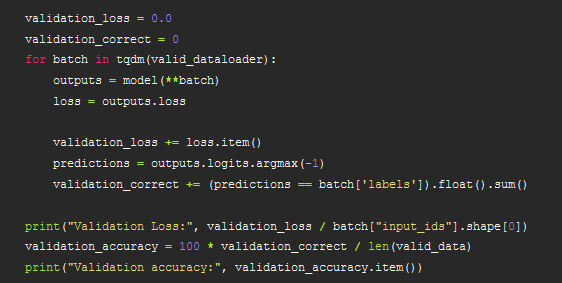


* 1. **Create Data loaders for training phase and testing/validation phase of the model**
     1. Training Data Loder
     2. Test Data Loader (Valid Data Loader)

# Model Architecture Understanding & Creation

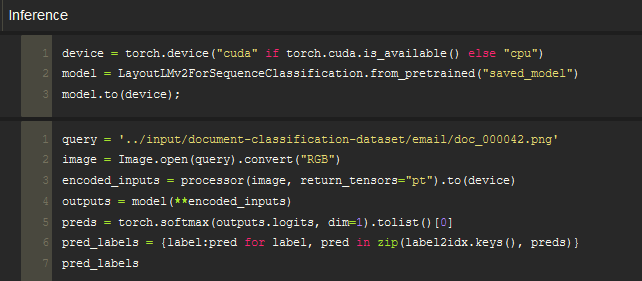
* 1. Layout
     + The LayoutLM model is based on BERT architecture but with two additional types of input embeddings. The first is a 2-D position embedding that denotes the relative position of a token within a document, and the second is an image embedding for scanned token images within a document.
     + Open source and made available in huggingface library
     + The LayoutLMv3 model was proposed in LayoutLMv3: Pre-training for Document AI with Unified Text and Image Masking by Yupan Huang, Tengchao Lv, Lei Cui, Yutong Lu, Furu Wei. LayoutLMv3 simplifies LayoutLMv2 by using patch embeddings (as in ViT) instead of leveraging a CNN backbone, and pre-trains the model on 3 objectives: masked language modeling (MLM), masked image modeling (MIM) and word-patch alignment (WPA).
     + LayoutLMv3 is a pre-trained multimodal Transformers for Document AI with unified text and image masking.
     + LayoutLMv3 is identical to its predecessor LayoutLMv2 in terms of Data Processing, except that: Images need to be resized and normalized with channels in regular RGB format. LayoutLMv2 on the other hand normalizes the images internally and expects the channels in BGR format.
     + text is tokenized using byte-pair encoding (BPE), as opposed to WordPiece. Due to these differences in data preprocessing, one can use LayoutLMv3Processor which internally combines a LayoutLMv3FeatureExtractor (for the image modality) and a LayoutLMv3Tokenizer/LayoutLMv3TokenizerFast (for the text modality) to prepare all data for the model.
     + <https://huggingface.co/docs/transformers/model_doc/layoutlmv3>
  2. Python module to use -> Pytorch (most familiar with)
     1. Hugging face LayoutLMV3 distribution in pytorch transformers by Microsoft
     2. We will use their pretrained models for tokenization vocabulary and layout of the model for feature extraction
  3. Identify libraries and modules to use for it
  4. Serialize/ deserialize inbuilt in pytorch with save\_model function on model object
  5. Variations or only single model? (Single model definition change opt parameters for accuracy improvement) – Single Model(Tune later post performance analysis )
  6. Identify accuracy measures
     1. Accuracy measure class in pytorch

# Coding

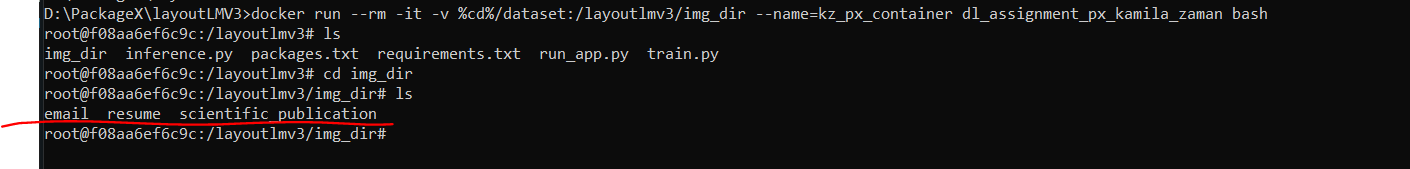
* 1. Data Pre-processing
     1. Explain Detail
  2. Building the Model – Instantiating a model object
  3. Training – Train on the 150 images prepared within the train dataloder, specify the optimizer and loss function and step for weight updates. Zero Grad allows us to avoid the problem of exploding or vanishing gradients.
  4. Number of epochs are the passes of the training data, 3 is good for a set of 150 samples. Increasing value for such a small dataset will cause overfitting
  5. Model Testing (Validation) – Performed on the valid data loader prepared from the 15 images set aside for validation from the 165 data points
  6. Validation Accuracy: the rate of correct predictions out of all prediction on unseen data
  7. Gives true measure of how the model is actually performing
  8. Save the Model: Pytorch allows to save the model, pickle or joblib can also be used but basic save operator used for now. Trained model is usually saved as a serialized file that is read from persistent storage at the time of inference, basically in inference.py the model with learnt weights is reloaded into a similar model object and used to make predictions on new input images as done for validation images
  9. Inference

LAyoutLMV3 used various features of the document to make the prediction on what category it belongs.

The main features are:

1. Words used in the document
2. Layout of phrases, titles and sentences – OCR support – creating bounding boxes etc
3. And lastly the structural layout od the document as well, where the name comes from.

Hence, given the understanding of the model, the training part has extracted all those features and mapped them against respective classes. The inference will now basically take an unseen input image and given the layout, vocabulary and placement feature learnt from the training sample it will infer what class this current document belongs to and assign that label to it.

1. **Docker Image** 
   1. Package and requirements => declare downloads in environment within Dockerfile
      1. *Python 3.8.9* - most compatible for LayoutLMV3 module components provided by huggingface
      2. *Tesseract* – OCR application that will extract features – used by LayoutLMV3 at backend
      3. *Pillow* – Image handling
      4. *Numpy*
      5. *Pandas*
      6. *Datasets* – required to create data loaders,
      7. *Torch* – deep learning module for python
      8. *Transformers* - Transformers provides thousands of pretrained models to perform tasks on different modalities such as text, vision, and audio.
   2. **Identify Tasks requirements**
      1. Environment variable for image name (inference)
      2. Mounting -> (Training Data Image Directory)
      3. Successful mount of image directory on image shown below
      4. Docker Build in my case will set up the environment
      5. Train.py and Inference.py will then be called via a third run\_app.py file as call to functions
      6. Although run together for thiscope of this assignement utthe train.py and inference.py have been developedevelopned to be run independently
      7. That is inference.py does not pass model as variable or anything rather picks up the trained model from the persisted object of the model
      8. Train.py run automatically on **docker build** (defined in dockerfile.py to run automatically on docker image build)
      9. Inference.py to run on **docker run** command
   3. **Create Dockerfile**
      1. Main file that is used on the docker build command
      2. Specifies the steps required to construct the required docker image
      3. Define env variables etc. with default values all here
   4. **Build docker image from docker file**
      1. Docker build command for my git
      2. This command scans the Dockerfile provided and executes each step provided in it sequentially

docker build -t dl\_assignment\_px\_kamila\_zaman -f Dockerfile .

* 1. **Sample run of docker image – test run**
     1. Docker Run Command with environment variable value for single image or directory path to be mounted on the declared volume – full command in readme.md

docker run dl\_assignment\_px\_kamila\_zaman python3 run\_app.py

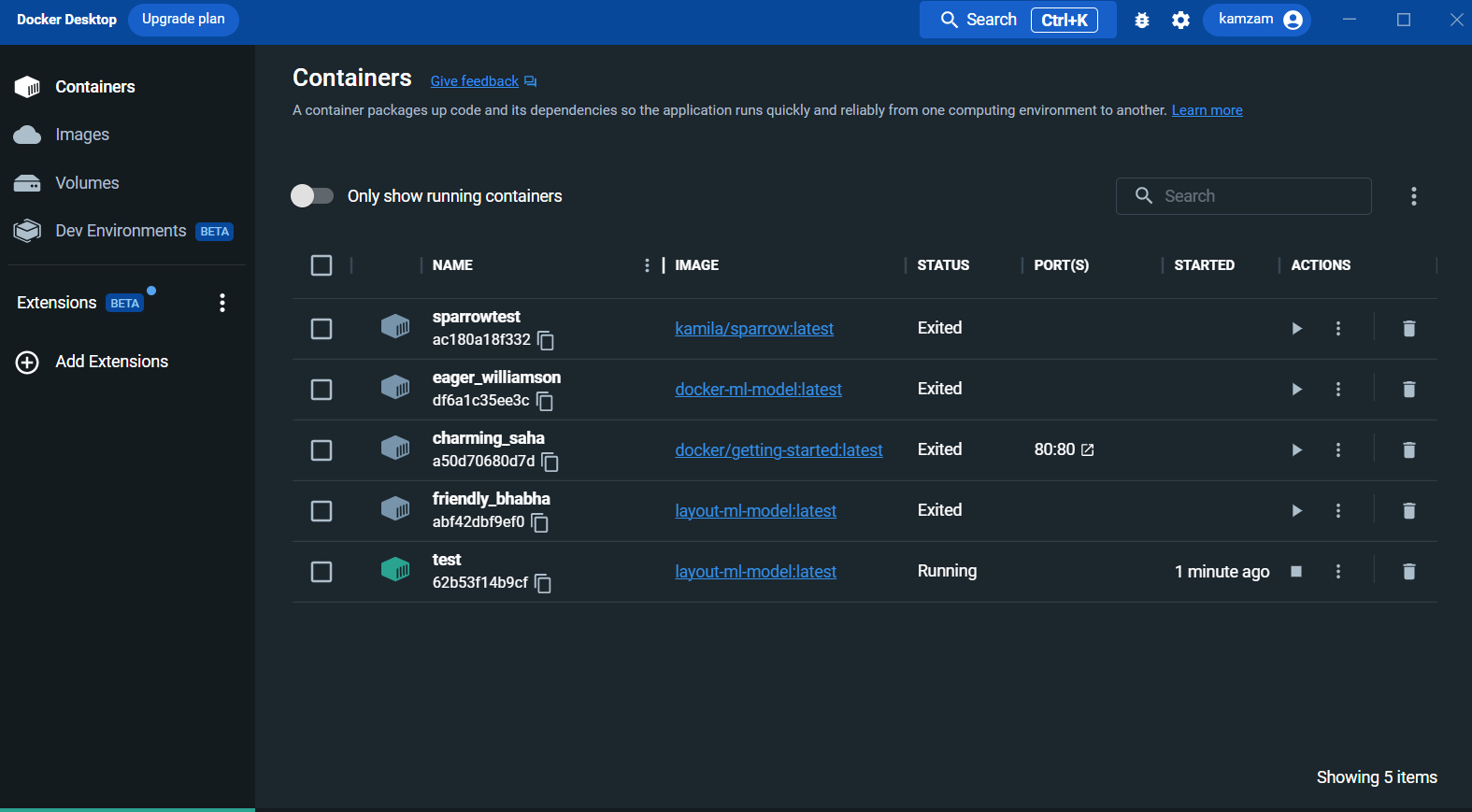
For linux terminal:

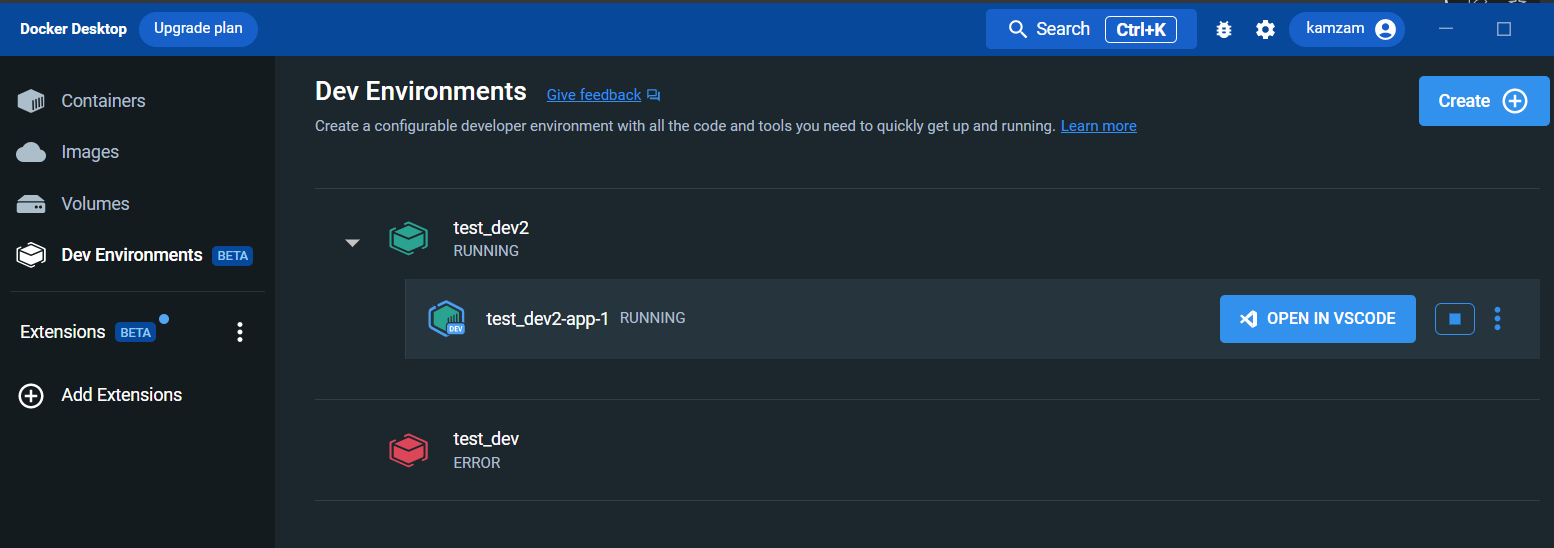
docker run --rm -it -v $(pwd)/dataset:/img\_dir --name= t1 dl\_assignement\_px\_kamila\_zaman:latest

For windows terminal:

docker run --rm -it -v %cd%/dataset:/img\_dir --name= t1 dl\_assignement\_px\_kamila\_zaman:latest

\*Successful test run of above image to start a valid environment container

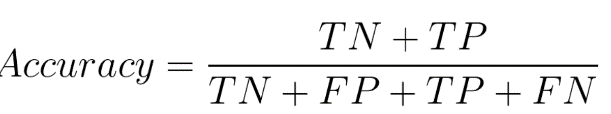
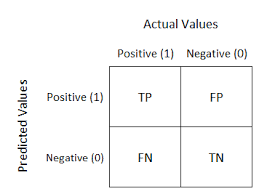


 Dev Environment container successfully created for remote development through VScode IDE

# Accuracy

For accuracy Analysis we compute the accuracy measure in train.py.

**Computed Accuracies:**

1. Training Accuracy – Accuracy
2. Validation Accuracy – Tells us true performance of the model (unseen data by model) but we know true labels hence we can mark performance against the predicted values

**Accuracy & Epochs:**

Currently we used 3 epochs which implies 3 passes of the training data for training of the model.

The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset. One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters.

The right number of epochs depends on the inherent perplexity (or complexity) of your dataset. A good rule of thumb is to start with a value that is 3 times the number of columns in your data. If you find that the model is still improving after all epochs complete, try again with a higher value.

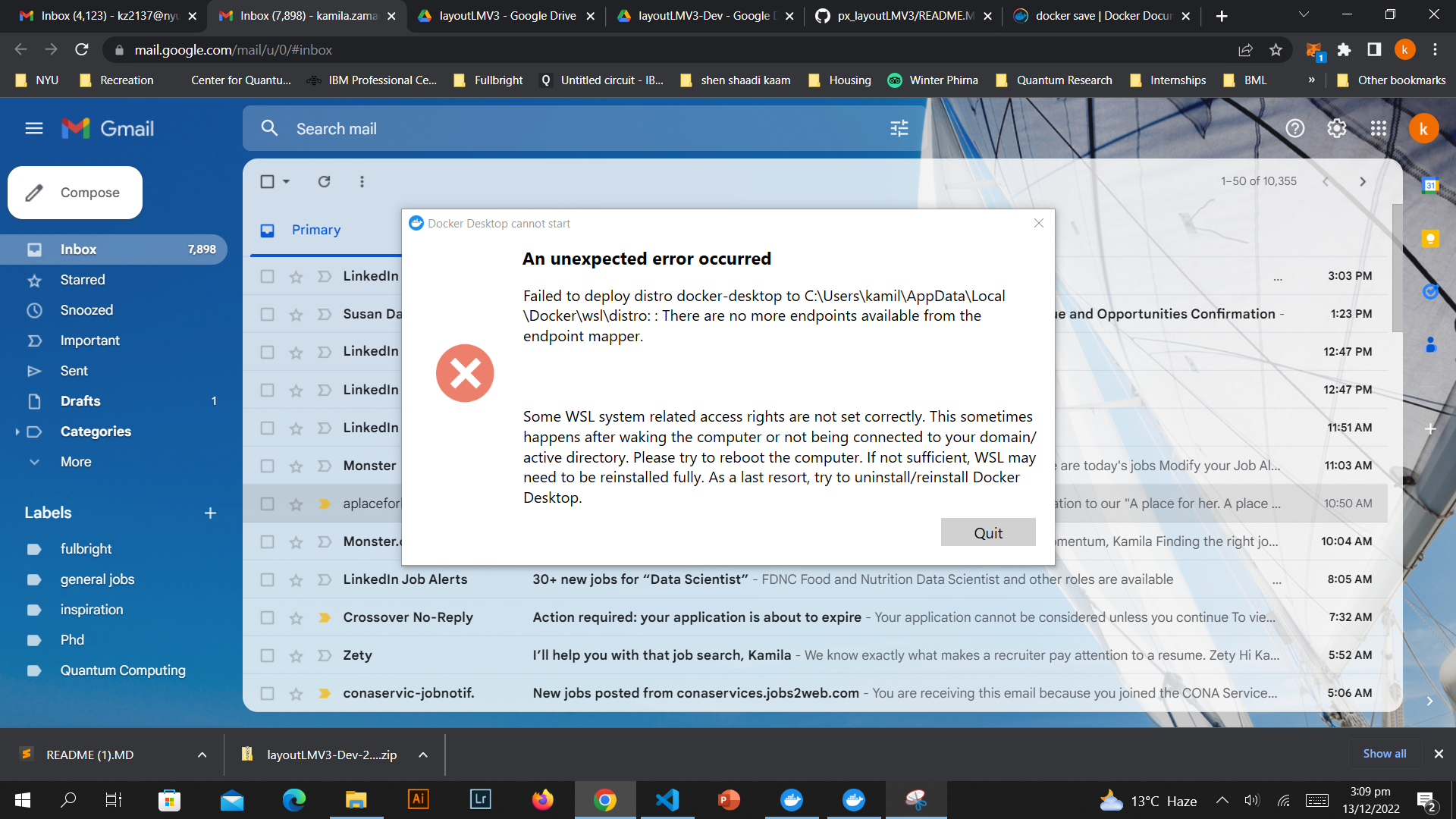
Hence keeping in view our task and the size of our dataset a lower number of epochs is good as we will be able to extract the features for such a small dataset quite reasonably and on the downside, we also shouldn’t be using very high epoch value as it can certainly cause overfitting by tuning to the particular features of the given documents hence making us use the essence of our model training that is generalization.

# GitHub Push elements

* 1. Dataset
  2. Train.py
  3. Inference.py
  4. Docker image
  5. Readme file
  6. ~~Output plots for local training -> accuracy analysis~~
  7. Documentation word file
  8. Dockerfile
  9. Run commands image samples

# Challenges:

* 1. Docker Virtualization and backend (WSL2) requirement compatibility issues in firmware – had to reset BIOS settings accordingly
  2. Unfamiliarity with docker raised numerous challenges but a rewarding experience to have been able to resolve most of them
  3. Docker Memory usage is unreasonably high and inefficient, created bottle necks for my machines - tested colab as well to utilize cloud dev environment did not work as expected – inconsistencies with packaged environment I required with layoutLMV3
  4. Docker StartUp was very slow despite cache cleanup every time – haven’t been able to identify the reason yet, will understand it later
  5. Docker builds for images are time consuming based on packages and vulnerable to unstable network connectivity – slowing progress and testing of the build – built multiple images of the same environment in parallel on two machines
  6. Python’s Transformer module is very sensitive to a lot of dependencies – Numerous conflict on wheels and versions of other required modules, it took me time to resolve all of them with right order and versions for achieving successful installation and run of transformer module
  7. Development within docker container – paid IDE versions allowing easy remote development mostly, opted for VSCode dev-containers extension for remote development on container
  8. Use of Dev- Environment in docker – no good documentation available online to specify docker configuration requirement formats – had to build and connect it with VSCode on test and trial basis myself – insufficient logs and error messages, no error codes or descriptions shared to enable and facilitate debugging
  9. VSCode Dev-Containers for remote docker container development ran into many bugs – is not yet very stable for local directory connection
  10. Environment crash of stable containers at some instances on remote development on vscode.
  11. Limited Dataset – Accuracy affected. Additional data correctly curated should improve performance
  12. Machine Limitations and Network Stability
  13. Resolving Package dependencies was the most difficult task
  14. LayoutLMV3 is a fairly new architecture, not widely used as of yet hence limited online community support and supplementary repos available
  15. Identified issues and ambiguity in published github modules using LayoutLMV3 on official hugging face as tutorial examples – my solution to it was to reach out to the creator, I connected with him on LinkedIn and was able to have my questions answered.
  16. Unable to create tar of final image as docker crashed.
  17. Otherwise For docker image to tar use: docker save command
  18. Then use docker load for running the container from that tar file of image. (Not possible at the moment as my docker image is inaccessible)
  19. Kindly use docker build from dokerfile command provided in readme.md

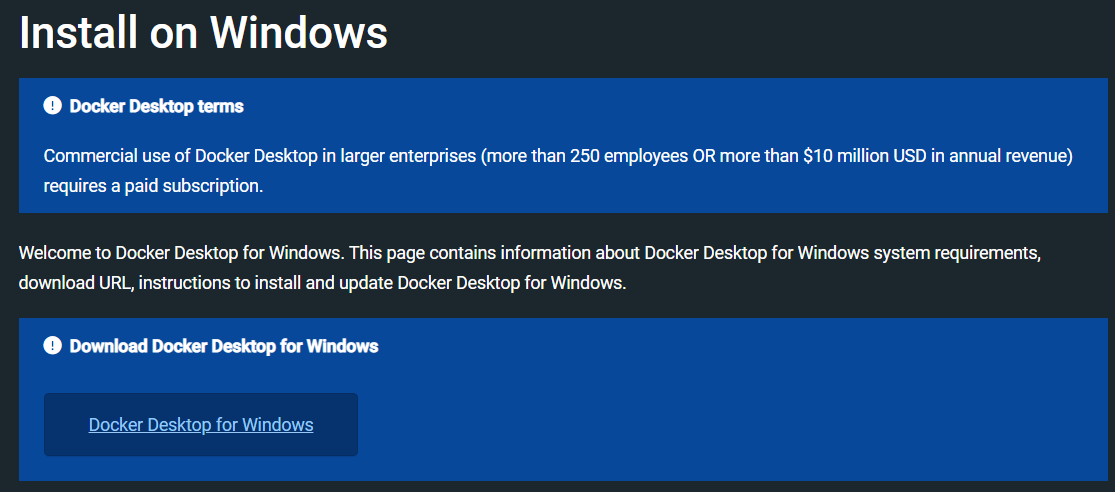


# References:

* 1. <https://huggingface.co/docs/transformers/model_doc/layoutlmv3>
  2. https://towardsdatascience.com/build-and-run-a-docker-container-for-your-machine-learning-model-60209c2d7a7f
  3. <https://github.com/xaviervasques/EEG-letters>
  4. https://github.com/NielsRogge/Transformers-Tutorials/blob/master/LayoutLMv3/Fine\_tune\_LayoutLMv3\_on\_FUNSD\_(HuggingFace\_Trainer).ipynb
  5. https://github.com/katanaml/sparrow
  6. <https://github.com/facebookresearch/detectron2>
  7. <https://towardsdatascience.com/build-and-run-a-docker-container-for-your-machine-learning-model-60209c2d7a7f>
  8. <https://stackoverflow.com/questions/39684974/docker-for-windows-error-hardware-assisted-virtualization-and-data-execution-p>
  9. <https://thegeekpage.com/docker-cannot-enable-hyper-v-how-to-fix/>

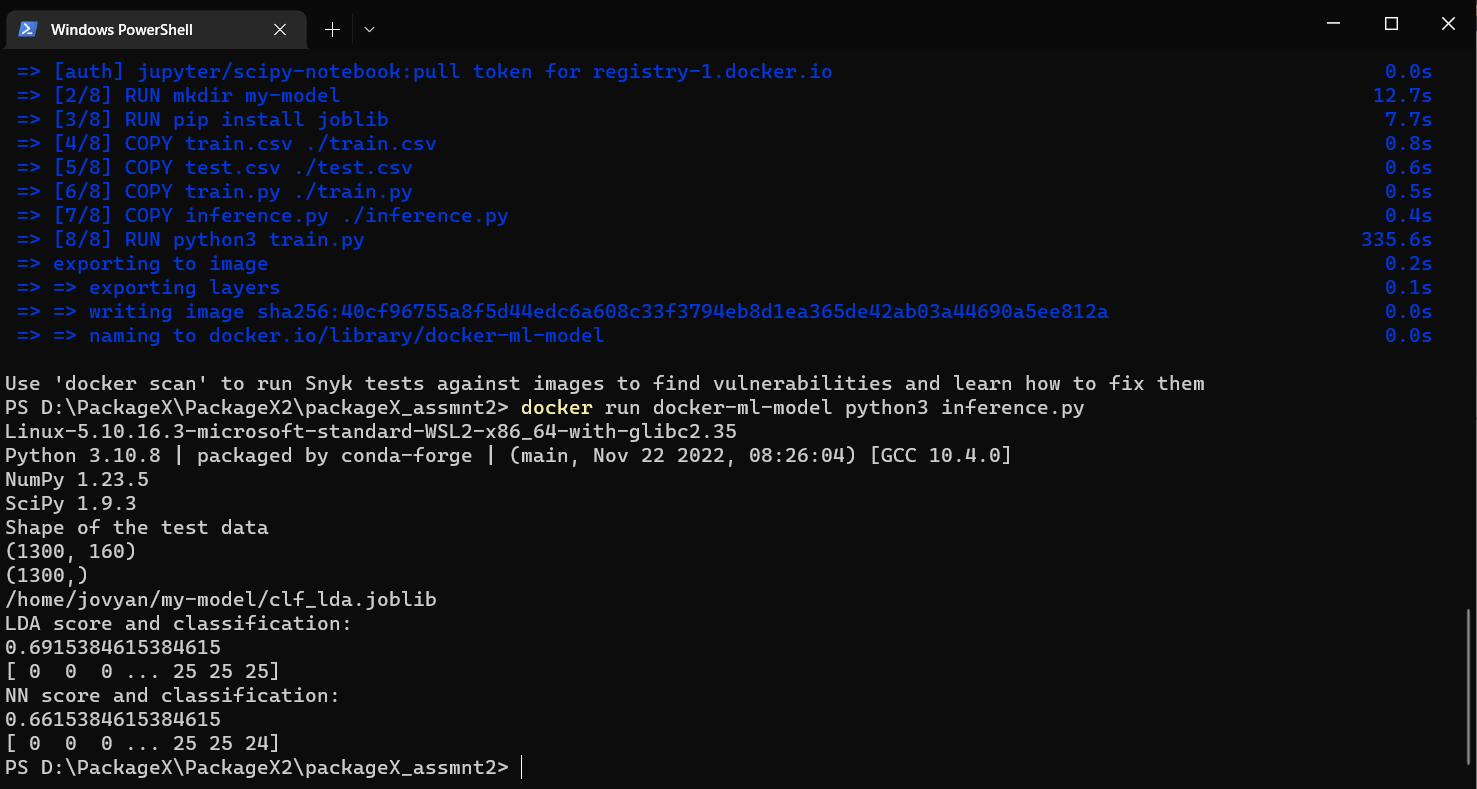
**APPENDIX:**

* + 1. Download and install docker



**Successful Installation**





docker build -t docker-ml-model -f Dockerfile .

docker run docker-ml-model python3 inference.py