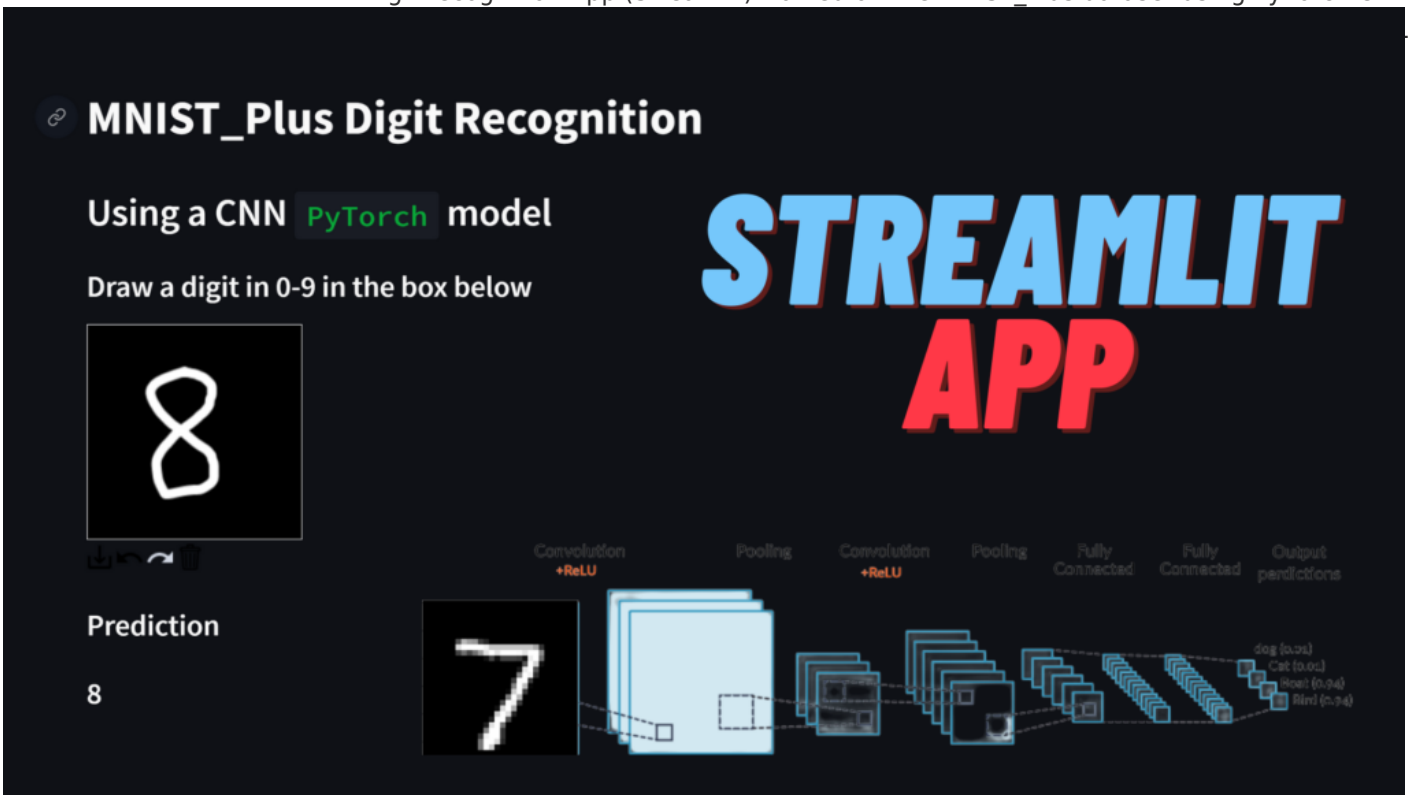


In my previous blog post, I shared with you a streamlit app that recognizes handwritten digits entered by the user through mouse/touch input. The app utilized a Convolutional Neural Network (CNN) trained on the MNIST dataset using the PyTorch machine learning library from Facebook.

However, the recognition had a few flaws. For example, in the case of a 7, the model predicted it to be 1 sometimes, and conversely, sometimes the 1 was predicted to be 7.

There were also some very rare issues in the prediction of other digits as well.

This prompted me to write my own digits to extend the MNIST dataset. I added 225 handwritten samples for each digit (2250 in total) for the training for the CNN. I am calling this extended dataset as MNIST_Plus and it is available on GitHub (https://github.com/manassharma07/MNIST-PLUS).

After, training the CNN on MNIST_Plus dataset for 9 epochs using the Categorical Cross Entropy loss and a batch size of 200, the recognition of the digits 1, and 7 was significantly improved.

You can test out the performance of the CNN network yourself using the app below:

Streamlit App

https://manassharma07-mnist-plus-mnist-plus-cnn-app-xmscba.streamlitapp.com/

Your browser does not support iframes.

The app above is quite neat and also shows the what the prediction process looks like.

The user's handwritten input is captured in a 200×200 pixel box, which is then converted to an image. Subsequently, image processing is done to find a rectangle that completely encompasses the digit blob. Then this rectangular crop of the user input is further processed. Firstly it is converted to grayscale, then resized to a 22×22 image using BILINEAR interpolation. Then a padding of 3 pixels is applied on all the sides of the image to get a 28×28 image. This image still has pixel values in the range of 0-255. These are then normalized by dividing by 255. Then the pixel values are standardized by using the mean and standard deviation of the training MNIST_Plus

dataset. Finally, the processed image is passed to the CNN model to make a prediction.

Source code for the app

https://github.com/manassharma07/MNIST-PLUS/blob/main/mnist_plus_CNN_app.py

```
from streamlit_drawable_canvas import st_canvas
import streamlit as st
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
import torch
import cv2
import torchvision

st.write('# MNIST_Plus Digit Recognition')
st.write('## Using a CNN `PyTorch` model')

Network = torch.load('model_torch_MNIST_plus_CNN_98_5_streamlit.chk')


st.write('### Draw a digit in 0-9 in the box below')
# Specify canvas parameters in application
stroke_width = st.sidebar.slider("Stroke width: ", 1, 25, 9)

realtime_update = st.sidebar.checkbox("Update in realtime", True)

# Create a canvas component
canvas_result = st_canvas(
    fill_color="rgba(255, 165, 0, 0.3)",  # Fixed fill color with some opacity
    stroke_width=stroke_width,
    stroke_color='#FFFFFF',
    background_color='#000000',
    #background_image=Image.open(bg_image) if bg_image else None,
    update_streamlit=realtime_update,
    height=200,
    width=200,
    drawing_mode='freedraw',
    key="canvas",
)

# Do something interesting with the image data and paths
if canvas_result.image_data is not None:

    # st.write('### Image being used as input')
    # st.image(canvas_result.image_data)
    # st.write(type(canvas_result.image_data))
    # st.write(canvas_result.image_data.shape)
    # st.write(canvas_result.image_data)
    # im = Image.fromarray(canvas_result.image_data.astype('uint8'), mode="RGBA")
    # im.save("user_input.png", "PNG")
```

```
    # Get the numpy array (4-channel RGBA 100,100,4)
    input_numpy_array = np.array(canvas_result.image_data)
    # Get the RGBA PIL image
    input_image = Image.fromarray(input_numpy_array.astype('uint8'), 'RGBA')
    input_image.save('user_input.png')
    # Convert it to grayscale
    input_image_gs = input_image.convert('L')
    input_image_gs_np = np.asarray(input_image_gs.getdata()).reshape(200,200)
    # st.write('### Image as a grayscale Numpy array')
    # st.write(input_image_gs_np)
    # Create a temporary image for opencv to read it
    input_image_gs.save('temp_for_cv2.jpg')
    image = cv2.imread('temp_for_cv2.jpg', 0)
    # Start creating a bounding box
    height, width = image.shape
    x,y,w,h = cv2.boundingRect(image)


    # Create new blank image and shift ROI to new coordinates
    ROI = image[y:y+h, x:x+w]
    mask = np.zeros([ROI.shape[0]+10,ROI.shape[1]+10])
    width, height = mask.shape
#     print(ROI.shape)
#     print(mask.shape)
    x = width//2 - ROI.shape[0]//2
    y = height//2 - ROI.shape[1]//2
#     print(x,y)
    mask[y:y+h, x:x+w] = ROI
#     print(mask)
    # Check if centering/masking was successful
#     plt.imshow(mask, cmap='viridis')
    output_image = Image.fromarray(mask) # mask has values in [0-255] as expected
    # Now we need to resize, but it causes problems with default arguments as it
changes the range of pixel values to be negative or positive
    # compressed_output_image = output_image.resize((22,22))
    # Therefore, we use the following:
    compressed_output_image = output_image.resize((22,22), Image.BILINEAR) #
PIL.Image.NEAREST or PIL.Image.BILINEAR also performs good

    convert_tensor = torchvision.transforms.ToTensor()
    tensor_image = convert_tensor(compressed_output_image)
    # Another problem we face is that in the above ToTensor() command, we should have
gotten a normalized tensor with pixel values in [0,1]
    # But somehow it doesn't happen. Therefore, we need to normalize manually
    tensor_image = tensor_image/255.
    # Padding
    tensor_image = torch.nn.functional.pad(tensor_image, (3,3,3,3), "constant", 0)
    # Normalization shoudl be done after padding i guess
    convert_tensor = torchvision.transforms.Normalize((0.1281), (0.3043)) # Mean and
std of MNIST_plus
    tensor_image = convert_tensor(tensor_image)
    # st.write(tensor_image.shape)
```

```
    # Shape of tensor image is (1,28,28)


    # st.write('### Processing steps:')
    # st.write('1. Find the bounding box of the digit blob and use that.')
    # st.write('2. Convert it to size 22x22.')
    # st.write('3. Pad the image with 3 pixels on all the sides to get a 28x28 image.')
    # st.write('4. Normalize the image to have pixel values between 0 and 1.')
    # st.write('5. Standardize the image using the mean and standard deviation of the
MNIST_plus dataset.')

    # The following gives noisy image because the values are from -1 to 1, which is not
a proper image format
    im = Image.fromarray(tensor_image.detach().cpu().numpy().reshape(28,28), mode='L')
    im.save("processed_tensor.png", "PNG")
    # So we use matplotlib to save it instead
plt.imsave('processed_tensor.png',tensor_image.detach().cpu().numpy().reshape(28,28),
cmap='gray')

    # st.write('### Processed image')
    # st.image('processed_tensor.png')
    # st.write(tensor_image.detach().cpu().numpy().reshape(28,28))

    device='cpu'
    ### Compute the predictions
    with torch.no_grad():
        # input image for network should be (1,1,28,28)
        output0 = Network(torch.unsqueeze(tensor_image, dim=0).to(device=device))
        # st.write(output0)
        certainty, output = torch.max(output0[0], 0)
        certainty = certainty.clone().cpu().item()
        output = output.clone().cpu().item()
        certainty1, output1 = torch.topk(output0[0],3)
        certainty1 = certainty1.clone().cpu()#.item()
        output1 = output1.clone().cpu()#.item()
#     print(certainty)
    st.write('### Prediction')
    st.write('### '+str(output))

    st.write('## Breakdown of the prediction process:')

    st.write('### Image being used as input')
    st.image(canvas_result.image_data)

    st.write('### Image as a grayscale Numpy array')
    st.write(input_image_gs_np)

    st.write('### Processing steps:')
    st.write('1. Find the bounding box of the digit blob and use that.')
    st.write('2. Convert it to size 22x22.')
    st.write('3. Pad the image with 3 pixels on all the sides to get a 28x28 image.')
    st.write('4. Normalize the image to have pixel values between 0 and 1.')
```

```
    st.write('5. Standardize the image using the mean and standard deviation of the
MNIST_plus training dataset.')

    st.write('### Processed image')
    st.image('processed_tensor.png')



    st.write('### Prediction')
    st.write(str(output))
    st.write('### Certainty')
    st.write(str(certainty1[0].item()*100) +'%')
    st.write('### Top 3 candidates')
    st.write(str(output1))
    st.write('### Certainties')
    st.write(str(certainty1*100))
```

Pretrained CNN PyTorch model

https://github.com/manassharma07/MNIST-PLUS/blob/main/model_torch_MNIST_plus_CNN_98_5_streamlit.chk

You can download it and load it in your python code using:

```
import torch
Network = torch.load('model_torch_MNIST_plus_CNN_98_5_streamlit.chk')
```

Code used for training the model

https://github.com/manassharma07/crysx_nn/blob/main/mnist_experiments/CNN_MNIST_PLUS_from_raw_png.ipynb

Details of the Convolutional Neural Network

Optimizer: Stochastic Gradient Descent
Learning Rate = 0.3
Number of epochs = 9
Batch size =200

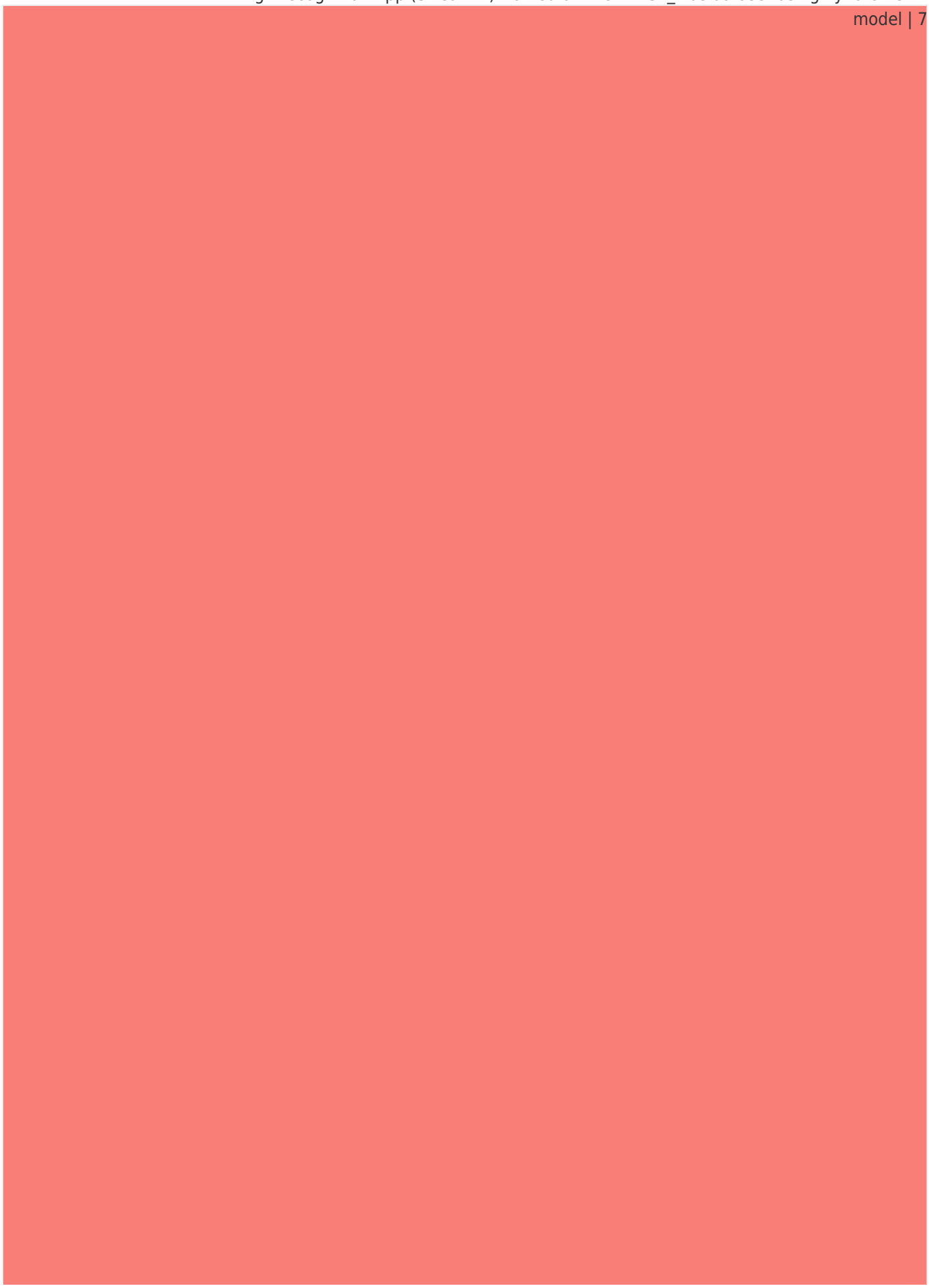
Code snippet for creation of CNN

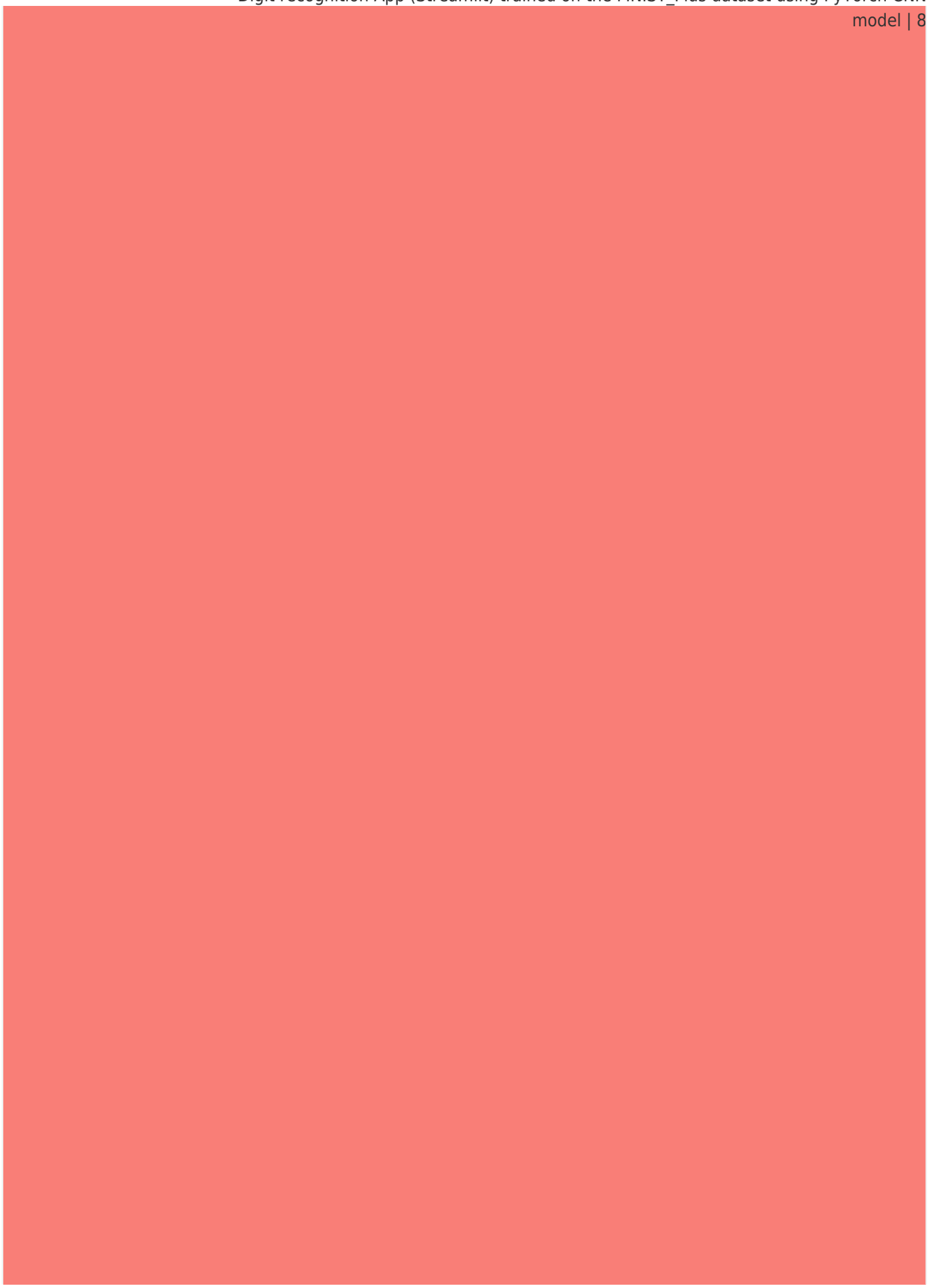
```
### Choose device: 'cuda' or 'cpu'
device = 'cpu'
# device = 'cuda'

Network = torch.nn.Sequential(      #  1x28x28
    torch.nn.Conv2d(1, 12, (9, 9)),  #  12x20x20
    torch.nn.MaxPool2d((2, 2)),     #  12x10x10
    torch.nn.ReLU(),
    torch.nn.Conv2d(12, 24, (5, 5)), # 24x 6x 6
    torch.nn.ReLU(),
    torch.nn.MaxPool2d((3, 3)),     # 24x 2x 2
    torch.nn.Flatten(),             #       96
    torch.nn.Linear(96, 256),        #       16
    torch.nn.ReLU(),
    torch.nn.Linear(256, 10),        #       10
#    torch.nn.Softmax(dim=1)
)
Network.to(device=device)

### Get information about model
totpars = 0
for par in Network.parameters():
    newpars = 1
    for num in par.shape:
        newpars *= num
    totpars += newpars
print(Network)
print('%i trainable parameters' % totpars)

### Initialize loss function and optimizer
# crit = torch.nn.BCELoss()
crit = torch.nn.CrossEntropyLoss()
opt = torch.optim.SGD(Network.parameters(), lr=0.3)
```

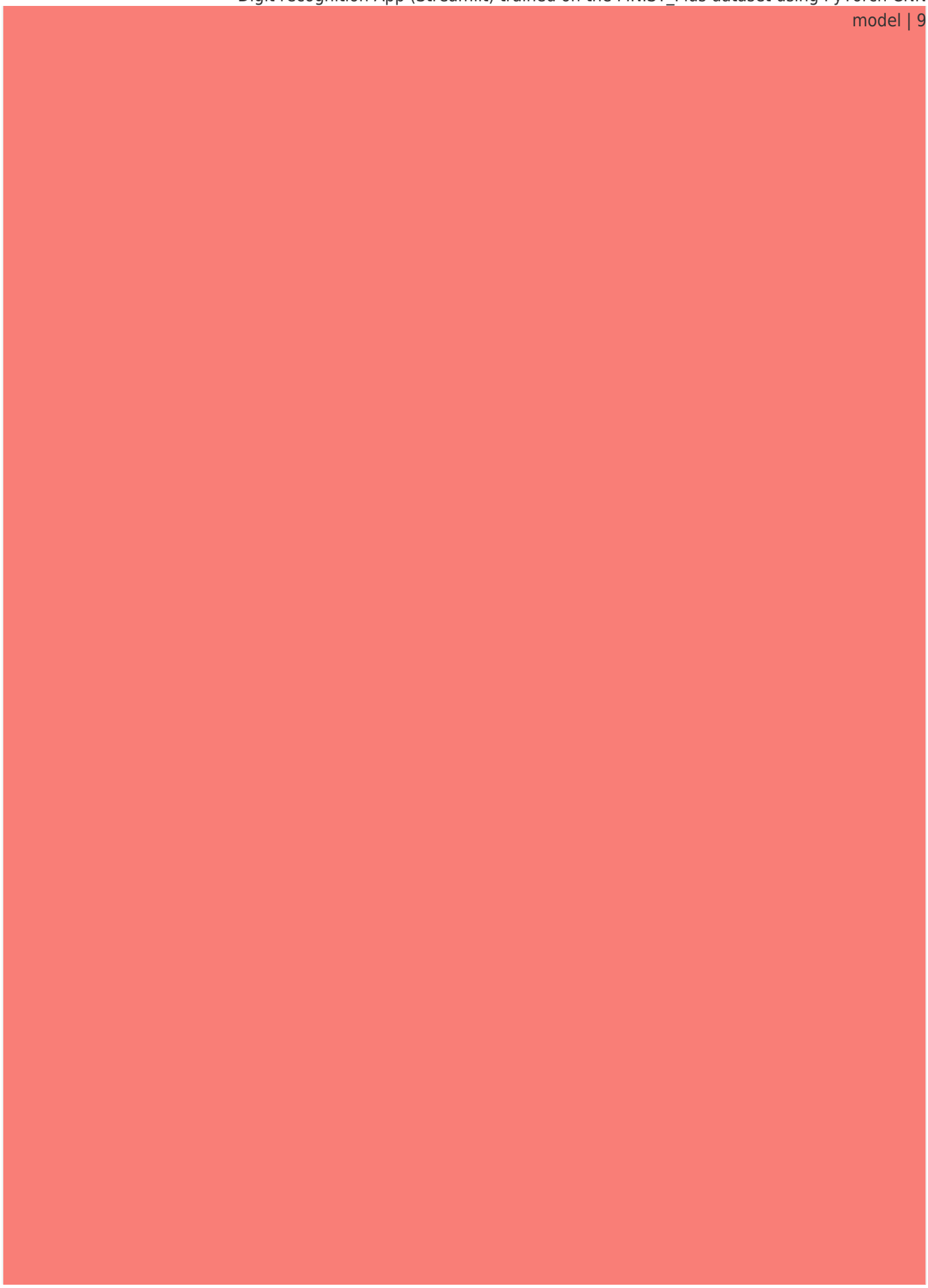
Manas Sharma

I’m a physicist specializing in computational material science with a PhD in Physics from Friedrich-Schiller University Jena, Germany. I write efficient codes for simulating light-matter interactions at atomic scales. I like to develop Physics, DFT, and Machine Learning related apps and software from time to time. Can code in most of the popular languages. I like to share my knowledge in Physics and applications using this Blog and a YouTube channel.

manas.bragitoff.com/







Share this:

Click to share on Facebook (Opens in new window)

Click to share on Twitter (Opens in new window)

Click to share on WhatsApp (Opens in new window)

Click to share on Pinterest (Opens in new window)

Click to share on Reddit (Opens in new window)

Click to share on LinkedIn (Opens in new window)

Click to email a link to a friend (Opens in new window)

Click to print (Opens in new window)

Click to share on Tumblr (Opens in new window)

Click to share on Pocket (Opens in new window)

Click to share on Telegram (Opens in new window)

[wpedon id="7041" align="center"]