CS-630: APPLIED MACHINE LEARNING
DELIVERY MOTION CLASSIFICATION
FINAL REPORT

By Adnan Mohsin Ali

PROBLEM DESCRIPTION

Surveillance cameras like RingCentral can detect motion but can't specify what's moving, which is a problem in places like Dorchester and Roxbury in Boston where package theft is common. The challenge is to distinguish between delivery-related movement and other activities and to identify the delivery company involved. This lack of detailed motion classification leads to a higher risk of theft, as alerts don't provide enough information to prevent or investigate incidents effectively. To address this, an integrated model that can accurately classify and categorize motion is crucial for enhancing security and reducing parcel theft.

PROJECT OVERVIEW

This project focuses on improving security systems by creating a model that can tell apart actions related to delivering and non-delivery humans. Making this distinction is key to tackling the issue of stolen packages.

Delivery Classification: The **heart** of the system is a trained model that distinguishes between delivery and non-delivery motion. This ability is critical for spotting possible theft.

Logo Detection: The model's ability to correctly identify enhanced images by using InceptionV3, a cutting-edge technology that detects logos. It helps the system recognize and sort well-known delivery companies like Amazon, USPS, and Walmart, sharpening its ability to classify deliveries.

Integration: The project's highlight is the smooth combination of sorting deliveries and detecting logos. This combination creates a detailed model that not only notices movement but also understands what kind of movement it is, providing a strong tool to make homes and businesses safer.

GOAL

The goal of this project is to cut down on package theft in areas where it happens a lot by giving users clear and useful information about delivery events that their security cameras pick up.

DATA PREPROCESSING

DELIVERY CLASSIFICATION

The extraction of X and Y displacement coordinates of body parts relative to the midpoint between the left and right hips in sequential video frames involves multiple Python files, each with a specific role:

1. **getData.py**: This file is likely responsible for loading the video data. It may contain functions to read a video file frame by frame.
2. **detect_pose.py**: This script probably includes the functionality to detect human poses in each frame of the video. It uses a pretrained pose detection model (loaded from **graph_opt.pb**, a TensorFlow model file) to identify body parts and their coordinates.
3. **save_pose_to_dataframe.py**: After detecting poses in each frame, CSV file is used to save the extracted X and Y coordinates of the body parts into a structured format using a pandas DataFrame.
4. **pose_constants.py**: This file likely contains constants or reference data related to the pose detection task, such as names or indices of body parts.

```
BODY_COLUMNS = ['NoseX', 'NoseY', 'NeckX', 'NeckY', 'RShoulderX', 'RShoulderY', 'RElbowX', 'RElbowY',
                'RWristX', 'RWristY', 'LShoulderX', 'LShoulderY', 'LElbowX', 'LElbowY', 'LWristX', 'LWristY',
                'RHipX', 'RHipY', 'RKneeX', 'RKneeY', 'RAnkleX', 'RAnkleY', 'LHipX', 'LHipY', 'LKneeX', 'LKneeY',
                'LAnkleX', 'LAnkleY', 'REyeX', 'REyeY', 'LEyeX', 'LEyeY', 'REarX', 'REarY', 'LEarX', 'LEarY',
                'BackgroundX', 'BackgroundY']
```

5. **Reading CSV:** The script reads a CSV file containing body part coordinates. This file is generated from video data where poses are detected and saved.
6. **Calculating Displacement**: For each body part, the script calculates displacement between frames. It uses a reference point (midpoint between left and right hips) in each frame and computes the Euclidean distance from this reference point to the current position of each body part. This gives the displacement for each body part in each frame.
7. **DataFrame Output:** The calculated displacements for all body parts are then organized into a pandas DataFrame. Each row corresponds to a frame, and each column represents the

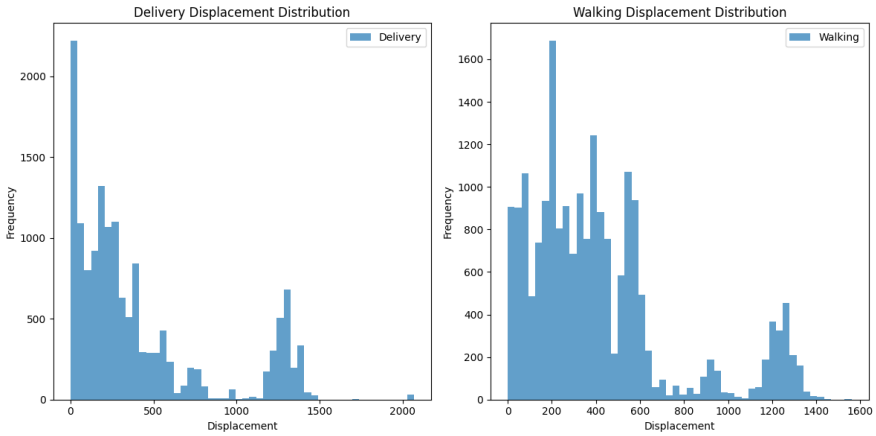
displacement of a specific body part in that frame as specified in BODY_COLUMNS list in the pose_constants.py.

LOGO DETECTION

1. **Image Resizing and Cropping:** Images are resized to a consistent shape (img_width x img_height) for input into the neural network. Cropping is performed based on annotation coordinates, ensuring that the relevant part of the image containing the logo is focused on.
2. **Data Augmentation: ImageDataGenerator** is used to augment the dataset by applying horizontal and vertical flips. This increases the diversity of the training set and helps prevent overfitting by simulating different perspectives of logos.
3. **Custom Data Generator:** A custom **LogoDataGenerator** class is implemented to feed the model batches of data. This generator processes each image by cropping and resizing, then applying any additional transformations specified in ImageDataGenerator.
4. **Normalization:** Pixel values are normalized to the range [0, 1] by dividing by 255. This standardizes the input data and can aid in training convergence.

PARAMETER TUNING WITH CHARTS –

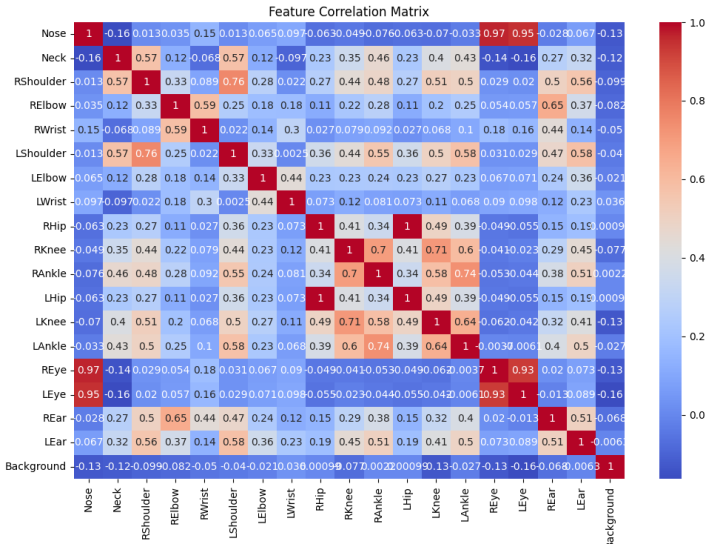
DELIVERY CLASSIFICATION



1. **Training Data Displacement Distribution** –

The histograms illustrate the displacement distributions for delivery and walking activities within the dataset. The delivery displacement shows a bimodal distribution, indicating periods of low and moderate movement typical during package handling or brief pauses. In contrast, the walking displacement histogram displays a multi-modal distribution with a broader spread of displacement values, reflecting the dynamic nature of walking. These differences in distribution patterns underscore the necessity for distinguishing between the two activities in our motion classification model.

2. **Feature Correlation** –



Highly correlated features can be redundant, and removing them can simplify the model without losing significant information.

3. **Feature Selection** –

   In enhancing the model, I analyzed feature correlations and importance. 'REye', 'REar', 'LEye', 'LEar', and 'Nose' were identified as highly correlated or less impactful and subsequently removed from the dataset. This reduction aimed to eliminate redundancy, improve model focus on key features, and potentially increase accuracy and training efficiency.

4. **Feature Scaling** –

   scaler = MinMaxScaler()

   train_data_scaled = scaler.fit_transform(train_data)

   test_data_scaled = scaler.transform(test_data)

In the preprocessing pipeline, I applied MinMax scaling to normalize the feature values. This scaling technique adjusts the data attributes to fall within a bounded interval of 0 to 1. Such scaling is beneficial for neural networks, as it ensures that all inputs operate on a comparable scale, promoting faster convergence. Additionally, it renders the model's output within a predictable range, which is particularly advantageous for activation functions that are sensitive to input magnitude.

GENERATION AND TUNING OF ALTERNATIVE MODELS –

DELIVERY CLASSIFICATION

Features/Insights	Model 1: "BasicNet"	Model 2: "StableNet"	Final Model (Model 3): "RobustNet"
Code	model = tf.keras.Sequential([layers.Input(shape=(len(BODY_PARTS),)), layers.Dense(64, activation='relu'), layers.Dense(32, activation='relu'), layers.Dense(1, activation='sigmoid')])	model = tf.keras.Sequential([layers.Input(shape=(len(BODY_PARTS),)), layers.Dense(64, activation='relu'), layers.Dropout(0.5), layers.BatchNormalization(), layers.Dense(32, activation='relu'), layers.Dropout(0.5), layers.BatchNormalization(), layers.Dense(1, activation='sigmoid')])	model = tf.keras.Sequential([layers.Input(shape=(len(BODY_PARTS),)), layers.Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.01)), layers.Dropout(0.5), layers.BatchNormalization(), layers.Dense(32, activation='relu', kernel_regularizer=regularizers.l2(0.01)), layers.Dropout(0.5), layers.BatchNormalization(), layers.Dense(1, activation='sigmoid')])
Basic Architecture	2 Hidden Layers	2 Hidden Layers + Batch Normalization	2 Hidden Layers + Batch Normalization
Batch Normalization	No	Yes (After Each Dropout Layer)	Yes (After Each Dropout Layer)
Regularization	No	No	Yes (L2 Regularization on Dense Layers)
Overview	Model 1 is characterized by its simplicity and faster training time, but this comes with an increased risk of **overfitting**, especially with these variable and large datasets, and the learning curve is plateau early,	Model 2 offers improved generalization capabilities through the incorporation of dropout and batch normalization, which lead to a more stable learning curve over epochs, though at the	The final model is designed to achieve the **best generalization, balancing dropout, batch normalization, and L2 regularization**. This combination likely leads to a slower training process, but with a **more gradual**

	indicating a potential underfitting to the task's complexity.	cost of slower convergence due to its added **complexity**.	**and consistent learning curve**, showing significant resistance to overfitting compared to the previous models.

FINAL MODEL "ROBUSTNET"–

1. **Data Import and Library Loading**: The script starts by importing necessary libraries like TensorFlow, Pandas, NumPy, and Matplotlib.

2. **Data Paths Specification**: It specifies file paths for training and testing data. These paths point to CSV files containing pose data for different activities (delivery and non-delivery).

```
delivery_paths_train = [
    'data_preprocessing/prepared_output/delivery1/csvFile/pose_data.csv',
    'data_preprocessing/prepared_output/delivery2/csvFile/pose_data.csv',
    'data_preprocessing/prepared_output/delivery3/csvFile/pose_data.csv'
]

walking_paths_train = [
    'data_preprocessing/output/sampleWalking/csvFile/pose_data.csv',
    'data_preprocessing/prepared_output/walking2/csvFile/pose_data.csv',
    'data_preprocessing/prepared_output/walking3/csvFile/pose_data.csv',
    'data_preprocessing/output/sampleWalking1/csvFile/pose_data.csv'
]

# Paths for delivery and walking data (testing)
delivery_paths_test = [
        'data_preprocessing/prepared_output/delivery1/csvFile/pose_data.csv',
        'data_preprocessing/prepared_output/delivery4/csvFile/pose_data.csv'
]

walking_paths_test = [
    'data_preprocessing/prepared_output/walking1/csvFile/pose_data.csv',
    'data_preprocessing/output/sampleWalking/csvFile/pose_data.csv'
]
```

3. **Data Loading and Preprocessing**:

   - The script loads displacement data for both activities from the specified paths using a custom **calculate_displacement** function.
   - It then concatenates data from all files for both training and testing sets and creates corresponding labels (1 for delivery, 0 for non-delivery).

4. **Data Sampling**: The script randomly samples 500 rows from both training and testing datasets to create smaller, manageable subsets. This step is crucial for handling large datasets or reducing training time.

5. **Neural Network Model Definition**

- The model is a Sequential model from Keras, with multiple Dense layers.
- It includes dropout layers for regularization (to prevent overfitting) and batch normalization layers (for faster convergence and better performance).
- The final layer uses a sigmoid activation function which is suitable for binary classification between non-delivery and delivery motion.

6. **Model Compilation**: The model is compiled with the Adam optimizer, binary crossentropy loss (appropriate for binary classification tasks), and metrics including accuracy, precision, and recall.

7. **Model Training**:

   - The model is trained on the sampled training data for 100 epochs with a batch size of 32.
   - Validation is performed using the sampled testing data.

BRIEF ABOUT THE LOGO DETECTION MODEL –

1. **Base Model Loading (InceptionV3):**

   - InceptionV3, a pre-trained model on ImageNet, is loaded without its top layer to serve as a feature extractor.
   - The output of the base model is fed into a global average pooling layer.

2. **Adding Custom Layers:**

   - A dense layer with 512 units and ReLU activation is added to learn more complex representations.
   - Dropout with a rate of 0.6 follows to prevent overfitting.
   - A final dense layer with a number of units equal to the number of classes, using softmax activation for multi-class classification.

3. **Unfreezing Layers:**

   - The last 50 layers of InceptionV3 are unfrozen, making them trainable. This allows the model to fine-tune these layers for the specific task of logo detection.

4. **Compilation:**

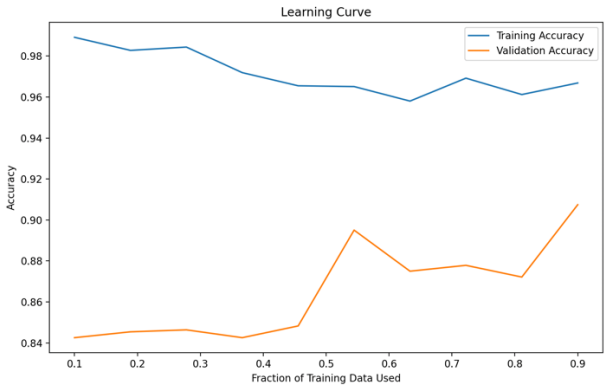
   - The model is compiled with the Adam optimizer with a learning rate that decays exponentially.
   - The loss function is categorical crossentropy, suitable for multi-class classification.
   - Accuracy, precision, and recall are the metrics for performance evaluation.

5. **Training:**

- The model is trained using a custom LogoDataGenerator, which yields batches of preprocessed images and their corresponding categorical labels.
- Early stopping is used to halt training if the validation loss does not improve for five consecutive epochs.

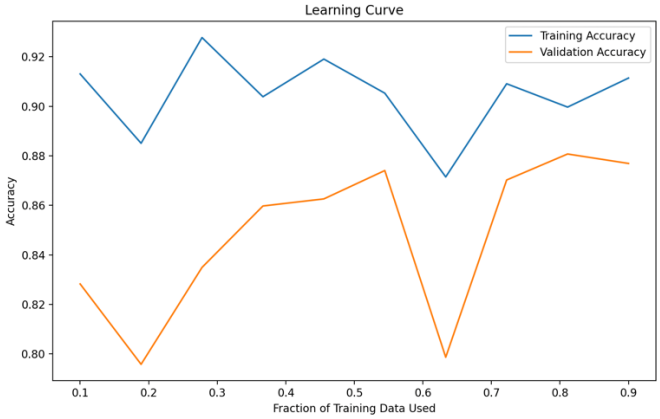
LEARNING CURVE ANALYSIS (DELIVERY CLASSIFICATION)–

1. **SimpleNet Model –**



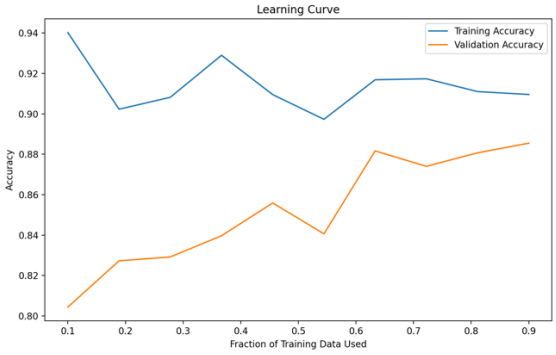
The learning curve indicates that as more training data is utilized, the model's validation accuracy fluctuates significantly, while the training accuracy remains relatively stable and high. This suggests that the model may be overfitting, as evidenced by high training accuracy paired with lower and more volatile validation accuracy. It implies that the model learns the training data well but struggles to generalize to unseen data, especially when the amount of training data increases.

2. **StableNet Model –**



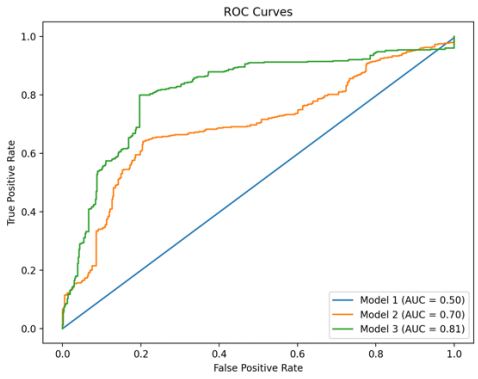
The learning curve shows the model's training accuracy remains consistently high, but validation accuracy exhibits significant variation with different fractions of training data used. Notably, there's a sharp decline in validation accuracy when half of the training data is utilized, followed by recovery as more data is included. This pattern could suggest that certain portions of the data are more challenging for the model, or there might be issues with the validation set's representativeness at specific training sizes.

3. Final (RobustNet) Model –

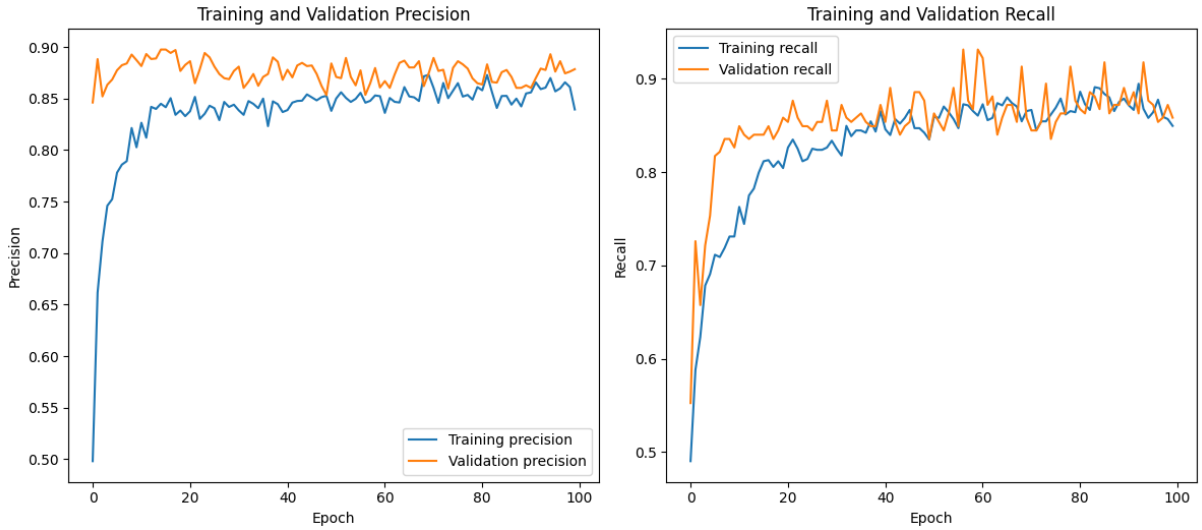


The learning curve illustrates the evolution of the model's accuracy as it is exposed to increasing proportions of the training exemplars. Initially, the model demonstrates signs of underfiting, as indicated by the lower validation accuracy. However, as additional data is incorporated, the model's performance on unseen data improves, reflecting its growing generalization capabilities. Notably, there is a point at which adding more data does not yield significant improvements in validation accuracy, which could suggest that the model has reached its capacity for generalization given the current architecture and dataset.

PERFORMANCE AND ERROR ANALYSIS (DELIVERY CLASSIFICATION) –



The ROC curves demonstrate the comparative performance of three classification models. Model 1 shows an AUC equal to 0.50, which implies no discriminative power between the classes. In contrast, Model 2 and Model 3 exhibit higher AUCs of 0.70 and 0.81, respectively, indicating their superior ability to correctly classify positive and negative instances. **Model 3 displays a commendable predictive performance, closely approaching the ideal point of perfect classification.**

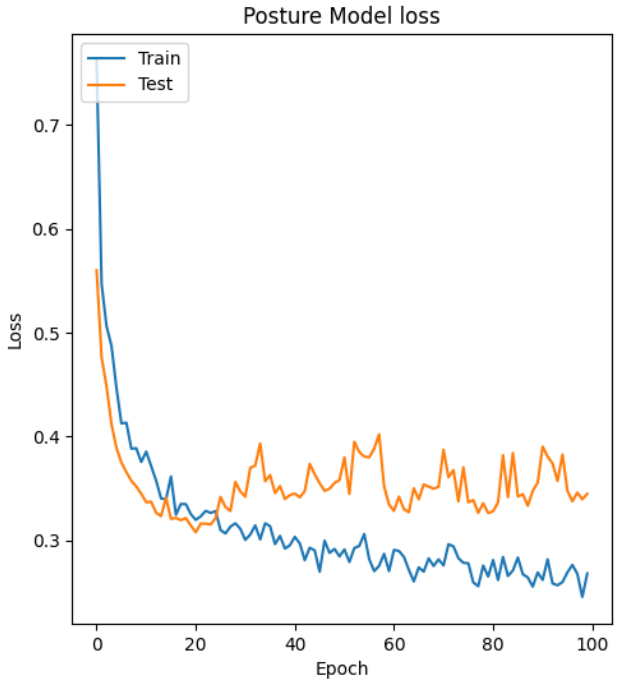


In the Training and Validation **Precision** graph:

- The training precision starts high and remains relatively stable throughout the training process, indicating that the proportion of true positive predictions to total positive predictions is consistently high.
- The validation precision fluctuates more but shows an increasing trend, suggesting that the model is gradually improving at correctly identifying positive cases as it processes more data.

In the Training and Validation **Recall** graph:

- The training recall sharply increases in the initial epochs, then levels off, maintaining a high recall which implies that the model is consistently identifying most of the actual positive cases during training.
- The validation recall also increases notably at the beginning and then demonstrates some variability, reflecting the model's ability to generalize in identifying true positives across different datasets.



Key observations in **Loss per epochs**:

- The training loss decreases sharply at the beginning, which is typical as the model initially learns from the data.
- After the initial descent, the training loss continues to decrease but at a slower and more fluctuating pace, indicating the model is making incremental improvements as it learns.
- The test loss, after an initial sharp decline, shows greater fluctuation compared to the training loss. This suggests the model may be experiencing difficulty in generalizing to unseen data, or the test set contains more complex or noisy examples than the training set.

WORKS CITED

- Durupinar, F. (2021). Perception of Human Motion Similarity Based on Laban Movement Analysis. *ACM Symposium on Applied Perception 2021*, 1–7. https://doi.org/10.1145/3474451.3476241
- Durupinar, F., Kapadia, M., Deutsch, S., Neff, M., & Badler, N. I. (2017). PERFORM: Perceptual Approach for Adding OCEAN Personality to Human Motion Using Laban Movement Analysis. *ACM Transactions on Graphics*, *36*(1), 1–16. https://doi.org/10.1145/2983620
- Wikipedia The Free Encyclopedia. (2022). *An example of computer animation using motion capture*. Computer animation. Retrieved December 2, 2022, from https://upload.wikimedia.org/wikipedia/commons/6/6d/Activemarker2.PNG.
- OpenAI. (2023). ChatGPT (Version GPT-3.5) [Software]. https://openai.com/chatgpt