CNN for rock paper scissors classification

Riccardo Conforto Galli

October 19, 2025

Abstract

The aim of this project is to develop different Convolutional Neural Networks to classify images of hand gestures from the classic game Rock, Paper, Scissors into their respective class. In order to achieve this goal three models have ben proposed and tuned on the dataset using nested cross validation in combination with grid search. The models have been tested not only on a test set obtained from the original dataset, but also on a secondary dataset autonomously collected to evaluate the model ability to generalize the knowledge learned.

1 Tools

The project has been developed both using the free plan of Google Colab, a cloud service that allows the creation and execution of interactive Jupyter's notebooks, and my PC (intel i5-8260U). One of the main advantages of Google Colab is the ability to execute code using dedicated T4 GPU, greatly speeding the training process. However the avalability of the GPU with the free plan is quite limited, so in addition to that the default CPU environment has also been used.

As a machine learning library I chose Pytorch simply because of my familiarity with it due to previous projects.

2 Introduction to CNNs

A Convolutional Neural Network (CNN) is a type of neural network, popular in computer vision tasks, that uses convolution layers. In this layer a convolution is computed between a kernel and a region of the image of the same size. The kernel is shifted gradually on all the input dimensions by a fixed quantity called stride until all the region of the image have been covered.

After a convolutional layer there is often a pooling layer used to reduce the dimensions of the output with the application of a function over a region of the input. Two common choices are taking only the highest value (Max-Pool) or only the mean value (Avg-Pool) of the region. Then similarly to the convolution kernel, the pooling window shift of a predefined amount of pixel called stride.

Usually after a certain number of convolutional and pooling layer there is a sequence of fully connected layer needed to reduce the output of the convolutional part to a number of neurons equal to the output class (in the case of a classification task).

3 Data

3.1 Data exploration

The dataset is available on Kaggle[1], it's composed of 2188 images subdivided in three folders: `rock`, `paper` and `scissors`. As the name implies, each folder contains images of hand gestures corresponding to one of the possible moves in the popular game rock, paper, scissors. The number of images in each folder, reported in Table 1, is quite balanced as will be discussed later.

Given the small size of the dataset, images have been firstly analyzed by hand in order to discover image that have been misclassified, corrupted or that are not relevant to the classification task.

During this exploration is possible to note that every image is captured from the same perspective on a green background and with relatively consistent lightning and white balance.

[1]https://www.kaggle.com/datasets/drgfreeman/rockpaperscissors

`rock`	726
`paper`	712
`scissors`	750

Table 1: Number of images in each folder

Hands have different positions in the frame, different distance relative to the camera and come with various angles but always from the right side of the frame. Hands belongs to different person with a range of size and shape, both with and without sleeves. It is however important to note that the population is quite limited and that could pose a challenge in a possible generalization of the model since, for example, they all have a narrow range of skin tones.

Each image has the same format (.png) and the same dimensions (200x300), this simplifies the preprocessing step since it's not needed to crop nor to convert the images.

3.2 Balancing

To give a quantitative measure of the heterogeneity of the dataset, it is possible to use metrics like the Imbalance ratio or the Gini index.

The imbalance ratio is defined as:

$$IR = \frac{N_{max}}{N_{min}}$$

where N_{max} and N_{min} are respectively the cardinality of the class with the highest and smallest number of elements. Intuitively a IR of 1 is perfectly balanced while values significantly greater than 1 represent an imbalance in the class ratio. The rock paper scissors dataset has an imbalance ratio of $IR = \frac{750}{712} \approx 1.053$, that is sufficiently close to one.

The Gini index is defined as:

$$G = 1 - \sum_{i=1}^{C} \left(\frac{N_i}{N}\right)^2$$

Where N_i is the cardinality of the class i and N is the number of elements in the dataset. To have a more immediate the evaluation of the index it is often used the normalized version of the index:

$$G_n = \frac{m}{m-1} G$$

Where m is the number of classes. The normalized Gini index is a number ranging from 0, indicating perfect homogeneity, to 1, indicating perfect balance of class representation. The Gini index for the rock paper scissors dataset is $G = \frac{3}{2}\left(1 - \left(\left(\frac{726}{2188}\right)^2 + \left(\frac{712}{2188}\right)^2 + \left(\frac{750}{2188}\right)^2\right)\right) \approx 0.9998$ that is also sufficiently close to 1. We can thus conclude that the dataset is balanced, this means that is not necessary to apply specific techniques to handle it.

3.3 Preprocessing

As previously mentioned the dataset consists of three folders each containing all images relative to one of the possible hand gestures used in rock, paper, scissors. For this reason the first step is to split the dataset into a training and test part, with the training set corresponding to 90% of the original dataset and the remaining being the test part. The reason behind this ratio is to avoid a training set too small for the application of nested cross validation for hyperparameter tuning. During nested cross validation the dataset will be split again twice, so choosing a big enough starting training set is crucial. On the other hand we still wish to maintain a test set with a relevant size, hence the choice of the aforementioned ratio.

Creating different folders for training and test dataset is the easiest way to create different dataloaders with different associated transformation in pytorch. This proves to be useful in two different situations.

Firstly to apply data augmentation techniques only on the training set: this is done in order to have a realistic evaluation of the model, as this techniques could change the data distribution of the test set.

Secondly, since most of the training has been performed on a PC, this allows to download and split the dataset only once, saving some time between separate runs and ensuring that the split is always the same.

Images are then converted into tensor, their dimension is halved and they are transformed using data augmentation techniques described in 3.4. Before being used as network's input, a normalization in order to have a faster convergence for the algorithm.

3.4 Data augmentation

As previously stated, the dataset is quite limited in size and especially in variety of the examples provided. For this reason, even a trivial network with only one layer could achieve reasonable performance on this dataset, and a more complex one would rarely show sign of overfitting because of the high similarity between training and test set. Moreover, while the aim of this project was not to obtain a perfectly generalized model for paper, rock, scissors classification, the lack of variety on the data would produce a predictor useless on any lighting and background conditions except for the exact ones used to produce the dataset. To mitigate this effect, a series of transformations have been applied to artificially augment the data. Note however that data augmentation is effective as long as the transformation applied is something obtainable in the original data distribution. For example an horizontal flip would probably be useless in our case since we expect all hands to always come from the right side of the frame. The transformation used in this project are the following:

3.4.1 Random vertical flip

Probably the easiest way to augment a dataset about hand gesture is to apply a random vertical flip. Since all hands pictured come from the right, this is equivalent to simulating the effect of having the left hand version of the gesture (right hand version when a left hand is pictured).

3.4.2 Random rotation

Useful to augment the dataset with other possible hand orientations. The range of possible rotation has been set to $(-20°, 20°)$, since as previously stated possible arm positions are restricted to the right side of the frame.

3.4.3 Color Jitter

In order to counteract the low variance in settings of the original dataset and allowing the network to concentrate on more important feature like the hand geometry. Color jitter applies random change in the image hue, brightness, saturation and contrast. The parameters of this transformation are represented in table 2. A value of 0.3 means that the applied variation is $\pm 30\%$ of the original value.

brightness	0.3
hue	0.3
saturation	0.3
contrast	0.2

Table 2: Color jitter parameters

4 Models

4.1 KissNet

This network consists in a single hidden layer, with 25×37 inputs and 3 outputs. Before entering the network the image is resized in order to have the desired resolution of 25×37 pixels. This serves as a baseline to evaluate other networks' performances.

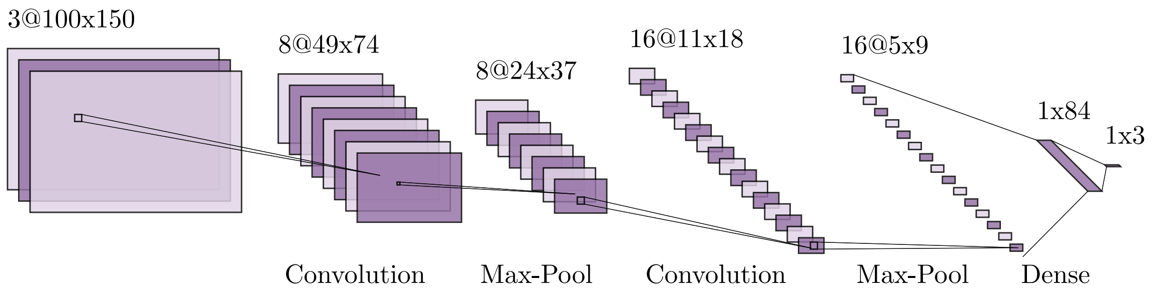


Figure 1: Architecture of EzNet for the default configuration of 5×5 kernel size with stride of 2.

4.2 EzNet

This network is heavily inspired from the famous LeNet-5[2], adapted for this specific task. Similarly to most CNNs, LeNet-5 and EzNet consists of two parts:

- Convolution part: consists of a series of subsequent convolution and pooling operations. In the case of EzNet there are two convolution-maxpool layer. Both convolution use the same kernel size, stride and padding. Padding is set to one, while kernel size and stride are considered hyperparameter and are object of a more in-depth explanation in 5.1. The pooling operation is Max-Pool with a 2×2 filter.
- Fully connected part: as the name suggest consists of a number of fully connected layers. In the case of EzNet, there are two layers one with 84 neurons, and a second output layer with 3 neurons.

Instead of Sigmoid activation function used in the original implementation, EzNet uses ReLu. The main advantages of using ReLu instead of Sigmoid are the lower computation cost and fewer vanishing gradient problems.

The number of filters in the first convolutional layer has also been increased from 6 to 8 in an attempt to better adapt the original architecture to the RockPaperScissors dataset, since the latter uses RGB images (with 3 channels) instead of the gray-scale of the original LeNet dataset (only one channel).

One last difference in the convolution part is the choice of Max-pool instead of Avg-pool. Max-pool offers two main advantage over Avg-pool: is cheaper to compute and, more importantly, gives the model translation invariance. This helps the model recognize the hand shape independently from its position in the space.

4.3 WideNet

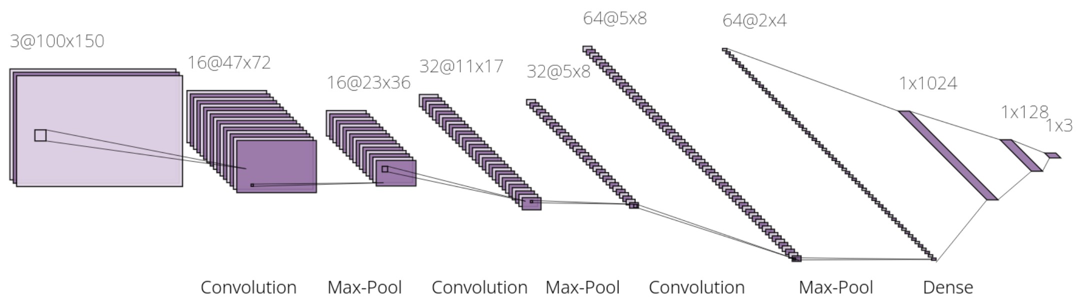


Figure 2: Architecture of WideNet for the default configuration: 16 starting channels, doubled at each step

[2]LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, vol. 86, no. 11, 1998, pp. 2278-2324.

Since each channel specializes in the learning of a feature, one intuitive way of improving the performance of a CNN is to increase the number of channels produced at each convolution step. WideNet has been developed with the intent of testing this intuition. This network consists of 3 convolutional layer, each doubling in output the number of channel from the previous layer except for the first one. The first one takes as input the three channel of the RGB images and outputs a number of channels defined at initialization time as an hyperparameter. A pooling layer of size 2×2 and stride equal 2 is applied after each convolutional layer.

The kernel size of the convolution decreases gradually, starting with 10 in the first layer, 5 in the second and 3 in the last one. The same happens with the stride (2 in the first and second, 1 in the third) and the reasoning behind is the same: as the pooling layer decreases the size of the output, a filter applied on it involves a larger area of the original image.

The fully connected layer consists of three layers: the first with 1024 neurons, the second with 128 and the third one with 3. To compensate the bigger size a dropout layer has been inserted after the first two to prevent overfitting.

4.4 DeepNet

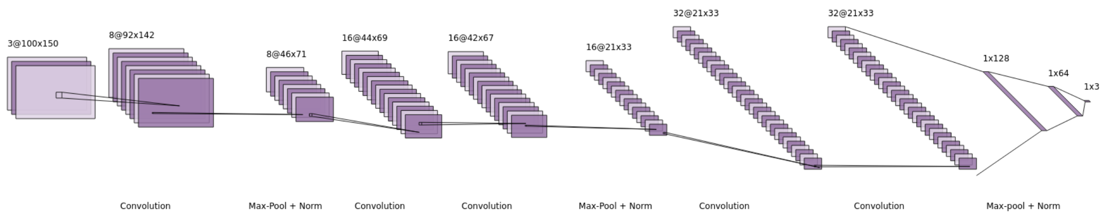


Figure 3: Architecture of DeepNet for the default configuration: three convolution blocks of parameter (1,8,11),(2,16,5),(2,32,3).

One problem with WideNet is that the output of the last layer is a mere 2×4 tensor. For this reason on this dataset the number of convolution-pooling operation is limited to three. However it is possible to apply more than a single convolution before reducing the output size and doing so yield interesting results. In fact successive application of two convolutions with 5×5 kernels, involve approximately as many parameter as three 3×3, but the second approach introduces more nonlinearity. In literature, deep convolutional neural network with smaller kernel size have proven to usually yield better results.

DeepNet is therefore composed of convolution blocks, each convolution block accepts three parameters: the number of convolution in the block, the number of output channel of the block and the size of the kernel. The last two layers of each convolution block are a Max-Pool layer with the usual parameters and a Batch normalization layer. This kind of layer normalize the output of the previous layer, ensuring a faster and more stable training.

After this step there is the usual fully-connected layer. In this case it is formed by three layers with 1024, 128 and 3 neurons.

5 Hyperparameter tuning

All algorithms previously presented are really family of algorithms composed by every possible values of the hyperparameters. To find the best combination of values for the problem it's necessary to firstly find a way to traverse the hyperparameter space. Various techniques have been used in literature such as RandomSearch or the one also used in this project: GridSearch. GridSearch exhausively searches each possible combination of hyperparameter and for this reason is computationally expensive. As a consequence in this project GridSearch has been applied only on EzNet, while on the other two only the network architecture has been tuned. Having only one hyperparameter eliminates the need of GridSearch since the possible values can simply be tested sequentially.

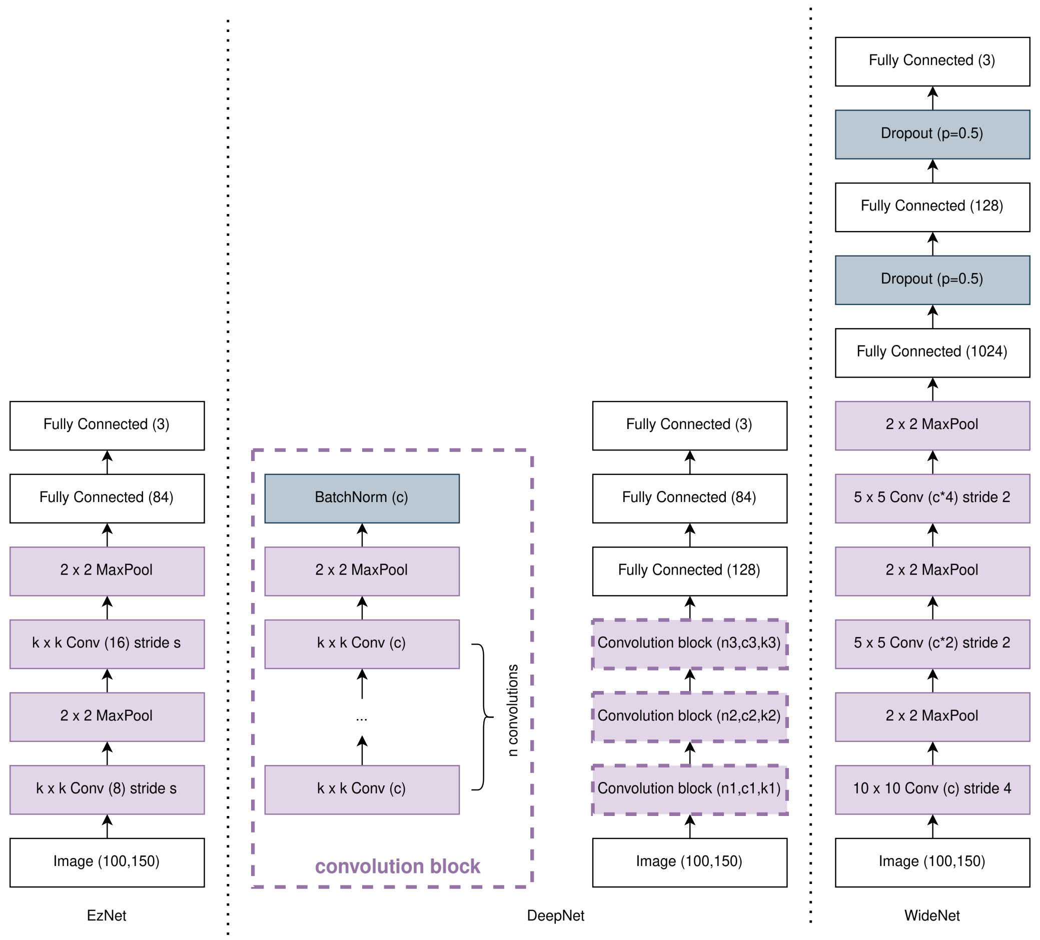


Figure 4: overview of the three model architectures

Once decided how to traverse the hyperparameter space, it is necessary to test the performance of the model with each hyperparameter in an accurate way. Here's where Nested Cross Validation comes into play

K-Fold Cross Validation divides the dataset into K fold of roughly the same size. On each iteration a different fold is chosen as test part and the network is trained using all other folds as training set. The resulting losses for each fold are then averaged. The result obtained is a way to estimate the expected risk of the algorithm.

Nested Cross Validation just adds an outer loop of fold subdivision, in this way the best hyperparameter is selected based on the performance on the inner folds but is then tested on the external testing part, thus ensuring separation between the data used for the model selection and data used for test.

5.1 EzNet tuning using grid search and nested cross validation

Due to the computation cost of the task, only a small subset of hyperparameter has been chosen:

- **Learning rate**: 0.001, 0.0005
- **Model architecture**: (5,2),(3,2),(3,1)

Moreover the test has been limited to a nested cross validation with $k = 3$ both in the internal and external loop.

The hyperparameter combination of learning rate 0.001 and a model architecture with kernel size 5×5 and stride 2 has performed better than other combinations in every fold, for this reason this will the chosen combination for the next steps.

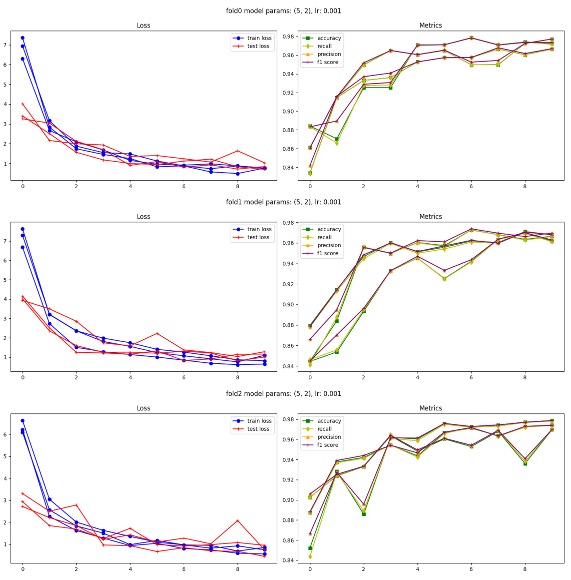


Figure 5: training loss and various metrics for EzNet with the best hyperparameters for each fold of nestedCV

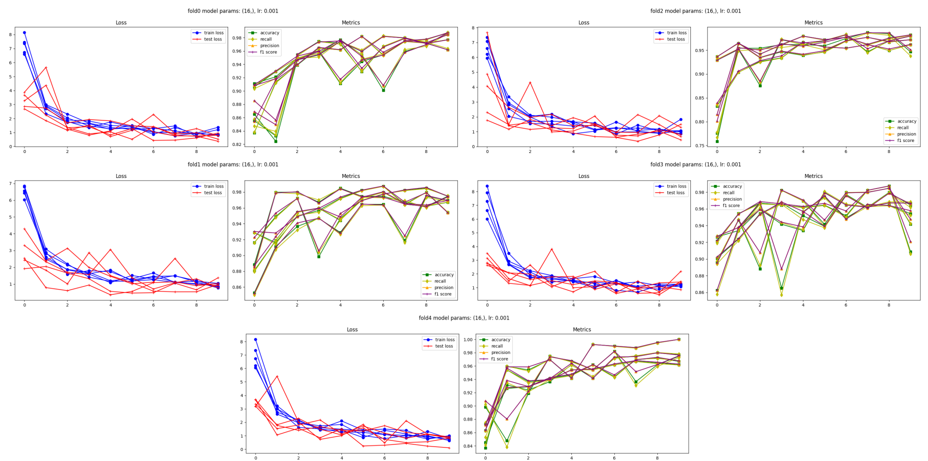


Figure 6: training loss and various metrics for WideNet with $c = 16$ in each fold of nested cv

5.2 WideNet tuning

For this network three different number of channel for the first convolution have been chosen: 8,16,32. In this case the hyperparameters have been tuned using cross validation with $k = 5$ both in the internal and external loop.

This time results are less easy to interpret, in fact two different hyperparamters both have performed well. $c = 8$ was chosen in two folds as the best hyperparameter, but in the end $c = 16$ was preferred because not only was chosen in more folds, but had an over all lower test loss.

5.3 DeepNet tuning

For DeepNet the architectures of choice were:

- (1, 8, 11), (2, 16, 5), (2, 32, 3)
- (1, 8, 5), (1, 16, 5), (2, 32, 3)
- (3, 8, 3), (3, 16, 3), (2, 32, 3)

These have been tested using cross validation with $k = 5$ both in the internal and external loop.

In this case the first two architectures seem to yield better results than the third one on most folds. The choice in the end was the first one, since it had slightly lower loss overall and a more consistent behaviour.

6 Evaluation and analysis

With fixed hyperparameters, the last step is to evaluate the models performance. This evaluation will be composed of two steps: in the first one k-fold cross validation is computed for all models in order to obtain a risk estimate and in the end the performance of the model is also tested on the test set, that has not been involved in the hyperparameter selection process.

6.1 Risk estimate for each model

For the analysis, 8-fold cross validation has been applied to each model for 20 epochs in order to obtain the expected risk for all models. The result is shown in figure 7 and are all more or less comparable, with DeepNet scoring slightly better than the other two.

In figure 8 is represented the mean across folds of accuracy for all models. It is possible to notice how all CNN perform exceptionally well. This is mainly due to the simple nature of the data and could be a problem when trying to apply to a real world scenario. To further test them on a more complex task, models trained on this dataset have been tested on an unseen and more generalized dataset in 7.

In figure 11 is possible to see the full training for each fold. From here is even more clear that, while KissNet struggles to lower the train loss in a clear state of underfitting, other models have no problem in learning the dataset.

Also, while some models exhibit some variation in the test loss, none shows sign of overfitting.

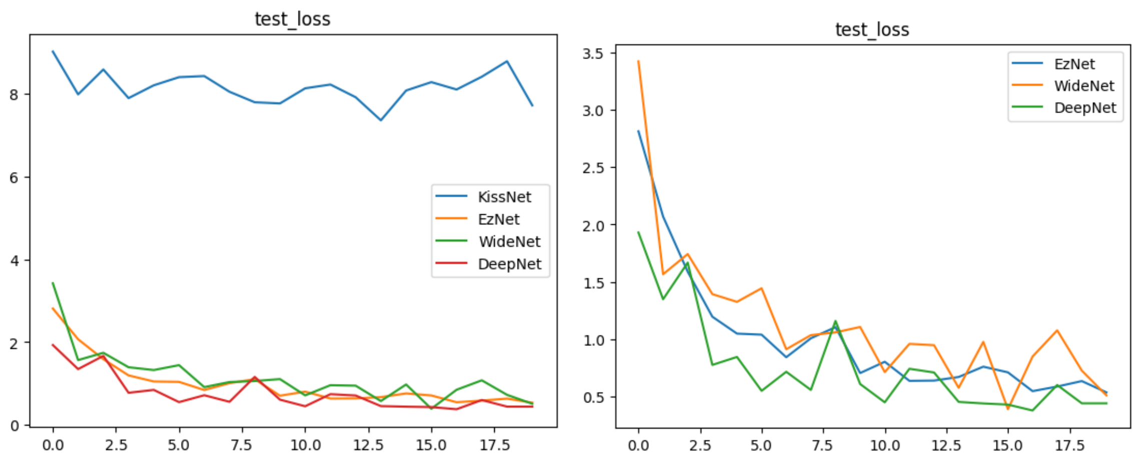


Figure 7: Expected risk for all models (left) and with a focus on the CNNs (right)

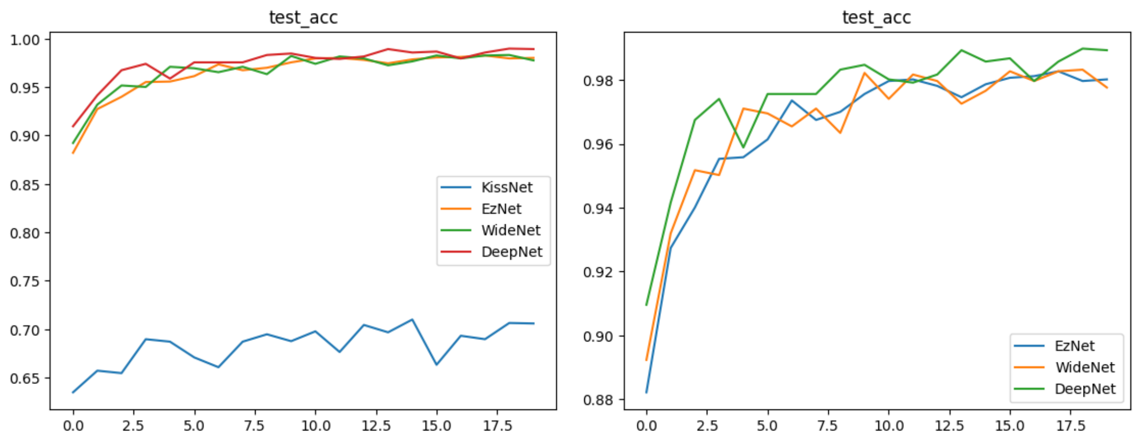


Figure 8: Mean Cross validation accuracy for all models (left) and with a focus on the CNNs (right)

6.2 Evaluation against test set

In this case each model is trained on the training set for 20 epochs. The test set is then sent as input to the resulting predictor for the evaluation.

The results in Figure 12 show that even the KissNet has decent performance on this specific set, however it is clear the difference with Convolutional Neural Networks, that are able to reach a perfect score.

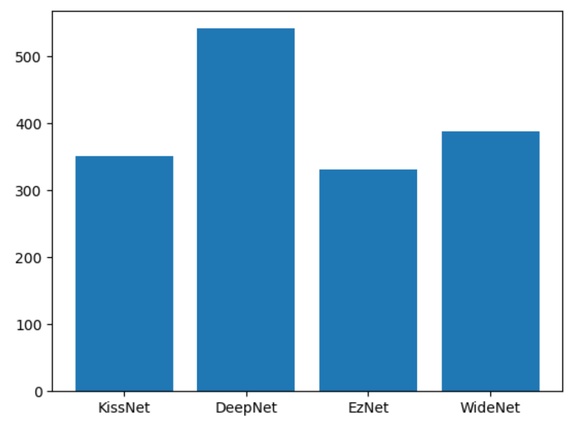


Figure 9: time needed to train the model on the dataset for 20 epochs (s)

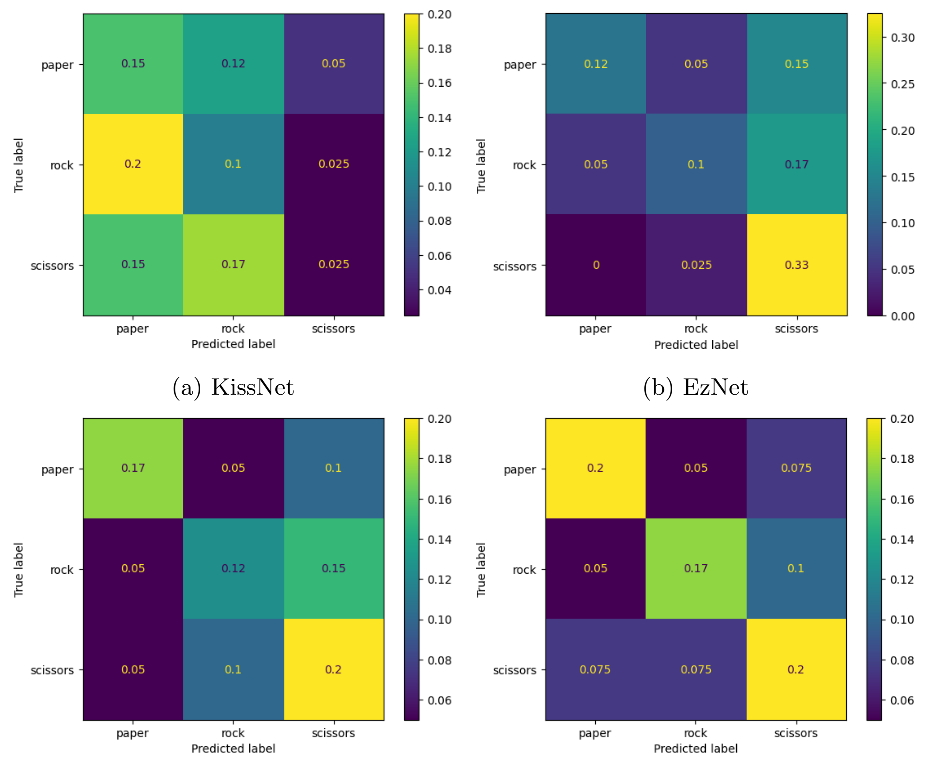
One last concern is training time. If the aim is to have a good predictor for the dataset then each of the presented models is similarly good, however looking at figure 9, it is possible to see how choosing DeepNet instead of EzNet causes around 60% increase in execution time.

7 Generalization

As previously shown, most of the implemented networks perform well on this specific dataset, thus the choice should be made based on computational complexity and thraining time. However there are cases where the real world scenario is more complex, for example there could be different lighting conditions, a different background color or perspective. This is where the higher complexity models shine if trained correctly.

For this experiment a second training set has been created with pictures taken of 5 different subjects. No indication was given them except for the hand gesture needed, as a result there is a more various representation of background, lighting conditions and perspective.

After collecting the sample the set has been tested on the models trained on the original dataset for 20 epochs.



(a) KissNet (b) EzNet

(c) WideNet (d) DeepNet

Figure 10: Confusion matrix of the generalized dataset

The confusion matrix of the prediction is shown in 10. It is possible to see how this dataset is more challenging than the original one. KissNet shows all of its limitations as it is able to correctly classify with a decent amount of true positives only the `paper` class, only because it's its most guessed one. The same is true for EzNet and `scissors`.

A better performance is obtained by WideNet. This time we start to see the diagonal shape forming, but has some trouble in distinguishing `rock` samples, that are often misclassified as `scissors`.

The best performance of the presented models is from DeepNet. While it's still far from perfect this shows how the higher complexity of the network allows to better learn features of the hand shape independently from the condition in which the photo has been taken.

In general however performances are not nearly as good as the one obtained on the first dataset. This further shows how simple is the original dataset and the challenge of developing a network able to generalize the information contained.

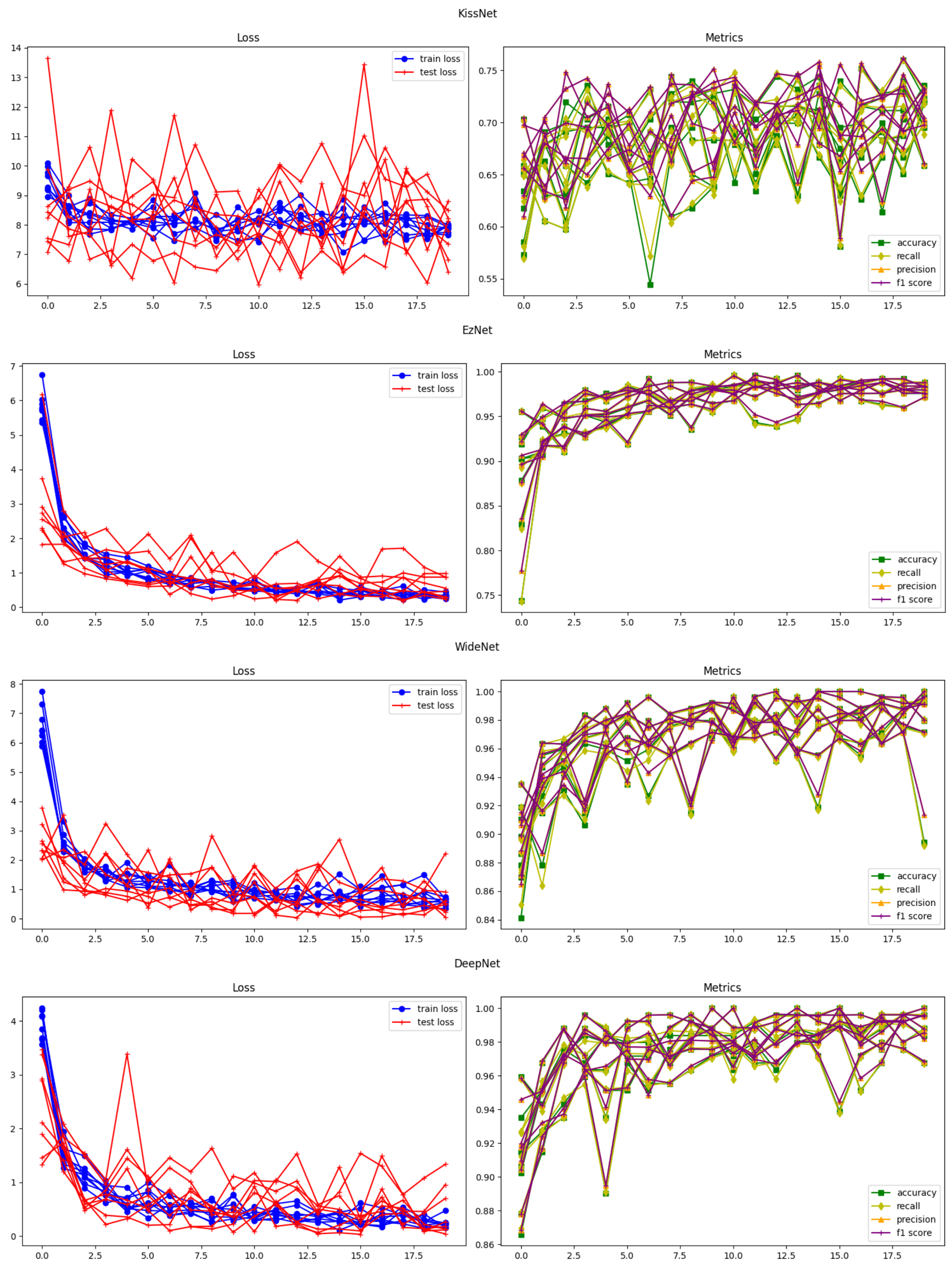


Figure 11: 8-Fold Cross Validation

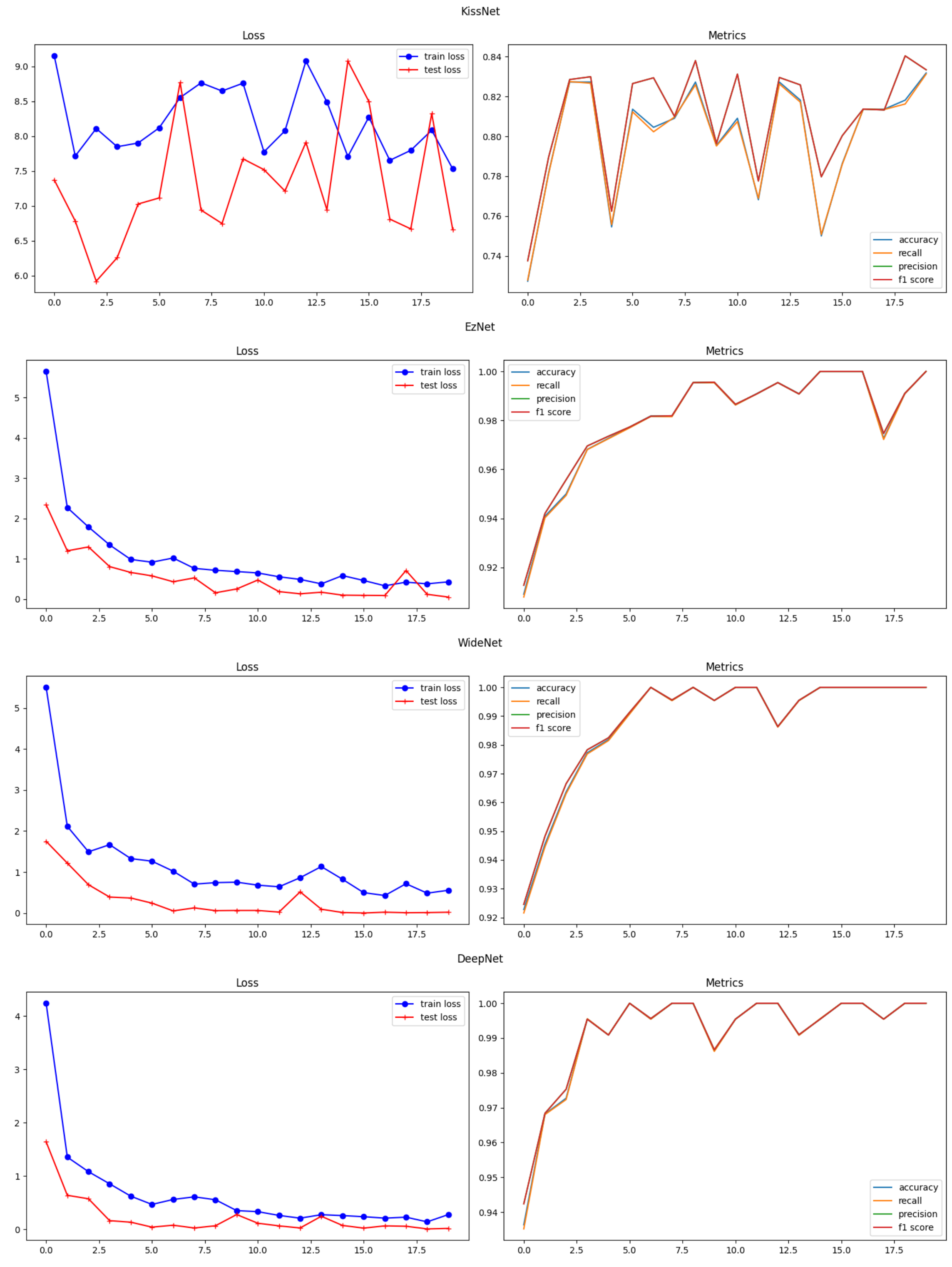


Figure 12: Models performance on test set