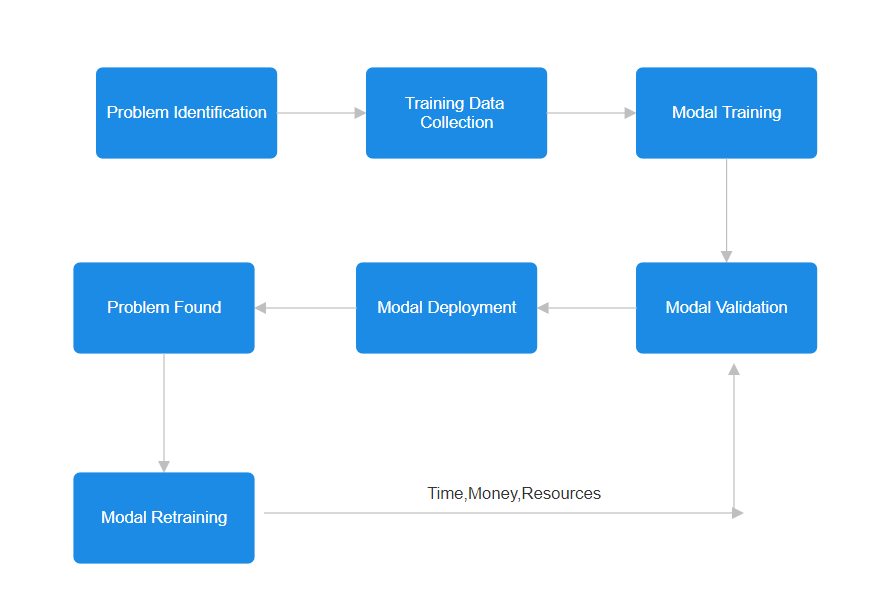
* **Title:** MUSIC: Make Utility Simple In Image Classification
* **Abstract:**

Training a model is the backbone of Machine learning because all the decisions will be done on its basis. Once it is trained we further improve it on the basis of feedback and our requirements. To perform this we use the process of fine-tuning, it is process to further retrain the pre-trained model on small target dataset, we combine multiple tasks and then retrain it on already trained model to improve the performance on these specific tasks. We observed the flaw in the process that when we merge our tasks and then do fine-tuning, our performance decreases. To solve this issue the concept of MUSIC (Make Utility Simple In Image Classification) takes place, we simply fine-tune the model on each task individually instead of merging them as one and as a result we achieved accuracy of 88% from 70% which is a drastic difference.

* **Introduction:**

In the last few years AI (Artificial Intelligence) has taken the world. AI models are trained on data to perform specific tasks. Common type of AI models include **Supervised learning models** which are trained in labeled data to predict outcomes such as regression and classification , **Unsupervised learning models**  which are trained on unlabeled data to identify patterns such as K means classification, **Deep learning models** which uses neural networks like humans and process complex data like images, voice notes etc. AI models are used commonly as core of AI, we often edit them after training to achieve desired results. We train them to change the behavior, if we want to increase the response of specific class in case of classification then we need to retrain the model accordingly and vice versa. This process is called fine-tuning and it is different from training a model. In the context of training the model has not learned anything we assign it parameters and weights to operate while fine-tuning is process after training model where we take the pre trained model’s behavior as a starting point and then tune it on specific tasks.



While performing fine tuning for experimentation on MNIST dataset we observed that overall our accuracy is decreasing, before fine tuning our accuracy was 91.70% and after fine tuning our accuracy decreased to 70% which is a huge difference.

Here what our research contributed is the best practice to follow while fine tuning a model. With fine tuning the model individually on each class the accuracy increases drastically and creates an impact.

* **Related work:**

# Editing Models with Task Arithmetic

* 1. Training a model is the backbone of Machine learning because all the decisions will be done on its basis. Once training is done we retrain it according to our needs and feedback. This process is very costly due to following factors:
  2. Time and capital is effected most on retraining as we need to perform that process again so more time is utilized due to which more money is exhausted. We only have trained model instead of training data so we need to collect it again so need resources to do it.
  3. Fine tuning was introduced to address this problem but it requires labeled data so we need more efficient solution, here comes the solution of editing models with task arithmetic operations using task vectors. Task vector is weights in direction (Positive / Negative) which help the model to perform better.
  4. For example if we want to mitigate undesirable behaviors then we can change behaviors of model by negating as task vector similarly if we can to make model to learn new thing then we can do it by adding task vector. This performs well target tasks or even also improves performance and also efficient to compute as compared to fine tuning.

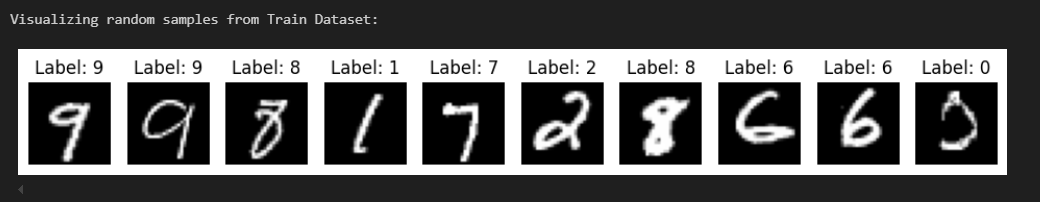
1. **LANGUAGE MODELS ARE HOMER SIMPSON! Safety Re-Alignment of Fine-tuned Language Models through Task Arithmetic**
   1. The introduction tells about the advances and growth of LLMs, which have demonstrated satisfaction in different tasks. However, the fine-tuning process, which is meant to improve the model functionality in defined areas, usually results in compromise of safety. The authors cite earlier works that explain how fine-tuning can argue indirectly make the models unsafe. Here comes the concept of RESTA, whose aim is to reinstate safety without paying a heavy price in performance appeal.
   2. The RESTA Method can be defined in either terms – simplicity and effectiveness. The primary activity consists of a safety vector being added in an elemental manner, to the model parameters. In addition to the authors also came up with a method called Drop and REscale (DARE) which assists in eliminating the extra parameters that were captured during the fine-tuning to increase the efficiency of the safety vectors.
   3. The authors made evaluations of RESTA on two ways of fine-tuning parameters called the parameter efficient fine-tuning PEFT and the full fine-tuning FullFT. It was observed that these two approaches compromised the safety of the models when used on several tasks, including those with harmless datasets. In order to evaluate the performance of RESTA, the authors created a safety evaluation benchmark known as CATQA, which contains 550 dangerous questions that have been divided into 11 groups, each of 5 sub-categories. Such a benchmark was purposefully developed in order to include all the abusive cases specified by OpenAI and Meta’s usage regulations.
   4. As it turned out, the results of the evaluations were promising. The authors noted that the fine-tuned models had a great decrease in the unsafety scores after applying RESTA. When for instance, the Llama-2 model was tested on CATQA, the unsafety score reduced from 33.57% to 12.17% in PEFT while in FullFT, the figure reduced from 22.16% to 4.34%. These results shows that RESTA enhances safety without compromising the performance of the model in a variety of tasks.
2. **Task Arithmetic in the Tangent Space: Improved Editing of Pre-Trained Models**
   1. Task arithmetic refers to the ability to perform arithmetic operations on the weights of a model to get desired outcomes for multiple tasks. This is cost-effective and scalable approach to edit pre-trained models directly in weight space. By changing the weights associated with different tasks, researchers can enhance a model's performance on those tasks or even negate certain tasks, leading to a phenomenon known as task forgetting. The authors highlight that traditional model editing methods often involve costly joint fine-tuning across multiple tasks, which can degrade the model's pre-training performance or zero-shot accuracy. Task arithmetic offers a promising alternative by allowing for more efficient adjustments to the model's weights.
   2. One of the most important contributions of the paper is a proposal to linearizing models in order to enhance weight disentanglement. The authors show that fine-tuning a model in its tangent space can amplify the disentanglement of weights and thus effectively improve performances across benchmark arithmetic tasks. Finally, the authors presented empirical results that linearizing models can bring 5.8 points of accuracy in task addition and 13.1 points less in task negation on various vision-language benchmarks.
   3. Experiment indicated that fine-tuning in the tangent space significantly improved the arithmetic benchmark for most tasks compared to the pre-trained models.

* **Design and Implementation:**

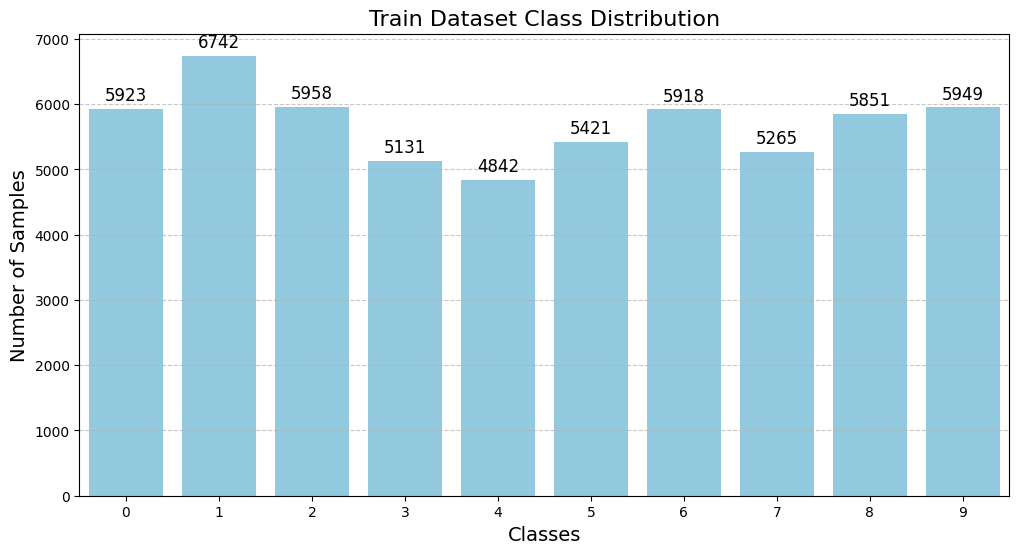
**Dataset:**

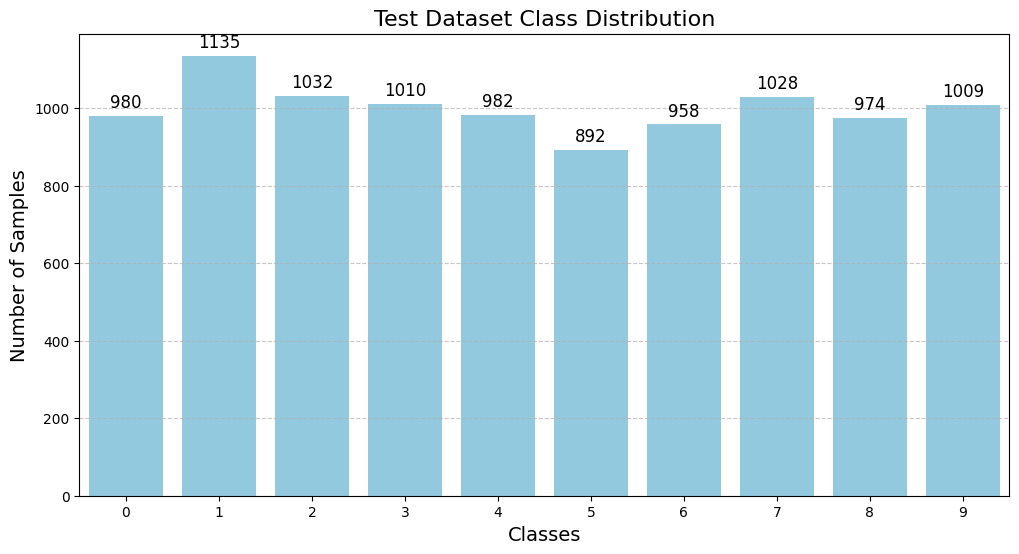
We used MNIST dataset (Images of handwritten digits), that contain 60,000 images as training data and 10,000 images as testing data. Images are classified into classes from 0-9. First of all we performed normalization and EDA.

Visualization of random samples in Training dataset



In training dataset Class 1 have maximum images (6742) and Class 4 have minimum images (4842) and in case of testing dataset Class 1 have maximum images (1135) and Class 5 have minimum images (892).





After this step we trained the model on SimpleNN with epochs of 50 as it is performing best on 50 epochs and batch size of 32. We achieved Train **Loss: 0.0233, Train Acc: 0.9929 (99%), Test Loss: 0.3859, Test Acc: 0.9178 (91%)**.

**Enviornment specifications:**

We achieved these results by running on Macbook pro operating on kernel release 24.1.0. The architecture is x86\_64, powered by a 2.8 GHz Quad-Core Intel Core i7 processor. The system has 4 physical CPU cores and 8 logical CPUs. It is equipped with 16 GB of 2133 MHz LPDDR3 RAM.

**Methodology:**

­­ We split 1000 images, each from class 3, 4 and 7 with 57,000 images remaining in training dataset. Separated images are saved in 3 different files on the basis of class.

Further for fine tuning we used the concept from real world. Let’s suppose you are a student and your professor assigned you a task to learn 30 topics till tomorrow, Will you be able to learn it? Yes you can but your performance will be compromised. Now if you are asked to learn 10 topics in a day and total 30 topics in 3 days. Now your response will be much better as compared to first case.

Here we used the same concept in our training. Instead of fine tuning our model on all three classes on a single run, we separately fine tuned the model on each class. As a result our model performed better when we separately fine tuned model.

* **Evaluation:**
* **Conclusion:**
* **References:**