COMP 495 – Mentored Research
Everett Desern
Code is available on https://github.com/EverettDesern

Introduction

In summer 2019, I worked with Dr. Mayer-Patel on a research project he proposed. The idea behind the project was to use machine learning to predict motion compensation between video frames, which could be used to compress videos more efficiently. We both came up with the idea of using a convolutional neural network to accomplish this task.

Methods of Approach

Convolutional neural networks need ground truth training data in order for the network to learn and be able to actually produce good results. Generating ground truth motion estimation has been proved to be very difficult from many years of deep learning video compression research. Many people in this field of research use the famous "flying chairs dataset" which uses still images and superimposes chairs onto the images, which they can manipulate and move in order to generate ground truth data. For our research, we wanted to generate ground truth data for any video we wanted, so I created an algorithm that takes in a block and region size as parameters, and exhaustively searches the prior frame's search region to find a best "match". This match is determined by the mean-squared error of the two matrices.

Ground Truth Algorithm

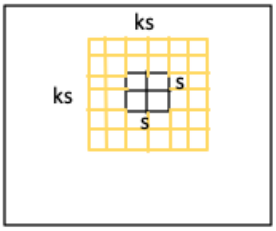
For my algorithm, I start by reading in the raw video data into several 2D arrays, each 2D array representing one video frame. The code prompts the user to enter the video stream, the height of the video, the width of the video, the block size, and the search size.

To find the region size, I multiply the user-inputted block size by the user-inputted search size. For example, if the block size was 2 and the search was 3, the overall search region will be 6x6. I then subtracted the search region minus the block size and divided by two. This gave me how many pixels needed to be on each side of the block. For this example, there would be two pixels on each side of the block.

Block Size (s) : 2
Search Size (k) : 3

Search Region = 2 * 3 = 6

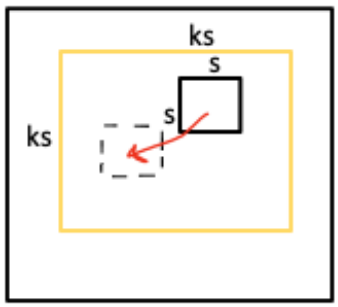
6 - 2 = 4
4 / 2 = 2 pixels on each side of the block



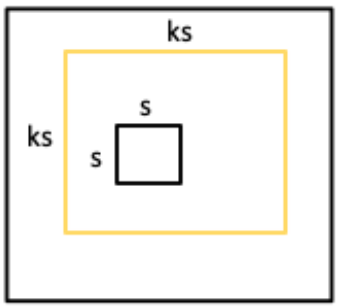
Current Frame

The way my search algorithm is set up is so that the X and Y values can only move a certain amount of distance. For the example above, the DX and DY values can be one of: (-2, -1, 0, 1, or 2). When creating a representation for the motion vector, I decided to create an array that easily displays the movement. In the example above, I created an array that holds 5 spaces for the X value (-2, -1, 0, 1, 2) and 5 for the Y value (-2, -1, 0, 1, 2). I also added a flag at the beginning of this array that denotes if there is a motion vector or not. For a block with no motion, the representation would be [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]. If the motion vector turns out to be (-2, 1), the representation would be [0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0]. My reasoning behind creating an array that only has 1s or 0s is that for neural networks, your training data needs to be 1 for correct values and 0 for anything else in order for the network to know right from wrong. It's important to also have a representation for a block with no motion vector instead of just saying "no motion", because neural networks have very specific inputs and outputs. All the parameters need to be the same size or else the network can't learn/work.

Next, my algorithm loops through every frame of the video and also loops through the height and width of the frame, incrementing by the block size. Once I have extracted a block from the current frame, my algorithm extracts the previous frame and exhaustively searches the parameters of the search region. I calculate the mean-squared error between each iteration, and the lowest one is the "match". I also have a MSE threshold, where if the minimum mean-squared error is above the threshold, I say that the block has no motion vector. If the value is below the threshold, then it is easy to extract the DX and DY of the motion vector by just subtracting the X and Y coordinates of the current frame's block and the prior frame's matched block.



Prior frame



Current frame

Once I iterate through every frame in the video, I write my training samples to a CSV file, where I can easily access the dataset for training. For each training sample, I attach the search regions of the current and prior frames and also the array displaying the DX and DY (or no motion). I decided to use the two search regions as inputs into my neural network and the array of DX and DY movements as the predicted outputs and ground truths that I later back

propagate on. I was considering to use a JSON file at first, but the DataLoader class that Pytorch uses to load training and validation datasets isn't compatible with a JSON file.

Convolutional Neural Network

For the first part of my research, I looked into several convolutional neural networks out in the world that already deal with motion estimation. One popular CNN for motion compensation is FlowNet. FlowNet provides the motion vectors between two video frames on a pixel by pixel basis, instead of a block basis that me and Dr. Mayer-Patel wanted. After trying to edit the code and architecture of the FlowNet model, I decided that I would just build my own neural network that is variable on the block and search region size.

A lot of the problems I ran into while building this neural network was how to design the architecture, how to load my CSV file into Pytorch that I could use to easily extract my training and validation sets, and also bugs and errors while training my CNN.

Having no machine learning background before this project, building the network seemed like a daunting task! I looked at a lot of tutorials online on how to build a simple neural network, and I also looked at popular CNNs, like the MNIST dataset for predicting numbers. I ended up slowly building my neural network with inspiration from the MNIST neural networks. Instead of having 10 outputs like the MNIST dataset (0, 1, 2, 3, 4, 5, 6, 7, 8, and 9), I can easily change how many outputs I need based on my block and search region size. For the above example with a block size of 2 and a search of 3, I have 11 possible outputs. For a bigger example of a block size of 3 and a search of 5, I have 27 possible outputs.

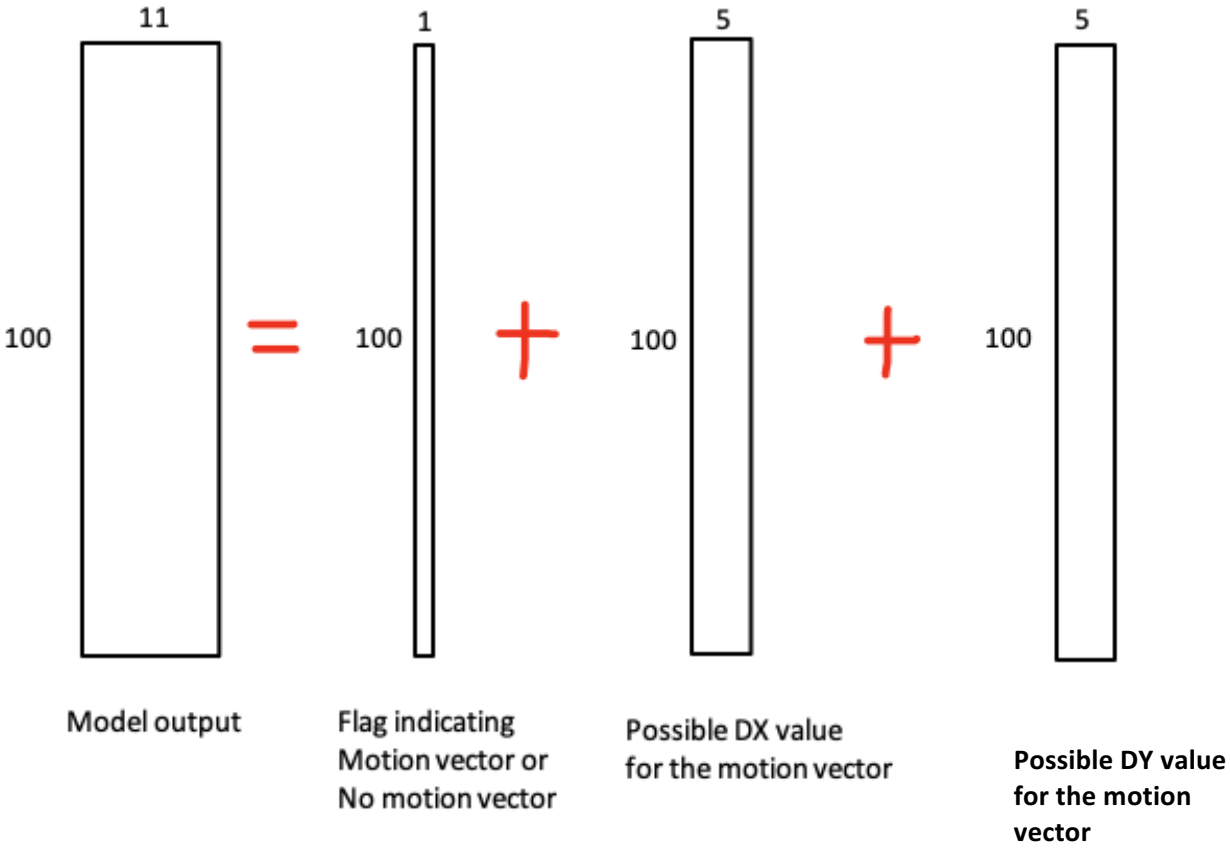
After researching how to actually load my data into my network, I came across the DataLoader class that is in the documents of Pytorch. DataLoader is a class that easily extracts data from a file of images and labels, and it comes with functions such as length and get_Item. Trying to figure out which file format is compatible with DataLoader, how to sort my data, and how to manipulate the length and get_item functions in order to use this class with my CSV file provided a great challenge!

After figuring out how to load the data, I was finally ready to write the code to start training the network! My training class first specifies the number of epochs, batch size, and the learning rate I wanted to use in my network. It took a lot of work trying to figure out optimal sizes for each of these parameters, but I ended up using 5 epochs, a batch size of 100, and a learning rate of 0.00001. Next, I split the indices of my data randomly, with 80% of it going to the training set and the remaining 20% going to the validation set. I then use DataLoader to extract the training and validation sets, and I also specify my network to use the ADAM optimizer, which is an optimization algorithm used to train the neural network.

Next in my training class, I begin to loop through the epochs and batches. In the loops, I start off by inputting my two search regions through the network and getting predicted

outputs. Since my batch size is 100, the matrices that return from the network are [100]x[11] (in the example with a block size of 2 and a search of 3.)

Neural Network Output



My reasoning for splitting the output into three different matrices is that if my network correctly predicts that there is no motion vector, I don't want the network to penalize getting the DX and DY values wrong, because there is no DX and DY values for a block with no motion! I had a lot of trouble trying to format the loss function to correctly to take this into account, since I have never seen a CNN that has a flag indicating when to back propagate.

Next, I use CrossEntropyLoss as my criterion and penalize it only if there's a motion vector and the model predicts the motion vector incorrectly. I then set up a loop for the validation set, test the accuracy of the model, and graph the loss and accuracy of the model to test the network's efficiency.

Next Steps

I am very excited to continue working on this project during the school year! This project provided me a lot of challenge, and I learned a ton about machine learning.

This fall, I will try to get my neural network functioning more efficiently and with more accuracy by making small changes to the architecture and parameters. I am also in the process of getting more training data by downloading raw video data and running it through my ground truth algorithm. I am also in the process of manipulating still image data by making random transformations and using that as ground truth.