SMML 2022-2023 Neural Networks Project

Matteo Castagna (27366A)
matteo.castagna2@studenti.unimi.it

Abstract

Image classification is a computer vision technique which consists of assigning a label or a tag to an entire image, usually based on preexisting already labeled, data. Among all the computer vision tasks, image classification is the most popular and successful one, in particular thanks to convolutional neural networks (CNNs), a kind of NN which mimics the way the human eye views things. In this work different CNNs are compared in the image classification task of distinguish chihuahuas from muffins, and vice versa, using 5-Fold Cross Validation. The CNNs are built using Tensorflow and Keras APIs. The best model of each architecture used is obtained through hyperparameter tuning conduct with the KerasTuner API.

1 Dataset

The dataset used is the "Muffin vs chihuahua"[1] dataset available on Kaggle. The raw dataset is composed by 5917 jpg RGB images, 3200 chihuahua images and 2719 muffin images, it comes with a train and test folders, each one containing separate folders for the chihuahua and muffin images. In order to preprocess the data correctly, all the images are merged into two folders, one for chihuahuas and one for muffins. Those two are the folders used in the project.

1.1 Preprocessing

Despite the description of the dataset stating that all the duplicates have been removed, this may be valid just for duplicate images of the same size, in fact, the dataset has various images of different sizes but with the same exact subject. Due to the different scales it is not possible to discover duplicates by means of a pixel wise comparison, and so the search for duplicates is conduct using image hashing. Differently from cryptography hashing, where hash values are random, image hashing is the results of image processing, therefore by the hashes can be tell if two images are the same or if they're just similar. The main image hashing techniques are: a-hash (average), p-hash (perceptual) and d-hash (difference, it use the image gradient). After comparing the results of both the 3 image hashing techniques, a-hash is the one chosen to conduct the duplicate search as it is the one able to individualize the largest amount of duplicates. The preprocesing computed by a-hash on the image is very simple:

- Reduce size: the fastest way to remove high frequencies and detail is to shrink the image. In this case, shrink it to 8x8 so that there are 64 total pixels. The hash will match any variation of the image, regardless of scale or aspect ratio.
- Reduce color: the 8x8 picture is converted to a grayscale. This changes the hash from 64 RGB pixels (64 red, 64 green, and 64 blue) to 64 total values.

[1] https://www.kaggle.com/datasets/samuelcortinhas/muffin-vs-chihuahua-image-classification?select=test

- Average the values: compute the average of the 64 values.
- Compute the bits: each bit is set based on whether the value is above or below the mean one computed at the previous step.
- Construct the hash: set the 64 bits to a 64-bit integer. At this point two images with the same hash are the same image.

a-hash has been performed with the ImageHash library[2] The hashes of all the images for both chihuahuas and muffin are automatically compared and duplicates are removed.

An additional manual search for noisy samples is performed. As the images making the dataset have been automatically downloaded from Google, the only way to ensure an actual correspondence between images and labels is through a manual search. Between the chihuahua examples a lot of images represent all kinds of dogs, not only chihuahua, those are removed. Images which represent stylized or drawn chihuahua, images with cropped part of chihuahuas and pictures with both human and chihuahuas, are not removed in order to keep variability in the dataset and keep the number of examples between between the classes balanced. In the muffin examples few images represent just the raw ingredients instead of the actual muffin, those are removed. In the end 2580 chihuahua images and 2604 muffins images remain.

Finally, the images are automatically rescaled to 120×120 RGB pictures when are load from the folders with `image_dataset_from_directory` Tensorflow Keras function.

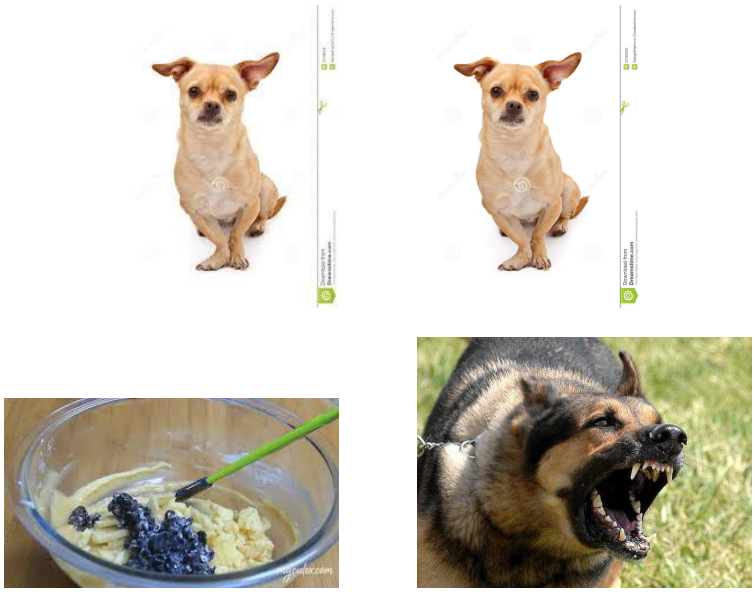


Figure 1: Top raw: example of duplicate images found with a-hash, img_0_951.jpg (left) and img_0_788.jpg (right). Bottom raw: example of manually removed images from the muffin samples (left) and chihuahua samples (right).

[2] `https://pypi.org/project/ImageHash/`

1.2 Notes

The possibility to perform data augmentation with the respective Keras methods, which perform random rotations, crops, and so on directly into the architecture off the neural network, was taken into account. In order to keep the computational time of each script manageable (around 20 minutes for ResNet50, VGG16 and the custom CNN, and around 1 hour for the vision transformer), since 5-Fold CV is used and so 5 models for each architecture are trained, image augmentation was not used. The actual dataset still presents a big variability in its examples, especially for the chihuahua images. For the same reason the number of epochs on which the models run, for both hyperparameters tuning and model training, are limited respectively to five and ten.

2 Models

In this section a summary on CNN functioning is presented and then the networks used are described.

2.1 CNN basics

Convolutional Neural Networks, is a class of NNs that specializes in processing data that has a grid-like topology, for this reason it found a huge usage in processing images. Consider a digital image, it is just a binary representation of visual data, a series of pixels arranged in a grid-like fashion that contains values to denote the color and brightness the pixel should be. CNNs are based on convolution and are generally built on blocks of three operations: convolution, non linearity (ReLU) and pooling.

The convolution operation on images consists of shifting a squared matrix usually of odd dimensions, called kernel, along the entire given image. It computes a new image in which each pixel is a function of the nearby pixels (including itself) in the input image, this function is determined by the coefficients of the kernel. The general expression for 2D convolution is:

$$g(x,y) = w * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} w(dx,dy) f(x-dx, y-dy) \tag{1}$$

where $g(x,y)$ is the output image, $f(x,y)$ is the input image, w is the kernel and every element in the kernel is considered $-a \leq dx \leq +a$ and $-b \leq dy \leq +b$. Depending on the coefficients of the kernel, it acts like as a filter detecting certain features (patterns), independently from where those are located in the image; therefore CNNs are also called shift invariant or space invariant artificial neural networks (SIANN). The output of the convolution is called feature map. The same convolutional layer can apply more then one filter, obtaining a "deeper" feature map. It's important to note that convolution reduce the size of the image, as when shifting the kernel along the input, when it is centered on border pixels it fall out of the image. When the output image is obtained, it can be padded by adding zeroes at its borders to restore its original size. Both VGG16 and ResNet50 use zero padding and it is adopted also by the custom CNN.

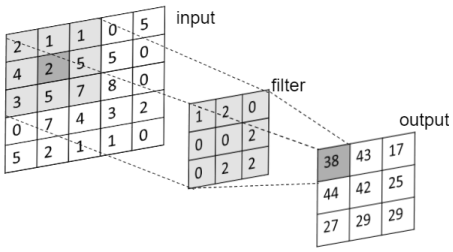


Figure 2: Visual representation of 2D convolution.

The feature map is passed through a pixel wise non linear function, as in classical feedforward neural network. In CNNs the non linear function used is the ReLU (Rectified Linear Unit), which for each pixel return its own value if its greater than zero, or zero otherwise.

After the non linearity a pooling operation is performed. Pooling aggregates the features obtained by the convolution by downsampling the feature map. A window of fixed size is shifted along the given feature map, the pixel in the output matrix will be either the maximum or the average of the value covered by the moving window. Differently from convolution, in pooling the window is shifted such that each portion considered is non overlapping with any other position of the window itself.

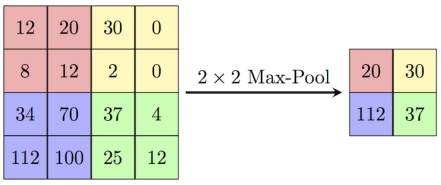


Figure 3: Visual representation of max pooling.

Several blocks of convolution, ReLU and pooling can be stacked one after the other building a deep convolutional neural network, extracting sequentially finer features. Once the last feature map is obtained, it is passed to a flatten layer which turns the matrix into a vector. Finally this vector is passed to a multilayer perceptron that ends with a softmax or a sigmoid activation function and outputs the probabilities of the image for each one of the classes of the classification task. In the case of binary classification the network can output just 0 or 1 according to the labels assigned to the two classes.

With CNNs becoming deeper and deeper, training them became more difficult, since it became slower and NNs became prone to overfitting. Thus, studies on methods to solve these problems are constant in Deep Learning research. Batch Normalization [1], is one of these methods. Batch Norm is a widely used technique that improves the learning speed of Neural Networks and provides regularization, avoiding overfitting. Batch Norm is a normalization technique done along mini-batches and between the layers of a Neural Network instead of on the raw data, it serves to speed up training by making possible to use higher learning rates. In the case of CNNs it is applied after the convolution and before the non linearity. While the effect of batch normalization is evident, the reasons behind its effectiveness remain under discussion. It was believed that it can mitigate the problem of internal covariate shift, that is, parameters initialization and changes in the distribution of the inputs of each layer affect the learning rate of the network. Recent studies have argued that batch normalization does not reduce internal covariate shift, but rather smooths the objective function, which in turn improves the performance.

Another technique commonly used in NNs and CNNs to avoid overfitting is the usage of a dropout layer in the classification head [2]. The term "dropout" refers to dropping out some nodes (in the input or hidden layer) in a neural network, randomly during training. All the forward and backwards connections with a dropped node are temporarily removed, thus creating a new network architecture out of the parent network. The nodes are dropped by a defined probability. Dropout makes the training process noisy, requiring nodes within a layer to take on more or less responsible for the inputs on a probabilistic basis.

2.2 VGG16

VGG16 [3] is a CNN architecture that's considered to be one of the best vision model architectures to date. VGG16 uses convolution layers with 3x3 filters of stride 1 with padding and maxpooling with a 2x2 window of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has two fully connected layers, followed by a softmax for output. The VGG16 has 16 layers that have weights (the convolution, dropout, fully connected and softmax layers) and it has about 138 million parameters.

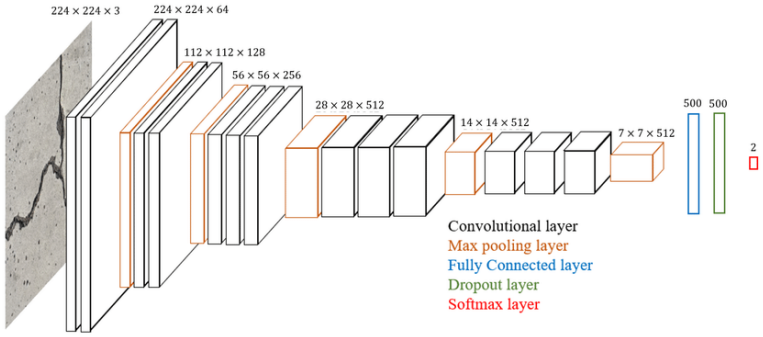


Figure 4: VGG16 architecture.

The built-in Keras VGG16 is used in the project. Its classification head is substituted with a custom one composed of: a flatten layer; 16 neurons dense layer; dropout layer with 0.2 dropout rate; 1 neuron dense layer followed by a sigmoid activation function. The output of the sigmoid corresponds to the network output and it can be round to the closest integer, either 0 or 1, to obtain the class predicted by the network for the input image. This same classification head is used with the ResNet50, the custom CNN and the vision transformer (ViT) too.

As mentioned, the model is pretrained. Keras built-in models come with possibility of loading them with the weights learned during the training on the ImageNet[3] dataset. ImageNet is an image dataset organized according to the WordNet hierarchy. Each meaningful concept in WordNet (an English lexical database of semantic relations between words), possibly described by multiple words or word phrases, is called a "synonym set" or "synset". There are more than 100,000 synsets in WordNet; the majority of them are nouns (80,000+). ImageNet aims to provide on average 1000 images to illustrate each synset. Images of each concept are quality-controlled and human-annotated. Both VGG16 and ResNet50 used adopt the weights learned on the ImageNet dataset.

[3] `https://www.image-net.org/about.php`

2.3 ResNet50

ResNet50 [4] provides a novel way to add more convolutional layers to a CNN, without running into the vanishing gradient problem by using skip connections or residual connections. Skip connections can help to preserve information and gradients that might otherwise be lost or diluted by passing through multiple layers and can also help to combine features from different levels of abstraction and resolution, which can enhance the representation power of the network. ResNet50 uses also a bottleneck design. A bottleneck residual block uses 1×1 convolution, known as "bottleneck", which reduces the number of parameters and matrix multiplications. This enables much faster training of each layer.

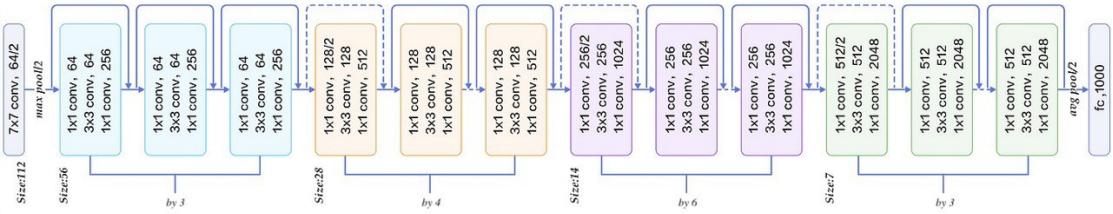


Figure 5: ResNet50 architecture.

2.4 Custom CNN

The custom CNN is built using KerasTuner API. The hyperparameters tuned are: the number of layers, between 1, 3 and 5 (a layer is comprising of convolution, batch normalization, ReLU activation and pooling); the number of filters per layer, between 16, 32 and 64; the kernel size, between 3 and 5, and the pooling technique to use, either maxpooling or average pooling. For both ResNet50, VGG16 and the custom CNN, learning rate and batch size are subject to tuning: learning rate is chosen between 0.1, 0.01, 0.001 and batch size is chosen between 16, 32 and 48. The model resulting from the hyperparameters tuning phase is the following:

- First layer: 16 filters of kernel size 5
- Second layer: 16 filters of size 3
- Third layer: 16 filters of size 3
- All the layers use Maxpooling

2.5 Vision Transformer

Vision Transformer (ViT) is an architecture that uses self-attention head [5] mechanism to process images. The Vision Transformer Architecture consists of a series of transformer blocks, each one consists of two sub-layers: a multi-head self-attention layer and a feed-forward layer.

In many deep learning models, data is processed by passing it through multiple layers of neural networks and as the data passes through these layers, it can become increasingly difficult for the model to identify the most relevant information. Attention mechanisms were introduced as a way to address this limitation in these models. Attention-based models can selectively focus on certain parts of the input when making a prediction. The general attention mechanism makes use of three main components, namely the queries, Q, the keys, K, and the values, V. An attention function can be described as mapping a query and a set of key-value pairs to an output.

The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key. Given an input sequence X to the attention mechanism, such that each element x in X is embedded into a $1 \times d$ vector, the attention for x is:

$$Attention(x, X) = softmax\left(\frac{xQ \cdot XK^T}{\sqrt{d}}\right) \cdot XV \tag{2}$$

with the query, keys and values matrices being $d \times d$ learnable matrices, and so adjusted during the training. In the case of vision transformers, the input image is split into patches. Each patch is encode into a vector through a learnable embedding and each vector is treated as an element from the sequence of embedded patches.

In the project each image of 120×120 RGB pixels is divided into 100 patches of size 12×12 and embedded into a 200 elements vector. A simple ViT with just one transformer block and just one head self-attention is used. The classification head used is the same as for the other three models described. The ViT used is the one proposed by Keras[4], which implements the model presented by Alexey Dosovitskiy et al. [6].

3 Compiling and fitting

3.1 Hyperparameters tuning

Learning rate and batch size for all the models are tuned with the KerasTuner API. Given the small set of values that the two parameters could assumed, the tuning for the VGG16 and ResNet50 networks is performed with Keras GridSearch Tuner[5]. Grid search is the simplest algorithm for hyperparameters tuning. The domain of the hyperparameters is divided into a discrete grid. Then the algorithm tries every combination of values of this grid, calculating some performance metrics. The metric chosen is the accuracy on the validation set. Grid search is an exhaustive algorithm that spans all the combinations, so it can actually find the best point in the domain. Since both learning rate and batch size can assume 3 different values each, the total number of trial to tune them is 9. In the case of the ViT, where Adam optimizer with weight decay is used, the hyperparameters to tune are: learning rate between 0.1, 0.01 and 0.001; batch size between 32, 64, 128 and 256 and weight decay between 0.1, 0.01 and 0.001. The tuning for the ViT took 36 trials. Each trial for both ResNet50, VGG16 and custom CNN took 5 epochs. As the ViT is not pretrained, and being generally a more difficult architecture to train, each trial took 10 epochs.

For both ResNet50 and VGG16 the best learning rate is 0.01, while the best batch size is respectively 32 and 48. Both the models have slightly lower validation accuracy with a batch size of 16. Generally the VGG16 has a considerably higher validation accuracy than the ResNet. For the ViT the best learning rate and weight decays values are both 0.001, with performances dropping considerably with higher values, especially 0.1; while the best value for the batch size is 32.

[4] `https://keras.io/examples/vision/image_classification_with_vision_transformer/`
[5] `https://keras.io/api/keras_tuner/tuners/grid/`

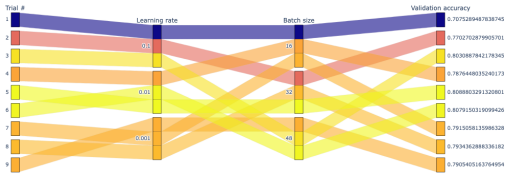


Figure 6: ResNet50 tested with various combination of values of the hyperparameters.

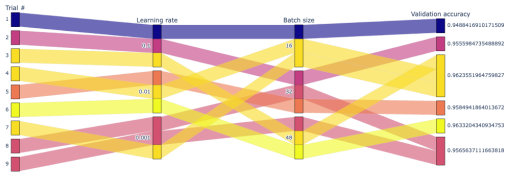


Figure 7: VGG16 tested with various combination of values of the hyperparameters.

The custom CNN requires a different hyperparameters tuning algorithm since the number of layers etc. need to be tuned too, and so the parameter space is too large. Keras RandomSearch Tuner is used. Random search is similar to grid search, but instead of using all the points in the grid, it tests only a randomly selected subset of these points. In the project, Keras RandomSearch is set to try 30 trials, therefore 30 combination of the hyperparameters values are explored and the one with the best validation accuracy is picked for the 5-Fold Cross Validation. The architecture of the custom CNN was explained in the previous chapter. Regarding the other hyperparameters: the best learning rate is 0.001 and the best batch size is 16. As designing CNNs it's a matter of both trial and error and the specific classification task they serve to, there are no notes to make about the number of layer and so on. Regarding the learning rate and batch size, higher values for both the hyperparameters has lead to worst performances.

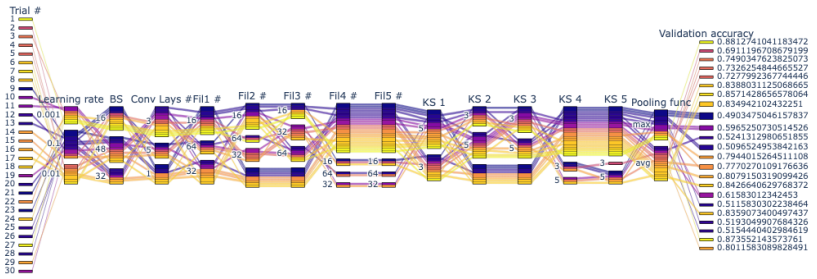


Figure 8: Custom CNN tested with various combination of values of the hyperparameters.

3.2 Compiling

Adaptive Moment Estimation (Adam) [7] is the optimizer used for ResNet50, VGG16 and the custom CNN. Adam uses the squared gradients to scale the learning rate and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum. To estimates the moments, Adam utilizes exponentially moving averages, computed on the gradient evaluated on a current mini-batch. The vectors of moving averages are initialized with zeros at the first iteration. The estimates are corrected through bias correction and finally are used to scale the learning rate individually for each parameter.

In the case of the ViT, after empirically determining that Adam was not effective, Adam with weight decay is used. While weight decay and standard regularization have the same effects when apply to the SGD optimizer, the two does not have to be confused when applied on Adaptive optimizers such as Adam, where standard regularization is not very effective [8].

3.3 Training

Models are evaluated through 5-Fold Cross Validation. Each model is trained on its respective training folds for 10 epochs. In particular: ResNet50 and VGG16's classification head are trained for 5 epochs while the rest of the network is freezed, then the top layers is unfreezed and fine tuned for 5 more epochs. Has mentioned before the ViT's hyperparameters are tuned for 10 epochs instead of 5, also the training took 20 epochs instead of 10. The training is performed using Keras callbacks for model checkpoints, that is saving the model each time it has a better training accuracy than in the previous epochs, and early stopping, that is stopping the training after a certain number of epochs in which the loss does not decrease. In particular early stopping is used also in the hyperparameters tuning stage. The number of epochs after early stopping happens, also called patience, is set to: 3 for both hyperparameters tuning and training for ResNet50 and VGG16; 3 for hyperparameters tuning and 5 for training for the custom CNN; 5 for hyperparameters tuning and 10 for training for the ViT.

The loss function to minimize during training is binary cross entropy. When evaluating the error on the validation set in the 5-Fold Cross Validation, the outputs of the network are round to either 0 or 1 and then zero-one loss is computed.

3.4 Notes

Few notes on how Keras implements parameters initialization and how Keras computes the binary cross entropy loss function. Despite having used a seed for each random operation applied in the project (such as shuffling the dataset and choosing the hyperparameters combination in the random search), there is still some slight variability in the results due to Keras weight initialization. Keras uses the Glorot, or Xavier, Uniform initialization. Given a layer indexed by i, its number of inputs (its number of neurons) is the fan in, n_i, while its number of outputs (the number of neurons of the subsequent layer it is connected to) is the fan out, n_{i+1}. Then, the weights of that layer are randomly initialized between $[-\frac{\sqrt{6}}{\sqrt{n_i+n_{i+1}}}, \frac{\sqrt{6}}{\sqrt{n_i+n_{i+1}}}]$.

Another thing to note when it comes to the precision of the computed results it's how Keras computes binary cross entropy. The function used during training, `binary_crossentropy`, comes with the argument `from_logits`, which by default is set to `False`. When `from_logits` is turned off, `binary_crossentropy` is supposed to get as input the probabilities obtained from a sigmoid; it turns the probabilities back to logits, clip them and then computing again a sigmoid and calculates the cross entropy with the function `sigmoid_cross_entropy_with_logits`. Despite not creating any serious issues for most of practical applications, it is still useful to report the

weight clipping applied by `binary_crossentropy` when concerns about the precision of generated probability values with sensitivity of less than 10^{-7} arise.

4 Results

4.1 Metrics

The metrics used to evaluate the performances of each model on its respective validation set during 5-Fold Cross Validation are:

- Zero-one los: the fraction of missclassified examples;
- Accuracy: the dual of zero-one loss, the fraction of correctly classified examples;
- Precision: in binary classification it's intuitively the ability of the classifier not to misclassify the examples of a class. It is the ratio `tp/(tp + fp)`. `tp` is the number of true positive examples (examples of the first class correctly classified) and `fp` is the number of false positive examples (examples of the second class missclassified for the first one);
- Recall: intuitively the ability of the classifier to correctly classify all the examples in a class. It is the ratio `tp/(tp + fn)`. `fn` is the number of false negative examples;
- F1-score: can be interpreted as the harmonic mean of precision and recall. F1 score reaches its best value at 1 and worst at 0.

In order to get more insight on the way the models worked, some saliency maps are presented. Saliency maps are from each one of the 5 models from the 5-Fold CV for all the four architecture, computed on the same 4 images randomly picked from the dataset. Using saliency maps is a way to query a CNN about the spatial support of a particular class in a given image. Saliency map can be computed just by calculating the derivative of the output class probability with respect to the the pixels in the input image, that is exactly computing backpropagation not just up to the first layer but up to the input image. Once the gradient vector is obtained, it is rearranged to get the saliency map [9].

4.2 Results

VGG16 is the architecture that obtained the best results. This may be due to several reasons: comparing the VGG16 with the custom CNN and ViT, it is a pretrained model, therefore it has already good tested weights. In comparison to the ResNet50, despite having more parameters, VGG16 has a lower number of layer, making it easier to manage when defining the layers to fine tune and then resulting in a better tuning. This last argument can be seen in Fig. 9. For VGG16, after an initial spike immediately after the start of fine tuning, the loss keep slightly decreasing. For ResNet50, after fine tuning the loss still remain higher than in the first 5 epochs.

Custom CNN was able to outperform both the ViT and ResNet50, this may be due to a matter of complexity, as the architecture of the custom CNN is considerably simpler than the ResNet and the ViT. The ViT is the architecture with the worst results recorded.

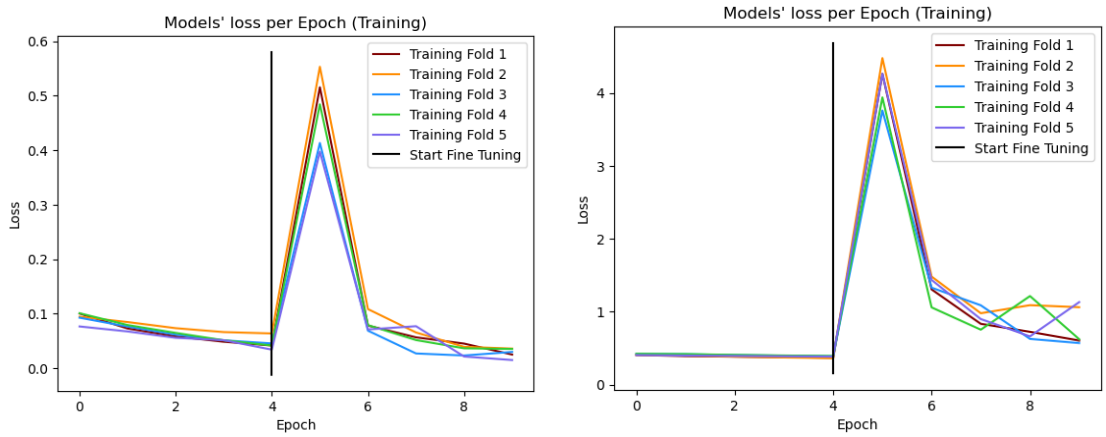
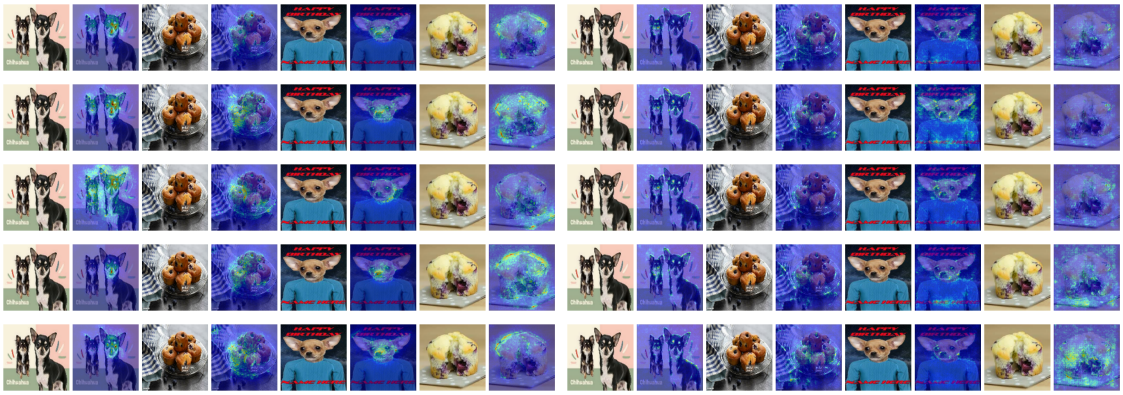


Figure 9: VGG16 (left) and ResNet50 (rigth) loss during training.

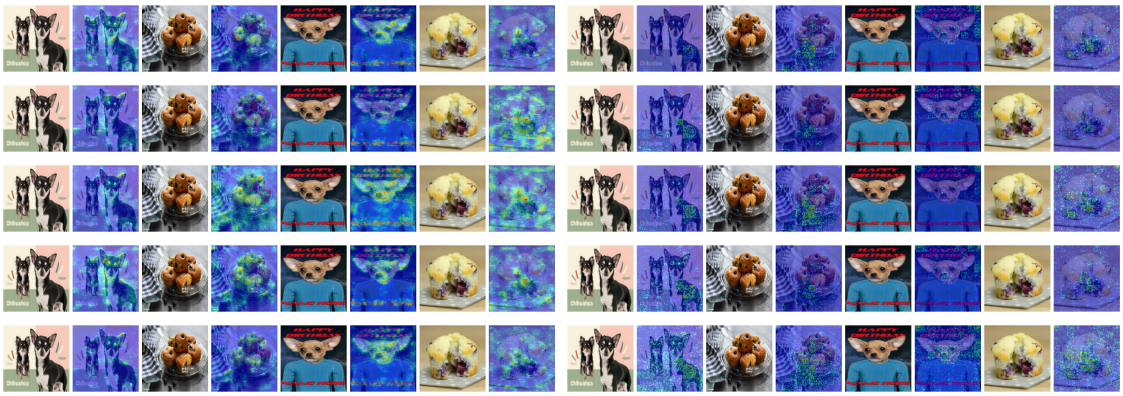
Reported results find a justification also in the saliency maps in Fig. 10. VGG16 is very precise in finding the most prominent features in the images, focusing on the face of the chihuahua, particularly the muzzle, and the contours of the muffins. The same goes for the custom CNN even though it can be noted that: in the first chihuahua image all the custom network tested are focused more on the eyebrows and in the second one are focusing even on the upper text in the picture, while saliency map for the muffins have a lot of noise. Also the saliency maps of ResNet50 and ViT are noisy and considerably sparse with respect to VGG16 and the custom CNN, still ResNet50 is able to catch features such as the muzzle of the chihuahuas and some contours of the muffins.

Model	Loss	Accuracy	Precision	Recall	F1 score
VGG16					
1	0.05	0.95	0.99	0.89	0.94
2	0.02	0.98	0.98	0.98	0.98
3	0.07	0.93	0.88	0.99	0.93
4	0.01	0.99	0.99	0.99	0.99
5	0.01	0.99	0.98	0.99	0.99
ResNet50					
1	0.21	0.79	0.95	0.60	0.74
2	0.15	0.85	0.89	0.81	0.85
3	0.36	0.64	0.99	0.29	0.45
4	0.32	0.68	0.99	0.38	0.55
5	0.19	0.81	0.97	0.74	0.84
Custom CNN					
1	0.19	0.81	0.95	0.65	0.77
2	0.23	0.77	0.79	0.74	0.77
3	0.18	0.82	0.95	0.67	0.78
4	0.16	0.84	0.95	0.72	0.82
5	0.11	0.89	0.89	0.90	0.89
ViT					
1	0.28	0.72	0.76	0.60	0.67
2	0.28	0.72	0.72	0.71	0.72
3	0.25	0.75	0.75	0.72	0.74
4	0.24	0.76	0.78	0.73	0.75
5	0.34	0.66	0.88	0.38	0.53

Table 1: Results of the models.



a) VGG16 b) ResNet50



c) Custom d) ViT

Figure 10: Saliency maps

References

[1] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

[2] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[8] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2017.

[9] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Workshop at International Conference on Learning Representations*, 2014.

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.