Surgical Tool Presence Detection

V L Sahithi
I.MTech CSE
Internation Institute of Information Technology
Bangalore, India
lakshmi.sahithi@iiitb.ac.in

***Abstract*—This document is a report of the Project of Surgical Tool presence Detection done under *Prof. Neelam Sinha, IIITB* and *Sai Pradeep, PhD, IIITB* in the Spring 2023.**

I. Aim

The objectives of the project can be summarized as follows:

- Understanding the m2cai-16 tool dataset and annotations
- understanding popular CNN architecture
- Applying CNN on m2cai16-tool dataset

II. Dataset Details

The *m2cai16 tools* dataset consists of 15 cholecystectomy vidoes with ground truth binary annotations of the present tools. The dataset consists of videos recorded during actual surgical procedures specifically cholecystectomy Each video is accompanied by various annotations and ground truth data. The annotations include information such as the surgical phase, instrument presence, instrument type, and instrument usage. The ground truth data provides additional details, such as the timestamped surgical events.

III. Literature Survey

In this project, I thoroughly reviewed the official papers like [1], [2], [4], and [3] to understand the dataset, I read these research papers to gain an understanding of CNN's and their architectures and how they work on various datasets like image net to extract the features. I have used various CNNs like lenet,alexnet,zfnet, google net, vgg, and poly net. So in this report, I will discuss the architectures of these models in brief.

IV. Architectures

The architecture of a CNN can be broken down into three main components: convolutional layers, pooling layers, and fully connected layers.

Convolutional Layers:

Convolutional layers are the building blocks of a CNN. Each convolutional layer consists of a set of filters or kernels that convolve over the input image to extract important features. These filters learn to identify patterns in the input image, such as edges or corners. Each filter produces an output in the form of a feature map. The depth of the feature map depends on the number of filters used.

Pooling Layers:

Pooling layers follow convolutional layers and help to reduce the dimensionality of the feature maps while preserving the important features. Pooling is done by applying a function to a small region of the input feature map, such as taking the maximum or average value. This reduces the size of the feature maps and makes the network more computationally efficient.

Fully Connected Layers:

The output of the final pooling layer is then flattened into a one-dimensional array and passed through a series of fully connected layers. These fully connected layers perform classification based on the features extracted by the previous convolutional and pooling layers. The output of the last fully connected layer is a probability distribution over the different classes in the dataset.

A. lenet

LeNet5 is a relatively simple CNN architecture with only 5 layers It has fewer parameters and is computationally less expensive compared to AlexNet. first layer is the input layer with feature map size 32X32X1. first convolution layer with 6 filters of size 5X5 and stride is 1. The activation function used at his layer is tanh. The output feature map is 28X28X6. Gives 6 feature maps. Subsampling uses max pooling to reduce the size of feature maps. Then we have avg pooling layer this doesn't affect the number of channels Then we have 2nd convolution layer with 16 filters of 5x5 Then avg pooling Final pooling layer has 120 filters so the output size is 120. This is given to fully connected layer. The last layer is the output layer and it has Softmax function. The Softmax gives the probability that a data point belongs to a particular class. The highest value is then predicted. This is the entire architecture of the Lenet-5 model.

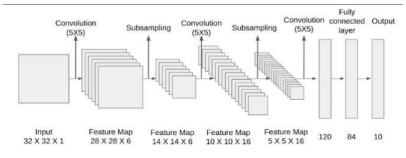


Fig. 1: lenet

B. Alexnet

Alexnet is a much deeper CNN architecture with 8 layers, designed to classify images in the ImageNet dataset, which contains over a million images in 1000 different classes. AlexNet has significantly more parameters and requires more

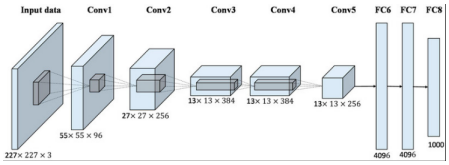


Fig. 2: alexnet

computational power than LeNet5, It has 8 layers with learnable parameters. It has 5 convolution layers with a combination of max-pooling layers. Then it has 3 fully connected layers. The activation function used in all layers is Relu. But the output layer uses softmax.

C. VGG

The VGG CNN architecture is a popular deep learning model architecture for image classification tasks, which involves classifying an image into one of several predefined categories.VGG16 has a total of 16 layers that has some weights. Only Convolution and pooling layers are used. Always uses a 3 x 3 Kernel for convolution. 20 2×2 size of the max pool. The input to the convolution neural network is a fixed-size 224 × 224 RGB image. The only preprocessing it does is subtracting the mean RGB values,

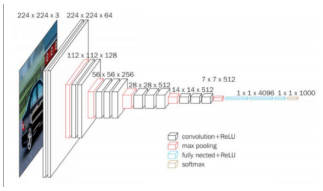


Fig. 3: VGG

D. Polynet

The PolyNet architecture is based on the Inception architecture, which is known for its use of ”Inception modules” that combine different convolutional filter sizes and pooling operations. However, PolyNet extends this concept by introducing a new type of module called a ”PolyInception module”. PolyNet traverses the whole network and explores the entire space, making decisions about weights and structure so that it may automate improvements to increase performance and functionality, with better results for the end user. A PolyInception module is similar to an Inception module, but it includes an additional ”polynomial activation function” that is applied to the output of the module. This activation function is designed to increase the expressive power of the network by allowing it to learn more complex features. In addition to the PolyInception modules, the PolyNet architecture also includes several other features that improve its efficiency and scalability.

V. ZFENET

The architecture of ZFNet is similar to the AlexNet architecture, but with some modifications to improve its performance.

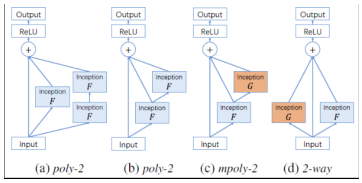


Fig. 4: Polynet

It consists of 8 layers, including 5 convolutional layers, 2 fully connected layers, and a softmax output layer. The first convolutional layer has a large receptive field of 11x11 pixels, which allows it to capture global features of the input image. The subsequent layers have smaller receptive fields, allowing them to capture more local features. In ZFNet we use a visualization technique called Deconvolutional Networks, which allows the network to produce a heat map showing which parts of the input image contributed the most to the classification decision. This visualization technique helps to understand how the network is making its decisions and can be used for debugging and improving the model. ZFNet uses a visualization technique called Deconvolutional Networks to understand how the model is making its classification decisions by generating a heatmap that highlights the important regions of the input image that contributed to the decision.

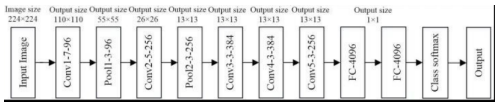


Fig. 5: VGG

VI. BACKGROUND

Main steps to follow to do surgical tool presence detection:
Taking video and text file as input
extracting frames
we have to use these frames to train the cnn we use
data augmentation
model architecture
optimizer
testing the model with testing data

VII. METHODOLOGY

A. DATA PREPARATION AND MODEL TRAINING

TRAIN-TEST DATA SEPERATION:

We define the path to the folder which contains the video files and create a list of video files with the '.mp4' extension. Now we define a dictionary called label-dict that maps video file names to their corresponding labels. we also initialize empty lists for storing frames and labels. Then we iterate over each video file and check if the video file's name is present in the label-dict and assigns the corresponding label. If not found, it assigns the label as ”unknown.” It constructs the path to the text file corresponding to the video file. It opens the text file and reads its lines, storing each line as an instrument. It opens

the video file using OpenCV's VideoCapture and checks if it was successfully opened.

In the code provided, the values 1, 2, 3, 4, 5, 6, and 7 are used to represent different surgical instruments in the 'instruments-array' variable. The mapping of the values to the instruments is as follows:

grasper
Bipolar
hook
scissors
clipper
irrigator
specimenbag

FRAME GENERATION:

Now we loop over each frame in the video and then resize the frame to (224, 224) dimensions. It appends the frame to the frames list. It constructs a label vector based on the assigned label. If the label is "label1," it sets the first element of the vector to 1; otherwise, it sets the second element to 1. It constructs an instrument array based on the detected instruments. Each instrument is mapped to a value from 0 to 6. It concatenates the label vector and instruments array into a single array and appends it to the labels list.

Now we convert frames and label lists into NumPy arrays and then we initialize an ImageDataGenerator for data augmentation during training.

CNN MODEL IMPLEMENTATION:

Now we define a cnn model and compile the model with the Adam optimizer, a learning rate of 0.0001, categorical cross-entropy loss, and accuracy metric. Then we initialize a train generator using the ImageDataGenerator and the X-train and y-train arrays. It trains the model using the fit method, specifying the trained generator, the number of steps per epoch, and the number of epochs.

TEST DATA PREPARATION:

Now we have to test this trained model on a testing video file which also contains a corresponding text file that contains the instrument information. We initialize test frames to store the frames from the test video. It loops over each frame in the test video and then it resizes the frame to (224, 224) dimensions. It also appends the frame to the test-frames list. It opens the test text file and reads its lines, storing each line as an instrument in the test-instruments list.

Now we convert the test-frames list into a NumPy array. Then we initialize an array called test-instruments-array with zeros to represent the instruments detected in the test video. It loops over each instrument in test instruments: It maps each instrument to a value from 0 to 6 based on the mapping you provided. It assigns the corresponding value to the appropriate index in test-instruments-array. Now it constructs the labels for the test video by concatenating a label vector with the test-instruments-array. The label vector has a single zero followed by test-instruments-array. It replicates the labels array for each frame in the test-frames array.

PREDICTION ON TEST DATA:

Now we use the trained model to predict the labels for the test frames and assign the predictions to test the prediction. We extract the true labels and instruments from test labels using argmax. It compares the predicted labels and instruments with the true labels and instruments to count the number of correct predictions. It calculates the test accuracy by dividing the number of correct predictions by the total number of predictions and multiplying by 100.

B. DATA SPLITTING AND RESAMPLING

DATA PREPARATION:

Here we use 3 videos to train the model and use train test split to split them into training and testing sets with 80 percent training and 20 percent testing.

The merge-data function takes a video file and a text dataset which contains instrument information as input and merges them into a single dataset. It reads frames from the video, resizes them to a specified shape, and retrieves the corresponding instrument information from the text dataset based on frame numbers. It then returns a list of tuples, where each tuple contains a frame and its corresponding instrument information.

RESAMPLING AND DATA AUGMENTATION:

The preprocess-data function takes the merged data as input and performs preprocessing steps. It first normalizes the pixel values of each frame by dividing them by 255.0. Then, it separates the frames and instrument information into two separate lists, X and y. Next, it oversamples the minority class (label 0) to balance the dataset using the resample function from the sklearn.utils module. After oversampling, it concatenates the oversampled minority class with the majority class (label 1) to create a balanced dataset.

Resampling:

Class imbalance occurs when the number of samples in each class is significantly different, which can lead to biased model predictions.

By performing resampling, specifically oversampling the minority class, the code aims to create a more balanced dataset where the number of samples in each class is approximately the same. This is achieved by randomly duplicating samples from the minority class until its size matches that of the majority class. Resampling is a technique used to address class imbalance in a dataset. Class imbalance occurs when the number of samples in each class of the target variable is significantly different. This can negatively affect the performance of machine learning models, as they tend to be biased towards the majority class.

Class imbalance can cause the model to be biased towards the majority class, resulting in poor performance on the minority class. By increasing the number of samples in the minority class, the model has more opportunities to learn from these samples and make more accurate predictions.

In the provided code, resampling is performed using the resample function from the sklearn.utils module. It is applied to the minority class (label 0) to increase the number of samples and balance the dataset. The augmentation step

further enhances the generalization capability of the model. By applying various transformations to the data, it introduces additional variability and helps the model learn more robust and diverse patterns.

Data augmentation:

Data augmentation refers to the process of artificially increasing the size of a training dataset by creating modified versions of the original images through a series of transformations, such as rotating, flipping, cropping, zooming, or changing the brightness or contrast of the images. The goal of data augmentation is to increase the variability and diversity of the training data, which can help improve the performance of machine learning models by reducing overfitting and improving their ability to generalize to new, unseen data. This code initializes an instance of the ImageDataGenerator class from the Keras library. ImageDataGenerator is a powerful tool for data augmentation in deep learning, allowing for the generation of artificially augmented training data from a relatively small set of original images. The parameters passed to the constructor specify the types of transformations that will be applied to the images during training. Specifically, the rotation-range parameter specifies the maximum degrees of random rotation that will be applied to the images, width-shift-range and height-shift-range specify the maximum amount of horizontal and vertical shift that will be applied to the images, zoom-range specifies the maximum zoom that will be applied to the images, and horizontal-flip and vertical-flip specify whether the images will be flipped horizontally and/or vertically.

It returns the preprocessed X-oversampled (frames) and y-oversampled (instrument information) arrays.

CNN MODEL IMPLEMENTATION:

We define a creat-model function. which creates and compiles a CNN model using the Sequential API from TensorFlow Keras. The model consists of several convolutional, pooling, and dense layers with activation functions such as tanh and softmax. The model is compiled with the SGD optimizer and categorical cross-entropy loss.

Now we initialize video1, video2, and video3 as video capture objects, and text1, text2, and text3 as data frames read from text files and then we call the merge-data function to merge the video and text data for each set (video1 with text1, video2 with text2, and video3 with text3), resulting in data1, data2, and data3. Now we call the preprocess-data function to preprocess the merged data for each set (data1, data2, and data3), resulting in X1, y1, X2, y2, X3, and y3. here we concatenate the preprocessed data from the three sets (X1, X2, X3, y1, y2, y3) to create the training and testing sets (X-train, X-test, y-train, y-test).

PREDICTION ON TESTING DATA:

Now we call the create-model function and we fit the the model to training data using the fit function. Then we will evaluate the model's performance on the 20 percent testing data and get a classification report.

VIII. Results for methodology-I

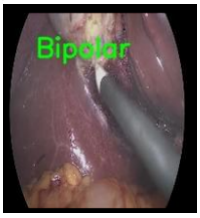
Model	Accuracy
Lenet	74.345
Zfnet	62.347
Google-net	59.592
Alexnet	54.316
Vgg	50.000
Polynet	46.128

Fig. 6: accuracies for methodology-I

For lenet

```
Accuracy for instrument class 1: 0.67
Accuracy for instrument class 2: nan
Accuracy for instrument class 3: 0.56
Accuracy for instrument class 4: nan
Accuracy for instrument class 5: nan
Accuracy for instrument class 6: nan
Accuracy for instrument class 7: nan
```

(a) Accuracies for lenet



(b) Random Frame

Fig. 7: leNet Results

For zfnet

```
Accuracy for instrument class 1: 0.75
Accuracy for instrument class 2: nan
Accuracy for instrument class 3: 0.42
Accuracy for instrument class 4: nan
Accuracy for instrument class 5: nan
Accuracy for instrument class 6: nan
Accuracy for instrument class 7: nan
```

(a) Accuracies for ZFNet

(b) Random Frame

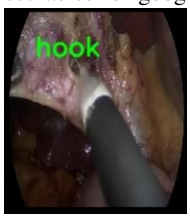
Fig. 8: ZFNet Results

For googelnet

IX. Results for methodology-II

```
Accuracy for instrument class 1: 0.53
Accuracy for instrument class 2: nan
Accuracy for instrument class 3: 0.34
Accuracy for instrument class 4: nan
Accuracy for instrument class 5: nan
Accuracy for instrument class 6: nan
Accuracy for instrument class 7: nan
```

(a) Accuracies for googleNet



(b) Random Frame

Fig. 9: googleNet Results

Fig. 10: Random Frame (VGG)

Fig. 11: Random Frame (AlexNet)

Fig. 12: Random Frame (PolyNet)

X. Conclusion

Lenet is giving more accuracy out of all models and the dataset is not balanced as it is giving zero accuracy for a few of the classes in both methodologies, even if we do resampling and data augmentation there is only a small change in the accuracies of the instrument classes. Maybe since we have only 3 videos to train and test the models the data of few instrument classes is not sufficient for the model to catch up with the complex features and predict them these videos in common contains more instruments of class 1 and 3 so after training the model and testing it is giving accuracies for these classes and 0 for few instrument classes, Maybe we can try to

```
Test loss: 3.528920888900757
Test accuracy: 94.45685958862
```

(a) Accuracy for LeNet

	precision	recall	f1-score	support
0	0.50	0.90	0.67	846
1	0.00	0.00	0.00	40
2	0.45	0.82	0.56	536
3	0.00	0.00	0.00	15
4	0.00	0.00	0.00	19
5	0.00	0.00	0.00	27
6	0.25	0.36	0.10	80

(b) Classification Report for LeNet

```
Test loss: 8.14467834
Test accuracy: 82.6934445
```

(c) Accuracy for ZFNet

Fig. 13: Performance Results

	precision	recall	f1-score	support
0	0.45	0.88	0.57	846
1	0.00	0.00	0.00	40
2	0.32	0.76	0.48	536
3	0.00	0.00	0.00	15
4	0.00	0.00	0.00	19
5	0.00	0.00	0.00	27
6	0.23	0.31	0.12	80

Fig. 14: accuracy for zfnet

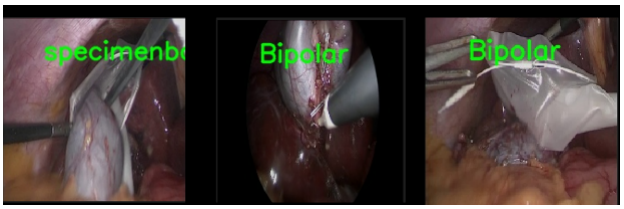


Fig. 15: zfnet

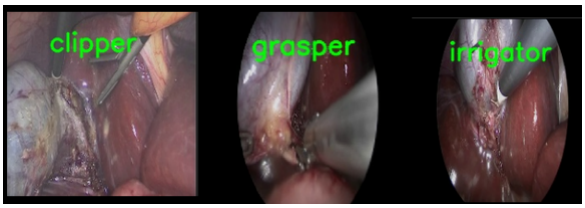


Fig. 16: lenet

	precision	recall	f1-score	support
0	0.45	0.88	0.57	846
1	0.00	0.00	0.00	40
2	0.32	0.76	0.48	536
3	0.00	0.00	0.00	15
4	0.00	0.00	0.00	19
5	0.00	0.00	0.00	27
6	0.23	0.31	0.12	80

Fig. 17: accuracy for zfnet

convert the videos to frames manually and try to do resampling manually to minority classes which may help in giving stable and consistent accuracies for all the instrument classes.

Using a train-test split to split the data and use gave more accuracy than giving the training data and testing data separately. Even when we see the support in the classification report we can see that there are very few instances of other instrument classes except 1 and 3 while training so they are either getting very less accuracies or zero maybe we can try increasing the size of the dataset to get good accuracies.

REFERENCES

[1] cnn1
[2] cnn2
[3] dataset
[4] cnn3