Tilburg University

Data Science and Society Master program

Deep Learning Assignment 2022
Report on Task 1 – Image Classification

Group Number: 18
Student names: Anh Ngoc Pham

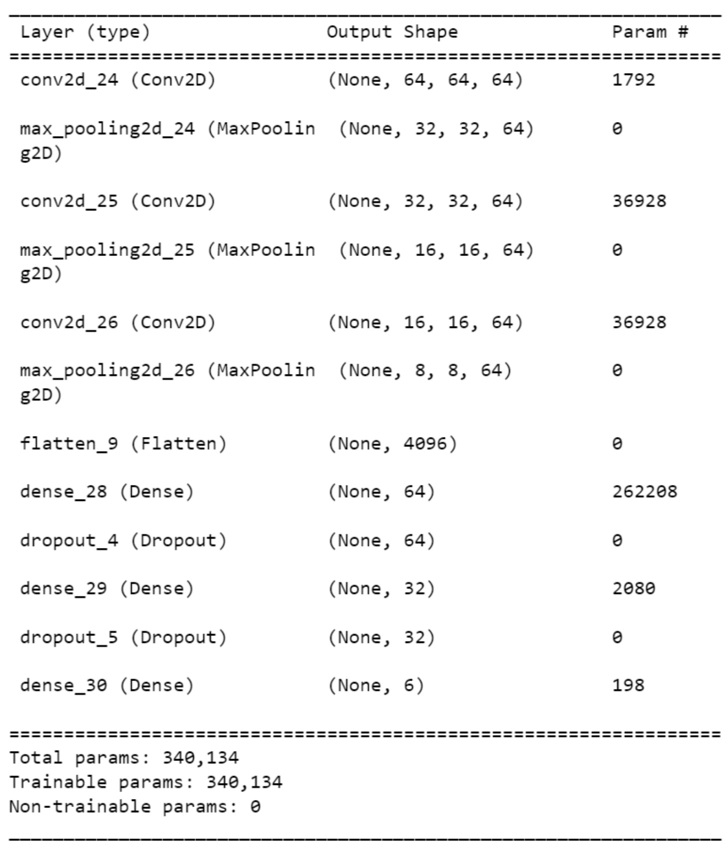
Student numbers: 2086454

Date: 10 October 2022

1. Summary of the final model:

In this project, we built a total of 9 models. We started with the baseline one (See Annex 6 for summary) and then experimented with six types of optimizing techniques. We justified the most suitable way to finetune the baseline model based on these results. We built our final model from this conclusion and tested it on test data. In addition, we also ran a test on VGG 16 model to compare the prediction advantages of this model regarding our optimized model.

The final model had a total of 12 layers. It first started with a convolutional layer with 64 filters of size 3 x 3. We applied the activation function Relu on this layer, set stride as default (1,1) and added the padding as 'same' toward up-down and left-right of the input. A max-pooling layer followed this layer with a pool size of 2 by 2. Next, we repeated this block of those two layers two more times. We ended the three convolutional and pooling processes with a 'flatten' layer, then proceeded with two fully-connected layers (Dense layer). Each of these two Dense layers was followed by a drop-out layer with a rate of 0.4. Both these fully-connected layers ran with Relu activation and L2 norm regularizer. The first dense layer had a size of 64 units, and 32 units formed the second one. We wrapped the model with a Dense output layer of size six and softmax activation. The model was compiled with an 'Adam' optimizer, 'categorical_crossentropy' for loss calculation, and 'accuracy' as the metrics (Figure 1). Train and validation data were combined to fit the final model. We set the batch size at 32 and stopped the model at seven epochs.


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_24 (Conv2D)          (None, 64, 64, 64)        1792

 max_pooling2d_24 (MaxPoolin  (None, 32, 32, 64)       0
 g2D)

 conv2d_25 (Conv2D)          (None, 32, 32, 64)        36928

 max_pooling2d_25 (MaxPoolin  (None, 16, 16, 64)       0
 g2D)

 conv2d_26 (Conv2D)          (None, 16, 16, 64)        36928

 max_pooling2d_26 (MaxPoolin  (None, 8, 8, 64)         0
 g2D)

 flatten_9 (Flatten)         (None, 4096)              0

 dense_28 (Dense)            (None, 64)                262208

 dropout_4 (Dropout)         (None, 64)                0

 dense_29 (Dense)            (None, 32)                2080

 dropout_5 (Dropout)         (None, 32)                0

 dense_30 (Dense)            (None, 6)                 198

=================================================================
Total params: 340,134
Trainable params: 340,134
Non-trainable params: 0
_________________________________________________________________
```

Figure 1: Summary of our final model

2. Experiments on optimizers and regularizer

As aforementioned, we implemented six optimizers and regularizers on the baseline model to find the best solution to improve the performance of our network. In general, we rebuilt all six models from the baseline, which was set according to the assignment's requirements. The baseline model consisted of the layers listed below in the same order (See Annex 6 for summary details):

- 3 blocks of the pair of convolutional - max-pooling layers with Relu activation
- a 'flatten' layer

- 2 fully connected layers (Dense) with the size of 64 and 32, respectively. Relu activation was applied for these two layers.
- Output layer: a Dense layer sized 6 with softmax activation
- Model compilation: a compile layer with 'adam' optimizer, loss set as 'categorical_crossentropy' and 'accuracy' for the metrics, running through 20 epochs.

We summarized the experiment's implementation in Table 1. Note that the explanation in Table 1 only mentioned the changes from the abovementioned setting (baseline) during each tryout.

Table 1: Details of our optimizing methods

No.	Name of technique	Explanation of the experiment
1	Resizing to a smaller model	We changed the number of units in the two fully-connected layers to 16 and 8 (instead of 64 and 32). Since the output layer is sized 6, the minimum size of its previous layer could only be 8
2	Resizing to a bigger model	We changed the number of units in the two fully-connected layers to the maximum threshold of 256 and 128, respectively
3	Adding one more layer	We added one more layer to the network after the last fully connected layer of the baseline. This added layer had the size of 16 with Relu activation. Since the model started to overfit very early, the extended layer will force the model to extract more features to learn. However, the baseline loss was high, and the accuracy was relatively low. Forcing feature extraction might mislead the prediction. For this reason, we only wanted to probe the model behaviors by adding one more fully-connected layer without changing the other existing layers. This action would make a fair comparison in the performance of the adding-layer method to the baseline.
4	Adding L2 weight regularizer by Keras default	In both fully-connected layers, we add a weight decay regularizer of 0.001. By that, each weight coefficient value from the matrix of these layers would be added with a factor of 0.001. We used Keras L2 function in this experiment
5	Adding L2 weight regularizer manually (custom)	We built our L2-custom regularizer function manually based on its formal formula. This experiment was to test the difference between the performance of manually-calculated-based L2 and Keras-based L2. The L2 norm of weight was also set at 0.001 in this experiment
6	Applying Dropout with the rate of 0.4	After each fully connected layer, we set a drop-out rate of 40% for its output.

We fitted these models on train data with a batch size of 32 and validated it on validation data. Each process was run through 20 epochs. After each experiment, we plotted the validation loss of the tryout model[1] and measured its performance with loss, accuracy, F1-score, sensitivity, and specificity (Annex 1). These results were our foundation to justify the most proper optimizing solution for the final model.

- ***Experiment results:***

All tryout models showed an improvement except for the bigger model. The bigger model fell into the loss bottom at the earliest stage after five epochs. After that, its loss drastically increased with

[1] Results of all experiments can be found in the Annex section

each epoch. On the other hand, the smaller model succeeded in limiting overfitting until epoch 11th. Yet, after its loss performance stopped improving, we also witnessed a periodically sharp fluctuation in overfitting (Annex 2).

Extending one more layer did not help to improve performance. Although the loss performance in this tryout dipped to the lowest point compared to other peers (minimum loss = 0.5 at seven epochs), the model behaved unstably with severe surges in the loss line (Annex 3).

Both L2-regularizers improved performance by fixing the drastic change after the best fit points. However, their loss stopped improving at epoch 7, the same period as the baseline's (Annex 4).

The dropout technique started to overfit at the latest point compared to its predecessor, at epoch 13. Although its performance also showed a sudden overturn after the best fitting point, the overall loss by epoch scaled in a lower position than any other models (Annex 5). In model evaluation, the Dropout method also got the best score in all performance measures, with the lowest loss and the highest accuracy (83.8%), F1-score, sensitivity, and specificity (Annex 1). For these achievements, 'dropout' became the best optimizer among all the six introduced methods.

Thus, we decided to employ Dropout to fine-tune the final model. We also used an L2 regularizer to constrain fluctuant behaviors around the best fit point. L2 regularizer under the Keras setup (standard) was in favor because its accuracy ranked second among six models (Annex 1).

Details of the final model are described in section 1. We piled up our training and validation set to fit the model with a batch size of 32. Based on loss performance plots (See in the Annex section), we agreed the best cut-off point would be at seven epochs to avoid overfitting.

3. Graphs and results

3.1. Baseline models:

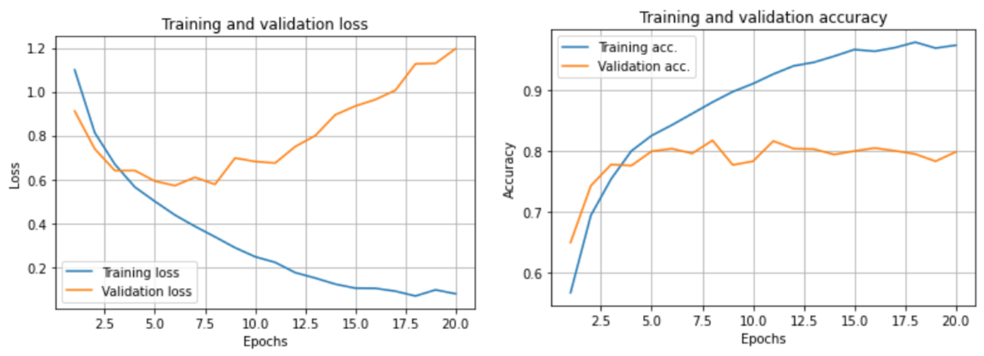


Figure 2: Baseline model performance plot based on loss and accuracy by epoch

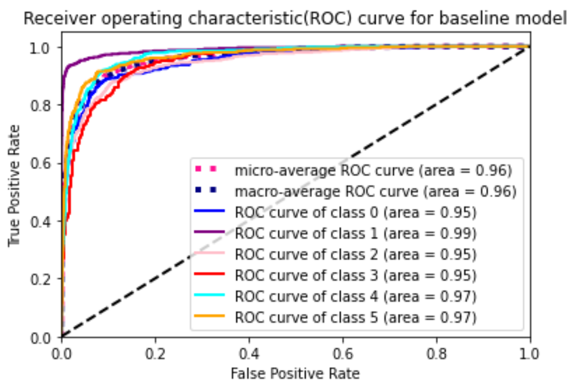


Figure 3: ROC curve of baseline performance on validation set

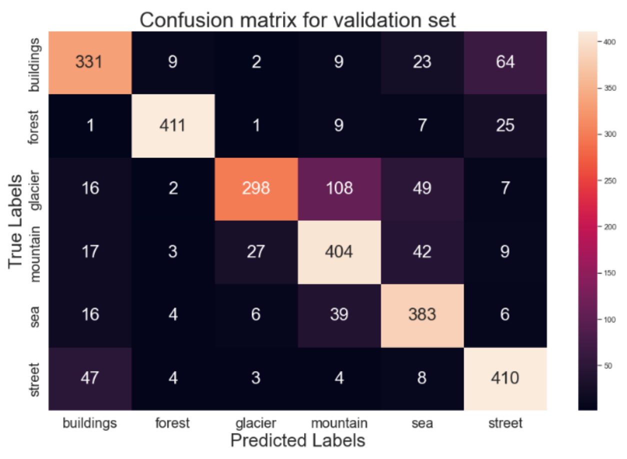


Figure 4: Confusion matrix of baseline performance on validation test

3.2. Our final (optimized) model:

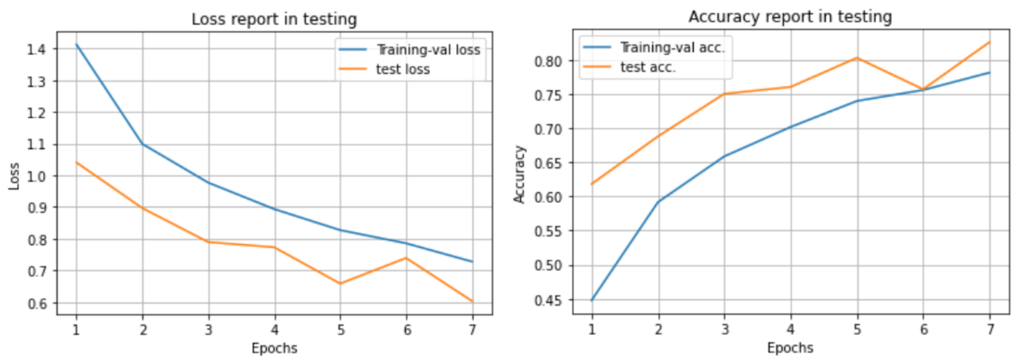


Figure 5: Performance of final model based on loss and accuracy by epoch

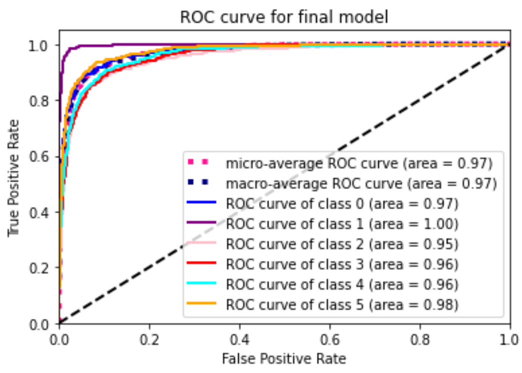


Figure 6: ROC curve of final model's performance on test set

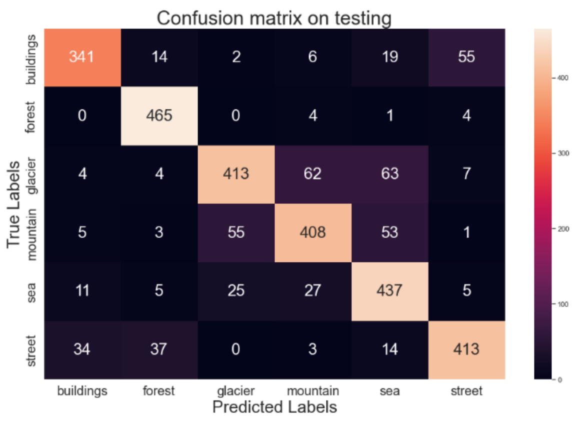
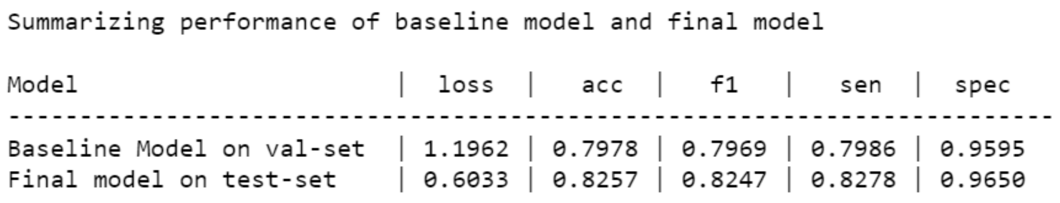


Figure 7: Confusion matrix of final model's performance on test set

3.3. Report performance measures of baseline and final model



```
Summarizing performance of baseline model and final model

Model                     |  loss  |  acc   |   f1   |  sen   |  spec
----------------------------------------------------------------------
Baseline Model on val-set | 1.1962 | 0.7978 | 0.7969 | 0.7986 | 0.9595
Final model on test-set   | 0.6033 | 0.8257 | 0.8247 | 0.8278 | 0.9650
```

Figure 8: Comparison of model performance measures between baseline model and final model

3.4. Discussion of performance between the baseline and final model

Our final optimized model showed a significant improvement compared to the baseline. The accuracy increased to 82.6%, and the loss reduced to half of the baseline level, at 0.6. Moreover, F1-score, sensitivity, and specificity all increased by around 1-2% (Figure 8).

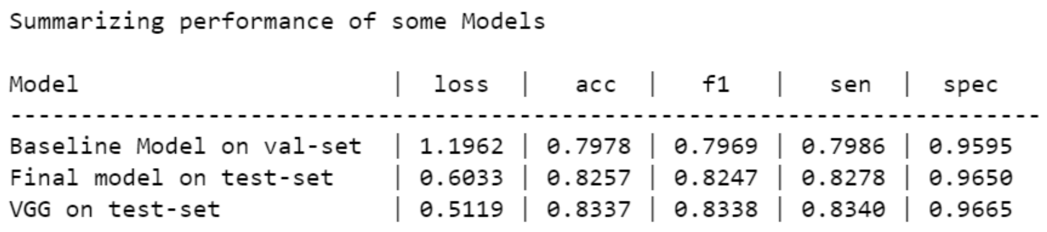
Stop-early became effective in impeding the model from overfitting. In Figure 6, no sign of overfitting was recorded though out seven epochs. Model prediction also reached its best performance on the test set at the last epoch (the 7th).

In terms of prediction in each class, the six classes were labeled from 0 to 5, corresponding to 'building', 'forest', 'glacier', 'mountain', 'sea', and 'street', respectively. The baseline model showed a severe problem in mislabeling glacier pictures as 'mountain' (108 pictures) and 'sea' (49 pictures) (figure 4). 'Glacier' was also the class with the highest wrongly labeled rate among the six, with the lowest AUC score (Figures 3 and 6). Even though the final model could mitigate the number of wrong ‘mountain’ labels on glacier pictures (Figure 4, 7), the reduction was not enough to improve the AUC score. In contrast, the increase of wrong ‘glacier’ labels on mountain pictures showed a warning sign in this final model. Another noticeable problem during baseline was labeling mountain pictures with ‘sea’, which the final model also failed to improve. On the other hand, the final model succeeded in fixing other identification troubles, which as misclassifying ‘building’ with ‘street’ and vice versa. Both models performed best in the ‘Forest’ class, with the most minor mislabeled numbers (Figure 3, 6) and the highest AUC scores. The AUC score of this label reached 100% in the final model. ‘Macro’ and ‘micro’ averages were equal in both ROC plots, which is understandable as there was not a big discrepancy between the class’s AUC scores and the number of pictures stored in each category.

The improvement proved the effectiveness of our optimizing solutions for the neural network. Many studies appraised the merit of Dropout regularizers in optimizing the neural network, especially complex ones (Phaisangittisagul, 2016; Srivastava et al., 2014). Dropout’ prevented the domination of some neurons, which forced the neurons to learn more different patterns, facilitating generalization. However, we suspected that the new-pattern learning caused the effect of trade-off recognition between the three classes:’ glacier’, ‘mountain’, and ‘sea’, which could be observed in the two confusion matrices (Figure 3, 6). Studies also supported the combination of L2-norm and Dropout to optimally reduce variance and constraint bias (Peng et al., 2015), such that Dropout should not be used as a single optimizing solution (Badola et al., 2020). As a result, both loss and accuracy were improved. Besides, we also used bigger data (training and validation set) during fitting the new model, bringing more resources for our new neural network to learn.

3.5. Transfer learning - VGG16 model:

To test data classification on the VGG16 model, we froze all the first layers until the fully connected layers. Following the frozen layers, two fully connected layers are added with the size of 64 and 32 units, respectively. Relu activation was employed in these two layers. We followed this structure because the fully connected layers in our baseline and final model had the same size as such, which would give us a fair assessment of whether the VGG16 might bring in better optimization.



```
Summarizing performance of some Models

Model                      |  loss  |  acc   |   f1   |  sen   |  spec
--------------------------------------------------------------------------
Baseline Model on val-set  | 1.1962 | 0.7978 | 0.7969 | 0.7986 | 0.9595
Final model on test-set    | 0.6033 | 0.8257 | 0.8247 | 0.8278 | 0.9650
VGG on test-set            | 0.5119 | 0.8337 | 0.8338 | 0.8340 | 0.9665
```

Figure 9: Summarizing prediction performance of Baseline model, the final model and the VGG16 model

Figure 9 presented a surpassing performance of the VGG16 model compared to the baseline and final models. VGG16 limited the loss at 0.51 and pushed accuracy to 83.4% in prediction on the test data. F1-score, sensitivity, and specificity also showed an increase of 1% compared to the final model. Figure 10 revealed that the VGG16 model started at the optimized level and preserved this performance until the last epoch. This phenomenon is logical. Because the model already uploaded its pre-trained rate from the beginning and froze it until the last Max-pooling operation, allowing update weights to be fine-tuned only during the new training. Other studies also recorded the same excellent result of this model in improving accuracy (Abu et al., 2022)(She et al., 2022). However, Mongkhonthanaphon and Limpiyakorn argued that VGG16 would perform slightly better if we unfreeze a few layers during the contracting path. Yet, this approach might result in computational expensive and longer running time (Mongkhonthanaphon & Limpiyakorn, 2020).

Regarding classification, the VGG16 performed slightly better, with higher AUC scores on most labels (Figure 11). The confusion matrix of VGG16 showed an improvement in differentiating pictures of ‘sea’, ‘mountain’, and ‘glacier’, although the number of mislabels among these three classes still caught attention. Contrastingly, the misclassifying mountain pictures with 'glacier' seemed more severe in VGG16's prediction (Figure 12), resulting in a significantly high number of wrong labels in this pair (82 pictures).

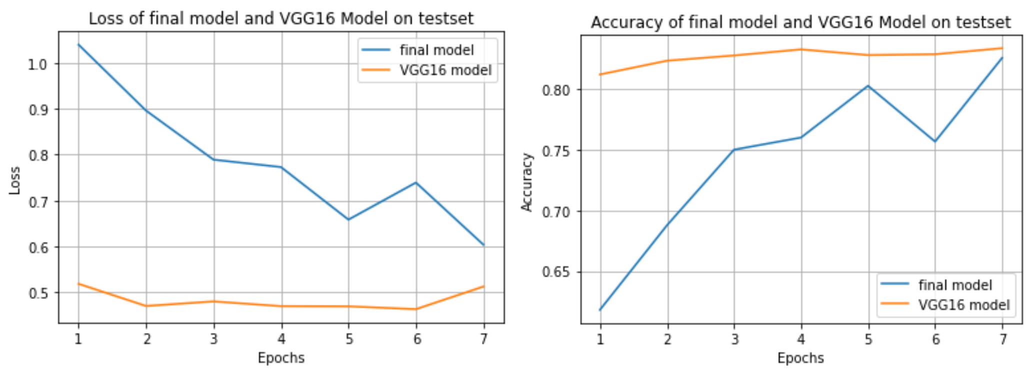


Figure 10: Prediction performance by epoch between the final model and VGG16 model

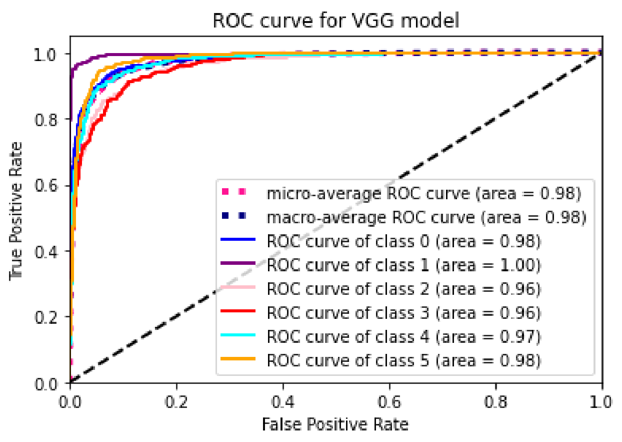


Figure 11: ROC curve of VGG16's performance on test set

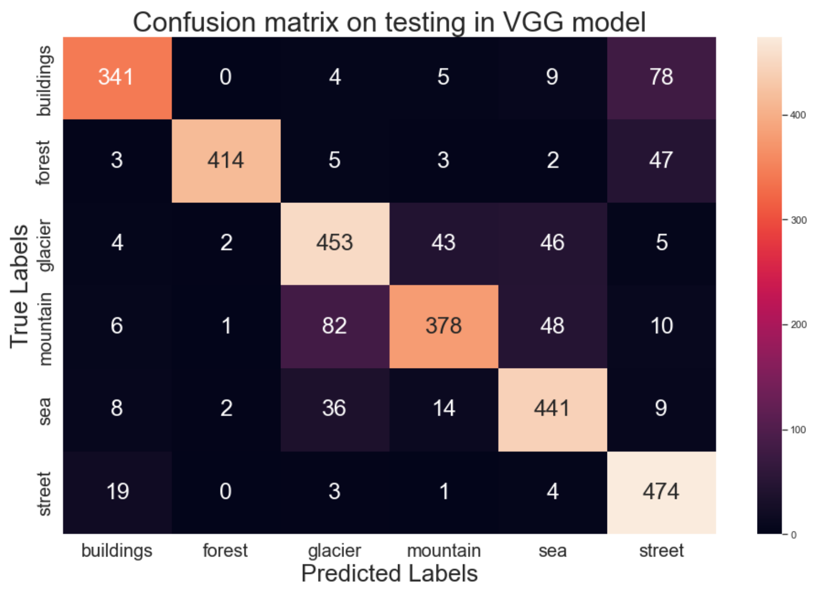
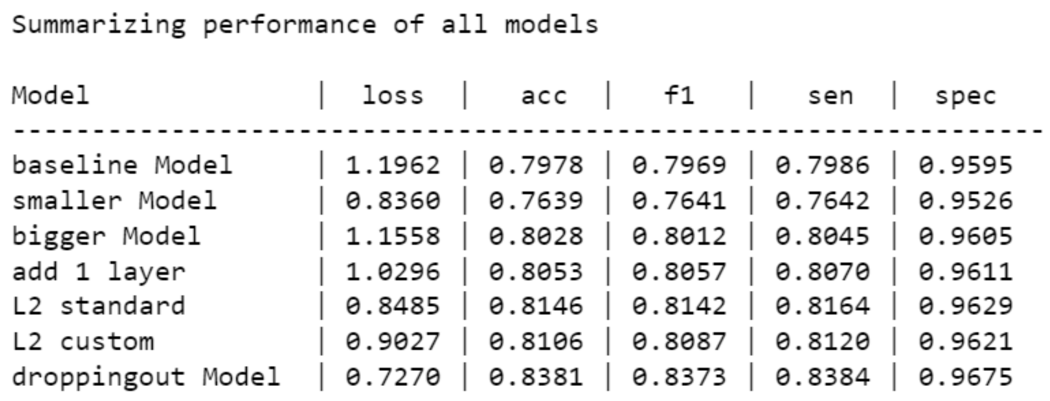


Figure 12: Confusion matrix of VGG16's performance on test set

Annex:

Annex 1: Summary of model's performance results during optimization experiments



```
Summarizing performance of all models

Model               |  loss  |  acc   |  f1    |  sen   |  spec
-------------------------------------------------------------------
baseline Model      | 1.1962 | 0.7978 | 0.7969 | 0.7986 | 0.9595
smaller Model       | 0.8360 | 0.7639 | 0.7641 | 0.7642 | 0.9526
bigger Model        | 1.1558 | 0.8028 | 0.8012 | 0.8045 | 0.9605
add 1 layer         | 1.0296 | 0.8053 | 0.8057 | 0.8070 | 0.9611
L2 standard         | 0.8485 | 0.8146 | 0.8142 | 0.8164 | 0.9629
L2 custom           | 0.9027 | 0.8106 | 0.8087 | 0.8120 | 0.9621
droppingout Model   | 0.7270 | 0.8381 | 0.8373 | 0.8384 | 0.9675
```

Figure 13: Performance measures of the baseline and tryout models on validation set

Annex 2: Comparison of validation loss between the smaller - bigger - baseline model

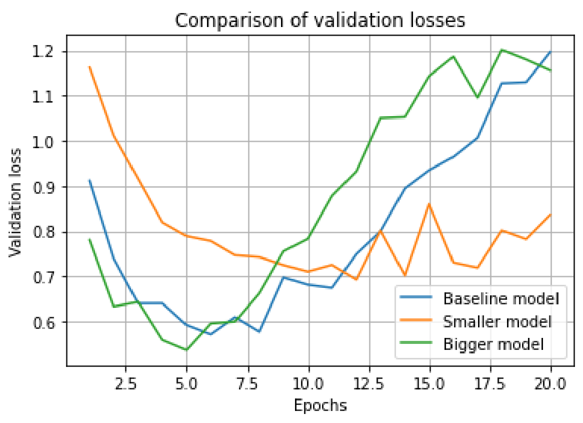


Figure 14: Validation loss by epochs of baseline model, smaller model, and bigger model

Annex 3: Comparison of validation loss between baseline model and model with an extended layer

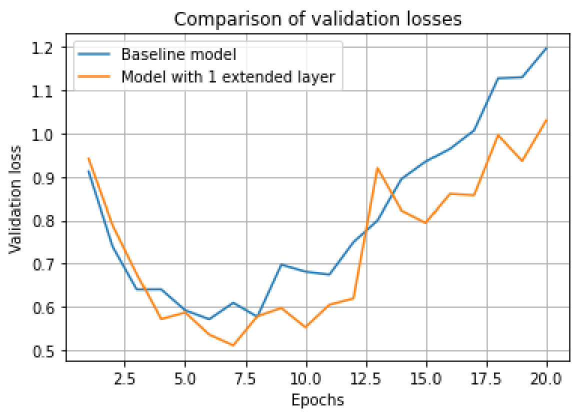


Figure 15: Validation loss by epochs of baseline model and model with an extended layer

Annex 4: Comparison of validation loss between baseline model and 2 models with L2 regularizers

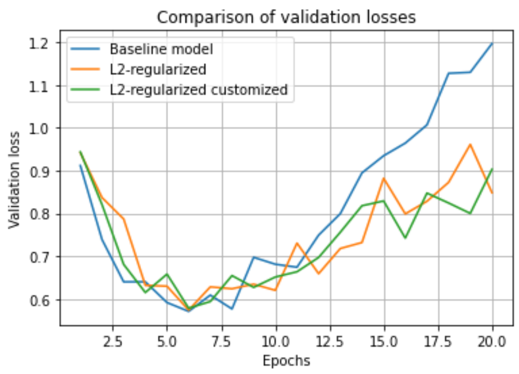


Figure 16: Validation loss by epochs of baseline, Keras L2-regularized (standard) and Manual L2-regularized models

Annex 5: Comparison of validation loss between baseline model and model with drop-out regularizer

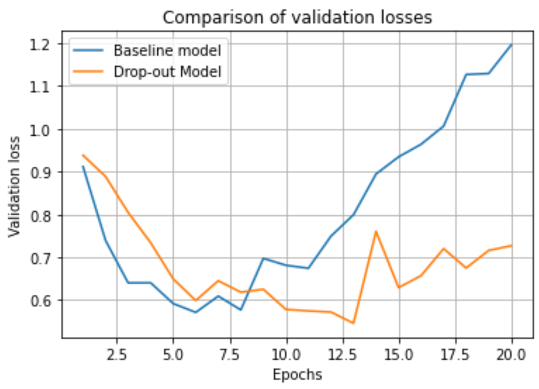
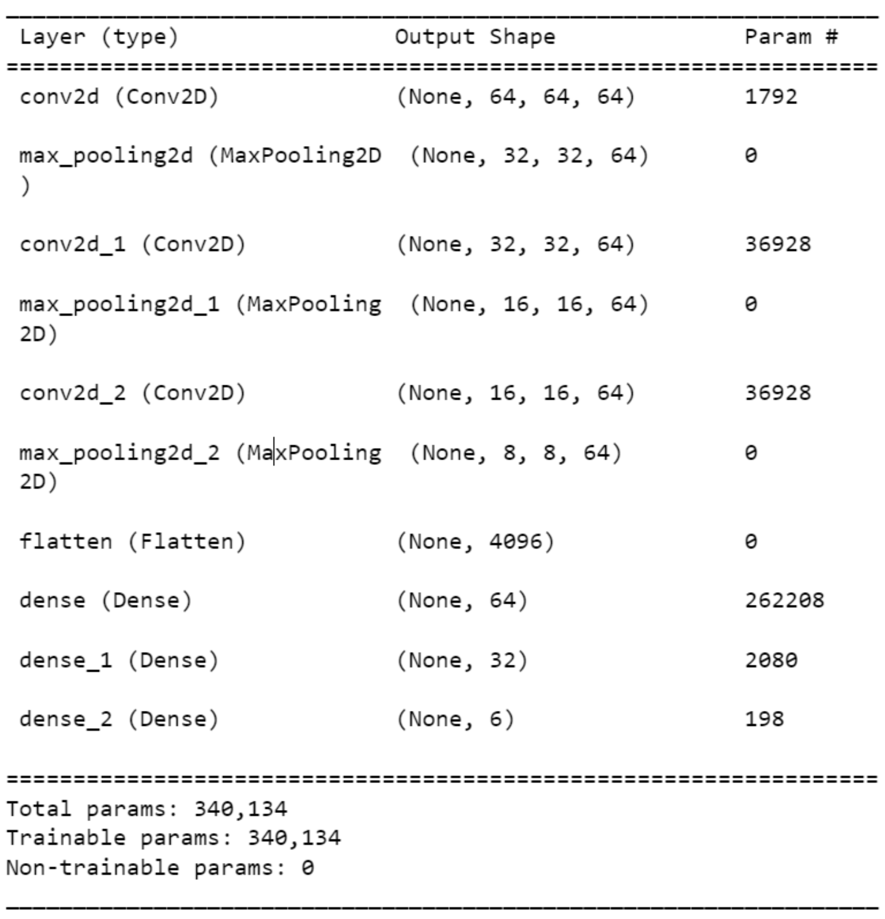


Figure 17: Validation loss by epochs of the baseline and Dropout regularized model (Drop-out rat = 0.4)

Annex 6: Summary of baseline model



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 64, 64, 64)        1792

 max_pooling2d (MaxPooling2D  (None, 32, 32, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 32, 32, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_2 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 flatten (Flatten)           (None, 4096)              0

 dense (Dense)               (None, 64)                262208

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 6)                 198

=================================================================
Total params: 340,134
Trainable params: 340,134
Non-trainable params: 0
_________________________________________________________________
```

Figure 18: Summary of Baseline model

References:

Abu, M., Zahri, N. A. H., Amir, A., Ismail, M. I., Yaakub, A., Anwar, S. A., & Ahmad, M. I. (2022). A Comprehensive Performance Analysis of Transfer Learning Optimization in Visual Field Defect Classification. *Diagnostics 2022, Vol. 12, Page 1258*, *12*(5), 1258. https://doi.org/10.3390/DIAGNOSTICS12051258

Badola, A., Nair, V. P., & Lal, R. P. (2020). An Analysis of Regularization Methods in Deep Neural Networks. *2020 IEEE 17th India Council International Conference, INDICON 2020*. https://doi.org/10.1109/INDICON49873.2020.9342192

Mongkhonthanaphon, S., & Limpiyakorn, Y. (2020). A Deep Neural Network for Pixel-Wise Classification ofTitanium Microstructure. *International Journal of Machine Learning and Computing*. https://web.archive.org/web/20200507131456id_/http://www.ijmlc.org/vol10/909-CA1-135.pdf

Peng, H., Mou, L., Li, G., Chen, Y., Lu, Y., & Jin, Z. (2015). A Comparative Study on Regularization Strategies for Embedding-based Neural Networks. *Conference Proceedings - EMNLP 2015: Conference on Empirical Methods in Natural Language Processing*, 2106–2111. https://doi.org/10.48550/arxiv.1508.03721

Phaisangittisagul, E. (2016). An Analysis of the Regularization Between L2 and Dropout in Single Hidden Layer Neural Network. *Proceedings - International Conference on Intelligent Systems, Modelling and Simulation, ISMS*, *0*, 174–179. https://doi.org/10.1109/ISMS.2016.14

She, L., Fan, Y., Xu, M., Wang, J., Xue, J., & Ou, J. (2022). Insulator Breakage Detection Utilizing a Convolutional Neural Network Ensemble Implemented With Small Sample Data Augmentation and Transfer Learning. *IEEE Transactions on Power Delivery*, *37*(4), 2787–2796. https://doi.org/10.1109/TPWRD.2021.3116600

Srivastava, N., Hinton, G., Krizhevsky, A., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, *15*, 1929–1958.