Classifying Pokemon's at NOVA
Deep Learning Assignment #1

Bernardo Calvo
MIEI — 59187
NOVA School of Science and Technology
Caparica
b.calvo@campus.fct.unl.pt
Group 2

Bruno Carmo
MIEI — 57418
NOVA School of Science and Technology
Caparica
bm.carmo@campus.fct.unl.pt
Group 2

Sahil Kumar
MIEI — 57449
NOVA School of Science and Technology
Caparica
ss.kumar@campus.fct.unl.pt
Group 2

Abstract

In this project, we investigate popular deep learning models to complete a task of image classification: Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), and pre-trained models like ConvNeXtXLarge and VGG16. We experiment with these models on a dataset of images of Pokemon's near the NOVA SST campus. In our experiments, after testing various kinds of models with different layers, we show that the created CNN outperforms both MLP and pre-trained models, achieving an accuracy of 99.2% and a loss of 3.56% for the multi-class classification task. We also designed and trained a neural network to predict multible classes from an image (multi-label classification) and found that the most performant architeture is also CNN with an accuracy of 84%. Unfortenately, we could not obtained the same performance for the pre-trained models. Our findings suggest that CNN is highly effective for image classification tasks and can learn complex features that are critical for accurate classification.

1 Introduction

Image classification consists in a supervised learning problem of assigning a class to an image. It is a subdomain of the computer vision field, as such, it has many real world applications.

This project aims to create and study different deep learning networks to perform image classification in order to understand which are the most suitable for the task. The problem in question is related to the classification of different types of Pokemon, where each Pokemon can have one type (single-label classification) or two types (multi-label classification) out of 10 different types. The types of the Pokemon's are: Bug, Fighting, Fire, Flying, Grass, Ground, Normal, Poison, Rock and Water.

The dataset consists in 4500 images of the NOVA SST campus with a Pokemon present in each sample, where 4000 images belong to the training set and 500 to the test set. Each image is composed of 64 by 64 pixels, with each pixel having 3 color channels (RGB) and normalized values between 0 and 1.

Preprint. Under review.

Therefore, in order to achieve the project objective, we solve four tasks. The first task and second tasks are related to assigning a type to the Pokemon present in the image, using a Multi Layer Perceptron (MLP) architecture and a Convulotional Neural Network (CNN), respectively. The third task is intended to carry out multi-label classification by identifying the two types associated with the Pokemon. In the last task, the objective is to use a pre-trained model on ImageNet to perform the two classification tasks described above.

2 Task 1

The first task was to create a MLP architecture to learn only one class related to the Pokemon in each image.

2.1 Network architecture

The architecture of the model starts with the Flatten layer which was added to vectorize the image data that is received as input, in this case, the images with shape (64, 64, 3) are transformed to a vector of size 64x64x3 = 12 288. We verified that with 0 or 1 hidden layers the model was underfitting and with 3 hidden layers it started to overfit, so we decided to keep 2 hidden layers with ReLu activation functions, which is normally the most indicated. Regarding the number of neurons, values were maintained between the number of neurons on the input layer and the number of neurons on the output layer, in powers of 2 and with the number of neurons decreasing along the hidden layers, trying to keep these numbers as low as possible. Because there are 10 classes, the output layer has 10 neurons and includes the Softmax activation function, in order to give probabilities for all the neurons summing up to 1, since we only intend to assign only one type to the Pokemon. After, various tests and experimentation the best architecture we found is represented in figure 1.

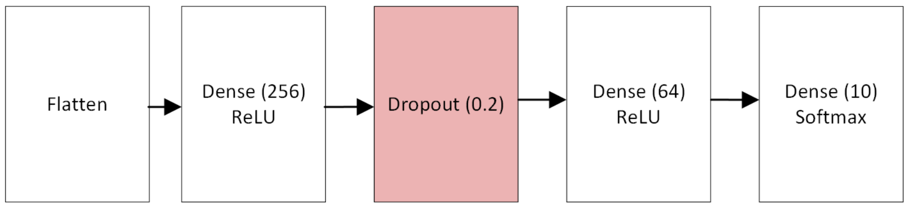


Figure 1: MLP network used

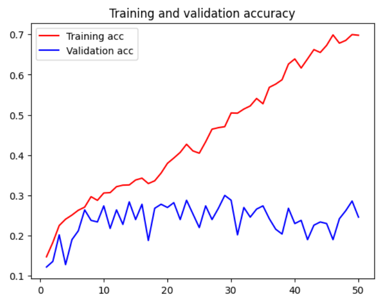
2.2 Loss functions

For the loss functions, we used a regression loss, in this case, the Mean Square Error loss and verified that the model could not learn, as such, we decided to apply a probabilistic loss, more specifically, the Categorical Cross-Entropy loss, since the labels are one-hot encoded and it showed better results.

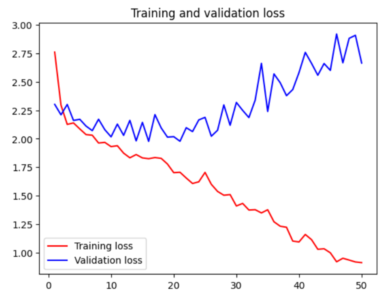
2.3 Training and overfitting

For training, 12.5% of the training set was used for validation, that corresponds to 500 images, as recommended. We trained the model for 50 epochs, varied the learning rate value in bases of 10 and used SGD and Adam for the optimizer. The configuration that demonstrated the best accuracy was with the Adam optimizer using a learning rate of 0.001.

As it can be seen in figure 2 , the model continues to overfit since the accuracy value for the training set increases, in contrast with the accuracy for validation set that remains the same. A similar observation can be made for the loss values that decreases for the training set but increases for the validation set. Therefore, we decided to add a Dropout layer with 0.2 rate after the first hidden layer in order to do regularization and consequently generalize better. Another way to regularize could be using early stopping. With this change, we obtained our best model as seen in figure 3.

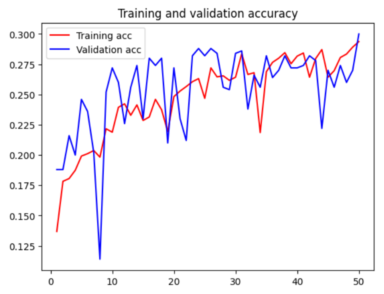


(a) Training and validation accuracy

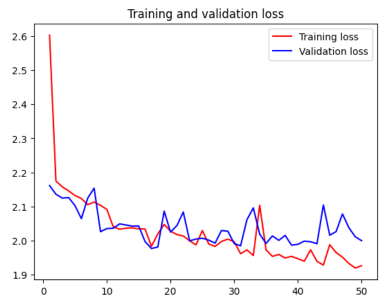


(b) Training and validation loss

Figure 2: MLP overfit model



(a) Training and validation accuracy



(b) Training and validation loss

Figure 3: MLP best model

2.4 Results and testing

The resulting model has an accuracy of approximately 28% on test data, which is not a good value for a further application of the model in an computer vision environment.

In conclusion, for image classification, the MLP model is not the most appropriate since it does not show great accuracy, easily overfits the training data and does not generalize correctly.

3 Task 2

This second task is similar to the previous one but this time using a CNN architecture, which is usually the most suitable for image classification.

3.1 Network architecture

Taking into account that the most common architectures for CNNs are Conv $\rightarrow$ Pool $\rightarrow$ Conv $\rightarrow$ Pool or Conv $\rightarrow$ Conv $\rightarrow$ Pool $\rightarrow$ Conv $\rightarrow$ Conv $\rightarrow$ Pool, we tested both in order to obtain the best model. Thus, we used a 3×3 kernel (to try to add more depth in order to minimize the number of parameters), initialized using the HeNormal initializer, with a ReLu activation function, using padding. The number of filters was also varied, to keep it to a minimum. For pooling, we used MaxPooling with a 3×3 kernel as usual. For the classification head, we started with a Flatten layer to transform the data into a single vector followed by one hidden layer with ReLu activation and finalised by the output layer with the number of neurons equal to the number of classes, using a Softmax activation, for reasons already explained above. After many experiments, we verified that for this case, it was only necessary one hidden layer. The best model that was found varying all these parameters is represented in figure 4.

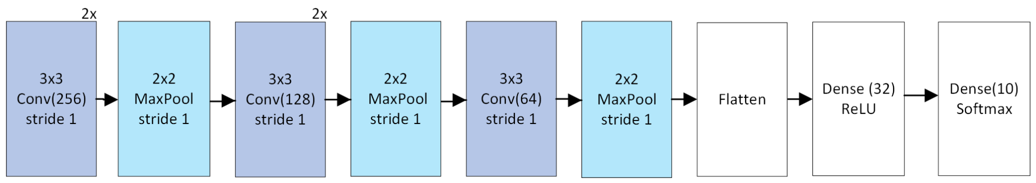


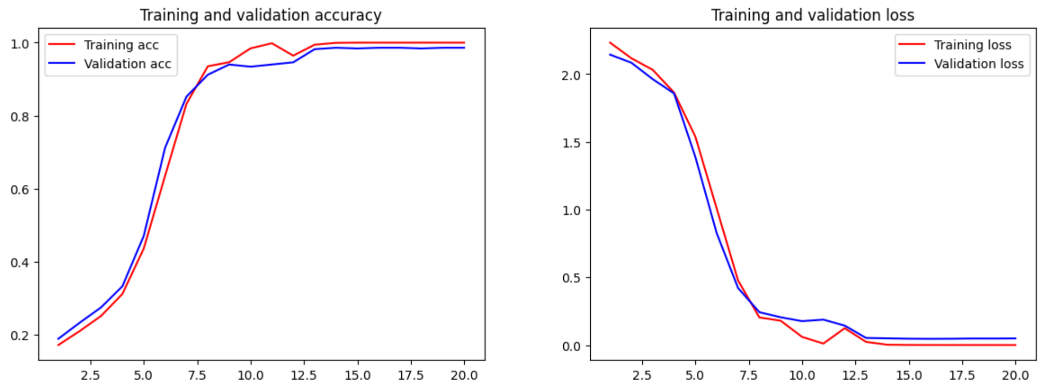
Figure 4: CNN network used

3.2 Loss functions

Again, Categorical Cross-Entropy loss was used since it is only intended to classify one type and the labels follow the one-hot encoding format.

3.3 Training and overfitting

Following the same procedure used in Task 1, the best results were obtained training the CNN for 20 epochs using the Adam optimizer with a learning rate of 0.001 and with no need to use regularization. The model started to overfit when the number of convolutional layers was high, when the number of filters was inadequate or when there were too many hidden layers with more neurons than was necessary, as such, the model did not generalize to data that it had never seen. In the figure 5 it is possible to verify the plots resulting from the training of the best model found.



(a) Training and validation accuracy (b) Training and validation loss

Figure 5: CNN best model

3.4 Results and testing

We ended up verifying an accuracy of 99.2% and a loss of 3.56% on the test set with this model, which is very good for further applications. Therefore we conclude for image classification, that compared to the MLP architecture, the CNN architecture is undoubtedly the architecture to adopt.

4 Task 3

The third task was to design and train a neural network that can predict the two types associated with a Pokemon from an image by doing multi-label classification.

4.1 Network architecture

After testing various models, we reached the conclusion that CNN-based architectures are the most appropriate for multi-label image classification. In order to reach the best performance in the feature extraction layers we used three sequences of two convolutional and one pooling layer followed by two convolutional layers. The convolutional layers use a 3×3 kernel (to be possible to add more depth and at the same time, minimize the number of parameters), with a ReLu activation function and a HeNormal initializer. The number of output filters in the convolutional layers starts at 32 and goes up to 256 in the last two layers. For pooling, we used MaxPooling with a 2×2 kernel size after the first layer and a 3×3 kernel in the remainder pooling layers.

For the classification head we first flatten the inputs to vectorize the image followed by a hidden layer with 512 units and a ReLu activation and finally the output layer. As mentioned before the output layer needs to have the number of neurons to be equal to the number of classes (10 classes for 10 types of Pokemon) but unlike the previous tasks where we use the Softmax activation, we do not require all probabilities to sum to one. Hence, the output layer uses the Sigmoid activation so that the model outputs a probability of class membership for the label, a value between 0 and 1 for each neuron, that when summed up to all 10 neurons does not add to 1.

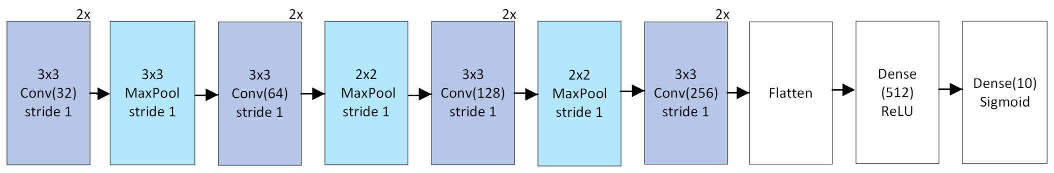


Figure 6: CNN network used

4.2 Loss functions

Unlike the first tasks where we use Categorical Cross-Entropy as a loss function, in multi-label classification there are no exclusive classes so we need to use Binary Cross-Entropy which is independent for each vector component (class). As such, the loss computed for each of the CNN output vector components is not affected by other component values, making the loss appropriate for multi-label classification, where the knowledge of an element belonging to a particular class should not affect the decision for another class.

4.3 Training and overfitting

Using the procedure described in Task 1, the best results were obtained by training the model for 20 epochs using the Adam optimizer with a learning rate of 0.001 without the need to use regularization. The model would not learn effectively when using a lower learning rate and would start to overfit when adding more convolutional layers or when the number of neurons in the hidden layers were inadequate. In the figure 7 it is possible to verify the plots resulting from the training of the best model found.

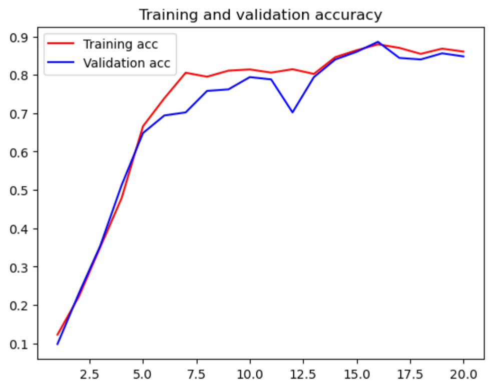
4.4 Results and testing

The resulting model has an accuracy of approximately 84% on test data, and therefore making it reasonably appropriate for further applications in multi-label image classification tasks.

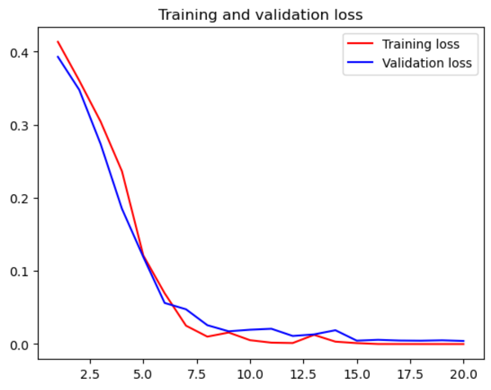
In conclusion, for image classification, our CNN model has a good performance on the multi-label classification task, although we believe there is a window for improvement.

5 Task 4

The fourth task was to use pretrained neural networks on the ImageNet dataset, available in the *keras.application* module, as feature extractors for the single-label and multi-label classification tasks, discussing the effect of using the pretrained networks against training our own convolutional layers in our network.



(a) Training and validation accuracy



(b) Training and validation loss

Figure 7: Multi-Label best model

5.1 Network architecture

From the available pretrained neural networks in the *keras.application* module we chose the following networks to use as feauture extractors:

- **ConvNeXtXLarge:** devoloped by researchers from the Facebook AI Research (FAIR) center. We chose this model because it had the biggest top-1 accuracy on the ImageNet dataset out of all the other models in the module.
- **VGG16:** devoloped by researcher from the University of Oxford. We chose this model because it had the lowest depth of all the other models.

For the ConvNeXtXLarge feature extractor, we used the classification head represented in figure 8, using the Softmax activation function for single-label classification and the Sigmoid activation function for multi-label activation. The implementation of the ConvNeXtXLarge in *keras.applications* module provided an *include_preprocessing* flag which we needed to set to false, so that the feature extractor does not normalize the pixels of our images (because they are already normalized) according to the ImageNet dataset.

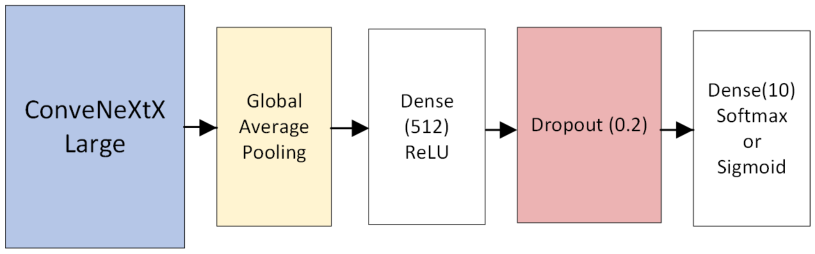


Figure 8: Model using ConvNeXtXLarge as feature extractor

For the VGG16 feature extractor, we used the classification head represented in figure 9, using the Softmax activation function for single-label classification and the Sigmoid activation function for multi-label activation. The VGG16 accepts only images of specific shapes, as such, we used a resizing layer to change the shape of our images (64×64) to the default shape the VGG16 feature extractor accepts (224×224).

Due to lack of time, the models used for multi-label followed the same architecture as those used for single-label, although the best thing would be to adjust the model to obtain the best performance for each specific task.

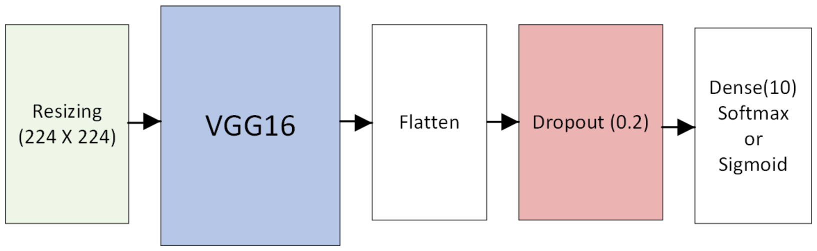


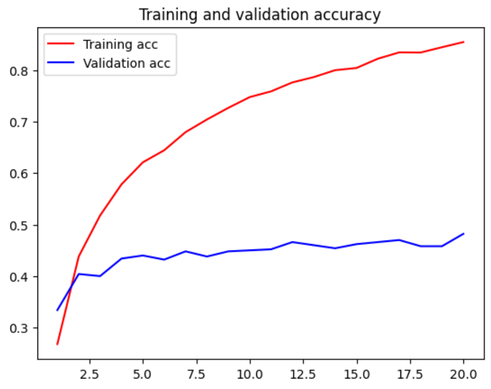
Figure 9: Model using VGG16 as feature extractor

5.2 Loss functions

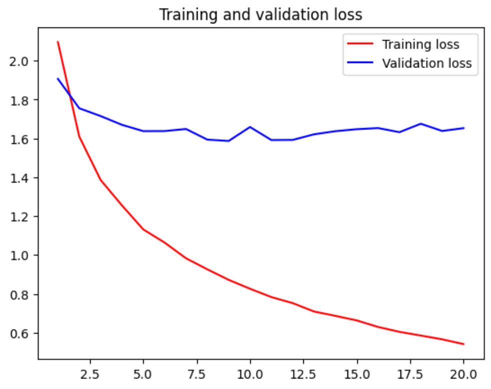
As mentioned above, we used the Categorical Cross-Entropy loss for the single-label classification and the Binary Cross-Entropy loss for the multi-label classification.

5.3 Training and overfitting

In relation to the classification head of the ConvNeXtXLarge feature extractor, we used the same process to train described in task 1, but differing the number of epochs. Unfortunately, we could not get a model that had a good accuracy in validation. For various classification head architectures, the training accuracy never got above 80% and always overfitted the data. In figure 10 and 11 we present the plots of the best model we obtained for single-label classification and multi-label classification, respectively.



(a) Training and validation accuracy



(b) Training and validation loss

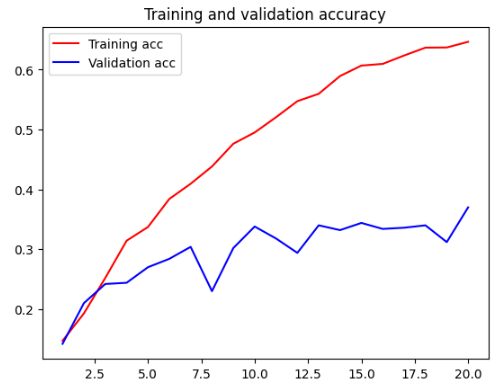
Figure 10: ConvNeXtXLarge best model for single-label classification

Regarding the classification head of the VGG16 feature extractor, we obtained better results than the ones described above. Using the training procedure mentioned in task 1, the best model we got was a model completely trained in relation to the train set, with an accuracy of 100% on the training set and with a validation accuracy above 80%. In figure 12 and 13 we present the plots of the best model we obtained for single-label classification and multi-label classification, respectively.

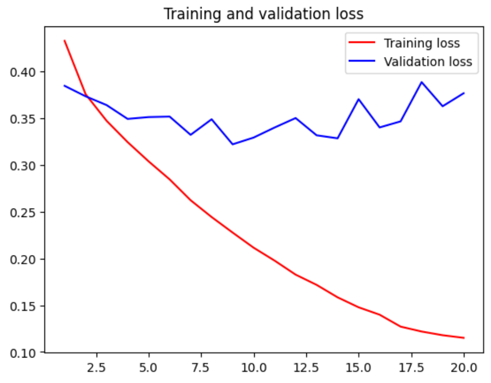
5.4 Results and testing

Regarding the model with the ConvNeXtXLarge feature extractor, because of overfitting, the accuracy on test data for single-label classification is 48% and for multi-label classification is 40%.

In relation to the model with the VGG16 feature extractor, the accuracy on test data for single-label classification is 82% and for multi-label classification is 60%. Comparing with the previous one, this model got a better performance and it would be the most suitable between the two.

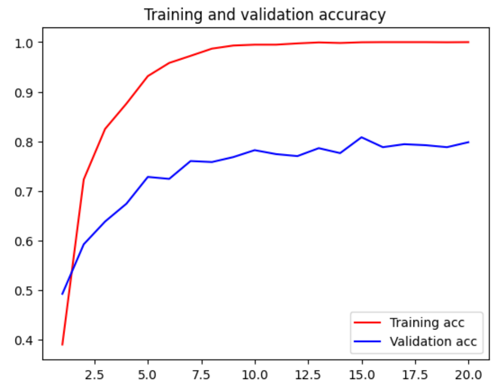


(a) Training and validation accuracy

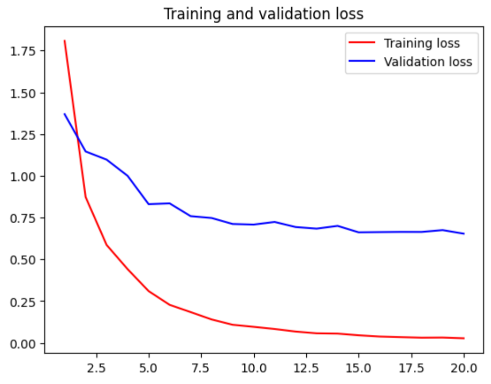


(b) Training and validation loss

Figure 11: ConvNeXtXLarge best model for multi-label classification

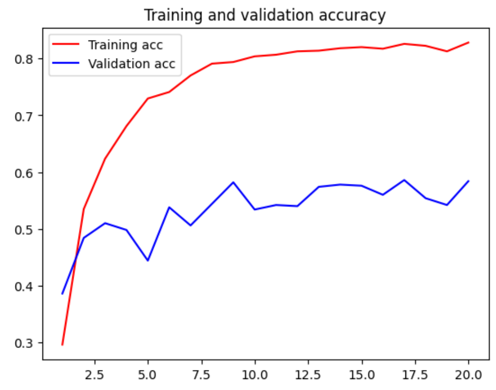


(a) Training and validation accuracy

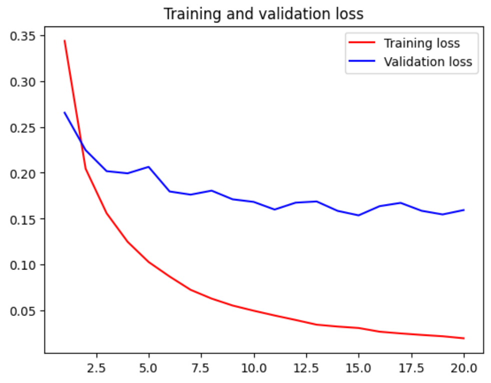


(b) Training and validation loss

Figure 12: VGG16 best model for single-label classification



(a) Training and validation accuracy



(b) Training and validation loss

Figure 13: VGG16 best model for multi-label classification

With this we conclude, that our CNN trained in our dataset performs considerately better on our dataset than models obtained from transfer learning trained on ImageNet, but simpler models have a test accuracy close to ours.

Taking into account that in a dataset with a large amount of data, which is not this case, and that it would take time to train on a new CNN model, the most appropriate thing would be to test several pre-trained models until it is found one with good performance, and if none is found, then proceed to a fine-tuning.