Tin Machine
(Automating Tinder Swipes)

Deep Learning Project

By -

Lokesh Vadlamudi

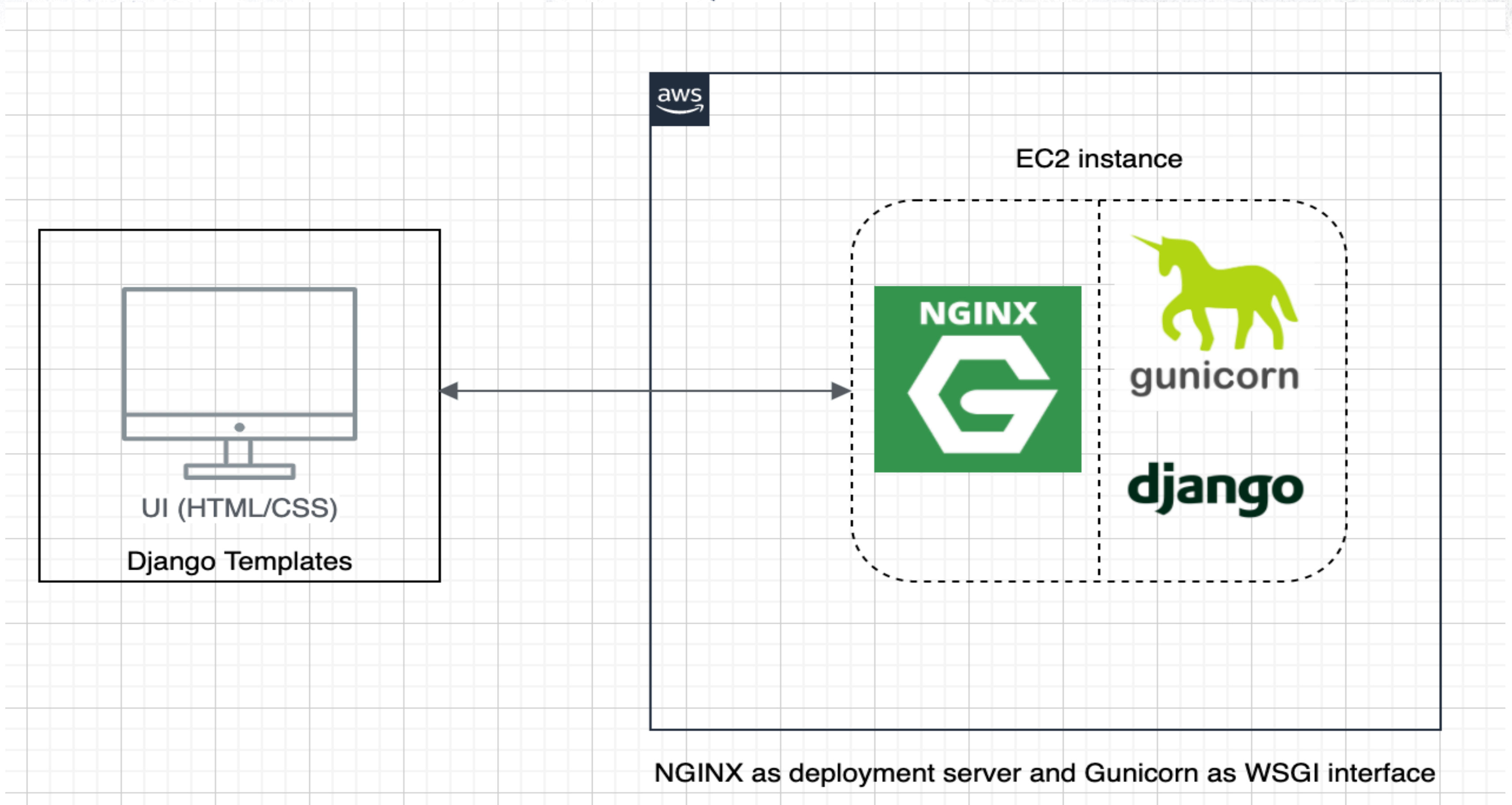
Nupur Yadav

Content

Introduction

- This project is based on Image feature recognition.
- We have tried to automate swiping to left or right on Tinder platform using our trained models based on one's preferences.
- We have built two models one in Fastai and the other in Keras to see which one fared better for our task. We have also fine tuned these models based on hyperparameters to improve their accuracies.
- We have used several evaluation metrices to compare performances of these models and tensorboard visualization.
- The final model has been deployed on AWS, which is taking images from our tinder account in real time and displaying the images on UI (3 at a time), and whether they have been liked or disliked based on the already trained model on our preferences.
- **Link to application** : http://ec2-54-176-2-190.us-west-1.compute.amazonaws.com/

Deployment Design



Steps Followed

1. Analysing Tinder Api calls
2. Collecting images from Tinder as part of Data Collection process
3. Preprocessing the Data (includes resizing data, grayscaling, data augmentation)
4. Annotating the images manually as “Like (=1)” and “Dislike (=0)” based on our choice of preference using ImageClassifier script
5. Training FastAi and Keras models based on annotated images
6. Fine tuning the models based on several hyperparameters to improve their accuracies
7. Evaluating the models based on different evaluation metrices to come up with the best model
8. Saving the final model for deployment
9. Deploying the model on AWS using Django framework (Created an application in Django where we are loading the trained model , getting live images **using Tinder “nearby_persons” API** and classifying those as like or dislike in real time based on inference from trained model)

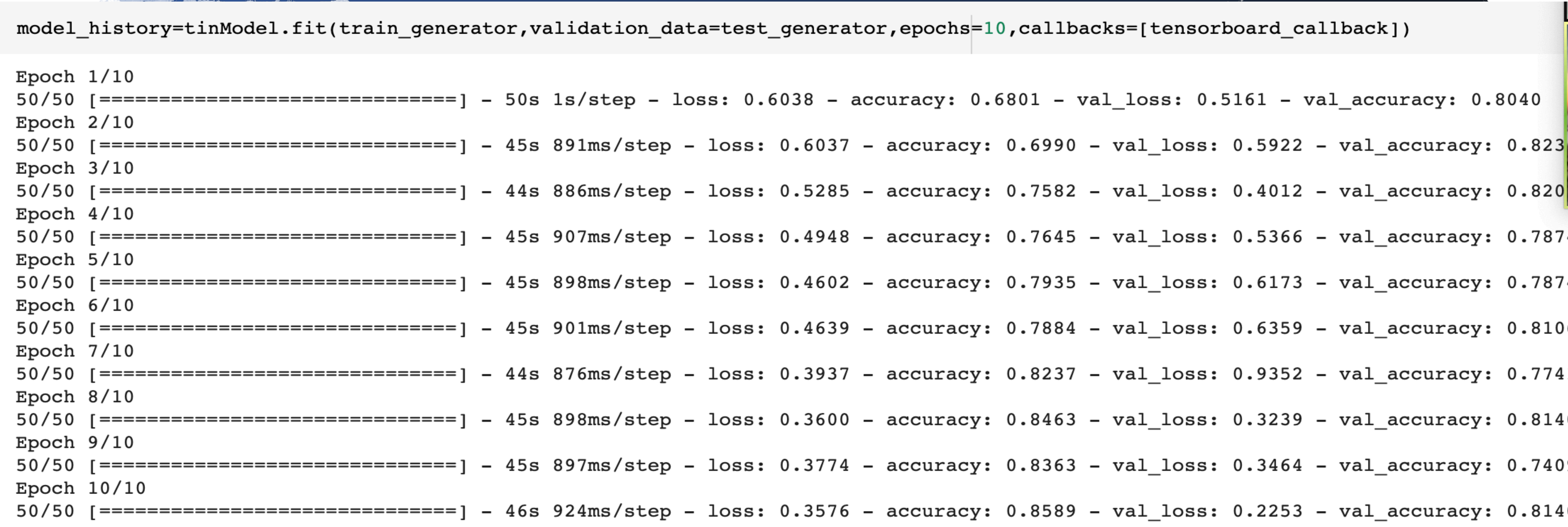
Data Collection and Preprocessing

- For data collection we have used **Tinder API <https://api.gotinder.com>** with Facebook auth token for authentication.
- We stored the data collected to our google drive and manually annotated the collected images as “like (=1)” and “dislike (=0)” based on our choice of preference into two separate folders using our ImageClassifier script.
- As part of data preprocessing, we then resized the images, grayscaled them, and also used data augmentation techniques for enrichment.
- For train-test split, we chose the ratio of 3:1 after several observations while training models to get the max accuracy.

Models Trained

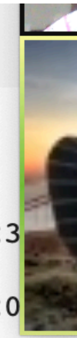
- Once we had the labelled data based on our choice of preference, we trained two separate models one in FastAi and the other in Keras.
- We trained Resnet34, Resnet50 and CNN models with different architectures.
- We fined tuned these models on several hyperparameters such as learning rate, no of hidden layers, activation functions at different layers, using various dropout rates, batch normalization layers, max-pooling etc.
- The CNN model 2 architecture (can be seen in colab) gave the best accuracy of approx. 86%
- We then saved the best model weights for generating future inferences.

Training summary for CNN model 2 architecture

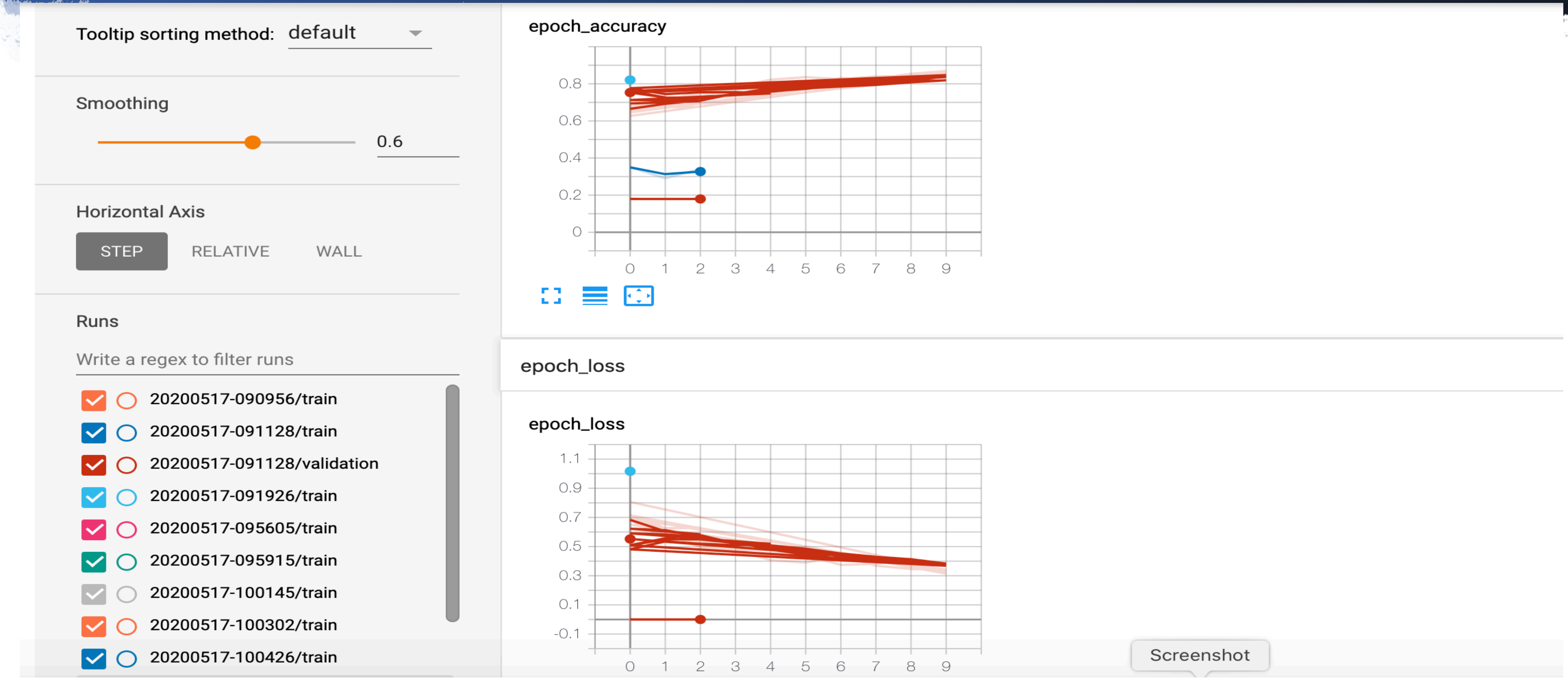


```
model_history=tinModel.fit(train_generator,validation_data=test_generator,epochs=10,callbacks=[tensorboard_callback])
```

```
Epoch 1/10
50/50 [==============================] - 50s 1s/step - loss: 0.6038 - accuracy: 0.6801 - val_loss: 0.5161 - val_accuracy: 0.8040
Epoch 2/10
50/50 [==============================] - 45s 891ms/step - loss: 0.6037 - accuracy: 0.6990 - val_loss: 0.5922 - val_accuracy: 0.823
Epoch 3/10
50/50 [==============================] - 44s 886ms/step - loss: 0.5285 - accuracy: 0.7582 - val_loss: 0.4012 - val_accuracy: 0.820
Epoch 4/10
50/50 [==============================] - 45s 907ms/step - loss: 0.4948 - accuracy: 0.7645 - val_loss: 0.5366 - val_accuracy: 0.7874
Epoch 5/10
50/50 [==============================] - 45s 898ms/step - loss: 0.4602 - accuracy: 0.7935 - val_loss: 0.6173 - val_accuracy: 0.7874
Epoch 6/10
50/50 [==============================] - 45s 901ms/step - loss: 0.4639 - accuracy: 0.7884 - val_loss: 0.6359 - val_accuracy: 0.8106
Epoch 7/10
50/50 [==============================] - 44s 876ms/step - loss: 0.3937 - accuracy: 0.8237 - val_loss: 0.9352 - val_accuracy: 0.7741
Epoch 8/10
50/50 [==============================] - 45s 898ms/step - loss: 0.3600 - accuracy: 0.8463 - val_loss: 0.3239 - val_accuracy: 0.8140
Epoch 9/10
50/50 [==============================] - 45s 897ms/step - loss: 0.3774 - accuracy: 0.8363 - val_loss: 0.3464 - val_accuracy: 0.7409
Epoch 10/10
50/50 [==============================] - 46s 924ms/step - loss: 0.3576 - accuracy: 0.8589 - val_loss: 0.2253 - val_accuracy: 0.8140
```



Tensorboard Visualization



Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis
[✓ STEP] RELATIVE WALL

Runs

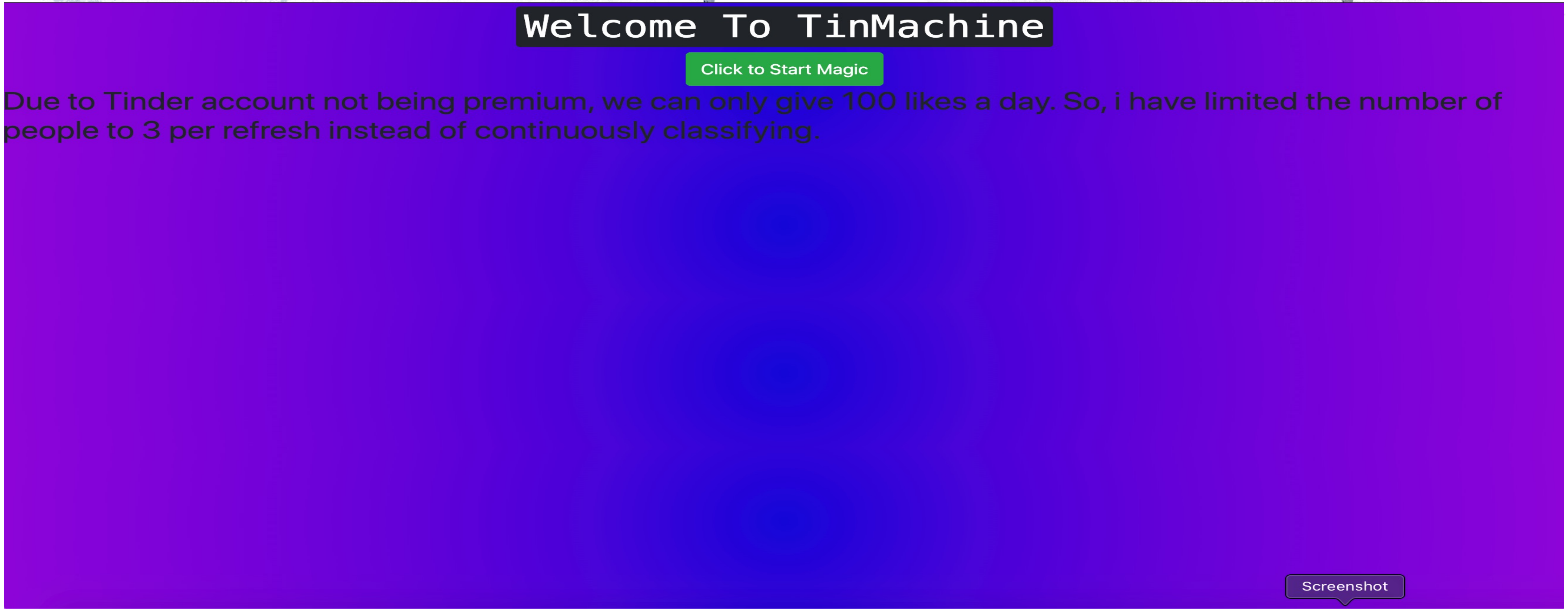
Write a regex to filter runs

- ✓ 20200517-090956/train
- ✓ 20200517-091128/train
- ✓ 20200517-091128/validation
- ✓ 20200517-091926/train
- ✓ 20200517-095605/train
- ✓ 20200517-095915/train
- ✓ 20200517-100145/train
- ✓ 20200517-100302/train
- ✓ 20200517-100426/train

Model Deployment

1. We built a python application using **Django Framework to deploy our model on AWS**. Following is the link : http://ec2-54-176-2-190.us-west-1.compute.amazonaws.com/
2. The UI is implemented in HTML/CSS using Django templates
3. On the server side we have used NGINX as our production deployment server with Gunicorn as WSGI interface between NGINX and Django framework.
4. Whenever a user hits the URL, it is loading the already trained model.
5. And when a user clicks on "Click to Start Magic" button, it is getting the images from the Tinder account using the auth token id and **"nearby_persons" API** in real time, running the inference model on these images and displaying the images on the UI (3 at a time) with name and inferred labels as "Like" or "Dislike".

Application Screenshots

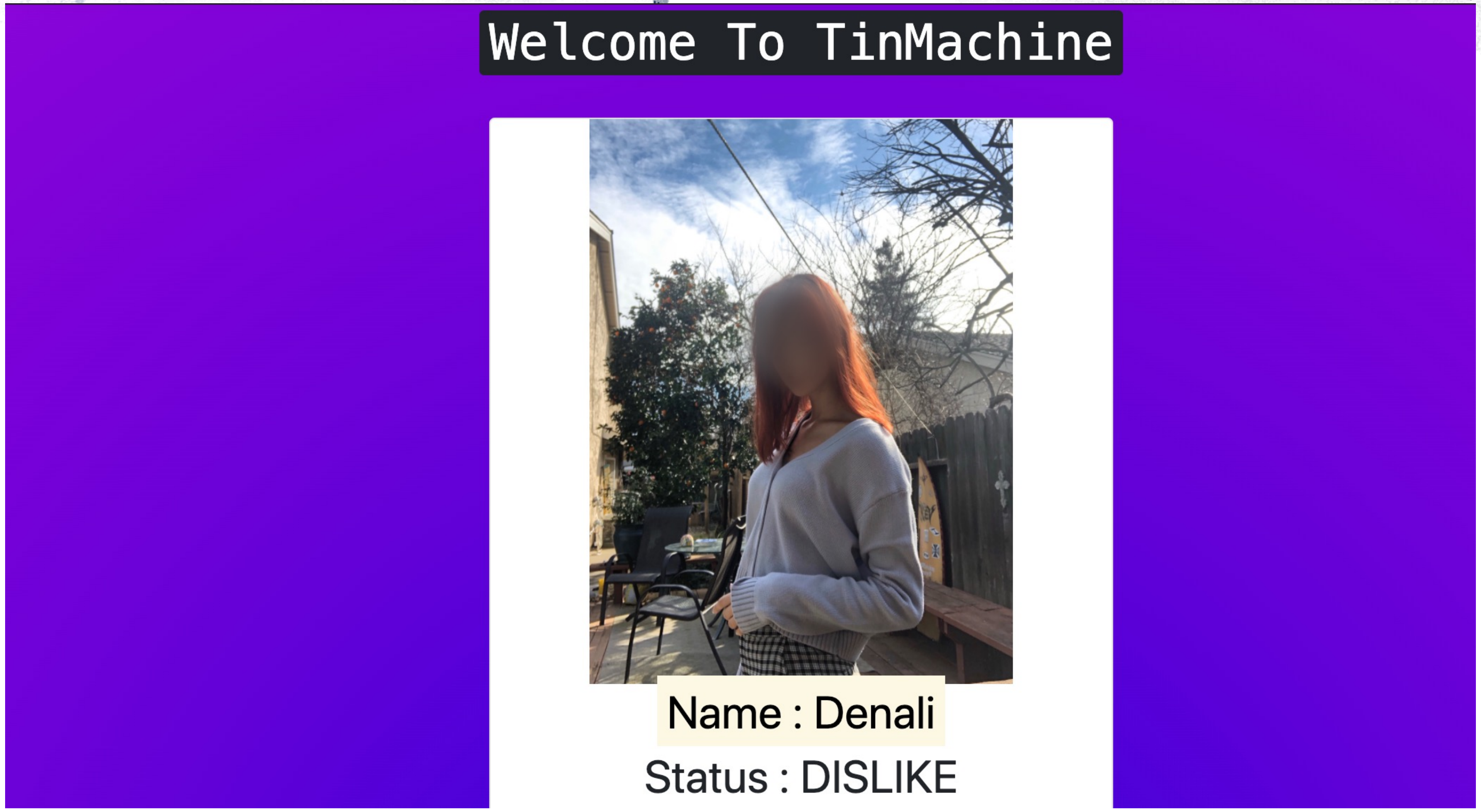


Welcome To TinMachine

Click to Start Magic

Due to Tinder account not being premium, we can only give 100 likes a day. So, i have limited the number of people to 3 per refresh instead of continuously classifying.

Application Screenshots



Welcome To TinMachine

Name : Denali

Status : DISLIKE

Conclusion

- We tried to automate Tinder swipes and used CNN model for the image classification task which was trained on one’s choice of preferences.
- The accuracy we achieved was approx. 86% , but we are sure this can be improved with further model tuning, better annotation and large datasets
- As future work, we will allow any Tinder user to train model based on his/her choice of preference using their xauth token id to automate Tinder swipes instead of doing it manually.