Wildfire Spread Prediction Model

APS360

Jiaxin (Audrey) Tian, Sicheng (Harry) Zhou,
Vanessa Lu, Yizhou (Linda) Cai

Table of Contents

1

Problem

What we are trying to solve and why it is important

Figure 1: 2023 Central Canada Fire

Identifying the Problem

- **Problem -** Wildfires cause immense setbacks to societies, leading to human safety risks and economic losses.

Wildfire Spread Prediction

Purpose

- Give early warnings
- Safe evacuations
- Efficient resource allocation
- Effective firefighting

Approach

Use **deep learning** models to analyze complex environmental data, and forecast fire spread patterns

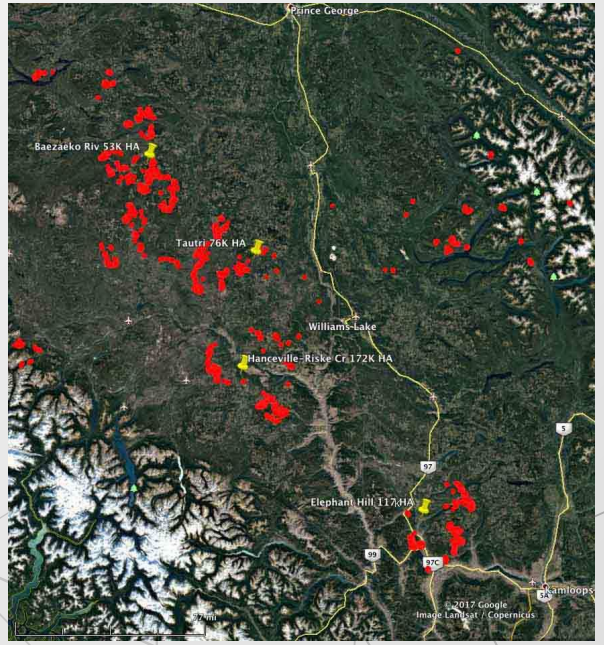


Figure 2. Wildfire shown in satellite

2

Data Processing

What data we used and the methods and justification for the cleaning and splitting of the data

About the Data

- **Next Day Wildfire Spread dataset**, remote-sensing images (TFRecord format) of past US wildfires [Figure 1]
    - Divided into **sets of 13 images**:
        - 1st-11th ⟶ environmental factor images:
            - wind speed
            - min/max temperatures
            - precipitation
            - etc
        - 12-13th⟶ previous and current fire masks
        - **Label ⟶ current fire masks**
- 7 of 15 training data ⟶ training set
- 1st of 2 available validation data ⟶ split ⟶ validation set & test set
- 2nd of 2 testing data ⟶ final testing set

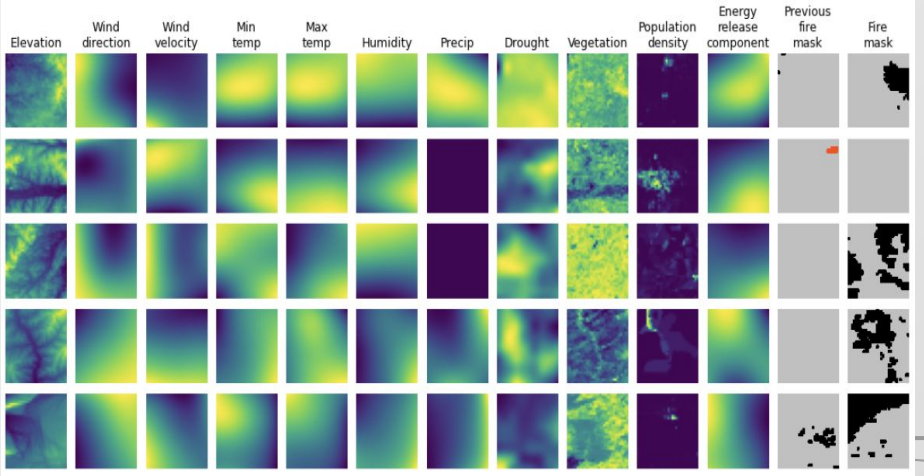


Figure 3: Sample of 5 sets of training images

Data Processing – Purpose

- **Load and save** the TFRecord data to Alexnet input shape
- **Remove** uncertain data with gray pixels
  - Gray ⟶ inaccurate fire masks due to environmental factors, such as cloud coverage

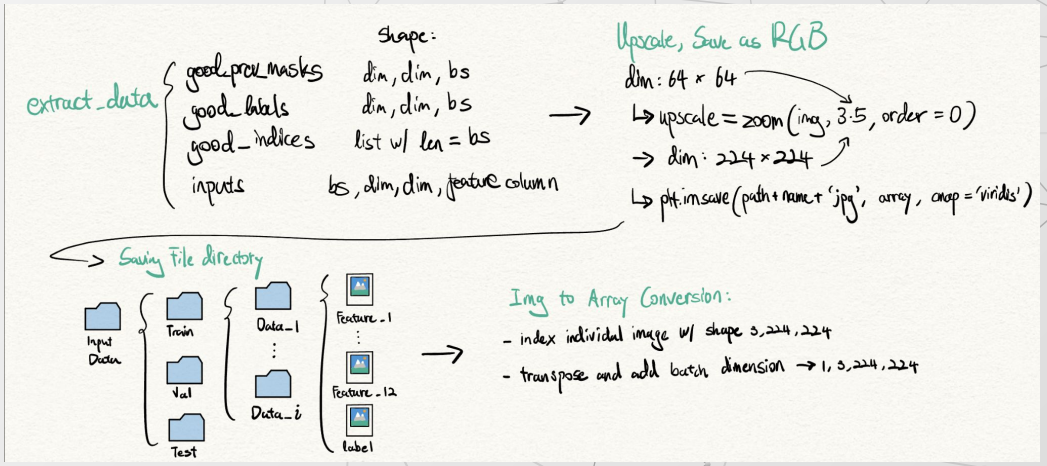


Figure 4: Flow of data through loading and cleaning processes

Data Processing – TFRecord to Array

- **Extract** 12 inputs and a label per data from tfrecord
- **Concatenated** the data into numpy arrays of **# of data x width x height x 13**
- **Results** [Figure 4]:
  - training array = 483 sets ⟶ 483x13 = **6279 images for training**
  - validation array = 73 sets ⟶ 73x13 = **949 images for validation**
  - test array = 67 sets ⟶ 67x13 = **871 images for testing**
  - **train/val/test = 77%/12%/11% split**

```
#@title Forming np array of all training data
train_arr1 = add_to_array(train_00_path)
train_arr2 = add_to_array(train_01_path)
train_arr3 = add_to_array(train_02_path)
train_arr4 = add_to_array(train_03_path)
train_arr5 = add_to_array(train_04_path)
train_arr6 = add_to_array(train_05_path)
train_arr7 = add_to_array(train_06_path)
whole_train_arr = np.concatenate((train_arr1, train_arr2), axis=0)
whole_train_arr = np.concatenate((whole_train_arr, train_arr3), axis=0)
whole_train_arr = np.concatenate((whole_train_arr, train_arr4), axis=0)
whole_train_arr = np.concatenate((whole_train_arr, train_arr5), axis=0)
whole_train_arr = np.concatenate((whole_train_arr, train_arr6), axis=0)
whole_train_arr = np.concatenate((whole_train_arr, train_arr7), axis=0)
print(whole_train_arr.shape)

#forming np array of val data
val_arr = add_to_array(val_00_path)
print(val_arr.shape)

#forming np array of test data
test_arr = add_to_array(test_02_path)
print(test_arr.shape)
```

```
(483, 64, 64, 13)
(73, 64, 64, 13)
(67, 64, 64, 13)
```

Figure 5: Shapes of each data array

Data Processing – Upscaling & RGB

a. **Upscaled 64 x 64** to **224 x 224**
b. **Grayscale** convert to **RGB** via matplot ‘viridis’, & custom colour map
c. **Result:** Final shape = **224x224x3** [Figure 2]

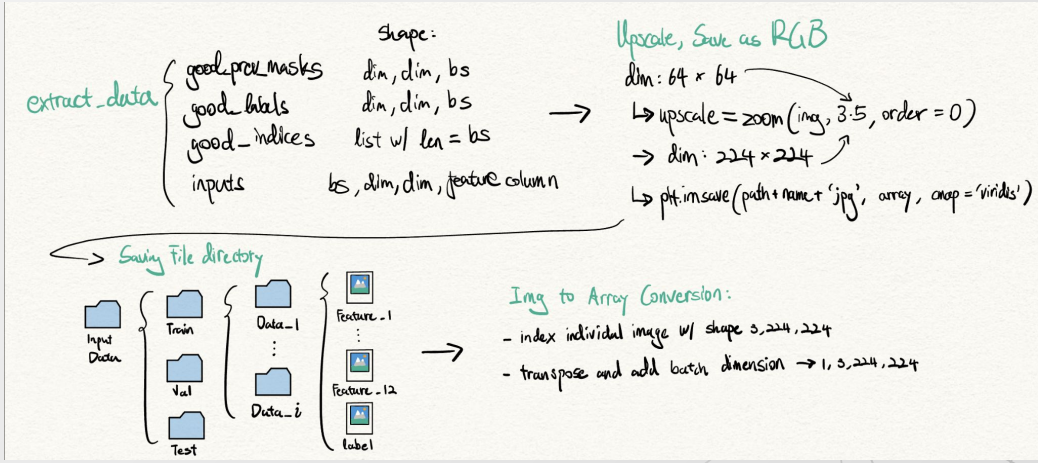


Figure 6: Flow of data through loading and cleaning processes

Data Processing

3. **Cleaning Data with Gray Pixels:**

   a. Located data in either the previous or current fire masks that have gray pixels with RGB value = **[192, 192, 192]**:
      i. 90 of 483 training data = gray
      ii. 6 of 73 validation data = gray
      iii. 5 of 67 testing data = gray
   b. Through os, removed the datasets with corresponding numbers in the google drive directly
      i. **Result:** 393 training, 67 validation, and 61 testing sets of cleaned data
         1. **75% / 13% / 12% = final train/val/test split**

```
for data, label in data_loader:
  if ([192, 192, 192] in data) or ([192, 192, 192] in label):
    data_grey = np.count_nonzero(data==192)
    label_grey = np.count_nonzero(label==192)
    if (data_grey>50) or (label_grey>50):
      print("File", count, "data_grey:", data_grey, "label_grey:", label_grey)
      grey_direct = folder_path + '/' + set_name + '/data_' + str(count)
```

```
for root, dirs, files in os.walk(grey_direct):
# For each file in the directory
  for file in files:
    # Construct the full path to the file
    file_path = os.path.join(root, file)
    # Delete the file
    os.remove(file_path)
    # For each subdirectory in the directory
  for dir in dirs:
    # Construct the full path to the subdirectory
    dir_path = os.path.join(root, dir)
    # Delete the subdirectory
    os.rmdir(dir_path)
# Delete the top-level directory
os.rmdir(grey_direct)
```

Figure 7,8 : Partial code from our function to clean gray data

3

Models

Our Neural Networks

Baseline Model

A self-coded mathematical model

a. **avg_red()** calculates average area fire from **previous fire mask**
b. **pred_correctness()** computes fire size of **previous fire mask** & **label**
c. Compare the fire size in current fire with a threshold pixel value
    i. threshold < previous, fire shrink
    ii. threshold = previous, fire maintain
    iii. threshold> previous, fire grow
d. If result from b) and c) match, prediction by the model is correct and returns True; otherwise, returns False
e. Percentage correctness based on the baseline predictions is used to evaluate its accuracies

```
def avg_red(dataset):
  red = 0
  count = 0
  for data, label in dataset:
    red += np.count_nonzero(data == 255)
    count += 1

  return red/count

def pred_correct(data, label, average):
  red_data = np.count_nonzero(data == 255)
  red_label = np.count_nonzero(label == 255)
  if (red_data < average) and (red_label < red_data):
      return True
  elif (red_data > average) and (red_label > red_data):
      return True
  elif (red_data == average) and (red_label == red_data):
      return True
  else:
      return False
```

Figure 9: Baseline Model

Structure of our Primary Model

Transfer Learning

- AlexNet as Encoder

Neural Network

- RNN → Decoder

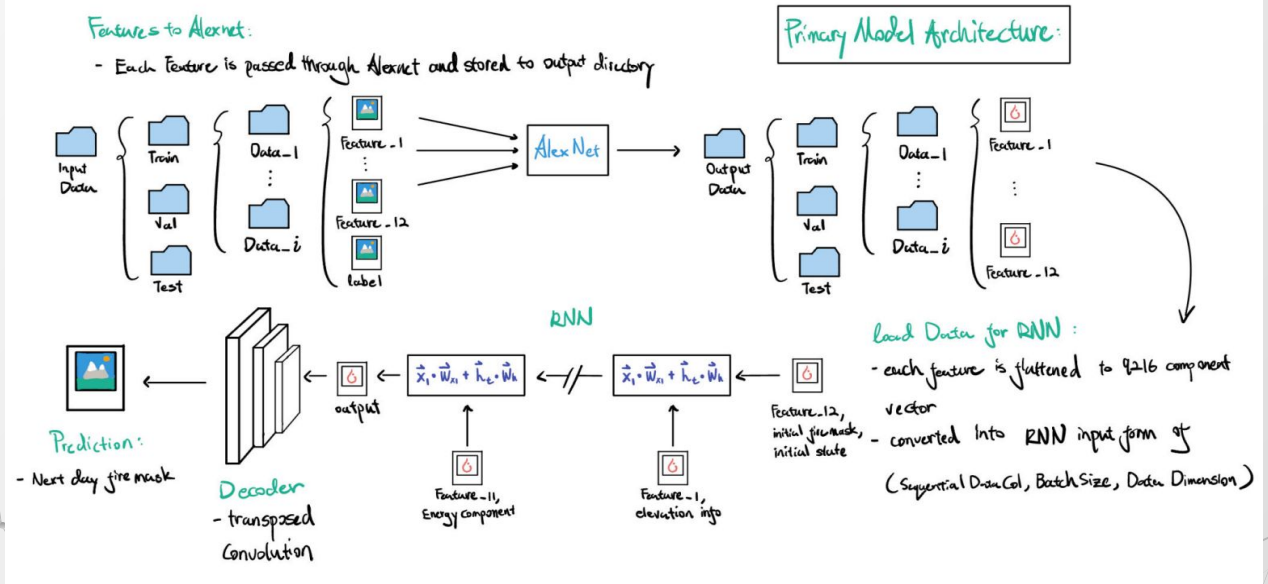


Figure 10: Illustration of our model structure

Transfer Learning

- pre-trained AlexNet neural network encodes input image data

Saving AlexNet features for each image to “Output” folder

1. Iterated through “Input” folder to access each set of 13 processed images
2. Converted the image **arrays to tensors** and passed them into the AlexNet to **obtain features**, which were then **saved in the appropriate file location** (same as when processing input data)
3. Further **separated label** from other features by directly saving it as a **3x224x224 tensor** instead of passing it through transfer learning
4. **Result:** Tensors are now processed through both data cleaning and transfer learning, and can be used to train our neural network

Neural Network

RNN

- $11 \times 9216 + 9216 \rightarrow \text{RNN} \rightarrow 9216$
- 1st-11th inputs describes geographic features
- 1th feature shows previous fire masks

Decoder

- $9216 \rightarrow \text{reshape} \rightarrow 256 \times 6 \times 6 \rightarrow \text{decoder} \rightarrow 3 \times 224 \times 224$
- Decode via 5 layer transposed convolution

```
class FirePrediction(nn.Module):

    def __init__(self):
        self.name = 'FirePrediction'
        super(FirePrediction, self).__init__()
        self.rnn = nn.GRU(9216, 9216, 1) #512, 256 smaller hidden size
        #self.rnn = nn.LSTM(input_size)
        kernel_size = 4
        stride = 2
        padding = 1
        out_padding = 0
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(256, 182, 3, stride, padding, out_padding),
            nn.ConvTranspose2d(182, 128, 6, stride, padding, out_padding),
            nn.ConvTranspose2d(128, 64, 8, stride, padding, out_padding),
            nn.ConvTranspose2d(64, 32, 9, stride, padding, out_padding),
            nn.ConvTranspose2d(32, 3, 10, stride, padding, out_padding),
        )

    def forward(self, x):
        x = np.array(x)

        if len(x.shape) ==  2:
          print(True)
          x = np.expand_dims(x, axis=1)

        rnn_input = torch.from_numpy(x[0:10, :, :])
        h0 = x[11,:,:]
        h0 = torch.from_numpy(h0[np.newaxis, ...])

        out, h = self.rnn(rnn_input, h0)

        output_array = []

        for i in range(h.shape[1]):
            decoder_input = h[:,i,:]
            decoder_input = decoder_input[0,:]
            decoder_input = torch.reshape(decoder_input, (256, 6, 6))
            decoder_input = decoder_input[np.newaxis, ...]
            output = self.decoder(decoder_input)
            output = F.relu(output)

            if i == 0:
                output_array = output
            else:
                output_array = torch.cat((output_array, output), 0)

        return output_array
```

Figure 11. Our Neural Network

Training

- Self written functions for data loader
  - Drop the last few data for training to avoid “index out of range” error
  - Concat individual data to batch
- Criterion via MSE
- ADAM Optimizer
- Custom accuracy function

```
def train(model, train_loader, valid_loader, batch_size=90, num_epochs=5, learning_rate=1e-4,
    """ Training loop. You should update this."""
    torch.manual_seed(42)
    criterion = nn.MSELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate, weight_decay=1e-4)
```

```
def get_accuracy(model, data_loader, use_cuda=True):
    accuracies = []
    counter = 0

    for features, labels in tqdm(data_loader, desc='Accuracy', unit='batch'):
        recon = model(features)
        recon = torch.transpose(recon, 1, 2) #give 0 3 1 3
        recon = torch.transpose(recon, 2, 3) # give 0 2 3 1
        difference = np.array(recon) - np.array(label)
        squared_difference = np.square(difference)
        mse = np.mean(squared_difference)/(255**2)
        accuracies.append(mse)
        counter += 1

    sigma = 0
    for item in accuracies:
        sigma += item
        #print('sigma is', sigma, 'number of error calculated is', counter, 'the

    acc_sum = float(sigma)
    item_num = int(counter)

    number = (acc_sum, item_num)

    return number[1]/number[0]
```

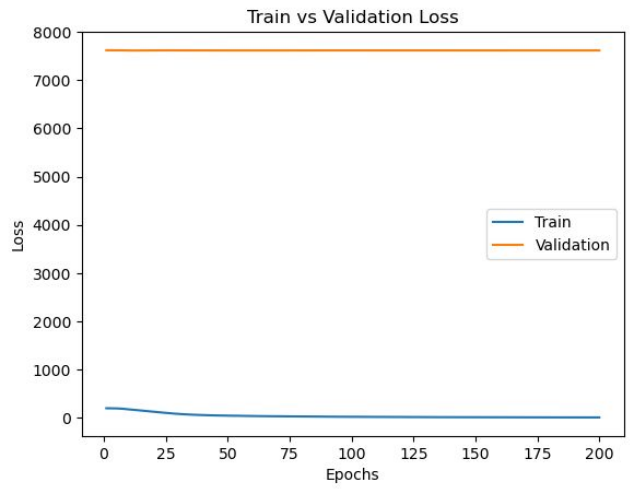
Figure 12, 13: Train and get_accuracy function code

4

Demonstration

Evaluating our model on new data

Trials of overfitting



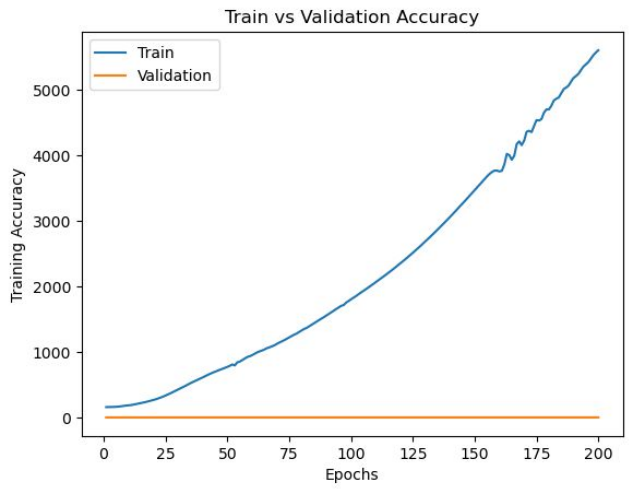
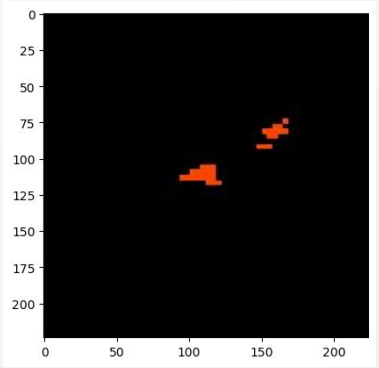
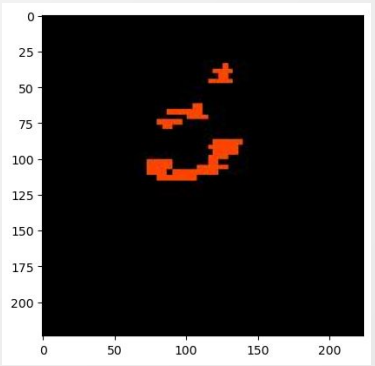
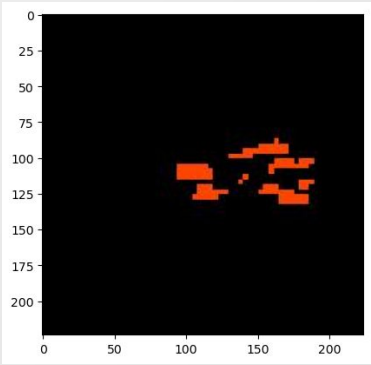


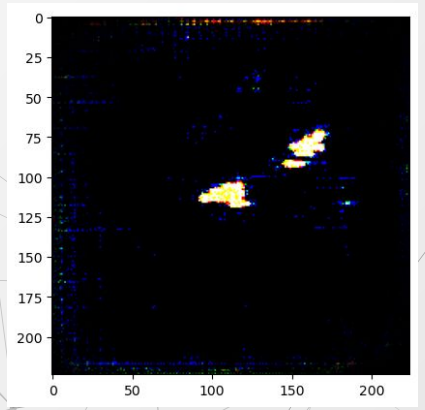
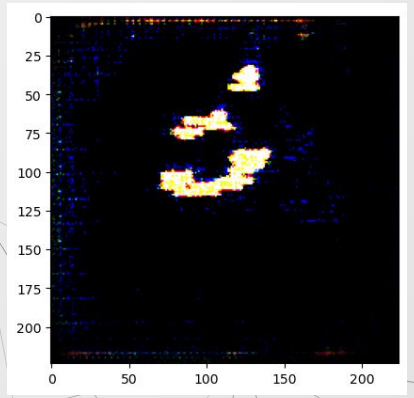
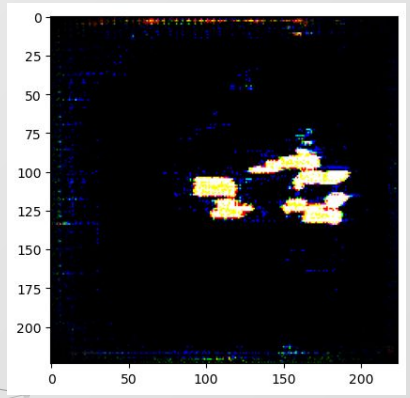
Figure 14,15: Performance graphs of overfitted data

- Model is overfitted with ten data
- Learning rate = 0.0001, batch size = 1, epochs = 200

Trials of overfitting

Figures 16, 17, 18:
Labels

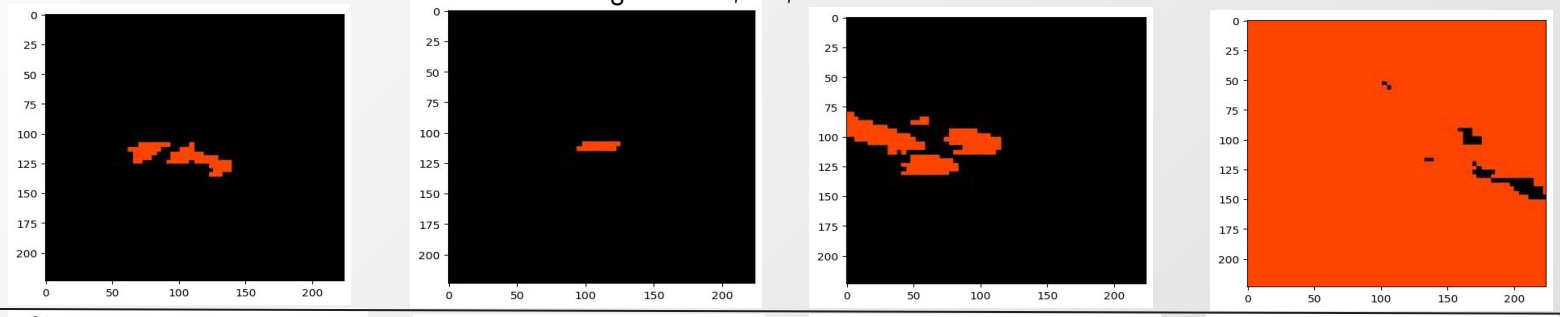




Figures 19, 20, 21:
Our predictions

Best results on unseen test dataset

Model: GRU, learning rate = 0.0001, batch size = 128, epochs = 300

Figures 22, 23, 24: **Labels**



Figures 25, 26, 27: **Trained model outputs on unseen test dataset**

5

Results

Quantitative and Qualitative

Baseline Accuracies

Training accuracy: 33.33%

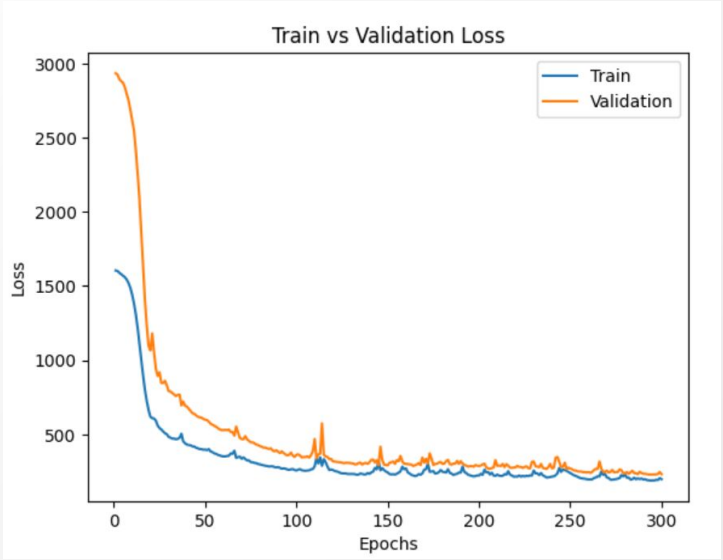
```
Number of correct predictions: 131
Number of incorrect predictions: 262
The training accuracy of our baseline model is 0.3333333333333333
```

Test accuracy on unseen data: 39.44%

```
Number of correct predictions: 28
Number of incorrect predictions: 43
The accuracy of our baseline model on an unseen test dataset is 0.39436619718309857
```

Figures 28, 29: Printed baseline results in Colab

Quantitative Results



Train Loss: 198.15288

Validation Loss: 231.68764

Train Accuracy: 257.20145

Validation Accuracy: 280.65799

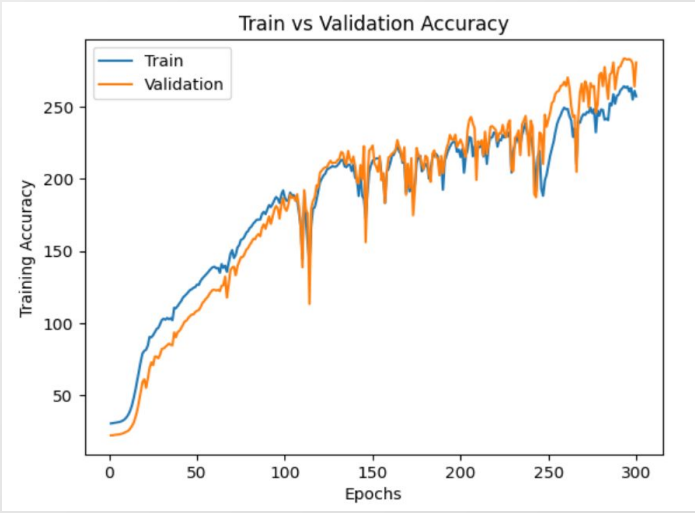
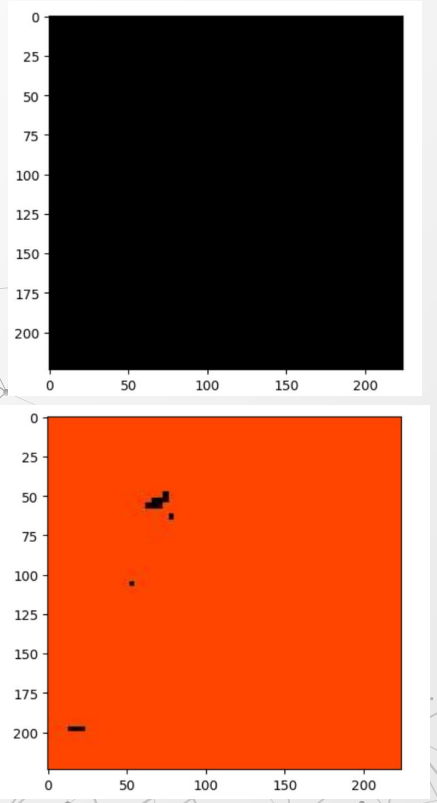
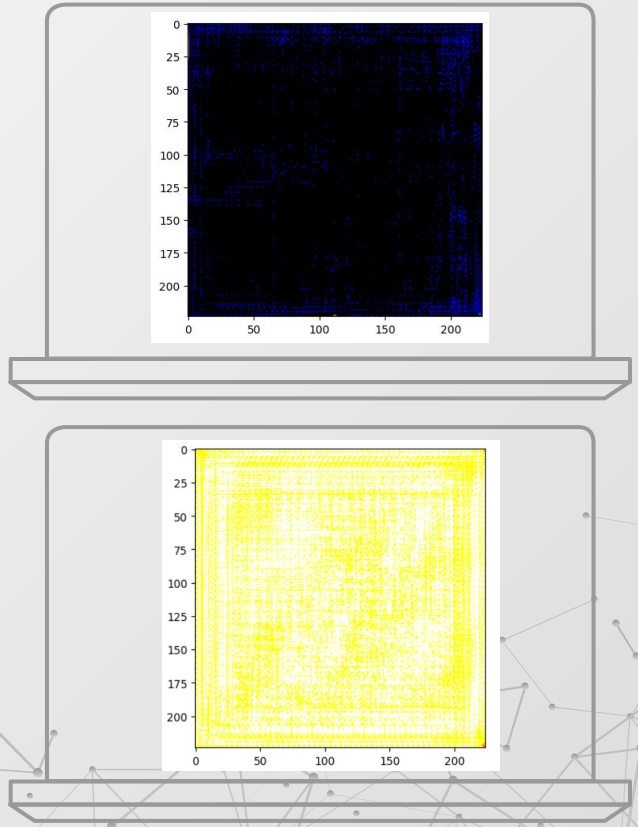


Figure 30, 31: Loss and accuracy of our model over 300 epochs

Qualitative Results

Figures 32, 33: **Labels**





Figures 34, 35: **Our Output on Test Data After Training**

Thank You