B20AI013_PA2

Ishaan Shrivastava B20AI013

This is a report for my submission of the DAI programming assignment 2.

I have completed the following parts in order:

- Question 1
  - A
  - B
  - C
    - a
    - b
    - c
  - D
  - E
  - F
  - G
  - I
- Question 2
  - A
  - B
  - C
  - D
  - E
  - F

Question 1 - CrisisMMD Multi-Modal Classification

Task Overview

CrisisMMD is a large multi-modal dataset collected from Twitter during different natural disasters. It consists of several thousands of manually annotated tweets and images collected during seven major natural disasters including earthquakes, hurricanes, wildfires, and floods that happened in the year 2017 across different parts of the World.

The purpose of the dataset is to model the task of extracting information related to natural disasters from both text and image content available on social media in order to aid crisis response and management tasks for different humanitarian organizations.

According to the instructions in the assignment, I have loaded and inspected the version 2.0 of this dataset. It includes three types of annotations:

- **Task 1: Informative vs Not informative**
  - Informative
  - Not informative
- **Task 2: Humanitarian categories**
  - Affected individuals
  - Infrastructure and utility damage
  - Injured or dead people
  - Missing or found people
  - Rescue, volunteering or donation effort
  - Vehicle damage
  - Other relevant information
  - Not humanitarian
- **Task 3: Damage severity assessment**
  - Severe damage
  - Mild damage
  - Little or no damage
  - Don't know or can't judge

According to the instructions in the assignment, I have taken 5 classes from the dataset, namely `['california_wildfires', 'srilanka_floods', 'iraq_iran_earthquake', 'hurricane_harvey', 'mexico_earthquake']`.

I have modelled the task to roughly be as follows:

Given an input image and an accompanying text (tweet), find the class label which the tweet and image's context belongs to.

VisualBERT

The VisualBERT model is an excellent choice for this task as it is able to model the joint context between an input image-text pair very effectively using cross-attention layers, enabling the learning of joint contextualized representations of vision and language. It is pretrained on image-caption data using the COCO dataset in order to learn the semantic correlation between images and their corresponding text captions

1A. Finetuning Results - VisualBERT

For the fine-tuning, I tried a couple of different settings for the learning rate:

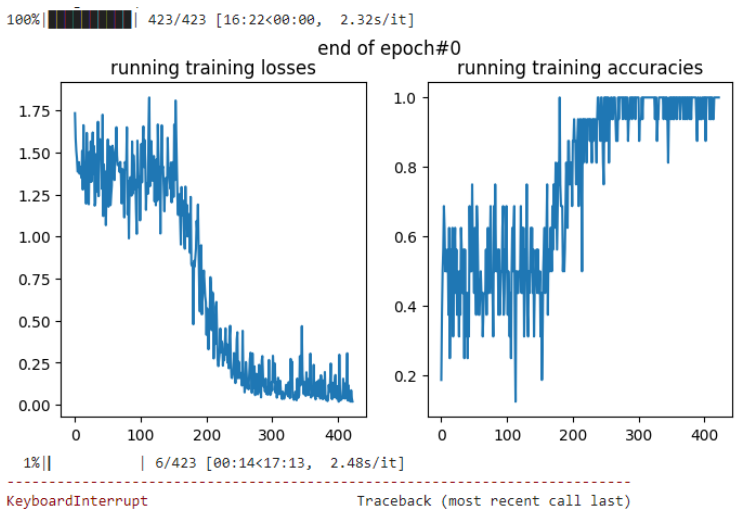
1. `learning_rate=1e-3`
2. `learning_rate=1e-4`
3. `learning_rate=1e-5`

I observed settings 1. was not able to minimize the loss at all and the loss exploded immediately on the first epoch and stayed high. This is because transformers and especially LLMs are extremely sensitive to perturbations caused by a large learning rate. I referred the HuggingFace docs, and the original VisualBERT paper in order to find out how to fine-tune the model on a new domain, and found that LLMs are typically fine-tuned with a learning rate of the order of `1e-5` and a very large weight decay of the order of `1e-2 to 1e-3`. The papers referred, use a very small value of learning rate equal to `2e-5` for the following reasons:

- The use of small learning rates with bias correction avoids vanishing gradients early in training which attributes to the instability in training LLMs.

Thus, I tried lower learning rates `1e-4` and `1e-5` and as predicted, the higher of the two was extremely unstable and took a long time to start minimizing itself, and proceeded to explode right afterwards every single time on every run I tried.

The lowest learning rate, `1e-5`, was found to be the best and the results for fine-tuning for this learning rate are shown below:



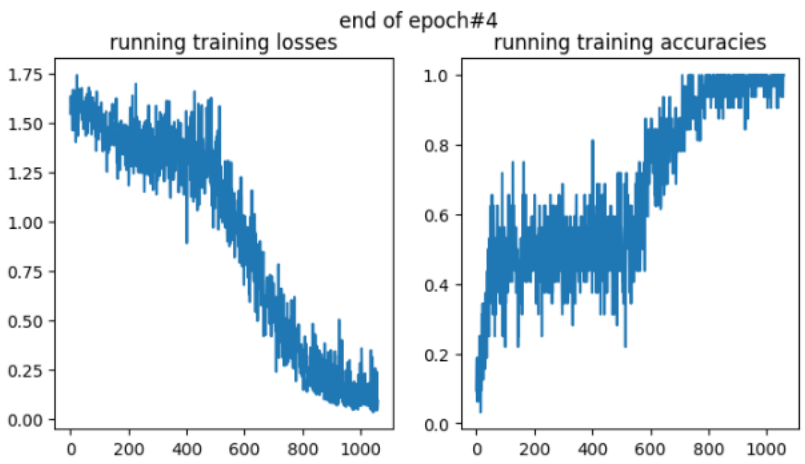
The error in the above cell is just a keyboard interrupt and it was done because the train accuracy had decreased substantially and the loss had almost zero'd in 1 epoch only.

This indicates that the choice of hyperparameters chosen is quite appropriate for the given task. As already discussed,
trying this section with other hyperparameters like a larger learning rate, resulted in extremely poor performance.

I also tried only training the VisualBertForReasoning model while keeping the Object Detection model (part of the image embedding extraction process) frozen. This resulted in the train

This behaviour can be easily explained by reasoning that the object-detection network is crucial to the inference pipeline as it proposes several regions of the input image to be embedded and passed to the multi-layer Transformer model along with the original set of text embeddings.

When I tried reproducing the best results shown above, I found that it was difficult to find a training session which converged right away. The effect of the first few samples seen by the model during training is clearly exhibited here. I resorted to adding a learning rate scheduler to warmup the learning rate in order to reduce the effect of the first few samples. This effected in stabler training that was more likely to achieve similar results to the ones shown above. I was thus finally able to reproduce this training run, albeit over 4 epochs instead of 1, as shown below:

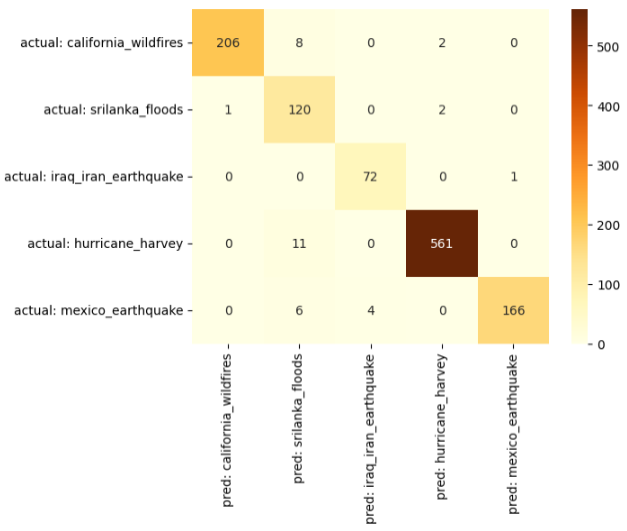


The final metrics for this were:

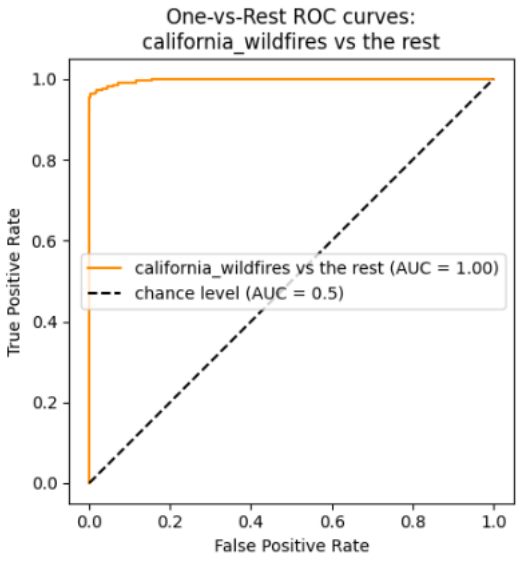
Accuracy for classification:

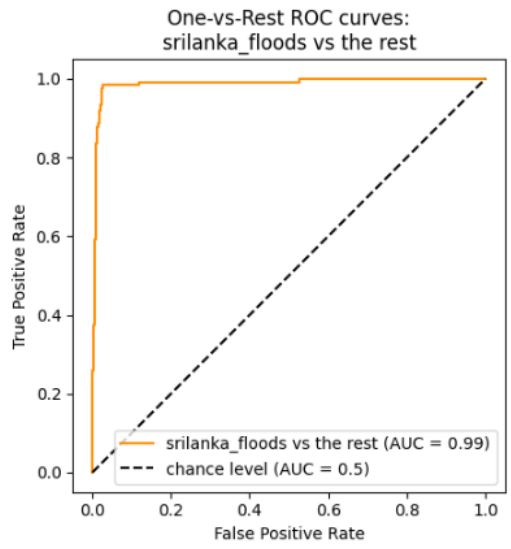
training loss: 0.061
training accuracy: 98.50
test loss: 0.085
test accuracy: 97.38

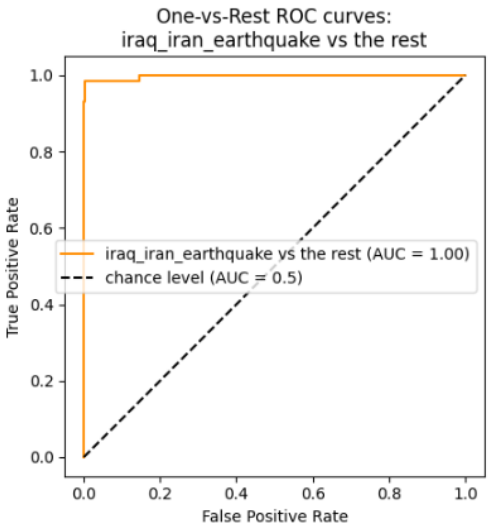
Confusion matrix for classification:

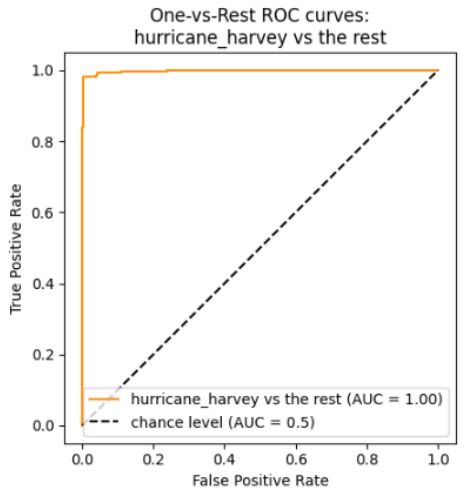


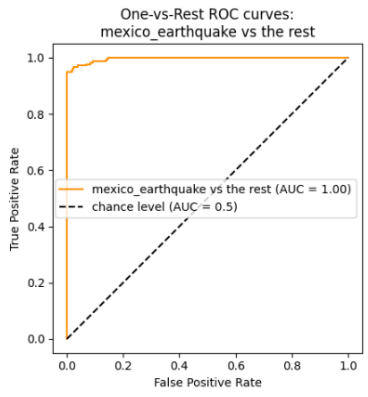
ROC-AUC Scores for every class











1B. Training from Scratch Results - VisualBERT

Training a multimodal LLVM from scratch is no joke. It takes a huge amount of computation power and often takes weeks upon weeks to converge, as language modelling is quite a complicated task in comparison to vision based tasks.

For the training from scratch, I followed the original VisualBERT paper where they have used a `learning_rate=5e-5` for the pretraining from scratch on image-caption reasoning on COCO.

The weight decay was not specified so I have assumed that it was zero, although it is probably a good idea to have the regularization of a weight decay since the dataset we are training on is pretty small.

In accordance with the standard procedures of training LLMs from scratch, I used a learning rate scheduler to provide a warmup to the learning rate for the first few steps. On the right of this I have shown the plot for training from scratch with `warmup_steps=300` (almost an epoch of warmup).

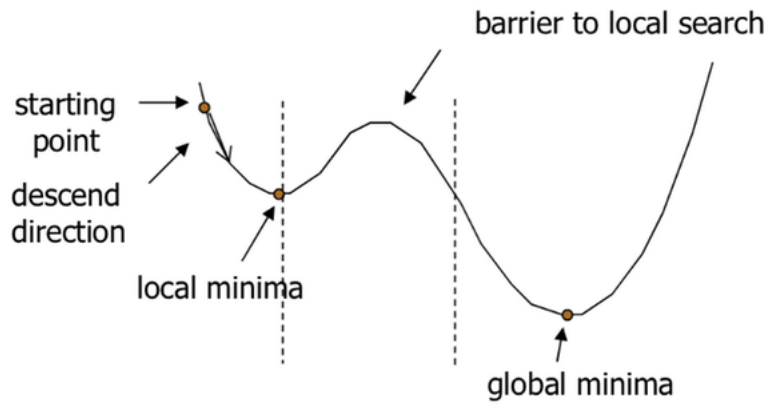
It is evident from the plot that the warmup is insufficient and the learning rate becomes too unstable only half an epoch into the training. I increased warmup with `warmup_steps=2000` which is around 5 epochs long. However, at the end of training for three epochs, the graph of the training losses looked not much different from before increasing the number of `warmup_steps` to `2000`.

Thus, I tried a different learning rate schedule called “Cosine Annealing Schedule”. This learning rate schedule is critical to training many big models today where the loss manifold has several local minima which can effectively trap the parameters into a suboptimal configuration and result in bad performance even after converging.

Cosine Annealing is based on the concept of Simulated Annealing, which is an approximation algorithm and, in general, a variant of hill-climb search (the standard formulation of most stochastic gradient descent based optimizers).

The way hill climbing works is that you start with a random state, then you find a new state which is a neighbour of the current state, and set the current state to that of the neighbour if the value of the objective is smaller there. The “neighbour” of any state is another state that you can get to by applying some small transformation to the previous state.

1. Start at some random x-value
2. Change x by either -1 or $+1$ (pick the smaller one). In this case $x-1$ and $x+1$ are the neighbors of the state.
3. Repeat until both $x-1$ and $x+1$ are larger.



The issue with this algorithm is that it often gets stuck in local minima which is an evergreen problem in deep learning optimization.

Simulated annealing helps fix this issue by sometimes allowing a step to a worse neighbour, which could allow one to reach a global minimum, even if it isn't the same as the local minimum. At each step of simulated annealing, you compute the probability of whether you will take the step to the neighbour or keep the current state. This probability is decided by the acceptance probability function, which determines whether you'll take some step or not, depending on the temperature (A value which determines how big of a "bad" step is permissible).

As itertions progress, the temperature is lowered, and the threshold for taking a bad step decreases correspondingly.

This algorithm proceeds until the loss converges to the global minimum.

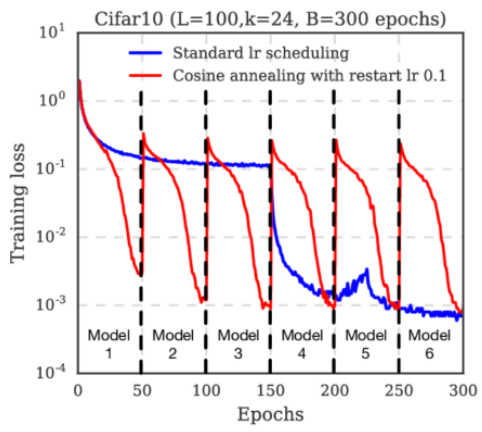
Cosine Annealing is based on the concept of sometimes taking a bad step to escape a local minima, adapted from simulated annealing.

Cosine Annealing is a type of learning rate schedule that has the effect of starting with a large learning rate that is relatively rapidly decreased to a minimum value before being increased rapidly again. The resetting of the learning rate acts like a simulated restart of the learning process and the re-use of good weights as the starting point of the restart is referred to as a "warm restart" in contrast to a "cold restart" where a new set of small random numbers may be used as a starting point.

$$\eta_t = \eta_{min}^i + \frac{1}{2}\left(\eta_{max}^i - \eta_{min}^i\right)\left(1 + \cos\left(\frac{T_{cur}}{T_i}\pi\right)\right)$$

Where η_{max}^i and η_{min}^i are ranges for the learning rate, and T_{cur} account for how many epochs have been performed since the last restart.

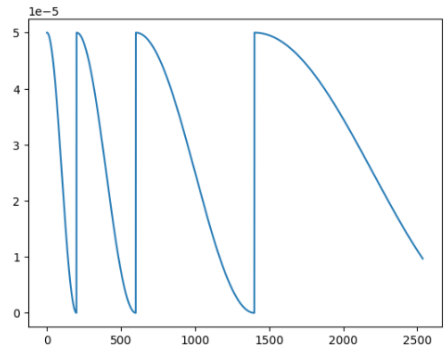
Essentially, the learning rate is scheduled in cycles where the learning rate starts off very high and dips to a very low value by the end of a cycle. The parameter state at the end of a cycle is essentially a local minimum and increasing the learning rate very rapidly after reaching it allows a "warm restart" from a position close to the local minima.



I have implemented this scheduling in my code as the scheduling with warmup did not seem to work as I already demonstrated. The results for the same are given below:

Even the annealing did not work as I did not have the time to further keep training the model or restart the training multiple times in order to get a lucky training run that converges.

A plot of the learning rate scheduled across the training steps (with y axis representing learning rate in units of 1e-5) below:

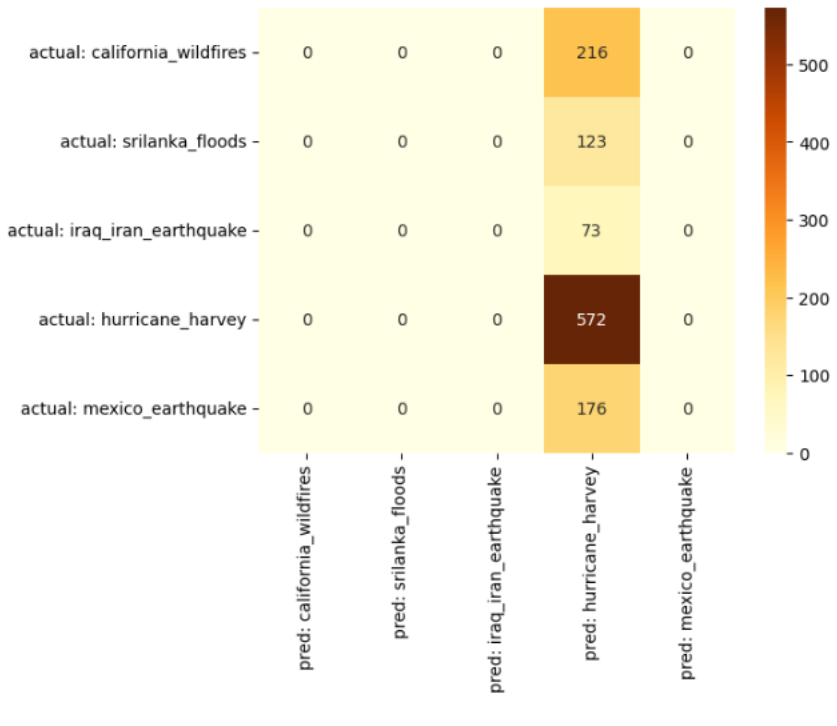


The final metrics for this were:

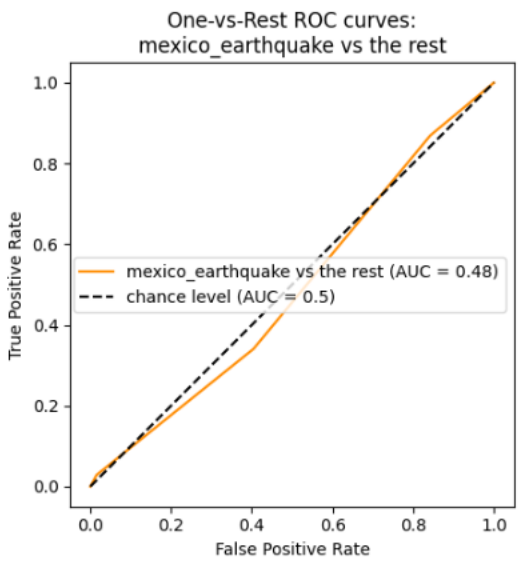
Accuracy for classification:

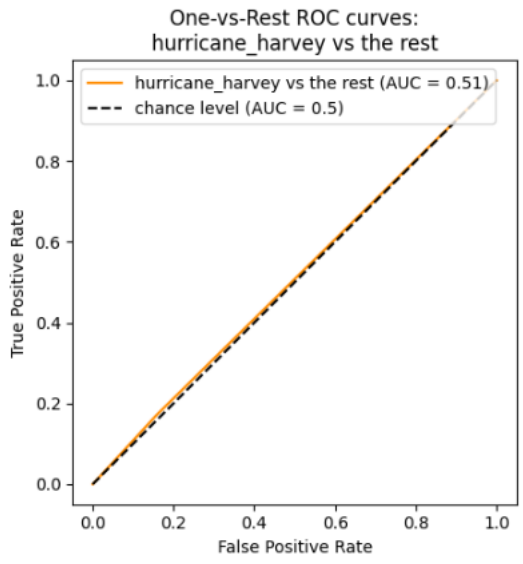
training loss: 1.369
training accuracy: 49.32
test loss: 1.359
test accuracy: 49.49

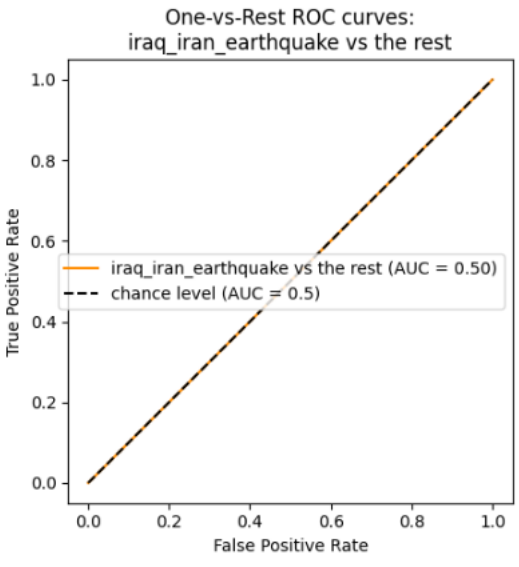
Confusion matrix for classification:

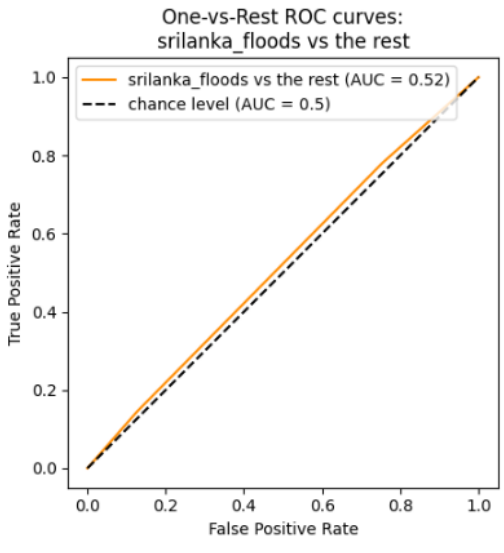


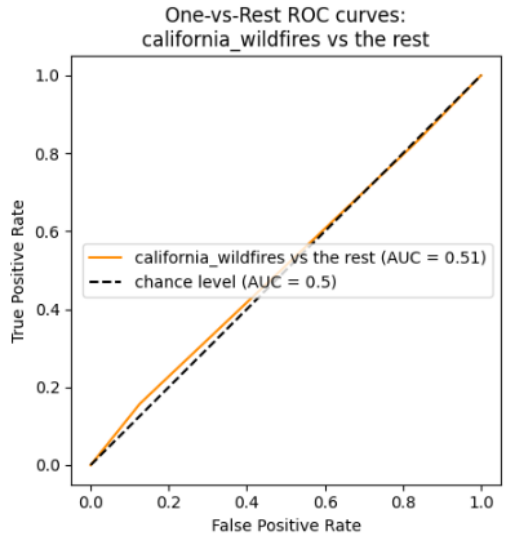
ROC-AUC Scores for every class:







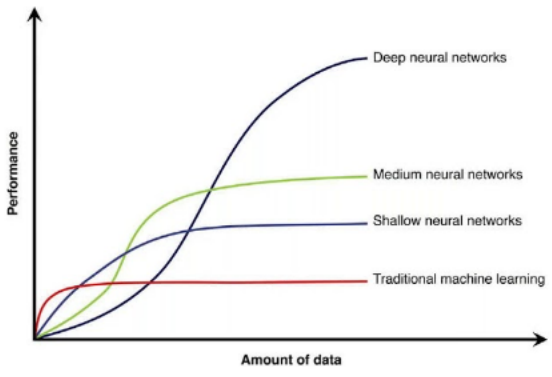




It is clear from these plots above that the training is yet to complete and that the outputs are severely biased and the model is not able to make confident predictions as it has not yet learnt the data distribution. It is important to give time and computation resources to the training of LLVMs from scratch.

In the end, the conclusion drawn from this is that it is significantly harder and requires way more computational resources to train an LLVM like VisualBERT from scratch. This is because these are models with several million or billion of parameters and typically have few or no inductive biases unlike CNNs, RNNs and LSTMs. This means that the training required would take more computation and much, much more amount of data in order to outperform smaller models with inductive biases.

Here is a graphic by Andrew NG demonstrating the same:



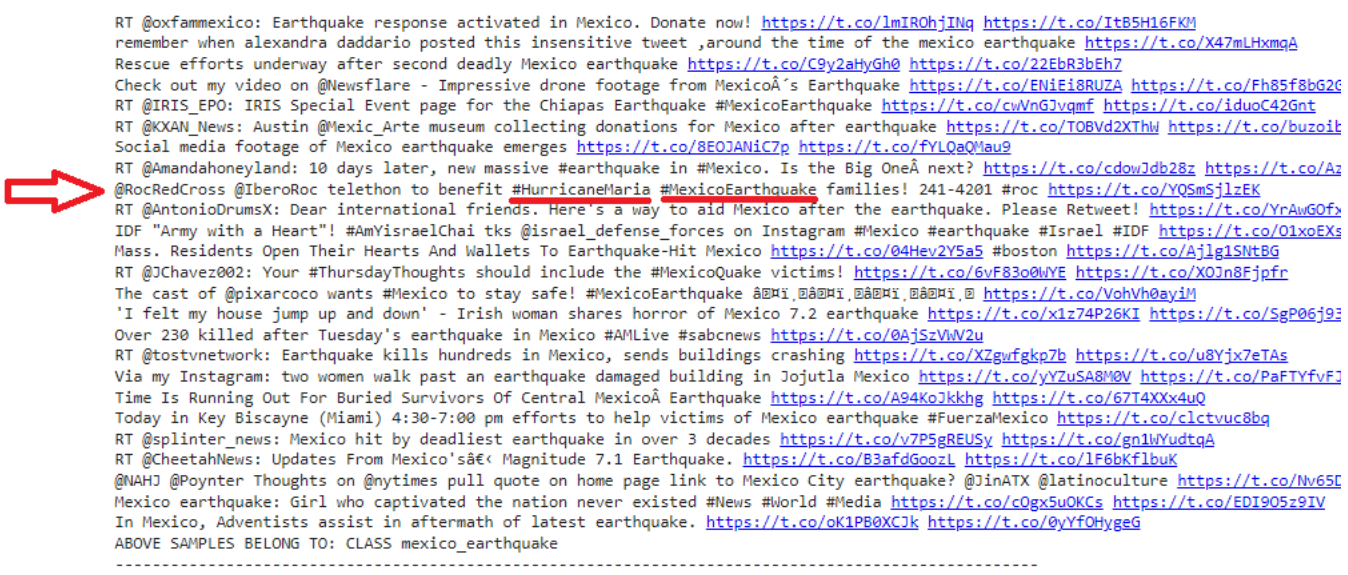
Of course, in this day and age where there are such a vast amount of data generated every day and computational power is increasing logarithmically even now, it is obvious that large language models are winning the race against traditional machine learning models and older deep learning architectures like CNNs.

1C. Bias in Pipeline

> 💡 NOTE:
> Q1.C.c. - bias analysis is already included in this section, I have accompanied instances of bias and their metrics with the in depth analysis of their reasons.)

1C.a) and 1C.c) Data Bias

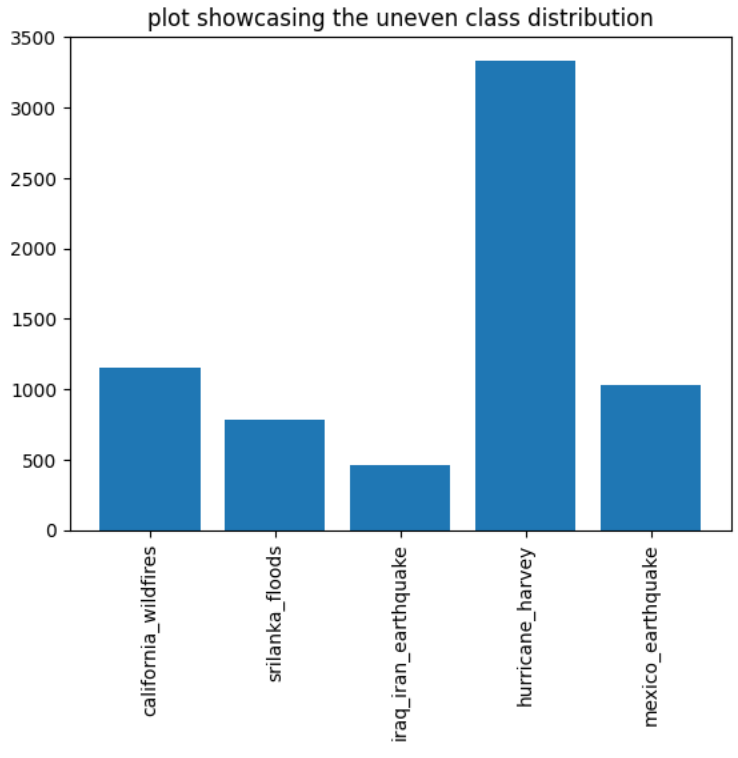
1. Graphical visualization of Bias - Confusion between Mexico Earthquake and Hurricane Harvey:
   I have conducted a visual inspection of the dataset and the tweet samples in order to determine any unusual patterns. For this purpose, I have inspected the top 25 tweets in the dataset for each of the selected classes. One such pattern I found is shown below. It is clearly visible that this tweet contains both mentions of a Hurricane Maria and the Mexico Earthquake. This is because Hurrican Maria occured from 16th sept - 2 Oct 2017 and the Mexico Earthquake happened on 19th September of that same month. This is a possible source of bias and confusion and it is possible that the predictions of other samples were skewed towards Mexico (as visible from the confusion matrices) because of this.
   It was very much possible that i could have included the class for `Hurricane_Maria` in the dataset in the 5 classes chosen, in which case such instances would have caused major confusion between the two classes.



```
RT @oxfammexico: Earthquake response activated in Mexico. Donate now! https://t.co/lmIROhjINq https://t.co/ItB5H16FKM
remember when alexandra daddario posted this insensitive tweet ,around the time of the mexico earthquake https://t.co/X47mLHxmqA
Rescue efforts underway after second deadly Mexico earthquake https://t.co/C9y2aHyGh0 https://t.co/22EbR3bEh7
Check out my video on @Newsflare - Impressive drone footage from MexicoÂ´s Earthquake https://t.co/ENiEi8RUZA https://t.co/Fh85f8bG2G
RT @IRIS_EPO: IRIS Special Event page for the Chiapas Earthquake #MexicoEarthquake https://t.co/cwVnGJvqmf https://t.co/iduoC42Gnt
RT @KXAN_News: Austin @Mexic_Arte museum collecting donations for Mexico after earthquake https://t.co/TOBVd2XThW https://t.co/buzoib
Social media footage of Mexico earthquake emerges https://t.co/8EOJANiC7p https://t.co/fYLQaQMau9
RT @Amandahoneyland: 10 days later, new massive #earthquake in #Mexico. Is the Big OneÂ next? https://t.co/cdowJdb28z https://t.co/Az
@RocRedCross @IberoRoc telethon to benefit #HurricaneMaria #MexicoEarthquake families! 241-4201 #roc https://t.co/YQSmSjlzEK
RT @AntonioDrumsX: Dear international friends. Here's a way to aid Mexico after the earthquake. Please Retweet! https://t.co/YrAwGOfx
IDF "Army with a Heart"! #AmYisraelChai tks @israel_defense_forces on Instagram #Mexico #earthquake #Israel #IDF https://t.co/O1xoEXs
Mass. Residents Open Their Hearts And Wallets To Earthquake-Hit Mexico https://t.co/04Hev2Y5a5 #boston https://t.co/Ajlg1SNtBG
RT @JChavez002: Your #ThursdayThoughts should include the #MexicoQuake victims! https://t.co/6vF83o0WYE https://t.co/XOJn8Fjpfr
The cast of @pixarcoco wants #Mexico to stay safe! #MexicoEarthquake â€¦ https://t.co/VohVh0ayiM
'I felt my house jump up and down' - Irish woman shares horror of Mexico 7.2 earthquake https://t.co/x1z74P26KI https://t.co/SgP06j93
Over 230 killed after Tuesday's earthquake in Mexico #AMLive #sabcnews https://t.co/0AjSzVWV2u
RT @tostvnetwork: Earthquake kills hundreds in Mexico, sends buildings crashing https://t.co/XZgwfgkp7b https://t.co/u8Yjx7eTAs
Via my Instagram: two women walk past an earthquake damaged building in Jojutla Mexico https://t.co/yYZuSA8M0V https://t.co/PaFTYfvFJ
Time Is Running Out For Buried Survivors Of Central MexicoÂ Earthquake https://t.co/A94KoJkkhg https://t.co/67T4XXx4uQ
Today in Key Biscayne (Miami) 4:30-7:00 pm efforts to help victims of Mexico earthquake #FuerzaMexico https://t.co/clctvuc8bq
RT @splinter_news: Mexico hit by deadliest earthquake in over 3 decades https://t.co/v7P5gREUSy https://t.co/gn1WYudtqA
RT @CheetahNews: Updates From Mexico'sâ€‹ Magnitude 7.1 Earthquake. https://t.co/B3afdGoozL https://t.co/lF6bKflbuK
@NAHJ @Poynter Thoughts on @nytimes pull quote on home page link to Mexico City earthquake? @JinATX @latinoculture https://t.co/Nv65D
Mexico earthquake: Girl who captivated the nation never existed #News #World #Media https://t.co/cQgx5uOKCs https://t.co/EDI9O5z9IV
In Mexico, Adventists assist in aftermath of latest earthquake. https://t.co/oK1PB0XCJk https://t.co/0yYfOHygeG
ABOVE SAMPLES BELONG TO: CLASS mexico_earthquake
--------------------------------------------------------------------------------------------------------------
```

2. Graphical Visualization of Bias - Visual Bias
   It is very easy to see, by just plotting the images, that they are biased towards the `California_wildfires` class since it is so visually distinct from the other classes. This could make it possible for the classifier to learn the california wildfire class while pushing the accuracy of the other classes down. This is also evident in the confusion matrices as it can be seen that this class has the highest recall of all the classes.

1. Numerical Bias - Uneven distribution of samples:
   It is clearly visible that the 5 different classes chosen are not equally distribute across the dataset. This can lead to issues such as the predictions being biased to a certain class without a higher frequency unless it is taken care of.



2. Numerical Bias - Data Duplicacy:
   it is evident from the previous parts that there is a problem of data duplicacy in the dataset as shown in the image below.
   I have quantified the average frequency of the samples and printed it as well:

```
average repetition frequency of each tweet: 1.1694036300777875
```

```
RT @ajwamood: #ajwamood : Harvey the first major hurricane to strike the #US since 2005 , causes at least 1 fatality https://t.co/RvDkebyAAC
RT @ajwamood: #ajwamood : Harvey the first major hurricane to strike the #US since 2005 , causes at least 1 fatality https://t.co/RvDkebyAAC
RT @ajwamood: #ajwamood : Harvey the first major hurricane to strike the #US since 2005 , causes at least 1 fatality https://t.co/RvDkebyAAC
RT @ajwamood: #ajwamood : Harvey the first major hurricane to strike the #US since 2005 , causes at least 1 fatality https://t.co/RvDkebyAAC
RT @yIIeza: When we get back to SCHS after Harvey hits : https://t.co/kHMnnURUAA
RT @yIIeza: When we get back to SCHS after Harvey hits : https://t.co/kHMnnURUAA
RT @yIIeza: When we get back to SCHS after Harvey hits : https://t.co/kHMnnURUAA
RT @yIIeza: When we get back to SCHS after Harvey hits : https://t.co/kHMnnURUAA
RT @Rique_210: We Survived Once We Can Survive Again #HurricaneHarvey https://t.co/280Qb9DhRj
RT @Rique_210: We Survived Once We Can Survive Again #HurricaneHarvey https://t.co/280Qb9DhRj
```

This is a problem because data repetition, like repetition of the tweets as shown here, can lead to serious overfitting problems where the model simply memorises a sample and fails to generalise. The extent and effect of the memorisation is hard to quantify. I have verified this from the paper: https://arxiv.org/pdf/2012.14660.pdf

1C.b) and 1C.c) Bias in algorithm

1. Numerical Bias - **DP** - `0.0672` for finetuning, `1.0` for scratch.
   DP is a bias metric. It stands for Demographic Parity. DP measures whether the proportion of positive predictions is the same for two groups, regardless of their actual outcomes. It can be calculated as |P_group1 - P_group2|, where P is the proportion of positive predictions.
2. Numerical Bias - **Recall** - Recall measures the proportion of true positives among all actual positives. It can be calculated as TP / (TP + FN) for each class, where TP is the number of true positives and FN is the number of false negatives. The numerical bias for the scratch model is insignificant because it was unable to converge so it would obviously be heavily biased and it happens to be heavily biased towards hurrican harvey which is the most likely class of all.
   As for the finetuning recall, the `'mexico_earthquake'` class has the least recall out of all. This is because the mexico earthquake event happened in the middle of hurricane harvey, however, hurricane harvey being much longer to last, had way more samples and overshadowed the mexico earthquake label whenever both were present together as it is safer for the model to predict hurricane in this case.

`[('california_wildfires', 0.9537037037037037),`
`('srilanka_floods', 0.975609756097561),`
`('iraq_iran_earthquake', 0.9863013698630136),`
`('hurricane_harvey', 0.9807692307692307),`
`('mexico_earthquake', 0.9431818181818182)]` for finetuning,

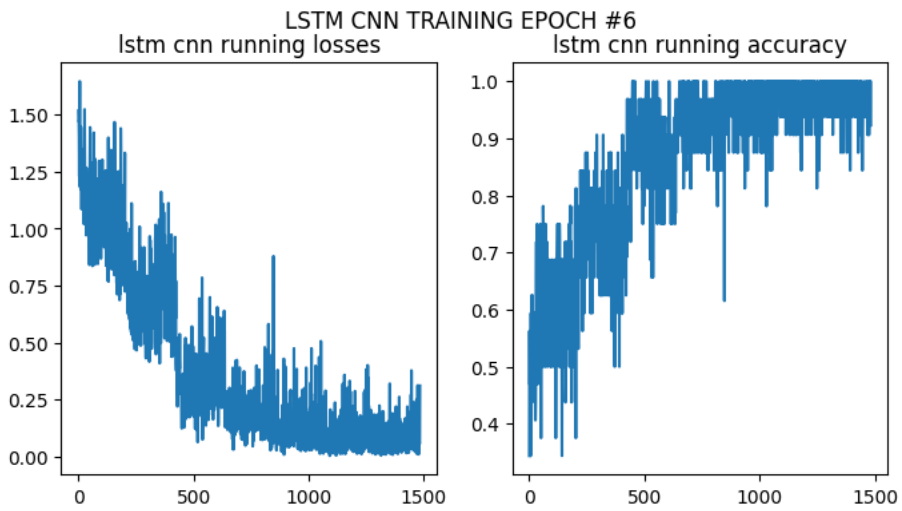
`[('california_wildfires', 0.0),`
`('srilanka_floods', 0.0),`
`('iraq_iran_earthquake', 0.0),`
`('hurricane_harvey', 1.0),`
`('mexico_earthquake', 0.0)]`
for scratch.

1D. LSTM-CNN

The LSTM-CNN architecture I have used is as follows:

1. A feature extractor CNN takes in images as inputs and extracts image features from them
2. An LSTM takes in tokenized and embedded text sequence data and projects the sequence into text embeddings
3. The visual and text embeddings are concatenated and fed into an MLP for classification.

I have used a bidirectional LSTM in order to extract the text embeddings, and a resnet18 architecture in order to extract the image embeddings from the inputs and tweets. The text sequences are padded till a sequence length of 256 for the model to work in a batch fashion.

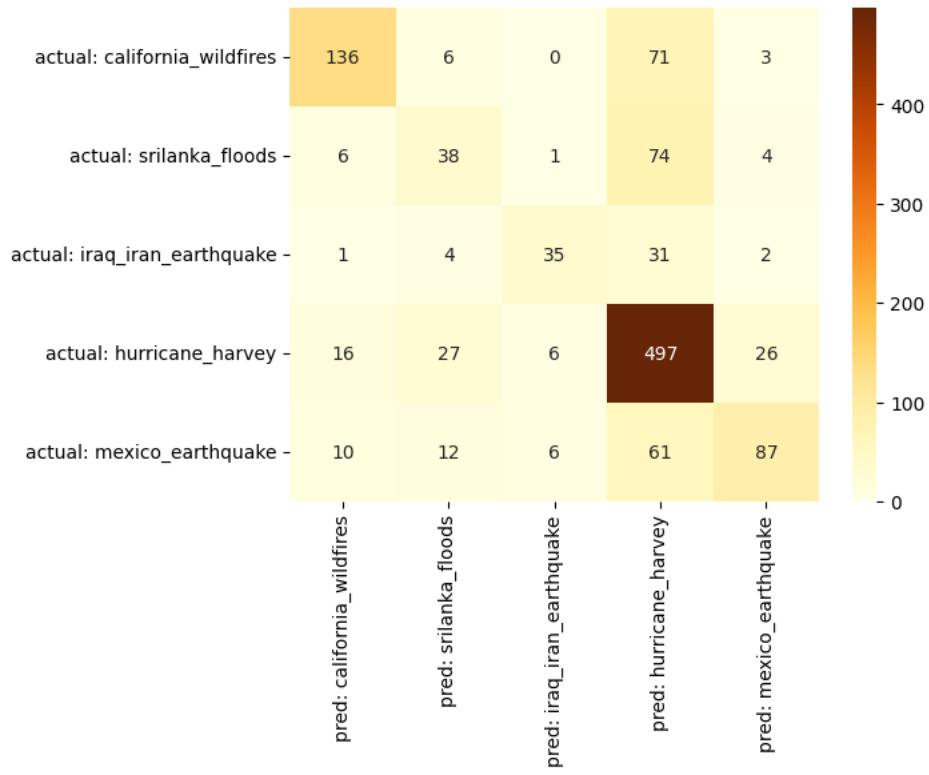


Here are the results of the RNN-LSTM model:

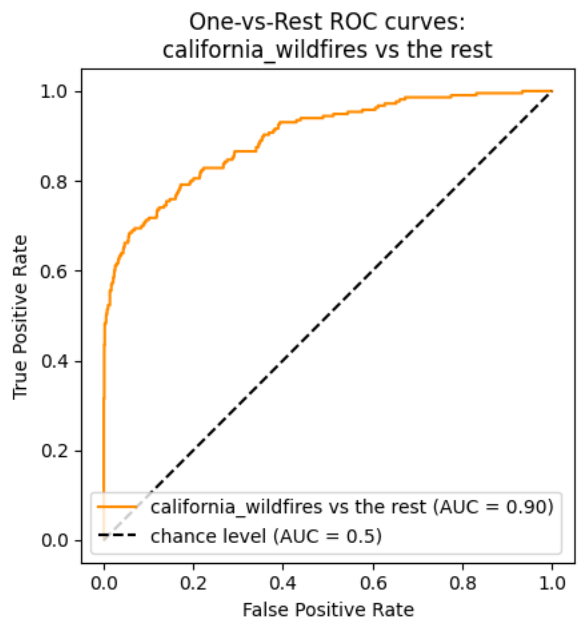
Accuracy for classification:

training loss: 0.036
training accuracy: 98.86
test loss: 1.768
test accuracy: 67.99

Confusion matrix for classification:



ROC-AUC Scores for every class:



1E. Bias dependency on model architecture

The performance of the LSTM-CNN architrecture is compared with that of the VisualBERT architecture and comparisons are drawn between the two, and c

The LSTM-CNN architecture utilizes a greater level of inductive biases than the VisualBERT architecture.

- It gets the text and visual embeddings / encodings separately. This can affect the understanding of an image as the images in this dataset are from tweets, meaning that the image generally would have relevance with the tweet/caption it is posted with.
- Inductive biases in the LSTM network due to how it is structured, encourage the mapping of dependencies between tokens that are closer together in the input sequence. Comparing VisualBERT, it has no such inductive biases in the architecture as it consists of stacks of transformer encoders that captures the interrelations between different parts of the sentences at every level in a very effective manner using self-attention.
- The VisualBERT architecture encodes a joint representation using both the image and text data internally. This joint encoding has the ability to capture better and more relevant information between the image and text pair, allowing the model to “put two and two together”, so to speak, as compared to the LSTM-CNN which encodes the text and image seperately.

In general, the VisualBERT architecture has shown superior performance in many multi-modal tasks compared to the LSTM-CNN architecture as it can capture complex dependencies and relationships between the image and text input. Additionally, VisualBERT has been pre-trained on large amounts of data, which has improved its performance on a wide range of tasks.

This is evident in the performance of the VisualBERT architecture which reaches 96% accuracy on the classification task (with finetuning) compared to the LSTM-CNN architecture which, despite converging to 100% on the training data, only reaches 68% test accuracy and overfits.

However, it is important to note that a model with less inductive biases is not necessarily a better one for every case. It is noted in literature that smaller architectures with more inductive biases can function as better function approximators in case of small datasets and less computation power available. This is because inductive biases are a very very important part of helping the model to learn representations faster, even if those representations are not as good or unbiased as that of a transformer encoder for example.

Overall, it is very notable that the performance of the VisualBERT is much much better on the test set than that of the LSTM-CNN and this can be attributed to the structural differences between the two architectures which lead the LSTM-CNN unable to learn the correlation between images and tweets which is necessary to learn a good classifier.

1F. New Metrics for measuring bias (BONUS)

I have come up with two new metrics for measuring bias.

New Metric 1 - Fair Accuracy

I have called this metric “Fair Accuracy” or “Fair Loss”. Basically the essence of this metric is to:

- quantify the bias or preference of the model to one task / class over the others
- also capture both the overall classification performance across multiple classes implicitly.

The formula for this metric is as follows:

1. Measure the classwise loss α_c by taking the binary crossentropy between the sigmoid of the corresponding class logit $\alpha_c = \text{sigmoid}(\mathbf{o}_c(\mathbf{x}))$ and its binary label, over all samples $\mathbf{x}$ and their corresponding classwise onehot labels $\mathbf{y}^{(c_i)}$.
2. Measure the variance between the classwise performance by measuring the standard deviation of the classwise accuracy vector α that results from step 1.

Essentially, we have:

$$\alpha_{c_i} = BCELoss(\text{sigmoid}(\mathbf{o}_{c_i}(\mathbf{x})), \mathbf{y}^{(c_i)}) \forall c_i \in C$$

$$\text{Fair Loss} = \left(\frac{\sum_{i=1}^{|C|} (\alpha_{c_i} - \bar{\alpha})}{|C| - 1} \right)^{0.5}$$

The esssence behind this metric is to quantify, first and foremost, the spread between the difference in classifiation performance for different classes. It is also to be noted that raising the performance of a classifier in an unbiased manner will tend to reduce this quantity as the margin for misclassification will decrease the further up the performance is pushed for each class. However this is just an observed tendency and not the rule, and it is very much possible that the performance increasing due to an increased bias can increase the fair loss while increasing the performance metric too.

I propose to use this metric for datasets and tasks where the evaluation and comparison of models before and after bias mitigation techniques is being done. Over similar models, this should be a good metric to quantify the bias in any particular approach over multiple tasks or classes.

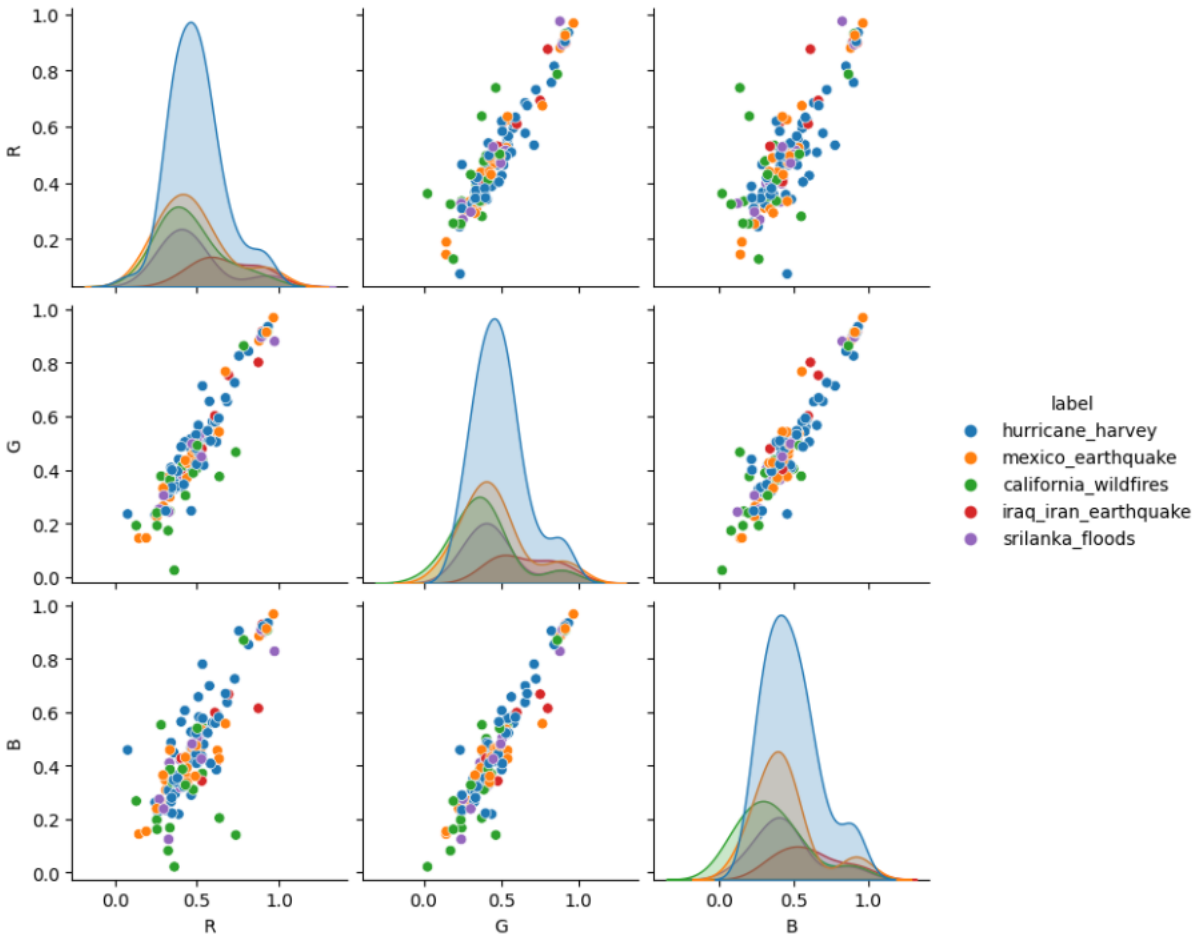
> 💡 The Fair Loss reported for VisualBERT with finetuning was: `tensor(0.0545)`
>
> The Fair Loss reported for VisualBERT with finetuning was: `tensor(1.0)`
>
> It is obvious from the confusion matrices earlier that the results of the VisualBERT in part B which was trained from scratch were completely biased towards the prediction of one class only. This is represented very well in this metric comparison between the two parts A and B.

New Metric 2 - Color Space Difference

I have come up with a visual metric for measuring the bias in the dataset samples. This metric essentially gives a visual indication of how the training process will proceed for a vision-based classifier, atleast in the early stages of training.

It is a well known fact that the first thing that deep neural networks learn is color. Thus, while gauging the performance of a classifier during the training process, especially if it is seeming to flatline, is to ascertain the possible performance by just learning the color space across the different classes.

For this, I average the image inputs across each of the three `['R', 'G', 'B']` channels and plot the pairwise distribution across the color space in order to get an idea of how the samples are distributed and is there any color-based separation possible between them.



For this datastet, it is evident that the samples are very well mixed however there is one class that is easier to classify and is a little separate from the rest - `"California_wildfires"`. It is expected that this class should have the least confusion from the other classes across the training inputs.

It is expected that due to the mixing across color space for all the classes, that the model would atleast initially just learn to classify the largest class and be very biased in the initial training phases.

This is confirmed from our previous examination of VisualBERT training from scratch where it learns to only predict a single class `"Hurricane_harvey"` for every sample (refer to the classwise confusion matrices).

1G. Visualizing salient features used for prediction

1G.a) GradCAM++

It is not very straightforward to use Grad-CAM in target detection; neither the Grad-CAM nor Grad-CAM++ papers mentioned generating CAM maps for target detection. I think there are two main reasons:
a) Target detection is different from classification. The classification network has only one classification loss, and all networks are the same (the last layer of several categories is several neurons), and the final prediction output is a single category score distribution. Target detection is different, the output is not single, and different networks such as Faster R-CNN, CornerNet, CenterNet, FCOS, their modeling methods are different, and the meaning of the output is different. So there will not be a unified method of generating Grad-CAM diagrams.
b) Classification belongs to weak supervision. Through CAM, you can understand the spatial position that the network mainly pays attention to when predicting, that is, "where to look", which has practical value for analyzing problems; while target detection itself is strong supervision, and the prediction border is directly indicated. "Where to look".

Essentially, the essence behind the GradCAM and the GradCAM++ can be simplified as follows:

1. The last layer that encodes some of the localization information is taken and fed into a single layer classifier network.
2. The class activations of the network are visualized using CAM
3. The GradCAM algorithm allows us to compute these activations without adding another Linear layer at the end of the localization layer selected.

The details of the algorithm are as follows:

- Uses gradients flowing from output class into activation maps of last convolutional layer as neuron importance weights (w_k^c).

$$w_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k}$$

where w_k^c = weight of k^{th} activation map w.r.t class c

$A^k = k^{th}$ activation map

- Siimilar to CAM, localization map $L^c_{Grad-CAM}$ is given by:

$$L^c_{Grad-CAM} = ReLU\left(\sum_k w_k^c A^k\right)$$

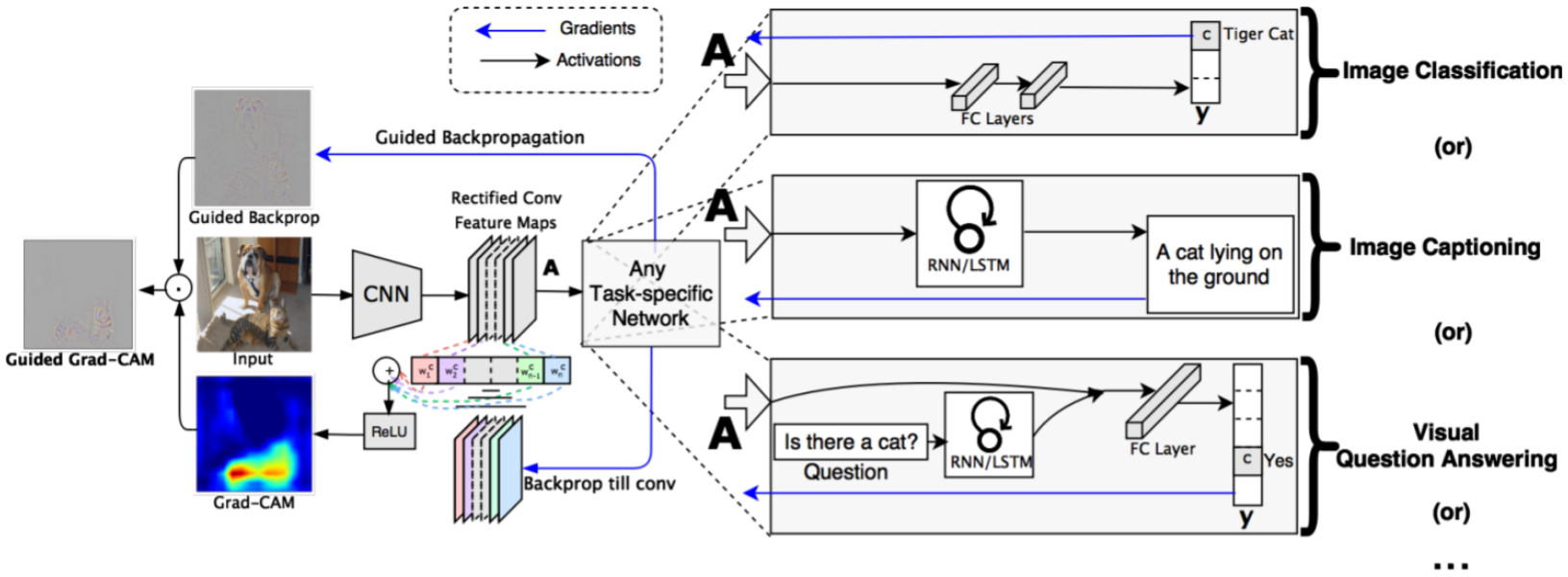


Figure 2: Grad-CAM overview: Given an image and a class of interest (*e.g.*, 'tiger cat' or any other type of differentiable output) as input, we forward propagate the image through the CNN part of the model and then through task-specific computations to obtain a raw score for the category. The gradients are set to zero for all classes except the desired class (tiger cat), which is set to 1. This signal is then backpropagated to the rectified convolutional feature maps of interest, which we combine to compute the coarse Grad-CAM localization (blue heatmap) which represents where the model has to look to make the particular decision. Finally, we pointwise multiply the heatmap with guided backpropagation to get Guided Grad-CAM visualizations which are both high-resolution and concept-specific.

Here we take the faster-rcnn network in detecron2 to generate a Grad-CAM map. The main idea is to directly obtain the border with the highest predicted score; backpropagate the gradient of the predicted score of the border to the feature map of the proposal border corresponding to the border, and generate the CAM map of the feature map.

I have visualized GradCAM++ below the plots for LIME a short while down in the report, instead of immediately after this section. Please take note.

1G.b) LIME

LIME modifies a single data sample by tweaking the feature values and observes the resulting impact on the output. Often, this is also related to what humans are interested in when observing the output of a model

The output of LIME is a list of explanations, reflecting the contribution of each feature to the prediction of a data sample. This provides local interpretability, and it also allows to determine which feature changes will have most impact on the prediction.

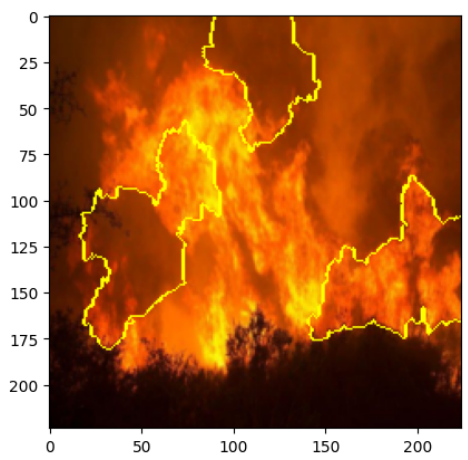
Elaborating, what LIME does is, for a general-modal model f which minimizes $\mathcal{L}$ over an input dataset $\mathcal{D}$:

1. Take a sample x for visualization.
2. Perturb the input by a small amount, and construct the dataset of perturbations π_x.
3. Fit a smaller, interpretable model g on the dataset π_x. Essentially, fit an approximator g while keeping its complexity $\Omega(g)$ low.
4. Highlight the features / support vectors from the dataset π_x that most influence the output prediction of the model in comparison to the prediction made by f.

$$argmin_g \, \mathcal{L}(f, g, \pi_x) + \Omega(g)$$

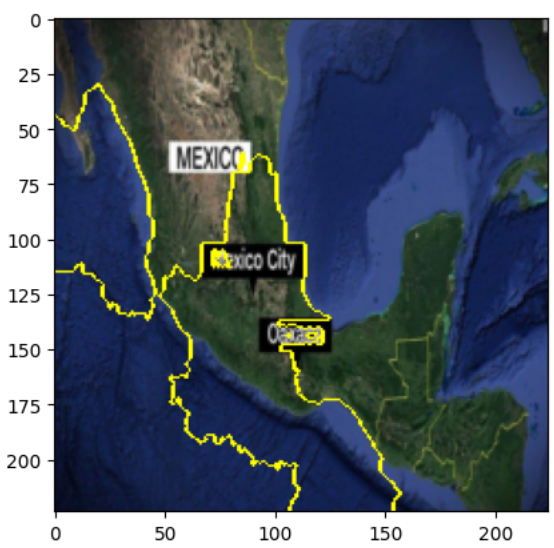
Below I have shown the LIME predictions for the LSTM-RCNN network:

```
california_wildfires          probability: 0.9999978542327881
srilanka_floods               probability: 4.3498877744241327e-07
iraq_iran_earthquake          probability: 1.4516788837681815e-07
hurricane_harvey              probability: 1.240833853444201e-06
mexico_earthquake             probability: 3.2443969644191384e-07
[None, None, None, None, None]
```

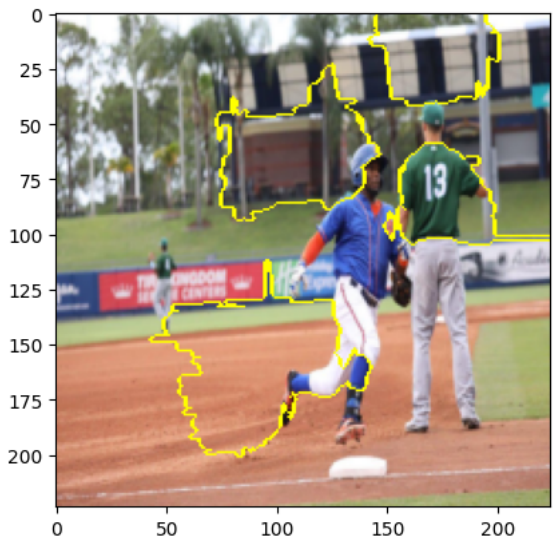


This sample corresponds to the california wildfires. It can clearly be seen that the network decides to focus on the features that highlight a fire.

```
california_wildfires          probability: 8.669991807819244e-12
srilanka_floods               probability: 2.1487034373990355e-10
iraq_iran_earthquake          probability: 2.0229611319422247e-09
hurricane_harvey              probability: 4.913236484327399e-10
mexico_earthquake             probability: 1.0
[None, None, None, None, None]
```



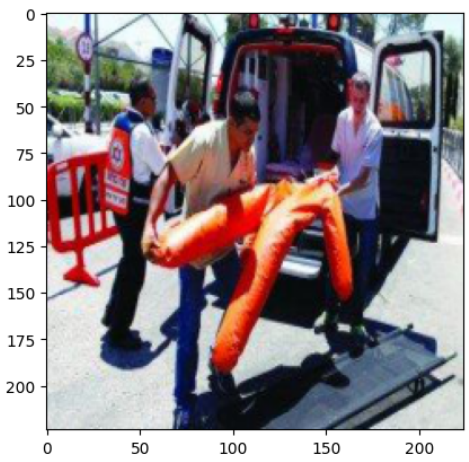
The model is even able to look at an image and see indications of text on it, or even match the shape of the country which indicates the level of learning of the lstm -cnn



```
california_wildfires          probability: 3.133396120347243e-08
srilanka_floods               probability: 4.397032535052858e-05
iraq_iran_earthquake          probability: 1.7411366570740938e-07
hurricane_harvey              probability: 0.9999552965164185
mexico_earthquake             probability: 5.182437234907411e-07
[None, None, None, None, None]
```

The model completely misclassifies this image input however. This is indicative of the bias in the dataset and the model predictions towards `"Hurricane_harvey"` when there is no correlation of the inputs with any natural disaster.

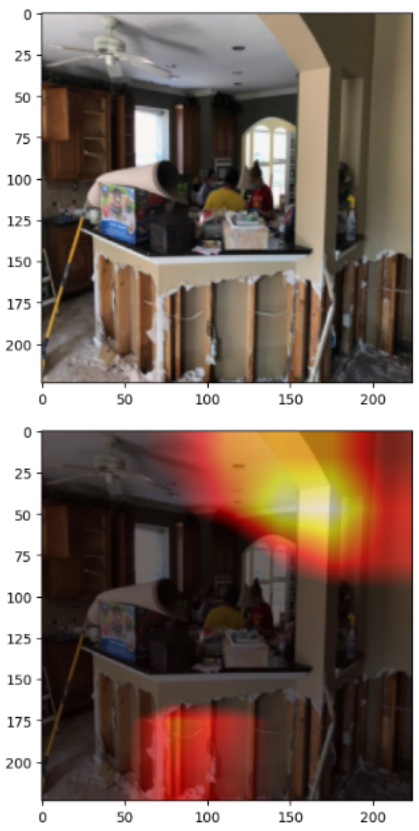
Below I have shown the LIME predictions for the VisualBERT network:

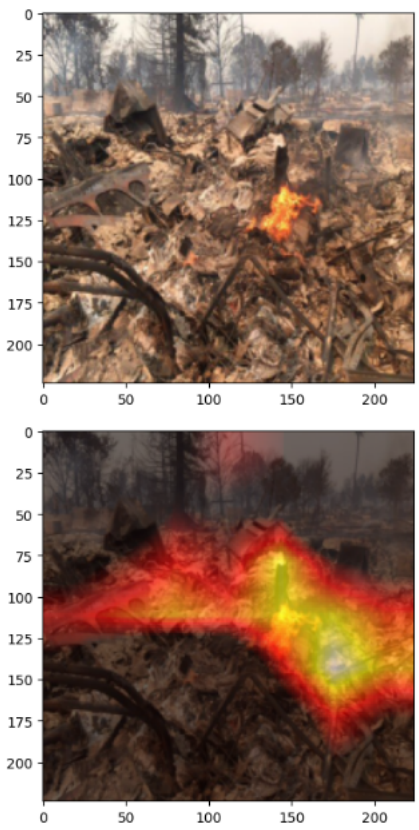


LIME, although implemented correctly for VisualBERT, which I verified, did not highlight any salient features in the image. This is because, as we already saw from the GradCAM++ output of VisualBERT, the visual embedding extractor in the network is not very good and does not attach importance to any feature in the visual modality at the current state at which the training terminated (as it did not converge).

Below I have shown the GradCAM++ for the LSTM-RCNN network:

To the left, I have plotted the GradCAM++ output for a correctly classified image It can be seen that the localization is around the right part of the image, as it localizes the part of the image which has a fire in it, as the image belongs to the "california wildfires" class.

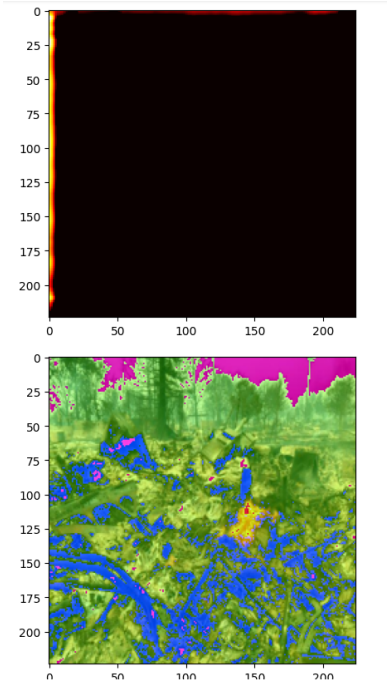




To the right, I have plotted an incorrect classification by the model and it can be easily seen that it does not localize the relevant area of the image very well. This causes a misclassification.

Below I have shown the GradCAM++ for the VisualBERT model in part B:

It can be seen that the model doesnt localise the information from the images at all. This is probably due to the fact that the model has not converged yet or else it would not output a single prediction for every sample as we have already observed before in the confusion matrix.



1I. Measuring Explainability in a quantitative way

Measuring model explainability is an important question that must be raised while comparing and deciding which models to use for Explainable AI. Many visual methods such as GradCAM are prone to failure and should not be relied on too much in the case of reinforcement learning agents, as the visualized output during the training phase is highly dependent on the nature of the training distribution. It is thus important not to lean on GradCAM or other visual/graphical metrics of evaluating and scoring explainability metrics.
We must consider that the end-goal of explainability metrics is to glean information to the general user or researcher about the inner workings of the model. Thus, it is important to involve humans in this process.

Considering these notions, there have been several proposed metrics for measuring explainability which involve human interaction and participation, which make the quantification of explainability possible. These metrics are listed below:

- **Forward Simulatability**

  Give users an explanation along with the input sample, and ask them to predict the same prediction as the model while tracking things like how long the user takes to make the prediction.

- **Perceived Homogeneity**
  Ask users to evaluate the purity or disentanglement of explanations, by e.g. verifying that a dimension corresponds to a single interpretable factor.

- **Intruder Detection**
  Given an explanatory prototype or disentangled concept, show users a set of instances of which one is an intruder, and ask which instance does not correspond with the explanation.

- **Teaching Ability**
  Train users with explanations to understand the model's reasoning, after which humans should predict the ground-truth for a new data instance without having an explanation. Additionally, the user's prediction speed can be measured.

- **Synthetic Artifact Rediscovery**
  A controlled experiment where a property of the predictive model is changed, after which it is evaluated whether humans can reveal this property with the help of explanations.

- **Subjective Satisfaction**
  Ask users to rate explanations on properties such as satisfaction, reasonableness, usefulness, fluency, relevance, sufficiency and trust.

- **Subjective Comparison**
  Show users explanations from different XAI methods (or explanations from humans) and evaluate which method is perceived as being better (in terms of e.g. perceived accuracy, usefulness or understandability).

Question 2

I have implemented two different approaches from the paper, "A Survey of Class Incremental Learning Approaches"

instructions on how to run the different experiments:

1. Navigate to `B20AI013_PA2/q2/FACIL` in terminal
2. Run the terminal command: "python -m pip install -r requirements.txt" in terminal
3. Navigate to `B20AI013_PA2/q2/FACIL/scripts` in terminal
4. Run the terminal command: "sh script_cifar100.sh lwf 0 base"
   a. The output will automatically generate in the terminal at the end of the training and testing process. It will do class incremental training using Learning without Forgetting, with no exemplars stored.
5. Run the terminal command: "sh script_cifar100.sh bic 0 fixd"
   a. The output will automatically generate in the terminal at the end of the training and testing process. It will do class incremental training using **Bias Correction** (BiC, from the paper Large Scale Incremental Learning), with no exemplars stored.

RESULTS

A log of the results is given below for reference.

Here is an explanation of the results below:

- **TAg Acc**: task-agnostic accuracy table.
  - **TAg Acc Avg**: task-agnostic average accuracies.
- **TAw Acc**: task-aware accuracy table.
  - **TAw Acc Avg**: task-aware average accuracies.
- **TAg Forg**: task-agnostic forgetting table.
- **TAg Forg**: task-aware forgetting table.

2A. Learning Without Forgetting (LwF)

I have implemented the paper “Learning without Forgetting” which is a paper on class incremental learning.

Incremental learning is a machine learning technique where a model is trained on new data continuously over time, gradually improving its performance without having to retrain the entire model from scratch. In incremental learning, the model is updated incrementally as new data becomes available, rather than being retrained on the entire dataset.

Class incremental learning is a type of incremental learning where the goal is to incrementally add new classes or categories to a classification model over time. In this approach, the model is initially trained on a set of classes and then gradually expanded to include new classes as they become available.

The steps involved in class incremental learning can be summarized as follows:

- The model is trained on the initial set of classes using standard classification algorithms.
- When new classes become available, the model architecture is augmented/modified to include these new classes. This involves using the knowledge representations learnt during the initial training to help the model learn the new classes.

In Class Incremental Learning, the learner does not have access to the task-ID at inference time, and therefore must be able to distinguish between all classes from all tasks.

Class incremental learning is particularly useful in applications where new categories are constantly emerging, such as in social media monitoring or news classification. By adding new classes to the model over time, the model can adapt to new categories and provide accurate predictions for a wide range of tasks.

I have evaluated the results of the paper “Learning without Forgetting” on the CIFAR-100 dataset in order to reproduce the results shown in the paper.

First, I will summarise the approach. The paper proposed to keep the representations of previous data from drifting too much while learning new tasks by using a knowledge distillation loss. The
method applies the following loss:

$$\mathcal{L}_{dis}\left(\mathbf{x};\theta^{t}\right)=\sum_{k=1}^{N^{t-1}}\pi_{k}^{t\text{-}1}\left(\mathbf{x}\right)\log\pi_{k}^{t}(\mathbf{x}),$$

where $\pi_k(\mathbf{x})$are temperature - scaled logits of the network, and $\mathbf{o}(\mathbf{x})$ is the output of the network before the softmax is applied, and T is the temperature scaling parameter (typically set to $T=2$ in most implementations, however, tuning can lead to better values). π^{t-1} to refer to the predictions of the network after training task $t-1$.

The mathematical explanation behind this cross entropy loss, as given by the authors of this paper, is that for large values of the temperature scaling T, this loss effectively becomes similar to minimizing the $L2$ loss between the logits of the old model and the new model on the old task. Essentially, we want to encourage our new model to remain stationary on the old task hence minimizing the $L2$ loss betwen the old task logits of the new and the old model is intuitively correct.

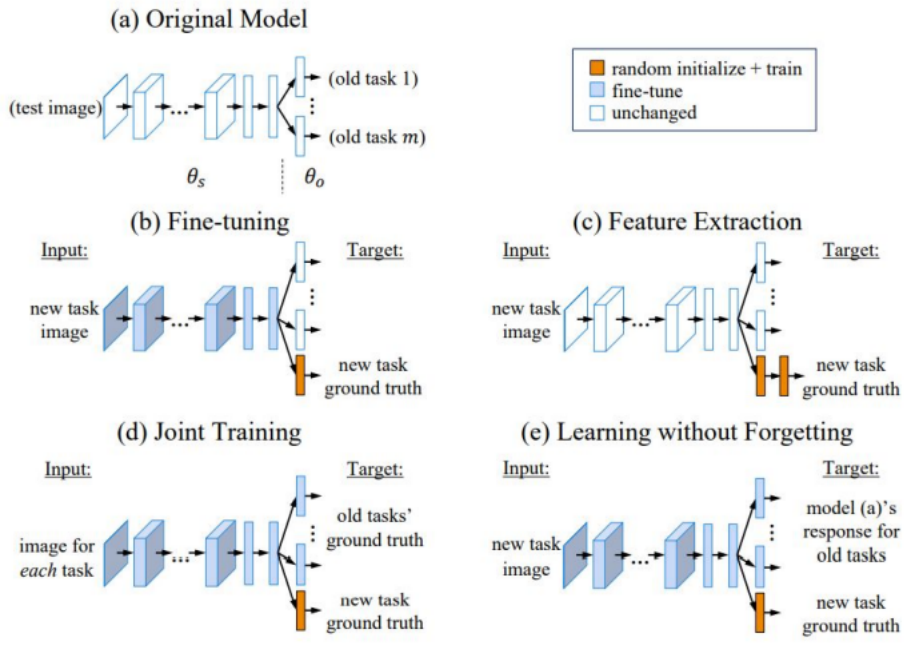
Overall, the algorithm for training and the loss function being minimized in it can be summarized as follows:

LEARNINGWITHOUTFORGETTING:
Start with:
θ_s: shared parameters
θ_o: task specific parameters for each old task
X_n, Y_n: training data and ground truth on the new task
Initialize:
$Y_o \leftarrow$ CNN$(X_n, \theta_s, \theta_o)$ // *compute output of old tasks for new data*
$\theta_n \leftarrow$RANDINIT$(|\theta_n|)$ // *randomly initialize new parameters*
Train:
Define $\hat{Y}_o \equiv$ CNN$(X_n, \hat{\theta}_s, \hat{\theta}_o)$ // *old task output*
Define $\hat{Y}_n \equiv$ CNN$(X_n, \hat{\theta}_s, \hat{\theta}_n)$ // *new task output*
$\theta_s^*,\ \theta_o^*,\ \theta_n^* \leftarrow \underset{\hat{\theta}_s,\hat{\theta}_o,\hat{\theta}_n}{\operatorname{argmin}}\left(\mathcal{L}_{old}(Y_o,\hat{Y}_o)+\mathcal{L}_{new}(Y_n,\hat{Y}_n)+\mathcal{R}(\hat{\theta}_s,\hat{\theta}_o,\hat{\theta}_n)\right)$

EXPLANATION OF THE ALGORITHM:

- $\mathcal{L}_{old}$ is the cross entropy loss between the old task logits output on the new task inputs,. The function of this loss term is to encourage the old task activations to remain stationary while the model is being trained and adapted on the new task.
- $\mathcal{L}_{new}$ is the cross entropy loss between the new task logits output on the new task inputs. The function of this loss term is to encourage the model to adapt to the new task information and perform a correct classification on this task.
- $\mathcal{R}$ is the regularization term applied on the model parameters, which is in practice often an $L2$ regularization loss or an $L1$ regularization loss.

Here is a visual comparison of Learning without Forgetting compared to other approaches:



Learning without Forgetting is a ground-breaking class incremental learning framework which is:

- Easy to implement

- Has good new and old task accuracies
- Very time efficient in training and testing compared to other techniques
- Typically used to set baselines in literature

It is considered SOTA or at the very least a baseline in literature related to Class Incremental Learning in settings where exemplars (input sample examples from previous tasks) are not available during newer tasks, due to memory constraints.

Experiments (LwF):

I have run the training of the model using LwF on a task sequence based on the CIFAR-100 dataset. In each task, I feed the groups of 20 classes (ordered by their original ids in the dataset) for each task training and evaluate the results after every task and once at the end of the entire setup.

Results on CIFAR-100 for LwF:

- **Taskwise accuracy**

  `34.4%  36.5%  51.8%  51.9%  58.0%` for the 5 tasks in order, respectively. It can be seen that the taskwise accuracy significantly drops for the old tasks because the model biases the new tasks. This is as already discussed previously in the report and the reasons for the same have been explained too. CIFAR-100 containing many classes with fine-grained and corase-grained differences, is a great datastet to exemplify the task recency bias and inter-task confusion.

- **Taskwise Forgetting**

  `7.4%  8.6%  1.9%  5.6%  0.0%` for the 5 tasks in order, respectively. Lower is better. It can be seen that the old tasks are forgotten to a large extent and their accuracy drops from their originall high acccuracy. We will see later on in the report that BiC has this problem too, but to a much, much lesser extent.

```
>>> Test on task  0 : loss=2.134 | TAw acc= 34.4%, forg=  7.4%| TAg acc=  2.9%, forg= 38.9% <<<
>>> Test on task  1 : loss=29.593 | TAw acc= 36.5%, forg=  8.6%| TAg acc=  5.3%, forg= 23.8% <<<
>>> Test on task  2 : loss=35.491 | TAw acc= 51.8%, forg=  1.9%| TAg acc= 26.2%, forg= 17.9% <<<
>>> Test on task  3 : loss=39.021 | TAw acc= 51.9%, forg=  5.6%| TAg acc= 20.1%, forg= 20.6% <<<
>>> Test on task  4 : loss=41.381 | TAw acc= 58.0%, forg=  0.0%| TAg acc= 45.4%, forg=  0.0% <<<
Save at C:/ai_sem_7/Courses/dependable-ai/Assignments/q2/FACIL/results\cifar100_icarl_lwf_base_0
****************************************************************************************************
TAw Acc
        41.8%   0.0%   0.0%   0.0%   0.0%      Avg.: 41.8%
        39.5%  45.1%   0.0%   0.0%   0.0%      Avg.: 42.3%
        38.5%  43.0%  53.8%   0.0%   0.0%      Avg.: 45.1%
        34.9%  40.6%  52.8%  57.6%   0.0%      Avg.: 46.4%
        34.4%  36.5%  51.8%  51.9%  58.0%      Avg.: 46.5%
****************************************************************************************************
TAg Acc
        41.8%   0.0%   0.0%   0.0%   0.0%      Avg.: 41.8%
        26.7%  29.2%   0.0%   0.0%   0.0%      Avg.: 27.9%
        14.4%  16.7%  44.1%   0.0%   0.0%      Avg.: 25.1%
         9.3%  10.5%  36.1%  40.7%   0.0%      Avg.: 24.2%
         2.9%   5.3%  26.2%  20.1%  45.4%      Avg.: 20.0%
****************************************************************************************************
TAw Forg
         0.0%   0.0%   0.0%   0.0%   0.0%
         2.3%   0.0%   0.0%   0.0%   0.0%      Avg.:  2.3%
         3.3%   2.2%   0.0%   0.0%   0.0%      Avg.:  2.7%
         6.9%   4.5%   1.0%   0.0%   0.0%      Avg.:  4.2%
         7.4%   8.6%   1.9%   5.6%   0.0%      Avg.:  5.9%
****************************************************************************************************
TAg Forg
         0.0%   0.0%   0.0%   0.0%   0.0%
        15.1%   0.0%   0.0%   0.0%   0.0%      Avg.: 15.1%
        27.4%  12.5%   0.0%   0.0%   0.0%      Avg.: 20.0%
        32.5%  18.7%   8.0%   0.0%   0.0%      Avg.: 19.7%
        38.9%  23.8%  17.9%  20.6%   0.0%      Avg.: 25.3%
****************************************************************************************************
[Elapsed time = 1.1 h]
Done!
```

2B. Highlighting biases

There are several biases observed in LwF. First of all, let us discuss from the perspective of the los functions used in LwF.

1. $\mathcal{L}_{old}$ is the cross entropy loss between the old task logits output on the new task inputs. It is essentially a distillation loss between the model and and old state of itself which encourages the response of the old logits to remain stationary, mitigating catastrophic forgetting.

   $\mathcal{L}_{new}$ is the cross entropy loss between the new task logits output on the new task inputs.

   It is very obvious that $\mathcal{L}_{new}$ is much simpler and easier to minimize than $\mathcal{L}_{old}$ because knowledge distillation is generally speaking a harder task than simply predicting a correct classification. This is because there is more information contained in the softmaxed probabilities of the old model, which can fall anywhere in a range between $[0, 1]$, than in the classification labels of the new task, which can only assume values either 0 or 1.

   This leads to a multi-task learning-like problem where one loss is much harder to minimize than the other. This leads to a gradual overdependence on the easier loss function to minimize and controlling the tradeoff between the two is harder. **LwF does not provide the methods to control this tradeoff** while **BiC directly solves this** by minimizing the classification cross entropy loss over the space of α and β bias correction layer parameters.
2. It can also be observed that the LwF approach is generally much less confident in predicting old classes which is evident from the confusion matrix below.

2C. Large Scale Incremental Learning (BiC)

Learning without forgetting as an approach has many limitations and cannot mitigate losses in accuracy due to class confusion. The class activations are found to be biased towards to new class since information regarding the same is easier to minimize the loss function on and to some extent catastrophic forgetting is impossible to mitigate even with LwF.

Thus, LwF and other exemplar-free knowledge distillation approaches suffer from **catastrophic forgetting** and **inter-class confusion**, along with **task-recency bias** (the performance of new classes being better than the old ones) while learning new classes incrementally, because the performance dramatically degrades due to the missing data of old classes.

While LwF performs pretty well on small datasets, it fails to scale upto larger datasets such as CIFAR-100 and ImageNet.

This is because of the combination of two factors: (a) the data imbalance between the old and new classes, and (b) the increasing number of visually similar classes. The features learned in case of very fine-grained datasets may pose a challenge in solving the inter-task confusion between classes. In this circumstance, it is impossible to solve the inter-task confusion without the use of exemplars from the old classes when the training data is unbalanced.

It is found in this paper that the last fully connected layer has a strong bias towards the new classes, which can be corrected by a linear model which is weighted after every task. This linear model is fitted on validation samples which are not seen by the model during the rest of the training. This ensures a sufficient spread of the data distribution across the response manifold of the model.

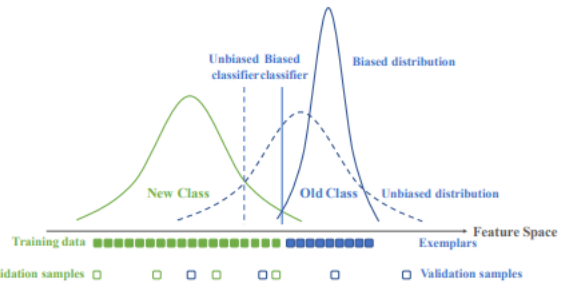


Figure 5. Diagram of bias correction. Since the number of exemplars from old classes is small, they have narrow distributions on the feature space. This causes the learned classifier to prefer new classes. Validation samples, not involved in training feature representation, may better reflect the unbiased distribution of both old and new classes in the feature space. Thus, we can use the validation samples to correct the bias. (Best viewed in color)

Bias Correction (BiC) proposes a simple and effective method to address this data imbalance issue.

The BiC method roughly functions similarly to the approach described in LwF. However, There is one addition to the algorithm which is the **addition of a bias-correction layer** at the end of the FC layers for the old and new tasks.

1. We keep the output logits for the old classes $(1, ..., n)$ and apply a linear model to correct the bias on the output logits for the new classes $(n+1, ..., n+m)$ as follows:

$$q_k = \begin{cases} o_k & 1 \leq k \leq n \\ \alpha o_k + \beta & n+1 \leq k \leq n+m \end{cases},$$

Where α and β are the bias parameters on the new classes and o_k are the output logits for the k-th class. Note that the bias parameters are chares by all new classes, allowing us to estimate them with a small validation set. When optimizer the bias parameters, the convolution and folly connected layers are frozen.

This allows us to re-align the response of the old and new logits in a manner that solves / mitigates the task recency bias by minimizing the classification cross entropy loss of the model over the old and new tasks (done by using exemplars from the validation set) jointly, thus making the classifier an unbiased one over all tasks.

Experiments (BiC):

I have run the training of the model using BiC on a task sequence based on the CIFAR-100 dataset. In each task, I feed the groups of 20 classes (ordered by their original ids in the dataset) for each task training and evaluate the results after every task and once at the end of the entire setup.

Results on CIFAR-100 for BiC:

- **Taskwise accuracy**

  `39.2%  37.4%  43.0%  44.2%  40.0%` for the 5 tasks in order, respectively, compared to LwF's

  `34.4%  36.5%  51.8%  51.9%  58.0%` It can be seen that the overall accuracy is slightly lower across all 5 classes as this approach requires slightly more training than LwF and both were given the same amount of compute and epochs to run, and were not trained to convergence due to time constraints.

  HOWEVER, Taskwise accuracy does not drop as much due to the bias correction layer as compared to LwF which means that the bias is mitigated successfully using this approach. This is as already discussed previously in the report and the reasons for the same have been explained too. CIFAR-100 containing many classes with fine-grained and corase-grained differences, is a great datastet to exemplify the task recency bias and inter-task confusion.

- **Taskwise Forgetting**

  `2.9%  6.6%  2.7%  2.0%  0.0%` for the 5 tasks in order, respectively, compared to LwF's

  `7.4%  8.6%  1.9%  5.6%  0.0%` Lower is better. It can be seen that the extent of forgetting is much lower in BiC especially for the old classes than in LwF due to the bias correction layer

```
>>> Test on task  0 : loss=7.329 | TAw acc= 39.2%, forg=  2.9%| TAg acc=  0.1%, forg= 40.7% <<<
>>> Test on task  1 : loss=5.589 | TAw acc= 37.4%, forg=  6.6%| TAg acc=  0.4%, forg= 42.6% <<<
>>> Test on task  2 : loss=4.498 | TAw acc= 43.0%, forg=  2.7%| TAg acc=  0.9%, forg= 44.3% <<<
>>> Test on task  3 : loss=3.719 | TAw acc= 44.2%, forg=  2.0%| TAg acc=  1.9%, forg= 44.0% <<<
>>> Test on task  4 : loss=3.440 | TAw acc= 40.0%, forg=  0.0%| TAg acc= 39.9%, forg=  0.0% <<<
Save at C:/ai_sem_7/Courses/dependable-ai/Assignments/q2/FACIL/results\cifar100_icarl_bic_fixd_0
****************************************************************************************************
TAw Acc
         40.8%   0.0%   0.0%   0.0%   0.0%      Avg.: 40.8%
         42.1%  43.9%   0.0%   0.0%   0.0%      Avg.: 43.0%
         40.9%  39.9%  45.6%   0.0%   0.0%      Avg.: 42.1%
         40.9%  39.6%  44.6%  46.2%   0.0%      Avg.: 42.9%
         39.2%  37.4%  43.0%  44.2%  40.0%      Avg.: 40.7%
****************************************************************************************************
TAg Acc
         40.8%   0.0%   0.0%   0.0%   0.0%      Avg.: 40.8%
         11.6%  43.0%   0.0%   0.0%   0.0%      Avg.: 27.3%
          3.1%   8.6%  45.2%   0.0%   0.0%      Avg.: 18.9%
          0.7%   4.3%   6.8%  46.0%   0.0%      Avg.: 14.4%
          0.1%   0.4%   0.9%   1.9%  39.9%      Avg.:  8.7%
****************************************************************************************************
TAw Forg
          0.0%   0.0%   0.0%   0.0%   0.0%
         -1.3%   0.0%   0.0%   0.0%   0.0%      Avg.: -1.3%
          1.3%   4.0%   0.0%   0.0%   0.0%      Avg.:  2.6%
          1.2%   4.2%   1.0%   0.0%   0.0%      Avg.:  2.2%
          2.9%   6.6%   2.7%   2.0%   0.0%      Avg.:  3.5%
****************************************************************************************************
TAg Forg
          0.0%   0.0%   0.0%   0.0%   0.0%
         29.2%   0.0%   0.0%   0.0%   0.0%      Avg.: 29.2%
         37.7%  34.5%   0.0%   0.0%   0.0%      Avg.: 36.1%
         40.2%  38.8%  38.5%   0.0%   0.0%      Avg.: 39.1%
         40.7%  42.6%  44.3%  44.0%   0.0%      Avg.: 42.9%
****************************************************************************************************
[Elapsed time = 0.9 h]
Done!
```

2D. Self-devised approaches to mitigate bias

I have devised my own approaches by a close analysis of the implementation code behind both LwF and BiC. Based on the hypothetical shortfalls, I have decided to do parts **2D.a) and 2D.b) jointly** in order to highlight the difference before and after implementing these approaches.

I have augmented the BiC pipeline by adding a data preprocessing step as well as a modification to the algorithm. The descriptions are given below. I am going to call this approach as **BiC - modified.**

2D.a) Data Method - Preprocessing

I have chosen a method to augment the dataset with the use of the `albumentations` library.

The intuition behind these augmentations is to help the model in learning more general representations behind the data. More general representations would help the model as these would not have to be modified as much while fine-tuning old representations on the new tasks.

The choice of augmentations is based on the following:

- Some of the images in the CIFAR100 dataset have more blur than the others. This means that the approach should be robust to blurring at all cost. Blurring is very detrimental to any convolution based architecture as it directly and severely impacts the ability of the lowest-level convolutional filters to extract patterns from the image.
- GaussNoise filter and a MotionBlur filter augmentations are some of the strongest at noising a given input image for CNN related tasks hence I have used these for my augmentations.

```
import albumentations
import albumentations.pytorch
albumentations_transform_oneof = albumentations.Compose([
    albumentations.MotionBlur(p=1),
    albumentations.GaussNoise(p=1),
    albumentations.pytorch.ToTensor()
])
```

2D.b) Data Method - Algorithmic

On instpecting the BiC paper, I found that they used a choice of temperature scaling parameter $T = 2$ as that was the one chosen in the referenced LwF paper. However, in the LwF paper, it is mentioned that the choice of parameter T is highly dependent on the dataset and task and T was chosen based on gridsearch over MNIST.

This can potentially pose a big problem as the effectiveness of the $\mathcal{L}_{dis}$ is contingent on the temperature scaling parameter, as otherwise it will not correctly approximate the $L2$ distillation loss as is intended by the original authors of the knowledge distillation paper, Hinton et al. The approximation is only true in the limiting sense.

Hence, I decided to get rid of the cross entropy-based $\mathcal{L}_{dis}$ and replaced the cross entropy loss inside of it with an $L2$ loss between the old task logits of the old model and the old task logits of the new model.

Experiments (BiC - modified):

I have run the training with a similar setup as described in the previous sections for LwF and BiC with 5 tasks of 20 classes each for CIFAR-100.

Results on CIFAR-100 for BiC - modified:

- **Taskwise accuracy**

  `43.4%  40.5%  43.2%  42.7%  36.9%` for the 5 tasks in order, respectively, compared to BiC's

- **Taskwise Forgetting**

  `0.2%   0.7%  -0.2%  -1.3%   0.0%` for the 5 tasks in order, respectively, compared to BiC's . Lower is better.

The results show that the model not only doesnt forget, but actually is **able to modify its representations to learn more!** The model achieves stellare accuracies for the given tasks and settings.

```
>>> Test on task  0 : loss=8.314 | TAw acc= 43.4%, forg=  0.2%| TAg acc=  0.1%, forg= 40.8% <<<
>>> Test on task  1 : loss=5.393 | TAw acc= 40.5%, forg=  0.7%| TAg acc=  0.5%, forg= 39.9% <<<
>>> Test on task  2 : loss=3.193 | TAw acc= 43.2%, forg= -0.2%| TAg acc=  0.4%, forg= 42.3% <<<
>>> Test on task  3 : loss=1.562 | TAw acc= 42.7%, forg= -1.3%| TAg acc=  1.6%, forg= 39.6% <<<
>>> Test on task  4 : loss=0.445 | TAw acc= 36.9%, forg=  0.0%| TAg acc= 36.8%, forg=  0.0% <<<
Save at C:/ai_sem_7/Courses/dependable-ai/Assignments/q2/FACIL/results\cifar100_icarl_bic_fixd_0
****************************************************************************************************
TAw Acc
         40.8%   0.0%   0.0%   0.0%   0.0%      Avg.: 40.8%
         42.4%  40.8%   0.0%   0.0%   0.0%      Avg.: 41.6%
         42.8%  41.2%  43.1%   0.0%   0.0%      Avg.: 42.4%
         43.6%  40.9%  43.0%  41.4%   0.0%      Avg.: 42.2%
         43.4%  40.5%  43.2%  42.7%  36.9%      Avg.: 41.3%
****************************************************************************************************
TAg Acc
         40.8%   0.0%   0.0%   0.0%   0.0%      Avg.: 40.8%
         11.6%  40.5%   0.0%   0.0%   0.0%      Avg.: 26.0%
          1.6%   8.6%  42.8%   0.0%   0.0%      Avg.: 17.6%
          0.5%   3.6%   6.3%  41.2%   0.0%      Avg.: 12.9%
          0.1%   0.5%   0.4%   1.6%  36.8%      Avg.:  7.9%
****************************************************************************************************
TAw Forg
          0.0%   0.0%   0.0%   0.0%   0.0%
         -1.6%   0.0%   0.0%   0.0%   0.0%      Avg.: -1.6%
         -0.3%  -0.4%   0.0%   0.0%   0.0%      Avg.: -0.4%
         -0.9%   0.3%   0.1%   0.0%   0.0%      Avg.: -0.2%
          0.2%   0.7%  -0.2%  -1.3%   0.0%      Avg.: -0.1%
****************************************************************************************************
TAg Forg
          0.0%   0.0%   0.0%   0.0%   0.0%
         29.2%   0.0%   0.0%   0.0%   0.0%      Avg.: 29.2%
         39.3%  31.8%   0.0%   0.0%   0.0%      Avg.: 35.6%
         40.3%  36.8%  36.4%   0.0%   0.0%      Avg.: 37.8%
         40.8%  39.9%  42.3%  39.6%   0.0%      Avg.: 40.6%
****************************************************************************************************
[Elapsed time = 0.9 h]
Done!
```

2E. Comparing BiC with the bias mitigated version (BiC -modified)

Here I have consructed a comparative analysis between the BiC and BiC - modified approaches as detailed above.

- **Taskwise accuracy**

  `43.4%  40.5%  43.2%  42.7%  36.9%` for the 5 tasks in order, respectively, compared to BiC's

  `39.2%  37.4%  43.0%  44.2%  40.0%.`

  Now, let us compare the **TAg Acc** of **BiC** to the **BiC - modified.**

  - **TAg Acc BiC : `40.7%`**
  - **TAg Acc BiC Modified: `41.3%`**

  It is observable in these numbers that the accuracy for the old tasks was much better preserved as the $L2$ loss is superior and perhaps the temperature scaling T was not large enough to carry the information on the old tasks as much.

- **Taskwise Forgetting**

  `0.2%   0.7%  -0.2%  -1.3%   0.0%` for the 5 tasks in order, respectively, compared to BiC's

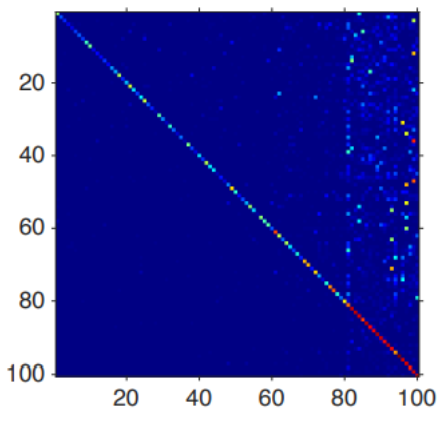
  `2.9%   6.6%   2.7%   2.0%   0.0%` Lower is better.

  It is clearly visible that with the $L2$ loss and augmentations, the rate of forgetting goes even lower. This is only possible if the representations learned are superior to those of the BiC in the intermediate steps between the tasks.

In conclusion, the bias mitigation techniques were able to modify and guide the learning of BiC in a manner that improves upon the original paper.

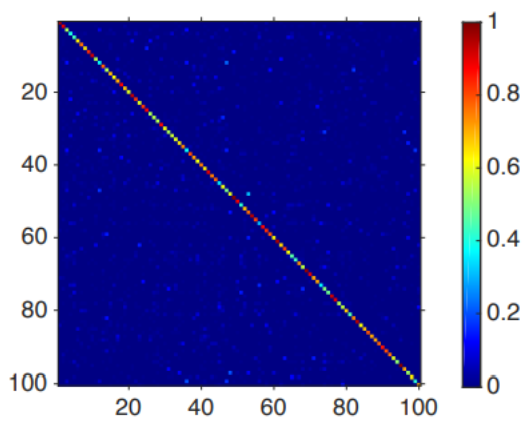
2F. Changes observed before and after applying bias mitigation techniques

I have plotted the confusion matrix for LwF and BiC before and after the bias mitigation. I will show the change in the results for both techniques.



Confusion matrix for LwF

There is a task recency bias which can be observed which leads to underconfident predictions for the old classes and overconfident outputs for the new ones.



Confusion matrix for BiC

It can be seen that the confusion matrix for the training of the ResNet32 model on the BiC approach is unbiased between the classes of the more recent tasks and those of the old ones.

References

- Code for the VisualBert model's Image Embeddings used from: Generating Visual Embeddings using Detectron2 for 🤗 VisualBert
- Code for the VisualBert model and Tokenizer and other modules from: HuggingFace Docs - VisualBERT
- References used for the hyperparameter choice: VisualBERT: A Simple and Performant Baseline for Vision and Language, CVPR 2019, LH Li et al. and On the Stability of Fine-tuning BERT: Misconceptions, Explanations, and Strong Baselines, ICLR 2021, M. Mosbach et al.
- https://github.com/yizt/Grad-CAM.pytorch#detectron2安装 for the idea on how to use GradCAM++ with object detection networks.
- For LIME: https://github.com/marcotcr/lime/blob/master/doc/notebooks/Tutorial - images - Pytorch.ipynb
- For LwF: https://arxiv.org/abs/1606.09282
- For BiC: https://arxiv.org/abs/1905.13260
- For the code to run LwF and BiC: https://github.com/mmasana/FACIL
- For the bonus question related to quantifying explainability - https://www.sciencedirect.com/science/article/pii/S1566253521001093
- Following the above - https://www.diva-portal.org/smash/get/diva2:1534152/FULLTEXT02