F21DL Data Mining and Machine Learning: CW2

This report presents the results of our CW2 implementation. All experiments were led using Python and some included libraries: scikit-learn & Keras.

All our graphics and tables are in the ANNEX section. Due to their size, we have more than 5 pages as a report, but it only contains 5 actual pages of text.

Our code is, once again, different notebooks:

- ann.ipynb : code concerning the Neural Networks
- linear_classifiers.ipynb: code of the linear classifier
- decision_trees.ipynb: code of the decision trees
- cw2_research_question.ipynb: code of the research question with InceptionV3
- cw2_research_question-homemade_cnn.ipynb: code of the research question without InceptionV3

All these notebooks were exported as HTML for easier readability.

For our CNN work, we transformed the .csv into a folder of .png images that is accessible on OneDrive at this link:

https://heriotwatt-my.sharepoint.com/:f:/g/personal/td53_hw_ac_uk/EqD1hxri-jVCqpbUAzhG0oQB4j_jEUjB-k9VP1M_-9Au4w?e=XRmqFI

We saved the session for two of our notebooks by using the python "*dill*" library. Thanks to this, trained models are easily accessible:
https://drive.google.com/drive/folders/1c0BGWZHZTOYog95du777Z8uv34qr5Tl4?fbclid=IwAR20KmPJ2OlYKMQ0OgSFZ3pABEHEt4NLF-243Jrm9ihDJuVg-iyA9KxmlJY

(linear classifiers + ANN).

The link to all our notebooks is:

https://heriotwatt-my.sharepoint.com/:f:/g/personal/td53_hw_ac_uk/EuGzAOz7x1BCvI31542DrXEBzJhBq1zCHciYZcG1Ckm6MA?e=lVVvOU

Decision Trees (Decision Tree + Random Forest + Extra Trees)	Auzimour Antoine, Di Martino Thomas
Linear Classifiers & ANN	Accad Dimitri, Deltel Clarence
Research Question	Di Martino Thomas

Variation in performance with varying learning parameters in Decision Trees

3 different algorithms for decision trees were used throughout this project: Decision Tree, Extra Trees and Random Forest.

The decision tree implementation of sklearn uses the CART implementation which differs from the C4.5 (also known as J48) only by its abilities to perform regression, which we did not find use here.

The random forest estimator just aggregates a certain number of decision trees and uses a voting system to make decisions: it is what is called an *ensemble* algorithm. Ensemble algorithms all work the same way: they fit a certain amount of weaker model to combine them into a stronger one. The Extra Trees algorithm is very similar to the Random Forest except that it does not compute the best features when making splits and instead, do them randomly.

a. Decision Trees

For our decision trees, we played with the following parameters:

- Split criterion: between *Gini* where is used the Gini impurity formula and E*ntropy* where is used the information gain formula. Gini and entropy are just two different formulas, originating from the theory of information stating how much information is held by an observation.
- Class weights: balancing the weights of each class when calculating the criterion value for each potential split or not balancing the weights.

We were expecting the balanced weights to have a positive influence on the F1-Score, to the sacrifice for a bit of accuracy. All the metrics are average of weighted binary metrics for each individual class.

When comparing the performance of the 4 possible models with cross validation, we found the following results:

	Accuracy	Precision	Recall	F1-Score
Gini, balanced	0.866311	0.846	0.854	0.854
Gini, not balanced	0.8728	0.843	0.863	0.856
Entropy, not balanced	0.892627	0.881	0.8813	0.880
Entropy, balanced	0.8891	0.883	0.875	0.878

As we see here, the best configuration is, surprisingly, the use of non-weighted class when computing the split criterion, being here the entropy criterion.

b. Ensemble algorithms

Now that we see what a single Decision Tree can do, we expect more coming from an ensemble. For random forest & Extra Trees, we played with the following parameters:

- Split criterion
- Number of estimators: we evaluated random forest with different number of decision trees that ranged between 5 and 50.

We hereby see that the criterion does not have a very strong influence on the evolution of the performance, but the Gini criterion still outperforms the Entropy criterion. Concerning the number of estimators, we see that the more we have, the better the algorithm is but the performance starts to stay still above 40 estimators.

Extra trees classifier is supposed to be better than RandomForest when dealing with noisy data. When all features are relevant, RandomForest is better. The parameters we best played with were the same as before: the criterion and the number of estimators. The best accuracy of our Random Forest (Gini + 45 estimators) is: 0.996. For Extra Trees (Entropy + 40 Estimators), it is 0.974. These results are displayed in Figure 1.

Variation in performance with size of the training and testing sets

For any algorithm, we would intuitively think that the biggest the training set, the better the performances. However, the bigger the training set, the smaller and hence, less relevant, the test set will be. So, we must find a correct balance between better training and reliable performance metrics.

a. Decision Trees

As we can see in our **Erreur ! Source du renvoi introuvable.**, the ensemble algorithms show pretty consistent results no matter the size of the test size: the best performances are, of course, in the 10-Fold scenario but for a bigger Test Sets, they do not suffer of drastic fall in performance. This may be linked to the fact that both Random Forest and Extra Trees algorithm already use subsamples of data and features to make their split decision with the use of what we call "*Bootstrap samples*".

The performance decline is more noticeable for the basic Decision Tree: with precision and recall around 0.9 in the 10-Fold situation, it all goes down to values closer to 0.8 when the test size is 70% of the whole dataset. This is pretty good overall results and it shows that when subsampling is done right, by having the same distribution of classes between the original training sample and the smaller training sample.

b. Neural Networks & Linear Classifiers

i. Linear Classifiers

We trained here two linear classifiers (Logistic Regression & Linear Discriminant Analysis) and 10 different ANNs. Over 10 folds, the accuracy of the linear discriminant was around 86% while the accuracy of the logistic regression was around 94%. The bar charts drawn from this experiment also show better results of precision, recall and thus F1 score for the logistic regression model.

Our ROC curves also show slightly better results for the Logistic regression model compared to the linear discriminant analysis model with ROC area higher than 0.95 for the logistic regression and closer to 0.9 for the linear discriminant analysis.

When consecutively testing on 3000, 4000 and 9000 test samples instead of the cross validation, we noted some interesting trends on the accuracy. For 3000 and 4000 test samples, the accuracy of both models is almost the same as the 10-fold cross validation, 86% for linear discriminant and 93% for logistic regression. The bar charts of recall, precision and F1-scores are roughly the same as the bar charts of the 10-fold cross

validation. However, it gets interesting when testing with 9000 test samples. Here, the accuracy of the Linear discriminant drastically drops to 66% while the accuracy of the logistic regression barely decreases to around 91.7%. The difference is important, and the Figure 3 shows that well.

ii. Artificial Neural Networks

For our Multilayer Perceptron architectures (cf Table 1), we have experienced sensibly the same performances on the 10-Fold CV scheme. This is when testing on the different train/test split schemes that we started to have noticeable differences:

On 3000, ANN 2, 5, 6, 7, 9 and 10 seems to struggle a lot more than the others.
On 4000, ANN 1, 5, 6, and 9 are underperforming.
On 9000, ANN 1, 5, 6, and 9 are underperforming.

Variation in performance with changes in the learning paradigm

We will now compare the performance of Decision Trees vs Artificial Neural Network. This comparison only stands for this specific dataset, but we will try to pull out a more general analysis that still holds in different yet common situations.

When comparing "***Erreur ! Source du renvoi introuvable.***" and "*Table 3: Results of the Neural Networks and Linear classifiers experiments*", we clearly note a difference in stability: Linear Classifiers are the most sensible to smaller training set and even MLP are outperformed by ensemble methods by almost 10% when using bigger test set and hence, smaller training set. MLP are more prone to overfitting than random forest with smaller training samples and our results in precision and recall (sometimes even reaching mediocre results for our ANN with 0.58) are the proof of this sensibility to data.

Variation in performance with varying learning parameters in Neural Networks

When jumping from on architecture to another, we start to see trends in our performances:

For instance, ANN 5, 6 and 9 tend to underperform: these are the ones with, respectively, less epochs, tanh activation functions and smaller second hidden layer.
The best performer is ANN 3, 4 and 8. ANN 3 use only one hidden layer and ANN 8 has a dropout of 0.4 instead of 0.2. We can see that these three networks are tweaked in a way where overfitting is reduced (by respectively having a shallower network and by enhancing the dropout rate). ANN 3 has less weights to consider so it fits to the training sample and will have more generalizable predictions. ANN 8 has a bigger dropout which forces the network to learn general knowledge on different neurons and not just learning the training sample. These differences of performances are noticeable in the

Variation in performance according to different metrics

When it comes to decision trees, the various performance metric show a balanced behavior: both the precision and recall are high and of about the same value. We see a pretty good consistency in the predictions, depicted by high precision score. Furthermore, very good recall measures are presented with value as high, if not higher, than the precision score. As expected, the lowest metrics scores are with the Decision Tree model, but he still achieves around 0.9 for both precision and recall on the 10-fold CV training procedure but it goes down to 0.85/0.8 for bigger test set (and smaller training set).

For the two ensemble methods, the performances are astounding: no matter the size of the test set, they always achieve more than 0.9 with values equal to 0.99 for every class in both precision and recall in the 10-Fold situation.

For these 3 algorithms, we computed ROC Area for every class: when we see values around 0.9 for the Decision Tree algorithm, no matter the dataset form, our ensemble methods reach 1.0 in the big majority of cases. This shows how low is the probability of false positive detection, in the context of binary classification for every class. This is a great sign that shows how False Positive are extremely rare, even for the "*basic*" Decision Tree.

For the Linear Classifiers and our ANN, we see that our accuracy measures are pretty good first sight of the "*health*" of our models: there is no big disequilibrium in our classes precision and recalls and the ROC area scores are consistent throughout the classes. We see, however, that the smaller the test set, the less reliable the accuracy starts to be: when the ANN still tops up with 90% accuracy, we see a huge drop in recall for the first two classes (the two speed limit signs) with only 58% recall, meaning that only 58% of examples of both these street signs are correctly classified as such, missing a lot others.

Research Question: CNN and transfer learning

In our research question, we dived deeper into advanced techniques for applications of Neural Networks in the Computer Vision field. The most common architecture used for Computer Vision is the Convolutional Neural Network. Introduced, in the form we know today, by Yann LeCun in 1989, the convolutional neural networks are based on the convolution process: they apply a certain range of filters to every location of the image. A parallel could be drawn here with the notion of having shared weights. The weights of the networks are stored in the filters. These filters are superposed across the image, but they keep their weights, no matter where they are positioned. This means that two different pixels from the same image will receive the same weights from the same filter.

These filters try to detect patterns in different position of the image: this is this property that gives the CNN their famous translation and shift invariance. Indeed, in a real-life situation, given a camera fixated to the front of a car, if we want to detect street signs, we must be able to detect them no matter where they are on the image. This is where any other common algorithms would struggle, even MLPs.

This is what inspired us to try ConvNets, even if in our dataset, all the images have been rescaled in a way that the street sign was positioned in the middle and were, on very rare occasions, shifted away. Such translation was mandatory to test the performance of non-shift invariant algorithms like all the other algorithms we tried.

Loosing this advantage, we were not hoping to see our ConvNets outperform our best classifiers like Random Forest or Extra Trees. For our testing procedure, we put side to side two CNNs: one coded by hand, another using transfer learning. This concept is simple. Most common CNNs are made of two steps:

1. A convolutional step where we encode our image in feature maps filled with information
2. A fully-connected step where we classify these features to their corresponding classes.

However, in most problems, the features that the convolutional step try to find are the same, no matter the dataset: it is usually contour of objects represented by pointy, round or squared shapes. The "*agglomeration*" of these detected features is what is used to classify our image in the fully-connected step. Therefore, the transfer learning concept is just re-using the convolutional step of a famous architecture, trained on a very large dataset, meaning that he learnt to recognize a lot of different kind of features. Then, from this convolutional step we add our own top fully-connected layers that we will train (i.e "*finetune*") to correctly classify our classes while, at the same time, correcting the weights of the convolutional layers.

The results of our homemade CNN are as follows:

Classes	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**accuracy**	**macro avg**	**weighted avg**
precision	0.99	0.96	1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.92	0.989336	0.986971	0.989597
recall	0.94	0.99	1.00	0.99	0.99	1.00	1.00	1.00	0.98	0.96	0.989336	0.988870	0.989336
f1-score	0.97	0.98	1.00	0.99	0.99	1.00	1.00	1.00	0.98	0.94	0.989336	0.987783	0.989341
support	287	386	87	264	411	465	157	44	365	66	0.989	2532	2532

We see astounding performances and our 10 classes with a lot of them reaching the incredible 1.00 value of F1-Score. Its architecture is basic but deep enough to encapsulate different concepts for every class (cf Figure 5: CNN architecture).

The use of a lot of Dropout layers helped us to minimize overfitting to the extent that we now have the best test performance (on a 60/20/20 dataset split) out of all the tested algorithms. But what about our transfer learning model?

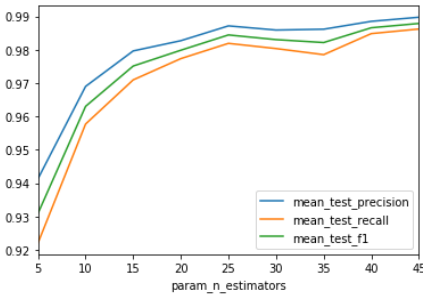
For our convolutional layers, we used the InceptionV3 model, a very deep network that reuses a lot of what are called "*Inception modules*" that are just aggregation of different convolutional operation, leading to better detection of same object but in different scales. We had a hard time finetuning our network as it gets very difficult to pick the good optimizer with the correct parameters and we were running out of time and processing power to train our networks. We relied fully on an RTX 2060 to train our images. We were able to pull out a 75% accuracy on our test set with the following metrics:

Classes	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**accuracy**	**macro avg**	**weighted avg**
precision	0.35	0.63	1.00	0.97	0.78	0.99	0.98	1.00	0.77	0.0	0.75	0.75	0.77
recall	0.64	0.28	0.11	0.92	0.99	0.92	0.89	0.56	0.96	0.0	0.75	0.63	0.75
f1-score	0.45	0.39	0.20	0.95	0.87	0.96	0.94	0.72	0.86	0.0	0.75	0.63	0.73
support	287	386	87	264	411	465	157	44	365	66	0.75	2532	2532

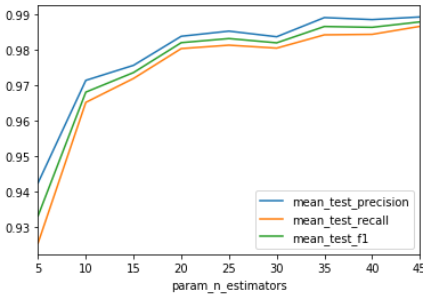
Our performance is still correct but way off compared to our CNN. We then draw the conclusion that a handmade CNN is a fast and reliable solution for any image classification tasks that may involve shifting object.

ANNEX: Figures & Tables

Random Forest

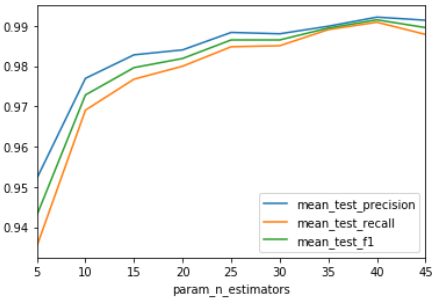


Evolution of precision and recall based on the number of estimators used for the entropy criterion

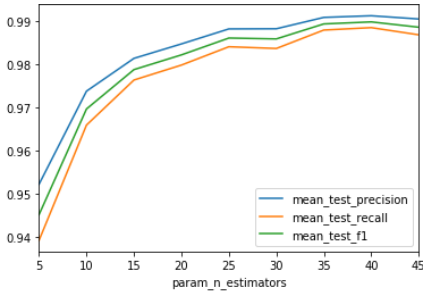


Evolution of precision and recall based on the number of estimators used for a Gini criterion

Extra Trees



Evolution of precision and recall based on the number of estimators used for the entropy criterion



Evolution of precision and recall based on the number of estimators used for a Gini criterion

Figure 1: Performance comparison between our two ensemble methods

	ANN_1	ANN_2	ANN_3	ANN_4	ANN_5	ANN_6	ANN_7	ANN_8	ANN_9	ANN_10
layers	3	4	2	3	3	3	3	3	3	3
neurons	512, 512, 10	512, 512, 250, 10	512, 10	512, 512, 10	512, 512, 10	512, 512, 10	512, 512, 10	512, 512, 10	512, 250, 10	250, 512, 10
dropout	0.2	0.2	0.2	0.2	0.2	0.2	0.1	0.4	0.2	0.2
activation_function	relu, relu, softmax	relu, relu, relu, softmax	relu, softmax	relu, relu, softmax	relu, relu, softmax	tanh, tanh, softmax	relu, relu, softmax	relu, relu, softmax	relu, relu, softmax	relu, relu, softmax
batch_size	128	128	128	128	128	128	128	128	128	128
epochs	20	20	20	40	10	20	20	20	20	20

Table 1: 10 different ANN architecture used for training

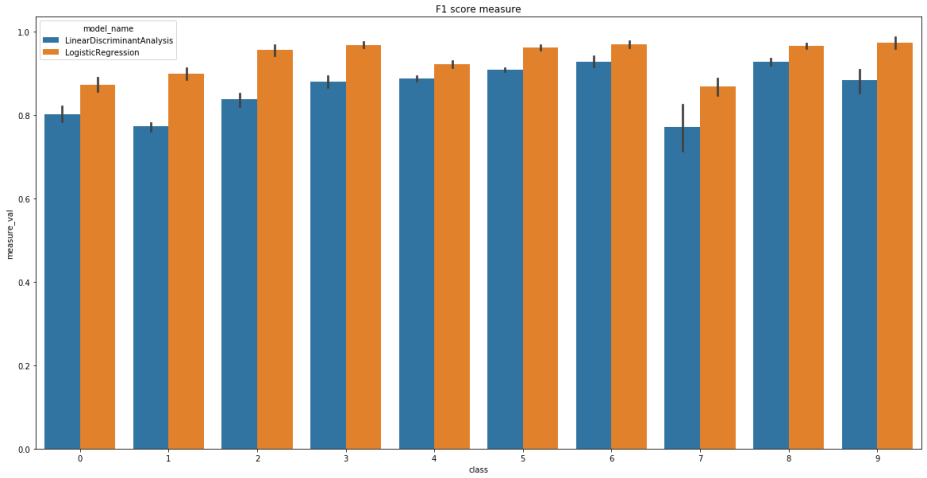


Figure 2: Bar Chart of Linear classifier F1-Score on 10-Fold CV

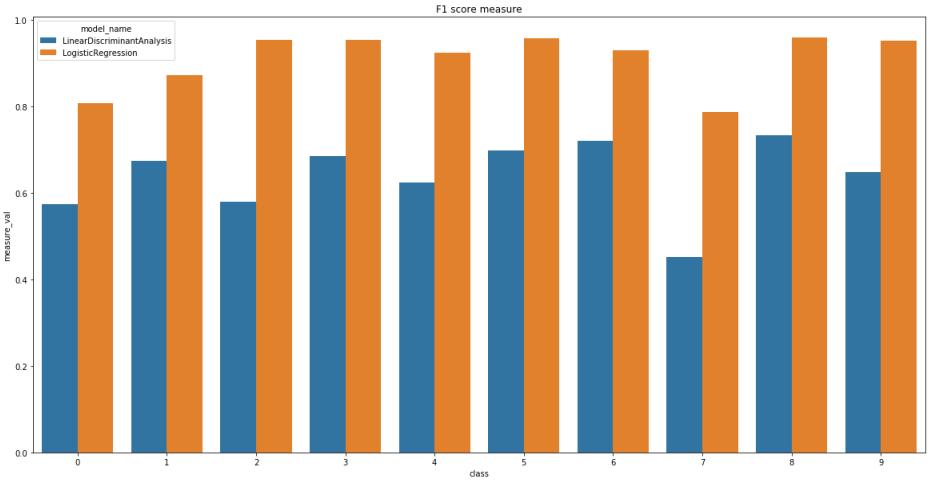


Figure 3: Bar Chart of Linear classifier F1-Score on 9000 sized test set

Setup	Models	Acc.	Measures	0	1	2	3	4	5	6	7	8	9
10-FOLD CV avg. precision recalls over folds	D. Tree	0.91	Precision	0.89	0.91	0.90	0.91	0.88	0.91	0.90	0.88	0.90	0.90
			Recall	0.91	0.92	0.91	0.90	0.89	0.88	0.90	0.88	0.90	0.90
	R. Forest	0.99		0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.99
				0.99	0.99	0.99	0.98	0.98	0.98	0.99	0.98	0.98	0.99
	E. Trees	0.99		0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
				0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.98
Test size: 3000	D. Tree	0.89	Precision	0.79	0.81	0.85	0.97	0.91	0.94	0.90	0.84	0.89	0.94
			Recall	0.84	0.81	0.848	0.89	0.89	0.93	0.89	0.90	0.95	0.88
	R. Forest	0.98		0.97	0.95	1.00	1.00	0.98	0.99	1.00	1.00	0.98	1.00
				0.94	0.97	0.97	0.98	0.98	0.99	0.99	1.00	1.00	1.00
	E. Trees	0.98		0.98	0.98	1.00	0.99	0.99	0.99	1.00	1.00	0.98	1.00
				0.96	0.98	0.98	0.98	0.99	0.99	0.99	1.00	1.00	1.00
Test size: 4000	D. Tree	0.88	Precision	0.77	0.79	0.88	0.94	0.92	0.90	0.97	0.83	0.93	0.95
			Recall	0.78	0.81	0.87	0.90	0.91	0.94	0.90	0.86	0.92	0.88
	R. Forest	0.98		0.97	0.96	0.99	0.99	0.98	0.98	1.00	0.98	0.98	1.00
				0.94	0.97	0.97	0.98	0.98	0.99	0.98	0.98	0.99	1.00
	E. Trees	0.98		0.98	0.96	1.00	0.99	0.98	0.98	1.00	0.98	0.99	1.00
				0.96	0.98	0.96	0.99	0.99	0.99	0.98	0.98	1.00	1.00
Test size: 9000	D. Tree	0.85	Precision	0.73	0.75	0.82	0.91	0.90	0.91	0.89	0.82	0.89	0.81
			Recall	0.75	0.75	0.80	0.86	0.89	0.91	0.88	0.86	0.91	0.87
	R. Forest	0.96		0.95	0.91	0.99	0.99	0.97	0.98	0.99	0.98	0.97	1.00
				0.88	0.95	0.96	0.97	0.98	0.99	0.97	0.94	0.99	0.99
	E. Trees	0.97		0.94	0.92	0.99	0.98	0.97	0.98	1.00	1.00	0.98	1.00
				0.89	0.95	0.99	0.98	0.99	0.99	0.97	0.95	0.99	0.99

Table 2: Results of experiments on Tree-based algorithms

Setup	Models	Acc.		0	1	2	3	4	5	6	7	8	9
10-FOLD CV avg. precision recalls over folds	Linear Discriminant Analysis	0.87	Precision	0.88	0.90	0.87	0.91	0.86	0.90	0.89	0.90	0.89	0.90
			Recall	0.84	0.87	0.81	0.86	0.80	0.84	0.84	0.86	0.85	0.85
	Logistic Regression	0.94		0.94	0.95	0.94	0.95	0.93	0.94	0.94	0.94	0.96	0.94
				0.94	0.94	0.95	0.92	0.94	0.92	0.91	0.94	0.94	0.94
	ANN-1	0.94		0.98	0.98	0.94	0.94	0.96	0.96	0.96	0.94	0.95	0.96
				0.97	0.97	0.97	0.91	0.92	0.95	0.95	0.95	0.91	0.93
Test size: 3000	Linear Discriminant Analysis	0.86	Precision	0.78	0.66	0.89	0.99	0.91	0.98	0.99	0.76	0.9	1
			Recall	0.76	0.95	0.91	0.8	0.84	0.84	0.84	0.68	0.95	0.86
	Logistic Regression	0.94		0.90	0.86	0.99	0.97	0.93	0.97	0.99	0.88	0.96	0.99
				0.86	0.86	0.94	0.97	0.95	0.90	0.96	0.93	0.84	0.99
	ANN-1	0.95		0.90	0.91	1	1	0.97	0.99	0.99	0.96	0.97	0.97
				0.87	0.87	0.92	0.98	0.94	0.96	0.98	0.98	0.98	0.99
Test size: 4000	Linear Discriminant Analysis	0.85	Precision	0.76	0.66	0.83	0.98	0.89	0.97	0.96	0.70	0.90	0.98
			Recall	0.77	0.94	0.90	0.79	0.82	0.82	0.84	0.66	0.93	0.87
	Logistic Regression	0.93		0.86	0.87	0.99	0.96	0.93	0.96	1	0.84	0.96	0.99
				0.84	0.84	0.93	0.96	0.94	0.91	0.95	0.93	0.83	0.97
	ANN-1	0.90		0.65	0.91	0.99	1	0.98	0.99	01	0.93	0.91	0.97
				0.89	0.89	0.66	0.89	0.92	0.93	0.99	0.94	0.97	0.97
Test size: 9000	Linear Discriminant Analysis	0.66	Precision	0.55	0.58	0.46	0.72	0.74	0.82	0.74	0.34	0.77	0.58
			Recall	0.60	0.81	0.78	0.65	0.54	0.61	0.7	0.68	0.7	0.74
	Logistic Regression	0.92		0.80	0.84	0.96	0.97	0.94	0.96	0.99	0.81	0.94	0.98
				0.82	0.82	0.9	0.95	0.94	0.91	0.96	0.87	0.77	0.98
	ANN-1	0.90		0.74	0.72	0.96	0.99	0.97	0.97	1	0.97	0.97	0.98
				0.58	0.58	0.86	0.98	0.92	0.94	0.99	0.93	0.93	0.96

Table 3: Results of the Neural Networks and Linear classifiers experiments

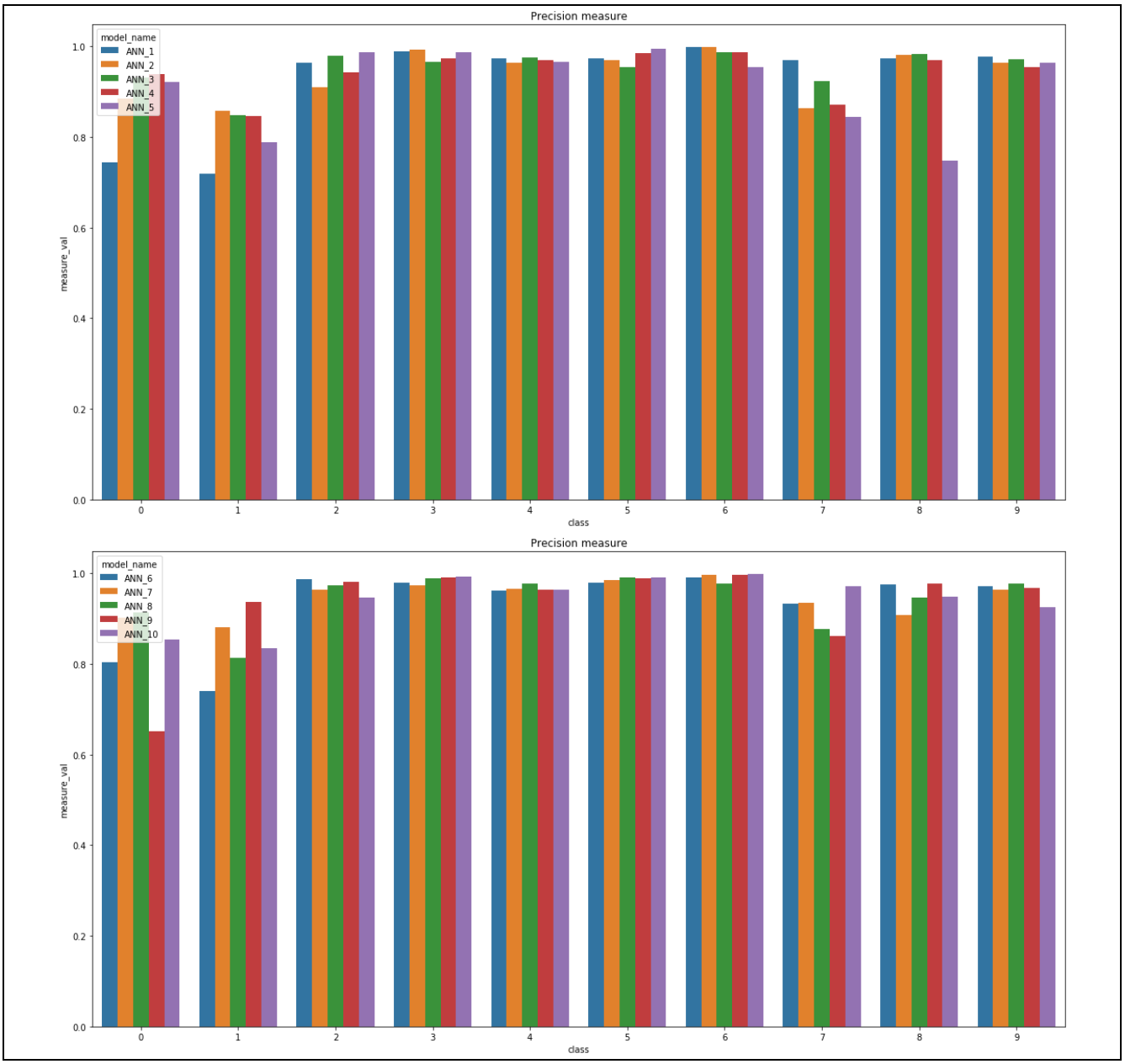


Figure 4:Precision measures for the 10 ANN architectures

Figure 5: CNN architecture

Input	48x48x3
Conv1	3x3@16
Conv2	3x3@32
MaxPooling	2x2
Dropout	0.2
Conv3	3x3@32
Conv4	3x3@16
MaxPooling	2x2

Dropout	0.2
Fully-Connected	1024
Dropout	0.3
Fully-Connected	256
Dropout	0.3
Fully-Connected	64
Dropout	0.3
Fully-Connected	10

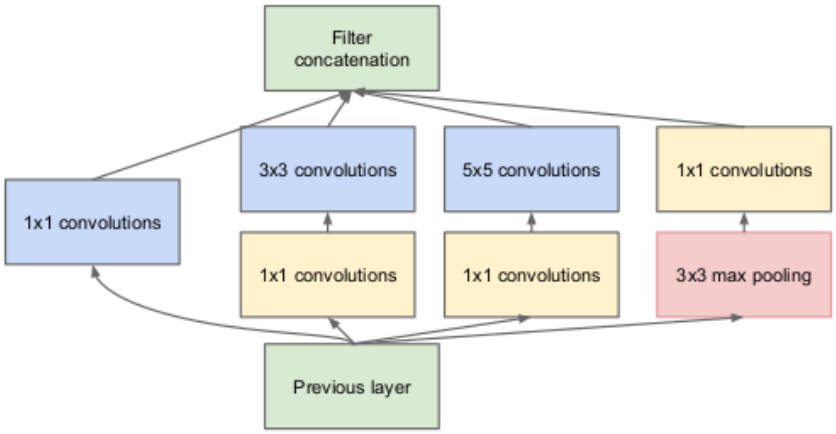


Figure 6: Illustration of the inception modules