CSE 676/B Deep Learning

Paper & Code Presentation and Discussions

Deploying DL models as a Web Server # 5

Authors: Kusha Kumar Dharavath | Adarsh Reddy Bandaru

UBIT: kushakum | adarshre

We know that the world of deep learning is advancing rapidly, bringing new models that can tackle complex tasks across different fields, such as recognizing images, processing human language, and enhancing recommendation engines. For these models to make a real difference, they need to be accessible. That's where deploying them as a web server comes in.

We're planning to deploy our model using a web application using Streamlit. This setup allows for a smooth interaction: users send data to the model via the app, the model processes this data, and then it sends back predictions. This ensures everyone can understand the model without any hassle.

Before we talk about Streamlit, let's make sure our model is ready. We've trained it with PyTorch which gave goodresults on our tests.

Introduction to Streamlit

Streamlit is game changing for data scientists/machine learning engineers by making it super easy to turn complicated Python scripts into easy, interactive web apps. This tool is all about keeping things simple and efficient, perfect for anyone who wants to focus more on their data and less on the aspects of web development.

With Streamlit, you can put together a web app with just a few simple Python commands. It works great with lots of different Python libraries—like NumPy for doing math stuff, Pandas for organising data, and Matplotlib for making charts. This makes Streamlit not just flexible, but also really powerful for building apps that handle lots of data.

What really makes Streamlit different is its ability to write a little code and get a lot done. Its feature that updates your app in real time is great, letting you tweak things as it shows. This is perfect for trying out new things.

But why Streamlit over other REST APIs or web application frameworks?

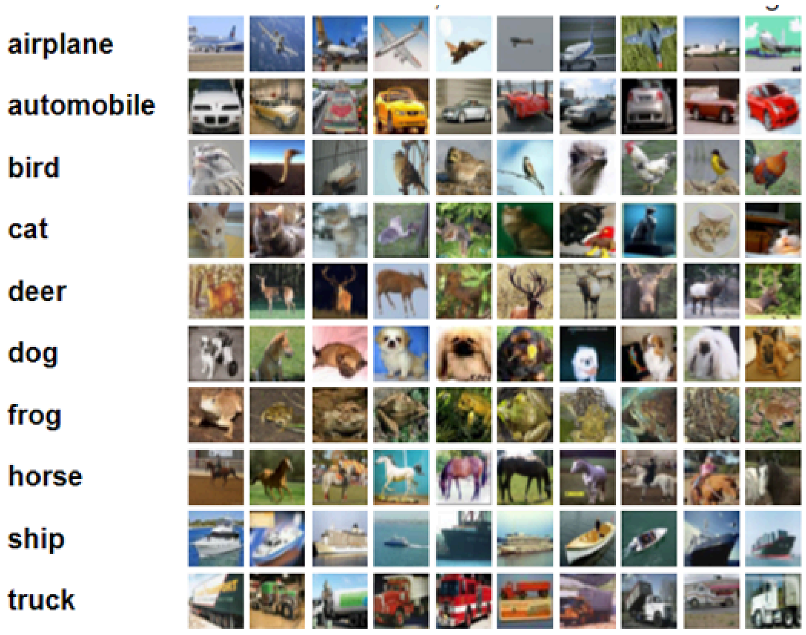
· **Ease of Use**: Streamlit allows the creation of web apps with minimal coding, streamlining the process for data scientists unfamiliar with detailed web development.

- **Rapid Prototyping**: Updates to applications happen in real-time, facilitating quick iterations and interactive experimentation.
- **Direct Integration**: Works seamlessly with major Python data science and machine learning libraries, simplifying the setup process compared to traditional web apps.
- **Less Code:** Requires significantly less code for handling backend operations like requests and responses, as Streamlit automates these aspects.
- **Built-in Interactivity**: Includes interactive components like sliders and buttons, which are more complex to implement in traditional web frameworks.
- **Easy Deployment**: Streamlit apps are easy to deploy and share, even on cloud platforms, with less configuration required than typical web applications.
- **Ideal for Non-web Developers**: It lowers barriers for those primarily skilled in data science, not web development, making it easier to build and showcase interactive web interfaces.

Model and Training:

For training purpose, we chose CIFA-10 dataset.

The **CIFAR-10 dataset** is a widely used benchmark dataset in the field of computer vision and machine learning. It is a collection of 60,000 color images that are categorized into 10 different classes with each class representing a different type of object. The classes are: Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, Truck.



The dataset was created by researchers at the University of Toronto and the Canadian Institute for Advanced Research (CIFAR), and it has become a standard resource for evaluating the performance of image classification algorithms. Each image in the dataset has a size of 32x32 pixels and is represented in the RGB color space. The dataset is divided into two parts: a training set consisting of 50,000 images and a test set containing 10,000 images. The images in the dataset are low-resolution and represent a diverse range of real-world objects.

Data pre-processing:

Converted the images from a Python Imaging Library format to a PyTorch tensor and Normalized the image tensor by setting its mean and standard deviation.

Model Architecture:

For our dataset we used CNN model which is designed for image classification tasks.

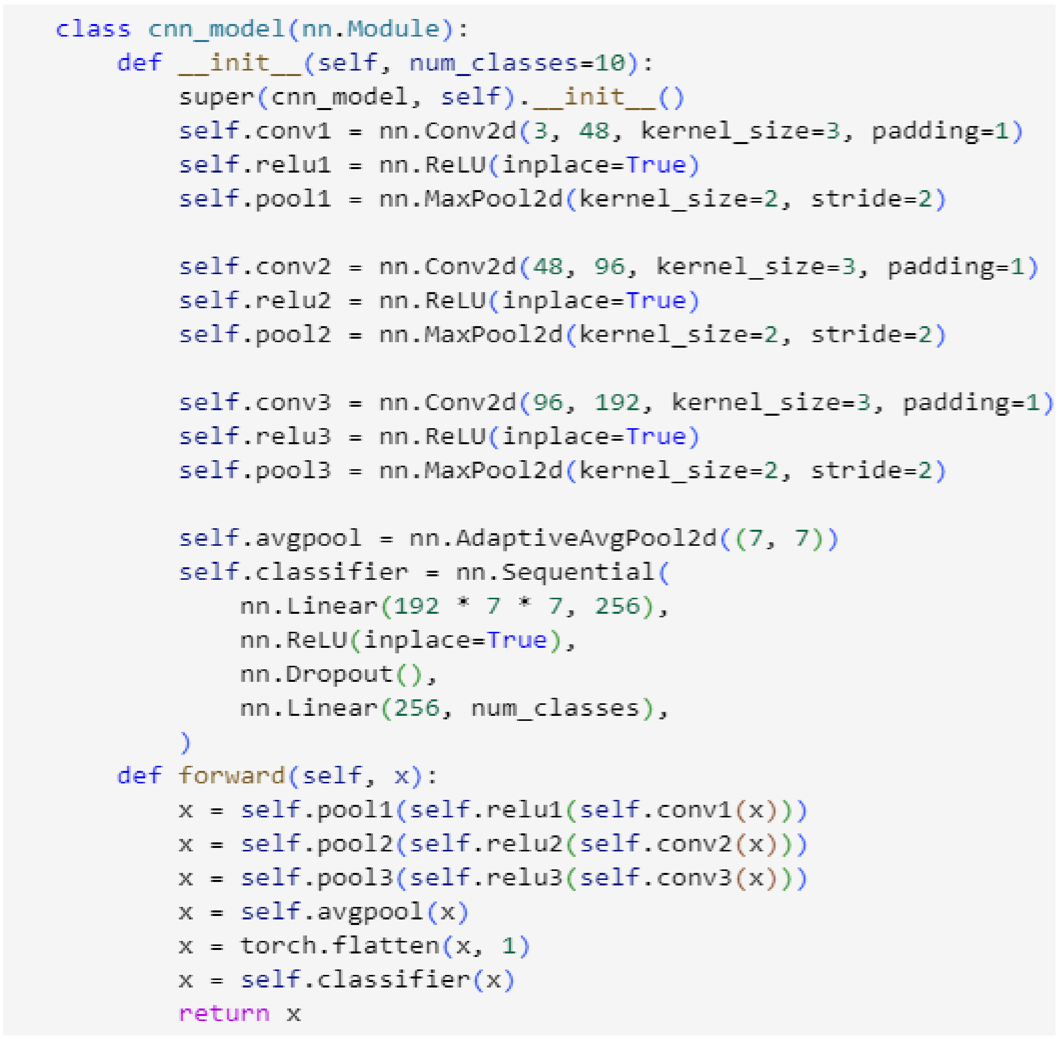
Below is the architecture of the model that we defined.

Our model features three convolutional layers (conv1, conv2, conv3). Each layer uses ReLU activation functions to add non-linearity, enabling the model to learn complex patterns

Each convolutional layer is followed by a max pooling layer (pool1, pool2, pool3). These layers reduce the size of the feature maps, which helps decrease computation and prevent overfitting.

Then an adaptive average pooling layer (avgpool) used, which resizes the feature maps to 7x7. This standardizes the output size, preparing it for classification.

The network includes a sequence of fully connected layers at the end. The feature maps are first flattened and then processed through a linear layer with 256 units, followed by a ReLU activation and a dropout layer to help with generalization. The final linear layer outputs to the 10 classes of the CIFAR-10 dataset.



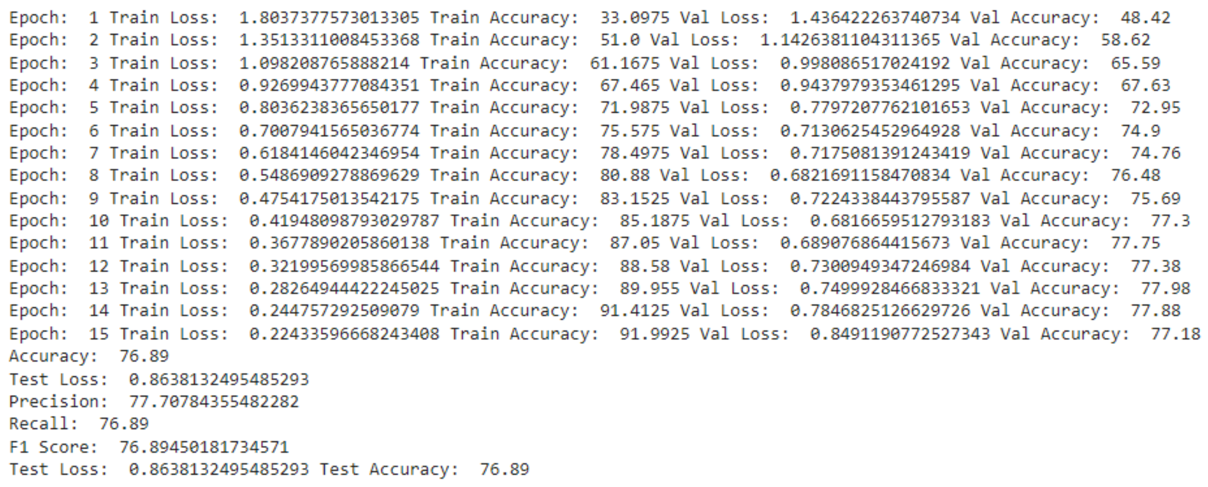
```
class cnn_model(nn.Module):
    def __init__(self, num_classes=10):
        super(cnn_model, self).__init__()
        self.conv1 = nn.Conv2d(3, 48, kernel_size=3, padding=1)
        self.relu1 = nn.ReLU(inplace=True)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv2 = nn.Conv2d(48, 96, kernel_size=3, padding=1)
        self.relu2 = nn.ReLU(inplace=True)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv3 = nn.Conv2d(96, 192, kernel_size=3, padding=1)
        self.relu3 = nn.ReLU(inplace=True)
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.avgpool = nn.AdaptiveAvgPool2d((7, 7))
        self.classifier = nn.Sequential(
            nn.Linear(192 * 7 * 7, 256),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(256, num_classes),
        )
    def forward(self, x):
        x = self.pool1(self.relu1(self.conv1(x)))
        x = self.pool2(self.relu2(self.conv2(x)))
        x = self.pool3(self.relu3(self.conv3(x)))
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

After constructing this model, we have trained the model on the CIFAR-10 dataset. For training the model we have used parameters like learning rate=0.1 and crossEntropyLoss function, SGD optimizer. Results of the model after training the model for 15 epochs are below:

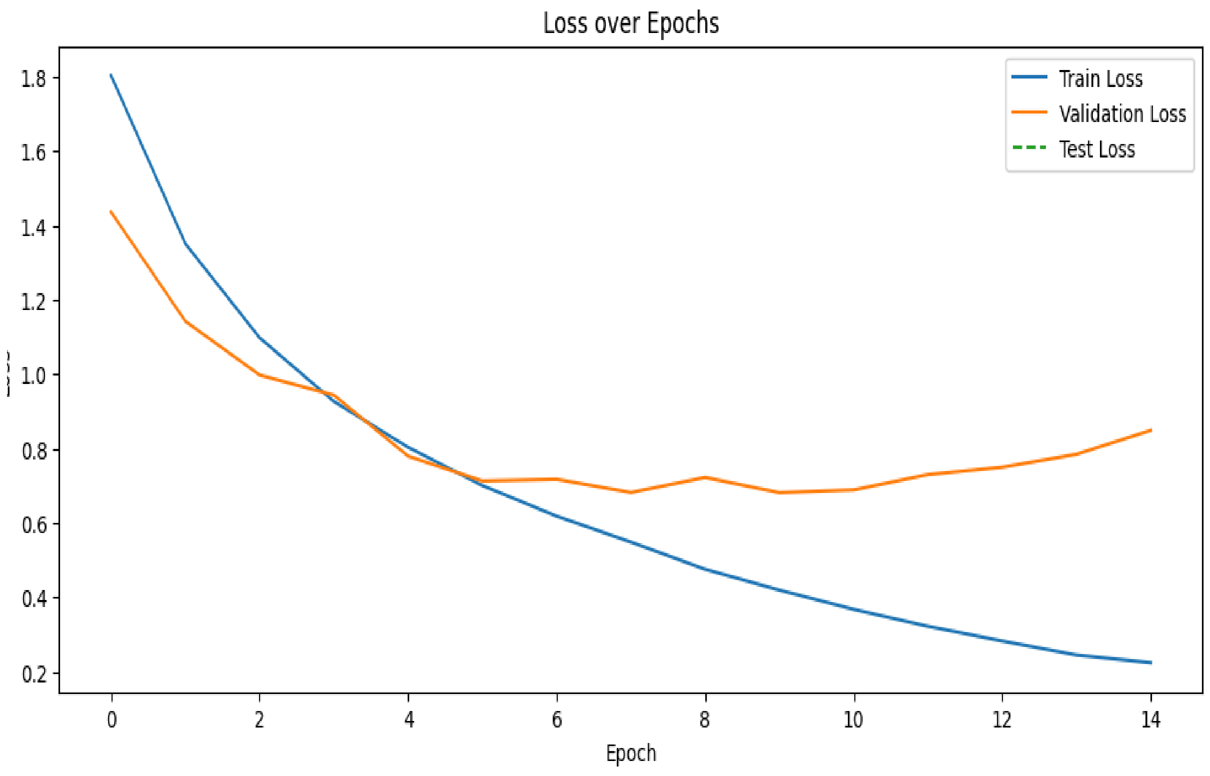
Train Loss: 0.22
Train Accuracy: 91.99
Val Loss: 0.84
Val Accuracy: 77.18
Test Loss: 0.86
Test Accuracy: 76.89

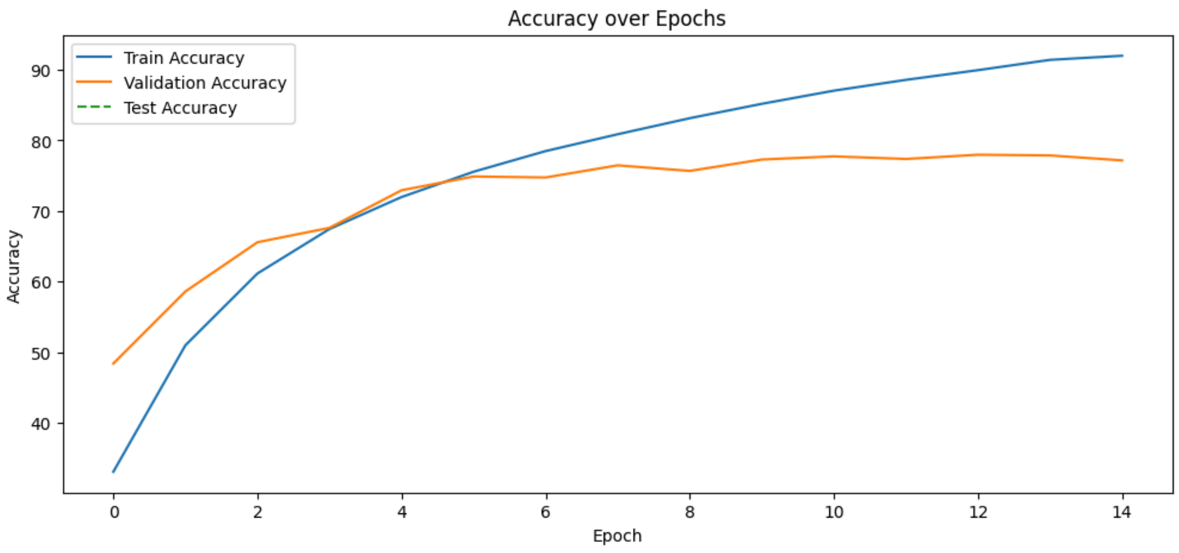
Evaluation Metrics:
Precision: 77.70
Recall: 76.89
F1 Score: 76.89

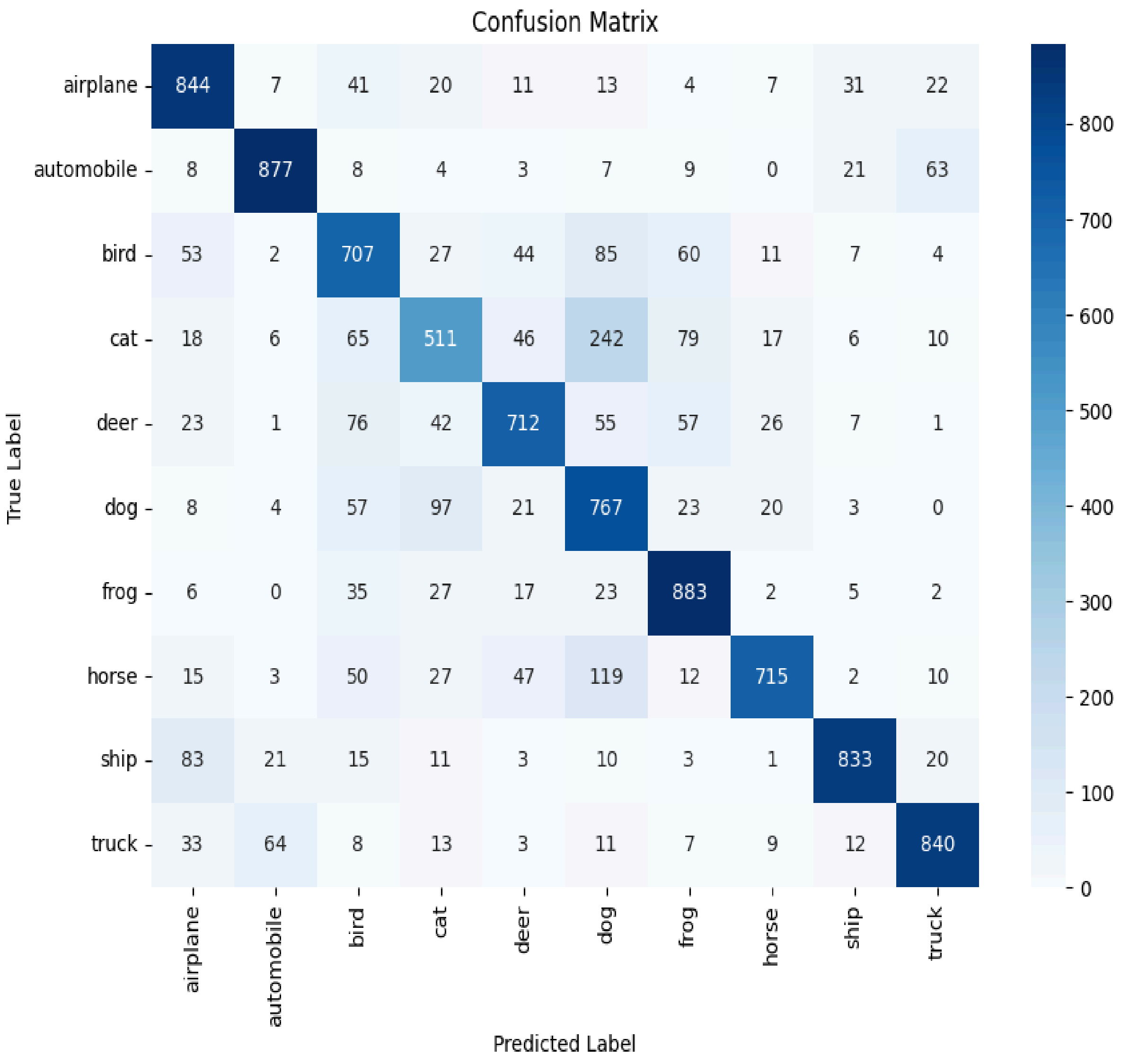
Below are the screenshots of epochs, accuracy & loss plots and confusion matrix



```
Epoch:  1 Train Loss:  1.8037377573013305 Train Accuracy:  33.0975 Val Loss:  1.436422263740734 Val Accuracy:  48.42
Epoch:  2 Train Loss:  1.3513311008453368 Train Accuracy:  51.0 Val Loss:  1.1426381104311365 Val Accuracy:  58.62
Epoch:  3 Train Loss:  1.098208765888214 Train Accuracy:  61.1675 Val Loss:  0.998086517024192 Val Accuracy:  65.59
Epoch:  4 Train Loss:  0.9269943777084351 Train Accuracy:  67.465 Val Loss:  0.9437979353461295 Val Accuracy:  67.63
Epoch:  5 Train Loss:  0.8036238365650177 Train Accuracy:  71.9875 Val Loss:  0.7797207762101653 Val Accuracy:  72.95
Epoch:  6 Train Loss:  0.7007941565036774 Train Accuracy:  75.575 Val Loss:  0.7130625452964928 Val Accuracy:  74.9
Epoch:  7 Train Loss:  0.6184146042346954 Train Accuracy:  78.4975 Val Loss:  0.7175081391243419 Val Accuracy:  74.76
Epoch:  8 Train Loss:  0.5486909278869629 Train Accuracy:  80.88 Val Loss:  0.6821691158470834 Val Accuracy:  76.48
Epoch:  9 Train Loss:  0.4754175013542175 Train Accuracy:  83.1525 Val Loss:  0.7224338443795587 Val Accuracy:  75.69
Epoch:  10 Train Loss:  0.41948098793029787 Train Accuracy:  85.1875 Val Loss:  0.6816659512793183 Val Accuracy:  77.3
Epoch:  11 Train Loss:  0.3677890205860138 Train Accuracy:  87.05 Val Loss:  0.689076864415673 Val Accuracy:  77.75
Epoch:  12 Train Loss:  0.32199569985866544 Train Accuracy:  88.58 Val Loss:  0.7300949347246984 Val Accuracy:  77.38
Epoch:  13 Train Loss:  0.28264944422245025 Train Accuracy:  89.955 Val Loss:  0.7499928466833321 Val Accuracy:  77.98
Epoch:  14 Train Loss:  0.244757292509079 Train Accuracy:  91.4125 Val Loss:  0.7846825126629726 Val Accuracy:  77.88
Epoch:  15 Train Loss:  0.22433596668243408 Train Accuracy:  91.9925 Val Loss:  0.8491190772527343 Val Accuracy:  77.18
Accuracy:  76.89
Test Loss:  0.8638132495485293
Precision:  77.70784355482282
Recall:  76.89
F1 Score:  76.89450181734571
Test Loss:  0.8638132495485293 Test Accuracy:  76.89
```



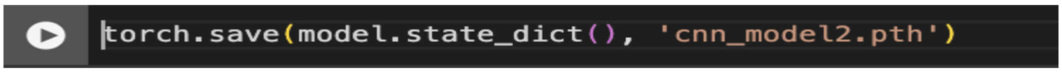




	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	844	7	41	20	11	13	4	7	31	22
automobile	8	877	8	4	3	7	9	0	21	63
bird	53	2	707	27	44	85	60	11	7	4
cat	18	6	65	511	46	242	79	17	6	10
deer	23	1	76	42	712	55	57	26	7	1
dog	8	4	57	97	21	767	23	20	3	0
frog	6	0	35	27	17	23	883	2	5	2
horse	15	3	50	27	47	119	12	715	2	10
ship	83	21	15	11	3	10	3	1	833	20
truck	33	64	8	13	3	11	7	9	12	840

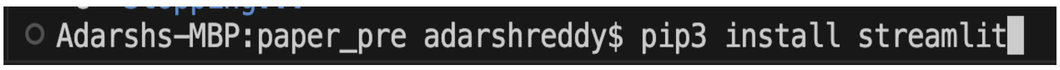
Deploying the model in Streamlit:

First step is to have the trained model saved.



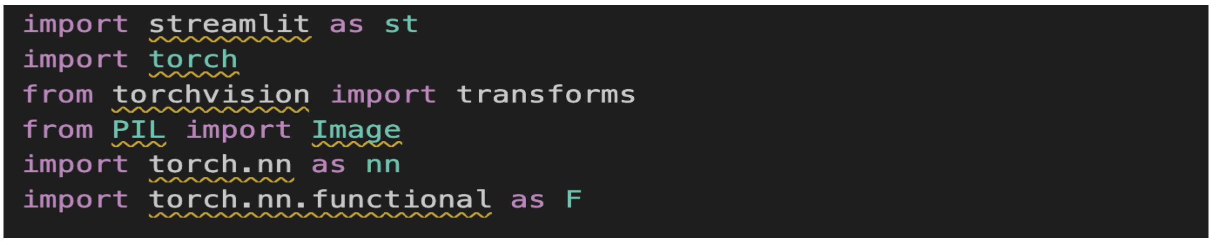
```
torch.save(model.state_dict(), 'cnn_model2.pth')
```

- Now, set up your streamlit environment. This part is easy in comparison with other web frameworks.



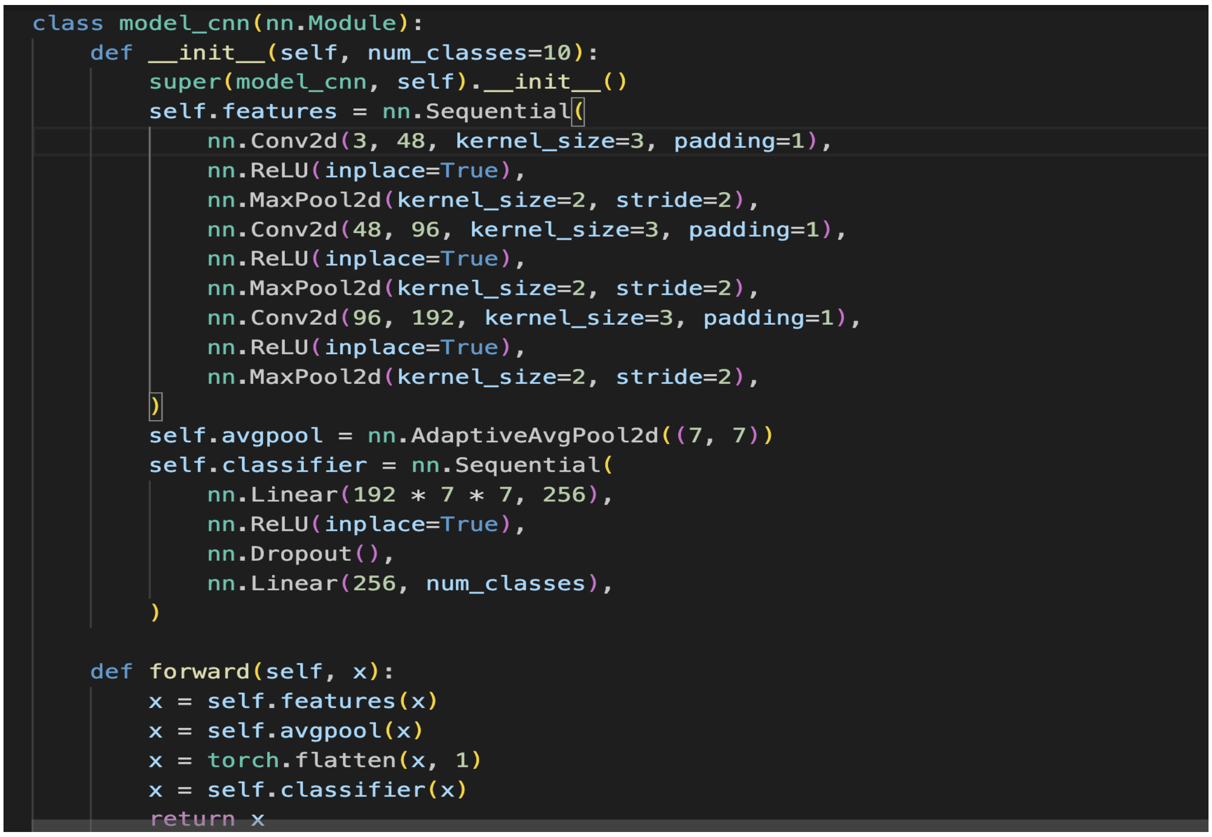
```
Adarshs-MBP:paper_pre adarshreddy$ pip3 install streamlit
```

- Import necessary libraries

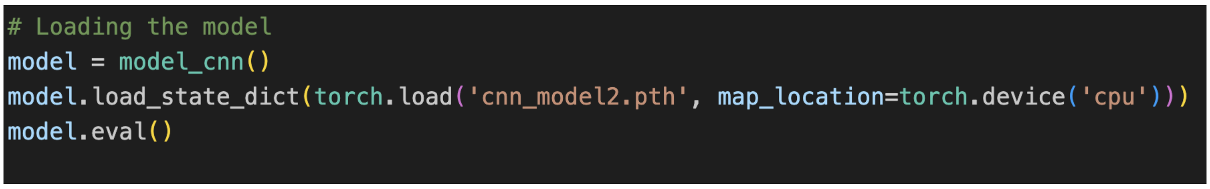


```
import streamlit as st
import torch
from torchvision import transforms
from PIL import Image
import torch.nn as nn
import torch.nn.functional as F
```

- Load the saved model and ensure you define the model again before loading.

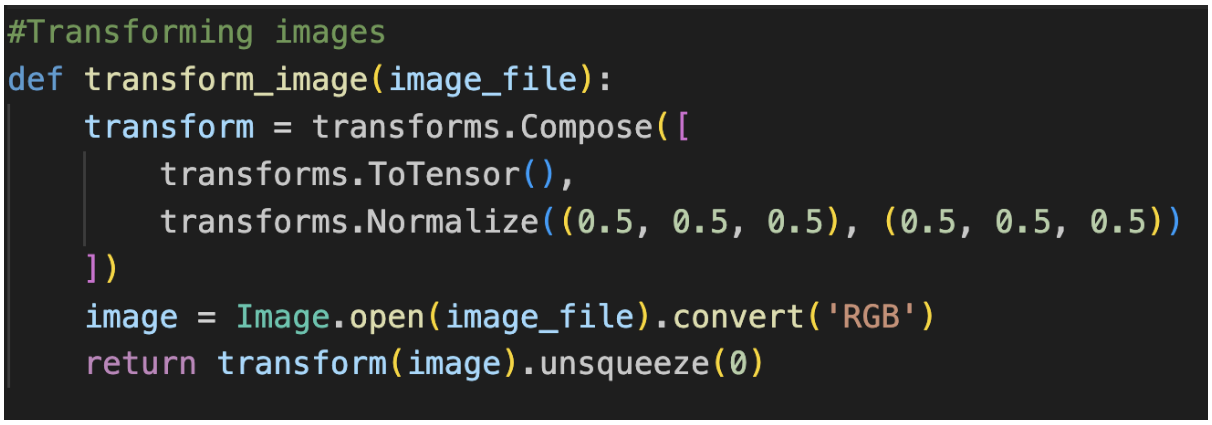


```
class model_cnn(nn.Module):
    def __init__(self, num_classes=10):
        super(model_cnn, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 48, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(48, 96, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(96, 192, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((7, 7))
        self.classifier = nn.Sequential(
            nn.Linear(192 * 7 * 7, 256),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(256, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```



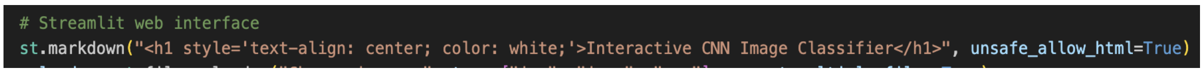
```
# Loading the model
model = model_cnn()
model.load_state_dict(torch.load('cnn_model2.pth', map_location=torch.device('cpu')))
model.eval()
```

- Define a function for preprocessing the input image that we get from the user in the accepted format of the model.



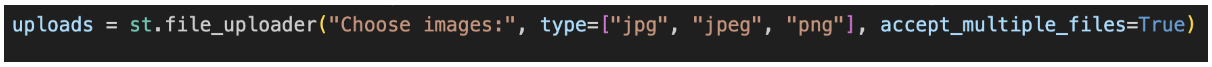
```python
#Transforming images
def transform_image(image_file):
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])
    image = Image.open(image_file).convert('RGB')
    return transform(image).unsqueeze(0)
```

- Now, to create the title one can simply use st.markdown(), but streamlit is left centred. To customise the formatting of content, we can use HTML script as streamlit supports it. Here we are defining our title and centering it.



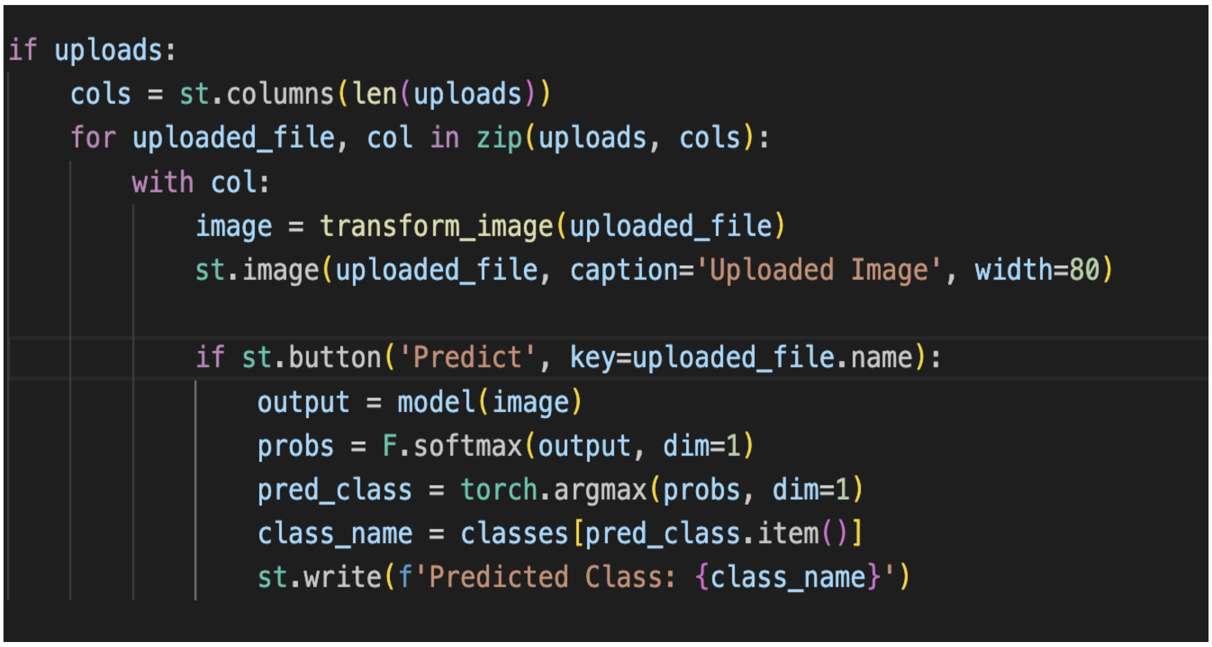
```python
# Streamlit web interface
st.markdown("<h1 style='text-align: center; color: white;'>Interactive CNN Image Classifier</h1>", unsafe_allow_html=True)
```

- We then use an uploader as a direct feature and allow the user to upload one or multiple pictures at the same time.



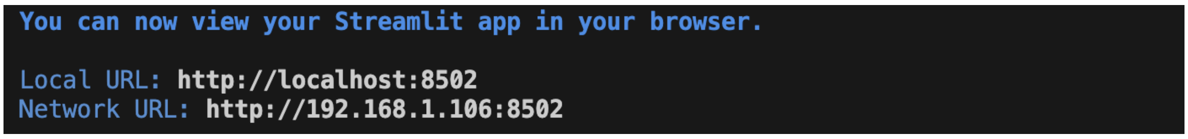
```python
uploads = st.file_uploader("Choose images:", type=["jpg", "jpeg", "png"], accept_multiple_files=True)
```

- Then we check if any files are uploaded and if there are any, we continue to loop through each uploaded image to preprocess and use the model to predict which class it belongs to.



```python
if uploads:
    cols = st.columns(len(uploads))
    for uploaded_file, col in zip(uploads, cols):
        with col:
            image = transform_image(uploaded_file)
            st.image(uploaded_file, caption='Uploaded Image', width=80)

            if st.button('Predict', key=uploaded_file.name):
                output = model(image)
                probs = F.softmax(output, dim=1)
                pred_class = torch.argmax(probs, dim=1)
                class_name = classes[pred_class.item()]
                st.write(f'Predicted Class: {class_name}')
```

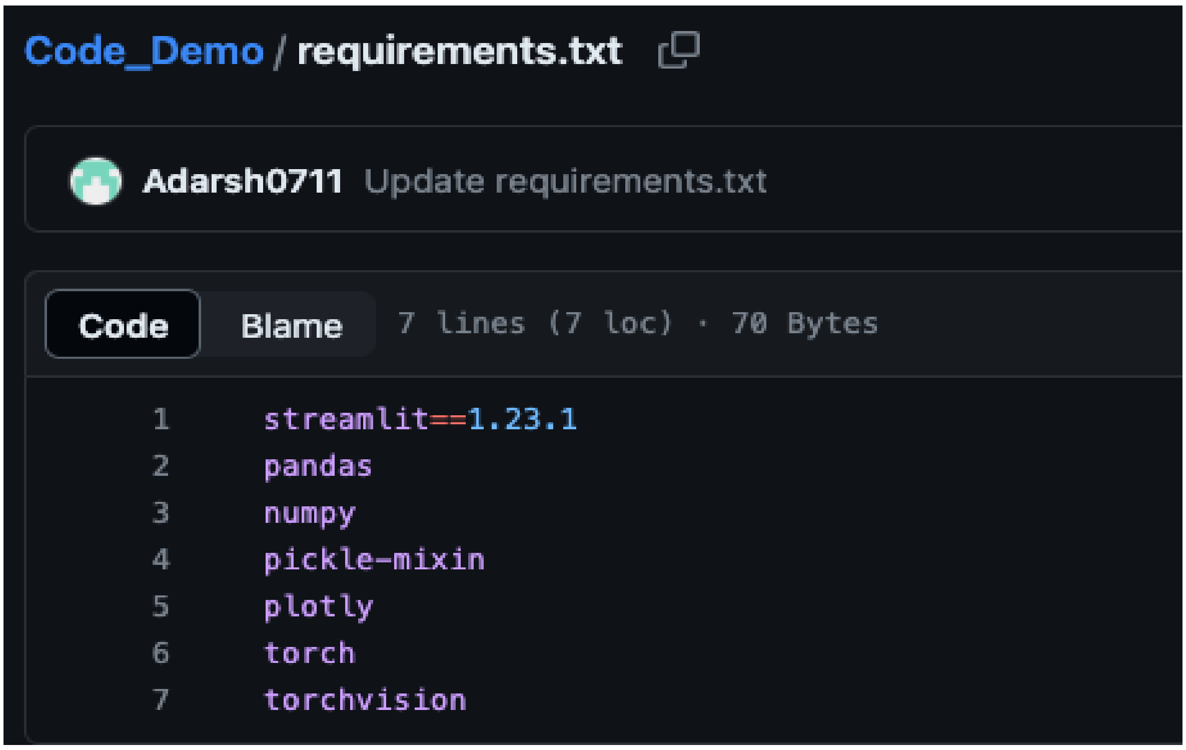
- To run the file locally, we can simply use the command “streamlit run code.py”. You can view the website in the below urls.



```
You can now view your Streamlit app in your browser.

Local URL: http://localhost:8502
Network URL: http://192.168.1.106:8502
```

To Deploy the application as a streamlit App follow these steps:

- Create an account in streamlit https://streamlit.io/ using your github profile.
- Create a repository and upload the code.py file that has the above discussed segments, the saved mode pth file.
- Along with the above file we should also create a requirements.txt file with listing the dependencies we are using in the code.py file.



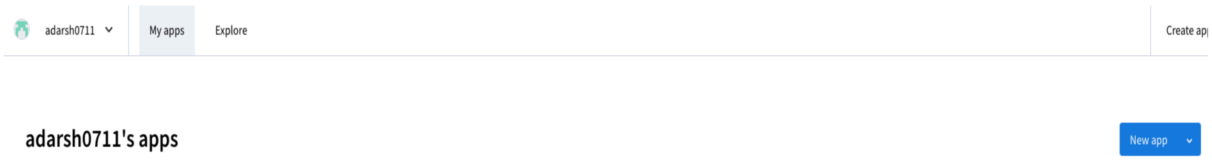
Code_Demo / requirements.txt

Adarsh0711 Update requirements.txt

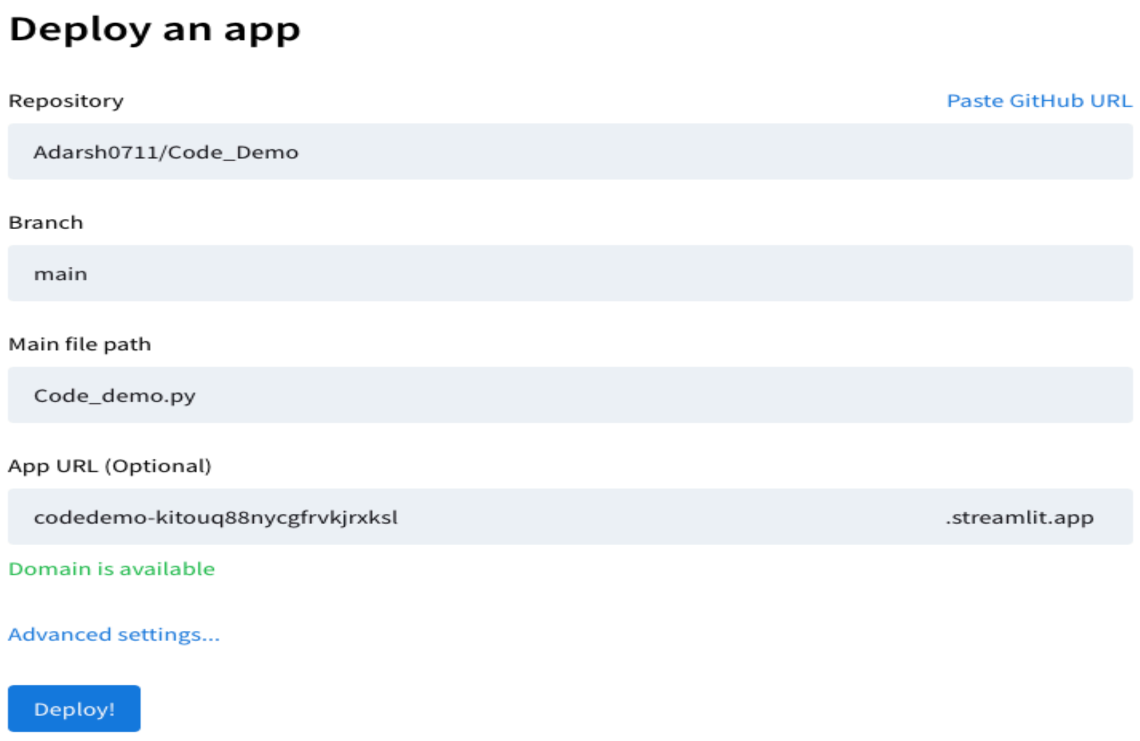
7 lines (7 loc) · 70 Bytes

```
streamlit==1.23.1
pandas
numpy
pickle-mixin
plotly
torch
torchvision
```

- Post this, click on the create app or new app button.



- We now have to select the repository and fill in the rest of the details and click deploy.


Deploy an app

Repository: Adarsh0711/Code_Demo

Branch: main

Main file path: Code_demo.py

App URL (Optional): codedemo-kitouq88nycgfrvkjrxksl .streamlit.app

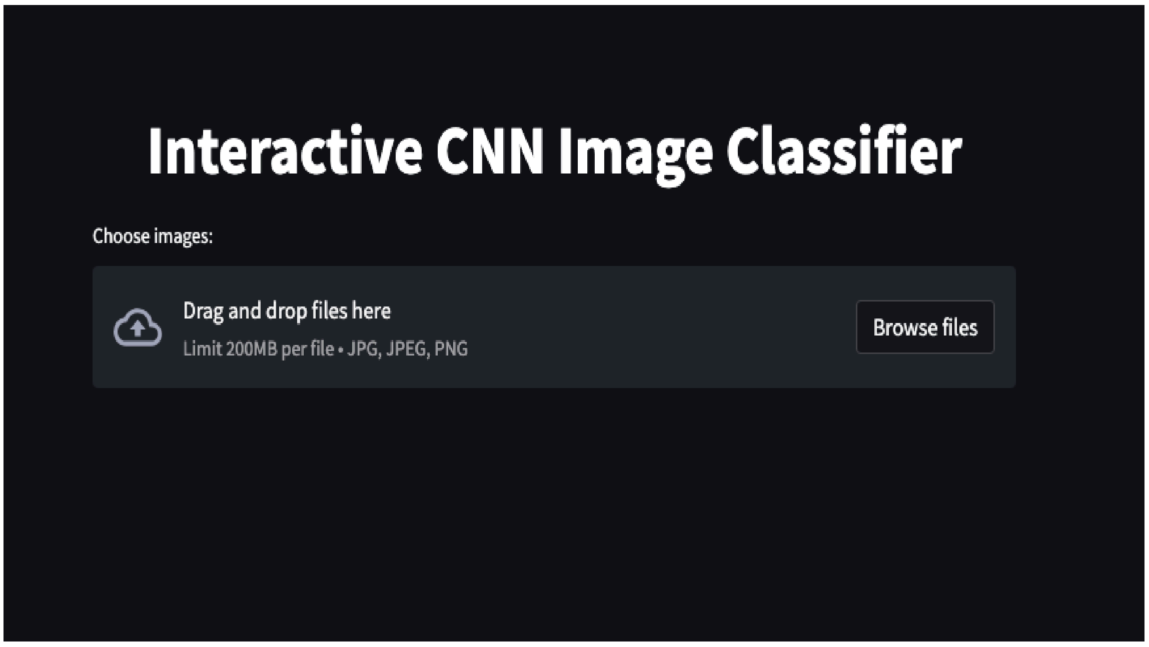
Domain is available

- You're all set!!!

Website Link and visuals:

https://code-demo.streamlit.app/

- Once you go to the above link, you will see the option to upload images.

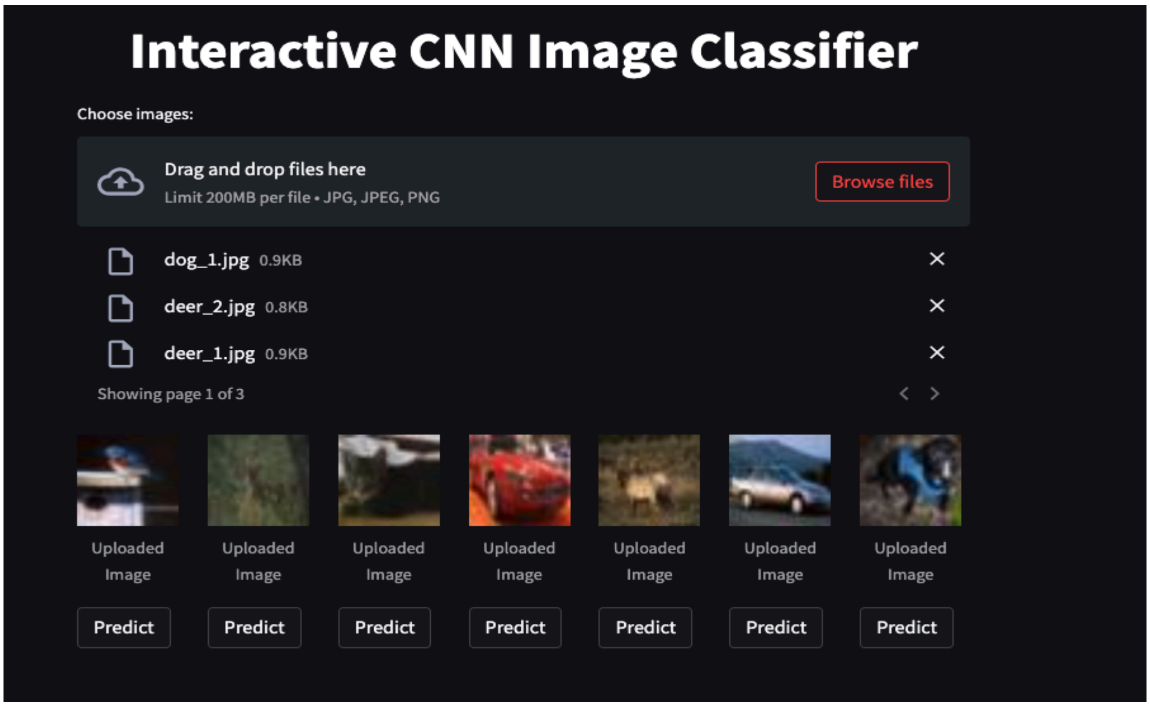

Interactive CNN Image Classifier

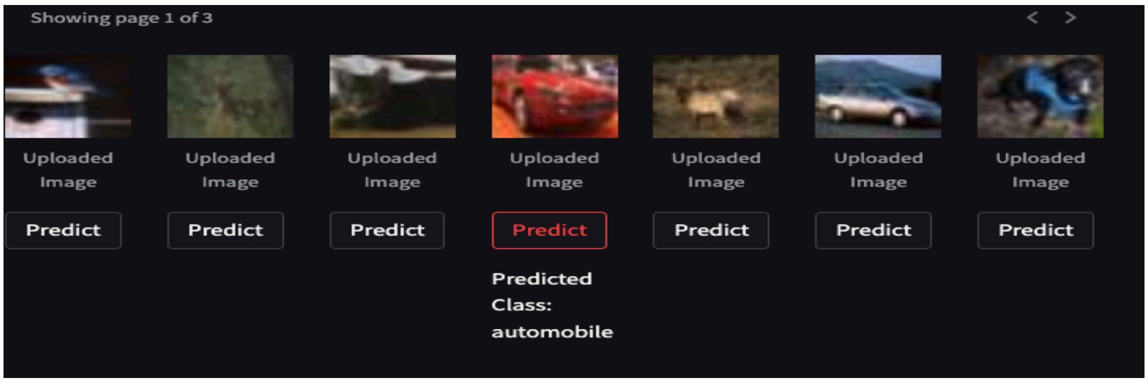
Choose images:

Drag and drop files here

Limit 200MB per file • JPG, JPEG, PNG

- You will have the option to upload a single image or multiple. Once done click on the predict button under each image to check the predictions.





Predicted Class: automobile

Task:
Using the Streamlit web app link, open the web app and upload various CIFAR-10 images to feel how the model is classifying the images. You can upload single or multiple images at a time and see the results. Attach screenshots as part of your submission.

References:

- Deep Learning Assignment 0 submission by Adarsh Reddy Bandaru
- Deep Learning Assignment 1 submission by Adarsh Reddy Bandaru
- https://share.streamlit.io/
- https://github.com/Adarsh0711/Code_Demo
- https://www.cs.toronto.edu/~kriz/cifar.html
- https://code-demo.streamlit.app/