Final Report

Your final report should be a .pdf or .ipynb document that summarizes all work you did on this project from your initial proposal to your last experiments. You should push it to your GitHub repo with a filename of `report.pdf` or `report.ipynb`.

Your report should be self-contained. That means if you want to refer to a goal you wrote in your proposal or update, you should copy it here. If you want to reference a plot that shows your model's accuracy over time, include that plot in your report. If it's not in your report, it doesn't contribute to your grade. The important exception to this is that your written report should not contain your code. See "code and documentation" below for how we will grade your code.

1. Goals and discussion (16 points)

The broad purpose of this project was for you to learn how to apply deep learning methods by working towards goals you chose. Every project team had different goals, and those goals varied in terms of their ambitious and specificity. Thus, your grade is not determined by counting up the number of goals you achieved. It's possible to get full points without completing many goals, and completing all your goals does not guarantee you full points. You should approach this section as an exercise in persuasive writing; try to convince us that you spend a substantial amount of time pursuing these goals and that you learned some specific and interesting things.

For completed goals that involve dataset collection or preprocessing, please include at least one example of what the output of that preprocessing was. For completed goals that involve experiments, please include at least one plot or table showing an experimental comparison.

This section will be graded holistically; you can still receive full points even if you abandoned all your stretch goals, for example if you went above and beyond on your desired goals.

Note: For this section, instead of including a 1.4 Section, we have incorporated everything we want to into the different "Goals"

1.1 Essential goals

List all essential goals that you proposed either in your second proposal or project update. For each, (a) include one or two sentences on how you completed it, what progress you made, or why you abandoned it, and (b) include one to two sentences on what was the most interesting or difficult part.

1. **Test our new model with increaed training data size using new data loader**

   Through our implementation of the torch dataloader with multiple workers (which allows us to go from 10k to the full dataset of 280K images with increased parallelism), we have successfully loaded in 280k training images with batch size of 128 from AffectNet and 15,339 images from RAF-DB using batch size of 32. This was later scaled down to 100,000 images in the interest of training and experimentaion time. The most difficult part was undoubtedly finding a platform that that is able to load in all the data + model that is able to run multiple tests without CUDA running out of memory, which in our case we have to shift from local machines to GCP(Google Cloud Computing) in order to run our models since we are using images as our dataset (each image being around 4kB). After that problem was solved, we also had to increase the base model of the GPU in our virtual machines to use the NVIDIA Ada A100 GPUs to fit our data + model during training and testing.

   Image Examples:

   128 Images Batch Example Tensor:

```
tensor([[[[0.7020, 0.7020, 0.7020,  ..., 0.7137, 0.7176, 0.7098],
          [0.7020, 0.7020, 0.7020,  ..., 0.7098, 0.7137, 0.7059],
          [0.7020, 0.7059, 0.7098,  ..., 0.7137, 0.7137, 0.7059],
          ...,
          [0.5059, 0.5020, 0.4941,  ..., 0.5059, 0.5529, 0.5922],
          [0.4863, 0.4980, 0.5059,  ..., 0.4784, 0.5373, 0.5765],
          [0.4863, 0.5098, 0.5255,  ..., 0.4627, 0.5255, 0.5647]],

         [[0.5569, 0.5569, 0.5529,  ..., 0.5804, 0.5843, 0.5765],
          [0.5569, 0.5569, 0.5529,  ..., 0.5765, 0.5804, 0.5725],
          [0.5529, 0.5569, 0.5490,  ..., 0.5804, 0.5804, 0.5725],
          ...,
          [0.4118, 0.4078, 0.4078,  ..., 0.3922, 0.4392, 0.4784],
          [0.3922, 0.4039, 0.4196,  ..., 0.3647, 0.4157, 0.4549],
          [0.3922, 0.4157, 0.4392,  ..., 0.3490, 0.4039, 0.4431]],

         [[0.4863, 0.4863, 0.4784,  ..., 0.5333, 0.5451, 0.5373],
          [0.4863, 0.4824, 0.4784,  ..., 0.5294, 0.5412, 0.5333],
          [0.4784, 0.4824, 0.4784,  ..., 0.5333, 0.5412, 0.5333],
          ...,
          [0.3725, 0.3686, 0.3647,  ..., 0.3608, 0.4157, 0.4471],
          [0.3529, 0.3647, 0.3765,  ..., 0.3412, 0.4039, 0.4353],
          [0.3529, 0.3765, 0.3961,  ..., 0.3333, 0.3922, 0.4314]]],
```

   On top of implementing the dataloader, we also attempted to speed up the data fetching process (which allows the dataloader to index the image dataset/folder to get the corresponding data) by indexing the files from 5 different locations, allowing us to

load in the data faster. This is because one of the biggest bottlenecks for us was the data IO process, which took around 20 ~ 30 minutes to load in 280k images, and this implementation solved that problem directly. This change helped us immensely when we were transitioning from our local machines to GCP, since we were able to load in the data faster and run more tests. When we were on our GCP VMs, we also specified SSD disks to ensure that the data was loaded in faster.

After optimizing the pipeline with the condensor and pkl files, this is the final data pipline...

Dataloader Pipeline:

Pkl Files -> Dataloader -> Model

1. **Conduct basic hyperparameter tuning to improve accuracy (we already have some of this setup from previous model)**

   We have sucessfully tuned/determiend our hyperparameters through discussing the model setup, the number of layers, the structure of the neural network, and extensive online research to ensure that our base model makes sense. From the basic 64x64 models in the PyTorch documnetation, to following online articles and discovering the basic pattern in general image/emotion recognition (expanding layer size as the model gets deeper), we finally settled on the baseline parameters and models we were going to use. The most interesting aspect of working towards this goal is the team going through multiple articles a day in the week of Feburary 13th and Feburary 20th to find out how to improve our base model through basic hyperparamter tuning, then arriving on the result just before turning in the update file, giving us a solid foundation to work from for latter steps.

   The process behind selecting this model was an extremely time consuming process and required a lot of research and discussion. We started with the basic model from the PyTorch documentation, then we expanded the model to include more layers and more filters. However, our first interpretation of the model given in the paper "Deep learning for real-time robust facial expression recognition on a smartphone" was incorrect, and the accuracy was about 15%, which is slightly better than random guessing 1 of the 8 emotion categories (12.5% chance of getting it right). We then realized that the model was incorrect, and one of our teammates received help from Professor Wood-Doughty in order to adjust the model such that it is representative of what we saw in the paper.

   Since then, through researching and discussion after the update was turned in, we realized that perhaps our current model was not suitable and needed to be changed to make our CNN better. Some of the articles we referenced are listed here:

   - Different Types of CNN Architectures

- Best deep CNN architectures and their principles: from AlexNet to EfficientNet

Thus, the "new base" archeitecture was created, as we added more layers, modified the linear layers, and gradually scaling the convolution layers' size of our model. Unfortuantely, due to local machie CUDA memory size, our model with increasing complexity would cause CUDA to run out of memory, so we were constrained by our hardware and had to move to GCP. Of course, that is not our final model as we still have hyperparameters to tune, which will be mentioned in Section 1.2.1.

Model Label	Model Details	Training Accuracy	Validation Accuracy
Basic 1	self.conv1 = nn.Conv2d(3, 6, 5) self.pool1 = nn.MaxPool2d(3, 3) self.conv2 = nn.Conv2d(6, 16, 5) self.pool2 = nn.MaxPool2d(3, 3) self.fc1 = nn.Linear(8464, 120) #44944 self.fc2 = nn.Linear(120, 84) self.fc3 = nn.Linear(84, 8) # 8 classes of emotions	N/A	N/A
Basic 2	self.network = nn.Sequential(nn.Conv2d(3, 64, 5), nn.ReLU(), nn.MaxPool2d(2, 2), nn.Conv2d(64, 64, 5), nn.ReLU(), nn.MaxPool2d(2, 2), nn.Conv2d(64, 64, 5), nn.ReLU(), nn.MaxPool2d(2, 2), nn.Dropout(.5), # dropout added later nn.Conv2d(64, 64, 5), nn.ReLU(), nn.MaxPool2d(2, 2), nn.Dropout(.5), #dropout added later nn.Flatten(), nn.Linear(6400, 512), # (when one linear layer this was (6400,8) nn.Linear(512,8) # added this later)	0.6	0.15
Mod 1	Basic, loss changed from avg -> agg (40 epochs w/ full affectnet dataset)	0.8	0.28
Mod 2	Mod1, extra liaenr layer (20 epochs)	0.66	0.32
Mod 3	Basic + 0.5 Dropout	0.63	0.28
New 1	self.network = nn.Sequential(# decreased sizes from paper because not enough memory in gpu nn.Conv2d(3, 64, 5), nn.ReLU(), nn.Conv2d(64, 128, 5), nn.ReLU(), nn.MaxPool2d(3, 3),	0.884	0.43

Model Label	Model Details	Training Accuracy	Validation Accuracy
	nn.Dropout(.1), nn.Conv2d(128, 192, 5), nn.ReLU(), nn.Conv2d(192, 192, 5), nn.ReLU(), nn.MaxPool2d(3, 3), nn.Dropout(.1), nn.Flatten(), nn.Linear(84672, 8),)		
Mod 4 (New Base)	New 1 + drop out=0.1 + momentum=0.9	0.82	0.47

1. **Start using RAF-DB for pretraining**

   We have successfully used RAF-DB for training, using all of the 15,399 images in batches of 32 to pre-train our model. In selecting the best pre-training model, we found the best model by evaluating the pre-train model's training accuracy on the RAF-DB dataset and the number of epochs trained using pre-training. In obtaining the model, the most interesting part was attempting to find the model that did not overfit on the RAF-DB dataset but still pre-trained enough such that it would make our training more accurate and converge quicker, so we ran 10+ trials in total on the dataset this past week and the week before this week to find the best model that had reasonable high accuracy but the least training epochs to avoid overfitting. It is important to note (more will be mentiond in the Reflections section), that the pre-training did not have too much of an effect on our final model's accuracy, since our AffectNet training dataset is 100,000 images, the latter training effectively overwrote the pre-training.

   It is important to note that although the pre-training did not have too much of an effect on our final model's accuracy, and we speculate the reason is because our model is complex enough to be able to learn the training dataset is with 100,000 images quickly, which is a big data set compared to our pre-training dataset is only 15,399 images, so the latter training effectively overwrote the pre-training. However, we did see a particular decrease in the starting accuracy when we used pre-training.

Number of Epochs For Pretraining	Validation Acc
Epochs = 8	55%
Epochs = 10	59%
Epochs = 15	61.54%
Epochs = 25	61.20%

Number of Epochs For Pretraining	Validation Acc
Epochs = 25	56%

1. **Train with different batch sizes to see correlation between accuracy and batch sizes**

   We have successfully trained our model with different dataset sizes and batch sizes to see the correlation between validation accuracy and dataset size, and validation accuracy and batch sizes. One of the issues with this is the same issues we faced when we are trying to finish Goal #1, where we ofund it was difficult attempting to fit all the data into our local machine's GPU, so we had to use GCP in order to make sure our testing can happen. The most difficult part about this is definitely the time aspect, as training each of our model takes upwards of an hour, making it very difficult to make efficient testing and forcing us to train concurrently whilst doing other work.

Batch Sizes	Validation Accuracies
16	0.3755
32	0.3664
64	0.3644
80	0.3659
128	0.3377

1.2 Desired goals

List all desired goals that you proposed either in your second proposal or project update. For each, (a) include one or two sentences on how you completed it, what progress you made, or why you abandoned it, and (b) include one to two sentences on what was the most interesting or difficult part.

1. **Conduct thorough hypertuning of paramters to further accuracy alongside trying different models and essembles to achieve best results provide a guide for how to reproduce our results and what factors (type of model, loss function, learning rates, optimization algorithm, etc.) affect the accuaracy of the model, with references below**

   During our thorough hyperparameter tunings on our "New Base" model, we tested the following hyperparameters: learning rate, batch size, optimizer, dropout rate, and momentum. In order to find the optimal parameters, we trained on our "new base" model mentiond in section 1.1.2. For all of the parameters, we ran them until the loss was converging or overfitting was starting to occur. We found the following data for

each of the hyperpatameters when we were tuning it (some may repeat from the section above for organization purposes).

As seen in the tests below, we have elected to choose the following parameters as our "optimal hyperparameters":

- Learning Rate: 0.0001
- Batch Size = 32
- Optimizer = Adam
- Dropout Rate = 0.65
- Momentum = 0.9

It should be mentioned here that given more time and fast computing power, we will be able to run more comprehensive tests faster, with more epochs, and perhaps find better parameters that we have not found yet. However, given the time, hardware, and money (for computing on GCP) constraints, these are the best parameters we are able to produce.

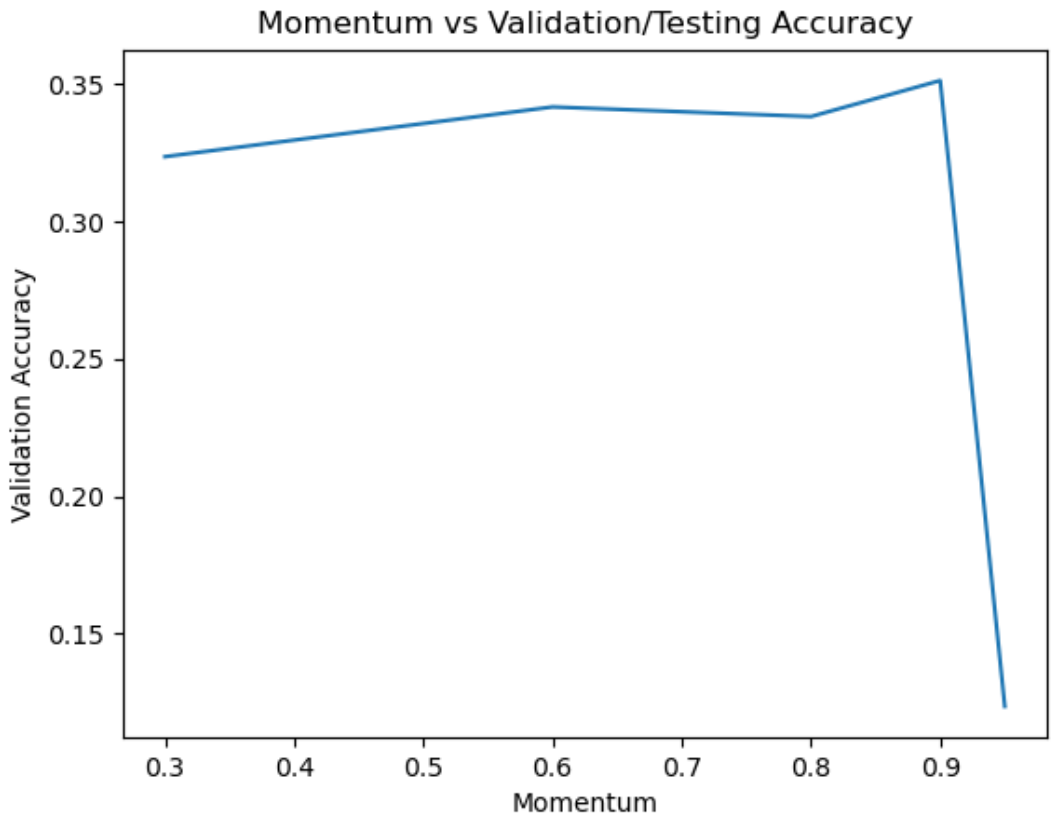
Learning Rates	Validation Accuracies
0.001	0.3945
0.0001	0.3780
0.00001	0.3560

Batch Sizes	Validation Accuracies
16	0.3755
32	0.3664
64	0.3644
80	0.3659
128	0.3377

Optimizers	Validation Accuracies
SGD	0.3458
Adam	0.3801

Dropout Rates	Validation Accuracies
0.0	0.42
0.1	0.41
0.2	0.41
0.35	0.42
0.5	0.42
0.65	0.42

Momentum	Validation Accuracies
0.3	0.3236
0.6	0.3417
0.8	0.3382
0.9	0.3513
0.95	0.1234



1. **Use google cloud computing to facilitate faster learning over full dataset (contigent on model performance, may not be necessary)**

   Given the size of our datasets (AffectNet~ 2.5 GB, RAF-DB ~ 1.5G), we eventually were forced to compute on GCP in order to have a GPU of sufficient size to train our model while being able to fit the model and the data. We were able to successfully accomplish this and have since then conducted all testing on GCP using the PyTorch Deep Learning VM Image.

   However, this process was not easy. Our group spent over 20+ hrs last week and this week to finish setting up and distributing the "data-image" and "boot-image" to ensure that it works on each of our accounts so we are able to have more GCP computing credits in total to allow multiple tests to be run at the same time. During these 20+ hours, we also had to switch some of our GPUs from the standard Nvidia T4 to Nvidia A100s to further increase the capacity of our GPUs. This entire process stands as a steep learning curve and is extremely time consuming, but we were able to successfully accomplish it and produce the results explained above.

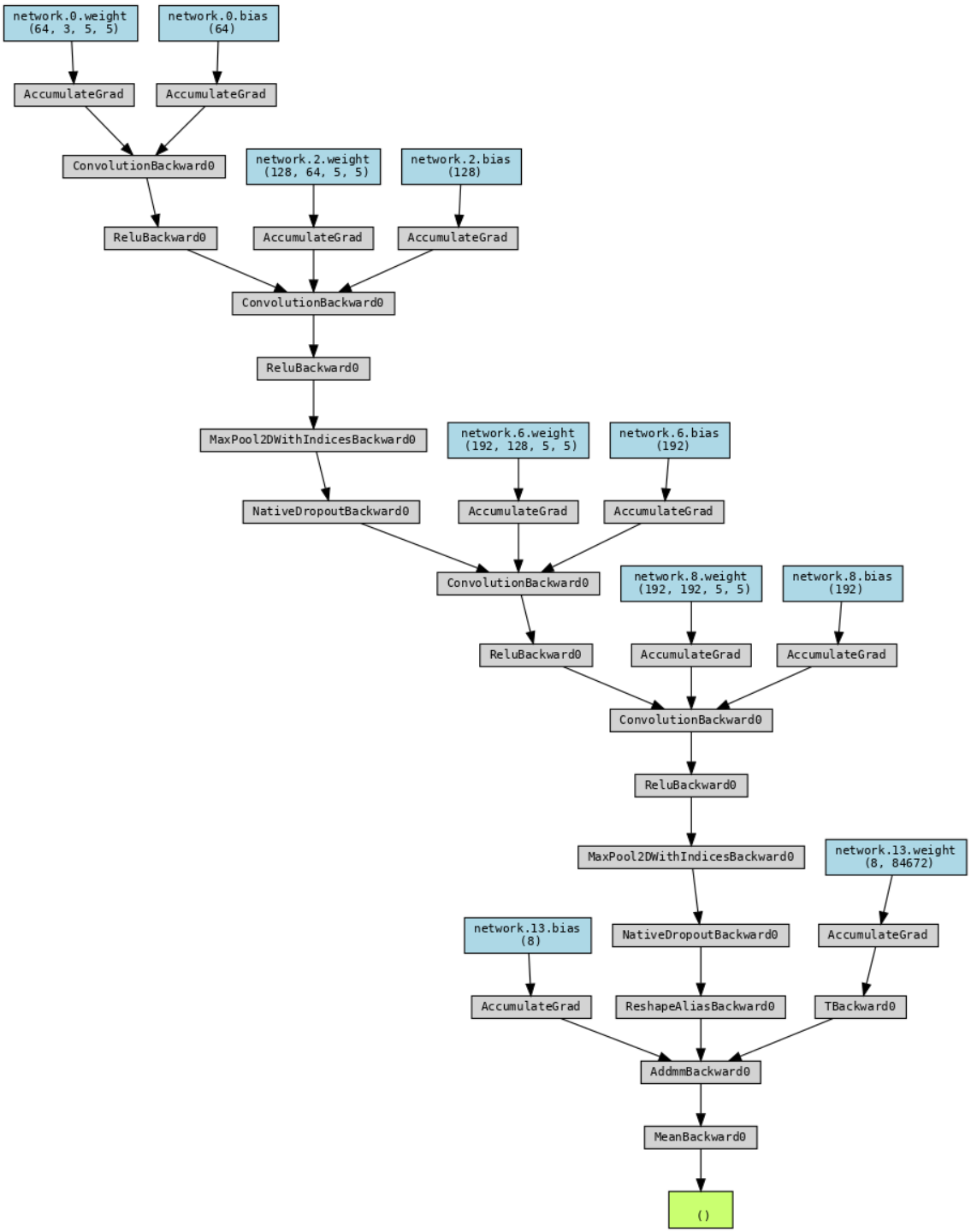
```
2.5G    449Project/train_files/train_set/images

1.5G    449Project/basic
```

1.3 Stretch goals

List all stretch goals that you proposed either in your second proposal or project update. For each, (a) include one or two sentences on how you completed it, what progress you made, or why you abandoned it, and (b) include one to two sentences on what was the most interesting or difficult part.

1. **Create visualizations of the CNN model similar to the reading we did on [CNNS]**

   Due to time constraints, we prioritized the model performance over the visualizations. Originall, our vision for the visualization is to have images be displayed at times throughout the training progress to showcase the actual model performing actions on the image, which would not only be veyr eye-catching by could give us a better sense as to how the model is surgically learning the images. However, we were able to create a simpler visualization of the model using the [torchviz] library. One of the biggest problems when we tackled this goal, or not for that matter, is time. Admittedly, more time could have been allocated in the beginning of the quarter when we started this project to provide buffer for us, but as the quarter progressed, it is evident that there was not enough time to do visualizations when our model wsa not on par with our expectations.



1. **Develop a ML pipeline that allows for our model to operate in real time on video footage, potentially developing into a zoom plugin/overlay to analyze facial expressions in real time**

   Simlar to the problem face in our first stretch goal, we were unable to create a model that can dissect videos into frames of images and process them. As a project, it was a little bit more ambitious then we have expected. Finding videos of people will not be difficult, nor is slicing them, but having to map out a box around the person's face, and

facing the potential of having to manually label some of them to ensure that we get a good set of data to represent our dataset, that means that we would have to have a lot of data to train on.

Granted with more time, and perhaps if we had a better model and it had sufficient accuracy earlier, we could have moved on to this goal. However, we were unable to do so this time.

1. **Use radically different datasets to perform pretraining, such as the google facial expression comparison dataset and models from different domains, such as langauge models, to see if we can achieve better results**

   Unlike stretch goal 1 and 2, we touched on this goal, as we used RAF-DB images to pre-train our model which is mostly based on AffectNet images. Upon eye inspection, it seems that the two sets of data did not have a lot of difference in their images, in terms of size, lighting, and position of the person's face. Now, when we pretrained our model on RAF-DB, we did see a slighly lower training loss in the beginning compared to no-pretraining, which indicates that the two datasets were quite similar, and thus does not really fulfill the goal of using radically different datasets to perform pretraining.

   Granted more time, we would have explored the "complex emotion" dataset from RAF-DB, which is a dataset that contains images of people with complex emotions, such as anger and disgust, which are not present in AffectNet. These will be radically different because of their complexity, however, we were unable to do so this time.

1.4 Everything else

Discuss the additional work you did as part of this project that was not covered by the goals you set for yourself. Try to organize this into "goals" that you could have set for yourself if you had known what needed to be done. For each of these goals, write one or two sentences on why you focused on it, and one or two sentences on what was the most interesting or difficult part. This can be something practical like "we couldn't load our data/model on Colab" or something conceptual like "we tried to understand the fancy model from this new paper but couldn't figure out how to implement it."

1. **Imporving Accuracy of Our Model**

   While this is a fairly standard process of any machine learning model procedure, we do want to highlight what we have done here as it wasn't explicitly mentioned in any of the goals above. First of all, there is the architecture evolution mentioned in Section 1.1.2, which examplifies the number of iterations we have went through to ensure our model is as optimal as possible. Then, then, there is pre-training. We figured that since our dataset is technically considered "small", as there is 120 GB dataset in the hands of the

University of Denver (an extension to our current AffectNet's dataset), using pre-training may help us to converge faster. In reality, whlie the initial loss was slightly lower due because of the pre-training, it did not help the model to converge faster in the long run as the pre-training dataset was fairly small at 15k images. Because of that, we have abandoned pre-training for our final model.

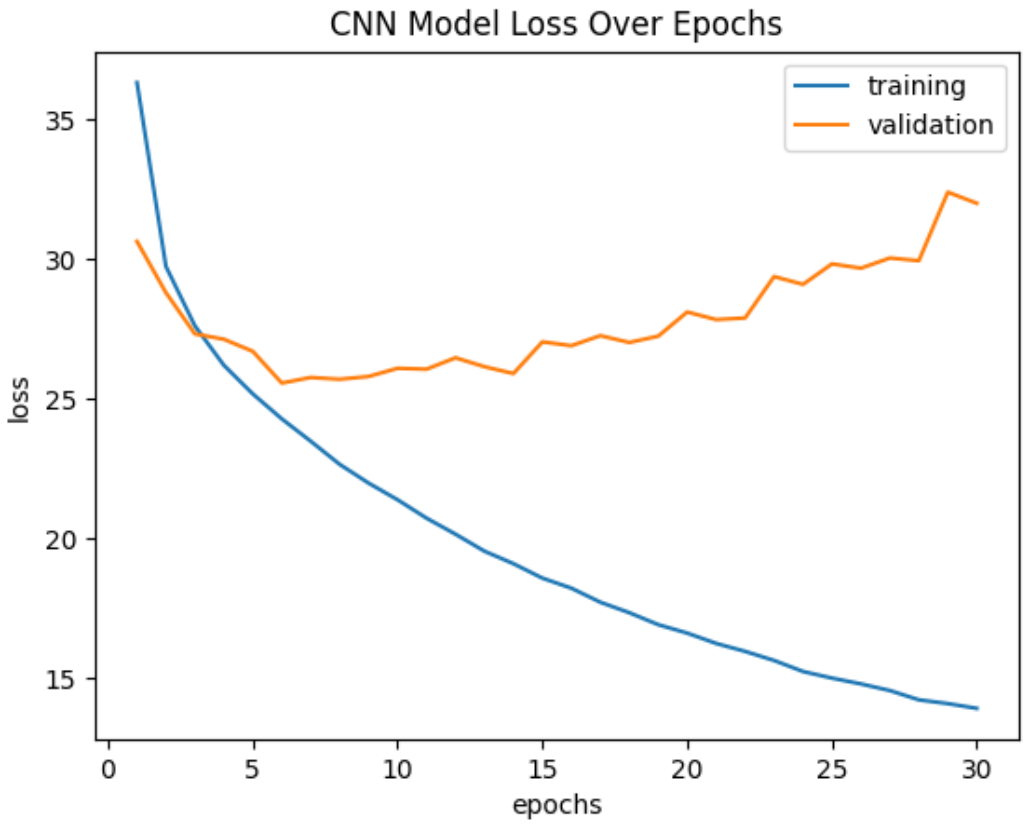
On top of these two changes, we have also tried to improve the accuracy of our model by using different hyperparameters and optimizers, as mentioned in Section 1.2.1, each of the training runs averaging about 2 hours of run time due to the size of our dataset, the complexity of our model, and the number of epochs it takes for our model to converge. For that, we have combined the most optimal hyperparameters shown in their respective tests and have used them for our final model. With these changes, it took all of the team to work together to ensure that we are able to run multiples models on GCP, coordinating well to record down the data, and ensure that the final model is indeed the optimal model.

Wrapping the section up, we will provide the final model's training, validation, and testing accuracy. We will also include our testing data's confusion matrix, which shows the accuracy of each class, and the most common misclassifications.

Testing Accuracy	Validation Accuracy	Training Accuracy
0.3990963855	0.3971774194	0.91685

- Row/Top-Axis (Predicted)
- Column/Left-Axis (Actual)

	Neutral	Happiness	Sadness	Surprise	Fear	Disgust	Anger	Contempt
Neutral	401	33	218	223	149	154	200	181
Happiness	58	461	44	104	45	87	40	281
Sadness	20	0	194	12	59	50	32	10
Surprise	6	0	3	126	85	8	9	2
Fear	0	0	5	18	122	13	5	1
Disgust	0	0	1	1	1	69	2	3
Anger	11	3	33	14	35	119	211	15
Contempt	1	0	0	0	0	0	0	6



1. **Hardware Constraints and Optimzations**

   We spent a significant amount of time working with hardware constraints and optimizations (~40hrs). Initially, we were only using 10,000 images for training, but this was obviously not enough. We then attempted to increase the size of the images loaded and implemented dataloaders through pytorch. However, we could not take advantage of the built in multiprocessing through the num_workers parameter. After several hours of digging, it turns out this was a windows and jupyter notebook issue. In addition, at the time our original 64x64 model was not working very well but we could not scale it because our GPU's physically could not fit a larger model. Hence, out of pure frustration, we went to GCP. It took about 10hrs to fully understand the ecosystem and get an instance up and running. We were then able to finally multiprocess our data loading but still found significant bottlenecks in the data loading. So to combat this, we increased the number of Vcpus on the machine to 8, increased memory to 52gb, moved all the data to an SSD with 3x loadspeeds, and condensed data loading by pre applying pytorch tensors then saving the data in batches (previously we were loading from 5 different files). Finally, we were able to fix the data loading bottleneck. The larger GPU's (going from our personal 6gb GPUS to 16Gb T4's and later even larger V100's) also allowed us to implement our new and larger model which gave us the largest boost in our validation accuracy.

2. Code and documentation (10 points)

While you should not include code in this report, all your code should be uploaded to your GitHub repository. In this part of your report, you should list all .py or .ipynb files that contain your code and a one or two sentence description of what that file contains. Then, in each of those files, you should make sure that there is enough documentation that we can read through the repository and understand what each part of the code does. This code does not necessarily have to be in immediately working order (e.g., if you cannot upload your data or model files to GitHub), but we should be able to read through your code and notebooks to understand how you implemented things. You absolutely do not need to include a comment for every line of code, but you should include docstrings for (most of) the functions you wrote and descriptions for the coding cells within any .ipynb files.

Files

1. ***main.ipynb*** - This is the main file that contains the code for the model. It contains the code for the model, the training, and the testing. It also contains the code for the pre-training, if we were to use it.
2. ***premodel.ipynb*** - This is the file that contains iterations of different models, architectures, and hyperparameters that we have tried to use for our model. It is not used in the final model, but it is used to test out different models and architectures. We believe this is contains a good history of what we have done and what we have tried.
3. ***pretrain.ipynb*** - This is the file that contains the code for the pre-training. It is not used in the final model, but it is used to test out the effect of using RAF-DB images to pre-train on our AffectNet data oriented model.
4. **_rafdb_import*guideline.ipynb*** - This is the file that contains the examplar code for importing rafdb images into our model. It is not used in the final model, but it is used to test out the effect of using RAF-DB images to pre-train on our AffectNet data oriented model.
5. **_random_graphs*gen.ipynb*** - This is the file that generates some plots based off manually inputted data we got from our GCP training sessions.
6. ***condensedFiles.ipynb*** - This is the file that contains the code for condensing the files (e.g. emotion, valence, image) into one file, applies tensor transformation, and batches them into a dataloader.

3. Reflections (6 points)

This section is your chance to reflect on this project. You should write at least a few sentences for each of these questions. However, try to be concise; a longer answer is not necessarily a better answer. If you want to write several paragraphs about something you're excited about, that's great! On the other hand, don't just write several paragraphs listing all the hyperparameter values you tried; if we fall asleep reading your answers, you might lose points.

- ***What was interesting?***

This entire project was qutie interesting, but the most interesting parts for us was figuring out the correct model/architecture, and seeing the effects of pre-training. Whilst the model architecture was arguably one of, if not the most, arduous part of this entire project, it was very enlightening to see how different layers and layer sequences can have such a big effecf on the model's performance. The sizes of each layer, the depth of the model, and the different functions accompanied with it all play a big role in the model's performance, which to us was very interesting to see it happening in real time.

Pre-training was also very cool, while we did not use it in the end, combining two datasets to create something that can be generalized to a larger dataset was very interesting to see. We wish to see better generalizations if we have the chance with increased time and resources.

- ***What did you learn from this project that wasn't covered by other parts of the class? What concepts from the lectures and readings were most relevant to your project?***

  Two of the biggest things we learned from this project is the conditions for pre-training to succeed, how we should construct our model archeitectures, and hardware constraints. In class, we talked about how CNN works, but how to construct one based on your data is a more practical skill that we definitely learned from this project. Through research, discussions, and trial and error, we were able to come up with a model that was fitting for our dataset and worked for us.

  For pre-training, we learned that pre-training is only effective in some cases. With a skewed data size between AffectNet and RAF-DB, pre-training was not as effective as we thought it would be. However, if we were to use a larger dataset for pre-training or we have a smaller dataset for the main training, pre-training would be more effective. We have concluded that it is a very situational tool and not generic such that it is applicable in all cases.

  Lastly, hardware constraints was something that we did not cover in class, but with such a big dataset and quite a complex model, we had to learn how to navigate the difficult process of implementing a model on GCP and also distributing it amongst the team members.

- ***What was difficult?***

  There were a lot of difficult things, but the most difficult things were definitely: communication, organization, and navigating hardware constraints. With all 3 of us

having packed schedules, it is quite difficult to coordiante between the group members to have dedicated time to talk about our collective work and sync up. A slight hiccup in synchronization between our work, the disorganization starts to show, which makes our work more difficult to do, as not everyone is on the same page when we are each doing work. This is a main cause of the delay in our project, as a good amount of time was spent on trying to get everyone on the same page. The hardware constraints was also very difficult, since we were not expecting this problem, so we had to dedicate extra time to learn about GCP and how to use it. Colab could not cut it nor can local machines, so GCP was the only option for us.

Overall, time and coordination are two of our biggest enemies in this project.

On the technical side, attempting to find the best hyperparameters and tuning it was extremely difficult. There are infinite number of combinations we could test, and mashing together optimal hyperparameters from their respective tests in theory does not guarantee a good model. Because of this, we had to go off of our intuition and hope that our parameters are are not local optimums. Given more time, we would have liked to test out more hyperparameters pairings and combinations.

- ***What were the hardest or most frustrating parts of this project? If someone were about to start on a similar project from scratch, what would you encourage them to do differently?***

  The most frustrating parts were definitely the hardware constraints, the lack of time, and organization. If someone were to start from scratch, we would recommend the following:

  1. Have a good understanding of your datasets and determine your hardware/platform to use before starting the project, because it is very difficult to change midway through the project, which unfortunately we had to do. This will save you a lot of time.
  2. Secondly, start early, and be consistent about your work throughout the work period, because there will be a lot of unexpecte things that will go wrong, which will delay your project. If you start early, you will have more buffer to fix these problems.
  3. Lastly, have a good communication system in place, so that everyone is on the same page and knows what is going on, so you will not have to go back to update everyone.
  4. On a general note, also make sure to use the professors and PMs more, we found that when we are stumped, usually they are there to help us over that hump. Instead of us spending a lot of time to solve it oursleves from scratch, we can get a boost from them and move on to the next problem.

- ***What's left to do?***

  Continuing this project, we have a few things that we would like to do to ensure our project is complete and polished...

  1. Conduct more hyperparameter tuning, and try to find the best hyperparameters for our model with more combinations. This will truly be "thorough".
  2. Try to implement a better pre-training method, such as using a larger dataset for pre-training, or using a smaller dataset for the main training. This will ensure that our pre-training is more effective. With a smallest main dataset could also help us with combatting overfitting and save us a lot of time.
  3. Try to implement a better model architecture, there are more models out there that has proven successful when evaluating emotions/
  4. Produce a pipeline as outlined in Section 1.3.1, so it will gives us a better understanding of how our model works and the workings of this black box.
  5. Implement a video -> images -> emotion pipeline, so that we can use our model to evaluate emotions in videos, which would make our mode more usueful in wider applications.

- ***If you were going to spend another month working full-time on this project, what would you try to accomplish? Why? If I gave your group 1,000,000 USD to use for data collection or compute resources, what would spend it on? Why?***

  If we were to spend another month working on this project, we would try to accomplish the following:

  1. Implement a better model architecture, and maybe try to lockdown specific parts of the image (e.g. mouth or eyes or facial twitches) to see if they can help the model to predict the emotion more accurately.
  2. Conduct more hyperparameter tuning, as stated above.
  3. Utilize more diversed dataset to ensure our model is more robust and can generalize to a wider range of data and inputs.
  4. Integrate the model into an application, so that we can see how our model performs in a real world setting, and can potentially be a fun feature for college students to use.
  5. Integrate valence and arousal into our model, so that we can have a more robust model that can predict more than just the emotion.

- ***If we were to be given $1,000,000, we would spend it on the following:***

    1. Theoretically a better GPU/machine, so that we can train our model faster and more efficiently. But realistically, more GPUs and Machines.
    2. Purchase more diverse datasets (exclusive), so that we can train our model on a wider range of data and inputs. Or hire people to collect real world examples + labeling them so we have even more real world data.
    3. Sign exclusive deal with OpenAI to integrate their technology into our model, so that we can have a more robust model.
    4. Hire more people, but not too many, with everyone having a specific role, so that we can work more efficiently and effectively. This will also help us to have more time to work on the project, as we will not have to spend time on coordinating and organizing our work.