AWS Hack Gameplan

Models/Networks:

1. Ball Model
2. Body Model

MoveNet - Pose output (x, y, confidence) for every point on body
Tensorflow - Neural Networks
Keras API - Neural Networks
TensorBoard - Visualize the Data
OpenCV - Image data
Flask - Visual Interface for classifying the images

Classifiers

HoldingBall-45deg-r | CrossedArms-90deg-r | HeadShot



Hero | HandsOnHips-45deg-r | FullBody



CrossedArms-frontal | Celebration | HandsOnHips-90deg-l

HalfBody | HoldingBall | CrossedArms-90deg-l

HoldingBall-45deg-l | CrossedArms-45deg-l | HandsOnHips-45deg-l

HandsOnHips-90deg-r | CrossedArms-45deg-r

Classifications:

HoldingBall-45deg-r
CrossedArms-90deg-r
HeadShot
Hero
HandsOnHips-45deg-r
FullBody
CrossedArms-frontal
Celebration
HandsOnHips-90deg-l
HalfBody
HoldingBall
CrossedArms-90deg-l
HoldingBall-45deg-l
CrossedArms-45deg-l
HandsOnHips-45deg-l
HandsOnHips-90deg-r
CrossedArms-45deg-r

Move net Points

# Define the keypoint names
keypoint_names = [**'nose', 'left_eye', 'right_eye', 'left_ear', 'right_ear', 'left_shoulder', 'right_shoulder', 'left_elbow', 'right_elbow', 'left_wrist', 'right_wrist', 'left_hip', 'right_hip', 'left_knee', 'right_knee', 'left_ankle', 'right_ankle'**]

- Each point has a (x,y,confidence_score)
- Pass in 51 fields of data for every image
- 52 since each image already has a predefined classifier

Neural Network

- Multi-Layer **Perceptron (MLP)** is used for solving classification tasks
- Perfect model to classify the images
- Below doc has all I need to do this

Multi-Layer Perceptron Learning in Tensorflow - GeeksforGeeks
ReLU Activation Function in Deep Learning - GeeksforGeeks
Softmax Activation Function in Neural Networks - GeeksforGeeks
Keras Cheatsheet [2025 Updated] - Download pdf - GeeksforGeeks
Keras: The high-level API for TensorFlow | TensorFlow Core

Download Keras Cheat Sheet:
https://media.geeksforgeeks.org/wp-content/uploads/20250131104255802050/KerasCheatSheet.pdf

Hyperparameter Optimization:]
Introduction to the Keras Tuner | TensorFlow Core

Input dimensionality: Input will be 51 features (17 key points × 3 values per keypoint - x, y, confidence), Input Layer must account for this
Architecture considerations:

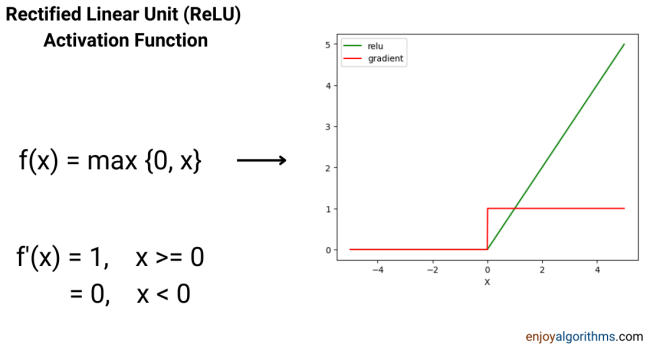
- 2-3 hidden layers (e.g., 128 → 64 → 32 neurons)
- Hidden layers compress the data and shows only the most important metrics for classifying the images
- Dropout (0.2-0.3) between layers to prevent overfitting
- Drop out randomly turns off a percentage of neurons at each layer (reduces overfitting and forces the network to not be dependent on specific neurons)
- Use ReLU activation for hidden layers

- At each hidden layer output, any negative values are set to 0. This means only the most important features and patterns for player poses are activated and passed forward.
- Fast and efficient
- For the output layer, use softmax activation with units matching your number of classes
- Softmax activation function is used to classify multiple classes based on the outputs from the hidden layers, the output layer will be in a range of (0,1), where each classifier has a confidence and all the confidences of classifiers sum to 1.

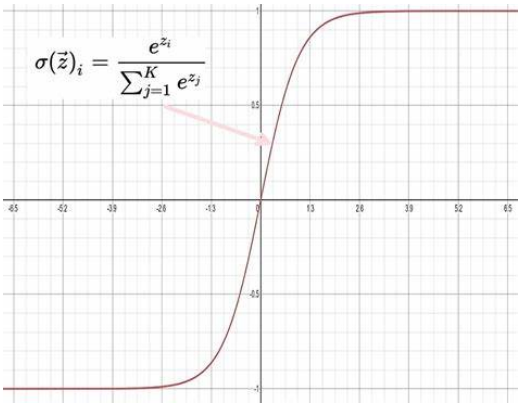
Normalization: Normalize your x,y coordinates to account for different image sizes (either to 0-1 range or using standardization)

Activation Functions

Hidden Layers: ReLU Activation Function



Output Layer: Softmax Activation Function



Differences between Activation Functions

Activation Function	Formula	Output Range	Advantages	Disadvantages	Use Case
ReLU	$f(x) = \max(0, x)$	$[0, \infty)$	- Simple and computationally efficient	- Dying ReLU problem (neurons stop learning)	Hidden layers of deep networks
			- Helps mitigate vanishing gradient problem	- Unbounded positive output	
			- Sparse activation (efficient computation)		
Leaky ReLU	$f(x) = \begin{cases} x & x > 0 \\ \alpha x & x \leq 0 \end{cases}$	$(-\infty, \infty)$	- Solves the dying ReLU problem	The slope α needs to be predefined	Hidden layers, as an alternative to ReLU
Parametric ReLU (PReLU)	Same as Leaky ReLU, but α is learned	$(-\infty, \infty)$	- Learns the slope for negative values	- Risk of overfitting with too much flexibility	Deep networks where ReLU fails
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	(0, 1)	- Useful for binary classification	- Vanishing gradient problem	Output layers for binary classification

			- Smooth gradient	- Outputs not zero-centered	
Tanh	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	(-1, 1)	- Zero-centered output, better than sigmoid	- Still suffers from vanishing gradients	Hidden layers, when data needs to be zero-centered
Exponential Linear Unit (ELU)	$f(x) = \begin{cases} x & x > 0 \\ \alpha(e^x - 1) & x \leq 0 \end{cases}$	$(-\alpha, \infty)$	- Smooth for negative values, prevents bias shift	- Slower to compute than ReLU	Deep networks for faster convergence
Softmax	$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$	(0, 1) (for each class)	- Provides class probabilities in multiclass classification	- Can suffer from vanishing gradients	Output layers for multiclass classification.

Keras API

- Can save the model after training and then recall for further training
- Can save the weights as well
- Preeeeatty neatttt

Hyper Parameters

- Units - Number of nodes at each layer
- Layers - Number of hidden layers
- Activation Functions - ReLU
- Epoch - Number of times running through the entire data set (forward and backward propagation for updating the weights based on the calculated weights)
    - new_weight = old_weight - learning_rate * gradient

- `model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])`

Extra Input Parameters

- Distance between hands and elbows (Help determine crossed arms)
- Distance between hands and hips (Help determine hands by side and hands on hips)

- Naive approach
    - Read all the images through opencv
    - Pass into the neural network