Carnegie Mellon University

Live Motion Direction Estimation

Peter Blumenstein (pblumens)
Rithwik Jayanth (rjayanth)
Akshay Raman (akshayra)

Carnegie Mellon University

Problem Statement

Correctly identifying camera motion from video feeds is integral for real world applications

- Self driving cars
- Stabilization of objects
- Trajectory mapping

Carnegie Mellon University

Carnegie Mellon University

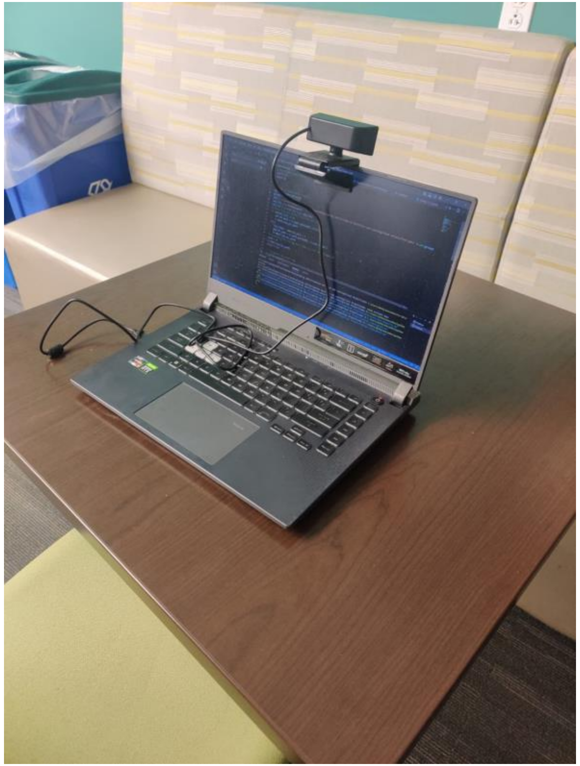
Data Collection

Dataset

- Consists of ~4000 images.
  - Videos were collected and converted to frames (images)
  - Images then converted into disparity maps

- Dataset has 5 classes
  - Each class is based on -
    - *Forward*
    - *Backward*
    - *Turning Left*
    - *Turning Right*
    - *Standing Still*

Dataset collection

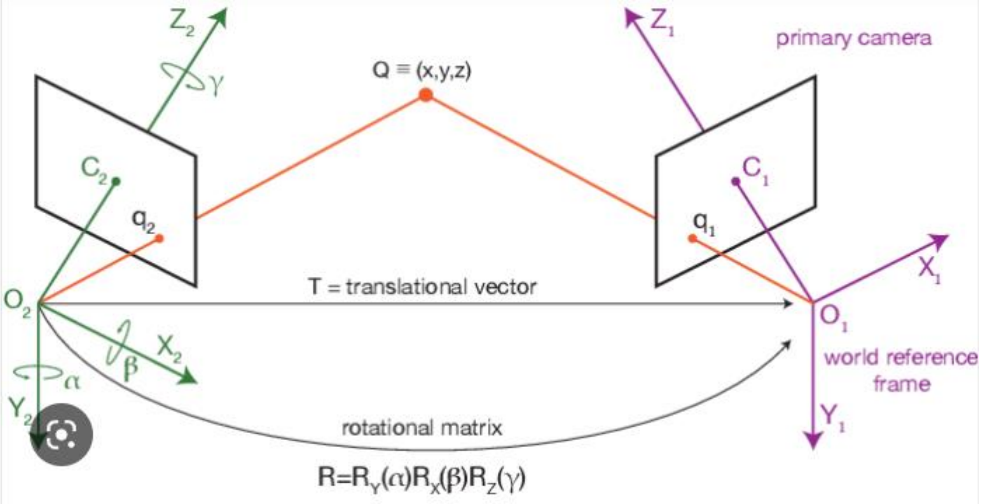


Feature Extraction

Feature Extraction: Frame Change Fundamentals

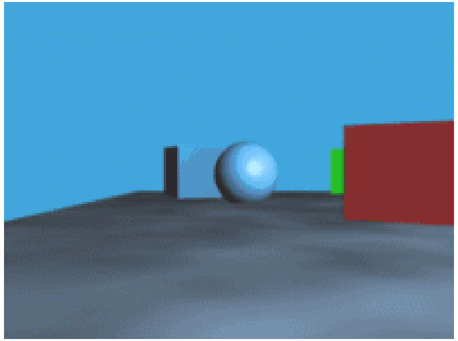
Capture Change in Movement:

- Process every 5th frame:
  - Allows for enough average change in pixels over the given period
  - Capture change in images over some translation to produce stereo image



Carnegie Mellon University

Feature Extraction: Disparity Maps



Disparity Map created from 2 images

- Loop through each test data class and each individual test number
  - Images were converted to grayscale first
  - CV2.stereoSGBM_create function used
    - *Parameters were played around with interactive tuning parameters*
    - *Tuning file given as stereo_vision_tuning.py*
    - *Final Parameters: numDisparities=16, blockSize=9*
  - Disparity Maps saved into respective folders

Why Disparity Map?

- Returns apparent pixel difference or motion between a pair of stereo images
- Normalizes the motion based on depth - Objects further shift less over time
- Estimating the behavior of disparity map across epipolar line - Learning

Carnegie Mellon University

Feature Extraction - Examples

Carnegie Mellon University

Classification Models

The Naive Bayes Classifier scored 74.7% accuracy on the test data.

- Preprocessing
  - Images were stored in folders according to their correct labels.
  - 2x2 max pooling block filter reduces overall size
  - Reshape the image to a vector of disparity values

- Model Parameters
  - Priors: none were used
  - Variance: $1x10^{-9}$
  - Training - 80 %
  - Testing - 20 %

Using Support Vector Classification yielded an accuracy score of 93%

- SVC - Multiclass Support Vector Machine (SVM)
- Preprocessing
  - Images were stored in folders according to their correct labels.
  - 2x2 max pooling block filter reduces overall size
  - Reshape the image to a vector of disparity values

- Model Parameters
  - Kernel: rbf
  - Degree of polynomial: 3
  - Tolerance: 0.001
  - Training - 80 %
  - Testing - 20 %

Using Linear Neural Networks yielded an accuracy score of 83%

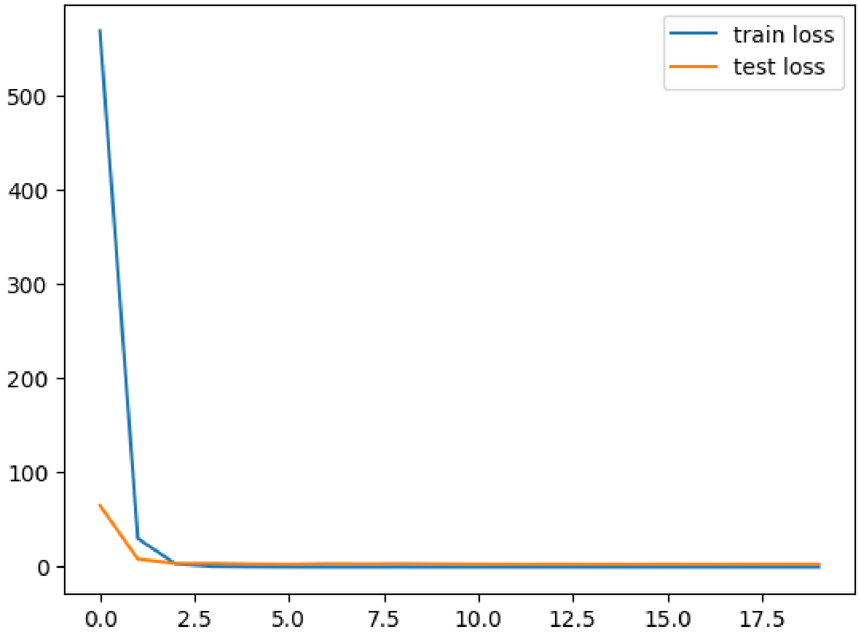
Preprocessing:

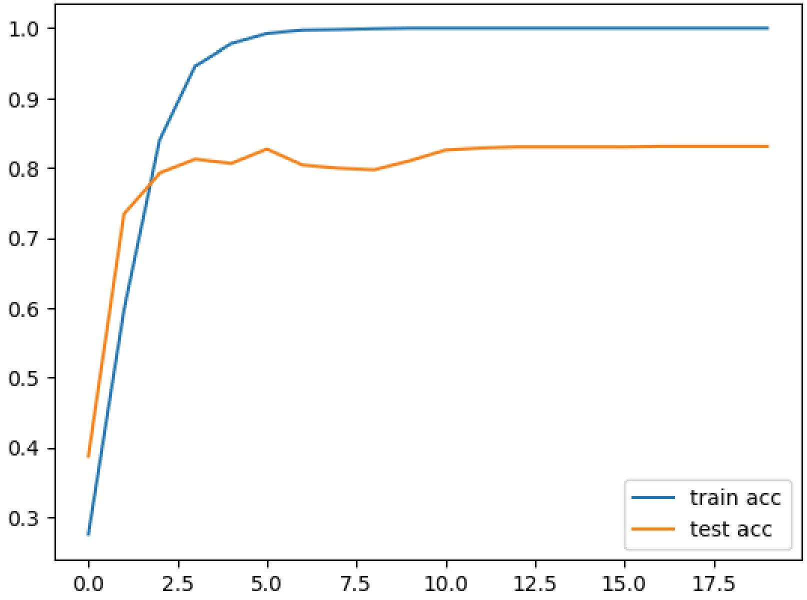
- Resizing image data to 300 x 300 pixels using Image library's resize method.
- Normalizing pixel values by dividing each by 255 to scale between 0 and 1.
- 46,966 images with corresponding labels split into 31,458 for training and 15,508 for testing using 67:33 split ratio.

Model Parameters:

- LinearModel is a feedforward neural network with 3 fully connected layers.
- Input image is flattened into 1D tensor before passing through the network.
- Output layer has 5 features, indicating a multi-class classification task.

Linear NN: Model Results over Epochs





Carnegie Mellon University

Classification Model: Convolutional Neural Network (CNN)

Preprocessing:

- Resized image from (480,640) to (300,300) to reduce dimensionality for quicker processing
- Loop over Disparity Map and place into tensor.
  - Concatenate with labels
- Set batch size to 100 per packet for CNN model

Model Parameters:

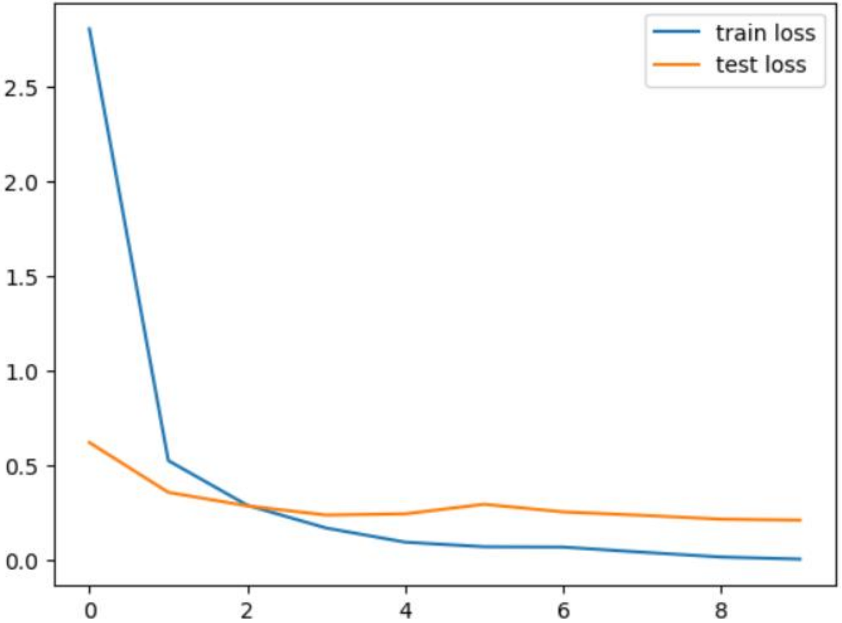
- 1 Input Channel: grayscale
- 3 hidden layers with ReLU and Pooling between each layer
- 2 Linear Layers with ReLU between each layer
- Cross Entropy Softmax layer to classify into 5 classes
- Run 20 epochs over dataset

Overfitting Problem!

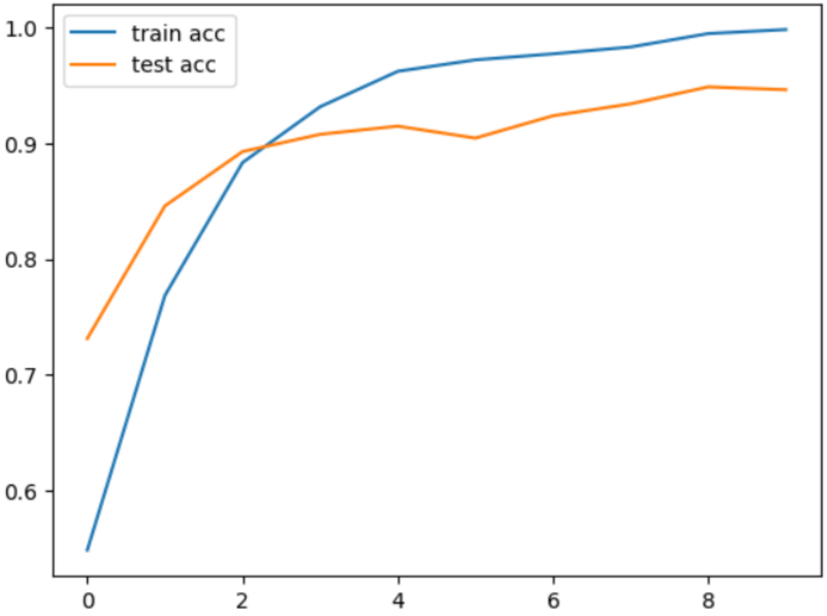
- Implemented dropout to correct for overfitting! (0.2 dropout)

Carnegie Mellon University

CNN: Model Results over Epochs



Loss for Train and Test over 20 Epochs



Accuracy for Train and Test over 20 Epochs

Carnegie Mellon University

Carnegie Mellon University

Results

Results Across all Models

Model	Train Accuracy	Test Accuracy
Naive Bayes	-	74.7%
SVC	-	93%
Deep Neural Network	99%	83%
Convolutional Neural Network	98%	94.6%

Carnegie Mellon University

Conclusion: Results Ablation Study

SVC Vs Naive Bayes:

Formula:

$$P(\text{class} \mid \text{data}) = \frac{P(\text{data} \mid \text{class})\, P(\text{class})}{P(\text{data})}$$

- Holds a conditional independence assumption - Pixels can often tell neighboring pixels. Usually the same.

Look at SVC:

- SVC creates a soft margin for classification
- Performs worse for larger data sets
    - Noise in dataset will cause overfitting towards training data

DNN Vs CNN:

- As an image processing problem, best model aligns with CNN method
- DNN is more variant to noise in dataset and trains weights closer to training dataset.
    - Overfitting to dataset main concern
- CNN can learn better filters to create more accurate weights
- Vary Kernel size, stride to create different filter iteration

Live Motion Capture Prediction:

Code for Running this located in CNN_model notebook file

Observations:

- Top Red text shows current class
- Overall good performance.
- Slight jitter causes the class to change even when ground truth remains same

Future Improvements

- Add data augmentation to account for jitter
- Reinforce network by training on individual frames in addition to disparity map
- Regularization

Carnegie Mellon University

Carnegie Mellon University

Code Snippets

Feature Extraction and Video Capture

Video Capture Loop

```
os.chdir(new_dir_path)
while(True):

    # Capture the video frame
    # by frame
    ret, frame = vid.read()

    # Display the resulting frame
    cv2.imshow('frame', frame)
    cv2.imwrite('Frame'+str(count)+'.jpg', frame)
    count = count + 1


    # the 'q' button is set as the
    # quitting button you may use any
    # desired button of your choice
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# After the loop release the cap object
vid.release()
# Destroy all the windows
cv2.destroyAllWindows()
```

Feature Extraction
Snippet of full file

```
# Function gets disparity map by comparing every 5 images
def get_disparity_map_test(path, class_type):
    # Get correct directory
    dir_list = os.listdir(path)
    dir_list = np.array(dir_list)
    class_dir = 'C:/Users/aksha/OneDrive/Desktop/Desktop/Spring 2023/Estimatio
    class_dir = class_dir + '/' + class_type + '/disp'
    count = 0
    # Loop through each test case
    for i in range(len(dir_list)-5):
        # Get image path for every 5th image and compare
        frame_num_1 = dir_list[i]
        frame_num_2 = dir_list[i+5]
        path_1 = path + '/' + frame_num_1
        path_2 = path + '/' + frame_num_2
        gray_1 = get_image_binarize(path_1)
        gray_2 = get_image_binarize(path_2)
        # Compare grayscale images between every 5th image and disparity map
        stereo = cv2.StereoBM_create(numDisparities=16, blockSize=9)
        # Save disparity map
        disparity = stereo.compute(gray_1,gray_2)
        disp_path = class_dir+str(count)+'.jpg'
        cv2.imwrite(disp_path, disparity)
        count = count + 1
```

DNN Model and CNN Model Code Snippet

```
class Conv2D(nn.Module):
    def __init__(self):
        super().__init__()
        self.network = nn.Sequential(

            nn.Conv2d(1,20,5),
            nn.ReLU(),
            nn.MaxPool2d(2,2),
            nn.Conv2d(20,64,5),
            nn.ReLU(),
            nn.MaxPool2d(4,4),
            nn.Conv2d(64,64,7),
            nn.ReLU(),
            nn.MaxPool2d(4,4),

            nn.Flatten(),
            #nn.Linear(3136,6000),
            #nn.ReLU(),
            nn.Linear(3136,1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512,5)
        )
```

CNN Model Initialization

```
class LinearModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.network = nn.Sequential(
            nn.Flatten(),
            nn.Linear(90000, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 5)
        )

    def forward(self, x):
        return self.network(x)

model = LinearModel()
```

DNN Model Initialization

```
## Model Creation and Application
X_train, X_test, y_train, y_test = train_test_split(

# Use support vector classification
clf = SVC()
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_pred,y_test)
print("Prediction accuracy:",accuracy*100,"%")
```

SVC Model Initialization

Carnegie Mellon University