Fingers Recognition

Computational Intelligence and Deep Learning

Leonardo Bellizzi

GOAL & OVERVIEW

Project goal was to built up different systems with different architectures able to make image label prediction successfully.

Another objective was to observe final results critically and possibly recognize the best model in the final models pool.To develop an accurate system two CNN were used. Two CNN were written and developed from scratch, in the first part of the project, one shallow and the other one deeper, then for the second network a hyper tuning phase were set up to figure out if with some other parameters the Network behaves in the same way.

The second part of this project leverage on a pre-trained network (ResNet-50) that use ImageNet weights and has been modified to cope with the specific task of the project. Then with this pretrained network, two optimizers (SGD, Adam) were tested with and without Data Augmentation.
When data augmentation is used, all ResNet50 layers were frozen to figure out some advantages can be obtained like faster execution. Eventually, the two augmented networks with the two optimizers were fine tuned by unfrozen final layers.

DATASET

Dataset was found at this link: https://www.kaggle.com/datasets/koryakinp/fingers/. Dataset contains 21600 images of left and right hands fingers.

All images are 128x128 and images own to 12 classes.
Training set contains 18000 images (1500 images per class) and the Test set contains 3600 images.
Training set was already balanced and to use It on Google Collab the upload on Google Drive was necessary.

Images own to these classes (0L,1L,2L,3L,4L,5L,0R,1R,2R,3R,4R,5R)

DATASET PREPROCESSING

Initially, dataset folder contained two subfolders train and test. Images were without labels so a function that takes label from the image path were used. For example, the path was ‘’/train/fff79b8e-4d29-…-4e98_2L.png” and the custom function taken the path returned the label.

Instead of using **train_test_split**, **image_dataset_from_directory** of keras was used. With this, folders were re organized hierarchically

train

—1L

…..png

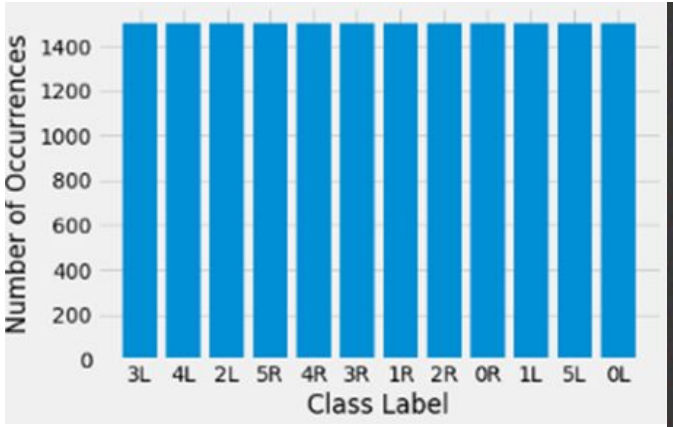
….png

—2L

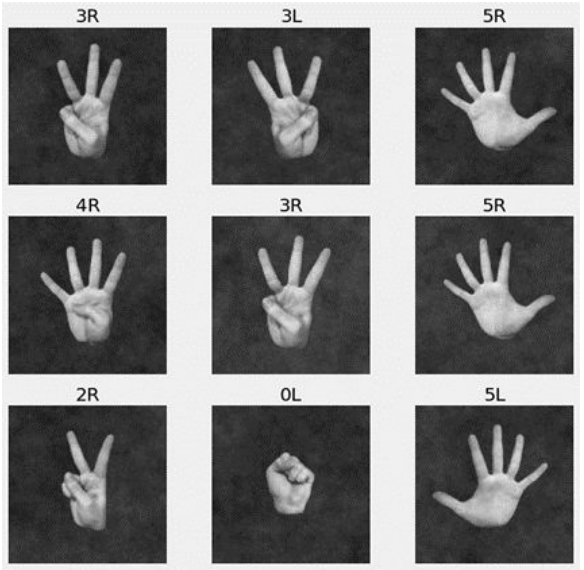
…..png

….png

DATA DISTRIBUTION AND PLOT




```
Class 3L has 1500 items
Class 4L has 1500 items
Class 2L has 1500 items
Class 5R has 1500 items
Class 4R has 1500 items
Class 3R has 1500 items
Class 1R has 1500 items
Class 2R has 1500 items
Class 0R has 1500 items
Class 1L has 1500 items
Class 5L has 1500 items
Class 0L has 1500 items
```

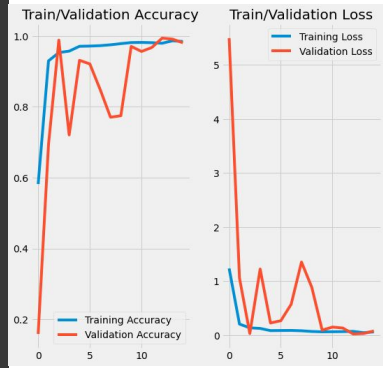


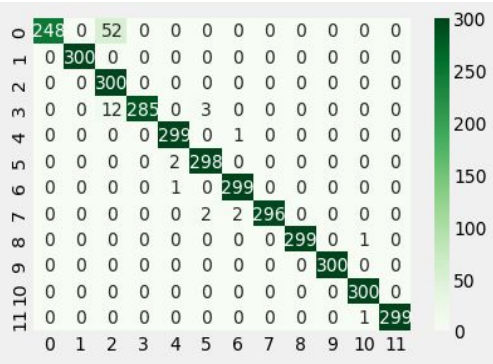
FIRST CNN

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_9 (Rescaling)      (None, 128, 128, 3)       0

 random_rotation_8 (RandomRo  (None, 128, 128, 3)       0
 tation)

 random_zoom_9 (RandomZoom)   (None, 128, 128, 3)       0

 conv2d_28 (Conv2D)           (None, 127, 127, 32)      416

 batch_normalization_28 (Bat  (None, 127, 127, 32)      128
 chNormalization)

 max_pooling2d_28 (MaxPoolin  (None, 63, 63, 32)        0
 g2D)

 conv2d_29 (Conv2D)           (None, 62, 62, 64)        8256

 batch_normalization_29 (Bat  (None, 62, 62, 64)        256
 chNormalization)

 max_pooling2d_29 (MaxPoolin  (None, 31, 31, 64)        0
 g2D)

 conv2d_30 (Conv2D)           (None, 30, 30, 128)       32896

 batch_normalization_30 (Bat  (None, 30, 30, 128)       512
 chNormalization)

 max_pooling2d_30 (MaxPoolin  (None, 15, 15, 128)       0
 g2D)

 flatten_10 (Flatten)         (None, 28800)             0

 dropout_10 (Dropout)         (None, 28800)             0

 dense_20 (Dense)             (None, 150)               4320150

 dense_21 (Dense)             (None, 12)                1812

=================================================================
Total params: 4,364,426
Trainable params: 4,363,978
Non-trainable params: 448
_________________________________________________________________
```

The first network has more the 4 millions of trainable parameters with Data Augmentation layers to prevent overfitting.

The model was compiled with Adam that is more sensitive to local minima in fact from plots oscillations are quite visible.





	precision	recall	f1-score	support
0	1.00	0.83	0.91	300
1	1.00	1.00	1.00	300
2	0.82	1.00	0.90	300
3	1.00	0.95	0.97	300
4	0.99	1.00	0.99	300
5	0.98	0.99	0.99	300
6	0.99	1.00	0.99	300
7	1.00	0.99	0.99	300
8	1.00	1.00	1.00	300
9	1.00	1.00	1.00	300
10	0.99	1.00	1.00	300
11	1.00	1.00	1.00	300
accuracy			0.98	3600
macro avg	0.98	0.98	0.98	3600
weighted avg	0.98	0.98	0.98	3600

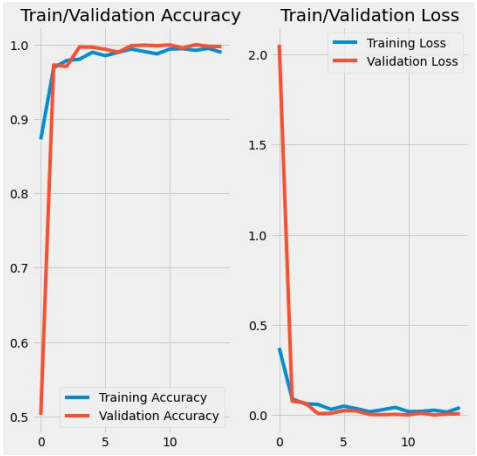
From the Confusion matrix it's visible that model made some misclassification errors due to Adam sensibility to data noise. Mistakes are mostly done on first class, even though globally on test model reaches out 98% of Accuracy.

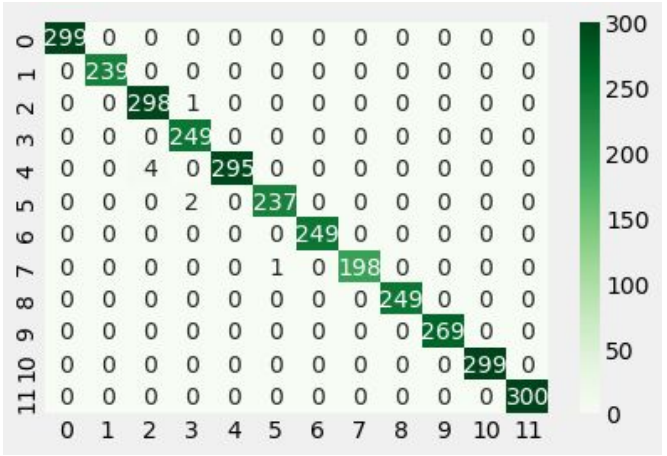
SECOND CNN

The second CNN is deeper than the previous one and filters with 512 units are used. Here data augmentation was not an option and model was compiled with Adam optimizer

Layer (type)	Output Shape	Param #
rescaling (Rescaling)	(None, 128, 128, 3)	0
random_rotation (RandomRotation)	(None, 128, 128, 3)	0
random_zoom (RandomZoom)	(None, 128, 128, 3)	0
conv2d (Conv2D)	(None, 127, 127, 32)	416
batch_normalization (BatchNormalization)	(None, 127, 127, 32)	128
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_1 (Conv2D)	(None, 62, 62, 64)	8256
batch_normalization_1 (BatchNormalization)	(None, 62, 62, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 31, 31, 64)	0
conv2d_2 (Conv2D)	(None, 30, 30, 128)	32896
batch_normalization_2 (BatchNormalization)	(None, 30, 30, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 15, 15, 128)	0
conv2d_3 (Conv2D)	(None, 14, 14, 256)	131328
batch_normalization_3 (BatchNormalization)	(None, 14, 14, 256)	1024
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 256)	0
conv2d_4 (Conv2D)	(None, 6, 6, 512)	524800
batch_normalization_4 (BatchNormalization)	(None, 6, 6, 512)	2048
max_pooling2d_4 (MaxPooling2D)	(None, 3, 3, 512)	0
conv2d_5 (Conv2D)	(None, 2, 2, 512)	1049088
batch_normalization_5 (BatchNormalization)	(None, 2, 2, 512)	2048
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 150)	307350
dense_1 (Dense)	(None, 12)	1812

Total params: 2,061,962
Trainable params: 2,058,954
Non-trainable params: 3,008





	precision	recall	f1-score	support
0	1.00	1.00	1.00	299
1	1.00	1.00	1.00	239
2	0.99	1.00	0.99	299
3	0.99	1.00	0.99	249
4	1.00	0.99	0.99	299
5	1.00	0.99	0.99	239
6	1.00	1.00	1.00	249
7	1.00	0.99	1.00	199
8	1.00	1.00	1.00	249
9	1.00	1.00	1.00	269
10	1.00	1.00	1.00	299
11	1.00	1.00	1.00	300
accuracy			1.00	3189
macro avg	1.00	1.00	1.00	3189
weighted avg	1.00	1.00	1.00	3189

From experimental results models seems to behave good during training since curves are always very close between each other. On test, accuracy was found to be near 100%. 8 misclassification overall

SECOND CNN HYPERTUNING

For the Hyper Tuning the entire second CNN was taken, and some parameters were tuned at the end of the CNN. The function uses the **Hyperparameters** object from the **hyperas** library to define several hyperparameters that can be tuned during the model training process. The number of units in the dense layer, the activation function used in the dense layer, whether to use dropout, and the learning rate of the optimizer are all defined as hyperparameters. This allows the model to be trained using a technique such as random search or Bayesian optimization to find the best set of hyperparameters for a given problem.

Random Search was preferred to **Grid Search.** The first one is more efficient from a time perspective, since not all parameters combination are tested. The second type guarantees a complete combination parameter coverage but takes more time, so the first solution is better if retrieve a good approximation faster is required.
More in general the **Hyper Tuning** can lead to better use of resources, Increased robustness and better generalization which means the risk of overfitting is lower and the model can perform better on unseen data.

HYPERTUNED PARAMS FIELD

```
model.add(keras.layers.Flatten())
model.add(
    keras.layers.Dense(
        # Tune number of units.
        units=hp.Int("units", min_value=32, max_value=512, step=32),
        # Tune the activation function to use.
        activation=hp.Choice("activation", ["relu", "tanh"]),
    )
)
# Tune whether to use dropout.
if hp.Boolean("dropout"):
    model.add(keras.layers.Dropout(rate=0.25))
model.add(keras.layers.Dense(12, activation="softmax"))
# Define the optimizer learning rate as a hyperparameter.
learning_rate = hp.Float("lr", min_value=1e-4, max_value=1e-2, sampling="log")
```

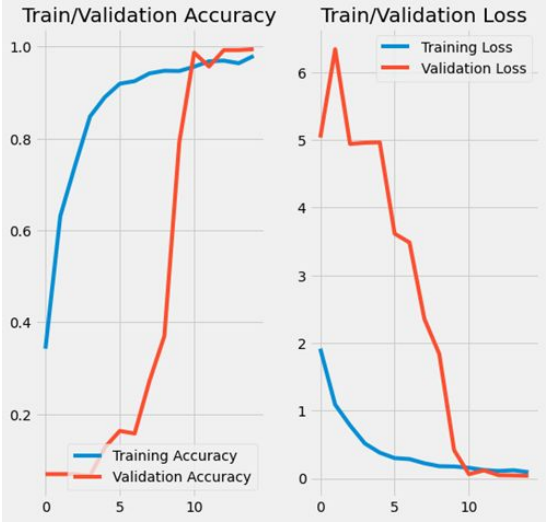
```
Search space summary
Default search space size: 4
units (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 'sampling': 'linear'}
activation (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'tanh'], 'ordered': False}
dropout (Boolean)
{'default': False, 'conditions': []}
lr (Float)
{'default': 0.0001, 'conditions': [], 'min_value': 0.0001, 'max_value': 0.01, 'step': None, 'sampling': 'log'}
```

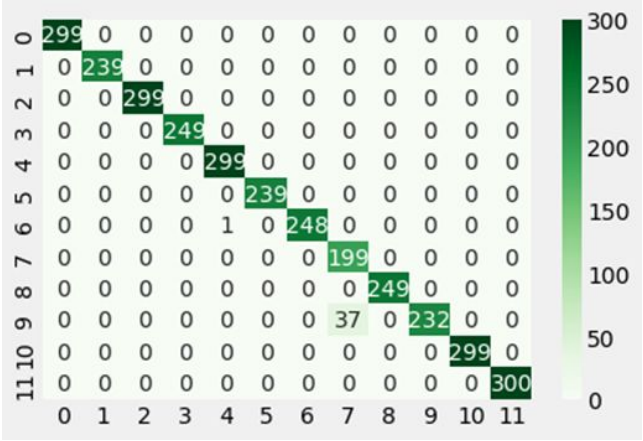
Params during search changed, 3 trial were done with different combination parameters like is shown below. At the end of the trials the first one was taken that achieved a 98% of accuracy.

```
Search: Running Trial #1

Value             |Best Value So Far |Hyperparameter
416               |?                 |units
tanh              |?                 |activation
False             |?                 |dropout
0.0008024         |?                 |lr

Trial 1 Complete [08h 13m 36s]
val_accuracy: 0.9849999845027924

Best val_accuracy So Far: 0.9849999845027924
Total elapsed time: 15h 29m 31s

Search: Running Trial #2

Value             |Best Value So Far |Hyperparameter
288               |416               |units
tanh              |tanh              |activation
False             |False             |dropout
0.0093292         |0.0008024         |lr
```

```
Trial 3 Complete [00h 26m 45s]
val_accuracy: 0.167293231934309

Best val_accuracy So Far: 0.17481203377246857
Total elapsed time: 01h 15m 38s
```

CNN HYPERTUNED RESULTS





	precision	recall	f1-score	support
0	1.00	1.00	1.00	299
1	1.00	1.00	1.00	239
2	1.00	1.00	1.00	299
3	1.00	1.00	1.00	249
4	1.00	1.00	1.00	299
5	1.00	1.00	1.00	239
6	1.00	1.00	1.00	249
7	0.84	1.00	0.91	199
8	1.00	1.00	1.00	249
9	1.00	0.86	0.93	269
10	1.00	1.00	1.00	299
11	1.00	1.00	1.00	300
accuracy			0.99	3189
macro avg	0.99	0.99	0.99	3189
weighted avg	0.99	0.99	0.99	3189

The hypertuned version reached 98% of accuracy on test performing worst w.r.t. to its non tuned version. This is due to pool of parameters chosen to tune the network. It's possible to figure out that there are more misclassification error plus a struggle in minimize loss and maximize accuracy from validation set until the 10th epoch. Seems to underfit until the 10th epoch to be again aligned until the end of the training.

CUSTOM CNN USING RESNET50

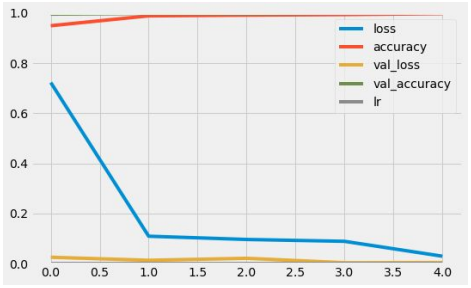
Overview:

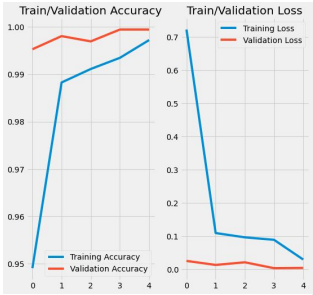
- Model built with Adam without Data Augmentation
- Model built with SGD without Data Augmentation
- Model built with Adam with Data Augmentation
- Model built with SGD with Data Augmentation
- Fine Tuning with Adam
- Fine Tuning with SGD

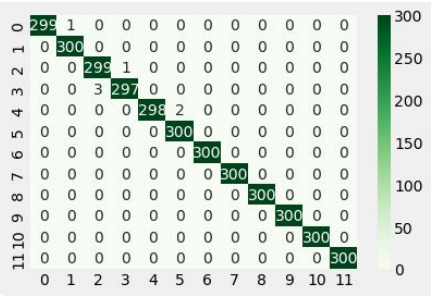
RESNET50 INSTANTIATION FOR FIRST TWO NN

Model built with ADAM and SGD without Data Augmentation were instantiated on a totally unfrozen ResNet50 network. Since this is transfer learning ResNet50 works as feature extractor here and then a custom classifier is attached to the network.By using all layers unfrozen and ResNet as feature extractor there are several advantages like **saving time and computation, high-quality features, transfer learning.**

- ADAM results (99% of accuracy on test)

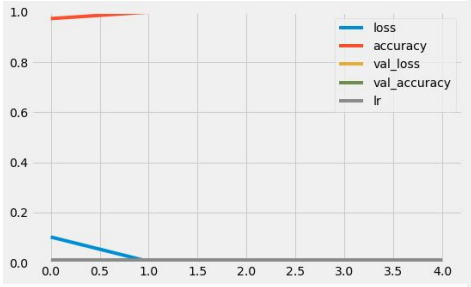


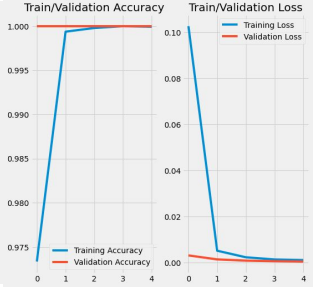


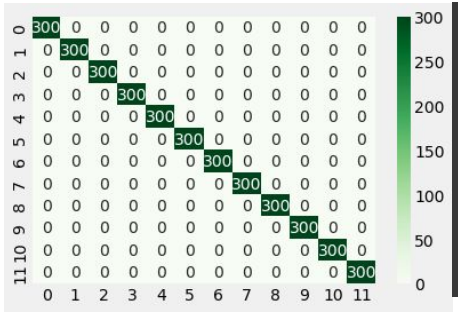


	precision	recall	f1-score	support
0	0.97	0.98	0.97	300
1	0.98	0.97	0.97	300
2	0.99	1.00	1.00	300
3	0.99	0.99	0.99	300
4	1.00	0.99	1.00	300
5	0.99	1.00	1.00	300
6	0.99	0.98	0.99	300
7	0.98	0.99	0.99	300
8	1.00	0.99	1.00	300
9	0.99	0.99	0.99	300
10	1.00	1.00	1.00	300
11	1.00	1.00	1.00	300
accuracy			0.99	3600
macro avg	0.99	0.99	0.99	3600
weighted avg	0.99	0.99	0.99	3600

- SGD results (100% of accuracy on test)





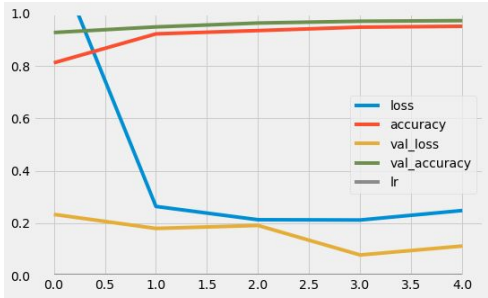


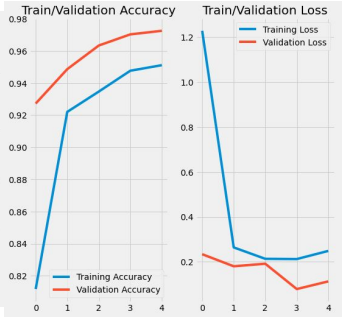
	precision	recall	f1-score	support
0	1.00	1.00	1.00	300
1	1.00	1.00	1.00	300
2	1.00	1.00	1.00	300
3	1.00	1.00	1.00	300
4	1.00	1.00	1.00	300
5	1.00	1.00	1.00	300
6	1.00	1.00	1.00	300
7	1.00	1.00	1.00	300
8	1.00	1.00	1.00	300
9	1.00	1.00	1.00	300
10	1.00	1.00	1.00	300
11	1.00	1.00	1.00	300
accuracy			1.00	3600
macro avg	1.00	1.00	1.00	3600
weighted avg	1.00	1.00	1.00	3600

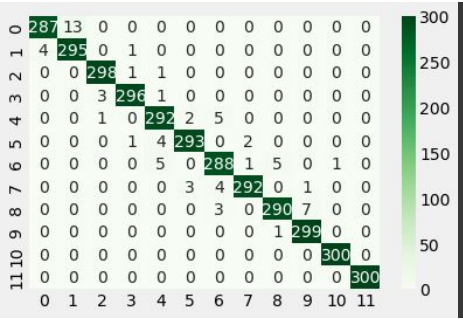
RESNET50 INSTANTIATION FOR THIRD AND FOURTH NN

In this case ResNet layers are completely frozen to take advantages like **stability**, **simplicity and reliable results.** With ResNet totally frozen maybe whole model built around it could be less flexible than its non frozen version. For the following models data augmentation was added (RandomRotation and RandomZoom)

- Adam (97% of accuracy on test set)

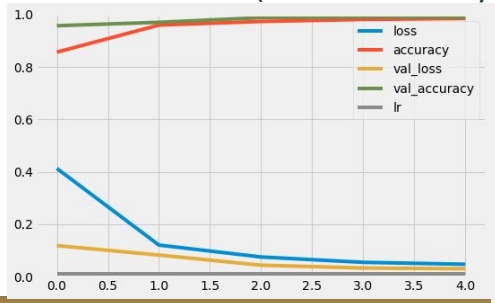


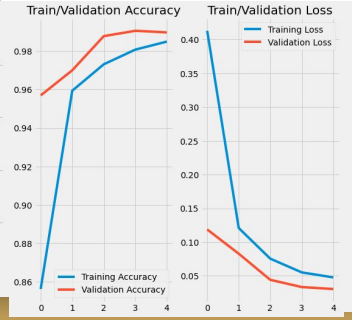


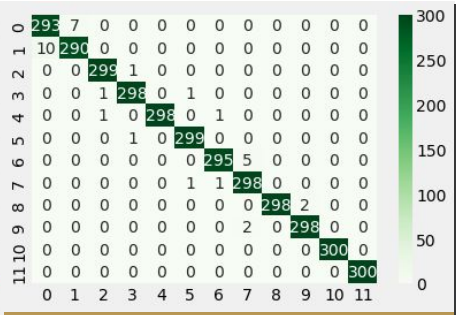


	precision	recall	f1-score	support
0	0.99	0.96	0.97	300
1	0.96	0.98	0.97	300
2	0.99	0.99	0.99	300
3	0.99	0.99	0.99	300
4	0.96	0.97	0.97	300
5	0.98	0.98	0.98	300
6	0.96	0.96	0.96	300
7	0.99	0.97	0.98	300
8	0.98	0.97	0.97	300
9	0.97	1.00	0.99	300
10	1.00	1.00	1.00	300
11	1.00	1.00	1.00	300
accuracy			0.98	3600
macro avg	0.98	0.98	0.98	3600
weighted avg	0.98	0.98	0.98	3600

- SGD (99% of accuracy on test set)







	precision	recall	f1-score	support
0	0.97	0.98	0.97	300
1	0.98	0.97	0.97	300
2	0.99	1.00	1.00	300
3	0.99	0.99	0.99	300
4	1.00	0.99	1.00	300
5	0.99	1.00	1.00	300
6	0.99	0.98	0.99	300
7	0.98	0.99	0.99	300
8	1.00	0.99	1.00	300
9	0.99	0.99	0.99	300
10	1.00	1.00	1.00	300
11	1.00	1.00	1.00	300
accuracy			0.99	3600
macro avg	0.99	0.99	0.99	3600
weighted avg	0.99	0.99	0.99	3600

SOME CONSIDERATIONS

In general, a pre-trained network with fully frozen layers may perform worse than a network with fully unfrozen layers because the frozen network is not able to fully adapt to the new requirements of the specific task.

When the layers of the pre-trained network are fully frozen, the weights cannot be modified during the training of the model that uses the network as a feature extractor. This means that the network cannot fully adapt to the new requirements of the specific task, which can result in lower performance.

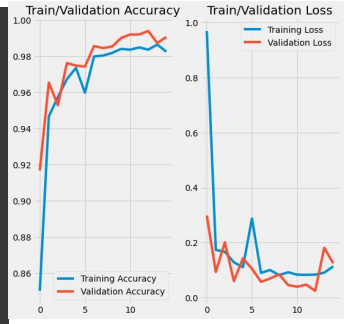
On the other hand, when the layers of the pre-trained network are fully unfrozen, it is possible to continue training the network on your own data, adapting it to the specific requirements of the task. This helps to improve the performance of the model compared to using a fully frozen network.

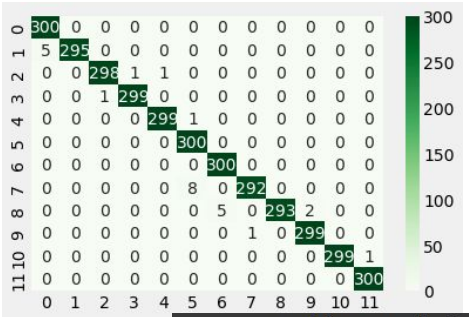
Ultimately, the choice between a network with fully frozen layers and one with fully unfrozen layers will depend on the specific requirements of the task and the availability of data for training the model.

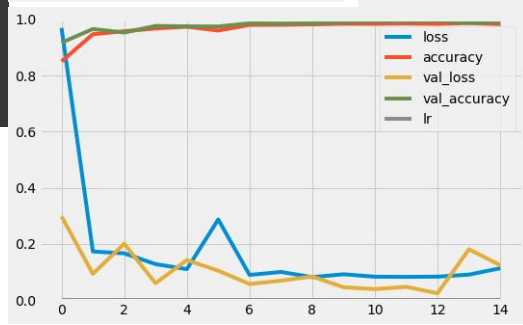
FINE TUNING WITH ADAM

Here the ResNet50 architecture are totally frozen except the last 6 layers. This is the maximum number of layers that allows non-overfitting situation. Unfrozen layers can adjust their weights to fit the training data too closely, leading to overfitting if they are allowed to modify their weights too much. This can result in a model that performs well on the training data but poorly on new, unseen data.

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, 128, 128, 3)]     0

 sequential_2 (Sequential)   (None, 128, 128, 3)       0

 resnet50 (Functional)       (None, None, None, 2048)  23587712

 batch_normalization_2 (Batc  (None, 4, 4, 2048)       8192
 hNormalization)

 flatten_2 (Flatten)         (None, 32768)             0

 dense_4 (Dense)             (None, 1024)              33555456

 dropout_2 (Dropout)         (None, 1024)              0

 dense_5 (Dense)             (None, 12)                12300

=================================================================
Total params: 57,163,660
Trainable params: 34,627,596
Non-trainable params: 22,536,064
_________________________________________________________________
```





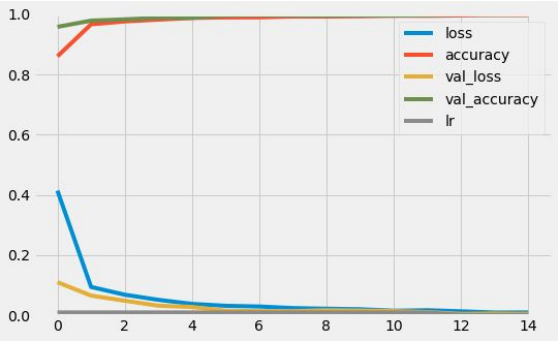


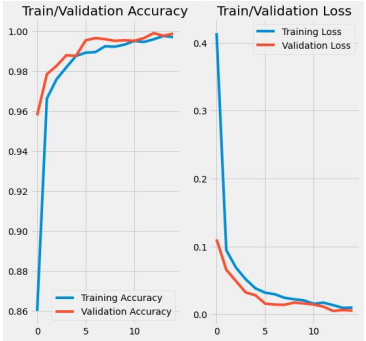
	precision	recall	f1-score	support
0	0.98	1.00	0.99	300
1	1.00	0.98	0.99	300
2	1.00	0.99	0.99	300
3	1.00	1.00	1.00	300
4	1.00	1.00	1.00	300
5	0.97	1.00	0.99	300
6	0.98	1.00	0.99	300
7	1.00	0.97	0.98	300
8	1.00	0.98	0.99	300
9	0.99	1.00	1.00	300
10	1.00	1.00	1.00	300
11	1.00	1.00	1.00	300
accuracy			0.99	3600
macro avg	0.99	0.99	0.99	3600
weighted avg	0.99	0.99	0.99	3600

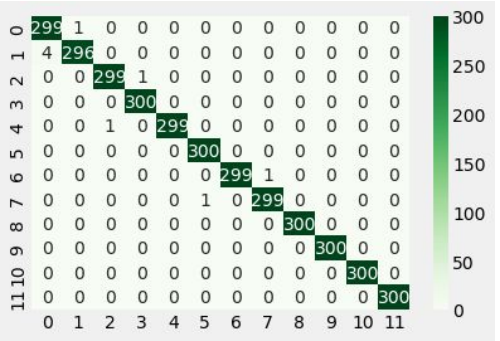
This model performed on test set with 99% of accuracy even if there is some instability looking to the plots, due to Network depth and params that made Adam unstable

FINE TUNING WITH SGD

Same network configuration is used but the optimizer is SGD, that theoretically must be more stable than ADAM even if the network is deeper.







	precision	recall	f1-score	support
0	0.99	1.00	0.99	300
1	1.00	0.99	0.99	300
2	1.00	1.00	1.00	300
3	1.00	1.00	1.00	300
4	1.00	1.00	1.00	300
5	1.00	1.00	1.00	300
6	1.00	1.00	1.00	300
7	1.00	1.00	1.00	300
8	1.00	1.00	1.00	300
9	1.00	1.00	1.00	300
10	1.00	1.00	1.00	300
11	1.00	1.00	1.00	300
accuracy			1.00	3600
macro avg	1.00	1.00	1.00	3600
weighted avg	1.00	1.00	1.00	3600

This model reached out up to 99.7% of accuracy on test set.

CONCLUSIONS

After the overview it’s not easy to choose the best. Each one has its peculiarities but from the pretrained network pool, those with SGD are the best while from those built from scratch the second network works better with this kind of problem. The Hypertuned version could be discarded.

SGD worked better for its native structure because it’s less sensitive to local minima w.r.t Adam and it could be observable from the plot especially from the first CNN built from scratch.

Adam takes information by mean and gradient variance to adapt dynamically the learning rate but this kind of structure can make Adam more sensitive to data noise.

For this task when the networks has more params it’s better to pick up a model built on SGD while if the params are not so high ADAM could be the best choice