Travis Vitello

tjv9qh

10.6 Codeathon 3: There's an App for That Deep Learning Model!

For this assignment, I wanted to implement a mask vs. no mask image classification model using a manually-built AlexNet framework (as observed earlier in this semester, such as the Module 6 homework for landmark classification). The model was structured per the following:

```
model.compile(loss='categorical_crossentropy', optimizer=tf.optimizers.SGD(lr=0.01), metrics=['accuracy'])
model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 96)        34944
_________________________________________________________________
batch_normalization (BatchNo (None, 30, 30, 96)        384
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 96)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 256)       614656
_________________________________________________________________
batch_normalization_1 (Batch (None, 14, 14, 256)       1024
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 256)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 6, 6, 384)         885120
_________________________________________________________________
batch_normalization_2 (Batch (None, 6, 6, 384)         1536
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 6, 6, 384)         147840
_________________________________________________________________
batch_normalization_3 (Batch (None, 6, 6, 384)         1536
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 6, 6, 256)         98560
_________________________________________________________________
batch_normalization_4 (Batch (None, 6, 6, 256)         1024
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 4096)              4198400
_________________________________________________________________
dropout (Dropout)            (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 8194
=================================================================
Total params: 22,774,530
Trainable params: 22,771,778
Non-trainable params: 2,752
_________________________________________________________________
```

Figure 1. AlexNet Build

The trained model on mask vs. no mask data achieved an accuracy of 99.1%. Details of the data used are available in the files associated with this assignment submission.

```
model.fit(train_ds, epochs=20,  validation_data = val_ds,callbacks=[Estop])
```

```
Epoch 1/20
313/313 [==============================] - 26s 75ms/step - loss: 0.7414 - accuracy: 0.8330 - val_loss: 0.2402 - val_accuracy: 0.9050
Epoch 2/20
313/313 [==============================] - 23s 73ms/step - loss: 0.1084 - accuracy: 0.9605 - val_loss: 0.1184 - val_accuracy: 0.9500
Epoch 3/20
313/313 [==============================] - 23s 73ms/step - loss: 0.0521 - accuracy: 0.9804 - val_loss: 0.0283 - val_accuracy: 0.9912
Epoch 4/20
313/313 [==============================] - 23s 73ms/step - loss: 0.0374 - accuracy: 0.9862 - val_loss: 0.7078 - val_accuracy: 0.8250
Epoch 5/20
313/313 [==============================] - 23s 73ms/step - loss: 0.0280 - accuracy: 0.9890 - val_loss: 0.2701 - val_accuracy: 0.9175
Epoch 6/20
313/313 [==============================] - 23s 74ms/step - loss: 0.0168 - accuracy: 0.9938 - val_loss: 0.0158 - val_accuracy: 0.9962
Epoch 7/20
313/313 [==============================] - 23s 73ms/step - loss: 0.0152 - accuracy: 0.9938 - val_loss: 0.0105 - val_accuracy: 0.9950
Epoch 8/20
313/313 [==============================] - 23s 73ms/step - loss: 0.0071 - accuracy: 0.9973 - val_loss: 0.0100 - val_accuracy: 0.9987
Epoch 9/20
313/313 [==============================] - 23s 74ms/step - loss: 0.0094 - accuracy: 0.9960 - val_loss: 0.0127 - val_accuracy: 0.9937
Epoch 10/20
313/313 [==============================] - 23s 74ms/step - loss: 0.0059 - accuracy: 0.9977 - val_loss: 0.0093 - val_accuracy: 0.9987
Epoch 11/20
313/313 [==============================] - 23s 73ms/step - loss: 0.0029 - accuracy: 0.9990 - val_loss: 0.0135 - val_accuracy: 0.9962
Epoch 00011: early stopping
<tensorflow.python.keras.callbacks.History at 0x7f4f71362e90>
```

```
[16] model.evaluate(test_ds)
```

```
31/31 [==============================] - 2s 67ms/step - loss: 0.0278 - accuracy: 0.9909
[0.02775118499994278, 0.9909273982048035]
```

Figure 2. Training of AlexNet Model

After having confidence that the model can be used to predict a mask vs. no mask classification locally on new images, the model was saved and uploaded to the Google Cloud Platform (GCP). Note that the Jupyter Notebook (downloaded from Google Colab) used in this study is available on my Github, here:

https://github.com/tvitello/SYS6016_Codeathon3

This file will show several cases where images were applied against the trained model. Code is also contained therein showing how the model was migrated to the GCP project, the details of which are shown below.

```
: Image.open('/content/gdrive/MyDrive/DL_Project/kaggle_pics/linked.JPG')
:
```

```
: img9 =  tf.io.read_file("/content/gdrive/MyDrive/DL_Project/kaggle_pics/linked.JPG")

: img9 = tf.io.decode_image(img9, channels=3)
  img9 = tf.image.resize(img9, [128, 128])
  img9 = img9/255.
  img9 = np.expand_dims(img9,axis=0)
  print(img9.shape)

  (1, 128, 128, 3)

: pred =  model.predict(img9)
  print(pred)

  [[0.2564176 0.8035079]]

: class_names[tf.argmax(pred[0])]

: 'no_mask'
```

Figure 3. Typical Image Classification Test of Model Locally within Notebook, Prior to Transferring Model to GCP

```
!gsutil cp -r /content/gdrive/MyDrive/DL_Project/04292021_model_0001 gs://tjv9qh_codeathon_3-sys6016

Copying file:///content/gdrive/MyDrive/DL_Project/04292021_model_0001/saved_model.pb [Content-Type=application/octet-stream]...
Copying file:///content/gdrive/MyDrive/DL_Project/04292021_model_0001/variables/variables.data-00000-of-00001 [Content-Type=application/octet-stream]...
Copying file:///content/gdrive/MyDrive/DL_Project/04292021_model_0001/variables/variables.index [Content-Type=application/octet-stream]...
\ [3 files][ 87.3 MiB/ 87.3 MiB]
Operation completed over 3 objects/87.3 MiB.
```

Figure 4. Notebook Code Showing Model Uploaded to GCP

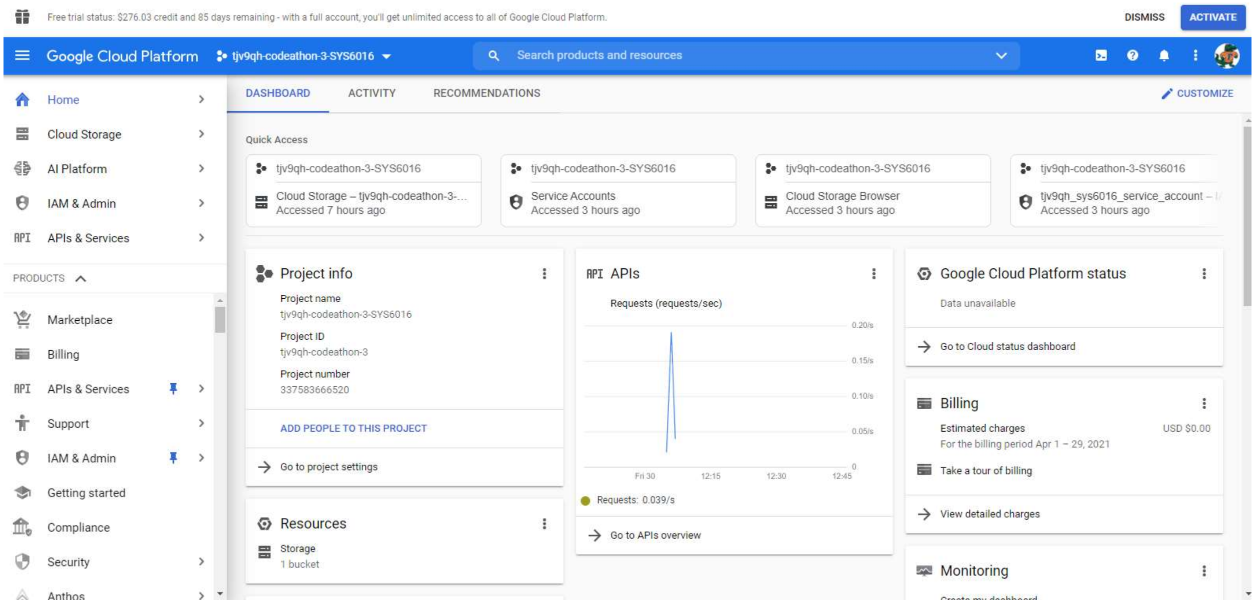


Figure 5. Overview of GCP Project

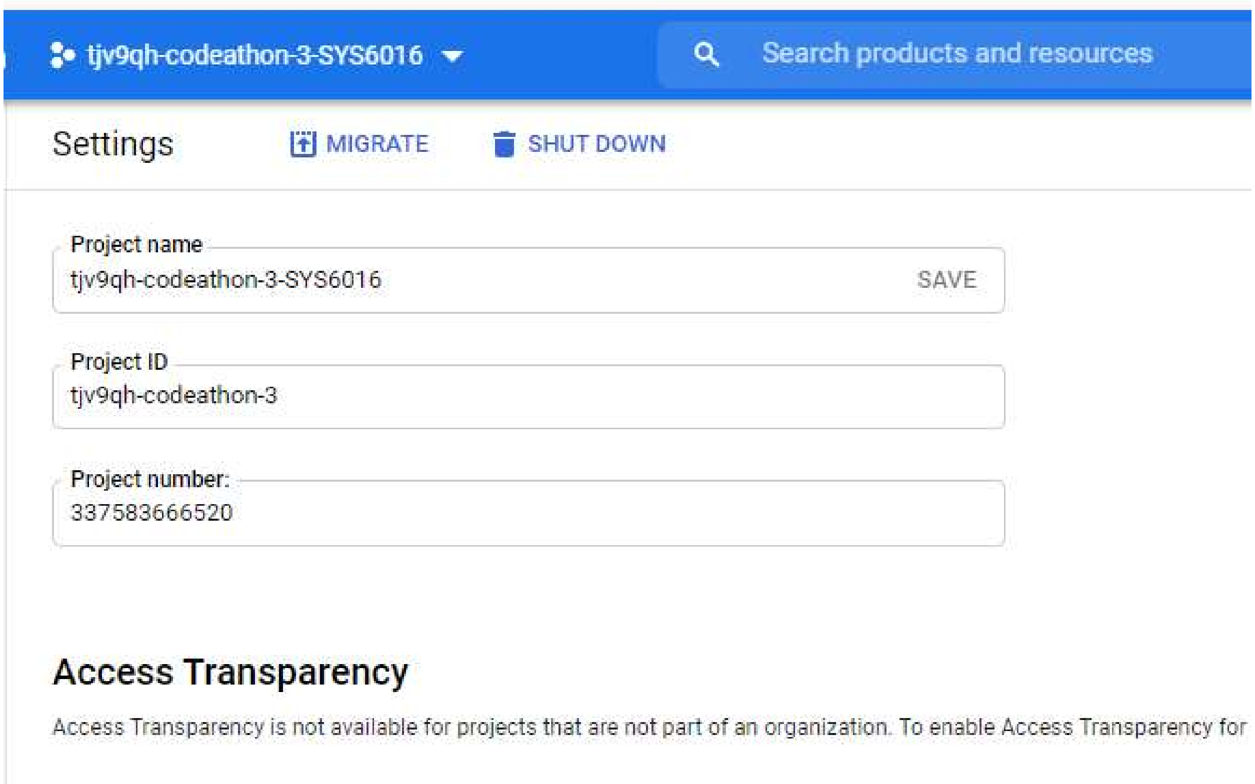


Figure 6. GCP Project Detail

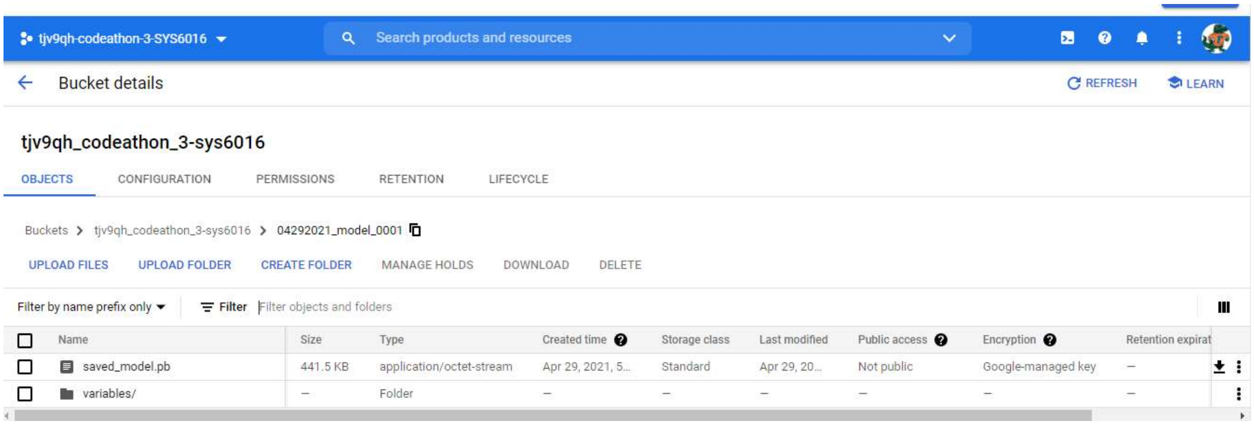


Figure 7. GCP Bucket Hosting ML Model

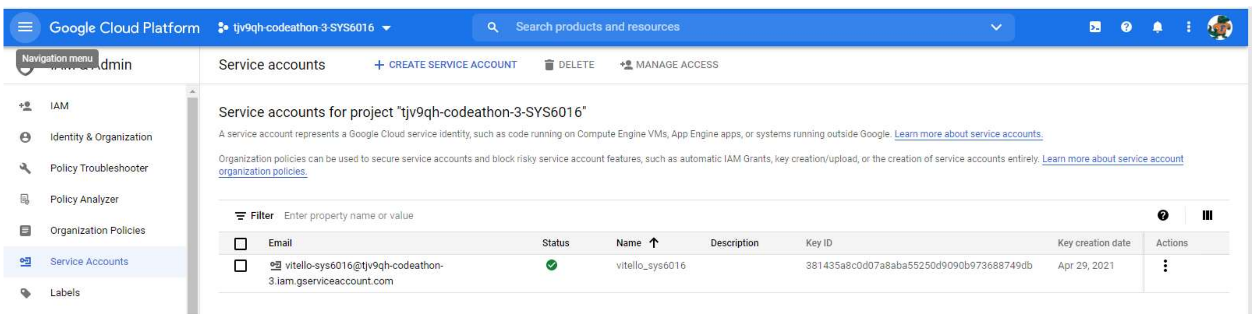
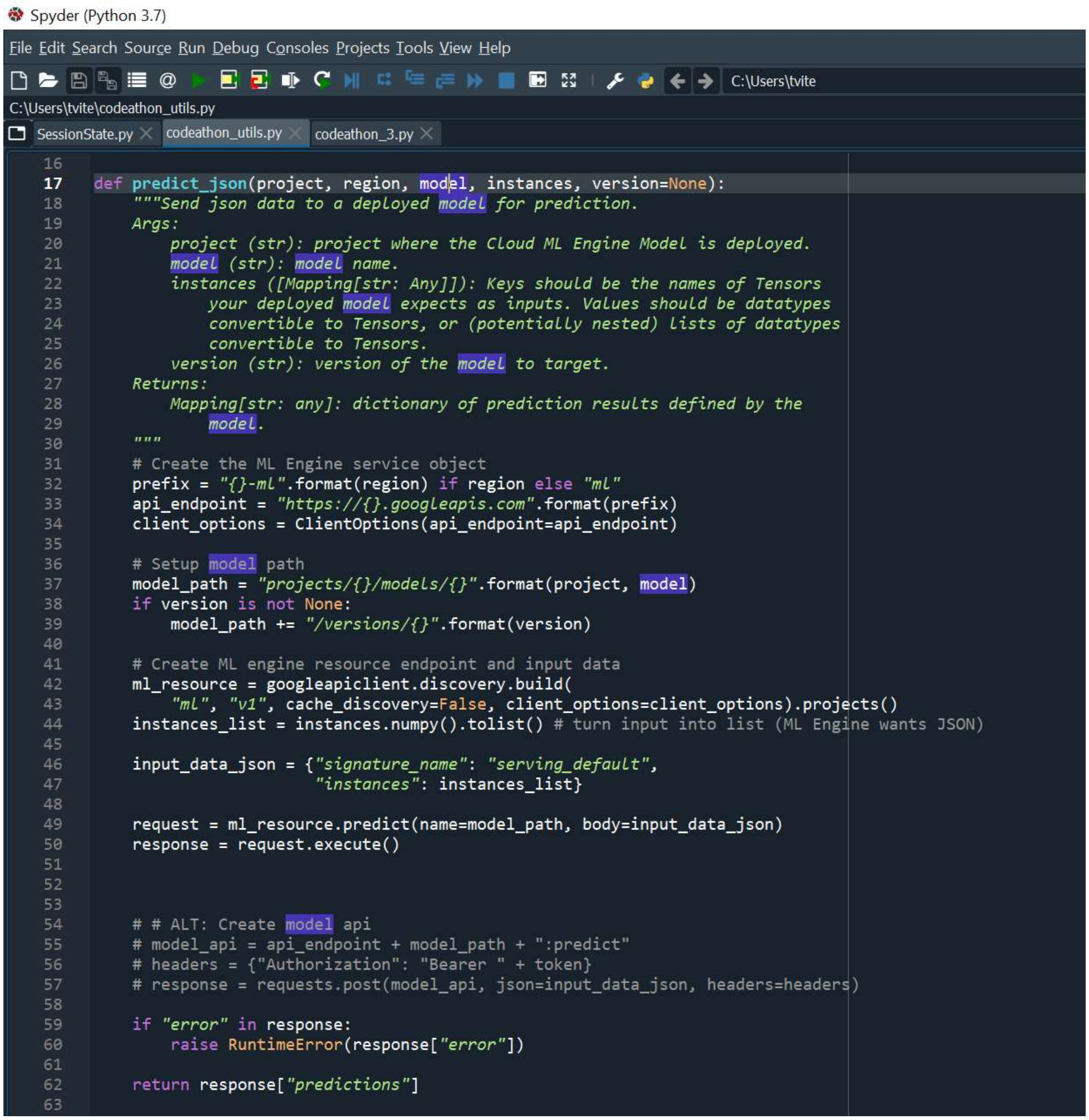


Figure 8. Service Access to GCP Project and Model

From hereon, .py files were created leveraging the Streamlit platform so as to locally host an app classifying mask vs. no mask images. Said local app's predictions utilize the ML model uploaded to the Google Cloud Platform, making calls to the GCP model per the following code (also included in the submission for this Codeathon).

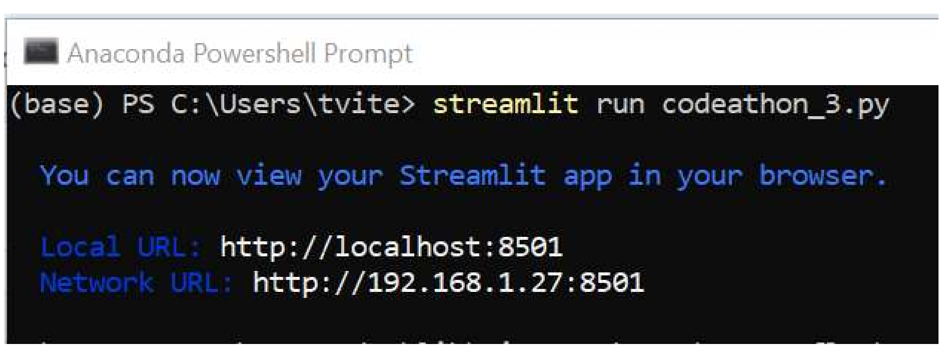
```python
### Code inspired by and based on https://github.com/mrdbourke/cs329s-ml-deployment-tutorial/
### Travis Vitello
### tjv9qh
import os
import json
import requests
import SessionState
import streamlit as st
import tensorflow as tf
import numpy as np
from codeathon_utils import load_and_prep_image, classes_and_models, update_logger, predict_json

tf.enable_eager_execution()

# Setup environment credentials (you'll need to change these)
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "tjv9qh-codeathon-3-381435a8c0d0.json" # change for your GCP key
PROJECT = "tjv9qh-codeathon-3" # change for your GCP project
REGION = "us-central1" #"us-east4" # change for your GCP region (where your model is hosted)

### Streamlit code (works as a straigtht-forward script) ###
st.title("SYS6016: Codeathon 3")
st.header("Identify Whether a Person is Wearing a Face Mask or Not")
st.subheader("Travis Vitello \n tjv9qh")
st.write("ML Model based on manually-reconstructed AlexNet model with an SGD optimizer.  Training was performed
         achieving an accuracy of 99.1% on masks vs. no masks.  The model was trained against Ashish Jangra's \
             'Face Mask 12K Image Dataset' available on Kaggle at https://www.kaggle.com/ashishjangra27/face-mas
                 This model consisted of a test set comprised of 483 mask vs. 509 no mask images, and a training
                     set comprised of 5000 mask vs. 5000 no mask images.  Augmentation was previously applied.")
from PIL import Image
image = Image.open('alexnet.JPG')
st.image(image, caption='Fig. 1 --- Manually-Built AlexNet Model Used in this Study')

st.write("References:")
st.write("[1] “Welcome to Streamlit — Streamlit 0.81.0 Documentation.” Streamlit, docs.streamlit.io/en/stable. A
st.write("[2] Jangra, Ashish. “Face Mask ~12K Images Dataset.” Kaggle, 26 May 2020, www.kaggle.com/ashishjangra2
st.write("[3] Bourke, David. “Mrdbourke/Cs329s-Ml-Deployment-Tutorial.” GitHub, 2021, github.com/mrdbourke/cs329
st.write("[4] Alake,  Richmond.  “Implementing  AlexNet  CNN  Architecture  Using  TensorFlow  2.0  and  Keras.”
st.write("[5] Verdhan, Vaibhav. Computer Vision Using Deep Learning: Neural Network Architectures with Python an

@st.cache # cache the function so predictions aren't always redone (Streamlit refreshes every click)
def make_prediction(image, model, class_names):
    """
    Takes an image and uses model (a trained TensorFlow model) to make a
    prediction.
    Returns:
     image (preproccessed)
     pred_class (prediction class from class_names)
     pred_conf (model confidence)
    """
```

Figure 9. Python Code that Builds the Local Streamlit App




```python

def predict_json(project, region, model, instances, version=None):
    """Send json data to a deployed model for prediction.
    Args:
        project (str): project where the Cloud ML Engine Model is deployed.
        model (str): model name.
        instances ([Mapping[str: Any]]): Keys should be the names of Tensors
            your deployed model expects as inputs. Values should be datatypes
            convertible to Tensors, or (potentially nested) lists of datatypes
            convertible to Tensors.
        version (str): version of the model to target.
    Returns:
        Mapping[str: any]: dictionary of prediction results defined by the
            model.
    """
    # Create the ML Engine service object
    prefix = "{}-ml".format(region) if region else "ml"
    api_endpoint = "https://{}.googleapis.com".format(prefix)
    client_options = ClientOptions(api_endpoint=api_endpoint)

    # Setup model path
    model_path = "projects/{}/models/{}".format(project, model)
    if version is not None:
        model_path += "/versions/{}".format(version)

    # Create ML engine resource endpoint and input data
    ml_resource = googleapiclient.discovery.build(
        "ml", "v1", cache_discovery=False, client_options=client_options).projects()
    instances_list = instances.numpy().tolist() # turn input into list (ML Engine wants JSON)

    input_data_json = {"signature_name": "serving_default",
                       "instances": instances_list}

    request = ml_resource.predict(name=model_path, body=input_data_json)
    response = request.execute()



    # # ALT: Create model api
    # model_api = api_endpoint + model_path + ":predict"
    # headers = {"Authorization": "Bearer " + token}
    # response = requests.post(model_api, json=input_data_json, headers=headers)

    if "error" in response:
        raise RuntimeError(response["error"])

    return response["predictions"]

```

Figure 10. Python Code that Applies GCP ML Model to Classify Imported Images




```
(base) PS C:\Users\tvite> streamlit run codeathon_3.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.1.27:8501
```

Figure 11. Typical Call of Local Streamlit App

The above command opens a browser window per the following figures.

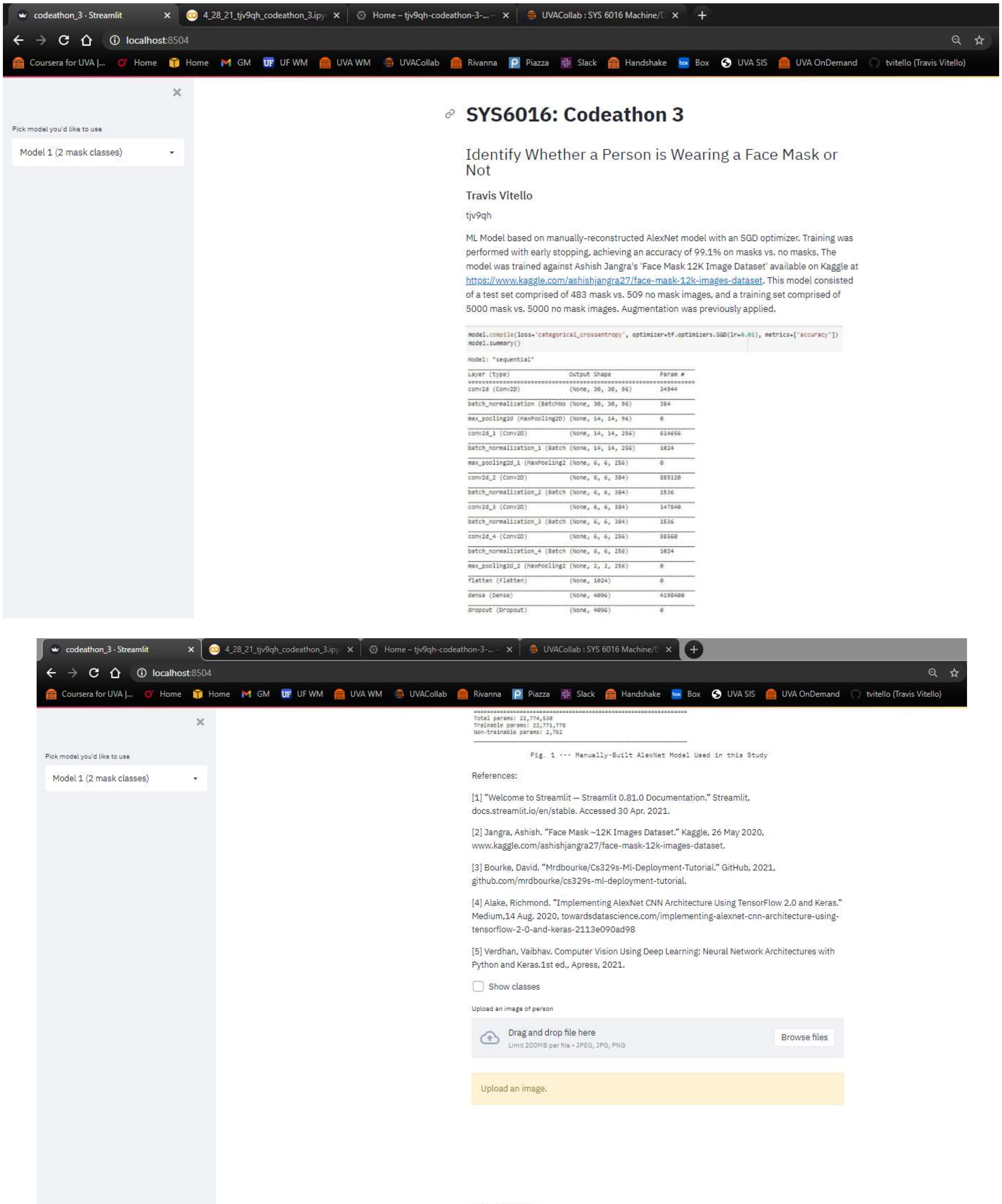


Figure 12. Local Streamlit App in Browser

From here, images can be opened in the local app with predictions made according to the GCP-hosted ML model. Not all of these images were classified correctly, suggesting that a more sophisticated model (such as one leveraging Xception) may improve performance. However, for the sake of this exercise, the performance herein is considered acceptable for demonstration purposes.

Figure 13. Spouse Wearing Mask - Correct, 100% Confidence

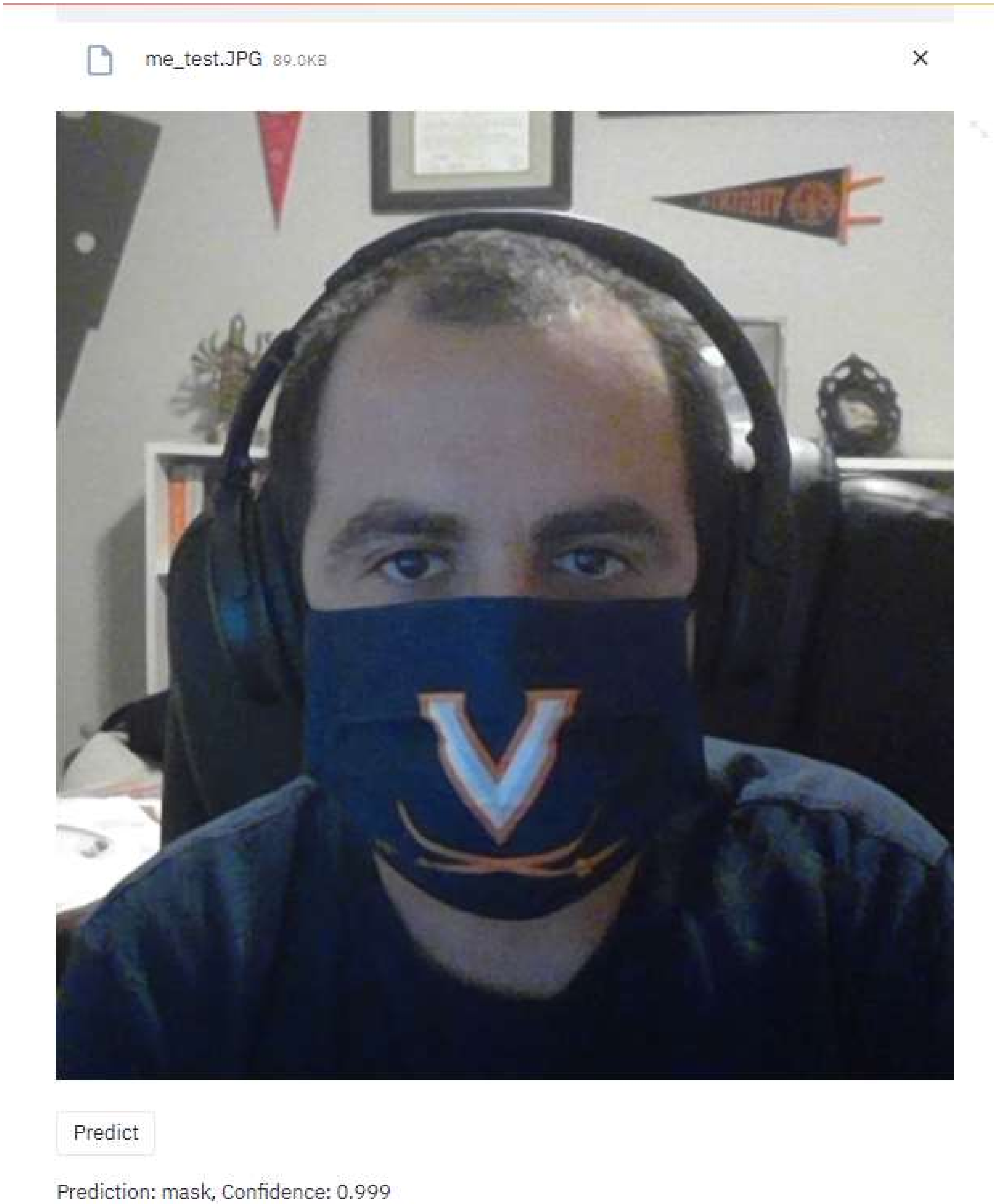


Figure 14. Me Wearing Mask - Correct, 99.9% Confidence

Predict

Prediction: no_mask, Confidence: 0.945

Figure 15. My Kid Not Wearing Mask - Correct, 94.5% Confidence

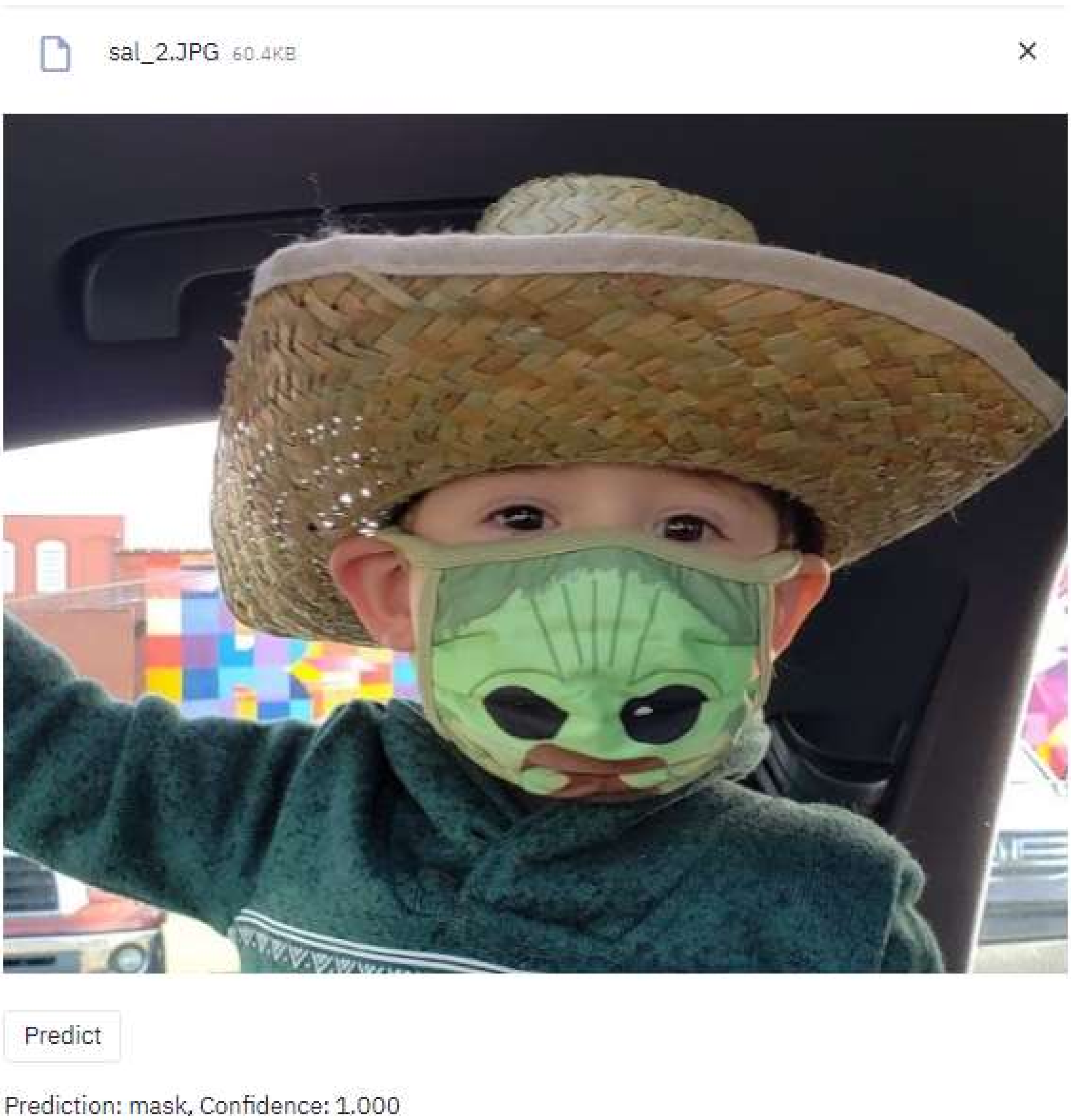


Figure 16. My Kid Wearing Mask - Correct, 100% Confidence

Figure 17. My Friend CJ Not Wearing Mask - Incorrect, 64.8% Confidence

This image represents an incorrect classification. More training on darker skinned individuals and individuals with facial hair may improve performance here.

cj_mask.JPG 20.5KB

Predict

Prediction: mask, Confidence: 0.997

Figure 18. My Friend CJ Wearing Mask - Correct, 99.7% Confidence

Figure 19. Professor Nguyen Not Wearing Mask - Incorrect, Confidence 58.8%

The near-50% confidence in the prediction here suggests that this image is borderline being classified correctly. It's not obvious to me why this image would be misclassified, as facial landmarks including nostrils, mouth, and chin are all visible. Further study would be needed to understand what happened in this case.

Predict

Prediction: no_mask, Confidence: 0.881

Figure 20. Ethan Not Wearing Mask - Correct, 88.1% Confidence

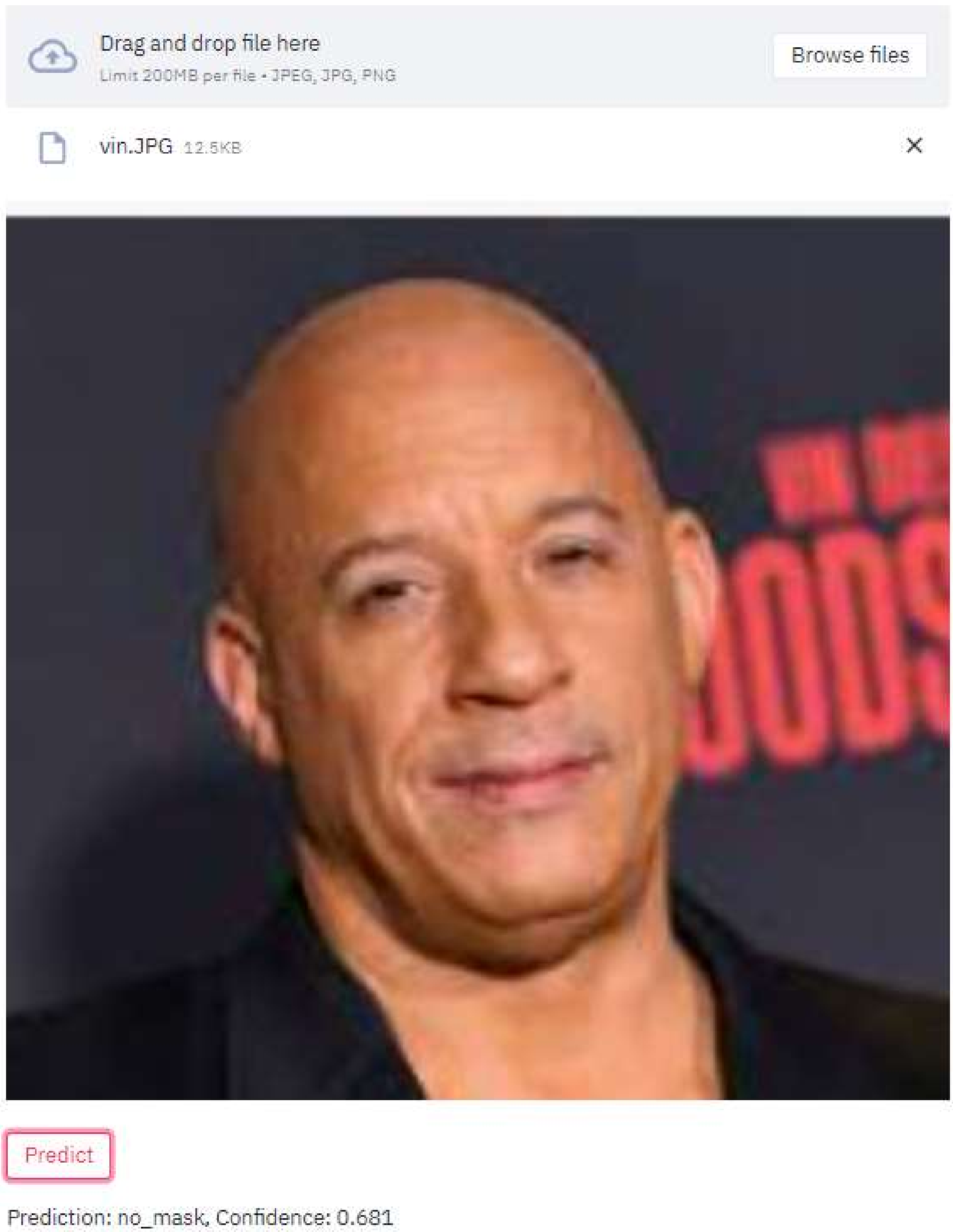


Figure 21. Vin Diesel Not Wearing Mask - Correct, 68.1% Confidence

cardi.JPG 14.4KB

Predict

Prediction: mask, Confidence: 0.993

Figure 22. Cardi B Wearing Mask - Correct, 99.3% Confidence

joe.JPG 12.2KB ×

Predict

Prediction: no_mask, Confidence: 0.916

Figure 23. Joe Biden Not Wearing Mask - Correct, 91.6% Confidence

Figure 24. Shaquille O'Neal Not Wearing Mask - Incorrect, 99.6% Confidence

This is another case where the image classification struggled with an individual of darker skin tone. In this case, the model was very confident (a staggering 99.6%) that Shaq is wearing a mask in this image. While he does appear to have some shadowing and/or slight facial hair, it is visually obvious that Shaq is not wearing a face mask in this image. It is expected that more training on images of individuals having darker skin would be needed here in order to improve future model performance.

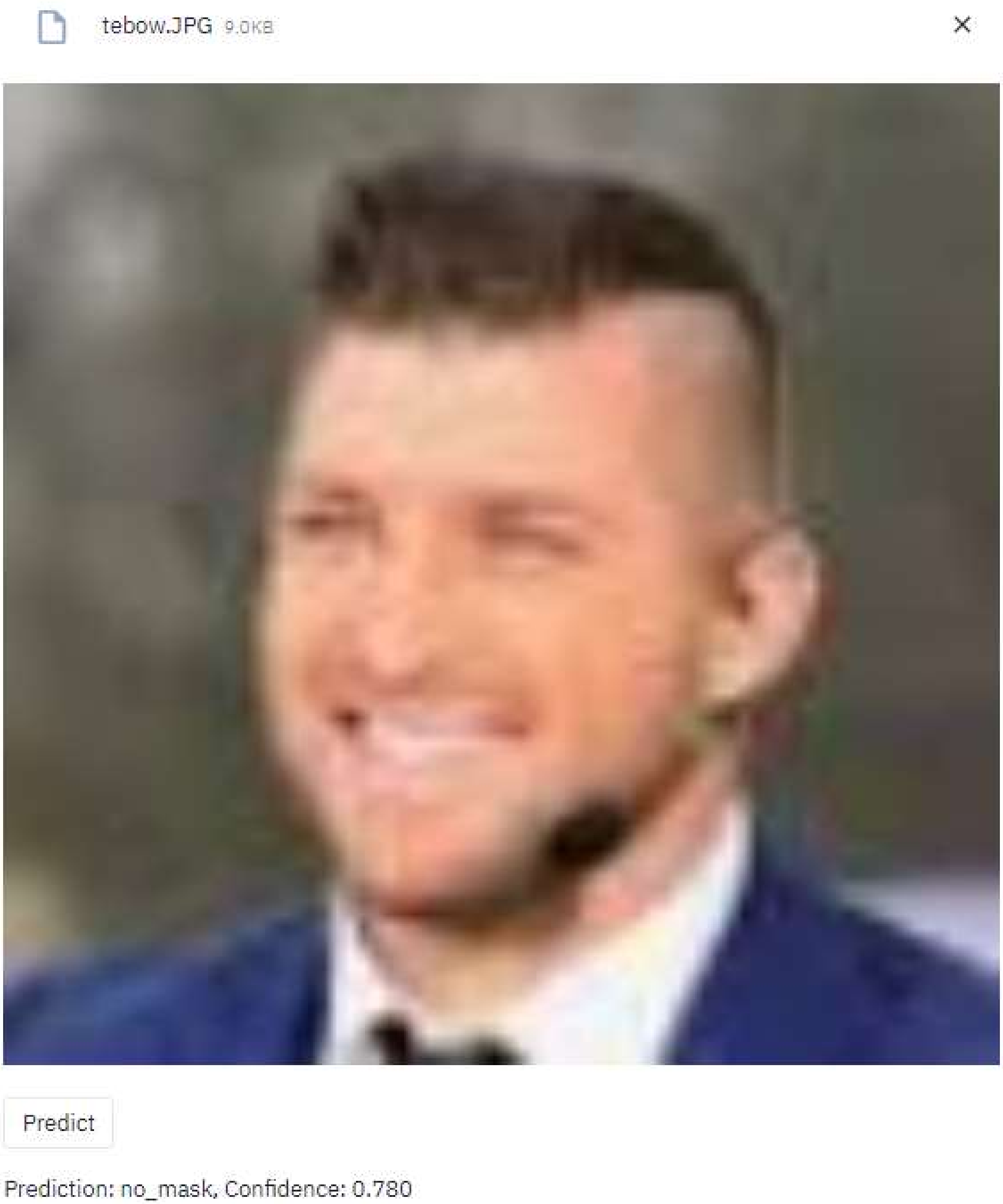


Figure 25. Tim Tebow Not Wearing Mask - Correct, 78.0% Confidence

Figure 26. Taylor Swift Not Wearing Mask - Correct, 97.8% Confidence

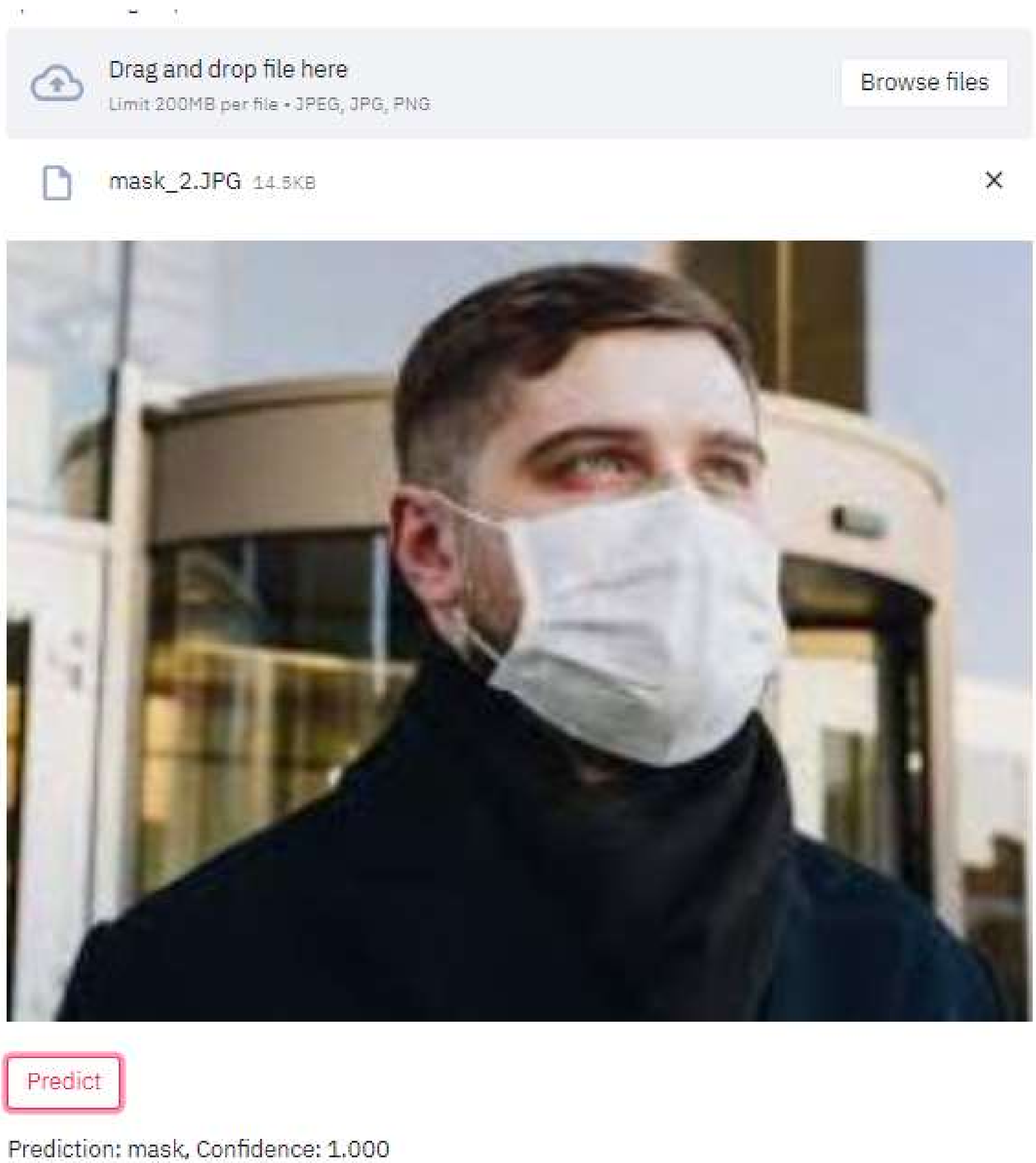


Figure 27. Random Guy Wearing Mask - Correct, 100% Confidence

Next steps would be to implement this app on the cloud so that it can be accessed online as opposed to only locally, given that the ML model is already available on the GCP. Docker would be one such method for getting the necessary files packaged for such ends. As far as the performance of this model, it was evident that the model *generally* performed well on classifying whether a person was wearing a mask or not, however the model would benefit from additional training on images of darker skinned individuals. It's also recognized that better performing CNNs (like Xception or ResNet) could help improve prediction accuracy as well.