UNIVERSITÉ DE SHERBROOKE

Université de Sherbrooke

Active Learning of Deep Neural Networks

by

Abir Riahi, 20184248, riaa3102

Nicolas Raymond, 16062546, rayn2402

Simon Giard-Leroux, 12095680, gias2402

in the
Faculty of Sciences
Computer Science Department

April 2021

Contents

List of Figures **iv**

List of Tables **v**

Introduction **1**

1 Design Choices and Project Management **2**

1.1 Design Choices . . . 2

1.2 Project Management . . . 3

2 Methodology **5**

2.1 Active Learning Framework . . . 5

2.1.1 Initialization . . . 6

2.1.2 Execution . . . 6

2.2 Query Strategies . . . 7

3 Experimental Setup **9**

3.1 Hardware Specifications . . . 9

3.2 Models . . . 9

3.3 Datasets . . . 10

3.3.1 Unlabeled Set and Labeled Test Set . . . 10

3.4 Data Augmentation . . . 10

3.5 List of Experiments . . . 11

3.5.1 Generalization . . . 11

3.5.2 Pretraining . . . 11

3.5.3 Query Strategy . . . 13

4 Results and Discussion **15**

4.1 Generalization . . . 15

4.1.1 Results . . . 15

4.1.2 Discussion . . . 21

4.2 Pretraining . . . 23

4.2.1 Results . . . 23

4.2.2 Discussion . . . 24

4.3 Query Strategy . . . 26

4.3.1 Results . . . 26

4.3.2 Discussion . . . 28

4.4 Further Analyses 30
4.4.1 Labeling Rate 30

Conclusion **32**

A Class diagram **33**

Bibliography **34**

List of Figures

2.1 Pool-based Active Learning Framework 5

4.1 G1: SqueezeNet 1.1, No Data Augmentation, No L2 Penalty
Mean Loss and Mean Accuracy Per Epoch 16
4.2 G2: SqueezeNet 1.1, Data Augmentation, No L2 Penalty
Mean Loss and Mean Accuracy Per Epoch 16
4.3 G3: SqueezeNet 1.1, Data Augmentation, L2 Penalty = 0.001
Mean Loss and Mean Accuracy Per Epoch 17
4.4 G4: SqueezeNet 1.1, Data Augmentation, L2 Penalty = 0.005
Mean Loss and Mean Accuracy Per Epoch 17
4.5 G5: SqueezeNet 1.1, Data Augmentation, L2 Penalty = 0.008
Mean Loss and Mean Accuracy Per Epoch 18
4.6 G6: ResNet18, No Data Augmentation, No L2 Penalty
Mean Loss and Mean Accuracy Per Epoch 18
4.7 G7: ResNet18, Data Augmentation, No L2 Penalty
Mean Loss and Mean Accuracy Per Epoch 19
4.8 G8: ResNet18, Data Augmentation, L2 Penalty = 0.001
Mean Loss and Mean Accuracy Per Epoch 19
4.9 G9: ResNet18, Data Augmentation, L2 Penalty = 0.005
Mean Loss and Mean Accuracy Per Epoch 20
4.10 G10: ResNet18, Data Augmentation, L2 Penalty = 0.008
Mean Loss and Mean Accuracy Per Epoch 20
4.11 Accuracy scores on the 2^{nd} validation set for the pretrained and not pretrained SqueezeNet 1.1 23
4.12 Accuracy Scores on the 2^{nd} Validation Set for the Pretrained and Not Pretrained ResNet18 23
4.13 SqueezeNet 1.1: Accuracy scores for four selection strategies: least confident, margin sampling, entropy sampling and random sampling 26
4.14 ResNet18: Accuracy scores for four selection strategies: least confident, margin sampling, entropy sampling and random sampling 26
4.15 SqueezeNet 1.1: Class labeling distribution for Margin sampling strategy . 27
4.16 SqueezeNet 1.1: Class labeling distribution for Random sampling strategy 27
4.17 ResNet18: Class labeling distribution for Least Confidence strategy 28
4.18 ResNet18: Class labeling distribution for Random sampling strategy . . . 28
4.19 Labeling rate scheduler pseudo code 31

A.1 Class Diagram of workflow components 33

List of Tables

3.1 Hardware Specifications 9
3.2 Image Classification Datasets Associated to Each Model 10
3.3 Hyperparameters Combinations of Generalization Experiments 11
3.4 Fixed Initialization Components of Generalization Experiments 12
3.5 Fixed Initialization Components of Pretraining Experiments 12
3.6 Accuracy Goal on the 2^{nd} Validation Set 13
3.7 Fixed Initialization Components of Query Strategy Experiments 14

4.1 Generalization Experiments Test Accuracy Results 21
4.2 Pretraining Experiments Test Accuracy Results 24
4.3 Time Costs (minutes), considering that labeling an image takes 1 minute ($C_L = 1$) and training takes λ times more time. 24
4.4 Query Strategy Experiments Test Accuracy Results 27

Introduction

This report was created as part of the semester project for the IFT780 course given at the Université de Sherbrooke during the Winter 2021 semester. The project concerns active learning techniques for deep neural networks applied to image classification tasks. The steps required to complete this project were as follows:

1. Identify at least three active learning selection criteria.
2. Identify at least two databases from the Datasets PyTorch class.
3. Select at least two types of deep neural networks, with at least one being a convolution-based network.
4. Design a code architecture allowing the training and testing of the neural networks with regards to the selection criteria.
5. Analyze the results.
6. Write a report.

Project Structure

This report is structured as follows:

Chapter 1 covers the code design choices and the used project management tools.

Chapter 2 presents the necessary background for the comprehension of the implemented Active Learning framework. We cover the overall architecture where each component is described in detail.

Chapter 3 brings the scientific procedure used and the experiments performed to validate the proposed techniques.

Chapter 4 includes an analysis of the obtained results.

Chapter 1

Design Choices and Project Management

1.1 Design Choices

The class diagrams can be found in Appendix A of this report. For each class, the top rectangle shows the name of the class, the middle rectangle lists the class attributes and the bottom rectangle lists the class methods.

The list below shows a high-level description of each class. More details about the interactions of each class can be found in the next chapters.

- DatasetManager:
  - Handles the creation of the training, validation and testing datasets. Loads the datasets from the torchvision.datasets class in PyTorch, downloads and saves them and applies transforms to them. A method was created to allow the stratified shuffle splitting of the datasets into subsets to allow the creation of validation subsets from the training dataset.
- DataLoaderManager
  - Handles the creation of the training, validation and testing data loaders. The data loaders are implementations of the DataLoader class in PyTorch and handle the creation of mini-batches for each epoch and allow multiprocessing. The Expert class provides a sampler attribute that can be passed to the training DataLoader in order to restrain its selection of images to the ones considered ”labeled” by the Expert. An update method has been created to allow the expert sampler to be updated during the active learning loop.

- Expert:
  - Simulates an expert who manually labels items from the datasets. The query strategy (or selection criterion) can be specified to be either : random sampling, least confident, margin sampling or entropy sampling.
- TrainValidTestManager:
  - Training, validation and testing manager. This class handles the training of the deep neural network, the validation of the model and the evaluation of the accuracy on the testing dataset. This class saves the loss and accuracy results for training and validation for plotting purposes.
- VisualizationManager:
  - Visualization manager to generate charts. This class can use results from previously ran training and validation routines and plots charts to show the loss and accuracy per epoch. It is also used to display a chart to show which classes have been labeled by the expert for each step in the active learning loop. Also, it is used to display a chart that compares the accuracy as a function of the number of instance queries for different active learning selection criteria.
- ActiveLearner:
  - Active learning loop class. This class implements a pool-based active learning framework and simulates the manual labeling of certain unlabeled images by an expert. The images to label are selected based on a selection criterion which has been defined in the Expert class. This class implements methods to save the history and automatically stop the active learning process based on a patience parameter.

The repository folders and files tree with a high-level description of each file and folder can be found in the README.md file in the root of the repository.

1.2 Project Management

The GitLab git-based repository manager was used to manage code version. The GitLab repository can be found at the following URL:

`https://depot.dinf.usherbrooke.ca/dinf/cours/h21/ift780/rayn2402/2FLM/projet-de-session`

The Trello Kanban-style project management platform was used during this project to track the progress and assignment of each task. The Trello board used for this project can be found at the following URL:

`https://trello.com/b/ztlFOR60/ift780-projet-de-session`

Chapter 2

Methodology

2.1 Active Learning Framework

In the context of our project, we decided to implement the pool-based active learning framework presented in Figure 2.1

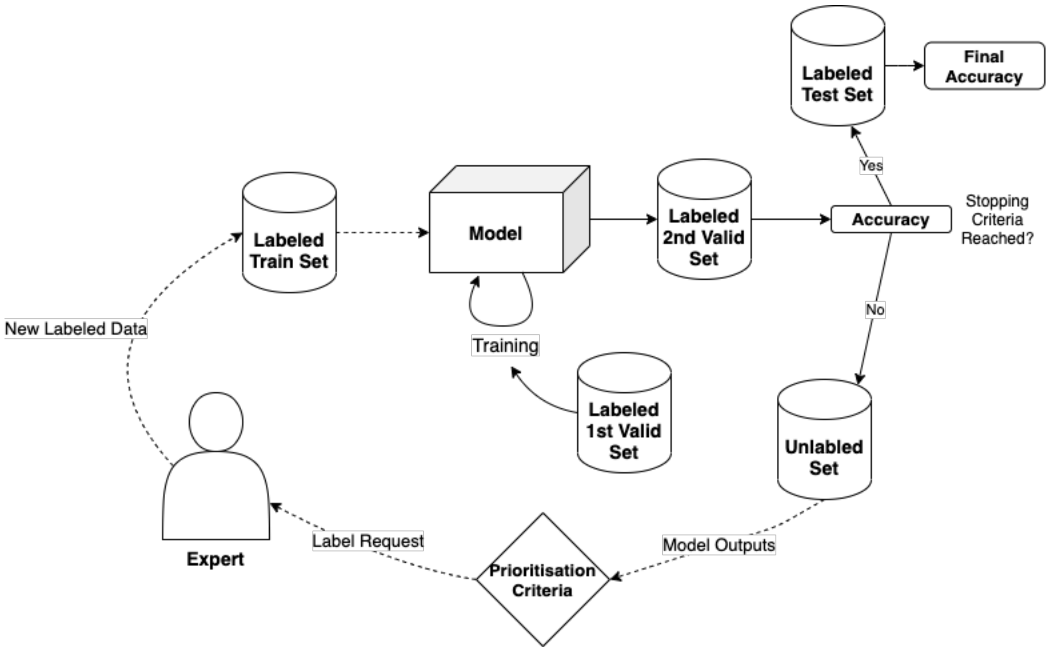

FIGURE 2.1: Pool-based Active Learning Framework

2.1.1 Initialization

In our framework, a few decisions need to be taken before launching the active learning loops.

- A model $\mathcal{M}$ (**pretrained** or **not**).
- A dataset $\mathcal{D}$ from which to extract an **unlabeled set**, two already labeled **validation sets** and an already **labeled test set**.
- A number n_{start} , which represents the quantity of items to randomly label within each class at start.
- A query strategy f sorting the unlabeled data in order to decide next items to label.
- A labeling rate η , which consists of the quantity of items to label for the next loop.
- A number $\mathcal{E}$ of training epochs to execute within each loop.
- A number n_{round} of active learning loops to execute.
- A maximal number P of consecutive active learning loops without improvement.

2.1.2 Execution

Our pool-based active learning framework goes as follow:

1. An expert randomly labels n_{start} items for each class in the **unlabeled set**. The new labeled data are added to the **labeled train set**.
2. $\mathcal{M}$ is trained for $\mathcal{E}$ epochs using the **labeled train set** and its losses and accuracy scores are compared to the 1^{st} **labeled validation set** at each epoch.
3. The accuracy of $\mathcal{M}$ is calculated according to its predictions on the 2^{nd} **labeled validation set**.
4. $\mathcal{M}$ executes a forward pass using all the remaining **unlabeled set**.
5. The softmax output of $\mathcal{M}$ is passed as an input to f in order to set an importance score to each remaining unlabeled element.
6. The η elements with the highest score are requested to be labeled by the expert and added to the **labeled train set**.

7. The steps 2-6 are repeated until step 3 is executed for n_{round} times or no accuracy improvement as been seen at step 3 for P consecutive active learning loops.
8. A final accuracy score is attributed to $\mathcal{M}$ according to its predictions on the **labeled test set**.

2.2 Query Strategies

When we present the unlabeled data to the expert, it searches for the most useful items using a query strategy and returns a fixed number of labeled data. Therefore, query strategies are used to determine which data items should be labeled.

For this project, the implemented query strategies are based on the expert's prediction uncertainty. In other words, we are labeling the items for which the current model is the least sure of what the correct output should be.

- **Least Confidence (LC)**:

  This strategy consists of selecting the item for which the model has the **least confidence** in its most likely class:

  $$x^*_{LC} = \underset{x}{\operatorname{argmax}}\, 1 - P_\theta(\hat{y}|x) \tag{2.1}$$

  where $\hat{y} = \operatorname{argmax}_y P_\theta(y|x)$ [3]

  Unfortunately, the fact that the least confidence strategy only takes into consideration the most probable class label and disregards information about the other probability distributions is one of its major disadvantages.

- **Margin Sampling**:

  The Margin Sampling strategy has a similar approach to the least confidence strategy, but instead of looking for the most uncertain prediction, it selects the items with the least difference between the first and second most probable class probabilities:

  $$x^*_M = \underset{x}{\operatorname{argmin}}\, P_\theta(\hat{y_1}|x) - P_\theta(\hat{y_2}|x) \tag{2.2}$$

  where $\hat{y_1}$ and $\hat{y_2}$ are the most probable class labels. [3]

  Thus, even when using the second most probable label, when having a large set of class labels, the margin sampling strategy is ignoring a large amount of information.

- **Entropy Sampling**:

  One of the most popular active learning strategy is the Entropy Sampling. It uses all of the predicted probabilities:

  $$x^*_H = \underset{x}{\operatorname{argmax}} - \sum_i P_\theta(\hat{y}_i|x) log(P_\theta(\hat{y}_i|x)) \tag{2.3}$$

  where $\hat{y}_i$ ranges over all possible predicted probabilities. [3]

Chapter 3

Experimental Setup

3.1 Hardware Specifications

The table below shows the hardware specifications of the two computers on which the experiments have been performed.

Item	Description
Operating System	Ubuntu 20.04
Manufacturer	Alienware
Model	Aurora R10
RAM	64 GB DDR4 @ 3200 MHz
GPU	Nvidia GeForce RTX 3090 24 GB GDDR6X
CPU	AMD Ryzen 9 3900X 12-Core
Storage	1 TB M.2 PCIe NVMe SSD

Table 3.1: Hardware Specifications

3.2 Models

For all experiment listed in 3.5, we decided to evaluate two different models available via the torchvision PyTorch library.

- ResNet-18 from ”Deep Residual Learning for Image Recognition” [1]
- SqueezeNet 1.1, a computationally more efficient version of the SqueezeNet from ”SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size” [2]

3.3 Datasets

For all experiments listed in 3.5, each model performed the same respective image classification task. In the table below, we provide details of the datasets $\mathcal{D}$ associated to each model $\mathcal{M}$.

Model	**Dataset**	**Number of classes**
ResNet18	EMNIST ('balanced' Split)	62
SqueezeNet 1.1	CIFAR10	10

Table 3.2: Image Classification Datasets Associated to Each Model

3.3.1 Unlabeled Set and Labeled Test Set

For both datasets $\mathcal{D}$ considered, labels and predetermined train and test splits are already available with the torchvision PyTorch library. Hence, using the same seed value for all experiments and each dataset, we decided to:

- Use the complete original test set as our **labeled test set**.
- Use a stratified sample of 20% of the original train set as the 1^{st} **validation set**.
- Use another stratified sample of 20% of the original train set as the 2^{nd} **validation set**.
- Take the remaining 60% of the original train set and mask their labels to create our **unlabeled train set**.

3.4 Data Augmentation

Without data augmentation, the following transforms from the torchvision.transforms PyTorch class are applied to the datasets when they are declared:

- Conversion of the image to RGB. This is required since we are using a 1-channel monochrome dataset (EMNIST) as one of our datasets and our models expect 3-channel images as inputs.
- Conversion of the image to a PyTorch tensor.

When the data augmentation option is activated, the following additional transforms are applied to the data:

- Random rotation within a range of -15° to +15°.
- Random contrast change within a range of -10% to 10%.
- Random hue change within a range of -10% to 10%.
- Random horizontal flip with a probability of occurrence of 50%.

3.5 List of Experiments

3.5.1 Generalization

As our first experiment, we decided to test the generalization ability of both models on their respective 1st **labeled validation sets** throughout the active learning loops. We are conscious that the **labeled train set** can be small at the beginning and might lead to overfitting problem. Hence, we evaluated the following combinations of hyperparameters.

Experiment	**Model**	**Data Augmentation**	**L2 Penalty**
G1	SqueezeNet 1.1	No	0
G2	SqueezeNet 1.1	Yes	0
G3	SqueezeNet 1.1	Yes	0.001
G4	SqueezeNet 1.1	Yes	0.005
G5	SqueezeNet 1.1	Yes	0.008
G6	ResNet18	No	0
G7	ResNet18	Yes	0
G8	ResNet18	Yes	0.001
G9	ResNet18	Yes	0.005
G10	ResNet18	Yes	0.008

Table 3.3: Hyperparameters Combinations of Generalization Experiments

In Table 3.4, we present all the components that were fixed for the active learning initialization of both models.

3.5.2 Pretraining

As our second experiment, we decided to evaluate the impact of using a pretrained model instead of using a model trained from scratch. More precisely, we compared the

Criterion	**SqueezeNet 1.1**	**ResNet18**
Pretrained	No	No
n_{start}	100	50
Query strategy f	Least Confidence (LC)	Least Confidence (LC)
Labeling rate η	100	1000
Training epochs $\mathcal{E}$	10	10
n_{rounds}	20	10
Patience P	3	3
Dataset $\mathcal{D}$	CIFAR10	EMNIST
Batch size	50	50
Learning rate	0.0001	0.0001
Validation set 1 size	0.2	0.2
Validation set 2 size	0.2	0.2

Table 3.4: Fixed Initialization Components of Generalization Experiments

accuracy scores obtained on the 2^{nd} **labeled validation** and the final score obtained on the **labeled test set** for both trained and pretrained models. Note that for pretrained models, we had to replace the last fully-connected layer since the number of classes in our task is different from the number of classes in the ImageNet dataset.

In Table 3.5, we present all the components that were fixed for the active learning initialization of both models.

Criterion	**ResNet18**	**SqueezeNet 1.1**
n_{start}	50	100
Query strategy f	Least Confidence (LC)	Least Confidence (LC)
Labeling rate η	1000	100
Training epochs $\mathcal{E}$	10	10
n_{rounds}	10	20
Patience P	3	3
Data augmentation	Yes	Yes
L2 Penalty	0.005	0.005
Dataset $\mathcal{D}$	CIFAR10	EMNIST
Batch size	50	50
Learning rate	0.0001	0.0001
Validation set 1 size	0.2	0.2
Validation set 2 size	0.2	0.2

Table 3.5: Fixed Initialization Components of Pretraining Experiments

3.5.2.1 Cost Analyses

Pool-based active learning provides a framework that can reduces time related to data labeling by an expert.

Put in a simple form, the cost related to the training of a model in an active learning framework can be expressed as

$$cost = C_L \cdot (n_{start} \cdot nb\_classes) + KC_T + \sum_{i=1}^{K-1} C_L \cdot \eta \qquad (3.1)$$

where C_L is the cost related to the labeling of a single item, C_T is the cost related to the training of $\mathcal{M}$ for $\mathcal{E}$ epochs and K is the number of active learning loops to execute to reach the stopping criterion (e.g a certain accuracy). Note that the labeling cost is summed only $K-1$ times since there are no label request at the last round.

For the sake of simplicity, we will suppose in this experiment that the cost of training is proportional to the cost of labeling ($C_T = \lambda C_L$, $\lambda \in \mathcal{R}$) and we obtain the following cost expression

$$cost = C_L \cdot (n_{start} \cdot nb\_classes + K\lambda + (K-1)\eta) \qquad (3.2)$$

Hence, this next experiment was put in place to evaluate the effects of the pretraining on the cost related to the training of a model for a classification task using a pool-based active learning framework.

To do so, using the results from the pretraining experiments for both train and pretrained models, we evaluated the number of active learning loops that were required to get to a certain accuracy goal for both models (K).

	SqueezeNet 1.1	**ResNet18**
Accuracy goal	0.35	0.80

Table 3.6: Accuracy Goal on the 2^{nd} Validation Set

Once found, we looked at the cost of the experiments for few values of λ.

3.5.3 Query Strategy

As our third experiment, we decided to evaluate the impact of the query strategy employed. To do so, we compared the accuracy scores obtained on the 2^{nd} **labeled validation** and the final score obtained on the **labeled test set** for the SqueezeNet 1.1 model, using each of these query strategies:

- Least Confidence
- Margin Sampling

- Entropy Sampling
- Random Sampling

In Table 3.7, we show the components that were fixed for the active learning initialization of both models.

	ResNet18	**SqueezeNet 1.1**
Pretrained	No	No
n_{start}	50	100
Labeling rate η	1000	100
Training epochs $\mathcal{E}$	10	10
n_{rounds}	10	20
Patience P	3	3
Data augmentation	Yes	Yes
L2 Penalty	0.005	0.005
Dataset $\mathcal{D}$	CIFAR10	EMNIST
Batch size	50	50
Learning rate	0.0001	0.0001
Validation set 1 size	0.2	0.2
Validation set 2 size	0.2	0.2

Table 3.7: Fixed Initialization Components of Query Strategy Experiments

3.5.3.1 Class Labeling Analyses

Considering the query strategy with the best test accuracy and the random sampling strategy, we decided to create pie charts showing the top-10 classes in which the expert was requested to add labels. The goal was to analyze which classes required most labels according to the best query strategy and compare the numbers to the random sampling results.

Chapter 4

Results and Discussion

4.1 Generalization

4.1.1 Results

The following subsections show the mean loss and mean accuracy per epoch charts for each generalization experiment, followed by a summary table which shows the test accuracy results for each experiment.

4.1.1.1 G1: SqueezeNet 1.1, No Data Augmentation, No L2 Penalty

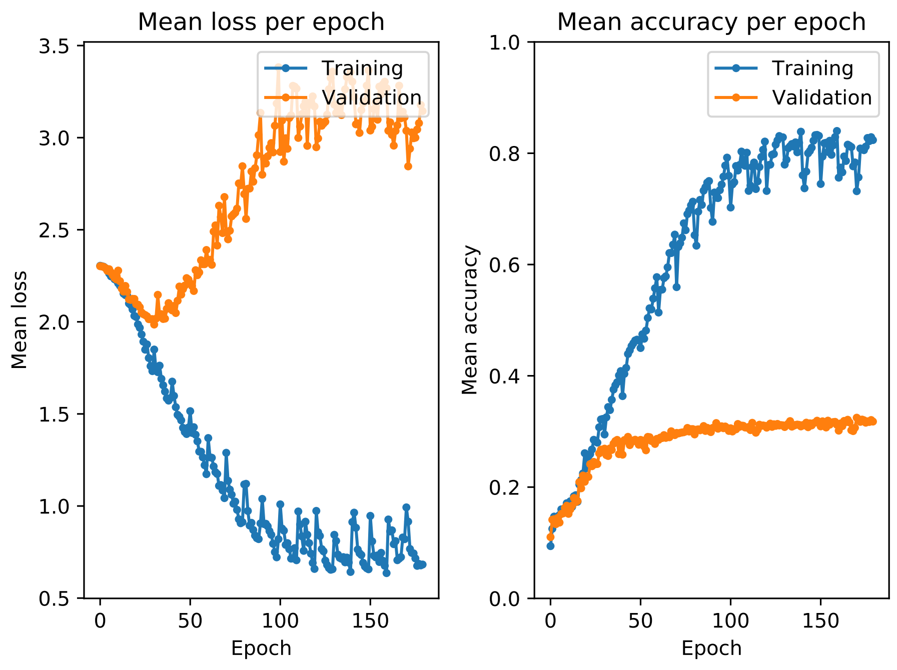

Figure 4.1: G1: SqueezeNet 1.1, No Data Augmentation, No L2 Penalty
Mean Loss and Mean Accuracy Per Epoch

4.1.1.2 G2: SqueezeNet 1.1, Data Augmentation, No L2 Penalty

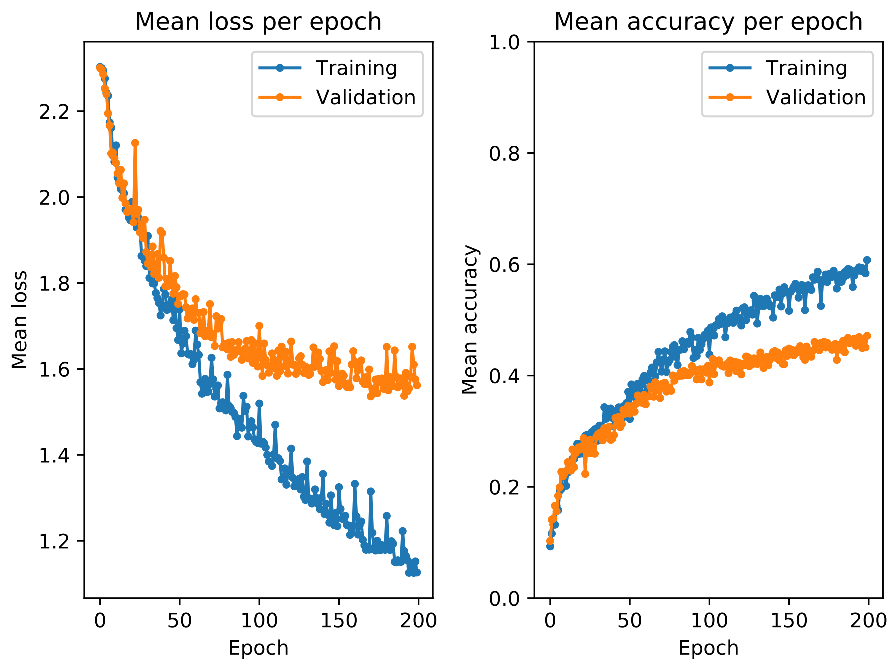

Figure 4.2: G2: SqueezeNet 1.1, Data Augmentation, No L2 Penalty
Mean Loss and Mean Accuracy Per Epoch

4.1.1.3 G3: SqueezeNet 1.1, Data Augmentation, L2 Penalty = 0.001

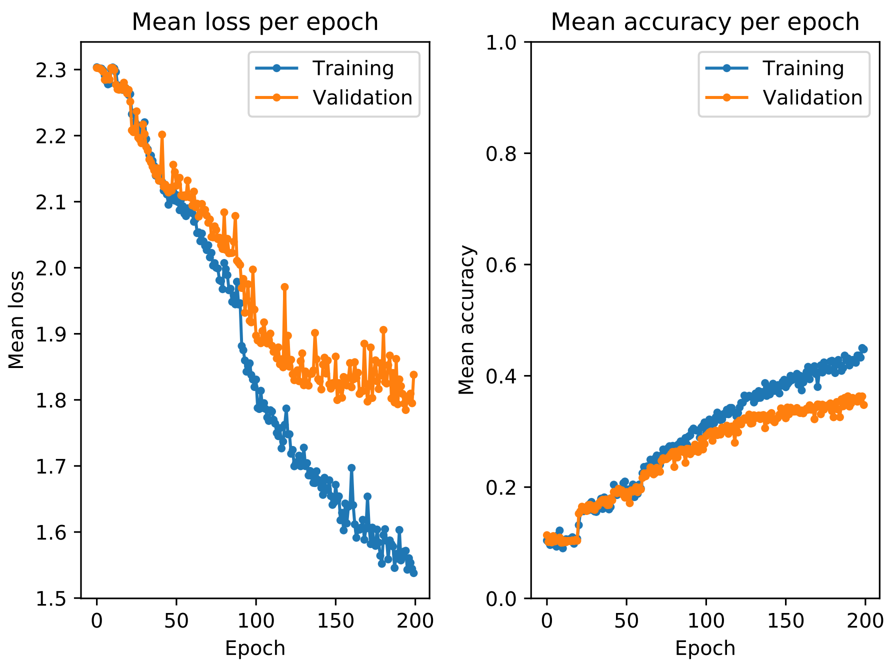

FIGURE 4.3: G3: SqueezeNet 1.1, Data Augmentation, L2 Penalty = 0.001
Mean Loss and Mean Accuracy Per Epoch

4.1.1.4 G4: SqueezeNet 1.1, Data Augmentation, L2 Penalty = 0.005

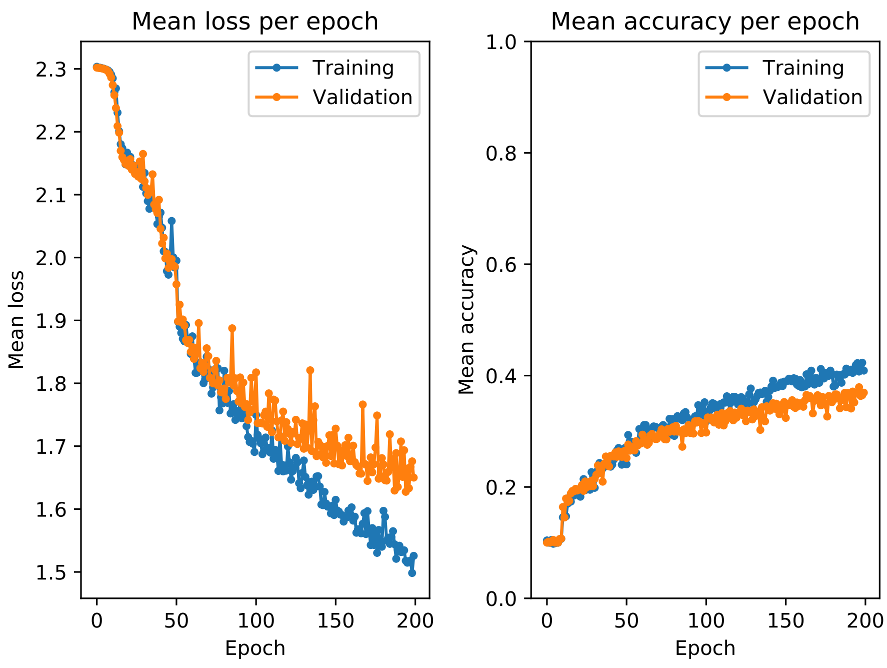

FIGURE 4.4: G4: SqueezeNet 1.1, Data Augmentation, L2 Penalty = 0.005
Mean Loss and Mean Accuracy Per Epoch

4.1.1.5 G5: SqueezeNet 1.1, Data Augmentation, L2 Penalty = 0.008

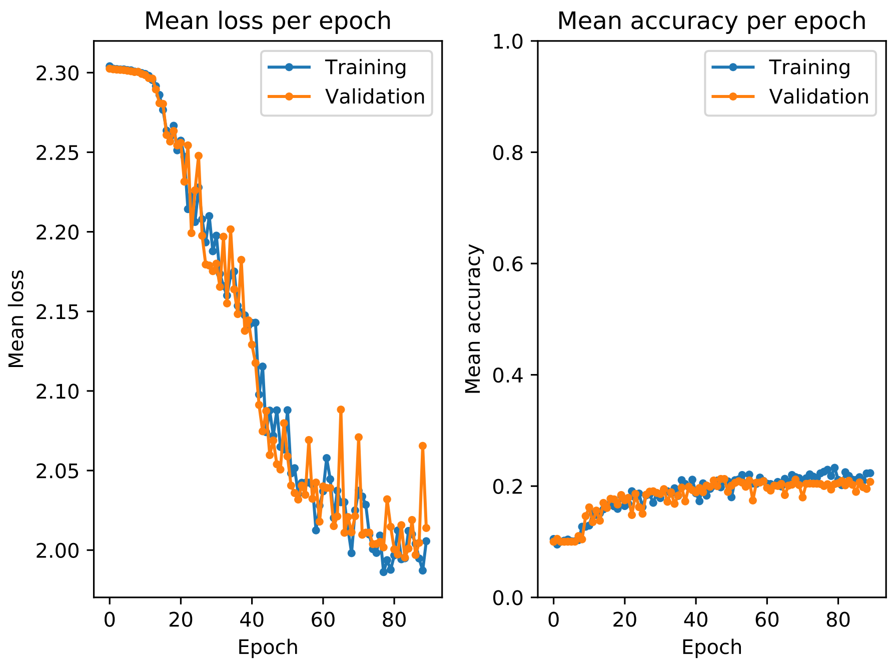

FIGURE 4.5: G5: SqueezeNet 1.1, Data Augmentation, L2 Penalty = 0.008
Mean Loss and Mean Accuracy Per Epoch

4.1.1.6 G6: ResNet18, No Data Augmentation, No L2 Penalty

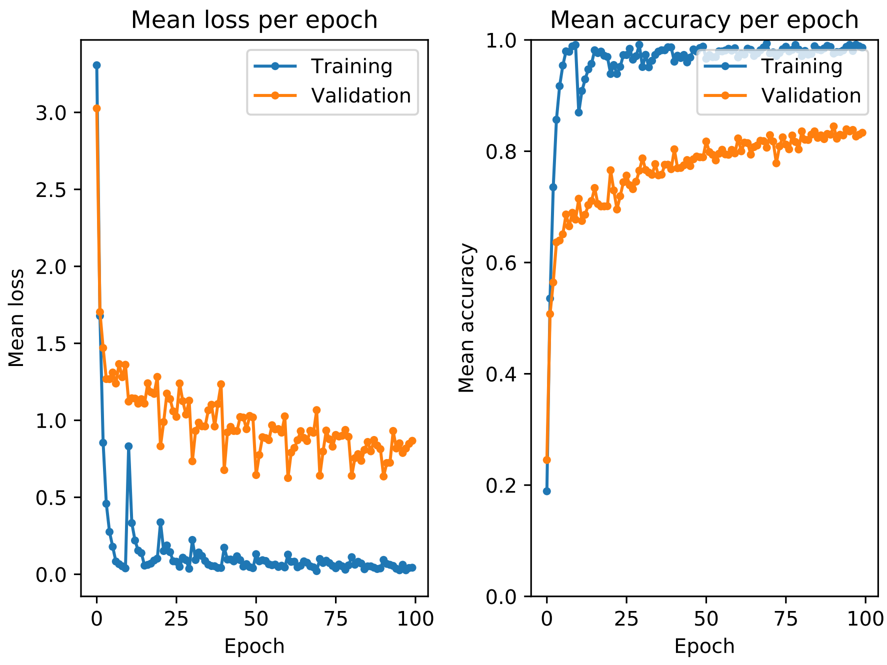

FIGURE 4.6: G6: ResNet18, No Data Augmentation, No L2 Penalty
Mean Loss and Mean Accuracy Per Epoch

4.1.1.7 G7: ResNet18, Data Augmentation, No L2 Penalty

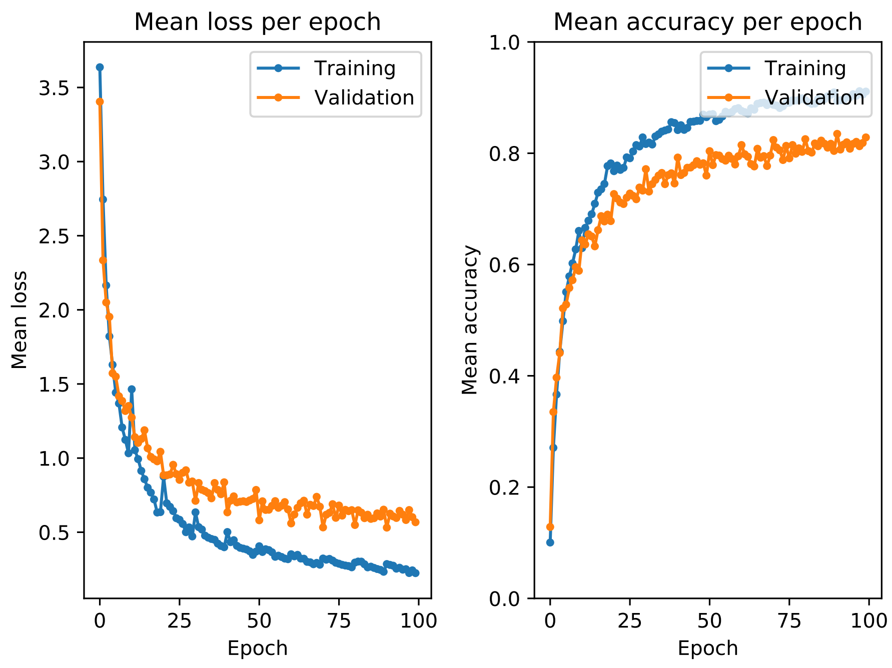

FIGURE 4.7: G7: ResNet18, Data Augmentation, No L2 Penalty
Mean Loss and Mean Accuracy Per Epoch

4.1.1.8 G8: ResNet18, Data Augmentation, L2 Penalty = 0.001

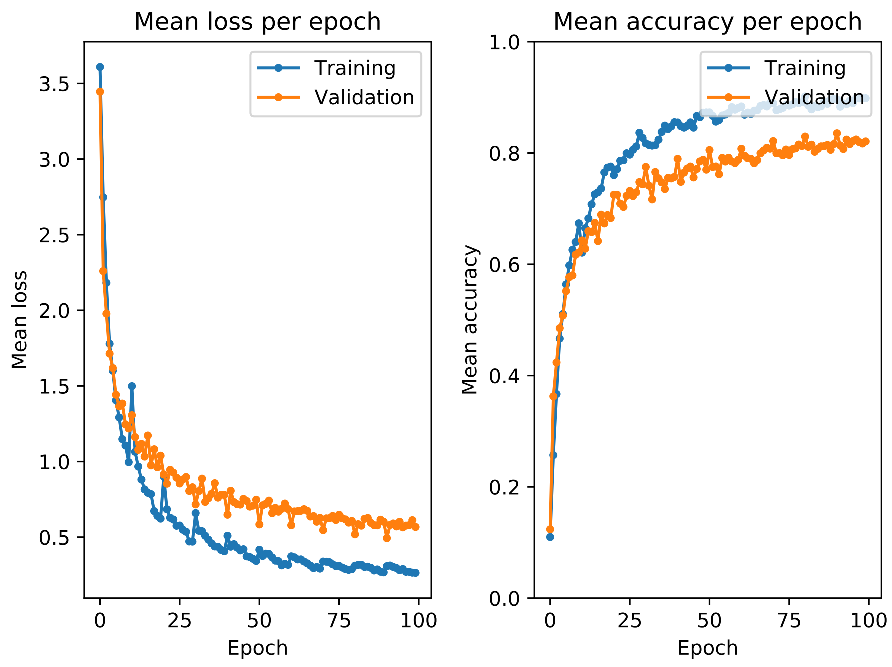

FIGURE 4.8: G8: ResNet18, Data Augmentation, L2 Penalty = 0.001
Mean Loss and Mean Accuracy Per Epoch

4.1.1.9 G9: ResNet18, Data Augmentation, L2 Penalty = 0.005

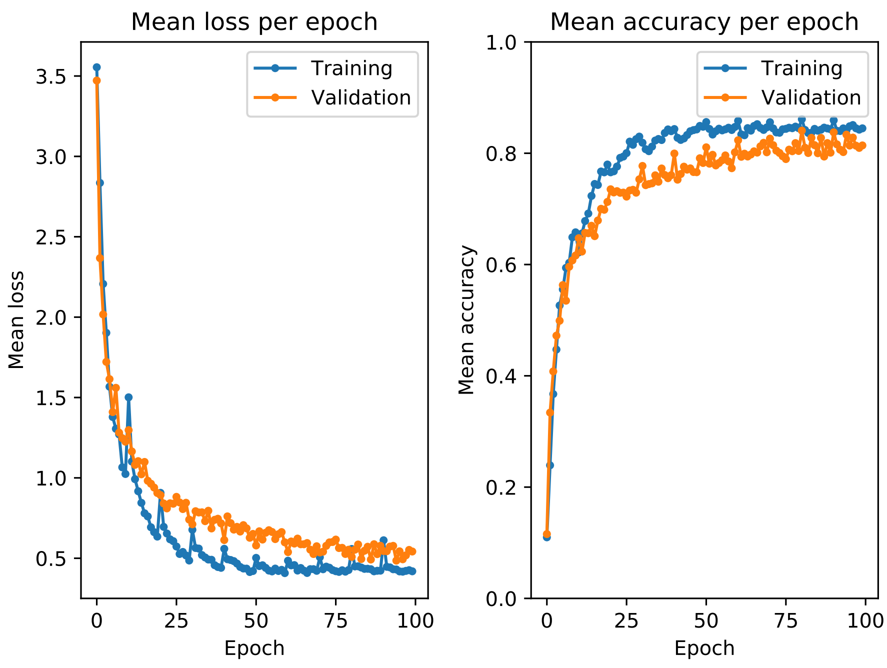

FIGURE 4.9: G9: ResNet18, Data Augmentation, L2 Penalty = 0.005
Mean Loss and Mean Accuracy Per Epoch

4.1.1.10 G10: ResNet18, Data Augmentation, L2 Penalty = 0.008

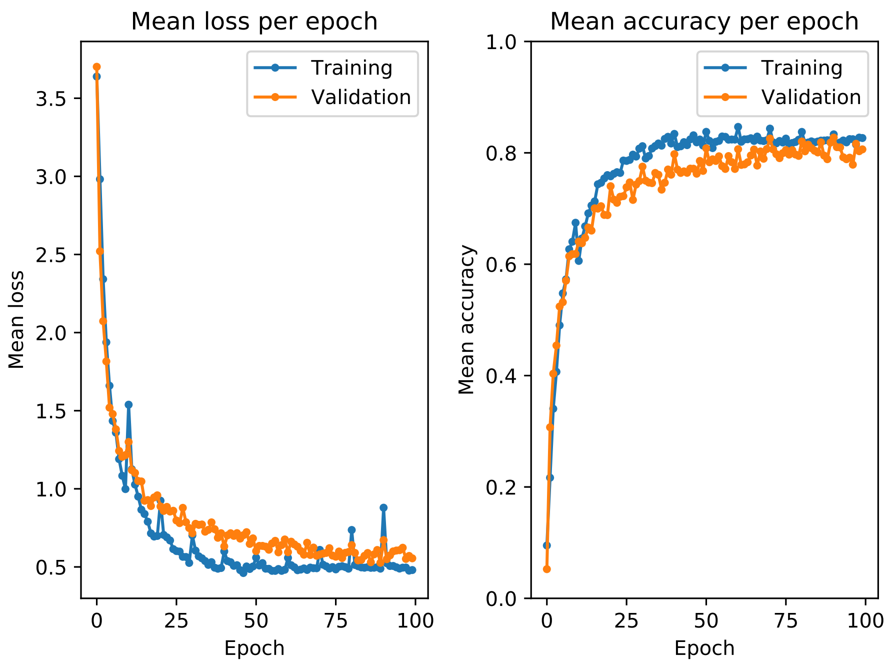

FIGURE 4.10: G10: ResNet18, Data Augmentation, L2 Penalty = 0.008
Mean Loss and Mean Accuracy Per Epoch

4.1.1.11 Final Test Accuracy

Experiment		**Final Test Accuracy**	
Data Augmentation	**L2 Penalty**	**SqueezeNet 1.1**	**ResNet18**
No	**0**	0.2913	0.8145
Yes	**0**	0.4544	0.8091
Yes	**0.001**	0.3563	0.8056
Yes	**0.005**	0.3775	0.7967
Yes	**0.008**	0.2065	0.7997

Table 4.1: Generalization Experiments Test Accuracy Results

4.1.2 Discussion

4.1.2.1 SqueezeNet 1.1

By observing the G1 experiment curve for SqueezeNet 1.1 with no data augmentation, we can see that the SqueezeNet 1.1 model tends to over-fit on the training set after around 100 epochs, since the training loss significantly decreases while the validation loss significantly increases.

By observing the difference between the G1 and G2 experiment curves, we see that enabling data augmentation on the training set significantly reduces over-fitting issues. We can also see an increase in final test accuracy from 0.2913 to 0.4544 by simply enabling the data augmentation, which is a significant improvement.

Adding a low L2 penalty regularization factor of 0.001 can further decrease over-fitting as shown in the figure of the G3 experiment, but also decreases the final test accuracy from 0.4544 to 0.3563. Increasing the value of the L2 penalty factor to a moderate value of 0.005 in experiment G4 and a high value of 0.008 in experiment G5 further decreases over-fitting, but also to the cost of sacrificing test accuracy, which significantly reduces due to moderate and high regularization factors. As shown in the G5 experiment figure, the training and validation loss and accuracy curves overlap each other, which proves that over-fitting is completely eliminated when a high regularization factor of 0.008 is used.

4.1.2.2 ResNet18

By observing the G6 experiment curve with no data augmentation, we can see that the training loss quickly drops to values close to zero and, at the same time, the training accuracy quickly rises to a value of 1. This indicates a strong over-fitting issue on the training set. However, the final test accuracy is still high at a value of 0.8145, which means that the ResNet18 architecture is bound to generalize well even in a situation where is has been training to over-fit on the training set. Our interpretation of this fact is that the task of image classification of the EMNIST 'balanced'-split dataset is quite an easy task for a sophisticated deep learning model such as ResNet18, and it can therefore easily generalize even when in an over-fitting scenario.

By adding data augmentation, as seen on the G7 experiment figure, the over-fitting issue is greatly decreased, as the training loss decreases more smoothly while the training accuracy increases more smoothly. However, the final test accuracy as shown in Table 4.1 shows a slight decrease from 0.8145 to 0.8091 when compared to experiment G6. This can be explained by the fact that the number of epochs is limited. We can observe an upward trend on the validation accuracy curve on the G7 experiment curve, therefore we expect that with more training epochs, the final test accuracy obtained with data augmentation would be higher than without data augmentation. This would need to be confirmed in a further analysis.

The three next experiment curves, G8, G9 and G10, show the impact of respectively low, moderate and high L2 penalty regularization factors. When comparing G7 and G8, we note that the curves are very similar and that the test accuracy is very close, respectively 0.8091 and 0.8056, therefore we conclude that a low regularization factor on the ResNet18 model has little to no impact. However, with a moderate regularization factor of 0.005 for experiment G9, the training and validation curves tend to get closer, which shows that over-fitting is slightly reduced when increasing the regularization factor. This is further shown by the G10 experiment, in which the training and validation curves get even closer. We do note that all three regularization factors tested in G8, G9 and G10 showed a slight reduction in final test accuracy of under 1% when compared to the G7 experiment.

4.1.2.3 Conclusion

To conclude, we can observe that the ResNet18 model is less sensitive to L2 penalty than SqueezeNet 1.1. However, we do note that by increasing the regularization factor, the training and validation curves then to get similar. This indicates that L2 penalty helps to reduce over-fitting on the training set.

Our hypothesis is that for a large number of active training iterations, the solutions with data augmentation in combination with moderate regularization tend to perform better, as observed on the experiment curves and test accuracy results.

4.2 Pretraining

4.2.1 Results

4.2.1.1 SqueezeNet 1.1, Pretained vs. Not Pretrained

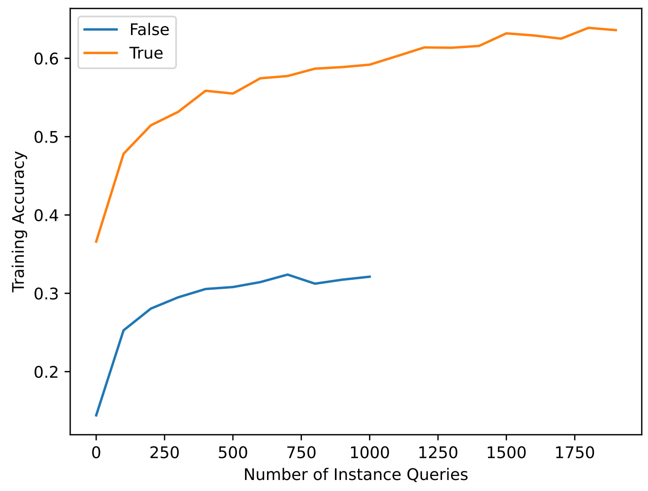

FIGURE 4.11: Accuracy scores on the 2nd validation set for the pretrained and not pretrained SqueezeNet 1.1

4.2.1.2 ResNet18, Pretained vs. Not Pretrained

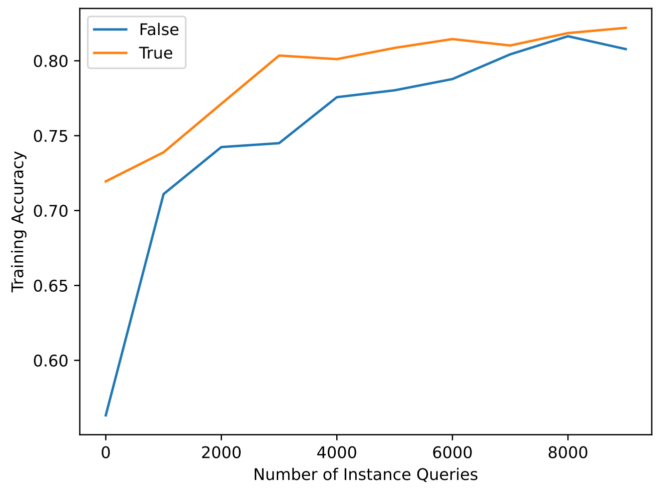

FIGURE 4.12: Accuracy Scores on the 2nd Validation Set for the Pretrained and Not Pretrained ResNet18

4.2.1.3 Final Test Accuracy

Experiment	**Final Test Accuracy**	
Pretraining	**SqueezeNet 1.1**	**ResNet18**
No	0.3206	0.8004
Yes	0.6286	0.8200

Table 4.2: Pretraining Experiments Test Accuracy Results

4.2.1.4 Fictive Time Costs

In the next table, we present the time costs (in minutes) to achieve the accuracy goal set in Table 3.6. The values were calculated using equation 3.1:

$$cost = C_L \cdot (n_{start} \cdot nb\_classes) + KC_T + \sum_{i=1}^{K-1} C_L \cdot \eta$$

with a training time cost proportional to the labeling time cost ($C_T = \lambda C_L$). Note also that K is the number of active learning loops that have been performed to reach the goal.

	SqueezeNet 1.1		**ResNet18**	
	Pretraining		Pretraining	
λ	Yes ($K = 1$)	No ($K = 11$)	Yes ($K = 3$)	No ($K = 7$)
15	1015	2165	5145	9205
30	1030	2330	5190	9310
60	1060	2660	5280	9520

Table 4.3: Time Costs (minutes), considering that labeling an image takes 1 minute ($C_L = 1$) and training takes λ times more time.

4.2.2 Discussion

4.2.2.1 SqueezeNet 1.1

As illustrated in Figure 4.11, the pretrained model took a considering step ahead of its peer directly at the start. Because of the lack of its performance, the model trained from scratch observed 3 consecutive active learning rounds without accuracy improvement and hence had its training ended prematurely. Hence, we see in this case that weights trained on ImageNet provide a better starting point than random weights. We raise the hypothesis that it is due to the resemblance between the CIFAR10 and ImageNet classification task.

The final test accuracy scores in Table 4.2 also goes in the same way, showing a final test accuracy for the pretrained model that is almost twice as high as its opponent.

4.2.2.2 ResNet18

As we can observe in Figure 4.12, the pattern of the curves is similar but a bit different from the one with SqueezeNet 1.1. The pretrained ResNet18 model shows a significant gain of performance at start in opposition to its not pretrained version. However, the model trained from randomly initialized weights rapidly increases its accuracy not far from its pretrained rival. In this case, we support the hypothesis that the performance similarities of both models within only few instance queries (i.e active learning loops), might be due to the simplicity of the task compared to the capacity of the ResNet18 and the difference between the EMNIST classification task that contains only monochrome hand-writttten digits and letters and the ImageNet classification task that contains colored RGB object images.

The final test accuracy scores in Table 4.2 show an improvement of two percentage points for the pretrained ResNet18.

4.2.2.3 Conclusion

Our results suggest that using a pretrained model in an active learning context can be benefical to get accuracy gains on both short and long terms. It can hence provide better results in a time efficient way (see Figure 4.3). Nonetheless, we saw higher benefits when the weights were pretrained for a similar task and the model to train had less capacity.

4.3 Query Strategy

4.3.1 Results

4.3.1.1 SqueezeNet 1.1, Uncertainty Sampling vs. Random Sampling

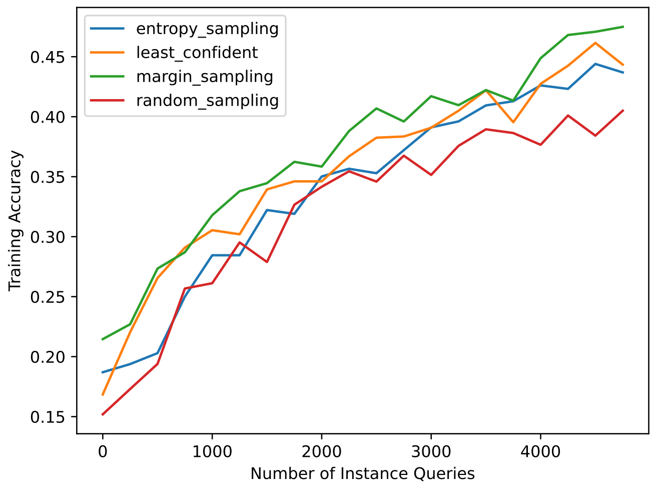

FIGURE 4.13: SqueezeNet 1.1: Accuracy scores for four selection strategies: least confident, margin sampling, entropy sampling and random sampling

4.3.1.2 ResNet18, Uncertainty Sampling vs. Random Sampling

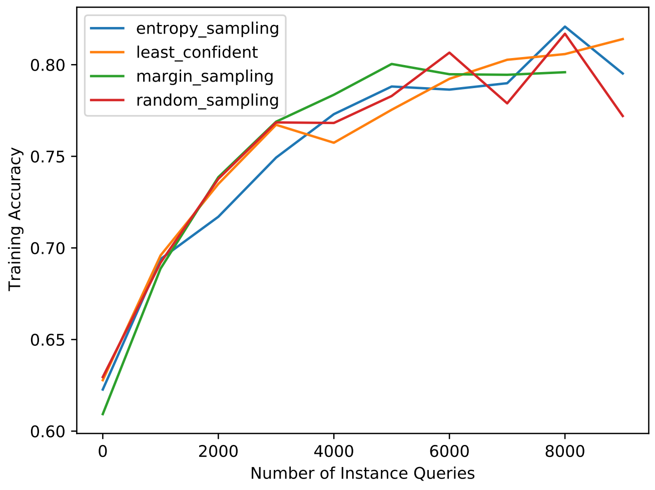

FIGURE 4.14: ResNet18: Accuracy scores for four selection strategies: least confident, margin sampling, entropy sampling and random sampling

4.3.1.3 Final Test Accuracy

Experiment	**Final Test Accuracy**	
Query Strategy	**SqueezeNet 1.1**	**ResNet18**
Least Confidence	0.4511	0.80947
Margin Sampling	0.4783	0.79149
Entropy Sampling	0.4449	0.78941
Random Sampling	0.4058	0.77112

TABLE 4.4: Query Strategy Experiments Test Accuracy Results

4.3.1.4 SqueezeNet 1.1, Margin Sampling vs. Random Sampling

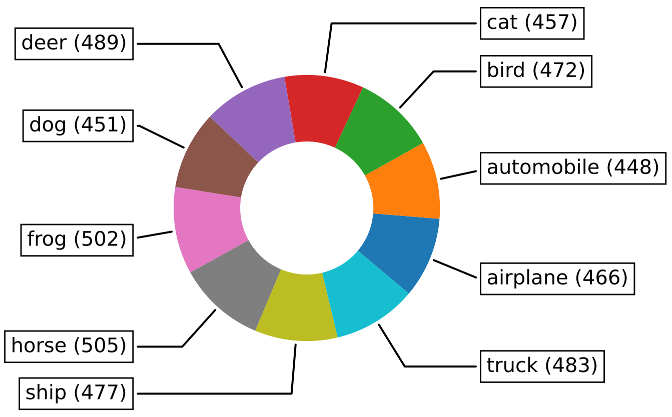

FIGURE 4.15: SqueezeNet 1.1: Class labeling distribution for Margin sampling strategy

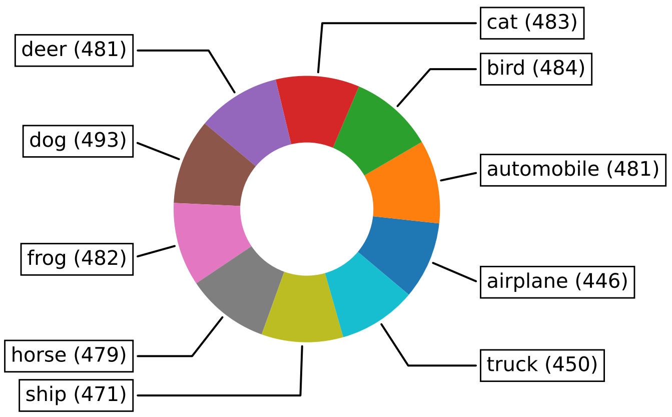

FIGURE 4.16: SqueezeNet 1.1: Class labeling distribution for Random sampling strategy

4.3.1.5 ResNet18, Least Confidence vs. Random Sampling

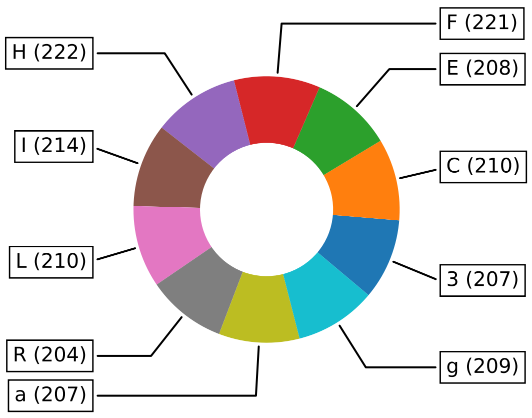

FIGURE 4.17: ResNet18: Class labeling distribution for Least Confidence strategy

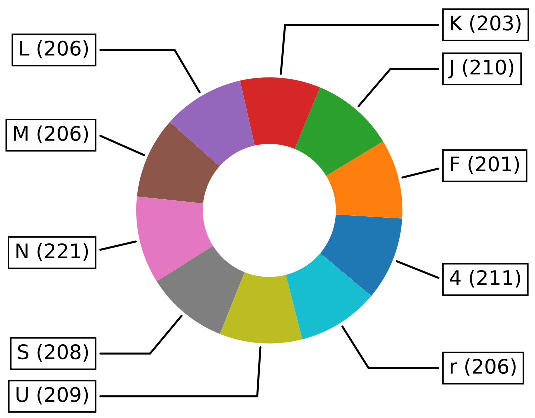

FIGURE 4.18: ResNet18: Class labeling distribution for Random sampling strategy

4.3.2 Discussion

4.3.2.1 SqueezeNet 1.1

Looking at the query strategy experiment curve for SqueezeNet 1.1 in Figure 4.13, we can see that all of the active learning strategies had a roughly equal performance. Besides, they had a better learning curve than the passive learning one. Additionally, the Margin Sampling seems to be slightly better than the Least Confidence and Entropy Sampling methods. This is not very surprising since the SqueezeNet 1.1 model was trained on the CIFAR10 dataset. In other words, having a few classes made the margin sampling performance quite satisfactory even though it only uses the first and second maximum probabilities.

The final test accuracy scores in Table 4.4 show that all of the active learning strategies had better accuracy scores than random sampling (passive learning). Besides, we can see an improvement of seven percentage points for the Margin Sampling strategy when compared to Random Sampling.

Figures 4.15 and 4.16 show class labeling distribution of the best uncertainty sampling strategy ”Margin Sampling” and the ”Random Sampling” strategy. We can see that when using the Margin Sampling method, the classes: **frog**, **horse**, **airplane** and **truck** required more labels when compared with the Random Sampling labeling distribution. On the other hand, the classes: **dog**, **cat** and **automobile** were the most labeled by the random sampling strategy when compared to the Margin Sampling labeling distribution.

4.3.2.2 ResNet18

Moving on to the query strategy experiment curve for ResNet18 in Figure 4.14, we can see that initially all of the strategies implemented had very comparable learning curves. By the time we hit 8000 labeled instances, the Entropy Sampling strategy had the best training accuracy, as its formula takes advantage of all class probabilities, which is very advantageous when training a dataset with numerous classes (EMNIST dataset, using the ’balanced’ split: 62 classes).

The final test accuracy scores in Table 4.4 show that all of the active learning strategies had close but better accuracy scores than Random Sampling. Moreover, it is interesting to see that the Least Confidence and the Margin Sampling strategies had close and even slightly better scores than Entropy Sampling. It makes them very assuring methods taking into account the high running efficiency.

Figures 4.17 and 4.18 show class labeling distributions of the top-10 labeled classes for the best active learning strategy ”Least Confidence” or the ”Random Sampling” strategy. We can see that the classes: **H**, **I**, **R**, **a**, **E**, **C**, **3**, **g** were labeled the most by the Least Confidence strategy, but not by the random sampling strategy.

4.3.2.3 Conclusion

Finally, we can conclude that the used active learning query strategies showed better scores and more stable learning curves when compared with the random sampling strategy.

4.4 Further Analyses

In this last section we explore another active learning related subject that we would like to investigate next.

4.4.1 Labeling Rate

As described in section 2.1.1, the number of new items labeled within each active learning loop was set to be fixed in all of our experiments. Here, we propose a labeling rate scheduler that could be further evaluated and enhanced in future experiments.

Let us suppose that we have a monotone increasing function

$$a : \mathbb{Z}^+ \to (0, 1)$$

taking the number of items in the **labeled train set** (n) as input and returning the accuracy of $\mathcal{M}$ over the **labeled test set**. Our goal would be to evaluate $\frac{da}{dn}$, the increasing of accuracy for a single item added to the **labeled train set** knowing the current number of items in it. This function might be complex and dependent on the model $\mathcal{M}$ and many other factors such as the items in the **labeled train set**. However, denoting $a^{(i)}$ and $\eta^{(i)}$ respectively the accuracy obtained and the requested number of new labeled items in the active learning loop i, we can estimate $\frac{da}{dn}$ as follows:

$$\frac{da}{dn} = \lim_{\Delta \to 0} \frac{a(n+\Delta) - a(n)}{\Delta} \approx \frac{a^{(i)} - a^{(i-1)}}{\eta^{(i-1)}} = \delta \tag{4.1}$$

Hence considering that our goal is to achieve an accuracy of A with our model $\mathcal{M}$ and we just finished active learning loop $i-1$. Instead of using a fixed labeling rate and setting $\eta^{(i)}=\eta^{(i-1)}$, we can estimate the optimal number of new labels that should be requested with the following calculation

$$\eta^{(i)} = \left\lceil \frac{A - a^{(i)}}{\delta} \right\rceil \tag{4.2}$$

However, the monotone strictly increasing function assumption is strong and might not be respected in a real experiment, meaning that it is possible to observe $a^{(i)} - a^{(i-1)} \leq 0$ for a certain i. To fix this problem, we propose the following implementation

where ϵ is a small float used to replace the accuracy scores difference if $a^{(i)} - a^{(i-1)} \leq 0$, R is the maximal number of items that can be labeled within a single active learning loop and g is the gain of accuracy we expect observe when training with the next additional

```
Algorithm 1: Labeling rate update
Input: $\eta^{(i-1)}$, $a^{(i-1)}$, $a^{(i)}$, $\epsilon$, $R$, $g$
Output: $\eta^{(i)}$
1 assert $\epsilon, R > 0$
2 $\delta = \max(\epsilon, a^{(i)} - a^{(i-1)})/\eta^{(i-1)}$
3 gain $= min(g, 1 - a^{(i)})$
4 return $\left\lceil \min\left(R, \frac{gain}{\delta}\right) \right\rceil$
```

FIGURE 4.19: Labeling rate scheduler pseudo code

items. Naturally, one could think that g should be set to $1 - a^{(i)}$, however the number of new requested labels might grow always to a value larger than R.

Conclusion

In conclusion, this project was performed to evaluate active learning methods for deep neural networks. It allowed the comparison of different active learning selection criteria and methods on multiple datasets for different neural network architectures.

This project enables us to conclude that active learning is a valid approach to solve the high resource requirement of supervised learning techniques, notably in image classification. It can be used to significantly reduce the number of manual labelled inputs required to obtain similar performance as fully-labelled input sets. This implementation of semi-supervised learning is an excellent choice when the costs of labelling are high in cases where experts are costly or rare, or when the input sets are large.

Appendix A

Class diagram

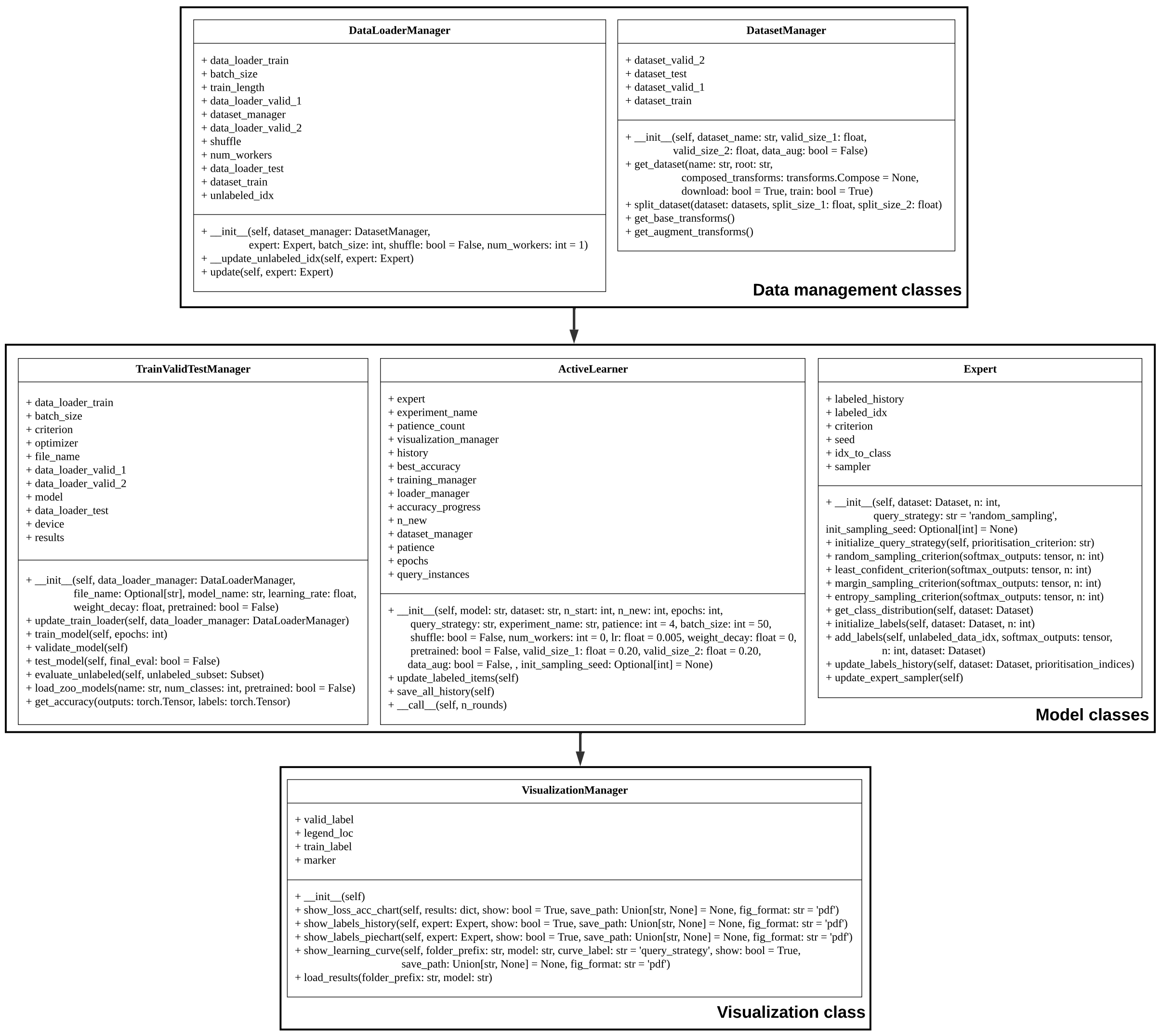
DataLoaderManager
+ data_loader_train
+ batch_size
+ train_length
+ data_loader_valid_1
+ dataset_manager
+ data_loader_valid_2
+ shuffle
+ num_workers
+ data_loader_test
+ dataset_train
+ unlabeled_idx
+ __init__(self, dataset_manager: DatasetManager, expert: Expert, batch_size: int, shuffle: bool = False, num_workers: int = 1)
+ __update_unlabeled_idx(self, expert: Expert)
+ update(self, expert: Expert)
DatasetManager
+ dataset_valid_2
+ dataset_test
+ dataset_valid_1
+ dataset_train
+ __init__(self, dataset_name: str, valid_size_1: float, valid_size_2: float, data_aug: bool = False)
+ get_dataset(name: str, root: str, composed_transforms: transforms.Compose = None, download: bool = True, train: bool = True)
+ split_dataset(dataset: datasets, split_size_1: float, split_size_2: float)
+ get_base_transforms()
+ get_augment_transforms()
Data management classes
TrainValidTestManager
+ data_loader_train
+ batch_size
+ criterion
+ optimizer
+ file_name
+ data_loader_valid_1
+ data_loader_valid_2
+ model
+ data_loader_test
+ device
+ results
+ __init__(self, data_loader_manager: DataLoaderManager, file_name: Optional[str], model_name: str, learning_rate: float, weight_decay: float, pretrained: bool = False)
+ update_train_loader(self, data_loader_manager: DataLoaderManager)
+ train_model(self, epochs: int)
+ validate_model(self)
+ test_model(self, final_eval: bool = False)
+ evaluate_unlabeled(self, unlabeled_subset: Subset)
+ load_zoo_models(name: str, num_classes: int, pretrained: bool = False)
+ get_accuracy(outputs: torch.Tensor, labels: torch.Tensor)
ActiveLearner
+ expert
+ experiment_name
+ patience_count
+ visualization_manager
+ history
+ best_accuracy
+ training_manager
+ loader_manager
+ accuracy_progress
+ n_new
+ dataset_manager
+ patience
+ epochs
+ query_instances
+ __init__(self, model: str, dataset: str, n_start: int, n_new: int, epochs: int, query_strategy: str, experiment_name: str, patience: int = 4, batch_size: int = 50, shuffle: bool = False, num_workers: int = 0, lr: float = 0.005, weight_decay: float = 0, pretrained: bool = False, valid_size_1: float = 0.20, valid_size_2: float = 0.20, data_aug: bool = False, , init_sampling_seed: Optional[int] = None)
+ update_labeled_items(self)
+ save_all_history(self)
+ __call__(self, n_rounds)
Expert
+ labeled_history
+ labeled_idx
+ criterion
+ seed
+ idx_to_class
+ sampler
+ __init__(self, dataset: Dataset, n: int, query_strategy: str = 'random_sampling', init_sampling_seed: Optional[int] = None)
+ initialize_query_strategy(self, prioritisation_criterion: str)
+ random_sampling_criterion(softmax_outputs: tensor, n: int)
+ least_confident_criterion(softmax_outputs: tensor, n: int)
+ margin_sampling_criterion(softmax_outputs: tensor, n: int)
+ entropy_sampling_criterion(softmax_outputs: tensor, n: int)
+ get_class_distribution(self, dataset: Dataset)
+ initialize_labels(self, dataset: Dataset, n: int)
+ add_labels(self, unlabeled_data_idx, softmax_outputs: tensor, n: int, dataset: Dataset)
+ update_labels_history(self, dataset: Dataset, prioritisation_indices)
+ update_expert_sampler(self)
Model classes
VisualizationManager
+ valid_label
+ legend_loc
+ train_label
+ marker
+ __init__(self)
+ show_loss_acc_chart(self, results: dict, show: bool = True, save_path: Union[str, None] = None, fig_format: str = 'pdf')
+ show_labels_history(self, expert: Expert, show: bool = True, save_path: Union[str, None] = None, fig_format: str = 'pdf')
+ show_labels_piechart(self, expert: Expert, show: bool = True, save_path: Union[str, None] = None, fig_format: str = 'pdf')
+ show_learning_curve(self, folder_prefix: str, model: str, curve_label: str = 'query_strategy', show: bool = True, save_path: Union[str, None] = None, fig_format: str = 'pdf')
+ load_results(folder_prefix: str, model: str)
Visualization class

Figure A.1: Class Diagram of workflow components

Bibliography

[1] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

[2] Iandola, F. N., Moskewicz, M. W., Ashraf, K., Han, S., Dally, W. J., and Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360.

[3] Settles, B. (2009). Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison.