
Score: 92
Coins Collected: 0
Idle Time: 0.0
Last Action Time: 0.0
Time Elapsed: 3.03
Panda Trainer
Quinn Roemer & Logan Hollmer

Introduction

Goal

Explore how AI can perform human-like tasks using the same inputs a human would have.

Solution

Create computer-vision models capable of solving a complex task (playing a game) that use only what is displayed on the current frame as input.

Overview

In this presentation we will discuss the *problem formulation*, *system & algorithm design*, *experimental results & evaluation*, *related work*, & a project *conclusion*. In addition we will talk about *work division* and our *learning experiences*.

Problem Formulation

Problem
Using a single image, predict the correct action in a game.

Solution
A computer vision model capable of predicting the correct action given a single frame from a game.

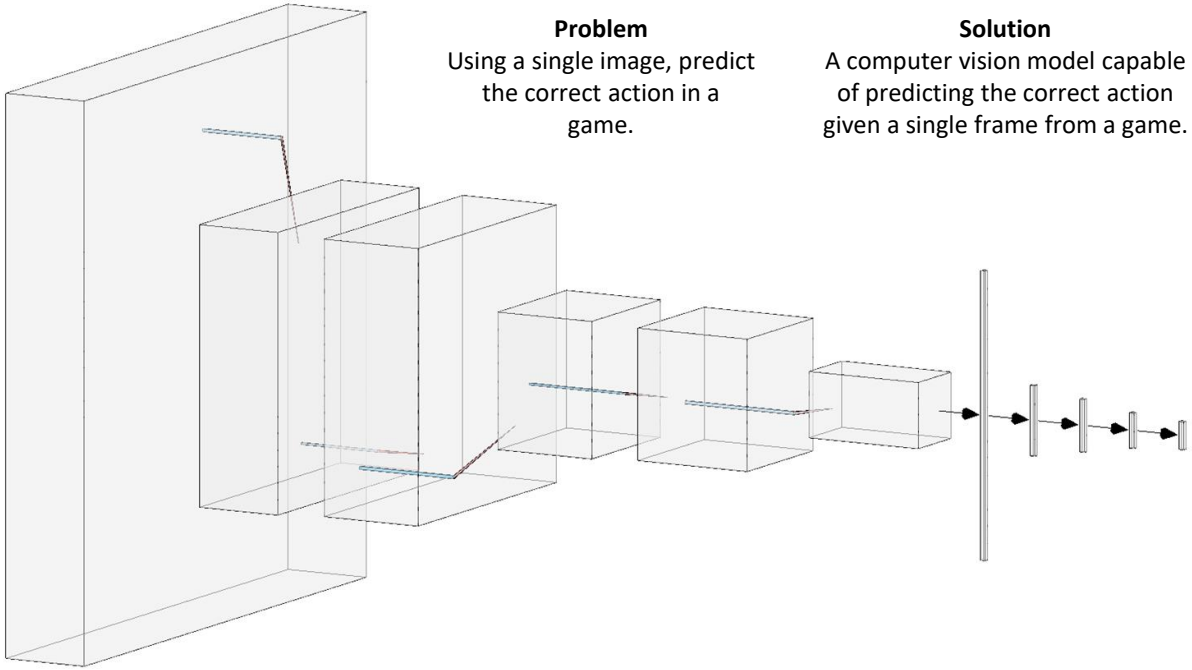
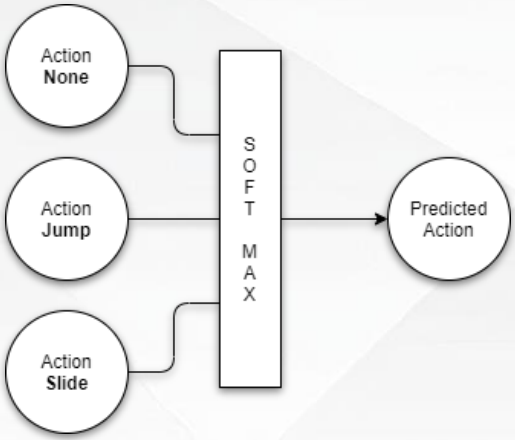


Figure: One of our custom 3 layer CNN models created for this project.

Figure: How actions are predicted in our NN's.



Possible Actions:

- None
- Jump
- Slide

Note: The player automatically moves forward in the game.

System/Algorithm Design

- Game development: Panda Runner

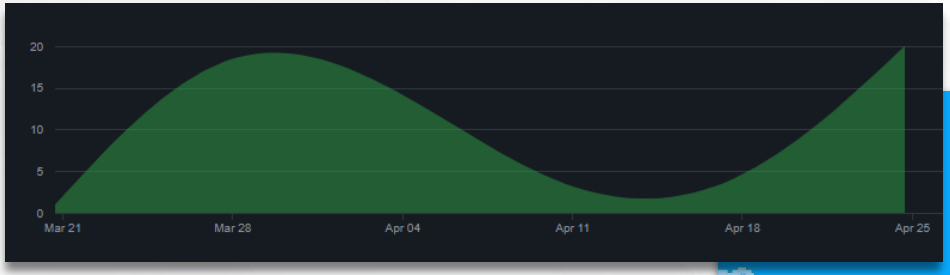


Image: Panda Runner's development history

Information

- 2D Side Scroller
- Written in Python using PyGame
- Human or AI player
- Includes a data recorder & model loader
- Developed from late March to late April
- Over 1.7k lines of code

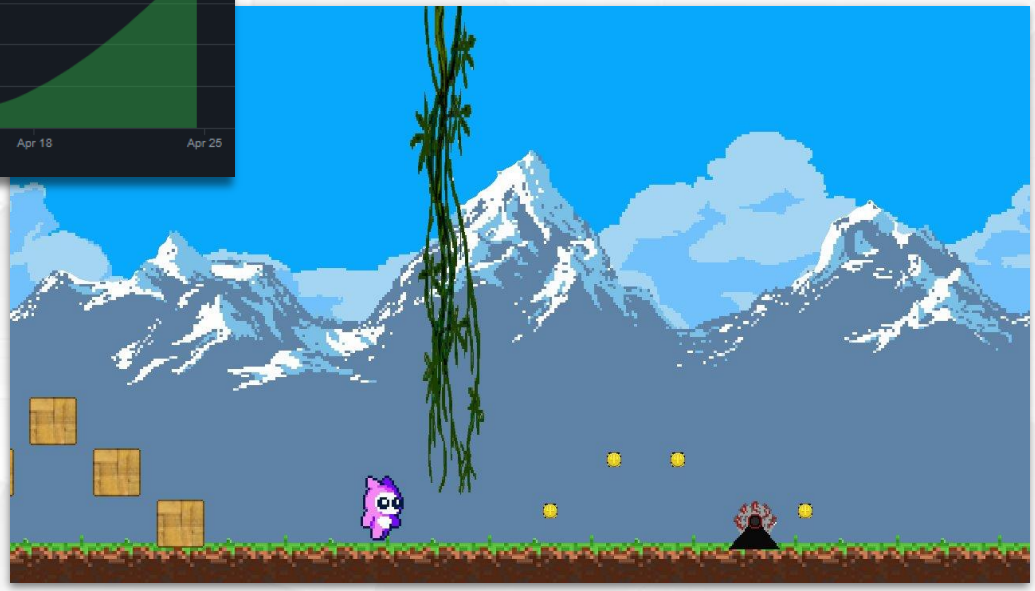


Image: Gameplay of Panda Runner

System/Algorithm Design

- Data Collection and Preprocessing

```
def takeScreenshot(self, keyState):
    if not constants.RECORD_SCREENSHOTS:
        if self.__gWorld.getPlayer().gameReset:
            self.__stat.resetTimeElapsed()
            self.__gWorld.getPlayer().gameReset = False
        return

    if (datetime.now() - self.__lastScreenshot).total_seconds() > constants.SCREENSHOT_FREQUENCY and not self.__stat.levelFinished:
        self.__lastScreenshot = datetime.now()
        screenName = str(self.__lastScreenshot).split(' ')[1].replace(":", ".") + '_' + str(keyState) + '.jpeg'
        pygame.image.save(self.__gWorld.getScreen(), self.__screenShotPath + screenName)

    if self.__iHandle.forcedScreenshot and not self.__stat.levelFinished:
        self.__iHandle.forcedScreenshot = False
        screenName = str(self.__lastScreenshot).split(' ')[1].replace(":", ".") + '_' + str(keyState) + '.jpeg'
        pygame.image.save(self.__gWorld.getScreen(), self.__screenShotPath + screenName)

    #If the game was reset, remove the current folder and all images and create a new one
    if self.__gWorld.getPlayer().gameReset:
        try:
            self.__gWorld.getPlayer().gameReset = False
            shutil.rmtree(self.__screenShotPath)
            self.__lastScreenshot = datetime.now()
            self.__stat.resetTimeElapsed()
            self.__screenShotPath = f'./data/run_{str(self.__lastScreenshot.ctime()).replace(":", ".")}/'

            if not os.path.exists(self.__screenShotPath):
                os.mkdir(self.__screenShotPath)

        except OSError:
            print('Unable to remove files after reset')
```

Image: Function used to take screenshots during data collection in Panda Runner

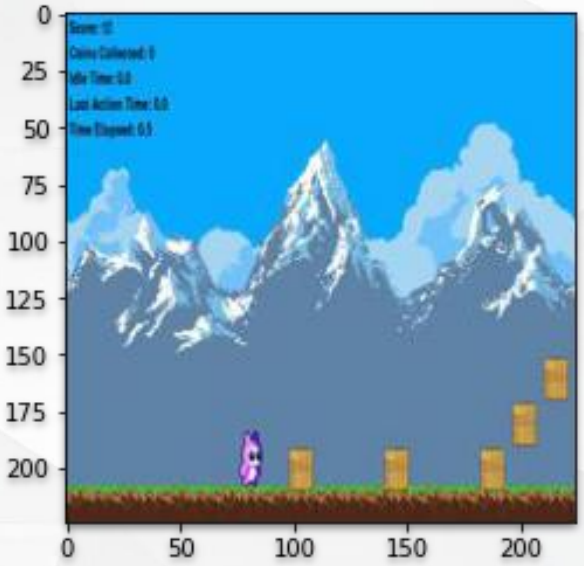


Image: A Preprocessed image ready for a neural network

System/Algorithm Design

- Transfer Learning

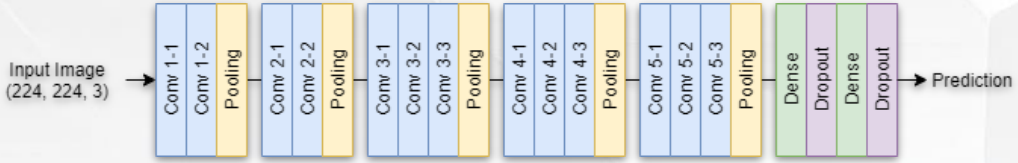


Image: Sample transfer learning model created using VGG16

```
#Define our toolbox
toolbox = base.Toolbox()
toolbox.register("individual", tools.initIterate, creator.Individual, create_individual)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

#Register strategies
toolbox.register("evaluate", evaFitness)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutUniformInt, low = 1, up = 6, indpb=0.1)
toolbox.register("select", tools.selTournament, tournsize=3)

#Statistics
stats = tools.Statistics(key=lambda ind: ind.fitness.values)
stats.register("avg", np.mean)
stats.register("max", np.max)
```

Image: Genetic algorithm toolbox strategies

- Transfer Learning Models
  - VGG16
  - MobileNetV2

- Genetic algorithms for Hyper-Parameter Tuning
  - Each model was represented as a series of 6 values between 0 and 9.
  - ["optimizer", "activation function", "number of trainable layers", "drop out percentage", "number of dense layers at the end", "model type"]
  - The fitness function returned the F1 score of a trained model.
  - Training occurred over a 30 hr period

System/Algorithm Design

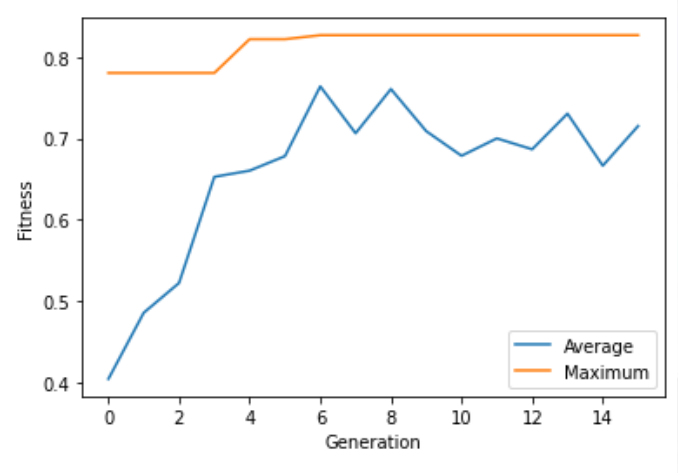
- Custom Learning Models
  - Three custom CNN models were created.
  - Trained via an almost identical genetic algorithm as the Transfer Learning models.
    - Chromosome structure differed:
      - Values ranged from 1 to 6
      - Length = 5
      - Allowed for CNN type selection (the custom CNN model to be created)
      - Fitness function returned the F1 score of the trained model
  - Models trained over several sessions with population counts ranging from 10-30 over 10-20 generations.
  - Example Chromosome:

```
["CNN Type", "Activation Function Type", "Optimizer Typer", "Number of Fully-Connected Layers", "Dropout Chance"]
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_188 (Conv2D)          (None, 222, 222, 32)      896
_________________________________________________________________
activation_54 (Activation)   (None, 222, 222, 32)      0
_________________________________________________________________
max_pooling2d_132 (MaxPoolin (None, 111, 111, 32)      0
_________________________________________________________________
conv2d_189 (Conv2D)          (None, 109, 109, 64)      18496
_________________________________________________________________
activation_55 (Activation)   (None, 109, 109, 8652864)      0
_________________________________________________________________
max_pooling2d_133 (MaxPoolin (None, 54, 54, 64)        0
_________________________________________________________________
conv2d_190 (Conv2D)          (None, 52, 52, 128)       73856
_________________________________________________________________
activation_56 (Activation)   (None, 52, 52, 128)       0
_________________________________________________________________
max_pooling2d_134 (MaxPoolin (None, 26, 26, 128)       0
_________________________________________________________________
flatten_35 (Flatten)         (None, 86528)             0
_________________________________________________________________
dense_352 (Dense)            (None, 16)                1384464
_________________________________________________________________
dropout_346 (Dropout)        (None, 16)                0
_________________________________________________________________
dense_353 (Dense)            (None, 8)                 136
_________________________________________________________________
dropout_347 (Dropout)        (None, 8)                 0
_________________________________________________________________
dense_354 (Dense)            (None, 4)                 36
_________________________________________________________________
dropout_348 (Dropout)        (None, 4)                 0
_________________________________________________________________
dense_355 (Dense)            (None, 3)                 15
=================================================================
Total params: 1,477,899
Trainable params: 1,477,899
Non-trainable params: 0
_________________________________________________________________
```

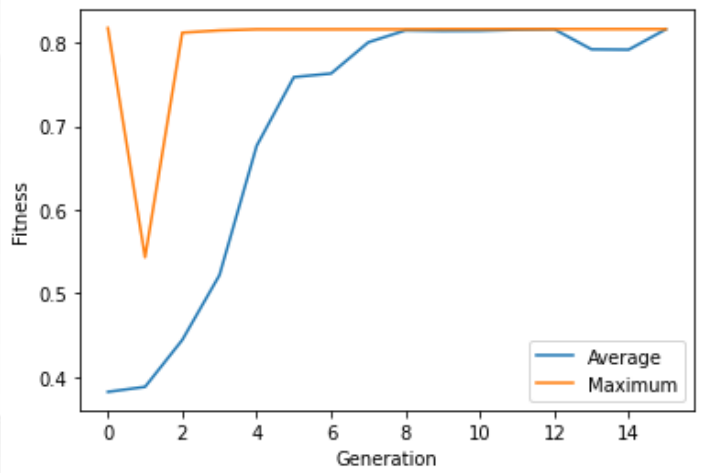
Image: A custom CNN model generated by our genetic algorithm. This particular models chromosome was [6, 3, 5, 1, 1].

Experimental Evaluation (Training)



Transfer Learning Models (Generational Plot)

- Population of 20
- Trained over 15 generations
- Greater than 8hrs of training time for a single run
- Best Model Created: F1 = 0.82667

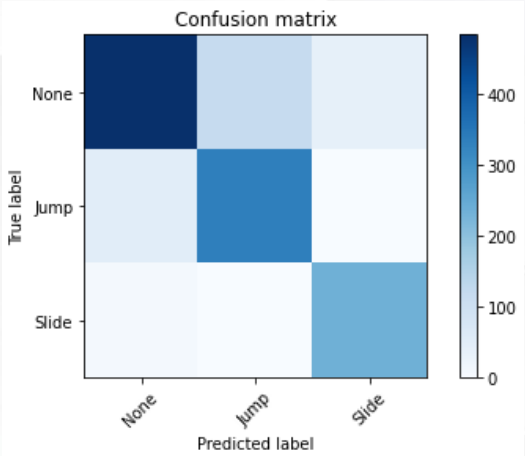


Custom CNN Models

- Population of 20
- Trained over 15 generations
- Greater than 3hrs of training time for a single run
- Best Model Created: F1 = 0.8212

Combined: Greater than 50 hrs of training across all populations

Experimental Evaluation (Model Performance)

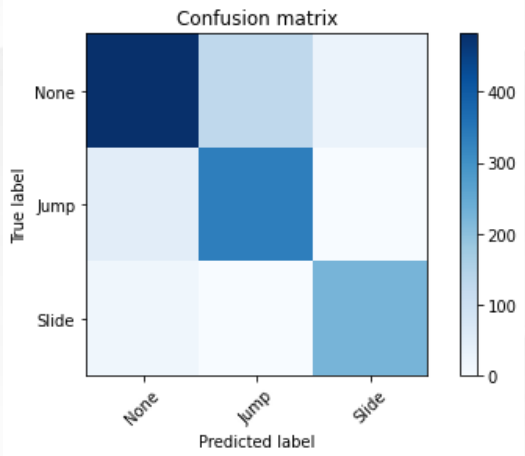


Best Model: Transfer Learning

- **F1 Score:** 0.82667
- **Accuracy:** None = 89% Jump = 74% Slide = 86%
- **Makeup:** VGG16 base, TanH Activation, SGD Optimizer, 7 trainable layers, 7 fully-connected layers (excluding output)

Almost identical!

The best transfer learning models had a high number of unfrozen layers. This means the best networks were produced by retraining the entire network meaning transfer learning was not ideal for our task.



Best Model: Custom CNN

- **F1 Score:** 0.8212
- **Accuracy:** None = 88% Jump = 72% Slide = 88%
- **Makeup:** CNN Type 1 (3 CNN layers), Relu Activation, SGD Optimizer, 3 fully connected layers (excluding output)

Experimental Evaluation (Game Performance)

Chromosome	F1 Score	Game Score	Coins Collected
[6, 3, 5, 1, 1]	0.8212	993	52
[3, 3, 5, 1, 1]	0.7986	993	51
[3, 3, 2, 2, 5]	0.6284	17	0
[5, 1, 3, 2, 4]	0.5496	17	0
[1, 6, 4, 3, 3]	0.4318	17	0

Table: Top custom CNN models in each F1 bracket

Chromosome	F1 Score	Game Score	Coins Collected
[7, 1, 7, 1, 5, 7]	0.8267	993	50
[1, 3, 3, 1, 5, 1]	0.7992	993	49
[2, 0, 3, 0, 2, 4]	0.7294	17	16

Table: Top transfer learning models in each F1 bracket. Including the worse model saved.

Level 1 - Trained on this Level

- Only models in the upper 0.7 and 0.8 F1 brackets completed the game.
- Models scored on game score (score of 993 means game completion) and coins collected.

Results

- Both strategies produced viable models. However the custom CNN models proved a ever so slightly more motivated to collect coins.

Chromosome	F1 Score	Game Score	Coins Collected
[3, 1, 3, 1, 4]	0.8189	781	44
[5, 0, 2, 1, 5, 5]	0.8197	627	35

Table: Top custom and transfer learned models on the untrained level 2.

Level 2 - No Training on this Level

Important to note, the goal of our project was not to create a generalized CNN capable of completing any level. Though we tested this as we were curious about what would happen.

- All models failed to complete the level. Falling prey to new obstacle combinations.
- Top models on level 2 were not the same top models on level 1.
- Models with a higher F1 score failed to generalize on data they had never seen. Models with a slightly lower score seemed more adept at this task.

Related Work

Game-Independent AI Agents for Playing Atari 2600 Console Games

Yavar Naddaf. 2010. Game-Independent AI Agents for Playing Atari 2600 Console Games. Master's thesis. University of Alberta, Edmonton, Alberta.

Generating War Game Strategies using a Genetic Algorithm

T. E. Revello and R. McCartney, "Generating war game strategies using a genetic algorithm," Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600), 2002, pp. 1086-1091 vol.2, doi: 10.1109/CEC.2002.1004394

Conclusion

Transfer learning showed little improvement over our custom CNN models. This was due to the uniqueness of our dataset resulting in the lack of many suitable pre-trained models for transfer learning

Both strategies produced models that completed the game. However, only models in the upper 0.7 and 0.8 F1 brackets showed this ability. Lastly, our custom CNN models showed a bit more motivation to collect coins.

When unleashed on a new level the previous best models showed shortcomings. Though our slightly lower scoring models showed some ability to identify and respond to unknown situations by progressing further into the new level.

Overall we accomplished our goal. We successfully created computer vision models that emulated human behavior. The models created demonstrated that they could successfully complete a complex task that they were trained for while showing the ability to perform a low level of generalization on unknown tasks.

Work Division & Learning Experiences

Task division:

- Game development (Both)
- Data collection (Both)
- Data preprocessing (Quinn)
- Transfer Learning (Logan)
- Custom CNN (Quinn)
- Report (Both)
- Presentation (Both)

Learning Experiences:

- Game development
- DEAP Python Library
- Data set consistency
- The AI development cycle
- How to write an ACM report

Demo

Try Panda Runner Yourself (case sensitive): https://bit.ly/3vDonqi