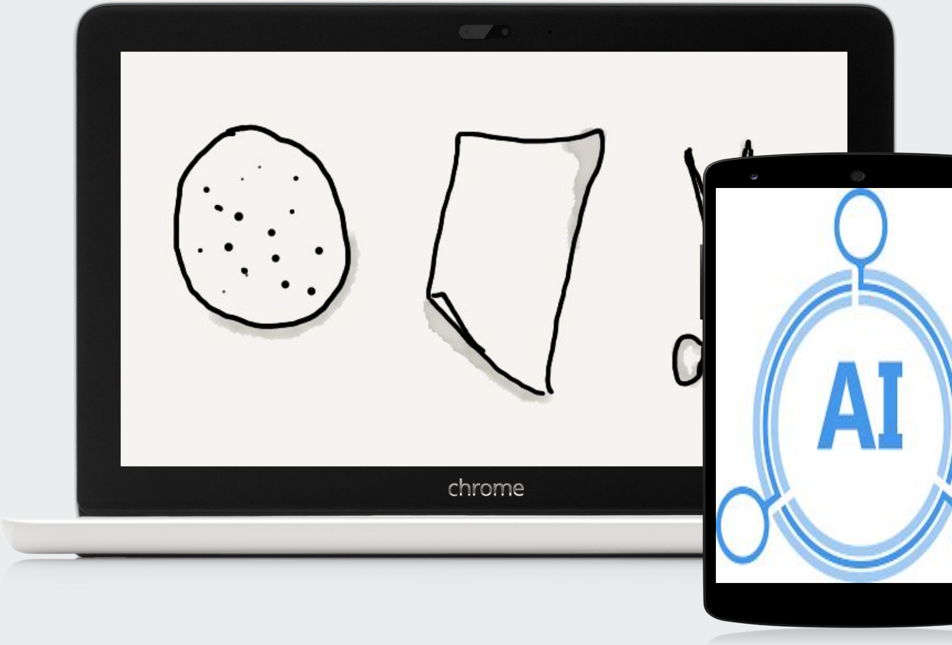
PaiRS App

An AI implementation of the game Rock, Paper, Scissors

By: Stacey Frasier, Brandon Hawkinson, Jonathan Mouchou, Michael Perna



Outline

Intro

01

Why Rock Paper Scissors?

- Provided a suitable challenge that was less daunting than our original goal of recognizing resistors.
- The potential applications of detecting hand shapes and signals:
  - Recognizing Sign language
  - Understanding biker's hand signals(Self driving cars)
- Provided a solid, achievable, demo-able product we can do in 3 months.

Where did we start?

- Never played with CNNs.
- Had some experience with ML using linear regression models.
- A general plan on what we needed to do to achieve our goal.
- Have built some apps in the past.

Implementation Overview

What methods we used to train the model

- Data
  - Over 1,900 images we took ourselves
- Keras
  - CNN
  - Tensorflow-CPU
  - Tensorflow-GPU via FloydHub
- App
  - Flutter
  - Tflite

Training the Model

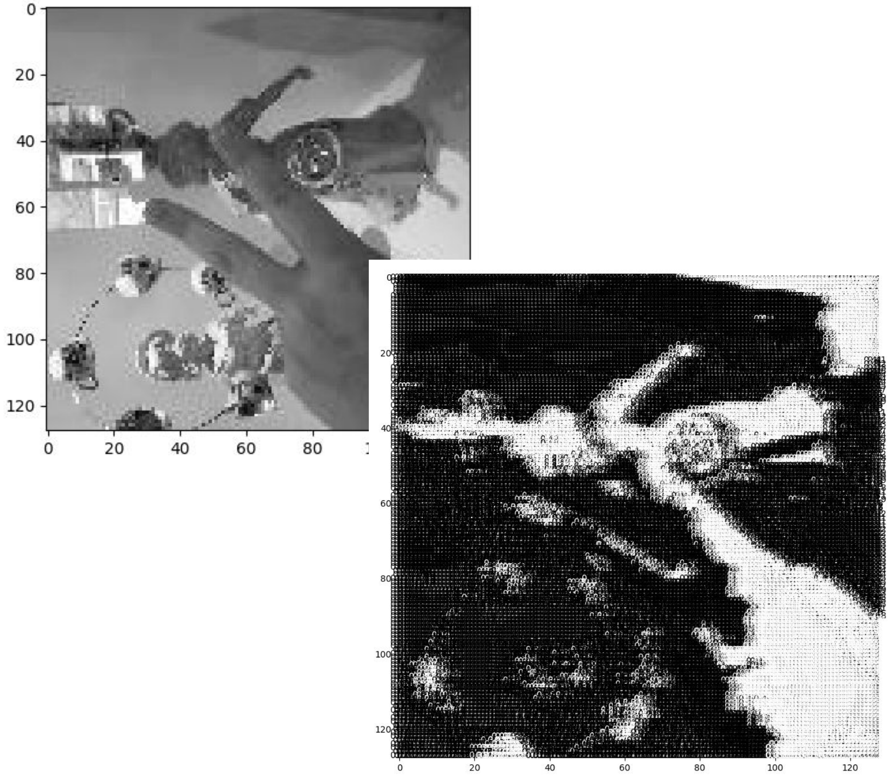
02

Our data

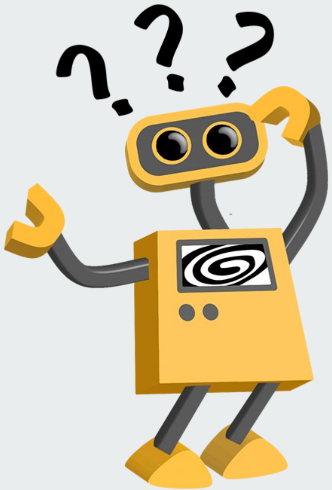
- As we said previously, over 1,900 images we took ourselves.
  - The original data set overfit tremendously, we were getting 100% accuracy on our model with 200 images.
  - We wanted to get real world examples
    - All photos pretty much have different backgrounds with different hand angles.
    - Knew that keras had functionality to manipulate the data as we trained as well.
  - Created a script with Augmentor(https://github.com/mdbloice/Augmentor) that generated over 15,000 images that we trained on.
    - Flips, Skews, Inverts and Zooms via probability to create a new image based on existing ones
- We wrote a script that resized them to our desired resolution of 128x128
  - Turns out the more data you have in an image, doesn't necessarily mean it'll be more accurate.
  - Little improvement between 128x128 and 256x256
  - Script allowed us to easily import more data once we got it.

What does an image look like to a computer?

We used MatPlotLib and an example script from the keras documentation to give us an accurate representation of what the computer sees when it digests an image.



How did we (humans) learn?



- We began by following CNN image classification examples from Keras documentation.
- Main example trained on CIFAR-10 dataset (10 classes).
- From there we had to learn how to apply the techniques used in these example networks to our dataset.
- Additional Assistance:
  - Guidance from Professor Avery (Our CS 483 - Introduction to Machine Learning professor)
  - Quality of life improvements (checkpoints) recommendations and guidance by the in class helper.
  - Professor Han for project guidance.

Training the model (For real)

How did we train the model once we figured everything out?

- We waited... and waited... and waited.
- Trained for over 40 Hours on Tensorflow-CPU
  - Tensorflow-GPU has some trouble with Keras locally for us.
  - Over 10,000 Epochs were run
  - Results were subpar
- The results from this was subpar
- We really needed to find a better solution.

When you fork out $12 for a GPU instance with Keras and Tensorflow-GPU preinstalled.

Training on the GPU instance

- Much faster results!
- Allowed us to really push the model to its boundaries and still train in a reasonable amount of time.
- Instance from floyd hub
  - 2 Hours Free
  - $12 for 10 hours
  - Stats
    - 12gb Nvidia Tesla K80 GPU
    - 61gb RAM
    - 100gb SSD
    - Ubuntu 16.04
    - Easy Github Integration
- Trained for 6 hours and ended up with our working model that we’ll be showing today.

```python
# define the model
model = Sequential()


model.add(Conv2D(filters=4, kernel_size=(3, 3),
                 padding='same', activation='relu', input_shape=input_shape))
model.add(Conv2D(filters=8, kernel_size=(3, 3),
                 padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.3))
model.add(Conv2D(filters=12, kernel_size=(3, 3),
                 padding='same', activation='relu'))
model.add(Conv2D(filters=16, kernel_size=(3, 3),
                 padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.3))
model.add(Conv2D(filters=20, kernel_size=(3, 3),
                 padding='same', activation='relu'))
model.add(Conv2D(filters=32, kernel_size=(3, 3),
                 padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Flatten())
model.add(Dropout(0.3))
model.add(Dense(128, activation='relu'))
model.add(Dense(3, activation='softmax'))

opt = keras.optimizers.RMSprop(lr=1e-04, decay=1e-6)
#opt2_ = keras.optimizers.SGD(lr=1e-04)
# old_opt = keras.optimizers.RMSprop(lr=1e-04);
model.compile(loss='categorical_crossentropy',
              optimizer=opt, metrics=['accuracy'])

filepath = "saved_models/rps.h5"
checkpoint = ModelCheckpoint(
    filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')
callbacks_list = [checkpoint]

if (os.path.isfile(filepath)):
    model.load_weights(filepath)
    loss, acc = model.evaluate(x_train, y_train)
    print("Restored model, accuracy: {:5.2f}%".format(100*acc))

# train the model
model.fit(x_train, y_train, batch_size=2048, epochs=600,
          validation_data=(x_valid, y_valid), verbose=1, shuffle=True, callbacks=callbacks_list)
```

Convolutional layers tells the model to check 4x4 blocks for similarities

Pooling layers combine Convolutional layers help control overfitting

Dropout layers remove some percentage of connections made, allows the model to train longer by removing random connections.

Adjusts the results vector to be completely vertical, makes dropout a lot more efficient

Still not exactly sure what Dense layers do.

Spooky numbers 👻 (They're really hyperparameters to help control how fast we train)

Tells the model to output accuracy as its training metric

Only save models where we decrease our validation loss

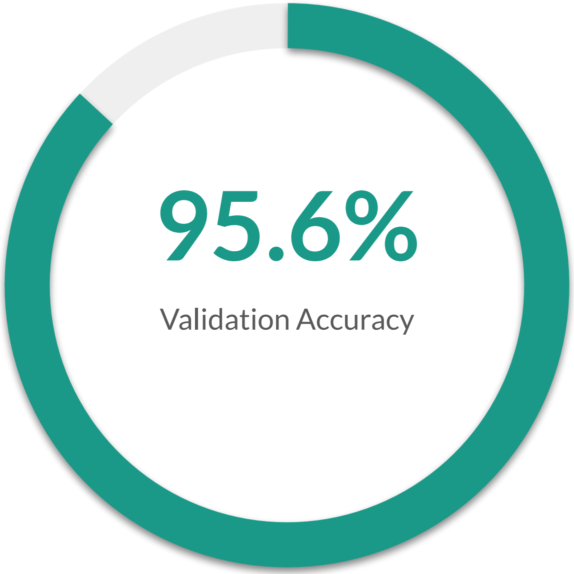
Load our previous checkpoint if it exists

We train the pictures in sets of 2048, iterate over the whole set, we do this 600 times

Dark themes are cooler >:D

Results

- With our data we were able to achieve 95.6% accuracy on our latest model.
    - 40% Validation Set Size
- This is highly subjective to our data set.
- Performance is best on solid backgrounds.
- Still have trouble with complex backgrounds

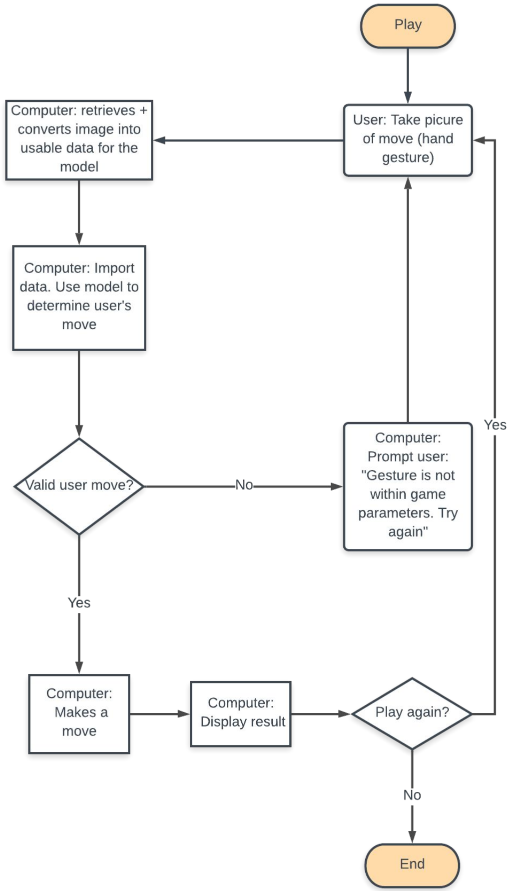


Mobile Application

03

Implementation

- Flutter
- Camera Module
- Tflite
- Game Logic



Deploying the model.

- With our awesome new PaiRS model, what can we do with it? Use it of course!
- Just take the model and place it on the phone and BAM! It all should work!

Not so fast...

Deploying the model.

- Convert our .h5 (Keras) model into a .tflite (Tensorflow Lite) model.
    - Thankfully the python tensorflow library had a function to convert! If we did this project last semester, it would've been nasty.
- What to do with the new model?
    - Use tflite flutter library and load it in!
- Ran into input problems
    - Normalization
    - Resizing the image to the proper input

If you can't tell, we really like dark themes 😎

```
void _takePicturePressed() {
  takePicture().then((String filePath) {
    if (mounted) {
      print("US" + filePath);
      ImageFluxer.Image img =
        ImageFluxer.decodeImage(new File(filePath).readAsBytesSync());

      ImageFluxer.Image smallImage = ImageFluxer.copyResize(img, 128);
      smallImage = ImageFluxer.normalize(smallImage, 0, 255);

      new File(filePath).writeAsBytesSync(ImageFluxer.encodeJpg(smallImage));
    }
    appLogic(filePath);
  });
}
```

- Had to resize and normalize our data before saving it
- Load the model each time we run the appLogic
- 60% prediction Threshold
  - Must be better than 60% sure
- RNG for the computer's move
  - The one true AI

```
Future appLogic(String filePath) async {
  String res = await Tflite.loadModel(
    model: "assets/rps.tflite",
    labels: "assets/labels.txt",
  );

  var recognitions = await Tflite.runModelOnImage(
    path: filePath,
    threshold: 0.6,
  );

  _prediction = recognitions.elementAt(0)["label"];
  print(_prediction);

  var rng = new Random();
  var guess = rng.nextInt(100);

  if (guess <= 32) {
    _random = "rock";
  } else if (guess > 32 && guess <= 65) {
    _random = "paper";
  } else if (guess > 65 && guess <= 99) {
    _random = "scissors";
  }

  if (_random == "scissors" && _prediction == "paper") {
    _state = "You lose, scissors cuts paper!";
  } else if (_random == "paper" && _prediction == "rock") {
    _state = "You lose, paper covers rock!";
  } else if (_random == "rock" && _prediction == "scissors") {
    _state = "You lose, rock smashes scissors!";
  } else if (_prediction == "scissors" && _random == "paper") {
    _state = "You win, scissors cuts paper!";
  } else if (_prediction == "paper" && _random == "rock") {
    _state = "You win, paper covers rock!";
  } else if (_prediction == "rock" && _random == "scissors") {
    _state = "You win, rock smashes scissors!";
  } else {
    _state = "Draw or unknown";
  }

  setState(() {});
}
```

Demo(s)

04

In Closing...

- We wish we had some more experience training models, we feel this could be way more accurate given our data set.
- We also wish we had more data, having a more diverse set of hands (size, color, shapes) on more backdrops would lead to better results.
- We chose something we felt would back us into a corner purposefully.
  - It did, and it was a very rewarding experience being able to push through and accomplish this.
  - We all had to learn how to implement and train a deep neural network.
  - Other team members learned how to build an app.

References & Code

- Github: https://github.com/Hawkinsonb/rockpaperscissors-cnn
- Code from or inspired by:
  - Keras Documentation
    - Visualization.py (View of the hand matrix)
    - Preprocessor.py (we ripped a lot of stuff out of this however)
  - Stackoverflow
  - Tensorflow-lite examples
  - Flutter camera examples
- To run: check our README
  - Note: App has only been tested on Android. In theory it should run on iOS because of Flutter.

Questions?