令和５年度卒業研究報告書

ジェスチャーで絵を描くシステムの作成

情報技術科　内田陽大　古川純至　吉田朋弥

指導教員　　菅野　研一

目次

第1章　はじめに

本研究は産技短展や学園祭に訪れた子供が情報の分野に興味を持ってもらうことを目的としこのシステムを作成した.python のライブラリである Mediapipe を使用したハンドトラッキングでイラストを描く．描いたイラストを学習させた AI に送り,自動的にステータスを生成する,そのステータスをもとに描いたイラストが戦闘するといった流れになっている.

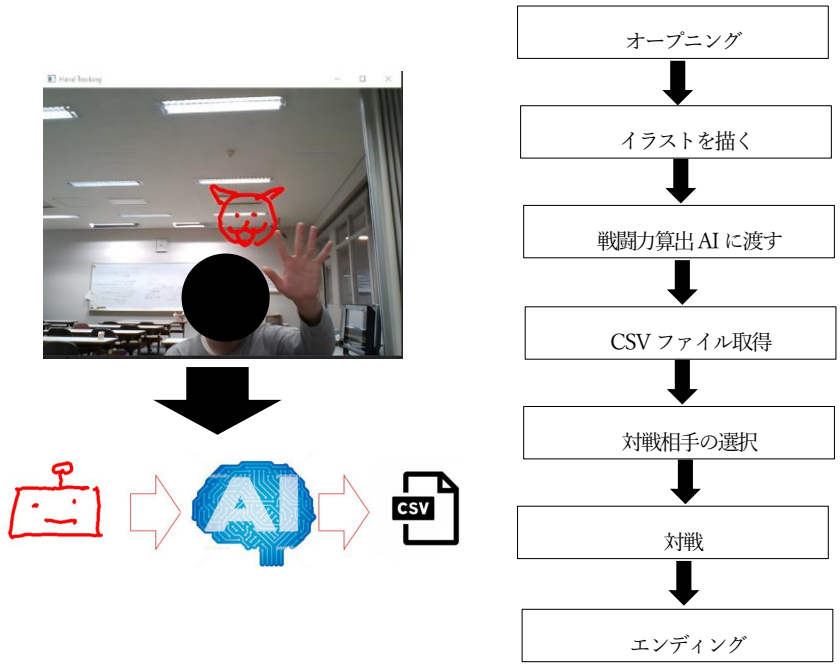

図 1.1.1 全体の流れ

第 2 章　研究概要

2.1 概要

この研究は大きく分けて手の動きを読み取って画像にするものと,画像からステータスを予測するもの,Unity で使いやすくゲーム化したものに分けられる.手の動きを読み取り画像にするもの, および Unity で使いやすくゲーム化したものは第 4 章で, Unity を使ったゲーム制作に,画像からステータス予測するものは第 3 章で述べる.

2.2 実行環境

システムの実行環境を以下の表に示す.

表 2.1　実行環境

OS	Windows10
言語	Python　C#
ライブラリ	mediapipe
	NumPy
カメラ	Web カメラ
開発環境	Unity

2.3 使用技術の紹介

2.3.1 Mediapipe

コンピュータビジョンや機械学習を活用してリアルタイムのメディア処理を行うためのツールキット,主にビデオやカメラからのデータを処理し,様々な画像や動画などの視覚データのタスクを実行するのに使用される.

2.3.2 C#

C#は Microsoft によって開発されたプログラミング言語で,2000 年に初めて導入されました.オブジェクト指向やイベント駆動型プログラミングをサポート,NET フレームワーク上で動作します.Windows アプリケーション,ウェブアプリケーション,モバイルアプリケーションなど,さまざまなプラットフォームで使用されており,豊富な標準ライブラリや開発ツールが提供されています.簡潔で直感的な構文を持ち,開発者が生産的にコードを書けるように機能が組み込まれています.

2.3.3 Unity

Unity は,ゲーム開発やシミュレーション,仮想現実,拡張現実などの 3D および 2D アプリケーションを開発するためのクロスプラットフォームの統合開発環境であり,幅広い機能やリソース,コミュニティサポートを提供されているソフトウェア.

2.3.4 Python

Python は.シンプルで読みやすい構文が特徴で、多岐にわたるライブラリやフレームワークに支えられており,データサイエンス,機械学習,ウェブ開発など広い分野で利用されているプログラミング言語である.

2.3.5 OpenCV

OpenCV は. Open Source Computer Vision の略称で,オープンソースのコンピュータビジョンライブラリであり,画像処理やコンピュータビジョンのタスクをサポートするために設計されている.画像や動画から特徴抽出、物体検出、顔認識、画像変換などの幅広いコンピュータビジョンの機能を提供し,研究,開発,プロトタイピングに広く利用されている.

.

2.4 基本機能

1. お絵描き機能

Python で Mediapipe を用いたハンドトラッキングをするプログラムをもとに作成し,カメラから手の座標を取得した後,人差し指の先端の座標を取得し,その座標に線を置く仕組みでイラストを描くことができる.

2. 戦わせる機能

絵を描く機能で準備した絵を AI に渡して,CSV ファイルに描いた画像の名前と,体力,攻撃力,防御力,クリティカル,速さの 5 つの値を書き込み,その値を基に Unity 内のシーンで戦闘時の計算を行う.

第 3 章　ステータス予測 AI

3.1 ステータス予測

画像を入力とし CNN(畳み込みニューラルネットワーク)でステータスを予測し,出力する AI を作った.入力画像は 300px,300px の png 形式で入力され同時に体力,攻撃力,防御力,クリティカル率,素早さの 5 個のステータスを予測している.

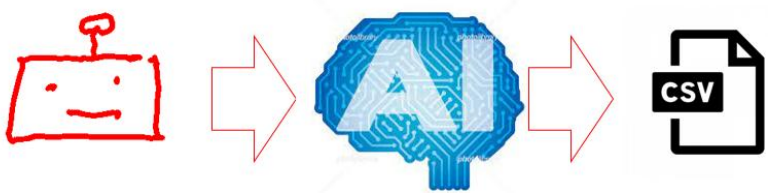

図 3.1 全体の流れ

3.2 CNN:畳み込みニューラルネットワーク

CNN とは,いくつもの深い層を持ったニューラルネットワークであり,主に画像認識の分野において価値を生んでいるネットワークである. この CNN は,「畳み込み層」や「プーリング層」といったいくつかの個性的な機能を備えた層を積み上げることで構成されているのが特徴である.

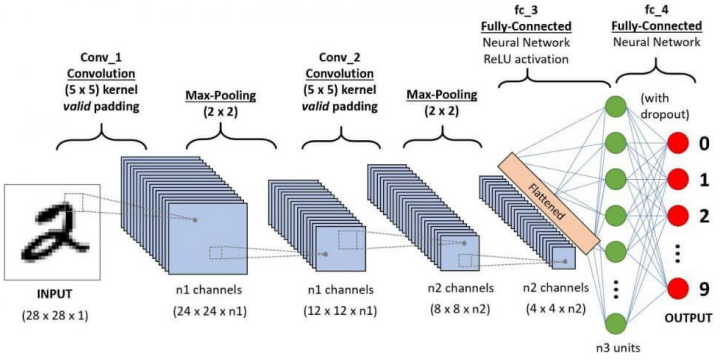

図 3.2 CNN

3.2.1 Convolution:畳み込み

一般的なニューラルネットワークの場合,層状にニューロンを配置して,その前後の層に含まれたニューロン同士に関しては網羅的に結線していく.しかし,CNN では,ニューロン同士の結合をうまく制限するとともに,ウェイト共有という手法によって画像の畳み込みに近い処理を,ニューラルネットワークという枠組みの中で表現している.

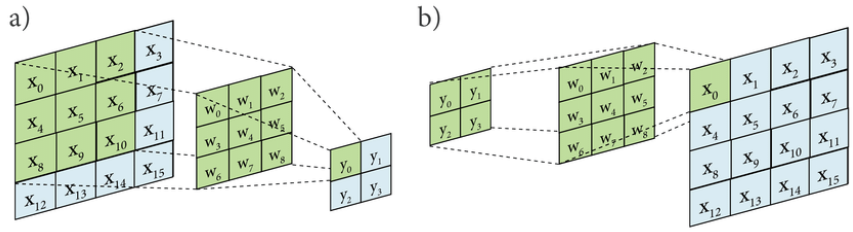

図 3.3 畳み込み

3.2.2 Pooling:プーリング

プーリング層は畳み込み層で抽出された特徴を平行移動などが起きても影響を受けることがないように外乱の影響によって変化することを阻止する性質(ロバスト性)を与えるという役割を担っている.

CNN は,主に画像をカテゴリごとに分類する作業（一般物体認識）において価値を発揮するネットワークとして知られている.ただ,写真に写った動物が犬なのか猫なのかを分類したい場合には,犬と猫が正しく識別できる能力を持つ必要がある.そのため,写真の犬が左端に映っているか,右端に映っているか,といった情報は特に重要ではない.つまり,画像のカテゴリを分けるタスクにおいて,あまり重要ではない位置に関する情報を上手に削ぎ落としているのが,このプーリング層というものである.

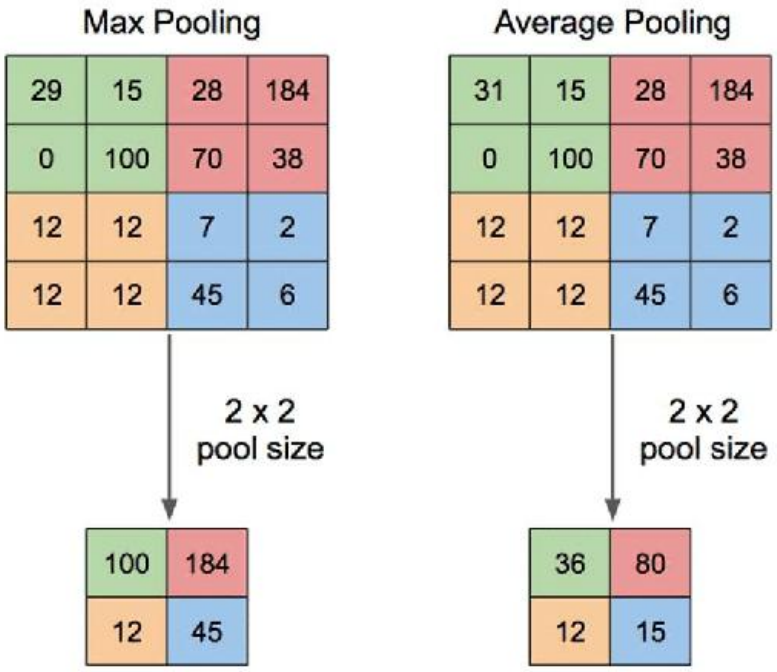

図 3.4 プーリング

3.3 作成したプログラム

以下に作成したプログラムを示す.プログラムは python を使用して組んでいる.

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import glob
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
from sklearn.preprocessing import MinMaxScaler

# データ拡張の設定
data_gen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

def plot_history(history):
    # 損失の推移をプロット
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    # 精度の推移をプロット
    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
```

```
    plt.legend()

    plt.show()
#フォルダの中の画像をすべて読み込んで numpy.ndarra 形式にする

folder_path = './pokemon'#フォルダを指定する
image_files = glob.glob(folder_path + '/*.png')

images = [] #  画像を NumPy 配列に読み込む
for filename in image_files:
    img = Image.open(filename)
    img_array = np.array(img)
    images.append(img_array)
images = np.array(images)
print(type(images))
#CSV ファイル
file_path = './pokemon.csv'   # CSV ファイルのパス
data = np.genfromtxt(file_path, delimiter=',')
#  正規化のためのスケーラーの初期化
scaler = MinMaxScaler()

#  データをフィッティングし,変換
normalized_data = scaler.fit_transform(data)

#トレーニング用とテスト用と分ける
#画像
split_index = int(images.shape[0] * 0.7);#7:3 で分ける
train_images = images[:split_index]
test_images = images[split_index:]
#SCV
split_index = int(normalized_data.shape[0] * 0.7);#7:3 で分ける
```

```
train_labels = normalized_data[:split_index]
test_labels = normalized_data[split_index:]

# データの読み込み（ここではサンプルのデータを使用）
#(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
print(type(train_images))
print(type(train_labels))
# データの前処理
train_images, test_images = train_images / 255.0, test_images / 255.0

# モデルの構築
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(300, 300, 4)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
# 最後の畳み込み層を削除
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(5))

# モデルのコンパイル

model.compile(optimizer='adam',
              loss='mean_absolute_error',      # 回帰の場合は平均二乗誤差を使用
              metrics=['mean_absolute_error', 'accuracy'])  # 平均絶対誤差 (MAE) をモデルの評価指標に使用

# モデルの概要表示
model.summary()
```

```
# モデルの訓練
history = model.fit(data_gen.flow(train_images, train_labels, batch_size=32), epochs=100, validation_data=(test_images, test_labels))

# 評価
model.save('pokemonModel')
plot_history(history)
```

3.3.1 プログラムの解説

以下のコードは各種ライブラリやモジュールをインポートしている.

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import glob
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.preprocessing import MinMaxScaler
```

表 3.1 モジュール一覧

tensorflow	AI 関連のモジュール
glob	ファイルのパスを取得するためのモジュール
numpy	数値計算のためのライブラリ
PIL	画像を処理するためのライブラリ
sklearn.preprocessing	データの前処理のためのスケーリングモジュール

```
data_gen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

このコードはデータの拡張を行うための設定を行っている.具体的には画像をランダムに回転させたり拡大・縮小を行ったり,水平方向や垂直方向にランダムに移動させてはみ出た分を反対側に戻したりする.これによりモデルの汎化性能を向上させ,少ない画像数でも過学習などのリスクを軽減することができる.

```
def plot_history(history):
    # 損失の推移をプロット
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    # 精度の推移をプロット
    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
```

```
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

これは学習の結果を表示するグラフの設定を行っている関数である.以下に示すように表示される左が学習を行っていく中でどのくらいミスをしているかを表したもので右がどのくらい正解を出したかを表している.

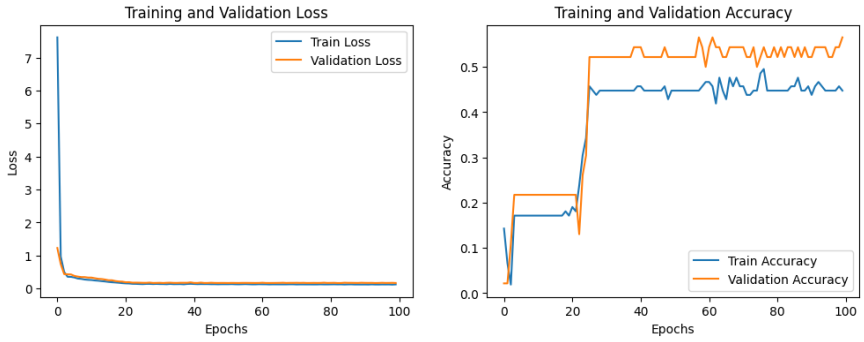

図 3.5　学習率

```
folder_path = './pokemon'#フォルダを指定する
image_files = glob.glob(folder_path + '/*.png')

images = [] # 画像を NumPy 配列に読み込む
for filename in image_files:
    img = Image.open(filename)
    img_array = np.array(img)
    images.append(img_array)
images = np.array(images)
print(type(images))
```

これは画像が入っているパスを指定してそこから画像を読み込んでいるるプログラムである. folder_path に相対パスで画像が入っているフォルダまでパスを書いている.その後の glob.glob(folder_path + '/*.png')で folder_path にある.png 画像すべてを正規表現で指定している.下の for 文は images = []に画像を NumPy 配列で格納するためのものである. Tensorflow では NumPy 配列でデータを扱うのが基本なので注意が必要である.

```
#CSV ファイル
file_path = './pokemon.csv'  # CSV ファイルのパス
data = np.genfromtxt(file_path, delimiter=',')
# 正規化のためのスケーラーの初期化
scaler = MinMaxScaler()
# データをフィッティングし,変換
normalized_data = scaler.fit_transform(data)
```

これは学習を行うための CSV ファイルからデータを取得し,正規化している部分である. file_path に CSV ファイルまでのパスを書いて np.genfromtxt(file_path, delimiter=',')でデータを取得している. delimiter=','と書いているのは CSV ファイルが[1,2,3]のように[,]区切りでデータを保存しているため[,]を区切り文字として指定して[1,2,3]を配列にして取得している. normalized_data =

scaler.fit_transform(data)は,CSV ファイルから取得したデータを 0~1 にスケールを変換するものとなっている.これを行っているか行っていないかで学習率などの影響はかなり大きい

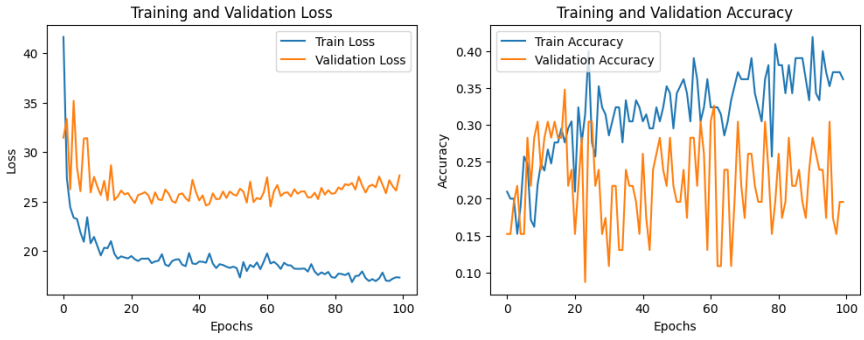

図 3.6 改善前

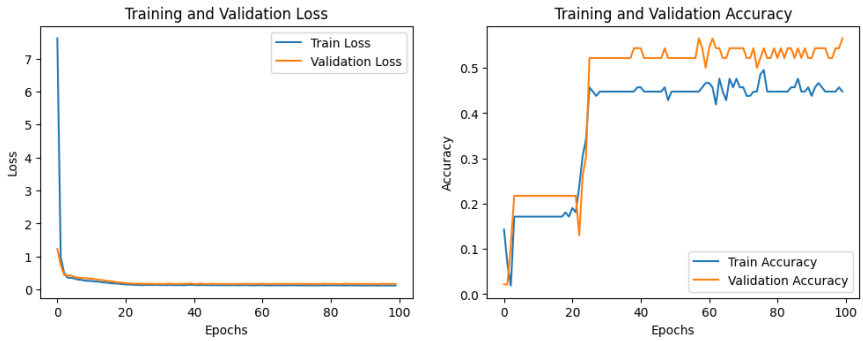

図 3.7 改善後

```
#画像
split_index = int(images.shape[0] * 0.7);#7:3 で分ける
train_images = images[:split_index]
test_images = images[split_index:]
#SCV
split_index = int(normalized_data.shape[0] * 0.7);#7:3 で分ける
train_labels = normalized_data[:split_index]
test_labels = normalized_data[split_index:]
```

上のプログラムは学習データを学習させるためのデータと,学習後に学習されていないデータで検証するためデータとわけるためのもので,学習データ 7:検証データ 3 でわけている.

```
train_images, test_images = train_images / 255.0, test_images / 255.0
```

上のプログラムは画像の RGBA 値(0~255)までのデータを AI での最適化アルゴリズムが安定して動作しやすくするために正規化(0~1)するためのコードである.これを行うことで学習の成果が高くなりやすく上の

図 3.6 ~~図 3.6~~ と図 3.7 ~~図 3.7~~ のように成果が高くなりやすいので学習させるために正規化するのはとても重要になる.

```
# モデルの構築
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(300, 300, 4)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
# 最後の畳み込み層を削除
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(5))
```

これは学習を行うためのモデルを構築するためのコードで Keras を使用して畳み込みニューラルネッ

トワーク（CNN）モデルを構築している.Keras は深層学習モデルを構築・トレーニングするための高レベルのニューラルネットワークライブラリで,ディープラーニングの研究や実用的なアプリケーション開発において,使いやすさと柔軟性を両立させた強力なツールとして広く利用されている. 以下に各層の説明を示す.

（1） 入力層:

```
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(300, 300, 4)))
```

・入力の形状は(300, 300, 4)で 300px：300px の画像で RGBA の色データを持ったもの

・32 個の 3x3 のフィルタを持つ畳み込み層

・性化関数は ReLU（Rectified Linear Unit）

（2） プーリング層:

```
model.add(layers.MaxPooling2D((2, 2)))
```

・2x2 の最大プーリング層

・特徴マップのサイズを縮小する

（3） 畳み込み層:

```
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

・64 個の 3x3 のフィルタを持つ畳み込み層

・活性化関数は ReLU

（4） プーリング層:

```
model.add(layers.MaxPooling2D((2, 2)))
```

・再び 2x2 の最大プーリング層です.

（5） フラット化層:

```
model.add(layers.Flatten())
```

・フィーチャマップを１次元に平坦化する.

・2 次元データを１次元データに変更し,ニューラルネットワークにつなげられるようにする.

（6） 全結合層（Dense 層）：

```
model.add(layers.Dense(64, activation='relu'))
```

・64 ユニットを持つ全結合層

・活性化関数は ReLU

（7） 出力層:

```
model.add(layers.Dense(5))
```

・5 ユニットを持つ全結合層

```
model.compile(optimizer='adam',
              loss='mean_absolute_error',
              metrics=['mean_absolute_error', 'accuracy'])
```

このコードの model.compile は Keras モデルのコンパイルを行うメソッドである.

・Optimizer（最適化アルゴリズム）：

optimizer='adam’ では,Adam 最適化アルゴリズムを使用している. Adam は勾配降下法の一種で,学習率の調整が自動で行われ,一般的に効果的な最適化が期待されるアルゴリズムである.

・Loss Function（損失関数）：

loss='mean_absolute_error'では,平均絶対誤差（MAE）を損失関数として使用している. MAE は回帰問題でよく使用され,予測値と実際の値の絶対値の平均を計算する. モデルが学習中にこの損失を最小化しようとする.

・Metrics（評価指標）：

metrics=['mean_absolute_error', 'accuracy'] では,モデルの評価指標として平均絶対誤差（MAE）と精

度（accuracy）を使用している.MAE はモデルの予測が実際の値からどれくらい離れているかを示す指標で,モデルの性能を評価するのに使う. 精度は主に分類問題で使用される指標で,正確に予測されたサンプルの割合を示す.

```
model.summary()

history = model.fit(data_gen.flow(train_images, train_labels, batch_size=32), epochs=100, validation_data=(test_images, test_labels))
model.save('pokemonModel')
```

model.summary()はモデルの概要を表示するためのメソッドで,

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 298, 298, 32)	1184
max_pooling2d_2 2D)	(MaxPooling (None, 149, 149, 32)	0
conv2d_3 (Conv2D)	(None, 147, 147, 64)	18496
max_pooling2d_3 2D)	(MaxPooling (None, 73, 73, 64)	0
flatten_1 (Flatten)	(None, 341056)	0
dense_2 (Dense)	(None, 64)	21827648
dense_3 (Dense)	(None, 5)	325

図 3.8 モデル

図 3.8 ~~図 3.8~~のように出力される.これを見て考えていたモデルと同じものができているか確認できる.

model.fit()はデルをトレーニングするためのメソッドである. 引数として,トレーニングデータとラベル（train_images と train_labels）,バッチサイズ（batch_size）,エポック数（epochs）,検証データ（validation_data）などが指定されている.

data_gen.flow(train_images, train_labels, batch_size=32) は,データジェネレータを通じてトレーニン

グデータを供給している．これにより,データの拡張やバッチ毎のデータの取得が行える．

validation_data=(test_images, test_labels)は,モデルのトレーニング中に検証データを用いてモデルの性能を評価するための設定である.トレーニングデータとは別に指定された検証データでの性能が表示される．

3.4 活性化関数

3.4.1 活性化関数

活性化関数とは,ニューラルネットワーク中の１つのニューロンにおいて,複数ノードの和を入力として,その出力を最終決定する関数である．前の通り「小さな入力値を活性化させて,大きな出力を得ること」が.活性化関数の主目的である．

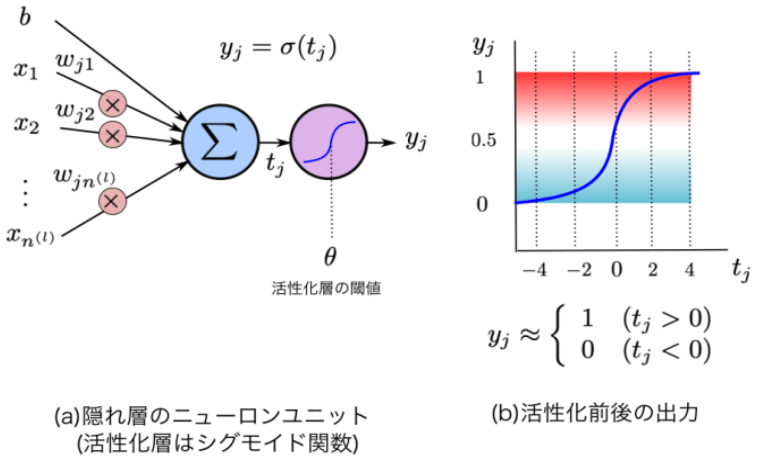

図 3.9 活性化関数

3.4.2 ReLU の基本型

ReLU(Rectified Linear Unit)は,コンピュータビジョンでの認識タスクで,CNN の中間層によく用いられる活性化関数である. 正の範囲はそのまま出力するのが特徴である.逆に,負の範囲は全ての入力値を出力値 0 にして押しつぶす. ReLU 型の活性化関数が CNN で定番化した第一の理由は,DNN の学習の収束時間が早くなることにある.収束が早くなる理由は図 3.10 ~~図 3.10~~-右のように,「正の範囲で微分値が全て 1」となるよう ReLU 型が設計されたことにある.これにより,シグモイドや tanh のように「勾配が 0 に飽和して学習が進行しなくなる」こともなく,勾配消失も生じない.つまり,深い DNN でもしっかりと大きな量の勾配を,序盤の層まで逆伝搬できるようになった.

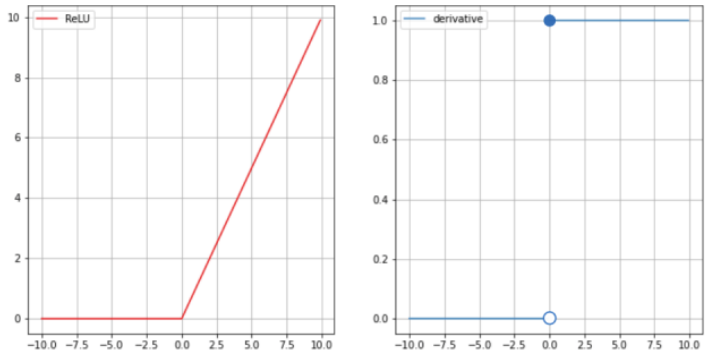

図 3.10 ReLU

3.5 モデル改善実験

AI を学習させるうえで設計図の役割をするモデルは一番学習精度に影響が出る部分だと考え,算出するデータが正しいうえにできるだけ計算量が少ないものを探し出すために実験を行った以下にそれぞれのプログラムとグラフを示す.また,学習を行うための epochs や損失関数などはいじっていない.

(1) 畳み込み層：多 全結合層：多

```
# 畳み込み層とプーリング層の追加
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(300, 300, 4)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(256, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(512, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(1024, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
# Flatten 層
model.add(layers.Flatten())

# 全結合層の追加
model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(5)) # 出力層
```

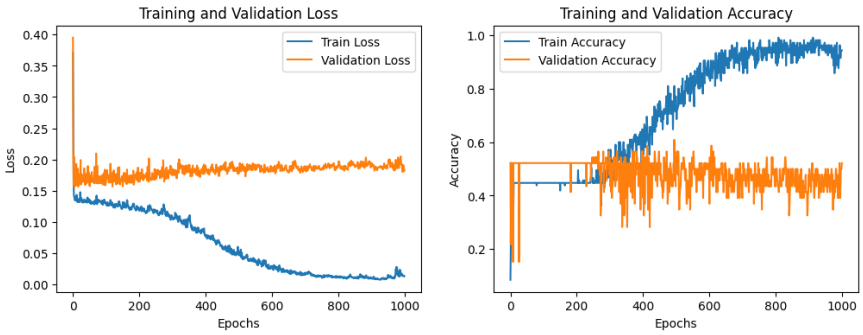

図 3.11 (1)　　畳み込み層：多 全結合層：多

(2)　畳み込み層：多 全結合層：少

```
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(300, 300, 4)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(256, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(512, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(1024, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(5)) # 出力層
```

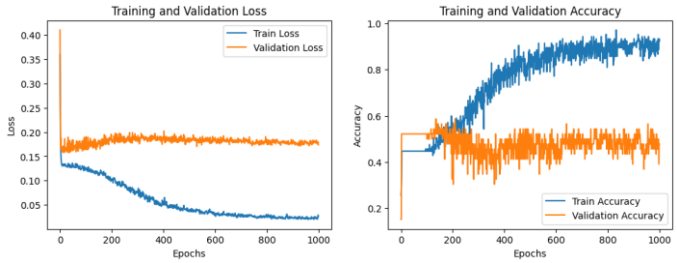

図 3.12（2）　畳み込み層：多 全結合層：少

（3） 畳み込み層：少 全結合層：多

```
model.add(layers.Conv2D(32, (3, 3), activation='relu',input_shape=(300, 300, 4)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(5)) # 出力層
```

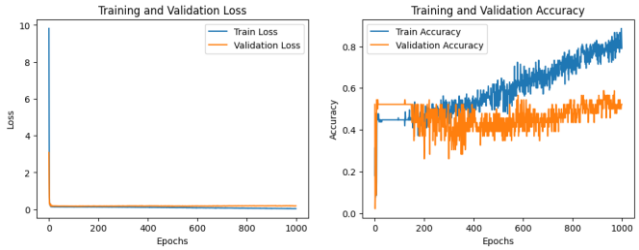

図 3.13（3）　畳み込み層：少 全結合層：多

（4） 畳み込み層：少 全結合層：少

```
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(300, 300, 4)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(5)) # 出力層
```

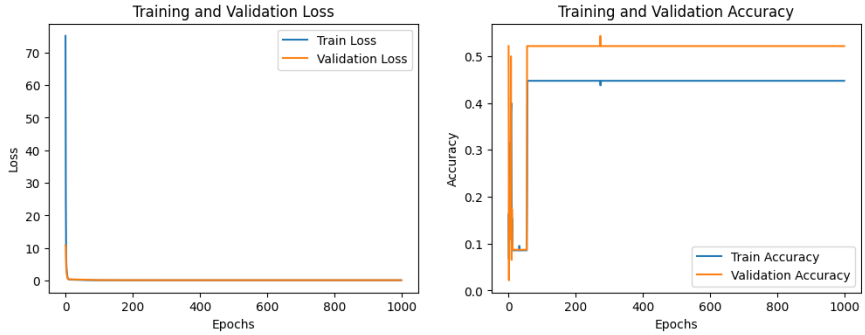

図 3.14 畳み込み層：少 全結合層：多

畳み込み層と全結合層を変えて行った実験結果である.これを見ると,畳み込み層が多いと最初から損失が少なく畳み込み層が少ないと最初の損失が大きくその後収束していく.全結合層の場合は量による変化は見られなかった.これらから畳み込み層や全結合層が多くなっても予測精度には影響が少ないと考え畳み込み層も全結合層どちらも少ないモデルを採用し,計算負荷を下げられるものを採用した.ただし,今回は同時に 5 個のステータスを予測しているため,通常とは異なる状況での実験のため,もし一つだけを予測するとなると今回とは異なる結果になる可能性はある.また,今回のグラフを見ると一見過学習を起こしているように見えるが,下の画像の最初の部分のように一度上がって下がるように繰り返す可能性はあるため,もっと試行回数を増やすことも解決策になる可能性がある.

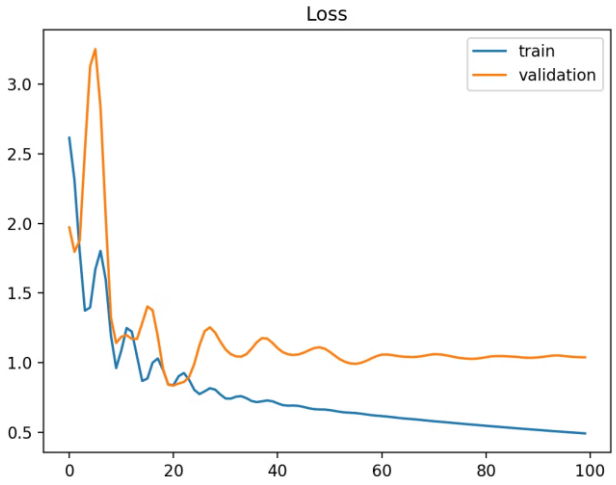

図 3.15 学習曲線

3.6 開発を行ってみての課題

今回 AI 開発専用 PC を使って開発を行ったが色々な障害があった,AI 関連の知識がないのはもちろんだが色々なもののバージョンが固定化されており苦労した.具体的に言いうと GPU ドライバーのバージョン,Python もバージョン,Docker のバージョン.これらが固定化されておりネットに落ちているものを試そうとしたときに大きな障害となった.試しに Python のバージョンを上げようとするために一度アンインストールしてインストールしてみたが,GUI がすべて飛んでしまい起動したらコンソールが表示され大変なことになった.もしその状態になったらホームディレクトリもファイルと Docker イメージなどを USB に保存し,復旧してほしいイメージは TensorFlow-pulus というものを Docker save とかで保存してほしい.もしやる気があるもしくはこれらでつまずいて何とかしたいなどと考えていたら,先生と相談のうえでピュアな UbuntuOS の導入を考えてほしい.ピュアな UbuntuOS ならバージョン固定などもないので開発がやりやすくなるはずなので考えておいてほしい.

コメントの追加 [KK1]: 赤の波線はスペルミスです。プログラムコードや固有名詞は別にして、本文中の波線は一通りチェックしてください。

コメントの追加 [y2R1]:

第 4 章　Unity を使ったゲーム制作

Python で mediapipe を使ったカメラでハンドトラッキングを行い画面上に線を描画するプログラムを Unity でゲームの一部にし,描いたイラストを自分のキャラクターとして遊ぶことでより楽しめる要素が増えると考えた.

今回制作したゲームは,タイトルシーン,イラスト作成シーン,名前を付けるシーン,対戦相手を選択するシーン,戦闘シーン,リザルトシーンを組み合わせたもので,展示で遊んでもらうことを考え,1 プレイの時間が約 5~10 分と手軽に遊べる時間になっており,短い時間で遊ぶことができるところと,パソコンと web カメラがあれば遊べる二つの手軽さを考慮して作成することにした.

ゲームの仕様

作成したゲームはタイトルシーン,イラスト作成シーン,名前を付けるシーン,対戦相手を選択するシーン,戦闘シーン,リザルトシーンの 6 つで構成されている.

イラスト作成シーンで,イラスト描く Python ファイルである「handread.py」を実行している.このシーンで描いたイラストと,次の名前を付けるシーンで入力された名前を,初代ポケモン 151 匹の見た目と強さを AI に学習させ,イラストを送ると強さを返すプログラムの「pokemon.py」を実行し csv ファイルで返してもらい,その値をそのイラストの能力とし,2 つ先のシーンである戦闘画面で利用している.対戦相手選択画面では,これまでに書いたイラストのステータスが保存されている csv ファイルを使って今までに書かれたイラストを呼び出して対戦相手を選択することができるようになっている.

4.1 タイトル画面

タイトル画面の「start」ボタンを押すと

PythonRunner.RunFile("./Assets/source_code/handRead.py")で handRead.py が起動され,イラスト作成シーンに移動するようになっている.

図 4.1 タイトル画面

ソースコード

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEditor.Scripting.Python;
public class pystart : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
PythonRunner.RunFile("./Assets/source_code/handRead.py");
    }
}
```

4.2 絵を描くシーン

コメントの追加 [KK3]: 節のタイトルの後いきなり写真になっていますが、絵の描き方を入れた方が良いです。展示のときもいちいち説明していましたね。説明が必要ということは書く必要があるということです。

カメラに手の平を向けると手が検出され,人差し指のみを伸ばすとそれに沿って線が描写されるようになる.人差し指の先端のｙ座標が中指の先端のｙ座標より下になると線の描写が止まるため,離れた線を描くことができるようになる.F キーを押すと直前に描いた線を削除し,D キーを押すと描いたすべての線が削除される.

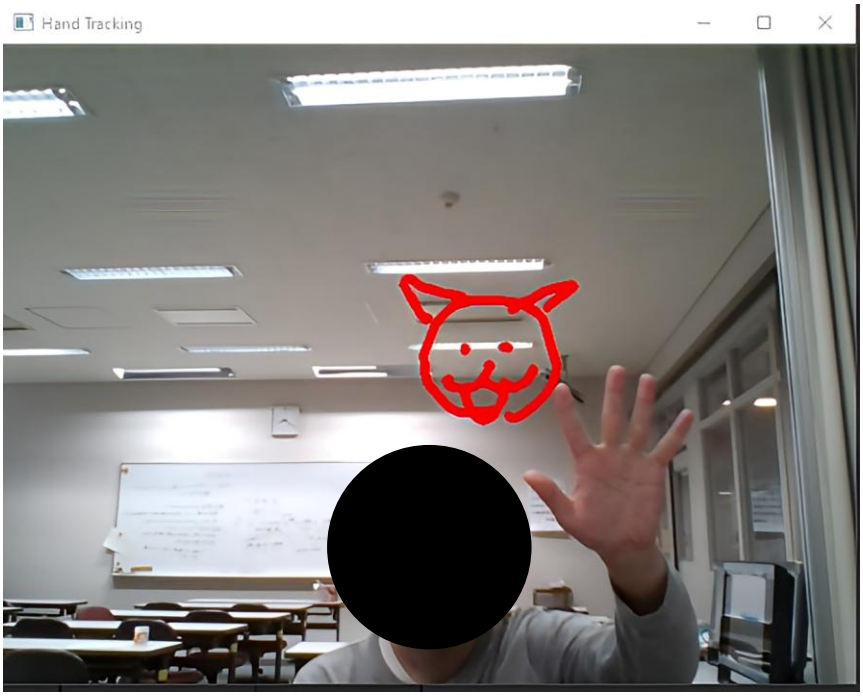

図 4.2 手で絵を描く

ソースコード

```
import cv2
import mediapipe as mp
```

```
import numpy as np
import math

mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands

cap = cv2.VideoCapture(0)

image_width, image_height = 640, 480

with mp_hands.Hands(min_detection_confidence=0.5, min_tracking_confidence=0.5,
max_num_hands=1) as hands:
    hand_landmarks_history = []
    all_hand_landmarks_history = []

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            continue



        frame = cv2.flip(frame, 1)
        frame = cv2.resize(frame, (image_width, image_height))



        key = cv2.waitKey(1) & 0xFF



        if key == ord('D') or key == ord('d'):
            all_hand_landmarks_history = []
            hand_landmarks_history = []
```

```
        if key == ord('F') or key == ord('f'):
            if len(hand_landmarks_history) > 0:
                hand_landmarks_history = []
            elif len(all_hand_landmarks_history) > 0:
                all_hand_landmarks_history.pop()


        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)


        results = hands.process(frame_rgb)

        if results.multi_hand_landmarks:
            for hand_landmarks in results.multi_hand_landmarks:

                index_finger=hand_landmarks.landmark
[mp_hands.HandLandmark.INDEX_FINGER_TIP]
                 middle_finger=hand_landmarks.landmark
[mp_hands.HandLandmark.MIDDLE_FINGER_TIP]
                wrist=hand_landmarks.landmark
[mp_hands.HandLandmark.WRIST]


                index_middle_distance = math.sqrt((index_finger.x - wrist.x) ** 2 +
(index_finger.y - wrist.y) ** 2)
                middle_wrist_distance = math.sqrt((middle_finger.x - wrist.x) ** 2 +
(middle_finger.y - wrist.y) ** 2)
                is_middle_finger_extended = index_middle_distance > middle_wrist_distance
```

```
                    if is_middle_finger_extended:

                        x, y = int(index_finger.x * image_width), int(index_finger.y * image_height)


                        hand_landmarks_history.append((x, y))

                    if not is_middle_finger_extended and hand_landmarks_history != []:

                        all_hand_landmarks_history.append(hand_landmarks_
history)hand_landmarks_history = []


            if len(hand_landmarks_history) > 1:
                for i in range(1, len(hand_landmarks_history)):
                    cv2.line(frame, hand_landmarks_history[i - 1], hand_landmarks_history[i], (0, 0,
255), 5)


            for landmarks_history in all_hand_landmarks_history:
                if len(landmarks_history) > 1:
                    for i in range(1, len(landmarks_history)):
                        cv2.line(frame, landmarks_history[i - 1], landmarks_history[i], (0, 0, 255), 5)

            lower_red = np.array([0, 0, 254])
            upper_red = np.array([0, 0, 255])

            mask = cv2.inRange(frame, lower_red, upper_red)

            result = cv2.bitwise_and(frame, frame, mask=mask)
```

```
            cv2.imshow('Hand Tracking', frame)
            cv2.imshow('Red Trajectory', result)
            mask = np.all(result == [0,0,0],axis=-1)
            dst = cv2.cvtColor(result, cv2.COLOR_BGR2BGRA)
            dst[mask,3] = 0

            # if key == 27:    esc
            #         break
            if key == 13: #enter
                  break

      cv2.imwrite('Assets/source_code/image/handreadResultImage.png', dst)

cap.release()
cv2.destroyAllWindows()
```

解説

```
cap = cv2.VideoCapture(0)
```

指定するカメラデバイスの ID を()内の引数で指定する.

カメラを 1 台接続している場合カメラデバイスの ID は 0 となるため上記の通り設定する.

```
  hand_landmarks_history = []
  all_hand_landmarks_history = []
```

描いた指の軌跡を保存するための配列

```
frame = cv2.flip(frame, 1)
frame = cv2.resize(frame, (image_width, image_height))
```

カメラの左右反転を直す cv2.flip()

第一引数には反転対象の画像やフレームを入れる(この場合は frame).

第二引数は反転の方向. 1 を指定した場合は水平方向に反転（左右反転）する.

画像やフレームのサイズを変更する cv2.resize

第一引数にはサイズ変更対象の画像やフレームがある（この場合は frame）.

第二引数には変更後のサイズを指定する.(image_width, image_height)は変更後の幅と高さ.

```
key = cv2.waitKey(1) & 0xFF
```

キーボードからの入力を受け付ける cv2.waitKey()

cv2.waitKey(1) は 1 ミリ秒だけキー入力を待機し,その結果が key に格納される.

```
if key == ord('D') or key == ord('d'):
        all_hand_landmarks_history = []
        hand_landmarks_history = []
```

D キーを押すと all_hand_landmarks_history = []と hand_landmarks_history = []を空にする.

```
    if key == ord('F') or key == ord('f'):
        if len(hand_landmarks_history) > 0:
            hand_landmarks_history = []
        elif len(all_hand_landmarks_history) > 0:
            all_hand_landmarks_history.pop()
```

F キーを押すと hand_landmarks_history = []を空にし all_hand_landmarks_history から pop し 1 画描いた軌跡を削除する.

```
results = hands.process(frame_rgb)
```

この行では,手の検出とトラッキングを行っている.hands は MediaPipe の手検出モデルを指す.process メソッドに RGB 形式のフレームを渡すことで,手の位置や姿勢を results に取得する.

```
index_finger = hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP]
middle_finger = hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_TIP]
wrist = hand_landmarks.landmark[mp_hands.HandLandmark.WRIST]
```

各手のランドマークを取得.

```
index_middle_distance = math.sqrt((index_finger.x - wrist.x) ** 2 + (index_finger.y - wrist.y) ** 2)
middle_wrist_distance = math.sqrt((middle_finger.x - wrist.x) ** 2 + (middle_finger.y - wrist.y) ** 2)
is_middle_finger_extended = index_middle_distance > middle_wrist_distance
```

中指が伸びているかどうかを判定.

判定基準は中指の先端と手首の距離

```
if is_middle_finger_extended:
    x, y = int(index_finger.x * image_width), int(index_finger.y * image_height)
    hand_landmarks_history.append((x, y))
```

中指が伸びでいる場合,その時点の中指の位置を hand_landmarks_history リストに追加.

```
if not is_middle_finger_extended and hand_landmarks_history != []:
    all_hand_landmarks_history.append(hand_landmarks_history)
    hand_landmarks_history = []
```

中指が伸びていない場合かつ hand_landmarks_history が空でない場合,手の移動履歴を保持するリスト all_hand_landmarks_history に hand_landmarks_history を追加し,hand_landmarks_history をリセットする.

4.3 名前入力画面

この Unity スクリプトは,テキストボックスに入力したテキストをファイル名として使用し,指定されたディレクトリにある画像を読み込んで PNG 形式で保存~~しするする~~.Enter キーが押されたときに実行され,保存が成功すると指定されたシーンに移動し,元の画像ファイルは保存後に削除され,保存された画像のファイルパスは CSV ファイルに記録される.

図 4.3 名前入力画面

ソースコード

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections.Generic;
using System.IO;
using UnityEngine.SceneManagement;

public class ButtonHandler : MonoBehaviour
{
    public Button RightButton;
    public Button LeftButton;
```

```
    public Button DecisionButton;
    private int CsvLength = 0;
    private List<string> CsvList = new List<string>();
    private int count = 1;
    public Image LeftImageComponent;
    public Image RightImageComponent;

    void ChangeSourceImage()
    {
        string imagePath1 = CsvList[count].Split(',')[0];
        string[] pathSegments1 = imagePath1.Split('/');
        string enemyName = pathSegments1[2] + '/' + pathSegments1[3].Replace(".png", "");
// .png などの拡張子を除く

        Sprite newSprite = Resources.Load<Sprite>(enemyName);

        if (newSprite != null)
        {
            RightImageComponent.sprite = newSprite;
            Debug.Log("Image changed successfully.");
        }
        else
        {
            Debug.LogError("Failed to load image: " + enemyName);
        }
        ChangeSourceImage(LeftImageComponent, CsvList[CsvLength - 1].Split(',')[0]);
    }

    void Start()
    {
        RightButton.onClick.AddListener(RightOnClick);
```

```
        LeftButton.onClick.AddListener(LeftOnClick);
        DecisionButton.onClick.AddListener(DecisionOnClick);
        ReadDataFromCSV();

        // 右側の画像
        ChangeSourceImage(RightImageComponent, CsvList[count].Split(',')[0]);

        // 左側の画像
        ChangeSourceImage(LeftImageComponent, CsvList[CsvLength - 1].Split(',')[0]);

        Debug.Log("Images set up successfully.");
    }

    void ChangeSourceImage(Image targetImage, string imagePath)
    {
        string[] pathSegments = imagePath.Split('/');
        string resourceName = pathSegments[2] + '/' + pathSegments[3].Replace(".png", "");
// .png などの拡張子を除く

        Sprite newSprite = Resources.Load<Sprite>(resourceName);

        if (newSprite != null)
        {
            targetImage.sprite = newSprite;
            Debug.Log("Image changed successfully for " + targetImage.name);
        }
        else
        {
            Debug.LogError("Failed to load image: " + resourceName);
        }
    }
```

```
void RightOnClick()
{
    ChangeSelect(1);
    ChangeSourceImage();
}

void LeftOnClick()
{
    ChangeSelect(-1);
    ChangeSourceImage();
}

void DecisionOnClick()
{
    Debug.Log("決定ボタンが押されました");

    // 画像データのファイルパスを PlayerPrefs に保存
    PlayerPrefs.SetString("SelectedImagePath", CsvList[count]);

    // バトルシーンに切り替え
    SceneManager.LoadScene("battleScene");
}

public void ReadDataFromCSV()
{
    string filePath = "Assets/source_code/status.csv";
    using (StreamReader sr = new StreamReader(filePath))
    {
        string line;
        while ((line = sr.ReadLine()) != null)
```

```
            {
                CsvList.Add(line);
                CsvLength++;
            }
        }
        Debug.Log("CSV file has been read.");
    }

    public void ChangeSelect(int move)
    {
        count += move;
        if (count == 0)
        {
            count = CsvLength - 2;
        }
        if (count == CsvLength - 1)
        {
            count = 1;
        }
        Debug.Log(CsvList[count].Split(',')[0]);
    }
}
```

以下のコードは、Unity の C#スクリプトで、CSVWriter クラスが CSV ファイルにデータを書き込むためのプログラムとなっている.

```
using System.IO;
using UnityEngine;
using UnityEditor.Scripting.Python;
public class CSVWriter
{
    // CSV ファイルのパス
    private string filePath = "Assets/source_code/status.csv";



    // CSV にデータを書き込むメソッド
    // 引数 data は CSV 形式の文字列（例："1,SampleName,100"）
    public static void WriteDataToCSV(string data)
    {
        string filePath = "Assets/source_code/status.csv";
        // StreamWriter を使用してファイルに書き込み
        // append: true で既存のファイルに追記,false で新規作成または上書き
        using (StreamWriter sw = new StreamWriter(filePath, true))
        {
            sw.WriteLine(data);
        }

        Debug.Log("Data written to CSV file.");

        PythonRunner.RunFile("./Assets/source_code/pokemon.py");
    }
}
```

ステータス計算用プログラム

以下のプログラムは手で描いた画像を読み込む機能と,描いた絵のステータスを予測する機能,予測したステータスと画像のファイルパスを書き込む機能に分けられる.画像を読み込む機能は CSV ファイルから最後の行を読み込んで画像までのファイルパスを取得し,画像を取得する.取得後 CSV ファイルの最後の行の削除を行う.ステータスを予測する機能は取得した画像を 300px：300px にリサイズして AI に画像を引数として渡し,ステータスを取得する. 予測したステータスと画像のファイルパスを書き込む機能は,AI が予測したステータスを CSV ファイルにファイルパスと一緒に書き込む機能となっている.

ソースコード

```
from tensorflow.keras.models import load_model
from PIL import Image
import numpy as np
import pandas as pd
import os

import UnityEditor
import UnityEngine
import csv

file_path="Assets/source_code/status.csv"

def get_latest_data(csv_file_path):
    try:

        df = pd.read_csv(csv_file_path)
```

```
            latest_data = df.iloc[-1]

            return latest_data
        except FileNotFoundError:
            UnityEngine.Debug.Log("CSV file not found at path: " + csv_file_path)
            return None
        except pd.errors.EmptyDataError:
            UnityEngine.Debug.Log("CSV file is empty: " + csv_file_path)
            return None
        except pd.errors.ParserError as e:
            UnityEngine.Debug.Log("Error parsing CSV file: " + str(e))
            return None



def delete_last_data(csv_file_path):
    try:
        with open(csv_file_path, 'r', encoding='utf-8') as file:
            lines = file.readlines()
        if lines:
            lines = lines[:-1]

        with open(csv_file_path, 'w', encoding='utf-8') as file:
            file.writelines(lines)

        UnityEngine.Debug.Log("Successfully deleted last data.")
    except FileNotFoundError:
        UnityEngine.Debug.Log("CSV file not found at path: " + csv_file_path)
    except Exception as e:
```

```
            UnityEngine.Debug.Log(f"Error: {e}")

latest_data = get_latest_data(file_path)
delete_last_data(file_path)

file_path = "Assets/source_code/status.csv"
image_path = latest_data['Image']

model = load_model('./pokemonModel')

image = Image.open(image_path).resize((300, 300))

image_array_rgb = np.array(image)
image_array_rgby = image_array_rgb[:, :, :3]  # RGB のみを抽出して 4 つのチャンネルにする
image_array_rgby = np.concatenate((image_array_rgby, np.zeros((300, 300, 1))), axis=-1)

image_array_rgby = image_array_rgby / 255.0

image_batch = np.expand_dims(image_array_rgby, axis=0)

predictions = model.predict(image_batch)

output_csv_path = './Assets/source_code/status.csv'

sum_status = predictions[0][0] + predictions[0][1] + predictions[0][2] + predictions[0][3] +
predictions[0][4]
normalization_status_number = 500 / sum_status

normalization_status = [
    int(round(predictions[0][0] * normalization_status_number)),
    int(round(predictions[0][1] * normalization_status_number)),
```

```
    int(round(predictions[0][2] * normalization_status_number)),
    int(round(predictions[0][3] * normalization_status_number)),
    int(round(predictions[0][4] * normalization_status_number))
]

data = pd.DataFrame({
    'Image': [image_path],
    'ClassH': [normalization_status[0]],
    'ClassA': [normalization_status[1]],
    'ClassB': [normalization_status[2]],
    'ClassC': [normalization_status[3]],
    'ClassS': [normalization_status[4]]
})
try:
    existing_data = pd.read_csv(output_csv_path)
    combined_data = pd.concat([existing_data, data], ignore_index=True)
    combined_data.to_csv(output_csv_path, index=False)

    UnityEngine.Debug.Log("CSVfile_OK")
except FileNotFoundError:
    data.to_csv(output_csv_path, index=False)
    UnityEngine.Debug.Log("CSVfile_create")

except pd.errors.EmptyDataError:
    UnityEngine.Debug.Log("CSVfile_not")

except pd.errors.ParserError as e:
    UnityEngine.Debug.Log(f"CSVfile_error: {e}")



text_asset = UnityEditor.AssetDatabase.LoadAssetAtPath(output_csv_path, UnityEngine.Object)
```

```
text_asset = UnityEditor.AssetDatabase.LoadAssetAtPath(output_csv_path, UnityEngine.Object)

if text_asset is not None:
    UnityEngine.Debug.Log("Successfully:", text_asset)
    with open(output_csv_path, 'r') as file:
        content = file.read()
        UnityEngine.Debug.Log("File Content:", content)
else:
    UnityEngine.Debug.Log("File not found at path: " + output_csv_path)
```

4.4 対戦相手選択画面

このシーンでは,画面に表示されている右矢印と左矢印の画像をクリックすると

Resources フォルダから読み込んだ画像を配列に格納し,ボタンが押されるたびに過去のイラストを保存しているフォルダの配列内の画像を切り替え,配列の最後または最初に達した場合,それぞれ最初または最後の画像にループして戻るようになっている.

図 4.4 対戦相手選択画面

ソースコード

```
using UnityEngine;
using UnityEngine.UI;

public class CharacterSelection : MonoBehaviour
{
    public Image characterImage;
    private Sprite[] characterSprites;
    private int currentIndex = 0;
```

```
    void Start()
    {
        // Resources フォルダから画像を読み込む
        LoadCharacterSprites();

        // 初期状態で最初の画像を表示
        //UpdateCharacterImage();
    }

    void Update() {
        if (currentIndex >= characterSprites.Length)
        {
            currentIndex = 0; // 配列の最後を超えたら最初に戻る
            Debug.Log("right_button");
        }
        if (currentIndex < 0)
        {
            currentIndex = characterSprites.Length - 1;

            Debug.Log("left_button");
        }
    }

    void LoadCharacterSprites()
    {
        // Resources フォルダから画像を読み込む
        Object[] loadedSprites = Resources.LoadAll
("source_code/image", typeof(Sprite));

        // 読み込んだ画像を配列に変換
```

```
        characterSprites = new Sprite[loadedSprites.Length];
        for (int i = 0; i < loadedSprites.Length; i++)
        {
            characterSprites[i] = (Sprite)loadedSprites[i];
        }
    }

    public void OnRightArrowButtonClicked()
    {
        // 右矢印ボタンが押されたときの処理
        currentIndex++;
        if (currentIndex >= characterSprites.Length)
        {
            currentIndex = 0; // 配列の最後を超えたら最初に戻る
            Debug.Log("right_button");
        }
        //UpdateCharacterImage();
    }

    public void OnLeftArrowButtonClicked()
    {
        // 左矢印ボタンが押されたときの処理
        currentIndex--;
        if (currentIndex < 0)
        {
            currentIndex = characterSprites.Length - 1;
// 0 より小さくなったら配列の最後に移動
            Debug.Log("left_button");
        }
    }
}
```

4.5 戦闘画面

この Unity スクリプトは,バトル画面における UI やゲーム進行を担当しており,スクリプトは,CSV ファイルから読み込んだキャラクターのステータス情報を使用し,読み込んだ敵とプレイヤーの初期ステータスを画面の右上,左下にそれぞれ表示する.バトル進行では,攻撃力にクリティカル値の確率を使った攻撃ダメージが計算され,お互いの HP が減少する.この際,結果に応じてプレイヤーが勝利した場合は resultScene(win)シーンに,負けた場合は resultScene(lose)シーンに移動する.戦闘時に用いられる HP は攻撃力が計算された後,ChangeSliderValue メソッドを用いて Slider の値が変更され,プレイヤーと敵の HP が視覚的に表現されるようになっている.

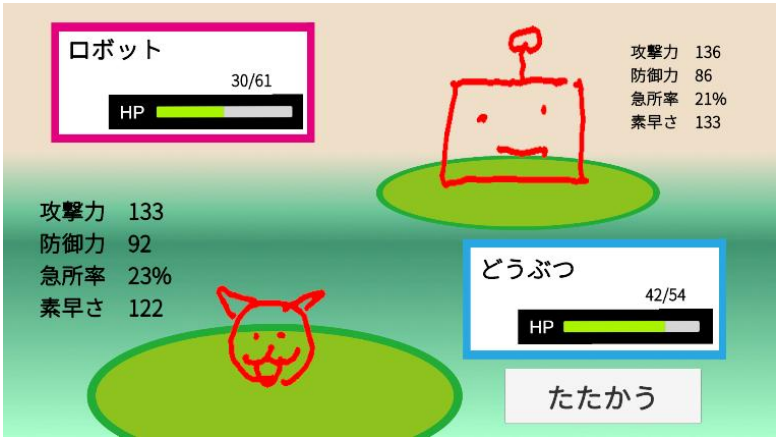

図 4.5 戦闘画面

ソースコード

```
using System.Diagnostics;
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System.IO;
using UnityEngine.SceneManagement;
using TMPro;
public class batleUI : MonoBehaviour
{
    private String EnemyCsvLine;
    private String selectedImagePath;
    public Slider Eslider;
 // Unity エディタでアタッチするための SliderController コンポーネント
    public Slider Pslider;
    public Button BattleButton;
    private int CsvLength = 0;
    public Image LeftImageComponent;
    public Image RightImageComponent;
    private List<string> CsvList = new List<string>();
    public struct Status
    {
        public int HP;
        public int Attack;
        public int Defense;
        public int Critical;
        public int Speed;
        public Status(int H, int A, int B, int C, int S)
```

```
        {
            HP = H;
            Attack = A;
            Defense = B;
            Critical = C;
            Speed = S;
        }
    }
    public Status EnemyStatus = new Status(1, 1, 1, 1, 1);
    public Status PlayerStatus = new Status(1, 1, 1, 1, 1);
    private int EnemyMaxHP;
    private int PlayerMaxHP;
    public TextMeshProUGUI EnemyName;
    public TextMeshProUGUI EnemyStatusGUI;
    public TextMeshProUGUI EnemyHP;
    public TextMeshProUGUI PlayerName;
    public TextMeshProUGUI PlayerStatusGUI;
    public TextMeshProUGUI PlayerHP;
    private String[] statusTemp = { "攻撃力　", "防御力　", "急所率　", "素早さ　" };

    void Start()
    {
        //CSV ファイル読み込み
        ReadDataFromCSV();
        //エネミーデータ受け取り
        EnemyCsvLine = PlayerPrefs.GetString("SelectedImagePath", "DefaultImagePath");
        // デフォルト値を指定
        selectedImagePath = EnemyCsvLine.Split(',')[0];
        // データを確認
        UnityEngine.Debug.Log("Selected Image Path in BattleScene: "
                              + selectedImagePath);
```

```
        //ステータスデータセット
        UnityEngine.Debug.Log("EnemyCsvLine" + EnemyCsvLine);
        setStatus(ref EnemyStatus, EnemyCsvLine);
        setStatus(ref PlayerStatus, CsvList[CsvLength - 1]);
        //ボタン設定
        BattleButton.onClick.AddListener(battleSceneButton);
        //画像設定
        ChangeSourceImage(LeftImageComponent, CsvList
                                [CsvLength- 1].Split(',')[0]);
        ChangeSourceImage(RightImageComponent, selectedImagePath);
        EnemyMaxHP = EnemyStatus.HP;
        PlayerMaxHP = PlayerStatus.HP;
        //GUI ステータスを変更
        changeText(selectedImagePath, CsvList
                        [CsvLength- 1].Split(',')[0]);
    }
    void ChangeSourceImage(Image targetImage, string imagePath)
    {
        string[] pathSegments = imagePath.Split('/');
        string resourceName = pathSegments[2] + '/' + pathSegments[3].Replace(".png", "");
        / / .png などの拡張子を除く
        Sprite newSprite = Resources.Load<Sprite>(resourceName);
        if (newSprite != null)
        {
            targetImage.sprite = newSprite;
            UnityEngine.Debug.Log("Image changed successfully for "
                                        + targetImage.name);
        }
        else
        {
            UnityEngine.Debug.LogError("Failed to load image: "
```

```
                                        + resourceName);
        }
    }
    // Update is called once per frame
    public void battleSceneButton()
    {
        UnityEngine.Debug.Log("EnemyStatus.HP=" + EnemyStatus.HP);
        UnityEngine.Debug.Log("PlayerStatus.HP=" + PlayerStatus.HP);
        if (EnemyStatus.Speed <= PlayerStatus.Speed)
        {
            int flag = battleCalculation(ref PlayerStatus,ref EnemyStatus);
            if (flag == 0)
            { }
            //farstAttack が勝利
            else if (flag == 1)
            {
                SceneManager.LoadScene("resultScene(win)");
            }
            //敗北
            else
            {
                SceneManager.LoadScene("resultScene(lose)");
            }
        }
        else
        {
            int flag = battleCalculation(ref EnemyStatus,ref PlayerStatus);
            if (flag == 0)
            { }
            //farstAttack が勝利
            else if (flag == 1)
```

```
            {
                SceneManager.LoadScene("resultScene(lose)");
            }
            //敗北
            else
            {
                SceneManager.LoadScene("resultScene(win)");
            }
        }
        changeText();
    }
    int battleCalculation(ref Status farstAttack,ref Status secondAttack)
    {
        System.Random rnd = new System.Random();
        int randomInt = rnd.Next(1, 101);
        // 最初の攻撃
        if ((farstAttack.Critical / 2) < randomInt)
        { // クリティカル判定
            secondAttack.HP -= (int)((farstAttack.Attack / secondAttack.Defense) * 7.5);
        }
        else
        {
            secondAttack.HP -= (int)((farstAttack.Attack / secondAttack.Defense) * 5);
        }
        ChangeSliderValue(Eslider, (int)((secondAttack.HP * 100 / EnemyMaxHP)));
        UnityEngine.Debug.Log("secondAttack.HP=" + secondAttack.HP);
        UnityEngine.Debug.Log("secondAttack.HP / EnemyMaxHP" + (int)((secondAttack.HP /
EnemyMaxHP) * 100));
        UnityEngine.Debug.Log("EnemyMaxHP" + EnemyMaxHP);
        UnityEngine.Debug.Log("PlayerMaxHP" + PlayerMaxHP);
        if (secondAttack.HP <= 0)
```

```
            { //
                return 1; //  エネミーが倒れたら,プレイヤーの攻撃をスキップ
            }
            // エネミーの攻撃
            if ((secondAttack.Critical / 2) < randomInt)
            { // クリティカル判定
                farstAttack.HP -= (int)((secondAttack.Attack / farstAttack.Defense) * 7.5);
            }
            else
            {
                farstAttack.HP -= (int)((secondAttack.Attack / farstAttack.Defense) * 5);
            }
            ChangeSliderValue(Pslider, (int)((farstAttack.HP * 100 / PlayerMaxHP)));
            UnityEngine.Debug.Log("farstAttack.HP=" + farstAttack.HP);
            UnityEngine.Debug.Log("farstAttack.HP / PlayerMaxHP" + (int)((farstAttack.HP /
PlayerMaxHP) * 100));
            if (farstAttack.HP <= 0)
            { // ゲーム終了判定
                return 2;
            }
            return 0;
        }
        public void ChangeSliderValue(Slider slider, int newValue)
        {
            slider.value = newValue;
        }
        public void setStatus(ref Status status, String CsvLine)
        {
            string[] values = CsvLine.Split(',');
            if (values.Length >= 6)
  // CSV のデータが想定通りの要素数を持っているかチェック
```

```
            {
                status.HP = int.Parse(values[1]);
                status.Attack = int.Parse(values[2]);
                status.Defense = int.Parse(values[3]);
                status.Critical = int.Parse(values[4]);
                status.Speed = int.Parse(values[5]);
            }
        }
        void ReadDataFromCSV()
        {
            string filePath = "Assets/source_code/status.csv";
            using (StreamReader sr = new StreamReader(filePath))
            {
                string line;
                while ((line = sr.ReadLine()) != null)
                {
                    CsvList.Add(line);
                    CsvLength++;
                }
            }
            UnityEngine.Debug.Log("CSV file has been read.");
        }
        void changeText(String EnemyPash, String PlayerPash)
        {
            EnemyStatusGUI.text = statusTemp[0] + EnemyStatus.Attack + "¥n" +
                                  statusTemp[1] + EnemyStatus.Defense + "¥n" +
                                  statusTemp[2] + EnemyStatus.Critical + "%" + "¥n" +
                                  statusTemp[3] + EnemyStatus.Speed;
            PlayerStatusGUI.text = statusTemp[0] + PlayerStatus.Attack + "¥n" + statusTemp[1] +
PlayerStatus.Defense + "¥n" + statusTemp[2] + PlayerStatus.Critical + "%" + "¥n" + statusTemp[3] +
PlayerStatus.Speed;
```

```
        EnemyName.text = EnemyPash.Split('/')[3].Split('.')[0];
        PlayerName.text = PlayerPash.Split('/')[3].Split('.')[0];

        EnemyHP.text = EnemyStatus.HP.ToString() + "/" + EnemyMaxHP.ToString();
        PlayerHP.text = PlayerStatus.HP.ToString() + "/" + PlayerMaxHP.ToString();
    }
    void changeText()
    {
        EnemyStatusGUI.text = statusTemp[0] + EnemyStatus.Attack + "¥n" +
                              statusTemp[1] + EnemyStatus.Defense + "¥n" +
                              statusTemp[2] + EnemyStatus.Critical + "%" + "¥n" +
                              statusTemp[3] + EnemyStatus.Speed;

        PlayerStatusGUI.text = statusTemp[0] + PlayerStatus.Attack + "¥n" +
                              statusTemp[1] + PlayerStatus.Defense + "¥n" +
                              statusTemp[2] + PlayerStatus.Critical + "%" + "¥n" +
                              statusTemp[3] + PlayerStatus.Speed;

        EnemyHP.text = EnemyStatus.HP.ToString() + "/" + EnemyMaxHP.ToString();
        PlayerHP.text = PlayerStatus.HP.ToString() + "/" + PlayerMaxHP.ToString();

    }
}
```

4.6 リザルトシーン

図 4.6 リザルト画面

対戦が終わると戦闘結果によってそれぞれのリザルト画面に遷移する.画面中央にある「タイトルに戻る」を押されるとタイトル画面に遷移する.

第5章　まとめ・今後の展開

今回作成した「ジェスチャーで絵を描くシステム」は,作る予定だったハンドトラッキングを使ったお絵描きのシステムや AI を利用した戦闘力算出などの基本機能を実装し遊んで楽しめる段階まで完成することができた.

戦闘時のエフェクトや絵を描く際の線の色の変更など実装できていない機能も多くあった.今後これらを改善するのであれば,先ほど記述した戦闘時のエフェクト,UI,戦闘システム,絵を描く際の利便性を改修することでより良いものになると思う.

第 6 章　付録

6.1 Unity Hub のインストール

図 6.1　UnityHP

Unity 公式サイト　引用元：https://unity3d.com/jp/get-unity/download

ダウンロードした「UnityHubSetup.exe」を起動し，Unity Hub をインストールする．

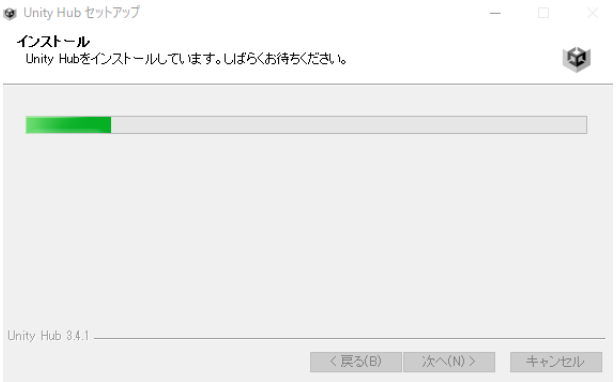

図 6.2 Unity Hub インストール画面

6.2 Unity エディターのインストール

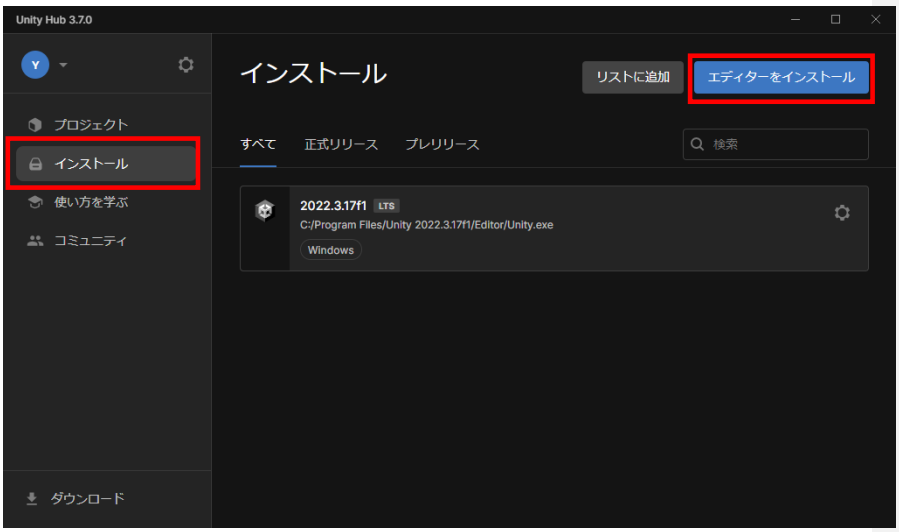

図 6.3 Unity Hub Top

図 6.4 Unity Hub Editor

自分の好きなエディターをダウンロード．今回は 2022.3.17 f 1 をダウンロードして使っている．

6.3 起動用バッチファイルの作成

起動用バッチファイルを作成し,Unity Hub を起動する.

```
@echo off
set HTTP_PROXY=http://172.16.0.2:8080
set HTTPS_PROXY=http://172.16.0.2:8080
start "" "C:¥Program Files¥Unity Hub¥Unity Hub.exe"
```

コード 6.1

このバッチファイルでプロキシを回避し,新規インストール,モジュールの追加,サインインなどが行える.

6.4 Unity の日本語化

歯車マークから Appearance を選び Language のところを選択し,日本語にする.

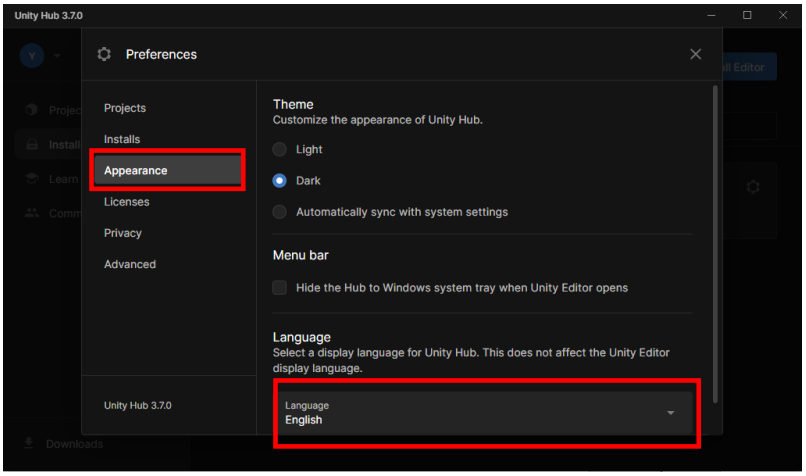

図 6.5 日本語化

6.5 Unity 内のテキストボックス等で日本語を使用したい場合

Google fonts から使いたい日本語のフォントを選ぶ

今回は Noto Sans Japanese を使用した.

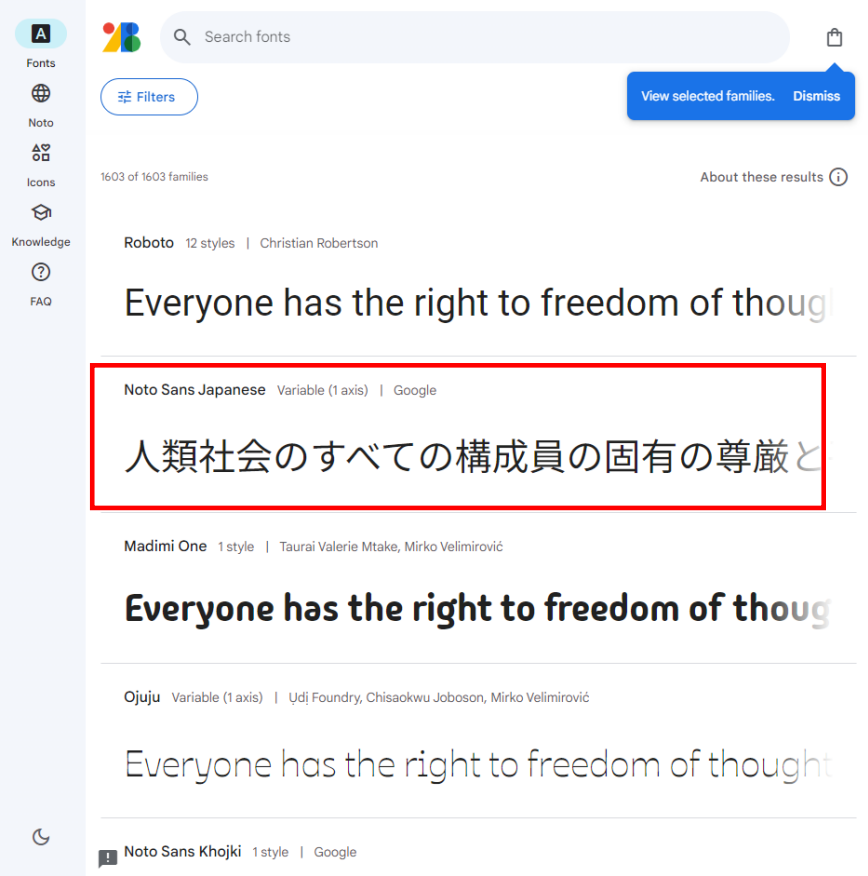

図 6.1　Noto Sans Japanese

Google fonts 公式　引用元：https://fonts.google.com/

ここでダウンロードしたファイルを展開し,static から NotoSansJP-Light.ttf を Unity 内で作成した

Font フォルダーに入れる.

https://gist.github.com/kgsi/ed2f1c5696a2211c1fd1e1e198c96ee4#file-japanese_full-txt

上記の URL から日本語データをダウンロードする.

上部のツールバーから Window > TextMeshPro > Font Asset Creator と進み

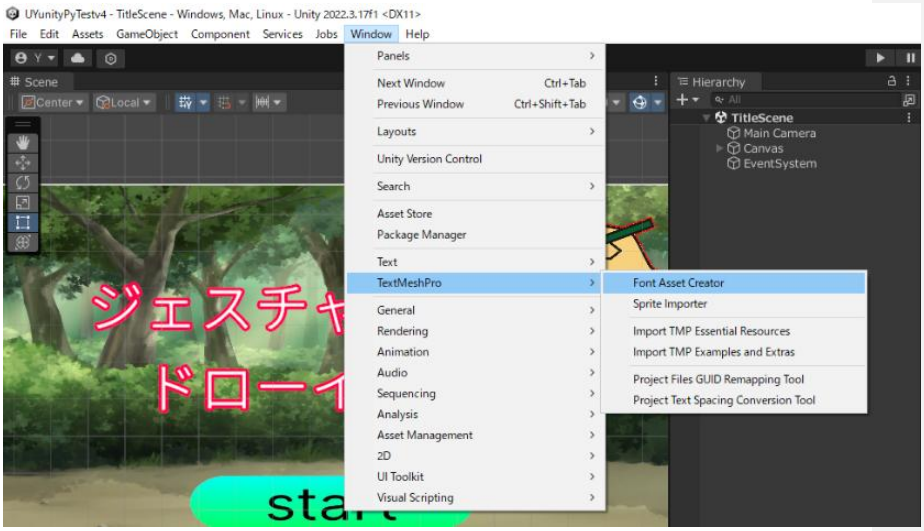

図 6.2　Unity ツールバー

Source Font File に Fonts 内に保存したファイルを設定する.

下図のように値を設定し,Generate Font Asset を押し Unity 内で使用できるフォントをダウンロードし保存する.

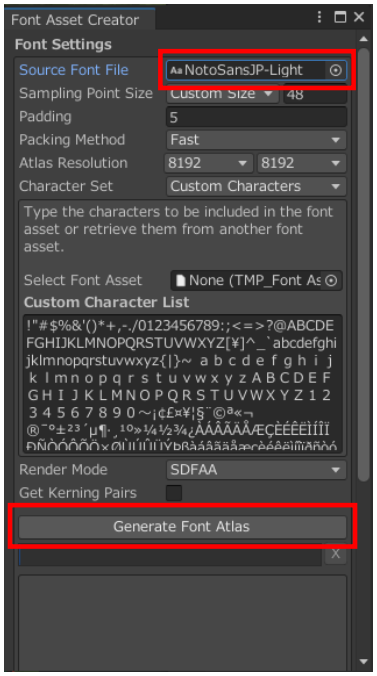

図 6.3 テキスト登録

参考文献

AI 関連

TensorFlow Core

https://www.tensorflow.org/tutorials/images/cnn?hl=ja

画像認識でよく聞く「CNN」とは？仕組みや特徴を 1 から解説

https://aismiley.co.jp/ai_news/cnn/

活性化関数(activation function) 【深層学習向け】

https://cvml-expertguide.net/terms/dl/layers/activation-function/#:~:text=%E6%B4%BB%E6%80%A7%E5%8C%96%E9%96%A2%E6%95%B0%20(acti

vation%20function%2C%20%E6%BF%80%E6%B4%BB%E5%87%BD%E6%95%B0)%E3%81%A8%E3%81%AF,%E3%81%AE%E4%B8%BB%E7%9B%AE%E7%9A%84%E3%81%A7%E3%81%82%E3%82%8B%EF%BC%8E

ハンドトラッキング関連

MediaPipe で遊んでみる

https://yoppa.org/mit-design4-22/14113.html

Web カメラでハンドトラッキングをしてみました

https://ccg-wheads.jp/journal/article-274/

MediaPipe を使って手から取得した骨格座標の情報を CSV に保存する

https://qiita.com/h-ueno2/items/b8dd54b396add5c3b12a

PC のマウス操作を非接触でやってみた -

https://techceed-inc.com/engineer_blog/6921/

MediaPipe で遊んでみる

https://yoppa.org/mit-design4-22/14113.html

初めての Unity 入門！プログラミングの基礎や独学法を徹底解説

https://www.sejuku.net/blog/56542

Unity 関連

【Unity】TextMeshPro で日本語を表示する方法

https://taidanahibi.com/unity/text-mesh-pro/

【Unity 入門】超簡単！別のシーンへ切り替える方法

https://www.sejuku.net/blog/49352