1 Topic

The goal of this project is to build and train a neural network model to recognize the cutting vegetables action. At the end, the results will be displayed in GUI (developed using Python Tkinter.)

Please note that while my original topic was "cooking," I narrowed it down to "cutting vegetables" upon Professor Jiang's suggestion, and may add more categories (e.g. "stirring," etc.) if time permits.

2 Resources

The model was trained on the Tamu HPRC Ada cluster. Training took about 30-60min. All data, scripts, and weights were stored on HPRC, as other methods (Google Colab) were too slow and limited in resources.

3 Dataset

Data collection was arguably the most challenging aspect of this project. A model is only as good as the data it is trained on; and the lack of a large, diverse dataset led me to create my own dataset.

The model is trained on 2 categories:

1. Cutting
2. Not Cutting

There are **2243** and **2702** images in each category, respectively. While this is an entirely even distribution, I plan on gathering more data for the "Cutting" section later, which will help to equalize the distribution.

Both categories are discussed in detail below.

3.1 Cutting

Data from 2 research studies was used to generate the data for this section.

1. The KIT Robo-Kitchen Activity Data Set
2. MPII Cooking 2 Dataset

3.1.1 Robo-Kitchen Dataset

The first study, the Robo-Kitchen Dataset, provided videos for a variety of actions – e.g. cutting, peeling, sweeping – on a diverse range of people. The benefit of this dataset was that videos were recorded from different angles, and captured the entire person's body. I used the videos from the "cutting vegetables" category, using the door, window, fridge, and sink camera angles, respectively. Each resultant category contained between 10-20 videos, and were split into frames at the rate of 0.5 frame/sec.

3.1.2 MPII Cooking Dataset

In contrast to the previous dataset, which focused solely on the act of cutting, these videos recorded preparation of an entire meal – e.g. preparing a salad, sandwich, or fruit snack. As such, I had to manually edit every video to remove sections such as washing vegetables/opening the fridge/other non-cutting movement in the kitchen.

Figures: Sample image from the Robo-Kitchen Dataset, MPII Cooking Dataset

3.1.3 Data Cleansing

For both categories, I had to manually parse the videos to ensure their suitability. Common issues encountered were:

1. Subject was seated instead of standing
2. Poor video angle
    a. Subject was turned away from the camera, and the cutting was not visible
    b. Subject did not fit within the camera frame, leading to omission of the head/shoulders
    c. Random table objects obstructed the view
    d. Subject posture while cutting was irregular and not very interpretable

Despite these efforts, I am aware that non-ideal images were still included in the dataset. This is largely because subjects in the video would make occasional non-cutting movements (placing cut vegetables into a bowl/arranging food on the cutting board) that were interspersed within the videos and too time-consuming to remove.

All images were also resized to (640, 480), where 640=image height, and 480=image width. This was for compatibility with the Robo-Kitchen Dataset, which constituted the majority of the training images.

3.1.4 Data Annotation

Images were annotated using Open-Pose-Keras, a Keras implementation of the open-pose library. While I had spent several hours trying to configure the CMU OpenPose library, I had many issues with the Caffe installation, and switched to a Keras implementation, which was much easier to download (and configure).

This library supported 18 different body points (eyes, ears, wrist, elbows, neck, hips, etc.). However, while annotating images, I realized only the upper torso was relevant to the image. The lower torso (hips, knees, ankles) did not contribute to the act of cutting, and moreover, added unnecessary detail to the annotated image. Thus, I edited the library script to remove the hips, knees, and ankle points. The images below illustrate this improvement.

Figures: Annotated image from the MPII Cooking Dataset: full body vs. only upper torso

Note how the presence of the hip/knee/ankle lines add more confusion to the original image (for the viewer, and most likely for the model as well). Removing them resulted them in a much cleaner image, more interpretable image.

3.1.5 Data Organization

The data hierarchy is rather unusual, as I built it up over time, instead of downloading a complete dataset. As such, I wrote 3 different Python scripts to split videos into frames – all dealing with different folder arrangement hierarchies. The GitHub provides more explanation, but due to the large number of images, I was unable to upload the entire Images/ folder onto the GitHub. However, for model prediction, the only thing required is the saved weights, which is provided on the GitHub.

3.2 Not Cutting

I included this category as a counterweight to the "Cutting" category. Without negative examples for the model to train on, it simply predicted every input image as "Cutting."

While many types of images could be classified as "Not Cutting," I sought to include human-based actions within this category, to better contrast against the "Cutting" category. To best accomplish this, I downloaded the HMDB51 Dataset, which has nearly 7000 short video clips distributed among 51 categories. From this list, I used 200 images from 10 categories (climb, dribble, golf, kick_ball, pullup, push, pushup, shoot_bow, situp, swing_baseball), sampled at 0.5 frames/sec.

These images were not annotated because of time constraints (although may be for future submissions). In addition, annotation often produced poor/incomplete results due to the action-based context of the images, and poor image resolution.

Figures: cartwheel, climb_stairs

3.2.1 Extra categories

After the first model run, I sought to gain information about false positives – images that the model was incorrectly classifying as "Cutting:" I scraped the internet for a variety of random actions (playing instruments/sports/etc) and discovered these false positives:

1. Eating (with knives/forks/chopsticks)
    a. Posture can be similar to that when cutting vegetables, especially due to similarity of the tool
2. Writing (with pen/pencil)
    a. Posture can be similar to that when cutting vegetables, especially due to similarity of the tool
3. Empty kitchen
    a. This is most likely representative of images without humans.

To remedy this, I found extra YouTube videos (included at the end of the report) related to these categories, and added them to the "Not Cutting" dataset.

4 Model

Because the dataset consisted of images, I used a CNN model to best capture the annotated images. While the annotations could have been captured with only a feedforward network, other items in the image, such the presence of a knife/vegetable, were also essential to learn.

Initially, I experimented with various combinations of convolutional layers/channel sizes, etc. However, given the real-world nature of the data, using pretrained weights from ImageNet is very sensible, as useful information would be carried over.

4.1 Architecture

My final model was a pretrained InceptionV3 model. All base layers were frozen, and I added my own classifier on top (2DGlobalAveragePooling, Dropout, and a Dense layer) which was trained.

The following code snippet illustrates the model architecture.

```
base_model = InceptionV3(weights = 'imagenet', include_top = False, input_shape =
(IMG_HEIGHT, IMG_WIDTH, 3)) # FIXME: Input_shape!
x = base_model.output
x = GlobalAveragePooling2D(name = 'avg_pool')(x)
```

```
x = Dropout(0.4)(x)
x = Dense(num_classes, name = 'preprediction')(x)
predictions = Activation('softmax', name = 'prediction')(x)
model = Model(inputs = base_model.input, outputs = predictions)

# Freeze all previous layers
for layer in base_model.layers:
layer.trainable = False

# Compile model - default learning rate = 0.001 for RMSprop
model.compile(loss = 'categorical_crossentropy', optimizer = 'rmsprop', metrics =
['accuracy', 'top_k_categorical_accuracy'])
print(model.summary())
```

4.2 Input Tensor Shape

Training data: (2966, 480, 640, 3)

Validation/Testing data: (989, 480, 640, 3)

4.3 Output Shape

Output: 2 probabilities (Cutting/Not Cutting)

4.4 Model Parameters

1. Train:Validation:Test Ratio = 60:20:20
2. Num_classes = 2
3. Batch_size = 32
4. Loss function = Categorical Crossentropy
5. Optimizer = RMSProp
6. Learning Rate = 0.001
7. Epochs = 10

Data Augmentation (achieved by horizontally flipping images) was also used. Not only did this increase the amount of data, but also make it more balanced, as the horizontal flip generated images of left-handed people, not just right-handed, cutting vegetables.

6 Annotated Code

Utility Functions

```
def print_training_data(acc, val_acc, loss, val_loss):
  print('Training Accuracy:\n', acc)
  print('Validation Accuracy:\n', val_acc)
  print('Training Loss:\n', loss)
  print('Validation Loss:\n', val_loss)

def write_data_to_file(acc, val_acc, loss, val_loss):
  # Write data to .csv file
  # Note: Clear data.csv each time! After clearing, add '0' to m
  open('data.csv', 'w+').close() # Clear file before writing to
  with open('data.csv', 'w') as f:
    f.write('0') # Add a value
  f.close()
  print('Writing data to .csv file...')
  data = pd.read_csv('data.csv', 'w')
  data.insert(0,"Training Acc", acc)
  data.insert(1,"Training Loss", val_acc)
  data.insert(2,"Validation Acc", loss)
  data.insert(3,"Validation Loss", val_loss)
  data.to_csv('data.csv')
  print('Finished writing data!')

def plot_graphs(acc, val_acc, loss, val_loss):
  # Plot results
  print("Starting plotting...")
  epochs = range(1, len(acc)+1)
  plt.plot(epochs, acc, 'bo', label='Training accuracy')
  plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
  plt.title('Training and Validation accuracy')
  plt.legend()
  plt.savefig('plots/Cooking_TrainingAcc.png')
  plt.figure()
  plt.plot(epochs, loss, 'bo', label='Training loss')
  plt.plot(epochs, val_loss, 'b', label='Validation loss')
  plt.title('Training and Validation loss')
  plt.legend()
  plt.ylim([0,4]) # Loss should not increase beyond 4!
  plt.savefig('plots/Cooking_TrainingLoss.png')
  plt.figure()
```

Load Data

```
# Load data into 60:20:20 split
def load_data(NUM_CLASSES, IMG_HEIGHT, IMG_WIDTH, seed=None, root_dir=None):
  if(root_dir==None):
    print("Please enter a valid data directory!")
    return

  random.seed(seed)

  def read_from_paths(paths):
    x=[]
    for path in paths:
      # print('path: ', path)
      img = load_img(path, target_size=(IMG_HEIGHT, IMG_WIDTH))
      img = img_to_array(img)
      x.append(img)
    return x

  classes = os.listdir(root_dir)
  classes = sorted(classes)

  train_path = []
  val_path = []
  test_path = []

  train_x, train_y = [],[]
  val_x, val_y = [],[]
  test_x, test_y = [],[]

  # Read paths and split data into 6:2:2 dataset
  for i, _class in enumerate(classes):
    paths = glob.glob(os.path.join(root_dir, _class, "*"))
    paths = [n for n in paths if n.endswith(".JPG") or n.endswith(".jpg")]
    random.shuffle(paths)
    num_plants = len(paths)
    # print("num_plants: ", num_plants)

    train_path.extend(paths[:int(num_plants*0.6)])
    train_y.extend([i]*int(num_plants*0.6))

    val_path.extend(paths[int(num_plants*0.6):int(num_plants*0.8)])
    val_y.extend([i]*len(paths[int(num_plants*0.6):int(num_plants*0.8)]))

    test_path.extend(paths[int(num_plants*0.8):])
    test_y.extend([i]*len(paths[int(num_plants*0.8):]))

  print("Loading images...")

  train_x = read_from_paths(train_path)
  print("Loaded all training images!")
  val_x = read_from_paths(val_path)
  print("Loaded all validation images!")
  test_x = read_from_paths(test_path)
  print("Loaded all testing images!")
```

Load Model

```
# Construct pretrained InceptionV3 model
def get_inceptionV3(num_classes, IMG_HEIGHT, IMG_WIDTH):

  base_model = InceptionV3(weights='imagenet',
  include_top=False, input_shape=(IMG_HEIGHT, IMG_WIDTH, 3)) # FI

  x = base_model.output
  x = GlobalAveragePooling2D(name='avg_pool')(x)
  x = Dropout(0.4)(x)
  x = Dense(num_classes, name='preprediction')(x)
  predictions = Activation('softmax', name='prediction')(x)
  model = Model(inputs=base_model.input, outputs = predictions)

  # Freeze all previous layers
  for layer in base_model.layers:
    layer.trainable = False

  # Compile model - default learning rate = 0.001 for RMSprop
  model.compile(loss='categorical_crossentropy',
    optimizer='rmsprop',
    metrics=['accuracy'])
  print(model.summary())
  return model
```

Process Data

```
# Convert all to numpy
train_x = np.array(train_x)/255.
train_y = np.array(train_y)
val_x = np.array(val_x)/255.
val_y = np.array(val_y)
test_x = np.array(test_x)/255.
test_y = np.array(test_y)

# Calculate class weight
classweight = class_weight.compute_class_weight('balanced',
  np.unique(train_y), train_y)

# Convert to categorical (~1-hot encoding)
train_y = np_utils.to_categorical(train_y, NUM_CLASSES)
val_y = np_utils.to_categorical(val_y, NUM_CLASSES)
test_y = np_utils.to_categorical(test_y, NUM_CLASSES)
print("Successfully loaded data!")
```

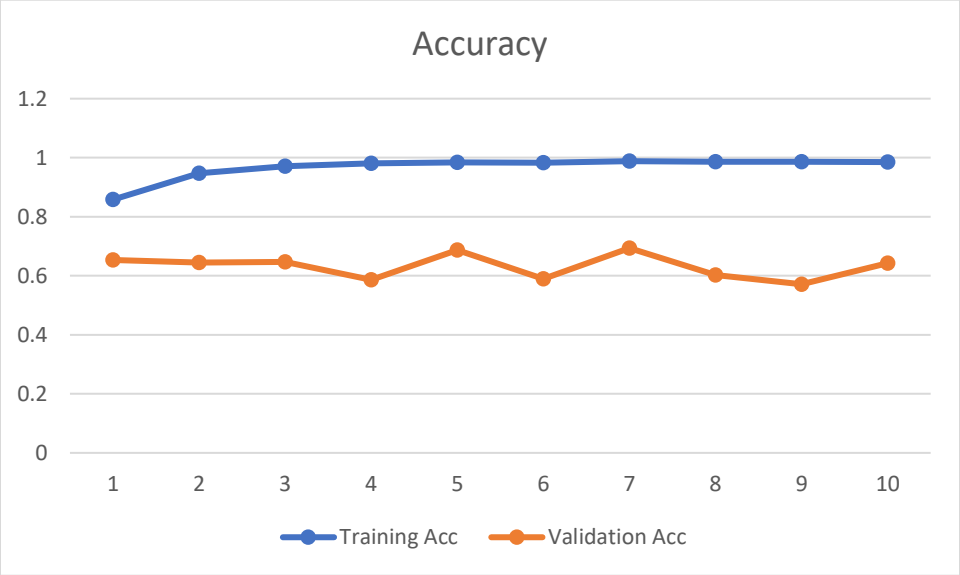
Main

Evaluate and Plot

```
# ------------------------------ Main ------------------------------ #

# Define parameters
NUM_CLASSES = 2
IMG_HEIGHT = 480
IMG_WIDTH = 640

# Load model
model = get_inceptionV3(NUM_CLASSES, IMG_HEIGHT, IMG_WIDTH)

# Load data - will take a while
train_x, val_x, test_x, train_y, val_y, test_y, classweight, classes =
load_data(NUM_CLASSES, IMG_HEIGHT, IMG_WIDTH, seed=7, root_dir=data_dir)

print(train_x.shape, val_x.shape, test_x.shape)
print(train_y.shape, val_y.shape, test_y.shape)

# Data Augmentation: Flip images horizontally
train_datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    horizontal_flip=True)
val_datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    horizontal_flip=True)
train_datagen.fit(train_x)
val_datagen.fit(val_x)

# Define callbacks & begin training
es = EarlyStopping(patience=5)
history = model.fit_generator(train_datagen.flow(train_x, train_y, batch_size=32),
  epochs=10, steps_per_epoch=len(train_x)/32,
  validation_data=val_datagen.flow(val_x, val_y, batch_size=32),
  callbacks=[es], class_weight=classweight)
```

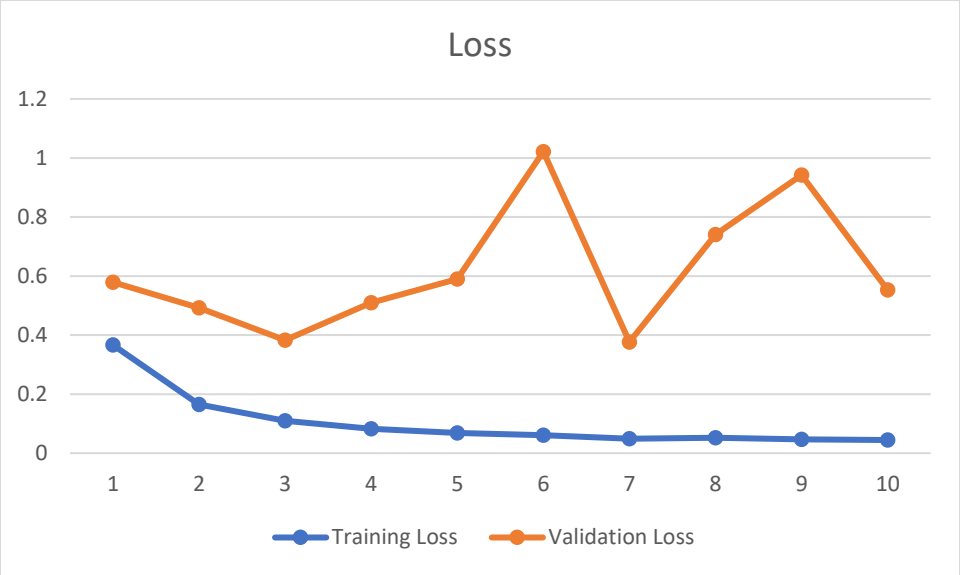
```
# Save model
print ("Saving model...")
model.save('cooking2.h5')

# Save training parameters
print('Obtaining training data...')
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

print_training_data(acc, val_acc, loss, val_loss)

# Get testing results
test_results = model.evaluate(test_x, test_y)
print("Printing test results...")
for metric, name, in zip(test_results, ['loss', 'acc', 'top 5 acc']):
  print(name, metric)

write_data_to_file(acc, val_acc, loss, val_loss)
plot_graphs(acc, val_acc, loss, val_loss)

print("Model finished!")
```

7 Training and Testing Performance



Training Accuracy: 98%

Validation Accuracy: 64%



Test Results

```
 32/990 [..............................] - ETA: 2:38
 64/990 [>.............................] - ETA: 2:35
 96/990 [=>............................] - ETA: 2:29
128/990 [==>...........................] - ETA: 2:23
160/990 [===>..........................] - ETA: 2:19
192/990 [====>.........................] - ETA: 2:07
224/990 [=====>........................] - ETA: 1:58
256/990 [======>.......................] - ETA: 1:49
288/990 [=======>......................] - ETA: 1:42
320/990 [========>.....................] - ETA: 1:35
352/990 [=========>....................] - ETA: 1:30
384/990 [==========>...................] - ETA: 1:24
416/990 [===========>..................] - ETA: 1:19
448/990 [============>.................] - ETA: 1:14
480/990 [=============>................] - ETA: 1:09
512/990 [==============>...............] - ETA: 1:04
544/990 [===============>..............] - ETA: 59s
576/990 [================>.............] - ETA: 55s
608/990 [=================>............] - ETA: 50s
640/990 [==================>...........] - ETA: 46s
672/990 [===================>..........] - ETA: 41s
704/990 [====================>.........] - ETA: 37s
736/990 [=====================>........] - ETA: 32s
768/990 [======================>.......] - ETA: 28s
800/990 [=======================>......] - ETA: 24s
832/990 [========================>.....] - ETA: 20s
864/990 [=========================>....] - ETA: 16s
896/990 [==========================>...] - ETA: 11s
928/990 [===========================>..] - ETA: 7s
960/990 [============================>.] - ETA: 3s
990/990 [==============================] - 125s 126ms/step
Using TensorFlow backend.
Printing test results...
loss 0.039118268814953894
acc 0.991919219493866
top 5 acc 1.0
Writing data to .csv file...
Finished writing data!
```

Testing Accuracy: 99%

The large discrepancy between the training and validation accuracy is a clear indication of overfitting. This is most likely because of the small/non-diverse data the model is trained on. Increasing data diversity, in addition to regularization (to a smaller extent) will help to remedy this.

8 Test Video Results

5 YouTube videos were downloaded, trimmed, split into frames, and then fed into the network. While the plot and .JSON file are already provided, I wanted to give some insights into the results:

8.1 Test Video 1

This video features a young child cutting broccoli for roughly 30sec, and then not cutting for the remainder of the time. The model does a good job of classifying the first 30 sec, but then occasionally errs during the latter portion of the video, resulting in the spikey graph.

8.2 Test Video 2

This video features a woman teaching pancake flipping. It has absolutely nothing to do with cutting vegetables (which is the point). The model successfully classifies the whole video as "not cutting."

8.3 Test Video 3

This video features a woman demoing spinach preparation. While only a small portion (seconds 15-25) are actually focused on cutting, the model does not classify them, but instead produces a very spikey result. Upon examination of the individual probabilities, we see that the predictions for several frames are quite close (e.g. 40-60), and that the model must be trained on a more diverse set of data to better capture this.

8.3 Test Video 4

Same analysis as #5. The watermark in the video might also be an obstacle.

8.3 Test Video 5

This video features a 'super onion-cutter.' While the subject is cutting throughout the entire video, the model only captures several motions as cutting. This is for several reasons:

1. Model is not trained to detect closeups of only hands cutting (it is trained on the whole person)
2. Several times the video focuses on the subject's face/other people
3. Frames capture movement in real-time and thus the knife object is sometimes not clearly visible

Regardless, the model should have output a higher probability for "Cutting." More diverse data will help to resolve this.

In addition, one might wonder (as I do) why such (occasionally poor) test videos were chosen. In all honesty, due to the challenging nature of finding videos that contain the whole person's body – it was not easy to find suitable videos, and those that were suitable I used for training instead.

9 How to Reproduce

9.1 Individual Images

Download the gui/ folder, and the .h5 file containing the weights. Run "python Cooking_GUI.py" to run the GUI. (Alternatively, the .ipynb file can be run in Google Colab/Jupyter). The GUI allows classification of individual images.

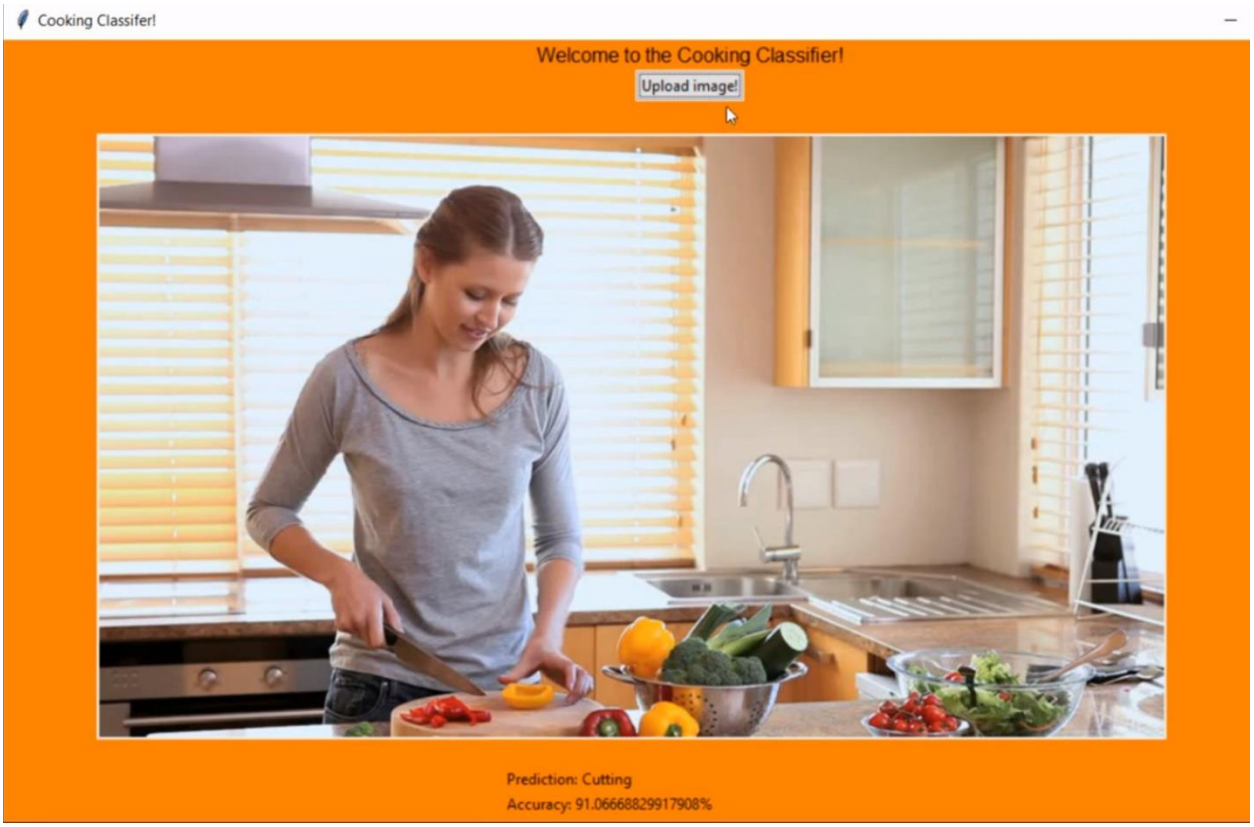
9.2 Videos

Download the predict.py script, and the .h5 file containing the weights. Specify the file location of the input images, then run "python predict.py" to run the predictor. Note: script assumes the video is already split up into frames.

9.3 Demo Link

This link directs to the GUI. Note that the 1st classification will take some time, as the model needs to setup. (This GUI remains a work in progress.)

https://youtu.be/Y-2u3SsR188



10 Next steps

1. Collect more data! (As much as possible)
    a. Take videos of family cutting vegetables
    b. Take videos at arbitrary angles, different types of vegetables, lighting conditions, etc
2. Annotate pictures with body *and* hand positions
3. Use Attention Maps to visualize what model is learning (and fine-tune accordingly)
4. Experiment with hyperparameters. (This did not happen enough in this stage as I was more focused on gathering a wholesome dataset)

11 Extra YouTube Videos

Eating with chopsticks:

1. https://www.youtube.com/watch?v=2GDLlETO0-w

2. https://www.youtube.com/watch?v=GnbSwkS_kxY

Writing:

1. https://www.youtube.com/watch?v=eh4b5zC0sB4

Empty room:

2. https://www.youtube.com/watch?v=Hs_6-NrcgBg

Testing:

1. https://www.youtube.com/watch?v=e2pWpKlBheo
2. https://www.youtube.com/watch?v=mTsXuPCZbRY
3. https://www.youtube.com/watch?v=p3x_VXcfLf8
4. https://www.youtube.com/watch?v=rguE2xe96pY
5. https://www.youtube.com/watch?v=qDFc-5Zc3HU