Image classification of Alzheimer's Dataset

IKT452

Lars-Erik Bakkland Moi, Oskar Eftedal Markussen
University of Agder
Faculty of Engineering and Science)
Grimstad, Norway
Group 13
lemoi18@uia.no, OskarEM@uia.no

***Abstract*—In this project we decided to train a variation of different models on Alzheimer dataset, to first look at which models is capable of the highest results. Further we decided to improve on one or two models. This was done by testing the augmentations like rescale, rotate, width shifting, height shifting, brightness adjustment, zooming, and shearing. Then we tried to use filters such as Sobel edge detection and Gabor filtering, and lastly we tried to use some other training techniques to deal with the class imbalance in our dataset. The techniques tested was class weighting, k-fold cross-validation and lastly oversampling. The result was more in favor of improving the class imbalance, rather then adding possible pattern detection such as Sobel and Gabor. As such, our final and best results from the models was with the usage of oversampling in ResNet and DenseNet, which resulted in 88% and 98.6 %. This was then used in an voting system to improve its accuracy, and the final result of the voting was 98.9 %. In conclusion, we observed that DenseNet was the one better at handling the classification of the images on an overall basis, but could be improved upon by utilising voting.**

I. Workload share

We each experimented on the different models and wrote on the rapport. The workload was therefor equality shared among us.

II. Introduction

In this project we will be looking at the image classification on a Alzheimer Dataset, to predict at which stage the patient is at, by looking at the Magnetic resonance imaging (MRI) of said patients brain activity. Following this task we have decided to implement a diversified number of models to try looking at which model would be capable of getting the best prediction result. This involved also looking at the different augmentations, and how they might effect the overall results of our best fitted models. Lastly we will be looking at the possibility of adding more models together in a voting system to try bettering our results. This usage of multiple models used to create a voting system, was inspired by a previous project conducted in the subject "IKT457-G 23H Learning Systems".

We decided to approach this problem because of its many applications for future use. Trying out different image classification models is something we consider a valuable lesson for further predictions problems, where we need to decide on which model we would want to use. The reason we chose the Alzheimer Dataset as our dataset to be trained on is because it seem interesting and a challenge considering its lesser details between each classifications. There was four classifications in the dataset we chose, which was Mild Demented, Moderate Demented, Non Demented and Very Mild Demented.

III. Background/Related Work

A. Convolutional neural network (CNN)

According to the paper "Understanding of a convolutional neural network" [1] a Convolutional Neural Networks (CNN) is a type of deep learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other. The importance of CNNs comes from their ability to develop an internal representation of a two-dimensional image. This allows them to learn, understand, and recognize spatial hierarchies in data, where higher-level features are derived from lower-level ones.

CNNs are particularly good at tasks that require recognizing patterns or objects in images. They perform well in image and video recognition and image classification. The reason CNNs are so effective for these tasks is due to their architecture, which mirrors the connectivity pattern of neurons in the human brain and thus provides a means of processing data with grid-like topology. For instance, in image recognition, CNNs can automatically detect important features without any human supervision, using learned filters that significantly reduce the number of parameters in the network.

The ability of CNNs to perform feature extraction means fewer pre-processing steps are required, reducing the complexity of the data and enhancing the efficiency of learning. They achieve this through a series of convolutional layers that apply convolutional filters, pooling layers that reduce dimensionality, and fully connected layers that interpret the features extracted by the convolutions.

B. Inception v3 model

The Inception-v3 model, as discussed in the paper "Rethinking the Inception Architecture for Computer Vision" [2], is a convolutional neural network architecture that builds upon the design of the earlier Inception models by introducing several

architectural enhancements. These include factorization into smaller convolutions to reduce computational cost while maintaining model effectiveness, and the use of auxiliary classifiers that help in optimizing the training process by providing regularization.

Inception-v3 aims to efficiently use computational resources by factorizing convolutions into smaller, more manageable operations. This reduces the number of computations required and helps the network perform well even under constraints of limited computational capacity, making it suitable for environments where resources are scarce, like mobile devices. The architecture also employs label smoothing as a form of regularization to prevent the model from becoming too confident in its predictions, which helps improve the generalizability of the model to new data.

C. DenseNet 121

As explained by the paper "Densely connected convolutional networks" [3] DenseNet-121 is a convolutional neural network architecture that stands out for its dense connectivity pattern. It is part of the DenseNet (Densely Connected Convolutional Networks) family that was introduced to improve the flow of information and gradients throughout the network, which helps in alleviating the vanishing gradient problem, strengthening feature propagation, and reducing the number of parameters.

DenseNet architectures are unique because each layer is connected to every other layer in a feed-forward fashion. For each layer, the feature maps of all preceding layers are used as inputs, and its own feature maps are used as inputs into all subsequent layers. This dense connectivity pattern leads to considerable improvements in efficiency and effectiveness in learning; the layers are very narrow (e.g., 12 filters per layer) and the network requires fewer parameters than traditional convolutional networks with a similar depth.

D. Residual neural network (ResNet)

The ResNet (Residual Network) model is, according to the paper "Why ResNet Works? Residuals Generalize" [4], an advancement in the architecture of convolutional neural networks, introduced primarily to handle the challenge of training very deep networks. As networks grow deeper, they often face the problem of vanishing gradients, where the gradients of the network's parameters become very small during backpropagation, making the network hard to train and often leading to poorer performance. To address this, ResNet introduces "residual blocks" with skip connections that allow gradients to flow directly through the network without passing through multiple layers of transformations.

The architecture of ResNet is designed such that the input to a residual block is added to the output of that block. This means that each block learns the residual functions with reference to the layer inputs, instead of learning unreferenced functions. For example, instead of learning a direct mapping of x to $H(x)$, each two-layer block in ResNet learns the residual mapping $F(x) = H(x) - x$. The original input x is then added back to the output of the network block, facilitating a kind of "shortcut" for the network training process. This setup encourages the stacked layers to learn perturbations of the identity mapping, rather than complete transformations, which stabilizes the gradient flow across the network, allowing for much deeper networks to be trained effectively.

E. K-fold cross-validation

K-fold cross-validation involves dividing the whole data set into k number of subsets, where you use $k-1$ subset in the training split and the last subset as a test set. This is done k times so that each subset is used as a test set once. Each fold can then be averaged and be used together to provide a more reliable estimate of model performance. [7]

F. Oversampling

Oversampling is used when there is an imbalance in the dataset, oversampling increases the minority class by either duplicating them or generating synthetic samples. In the project we used SMOTE(Synthetic Minority Oversampling Technique) which generates synthetic samples from the existing samples. [5]

G. Sobel edge detection

Sobel edge detection is a technique in image processing for detecting edges within an image. It functions by using two distinct convolution filters, one to detect horizontal variations and the other to capture vertical variations. These filters emphasize areas with high spatial frequency that correspond to edges. The Sobel operator usually determines the intensity gradient of the image at each pixel, indicating the direction with the sharpest shift from light to dark and the rate of change in that direction. This method reveals how suddenly or gradually the image varies at a given point, effectively identifying edges. [11]

H. Gabor filtering

Gabor filtering involves using Gabor filters, which are bandpass filters that select certain frequencies and directions in localized regions of an image. These filters are particularly effective in texture analysis, as they can capture properties such as spatial frequency, spatial localization, and orientation selectivity. Gabor filters are defined by a harmonic function multiplied by a Gaussian function, allowing for the extraction of edge and texture information from an image at various scales and orientations. This makes them highly useful in applications like image segmentation, texture recognition, and feature extraction in computer vision tasks. [10]

I. Model voting system on classification

A model voting system on classification, also known as ensemble voting or simply voting, is a machine learning ensemble technique that involves combining multiple classification models to improve predictive performance. The idea is based on the principle that different models may capture different patterns or aspects of the data, so combining their outputs can lead to more accurate and reliable predictions. [6]

IV. Methodology

The approach that was chosen for this project, was to first implement multiple models and train them based on the standard dataset. From this we chose one or two models that seemed to have the most improved results and built on to them by utilising augmentations to increase the dataset, filters to improve possible pattern recognition, and lastly use some techniques to optimize the problems we observed when training the models.

As previously mentioned, we chose a set of models to train our standard dataset on. The models in question was the CNN, the Inception model, the DenseNet model and lastly the ResNet model. Following this we came to the conclusion to improve the models with the best results. In our case this was the ResNet and DenseNet models. The ResNet and DenseNet models was then tried to be optimized on, by increasing the dataset with seven primary augmentations which was to rescale, rotate, width shifting, height shifting, brightness adjustment, zooming, and shearing a percentage of the images in the dataset. This was done to improve the quantity of the training dataset. Following this we also tried to incorporate some filters to further improve our learning and dataset quantity. The filters we decided to use on different models was the Sobel edge detection and the Gabor filter, with 8 kernels. We also tried Histogram of gradients(HoG) and then passing them alongside with the images, however this yielded in an not so satisfactory results and was discarded.

After augmentation and application of filters on the dataset, we observed that there was a possibility for further training by adding some statistical methods such as K-fold cross-validation, class weights and oversampling on the minority class. By adding these methods we discovered that we didn't initially need too add much augmentation and filtering on our images, as our best result ended up being train on the original dataset with no change to it other than oversampling.

When this was done, we now had a set of models that we could use and try to predict in a voting based system. Most of our training was logged using the library "Wandb".

V. Experimental Results and Analysis

As we tried to find an optimized model to further improve upon, as previously mentioned, the choice was between CNN, Inception, DenseNet and ResNet. The following five sub-chapters describe our first initial dataset and training of the different models and the initial results we got without the augmentations of our dataset.

A. Dataset

The dataset consists of 4 classes, where each category is an images of an MRI scans. This MRI scan was done on the brain of people with and without the diagnoses of Alzheimer's. These are the resulting classes

- MildDemented
- ModerateDemented
- NonDemented
- VeryMildDemented

MRI scans are sequential in nature and therefore there are multiple images from the same patient from different states of the MRI scans, examples on how these look are in figure 1. Upon further analysis, we can determine that there is a significant class imbalance between ModerateDemented class and the other classes as seen in figure 2. Later in the chapters we will also look at some techniques that we employed to try generalize our models.

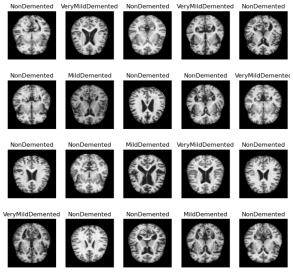

Fig. 1: Samples from the dataset

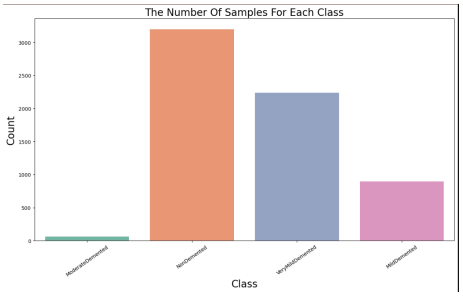

Fig. 2: The number of samples for each class

B. Results of training CNN

The CNN model was observed to be slow and not very accurate as it gave one of the lowest result we got, which was 66% - 68%. This was trained with the learning rate of 0.0005, and was train over a 100 epochs. The batch size that was train at a time was 32, with the dense layer having 256 neurons, 32 filters, max pooling with a pool size of 2 by 2, and dropout layer of 0.2. The model also incorporated a global average pooling in its model architecture. Its activation functions was relu and a softmax at the end. Lastly it also uses Adam optimizer with the specification of categorical_crossentropy. So to summaries, it was a very normal CNN model architecture.

Fig. 3: Example of one of the CNN models trained on the original dataset

From the Figure 3 we observe that the model itself is over-fitting on the training data and cant fully have a stable ascent in validation/test accuracy. We believe this to be because of its class imbalance and lack of generalized data. As the paper "Exploratory Undersampling for Class-Imbalance Learning" [8] explains, an imbalanced class learning could potentially lead to poor generalization, because it can over-fitt to the patterns and characteristics of the dominant classes. Another problem could be that it have insufficient regularization such as low drop out probability, it could also need the usage of regularization techniques such as L1/L2. Another possible improvement that could have been done considering this is the first training of the models, where only the original dataset is used, is that it could benefit from some augmentations or filters to improve its overall accuracy.

This suggestion of class imbalance is further hypothesize by looking at its confusion matrix in figure 4. In the confusion matrix we can see that it predicts much better on the more dominant classes, which could lead to over-fitting or showing that there is an imbalance in the dataset.

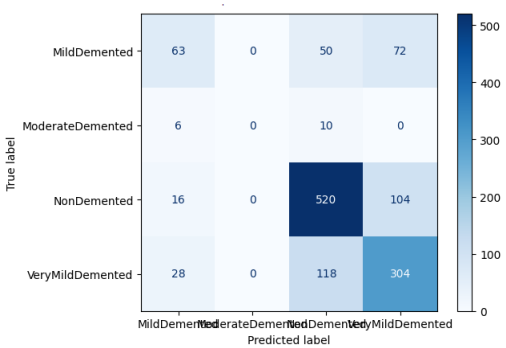

Fig. 4: Example of a confusion matrix describing one of the CNN models trained on the original dataset

C. Results of training Inception model

Following the training of the CNN model, we trained an Inception model on the original dataset. This was slower than the CNN training and the training provided us with worse result of approximately 60% to 62 %. The models we trained used a learning rate of 0.0005, a dense layer with 128 or 256 neurons and a drop out layer of 0.2 to 0.5, the models each gave approximately the same results. Similar to the CNN model it also used global average pooling, Adam optimizer, and used relu and softmax as it activation functions. As the Inception model is a pre-trained model we decided to only unfreeze the 10 last layers of the model and train on them.

Fig. 5: Example of one of the Inception models trained on the original dataset

However despite the worse result given by the Inception model, we can see from the graph in figure 5 that there is less stagnation and especially over-fitting in the inception model than the previously trained CNN. This prompt us to believe there is something wrong with the optimization of the hyper parameters. As we observed that the learning rate of 0.0005 might have been to little for the Inception model, and it could be improved by increasing the learning rate. other possible problems could be that the complexity of the model was too great for the initial size of the dataset, or that it lacks diversity to fully utilise the model properly.

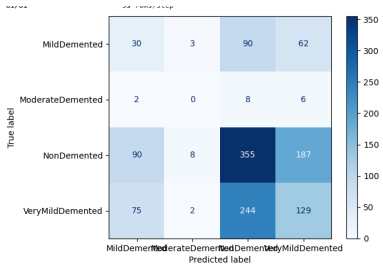

Fig. 6: Example of a confusion matrix describing one of the Inception models trained on the original dataset

With the decreased accuracy, we can also see from the confusion matrix in figure 6 that the model tend to predict the more dominant classes, more and better, this indicates a class imbalance as we have seen previously in CNN. However this is expected as our model have a lot fewer examples of the moderate class compared to the none dementia class. The not so optimal accuracy could also be attributed to the models slow learning and complex architecture.

D. Results of training DenseNet model

The DenseNet model was observed to be the best model we trained, when considering it did not incorporate any

augmentations. The DenseNet model gave us a result from 74% to 76% and it was observed to be trained very fast in comparison to some of the other models such as the CNN. The models we trained incorporated a learning rate of 0.0005, a dense layer with 128 or 256 neurons and a drop out layer of 0.2 to 0.5, the models each gave approximately the same results. Similar to the CNN model it also used global average pooling, Adam optimizer, and used relu and softmax as it activation functions. As the DenseNet is a pre-trained model we decided to only unfreeze the 10 last layers of the model and train on them as well.

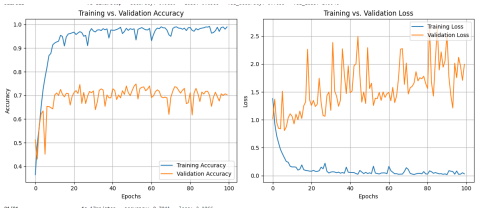

Fig. 7: Example of one of the DenseNet models trained on the original dataset

From the observation of our graph in figure 7, we can see that the validation accuracy is stagnating over time, and that the validation loss continues to increase. As we don't want our loss to increase since this is a descending optimization, we want to look at why this increase happens. We believe this is because of over-fitting of the data. This observation of over-fitting is quite common among our initial models. Other problems this graph could describe, is the model might be too complex relative to the complexity and the amount of data in our dataset. As the DenseNet121 is a complex deep network with a lot of parameters, it has been described to be susceptible to over-fitting. The paper "Deep Architecture based on DenseNet-121 Model for Weather Image Recognition" [9] also recognize this issue and describes the importance of augmentation on the data to increase the amount of training data. Another possible problem could be the learning rate may be too high, and makes the model converge to sub-optimal solutions. This problem could be solved with a descending learning rate based on model accuracy or convergence.

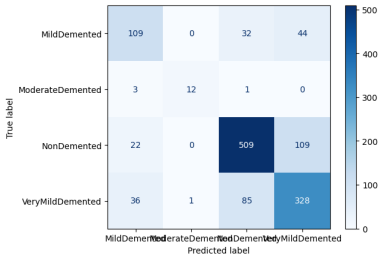

Fig. 8: Example of a confusion matrix describing one of the DenseNet models trained on the original dataset

As we can observe in the confusion matrix in figure 8, the DenseNet is better at predicting the moderate class in comparison to the rest of the previously mentioned models.

E. Results of training ResNet

The ResNet model was shown to be the initial second best fitted model as it provided a result of approximately 66% to 69% accuracy on the original dataset. This small improvement from the CNN model to the ResNet model was not that significant, but it was enough for us view the DenseNet and ResNet as our best options for going further with. This mean we decided to proceed with both the DenseNet and the ResNet model as our models to be improved upon. The hyper parameters that was used during the initial training and testing, of the ResNet, was the same as the mentioned DenseNet, except the learning rate which was lowered to 0.000005 and the batch size was increased to 32. It also included the unfreezing of the 10 last layers as well. The lowering of learning rate to 0.000005 was done because of the jumping in validation accuracy and increase in validation loss during the training of the model.

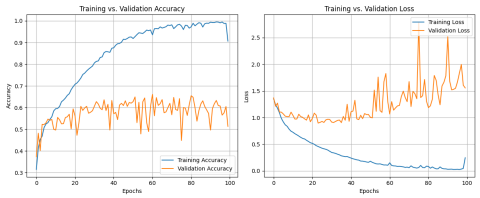

Fig. 9: Example of one of the Resnet models trained on the original dataset

As we can observe from the figure 9, the ResNet model exhibits a lot of jumps in its validation accuracy, indicating potential over-fitting to the training data or sensitivity to the specific distribution of the validation set. These fluctuations suggested that adjustments in the model's learning rate or batch size might be necessary to stabilize performance and improve generalization. We can also see this by looking at the difference in accuracy between the training and validation accuracy similar to the previously described models.

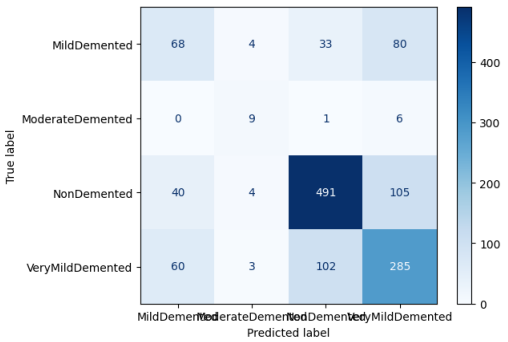

Fig. 10: Example of a confusion matrix describing one of the ResNet models trained on the original dataset

As shown in the confusion matrix, in figure 10, we can observe some of the same as with the DenseNet, which is that the model is capable of predicting the moderate class. Which

is an observation we can not say about the CNN and Inception illustration of the confusion matrix.

F. Additional testing of some class imbalance improvement on the models

As we observed the confusion matrix had problems with class imbalance, we decided to add class weighting to some of our models. The class weighting was set to be balanced and was calculated based on the difference in how many images there was in each class, in comparison to each other. This made the loss function add a bigger weight on the mistakes done to predict the minority class. for some of our models it improve the accuracy slightly, and increased the distribution in the confusion matrix quite significantly. However for some of the models such as the CNN it did not increase but rather decrease its accuracy. However on our better models we decided to not use class weighting, since we used oversampling instead.

G. k-fold with DenseNet model

As DenseNet was our best model so far we wanted to see if we could improve its training by using k-fold cross-validation on the original dataset. This gave us a increase of approximately 2% in the accuracy and it went from 74 - 76 % to 76 -78 %. This increase was quite minor and as we can observe in the confusion matrix, in figure 11, the overall generalization was worse even if the accuracy was better. The dataset was divided in to 5 folds each giving their own model which we used a average voting system to achieve the result.

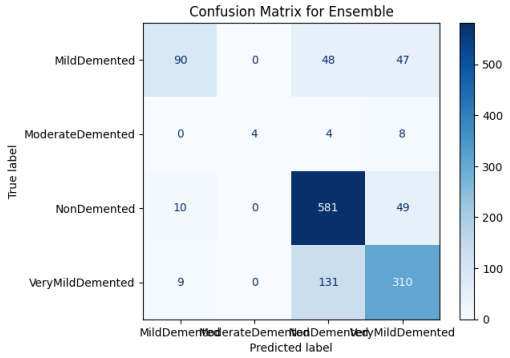

Fig. 11: Example of a confusion matrix describing one of the DenseNet models trained with k-fold cross-validation on the original dataset

H. ResNet with sobel and aumentations

The Sobel edge detection improved upon the base model by around 10% increase. The ResNet with sobel and augmentations of rescale, rotate, width shifting, height shifting, brightness adjustment, zooming, and shearing ended up on to around 78% validation accuracy, however this was on epoch 1639. We don't know if this kind of improvement would have happened on the other models if they also was trained for that many epochs, which is why we cant conclude with this being an improvement on the technique, but more on our results instead.

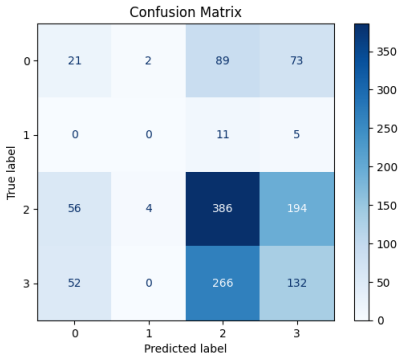

Fig. 12: Example of a confusion matrix describing the Sobel augmentation model trained on the original dataset

As shown in the confusion matrix, in figure 12, the model still had problems with prediction on the minority classes.

I. ResNet with gabor and k-fold

After considering what more could be beneficial for our model, we ended up trying the Gabor filter with a k-fold cross-validation training technique. The Gabor filter was set to employ 8 different filters to increase our dataset volume. From the training of the model, we discovered the validation results to be quite satisfactory for each fold, with accuracy upwards of 88%. This prompted us to have hope that this would be one of the better models to use during our voting, but we soon discovered that the result of trying to predict on our test set, after the initial training, gave us a lower result of only upwards of 60%. This was also shown even if we used a voting system among the fold models, to try to improve the accuracy. We believe this is because of the potential over-fitting to the training data, where the model may have learned to depend heavily on the specific features enhanced by the Gabor filters, which are not as prevalent or relevant in the unseen test data. Additionally, the difference in performance could be attributed to a mismatch in the distribution of data features between the training and test sets, suggesting that our model has not generalized well beyond the training examples. The application of the Gabor filters, while increasing the feature set, may have also amplified this issue by making the model sensitive to particular textural characteristics that don't translate to broader, more diverse datasets.

J. SMOTE-oversampling results

As we tried to further deal with the problem of imbalanced classes, we decided to use oversampling as a way to fill our minority classes. After oversampling the dataset using the SMOTE-oversampling technique, we can now observe that there is a balance between the classes, as we can observe in figure 13

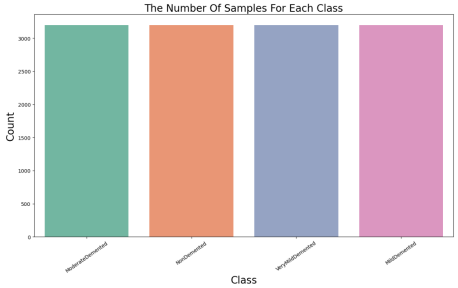

Fig. 13: The number of samples for each class after oversampling

K. Model Performance with Oversampling

This section evaluates the performance of the DenseNet, Inception, and ResNet models when trained with oversampling techniques to address class imbalance in the dataset. Each model was subsequently tested on both oversampled dataset and the original test dataset to assess generalizability and effectiveness of the oversampling approach. All of the models were trained on the configurations, as seen in table I

TABLE I: Configuration Parameters for Different Models

Parameter	ResNet50	InceptionV3	DenseNet121
Learning Rate	0.00001	0.001	0.001
Architecture	ResNet50	InceptionV3	DenseNet121
Dataset	alzheimer	alzheimer	alzheimer
Epochs	30	30	30
Batch Size	16	16	16
Input Shape	(176, 176, 3)	(299, 299, 3)	(176, 176, 3)
Number of Classes	4	4	4

1) DenseNet: The DenseNet model achieved validation accuracy ranging from approximately 85% to 93%. Both the training loss and validation loss steadily decreased with minor fluctuations in validation loss, as shown in Figure 14. The training and validation accuracy levels stabilized above 90%. On the oversampled dataset, test accuracy reached 95%.

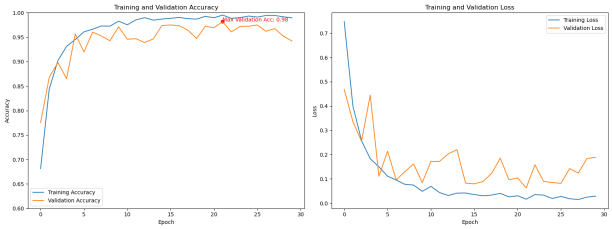

Fig. 14: DenseNet Training and Validation Loss with Oversampling

2) Inception: The Inception model demonstrated a validation accuracy ranging from approximately 85% to 91%. Both the training and validation losses decreased with occasional validation fluctuations, and the training accuracy stabilized while validation accuracy fluctuated between 85% and 92%. The test accuracy was around 91% on the oversampled dataset.

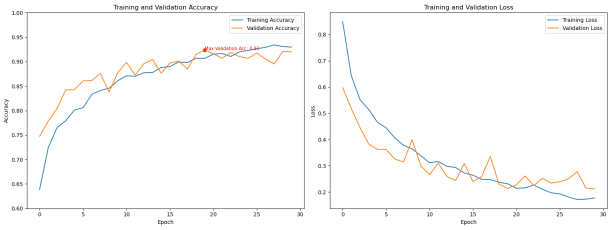

Fig. 15: Inception Training and Validation Loss with Oversampling

3) ResNet: The ResNet model achieved validation accuracy ranging from approximately 80% to 93%. The training loss and validation loss steadily decreased with some validation loss fluctuations, while accuracy and validation accuracy exhibited similar trends. Figure 16 shows these trends clearly.

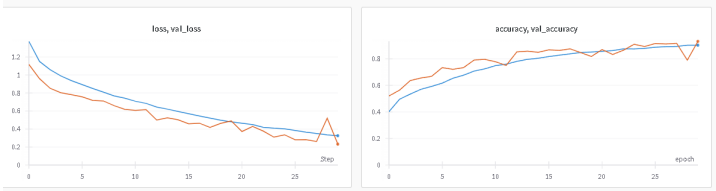

Fig. 16: ResNet Training and Validation Loss with Oversampling

To challenge the oversampling approach, the models were tested on the original test dataset with no synthetically generated samples. This yielded various results, were ResNet went from 93% to 88% and Inception decreased from 92% to 61% in accuracy and DenseNet improved onto 98%. The confusion matrix of the orignial dataset in Figure 17 indicates that DenseNet generalized better than other models. On the Left we have Inception and on the right we have DenseNet.

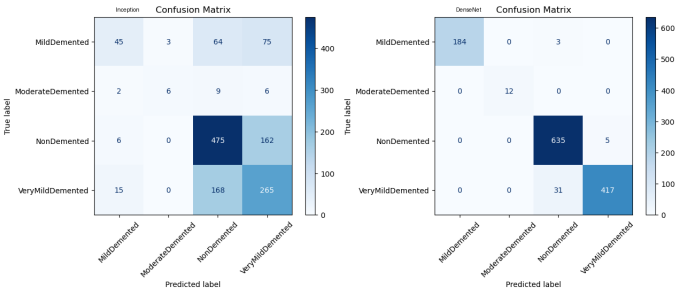

Fig. 17: Confusion Matrix for DenseNet vs Inception Model on Original dataset

Model	Accuracy on Original Dataset
DenseNet	98%
ResNet	88%
Inception	61%

TABLE II: Test Accuracies on Original Dataset

L. Additional modifications to the models

Modifications to the DenseNet, Inception, and ResNet models included the addition of batch normalization, dropout layers, and Dense layers a top the ImageNet base. In the models that we used oversampling, we made the additional change in the model as seen in table III, we decided to squeeze these changes between the base model and the predictions. Parameter updates for each model are summarized in the table IV.

Step	Layer Type	Key Parameters
1	Dense	128 units, use_bias=False
2	Batch Normalization	
3	Dropout	Rate of 0.5
4	Activation	'relu'
5	Dense	32 units, use_bias=False
6	Batch Normalization	
7	Activation	'relu'

TABLE III: Additional layers on the oversampled models

Model	Total Parameters	Trainable	Non-trainable
DenseNet	7,780,942	303,748	7,477,194
Inception	22,603,566	266,884	22,336,682
ResNet	23,861,124	4,738,628	19,122,496

TABLE IV: Summary of Model Parameters after Modifications

As we can see in the figure IV the amount of parameters in the DenseNet is less than the two other models, but despite this we observe that our DenseNet model with oversampling performs better on the test dataset after the initial training. We believe this is because of its dense architecture that is fully connecting all layers in our base model. This architecture ensures that each layer has direct access to the gradients from the loss function and the original input signal, leading to highly informative feature maps that preserve essential information throughout the network.

M. Voting system

After finding our best fit models, and some of the other models with decent accuracy, we tried to add them to a voting system to observe if the addition of multiple models could improve our accuracy, on the test dataset. There was a split result, where if we added the two models, oversampling with DenseNet and ResNet, together we increased our result, but only with 0.3% . This little increment is reasonable considering we have already gotten a very high accuracy of 98.6%, and the increment to 98.9% is quite good in terms of how little we can overall increase our accuracy. However if we tried to add other models such as the sobel augmentation model or the k-fold model together with the previously mentioned models, we would end up with a worse result. This is most likely because the best models have too high of an accuracy in compression to the other ones, so the overlap is complete. The voting was done through an averaging of the prediction results. The confusion matrix, in figure 18, shows the final best result from the voting.

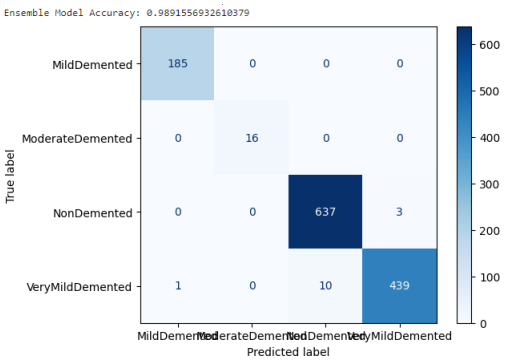

Fig. 18: Example of a confusion matrix describing the voting of the over sampled DenseNet and ResNet

N. All results of the models

In the following figure V , we display the final results for the prediction on the test dataset.

Model	Accuracy
No augmentation CNN	66% - 68%
No augmentation Inception	60% - 62%
No augmentation DenseNet	74% - 76%
No augmentation ResNet	66% - 69%
K-fold DenseNet	76 % -78 %
Sobel and augmentation	78% - 79%
K-fold and gabor ResNet	60%
over sample Inception	61%
over sample ResNet	88%
over sample DenseNet	98.6 %
voting over sample ResNet and DenseNet	98.9 %

TABLE V: Results of all models prediction on the test dataset

O. Mistakes and possible improvements

Some of the mistake we observed after implementing and training our models was that, despite the better results from the initial ResNet and DenseNet models, we should have also tried to add more augmentations, filters and training technique to the other models beside the two as well. The reason for this mistake was that we where swept up with trying to improve our model to a satisfactory level, but this also made us neglect the other models more than we should, and our combined result could have been improved if we increased our other models, and then added them to the voting.

Another improvement we could have done was to optimize our hyper parameters better. We have done this before with the usage of sweeps by wandb. However we observed this mistake a little too late to go through a proper sweep of the best models. Following this we could have also added some more filters to increase our training differences and which might have allowed us to discoverer more models that was better at the patterns that our current best models could not observe/discover. This would also prompt us to adapt a more dynamic learning rate such as a cyclical learning rate, which is a learning rate scheduler to gradually adjust learning rates, that could prevent premature convergence.

As we had used three models that incorporates pre-trained models, we had decided to unfreeze the last 10 layers of the

models to improve upon. The amount of layers unfreezed did have an effect on our models as we had tried with less unfreezed layer and it gave varying results. So one improvement we could have done in our project was to sweep the models based on the amount of layers unfreezed. With this we could have in theory gotten a better result, not only on our primary models, but on all of the models using this pre-trained model architecture.

Some other minor stuff we could have done, would be to add more or less layers to fit complexity of our dataset, as well as evaluate not only by its accuracy but also by its precision, recall, and F1 scores alongside accuracy. Another thing we could have done to improve or reach a satisfactory result would be to properly use the k-fold cross-validation on the more augmented and maybe over sampled datasets. This could have improved our overall over-fitting on most of our models, but unfortunately the time it took to train with k-fold cross-validation was often multiplied by the amount of folders created/used.

The second to last improvement we could have added would be to incorporate our voting through weighting, either by weighting the vote of each model based on their accuracy, confidence or both. This improvement to the voting system could make it so that the models that where more likely to be right would have a higher voting power within our system and we might have improved our accuracy by a little.

The last improvement we could have done was to use a more state of the art version of the different architectures as a base model for our project. As we have found some more advanced models created by different people.

P. Conclusion

In conclusion we tried a variation of different models and techniques to improve our accuracy. This led us to a result of 98.6 % using oversampling with DenseNet as an architecture. This was then combined together in a voting system, with the oversampling of ResNet, to achieve the final result of 98.9 %. In the future we would like to add on to the amount of models going over the 90% accuracy and then try testing if this could further improve our overall accuracy of the voting. As the result of our model shows a very high accuracy we will also note that this is in a closed environment, where we believe that the accuracy will go down when tested against other datasets, considering the class imbalance that still is prevalent in our dataset. This is why we would in the future gather more images for the minority classes to counter this imbalance in scenarios outside our environment.

REFERENCES

[1] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *Proc. 2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6, 2017.

[2] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," *arXiv preprint arXiv:1512.00567*, 2016.

[3] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4700–4708, 2017.

[4] F. He, T. Liu, and D. Tao, "Why ResNet Works? Residuals Generalize," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 12, pp. 5349–5359, 2020.

[5] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Oversampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, 2002.

[6] A. Dogan and D. Birant, "A weighted majority voting ensemble approach for classification," in *2019 4th International Conference on Computer Science and Engineering (UBMK)*, pp. 1-6, 2019.

[7] T.-T. Wong and P.-Y. Yeh, "Reliable accuracy estimates from k-fold cross-validation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 8, pp. 1586-1594, 2019.

[8] X.-Y. Liu, J. Wu, and Z.-H. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Trans. Syst., Man, Cybern., Part B (Cybern.)*, vol. 39, no. 2, pp. 539–550, 2008.

[9] S. A. Albelwi, "Deep architecture based on DenseNet-121 model for weather image recognition," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 10, pp. 1-8, 2022.

[10] A. K. Jain, N. K. Ratha, and S. Lakshmanan, "Object detection using Gabor filters," Pattern Recognition, vol. 30, no. 2, pp. 295–309, 1997.

[11] O. R. Vincent and O. Folorunso, "A descriptive algorithm for sobel image edge detection," Proceedings of informing science & IT education conference (InSITE), vol. 40, pp. 97–107, 2009.